SBSeg16

XVI SIMPÓSIO BRASILEIRO
EM SEGURANÇA DA INFORMAÇÃO
E DE SISTEMAS COMPUTACIONAIS

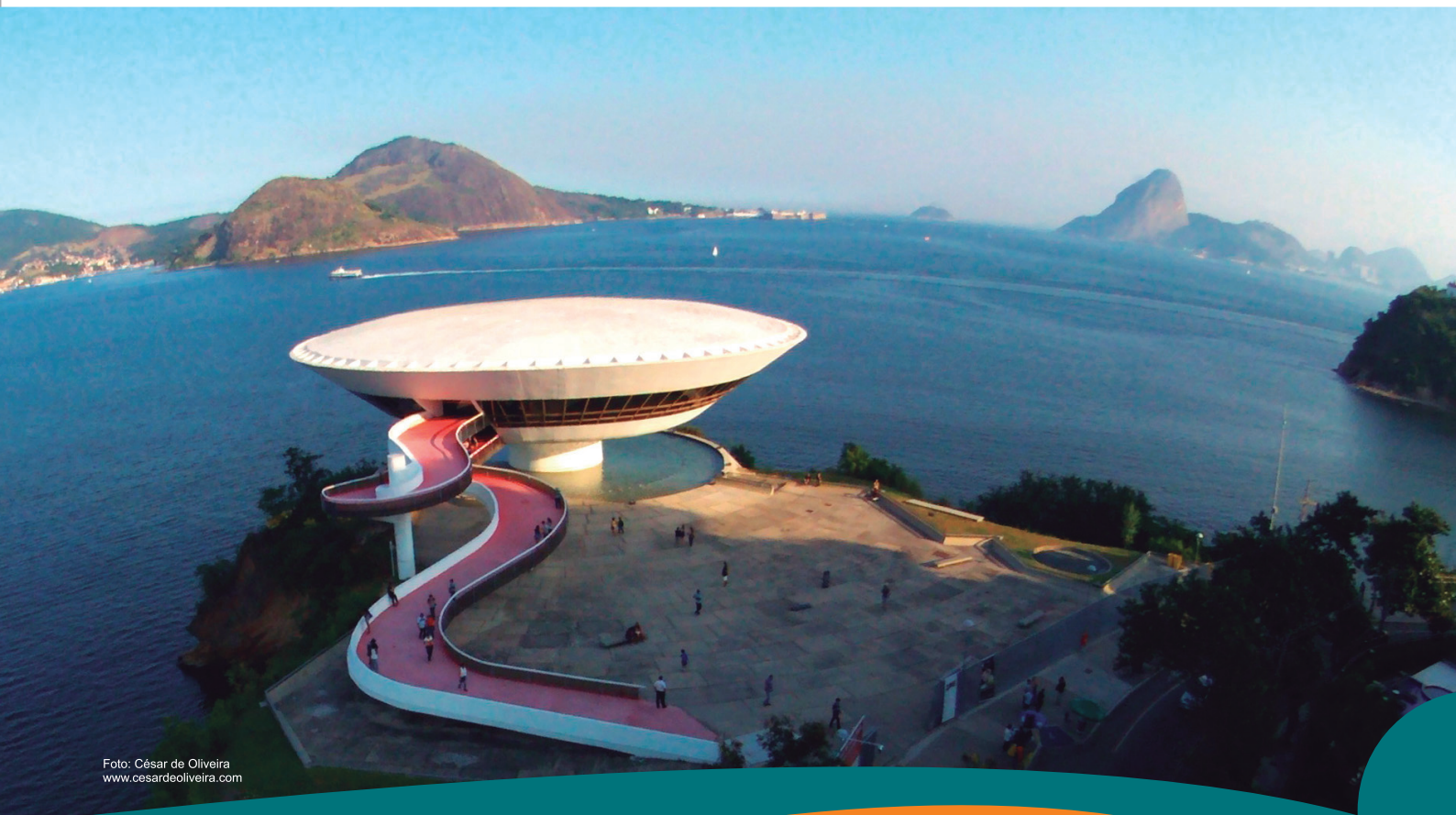


Foto: César de Oliveira
www.cesardeoliveira.com

ANAIS

SBSeg16

XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

Niterói, RJ, 7 a 10 de novembro de 2016

ANAIS

Sociedade Brasileira de Computação – SBC

Organizadores
Igor Monteiro Moraes, UFF
Antônio Augusto de Aragão Rocha, UFF

Realização
Universidade Federal Fluminense – UFF
Sociedade Brasileira de Computação – SBC

Capa: Fatima Jane Ribeiro
Editoração: Ian Vilar Bastos, UFF

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da Universidade Federal Fluminense

S612 Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (16. : 2016 : Niterói, RJ)
Anais [do] XVI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais, 07 a 10 de novembro de 2016 / Sociedade Brasileira de Computação ; Organizadores, Igor Monteiro Moraes, Antônio Augusto de Aragão Rocha — Niterói, RJ: Sociedade Brasileira de Computação, 2016.
789 p.
CD-ROM : il. ; 4¾ pol..

ISSN: 2176-0063

1. Ciência da computação. 2. Informática. 3. Segurança da informação. 4. Segurança de sistemas. I. Moraes, Igor Monteiro (org.) II. Rocha, Antônio Augusto de Aragão (org.) III. Título.

CDD 005.8 (21. ed)

Sociedade Brasileira de Computação – SBC

Contato
Av. Bento Gonçalves, 9500
Setor 4 - Prédio 43.412 - Sala 219
Bairro Agronomia
91.509-900 – Porto Alegre RS

CNPJ: 29.532.264/0001-78
http://www.sbrc.org.br

Mensagem da Coordenação Geral

É com satisfação que damos as boas-vindas ao XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2016), pela primeira vez realizado na cidade de Niterói. Apesar de desafiador, sempre entendemos que trazer um evento desse porte e de tal importância para a comunidade científica seria, sem dúvida, uma grande oportunidade para a nossa instituição, em especial para os alunos da Universidade Federal Fluminense (UFF). Certamente, por compartilharem desse entendimento, toda a comunidade do Departamento de Ciência da Computação (DCC) do Instituto de Computação (IC) da UFF apoiou desde o início a realização do evento em nossas dependências. É, portanto, a esta comunidade que agradecemos primeiramente o apoio à realização do evento, e fazemos isso através da direção do Instituto, Prof. José Henrique Carneiro de Araújo, e da chefia do departamento, Prof. José Raphael Bokehi. Não podemos deixar de ressaltar ainda o apoio institucional da reitoria, na pessoa do Magnífico Reitor Prof. Sidney Luiz de Matos Mello, que não mediu esforços em ajudar a organização do evento, mesmo diante do momento delicado vivido pelo país e das instituições federais de ensino.

A programação técnica do SBSeg 2016 está abrangente e sem dúvida de grande qualidade, formada por 16 sessões técnicas (que contará com a apresentação de 31 artigos completos), 5 palestras internacionais e 1 nacional, todas de pesquisadores renomados internacionalmente na área de Segurança, 4 minicursos que tratam de temas atuais e relevantes para a comunidade, 3 workshops tecnológicos, além do concurso de teses de dissertações e o workshop de trabalhos de iniciação científica e graduação. Enfim, serão quatro dias de evento e uma grade repleta de atrações.

A qualidade desta grade de programação é resultado do empenho de diversas pessoas. Portanto, um agradecimento especial aos Coordenadores do comitê de programa (Leonardo Barbosa e Oliveira, UFMG, e Pedro Braconnot Velloso, UFRJ), ao Coordenador de palestras e tutoriais (Luís Henrique Maciel Kosmalski Costa, UFRJ), ao Coordenador de minicursos (Michel Abdalla, ENS & CNRS), aos Coordenadores do concurso de teses e dissertações (Diego de Freitas Aranha, UNICAMP, e Luciano Paschoal Gaspary, UFRGS), e ao Coordenador do workshop de trabalhos de iniciação científica e de graduação (Daniel Macêdo Batista, USP). Além desses, sem dúvida o evento não poderia ter alcançado o nível atual de organização sem a ajuda dos Coordenadores locais, e portanto, um agradecimento especial aos professores Célio Vinicius Neves de Albuquerque, Karina Mochetti, Luis Antonio Brasil Kowada e Natalia Castro Fernandes, da UFF, e Miguel Elias Mitre Campista, da UFRJ. Agradecemos também aos coordenadores do workshop de Gestão de Identidades Digitais (Michelle Wangham, UNIVAL), do workshop de Forense Computacional (Rafael Obelheiro, UDESC, e Sandro Süffert, Apura) e do workshop de Regulação, Avaliação da Conformidade e Certificação de Segurança (Raphael Machado, Inmetro).

Por fim, somos muito gratos ainda à instituição parceira, Universidade Federal do Rio de Janeiro (UFRJ), à diretoria da Sociedade Brasileira de Computação e à Coordenação e Comitê técnico de apoio ao evento da Comissão Especial em Segurança da Informação e de Sistemas Computacionais da SBC (CE-Seg). Agradecemos ainda aos patrocinadores do simpósio: Comitê Gestor da Internet no Brasil (CGI.br); órgãos de fomento CNPq, CAPES, FAPERJ, e Proppi/UFF; Intel; Clavis; e, Google. Sem esse apoio, a viabilização financeira do evento seria extremamente dificultada.

Em nome da comissão organizadora do SBSeg 2016, desejamos a todos uma semana enriquecedora em conhecimento e de agradável convívio em Niterói.

Igor Monteiro Moraes, UFF
Antônio Augusto de Aragão Rocha, UFF
Coordenadores Gerais do SBSeg 2016

Mensagem da Coordenação do Comitê de Programa

O Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg) é o maior evento acadêmico na área de Segurança Digital. Ele é anualmente realizado pela Sociedade Brasileira de Computação (SBC). Este ano o evento ocorre pela primeira vez na cidade de Niterói – RJ e é realizado pelo Instituto de Computação da Universidade Federal Fluminense (UFF).

A XVI edição do SBSeg desempenhará o papel de principal fórum para o debate de questões relativas à pesquisa, desenvolvimento e inovação nesta área. Será uma reunião de centenas de participantes, dentre eles, discentes, pesquisadores, docentes e profissionais do Brasil e exterior, bem como representantes da indústria e da segurança pública.

Este ano, o SBSeg traz consigo números bastante significativos. Isso porque o número de trabalhos submetidos e aceitos foram, respectivamente, 87 e 32. Assim sendo, a taxa de aceite se manteve próxima aos tradicionais 35%; mas o número de artigos aceitos foi ampliado em termos absolutos. Isso, por sua vez, permitiu que todo o período disponível para a apresentação dos trabalhos fosse preenchido com trabalhos completos.

É importante lembrar que o êxito obtido na seleção de trabalhos com alta qualidade técnica e científica para o SBSeg 2016 foi graças ao empenho e competência de inúmeros atores. Assim, o nosso muito obrigado aos: autores, membros do comitê de programa, organizadores, revisores, patrocinadores, promotores e voluntários. Vale frisar nossos agradecimentos à Coordenação Geral do SBSeg pela confiança em nós depositada.

Por fim, desejamos a todos os participantes do SBSeg 2016 uma semana instigante, profícua, e que as apresentações dos trabalhos selecionados possam contribuir para discussões e debates para o avanço das pesquisa na área de segurança da informação e de sistemas computacionais.

Leonardo B. Oliveira, UFMG
Pedro B. Velloso, UFRJ
Coordenadores do Comitê de Programa do SBSeg 2016

Mensagem da Coordenação do CTDSeg 2016

O Concurso de Teses e Dissertações (CTD) é um evento bienal da Comissão Especial em Segurança da Informação e de Sistemas Computacionais (CESeg) da Sociedade Brasileira de Computação (SBC). Seu objetivo é premiar as melhores teses de doutorado e dissertações de mestrado da área de Segurança da Informação e de Sistemas Computacionais do país. Na sua quarta edição foram submetidos 19 trabalhos, 4 de doutorado e 15 de mestrado, o que denota o prestígio do concurso junto à comunidade científica brasileira.

Para a avaliação dos trabalhos submetidos foi montado um comitê de 20 membros. Os artigos e as teses/dissertações submetidos passaram por um processo inicial de seleção visando escolher as melhores teses de doutorado e dissertações de mestrado. Cada artigo foi revisado por dois especialistas. Esses trabalhos compõem o "programa técnico"do CTD e encontram-se publicados nos Anais. A distinção das teses e dissertações melhores colocadas será conhecida ao final do Congresso após análise detalhada, por parte de um Comitê de Avaliação local, do conteúdo completo das 4 teses e 6 dissertações selecionadas na já referida etapa inicial e das respectivas apresentações.

Desejamos manifestar nossos sinceros agradecimentos a todos os membros do Comitê de Avaliação pelo árduo trabalho realizado em um curto espaço de tempo. Finalmente, agradecemos aos autores, que prestigiaram este evento submetendo seus trabalhos para apreciação.

Diego Aranha, UNICAMP
Luciano Paschoal Gaspary, UFRGS
Coordenadores do CTDSeg 2016

Mensagem da Coordenação do WTICG

O Workshop de Trabalhos de Iniciação Científica e de Graduação (WTICG) é um evento itinerante, organizado em conjunto com o Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg). Neste ano de 2016, em sua 10ª edição, o evento será realizado na cidade de Niterói. O objetivo do workshop é promover a divulgação de trabalhos científicos nas áreas de segurança da informação e de sistemas computacionais. O WTICG é destinado aos potencias futuros pesquisadores das áreas citadas anteriormente, que estejam na condição de estudante de graduação ou recém-graduados no período da submissão.

Nesta edição o WTICG contou com a valiosa colaboração de 42 revisores de alta competência técnica, que após o processo de avaliação de 23 trabalhos submetidos na categoria de trabalho completo e 5 trabalhos submetidos na categoria pôster, selecionaram 11 trabalhos completos e 3 pôsteres.

Neste ano, pela primeira vez, foi instituído o CESeg *Student Grants*, um programa da CESeg (Comissão Especial em Segurança da Informação e de Sistemas Computacionais da Sociedade Brasileira de Computação) que tem por objetivo estimular a participação de estudantes no WTICG. O estímulo foi realizado por meio do subsídio da inscrição dos estudantes de graduação ou recém-graduados que tiveram trabalhos aceitos no WTICG. Os autores de todos os 14 trabalhos nesta condição tiveram direito ao subsídio.

A programação do WTICG terá quatro seções técnicas (Segurança em redes, Segurança em Internet das Coisas, Forense computacional & vazamento de informação e Criptografia & criptoanálise: algoritmos, protocolos, técnicas e aplicações) e uma sessão de pôster.

Após as apresentações orais do WTICG serão anunciados os três melhores trabalhos completos, sendo que 1 autor do trabalho que ficar em primeiro lugar ganhará da Clavis, patrocinadora prata do SBSeg16, 1 inscrição gratuita no curso de formação “Perito em Análise Forense Computacional", com carga horária de 80 horas. Além disto também será anunciado o melhor pôster.

Eu gostaria de agradecer aos autores pela escolha do WTICG para submeter seus trabalhos, aos revisores pelo trabalho incansável de revisão, aos membros do comitê de programa pela ajuda nas várias decisões que tive que tomar na organização do workshop, e aos coordenadores gerais, Prof. Antônio Augusto de Aragão Rocha e Prof. Igor Monteiro Moraes, pela confiança depositada em mim para coordenar este workshop.

Gostaria de dizer também que foi muito prazeroso para mim conduzir os trabalhos do WTICG e espero que todos aproveitem o evento ao máximo, em Niterói.

Daniel Macêdo Batista, IME-USP
Coordenador do WTICG 2016

Mensagem da Coordenação do WGID

O Workshop de Gestão de Identidades (WGID), evento integrante do SBSeg, visa ser um fórum para discussões sobre o estado da arte de tecnologias relacionadas à gestão de identidades digitais, identificar os principais desafios de pesquisa e aproximar os grupos de pesquisadores atuantes na área.

Em sua sexta edição, o programa do Workshop contemplará um painel de discussão sobre Gestão de Identidades na Internet das Coisas com os professores especialistas nestas áreas, Fábio Borges de Oliveira do LNCC, Leonardo B. Oliveira da UFMG, Raphael Machado do INMETRO, e que terá como moderadora a professora Noemi Rodriguez da PUC-Rio.

Um ponto de destaque do workshop é a apresentação de artigos científicos e resumos estendidos nas seções técnicas. Pesquisadores foram convidados a submeter trabalhos originais relacionados à Gestão de Identidades, sendo que três (3) artigos completos e três (3) resumos estendidos foram aceitos. Gostaria de agradecer aos membros do comitê de programa pela qualidade do trabalho realizado nas revisões. Registro um agradecimento especial a todos os autores que prestigiaram o WGID ao submeterem trabalhos relatando suas pesquisas.

Encerrando o programa de 2016, teremos três palestras técnicas dos analistas da RNP (DAGSer) sobre os serviços ICPEdu, Eduroam e CAFe, além de uma breve apresentação do ambiente de experimentação em gestão de identidade (GIdLab), mantido pela RNP.

Gostaria também de agradecer a todos que colaboraram na organização do WGID, especialmente, os membros do Comitê Técnico de Gestão de Identidades da RNP e os coordenadores do SBSeg 2016, Antônio Augusto Rocha e Igor Moraes (UFF).

Em nome do Comitê de Programa do WGID, saudamos a todos os participantes desta edição, com votos de um evento bastante profícuo.

Profa. Michelle S. Wangham, UNIVALI
Coordenadora do WGID 2016

Mensagem da Coordenação do WFC

O Workshop de Forense Computacional (WFC) visa ser um fórum para discussões e apresentações técnicas de trabalhos em torno do estado da arte de tecnologias relacionadas com forense computacional. Além disso, busca-se também identificar os desafios de pesquisa e possibilitar a integração de pesquisadores e profissionais da área.

Em 2016 o WFC chega à sua quinta edição, sendo novamente realizado em conjunto com o SBSeg, dando continuidade à parceria de sucesso iniciada no SBSeg 2012, em Curitiba. Nesta edição, o evento terá duas sessões técnicas de artigos científicos. Este ano tivemos 17 submissões, mais que o dobro da edição anterior, e o processo de seleção foi bastante competitivo, resultando na aceitação de sete artigos de alta qualidade.

Além das sessões de artigos, teremos também uma oficina e um desafio forense, que propiciarão uma introdução *hands-on* à análise forense, buscando assim fomentar o interesse pela área. O desafio foi introduzido no WFC em 2015, tendo sido muito bem recebido pela comunidade, e esperamos reeditar o grande sucesso da edição anterior.

Gostaríamos de agradecer, em primeiro lugar, aos pesquisadores que prestigiaram o WFC submetendo artigos para o evento. Muito obrigado aos integrantes do comitê de programa e revisores adicionais, que contribuíram com sua *expertise* para garantir a qualidade do programa do workshop. Agradecemos também à comissão organizadora do SBSeg, especialmente aos coordenadores gerais do simpósio, Profs. Antônio Augusto de Aragão Rocha e Igor Monteiro Moraes, da Universidade Federal Fluminense (UFF), que não pouparam esforços para fazer um ótimo evento.

Em nome do WFC, cumprimentamos a todos os participantes, desejando uma excelente estada em Niterói, e que possam desfrutar dos atrativos da Cidade Sorriso e aproveitar todas as atividades do SBSeg.

Rafael R. Obelheiro (UDESC)
Sandro Süffert (Apura)
Coordenadores do WFC

Mensagem da Coordenação do WRAC+

O Workshop de Regulação, Avaliação da Conformidade e Certificação de Segurança (WRAC+) tem por objetivo proporcionar um fórum onde Governo, Laboratórios, Academia, Indústria e Sociedade possam discutir os potenciais e os limites das ferramentas da regulação e da avaliação da conformidade no campo da Segurança da Informação.

Este segundo WRAC+ dá prosseguimento à missão iniciada em 2015, de trazer para a comunidade científica os desafios de quem precisa especificar requisitos de segurança da informação e avaliar a conformidade a tais requisitos. Neste segundo WRAC+, temos o privilégio de contar com a presença dos diversos segmentos da sociedade, representados através de palestras que trazem as visões do Governo, dos Laboratórios, das Universidades e da Indústria a respeito da Segurança da Informação e das formas através das quais as ferramentas da Regulação e da Avaliação da Conformidade podem auxiliar a Sociedade a alcançar um adequado status de Segurança. Contaremos com a presença e a participação de palestras de Ruy Ramos, do ITI, abordando os dispositivos da ICP-Brasil, de Sanderson Barbalho, da UnB, que discutirá o Sistema de Homologação e Certificação de Produtos e Serviços de Defesa Cibernética, com destaque para o caso dos equipamentos de videoconferência, além de painéis que contarão com a presença dos principais laboratórios envolvidos com a atividade de avaliar a segurança de dispositivos inteligentes.

Esperamos que o WRAC+ contribua para uma discussão cada vez mais esclarecida a respeito da regulação e da avaliação da conformidade no campo da Segurança da Informação.

Raphael Machado, Inmetro
Coordenador do II WRAC+

Comitê de Programa do SBSeg 2016

Alberto Schaeffer Filho, UFRGS
Aldri dos Santos, UFPR
Altair Olivo Santin, PUCPR
Anderson Nascimento, UnB
André Grégio, UFPR
Carla Merkle Westphall, UFSC
Carlos da Silva, UFRN
Carlos Maziero, UTFPR/UFPR
Carlos Westphall, UFSC
Célio Vinicius Neves de Albuquerque, UFF
Davidson Boccardo, INMETRO
Denise Goya, UFABC
Diego Aranha, Unicamp
Dorgival Guedes, UFMG
Edna Canedo, UnB
Eduardo Alchieri, UnB
Eduardo Feitosa, UFAM
Eduardo Góes, DPF
Eduardo Souto, UFAM
Emerson Ribeiro de Mello, IFSC
Hao Chi Wong, Intel
Jean Martina, UFSC
Jeroen van de Graaf, UFMG
Joaquim Celestino Júnior, UECE
Karina Mochetti, UFF
Lau Cheuk Lung, UFSC
Leonardo Fagundes, UNISINOS
Leonardo Oliveira, UFMG
Lisandro Zambenedetti Granville, UFRGS
Luciano Paschoal Gaspary, UFRGS
Luiz Carlos Albini, UFPR
Luiz Fernando Rust da Costa Carmo, INMETRO
Marco Aurelio Amaral Henriques, Unicamp
Marcos Simplicio Jr, Poli-USP
Marinho Barcellos, UFRGS
Marjory da Costa Abreu, UFRN
Mário Sérgio Alvim, UFMG
Michele Nogueira, UFPR
Michelle Wangham, Univali
Miguel Elias Mitre Campista, UFRJ
Otto Carlos Muniz Bandeira Duarte, UFRJ
Paulo André da Silva Gonçalves, UFPE
Paulo Lício de Geus, Unicamp
Pedro Braconnot Velloso, UFRJ
Priscila Solis, UnB
Rafael de Sousa Junior, UnB
Rafael Misoczki, Intel Labs - USA
Raphael Machado, INMETRO
Raul Ceretta Nunes, UFSM
Raul Weber, UFRGS
Ricardo Custódio, UFSC
Ricardo Dahab, Unicamp
Roberto Gallo, KRYPTUS Seg. da Informação
Rossana Andrade, UFC
Routo Terada, IME-USP
Sergio de Oliveira, UFSJ

Avaliadores do SBSeg 2016

Alan de Sá, Marinha do Brasil / UFRJ
Aldelir Fernando Luiz, IFC
Alvaro Robles Rincon, UFRJ
Anderson D'Aquino, UFRJ
Antônio Lobato, UFRJ
Antônio Rodrigo Delepiane de Vit, UFSM
Carlos da Silva, Unicamp
Carlos André Batista de Carvalho, UFPI
Charles Prado, Inmetro
Cleber Souza, Unicamp
Cleverton Vicentini, PUC-PR
Daniel Presser, UFSC
Dario Fernandes, Unicamp
Diego Saraiva, IFRN
Diogo Mattos, UFRJ
Douglas Silva, UFSC
Eder John Scheid, UFRGS
Eduardo Viegas, PUC-PR
Elisa Mannes, UFPR
Ewerton Andrade, USP
Fernando Ortiz, UFRJ
Fernando Augusto Teixeira, UFSJ
Flávio Santos, Chaordic Systems
Geovandro Pereira, USP
Glederson Santos, IFSul
Hylson Netto, UFSC
Igor Jochem Sanz, UFRJ
Jorge Werner, UFSC
Juliano Wickboldt, UFRGS
Leonardo Rocha, UECE
Lucas Bondan, UFRGS
Lucas Boppre, UFSC
Lucas Freire, UFRGS
Lucas Perin, UFSC
Luciana Moreira Sá de Souza, UFSC
Lucila Bento, UFRJ
Maicon Stihler, PUC-PR
Marcelo Colomé, UFSM
Marcelo Palma Salas, Unicamp
Marcial Fernandez, UECE
Marco Alves, UFPR
Marcus Botacin, Unicamp
Marlon Domenech, Univali
Martín Vigil, UFSC

Martin Andreoni Lopez, UFRJ
Maurício Oliveira, UFSC
Márcio Carvalho, UFRGS
Mário Alvim, UFMG
Pedro Alves, Unicamp
Pedro Henrique Cruz Caminha, UFRJ
Pedro Maat Massolino, Radboud Univ.
Renê Oliveira, UFSC
Ricardo Pfitscher, UFRGS
Rick Lopes de Souza, UFSC
Robson Albuquerque, UnB
Rone Silva, UFSJ
Rosembergue Souza, UFRJ
Sérgio de Medeiros Câmara, UFRJ
Sergio Gutierrez, Univ. Nacional de Colombia
Taciane Martimiano, UFSC
Thomas Filipe Diniz, Anolis Tec. da Informação
Victor Furuse Martins, Unicamp
Victor Martins, Unicamp
Vilmar Abreu, PUC-PR
Vinícius Guimarães, IFSul
Vitor Afonso, Unicamp
Weverton Cordeiro, UFRGS
Wilson Melo Jr, Inmetro

Comitê de Programa do CTDSeg 2016

Aldri dos Santos, UFPR
Altair Santin, PUCPR
Anderson Nascimento, UnB
André Grégio, UFPR
Carlos Maziero, UFPR
Eduardo Alchieri, UnB
Fabio Piva, Unicamp
Fernando Quintao Pereira, UFMG
Jean Martina, UFSC
Jeroen van de Graaf, UFMG
Luciano Paschoal Gaspary, UFRGS
LuisMenaschéSchechter, UFRJ
Luiz Fernando Rust da Costa Carmo, Inmetro
Luiz Carlos Albini, UFPR
Marcos Simplicio Jr, USP
Mário Alvim, UFMG
Michele Nogueira, UFPR
Paulo de Geus, Unicamp
Ricardo Dahab, Unicamp
Sandro Rigo, Unicamp

Comitê de Programa do WTICG 2016

Altair Santin, PUCPR
Anderson Nascimento, UnB
Andre Gustavo Degraf Uchoa, The University of York
Arlindo Marcon Jr., IFPR
Carla Merkle Westphall, UFSC
Carlos Maziero, UFPR
Diego Aranha, Unicamp
Edna Dias Canedo, UnB
Eduardo Feitosa, UFAM
Eduardo Souto, UFAM
Emerson Ribeiro de Mello, IFSC
Heverson B. Ribeiro, Université de Neuchâtel
Iguatemi E. Fonseca, UFPB
João Gondim, UnB
Jose Antonio Xexeo, IME
Lau Cheuk Lung, UFSC
Leonardo B. Oliveira, UFMG
Lisandro Zambenedetti Granville, UFRGS
Luiz Fernando Rust da Costa Carmo, Inmetro
Marcos Simplicio Jr, USP
Michelle Wangham , UNIVALI
Paulo André da Silva Gonçalves, UFPE
Priscila Solis, UnB
Rafael Obelheiro, UDESC
Raphael Machado, Inmetro
Raul Ceretta Nunes, UFSM
Ricardo Custódio, UFSC
Rossana Andrade, UFC
Vivek Nigam, UFPB
William de Souza, University of London

Avaliadores do WTICG 2016

Antônio Lacerda Jr., Inmetro
Daniel Presser, UFSC
Eduardo Cominetti, USP
Elisa Mannes, UFPR
Hylson Netto, UFSC
Jessica Carneiro, UFMG
Luiz Arthur, UTFPR
Maicon Stihler, PUC-PR
Maykon de Souza, Univali
Pedro Pinto, UFMG
Rodolfo Souza, Inmetro
Rodrigo Campiolo, UTFPR
Thiago Almeida, USP

Comitê de Programa do WRAC+ 2016

Carlos D'Avila, UFRJ
Danielle Vieira, Inmetro
Davidson Boccardo, Inmetro
Jayme Szwarcfiter, UFRJ
Jonny Doin, Grid Vortex
Luiz Fernando Rust da Costa Carmo, Inmetro
Ruy Ramos, ITI

Comitê de Programa do WFC 2016

Alceu Britto Jr., PUC-PR
Altair Santin, PUC-PR
André Grégio, UFPR
Charles Christian Miers, UDESC
Cristine Hoepers, CERT.br
Emerson Ribeiro de Mello, IFSC
Flavio Elias de Deus, UnB
João Gondim, UnB
Klaus Steding-Jessen, CERT.br
Marcelo Caiado, PGR/MPF
Paulo Lício de Geus, Unicamp
Paulo Mafra, SENAI-SC
Rafael Timóteo Sousa Jr., UnB
Ricardo Dahab, Unicamp
Ricardo Kléber Galvão, IFRN

Avaliadores do WFC 2016

Christian Bachmann, PUC-PR
Gabriel Cavalcante, Unicamp
Maicon Stihler, PUC-PR
Marcus Felipe Botacin, Unicamp
Victor Furuse Martins, Unicamp
Vitor Afonso, Unicamp

Comitê de Programa do WGID 2016

Altair Santin, PUC-PR
Antônio Tadeu Gomes, LNCC
Carlos da Silva, UFRN
Carlos Ferraz, UFPE
Debora Muchaluat-Saade, UFF
Emerson Ribeiro de Mello, IFSC
Fábio Borges, LNCC
Flavio Deus, UnB
Gustavo Motta, UFPB
Jean Martina UFSC
Leonardo Oliveira, UFMG
Marco Aurélio Amaral Henriques, Unicamp
Michelle Wangham, Univali
Natália Fernandes, UFF
Noemi Rodriguez, PUC-Rio
Rafael de Sousa Junior, UnB
Ricardo Custódio, UFSC
Ricardo Dahab, Unicamp
Roberto Samarone Araujo, UFPA
Vinod Rebello, UFF

Sumário

SBSeg16

XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

Niterói, RJ, novembro de 2016

Artigos Completos

Autenticação de Sensores usando Eventos Físicos

Wilson S. Melo Jr.[1], **Charles B. Prado**[1], **Luiz F. R. C. Carmo**[2]

[1]Instituto Nacional de Metrologia, Qualidade e Tecnologia (Inmetro)
Av. Nossa Senhora das Graças, 50 – Duque de Caxias – RJ

[2]Universidade Federal do Rio de Janeiro
Rio de Janeiro – RJ.

{wsjunior,cbprado}@inmetro.gov.br, lfrust@nce.ufrj.br

Abstract. *Sensors authentication is a challenge that need to be properly addressed. In this paper we propose the use of physical events for sensors authentication. Unpredictability and uniqueness physical systems properties are explored to generate event identifiers from sensing information. Being unique and hard to guess, event identifiers can be used for strengthening authentication. Moreover, event identifiers provide evidences of authentication co-location and simultaneity, preventing relay and replay attacks. The authentication feasibility is demonstrated using sensing systems real cases related to vehicle crash test instrumentation, with promising results.*

Resumo. *A autenticação de sensores é um desafio que necessita ser tratado de forma apropriada. Neste artigo é proposto o uso de eventos físicos para autenticação de sensores. A imprevisibilidade e unicidade dos sistemas físicos são exploradas para gerar identificadores de evento a partir de dados de sensoriamento. Sendo únicos e difíceis de se deduzir, identificadores de evento podem ser usados para reforçar a autenticação. Além disso, eles proveem evidências de co-alocação e simultaneidade da autenticação, evitando ataques do tipo relay e replay. A viabilidade dessa proposta é demonstrada em sistemas de instrumentação de testes de impacto veiculares, com resultados promissores.*

1. Introdução

O mundo testemunhou na última década um crescimento significativo no uso de tecnologias pervasivas. Os computadores estão hoje presentes nas mais diferentes áreas de aplicação. Novas tendências, como Internet das Coisas (IoT) [Al-Fuqaha et al. 2015] e Sistemas Físicos-Cibernéticos (CPS) [Mitra et al. 2013], surgem como campos promissores de pesquisa e desenvolvimento. Atividades de trabalho, negócios, ensino, saúde, mobilidade e lazer podem ser monitoradas, medidas e controladas por aplicações complexas classificadas como "inteligentes" (*smart*).

Neste complexo conjunto de conceitos e tecnologias, o sensoriamento é parte fundamental. Sensores podem ser descritos como blocos elementares na criação das funcionalidades esperadas de uma aplicação "inteligente". Complexos industriais, edificações, veículos, eletro utilitários e mesmo dispositivos biomédicos empregam atualmente uma diversidade de sensores capazes de medir praticamente qualquer tipo de evento físico. Todavia, a segurança da informação continua a ser um preocupação quando tais sistemas complexos são analisados [Al-Fuqaha et al. 2015]. Algumas questões resultam do fato

de que estes dispositivos interagem diretamente com o mundo físico, um desafio relativamente novo cujas consequências não são ainda totalmente conhecidas [Mitra et al. 2013]. Neste contexto, processos como autenticação e autorização são especialmente críticos para se garantir propriedades essenciais de segurança da informação.

Uma vez que sensores se conectam a outros sensores e também a componentes do sistema em níveis mais altos, eles necessitam prover mecanismos de autenticação para garantir acesso mútuo e proteção de informações. Diversas abordagens propondo autenticação de sensores são encontradas na literatura [Priya and Patil 2014]. Pode-se citar as implementações de ICP (Infraestrutura de Chave Pública), assinaturas HMAC que são comuns em sensores inteligentes, soluções baseadas em PUF (*Physical Unclonable Function*) [Suh and Devadas 2007] e funções *one-time* [Hsieh and Leu 2011]. Entretanto, estas soluções podem ser custosas quando dependem de infraestrutura específica. Em outros casos, restrições inerentes à construção dos sensores (bateria, memória e poder computacional por exemplo) podem restringir o uso de soluções de autenticação mais sofisticadas. Assim, mecanismos de autenticação que possam ser implementados sem a necessidade de qualquer recurso adicional são muito desejáveis.

Este trabalho se origina da seguinte questão. Considerando-se a existência de um grupo de sensores que necessitam ser autenticados e que estes sensores monitoram um mesmo evento físico, que possui propriedades físicas intrínsecas, seria possível propor um mecanismo de autenticação baseado em um identificador deste evento físico? Considerando esta questão, este trabalho propõe condições nas quais sistemas físicos são usados como geradores de eventos não determinísticos. Ao mesmo tempo, esses sistemas podem ser vistos como um canal exclusivo para distribuição de identificadores em condições onde um atacante não possua acesso ao ambiente físico, assim evitando ataques *eavesdrop*. Além disso, eventos físicos servem como evidência de simultaneidade e co-alocação, dois conceitos que reforçam a autenticação em situações relacionadas a ataques do tipo *relay* e *replay*. Esta estratégia é denominada *Autenticação baseada em Eventos Físicos* e pode ser implementada em sistemas que utilizam sensoriamento sem a necessidade de qualquer infraestrutura adicional. A proposta é validada usando casos reais de sensoriamento associados com a instrumentação de testes de impacto veiculares.

2. Trabalhos Relacionados

Ideias relacionadas à presente proposta são encontradas na biometria comportamental [Yampolskiy and Govindaraju 2008], que se baseia na análise das ações físicas de um usuário tais como piscar de olhos, técnica de digitação, modo de caminhar, ou qualquer ação que possa ser medida e classificada dentro de um padrão específico. Estabelecido o padrão, este serve para identificar o usuário. Capturar um padrão comportamental envolve o uso de diferentes sensores que descrevem, essencialmente, um evento físico.

A associação entre biometria comportamental e eventos físicos torna-se mais evidente em trabalhos que usam acelerômetros para compor padrões. Em [Gafurov et al. 2007] é descrito um método que usa acelerômetros para reconhecer o jeito de andar de uma pessoa. Aspectos como o posicionamento do sensor, a carga carregada pela pessoa e a taxa de amostragem do sinal afetam o processo de identificação. A mesma ideia é explorada em [Mayrhofer and Gellersen 2007] ao propor um método intuitivo de autenticação mútua de dois dispositivos inteligentes usando acelerômetros. Ambos os

dispositivos devem ser chacoalhados juntos de modo que seus acelerômetros exibirão um alto grau de correlação. O mesmo princípio é explorado em [Wu et al. 2011] que introduz o termo "aperto de mão físico-cibernético". Dois usuários usam dispositivos inteligentes similares a relógios e equipados com acelerômetros. Ao apertarem de mãos, o que constitui um evento físico, os usuários criam um padrão para autenticação mútua.

Também é possível encontrar ideias relacionadas em trabalhos que propõem autenticação a partir de eventos da camada física de redes de comunicação. O uso de *traces* de pacotes de rede para gerar identificadores que atestam que duas entidades estão fisicamente próximas uma da outra, conceito denominado pelos autores como *co-alocação*, é descrito em [Scannell et al. 2009]. Já o conceito de autenticação baseada em canal, com o argumento de que o canal de rádio entre duas entidades é único, é introduzido por [Mathur et al. 2010]. Esta propriedade física é usada para gerar chaves secretas e prover funcionalidades seguras, incluindo autenticação.

Quando comparada aos trabalhos prévios discutidos, a proposta apresentada neste artigo pode ser dita inovadora uma vez que integra diferentes ideias dentro de um método formal e conciso para se implementar autenticação baseada em eventos físicos. Nas próximas seções essa proposta de autenticação é discutida e descrita em detalhes, o que possibilita seu uso em diferentes casos práticos envolvendo sistemas dotados de sensoriamento. Além disso, os conceitos de co-alocação e simultaneidade são apresentados como condições importantes para se evitar ataques do tipo *replay* e *relay*. Por fim, é apresentado um caso de estudo relacionado à instrumentação de testes de impacto veiculares, onde nosso mecanismo de autenticação é aplicado usando dados reais de sensoriamento, sem a necessidade de qualquer componente físico adicional.

3. Autenticação baseada em Eventos Físicos

3.1. Propriedades de um Evento Físico

A existência de uma camada física é provavelmente o principal fator que diferencia sistemas que usam tecnologias pervasivas dos sistemas de computação tradicionais. Nestes sistemas, um conjunto de entradas de dados do mundo físico são capturadas por diferentes sensores. Seus sinais são tratados de modo a coletar dados ou implementar estratégias de controle. Pode-se dizer que estes sensores capturam eventos físicos que por sua vez influenciam o comportamento do sistema como um todo. Sensores descrevem estes eventos medindo grandezas físicas como comprimento, massa, velocidade e temperatura.

Eventos físicos são imprevisíveis e ocorrem apenas uma vez. Isso sugere que cada evento físico é único. Na maioria dos sistemas que interagem com o mundo físico, sensores capturam diversas informações que são filtradas e descartadas quando não são relevantes à aplicação do sistema. Todavia, tais informações podem ser úteis para se caracterizar um evento físico como único. Por sua vez, sua unicidade pode ser associada a uma assinatura ou identificador do evento, extraído das informações de sensoriamento.

Além desse aspecto, quando duas entidades descrevem um mesmo evento físico, pode-se afirmar que as seguintes condições são satisfeitas:

- *Co-alocação*: as entidades estão na mesma localização física ou relativamente próximas. Esta condição é discutida por [Scannell et al. 2009] como estratégia contra ataques do tipo *relay*.

- *Simultaneidade*: as entidades capturam o evento físico ao mesmo tempo. Entende-se que esta condição é importante para evitar ataques do tipo *replay*, algo que apenas a co-alocação não provê.

Nesse trabalho, nossa proposta de autenticação é baseada nas propriedades descritas dos eventos físicos. Assumindo que um evento físico é único e que as entidades a serem autenticadas mensuram este evento, elas serão capazes de descrever o evento físico em termos de grandezas físicas, determinando sua "assinatura" ou identificador.

3.2. Modelo de Ataque

A proposta de autenticação baseada em eventos físicos considera o seguinte modelo de ataque. Duas entidades A e B trocam informações ou proveem serviços uma à outra em um sistema que interage com o mundo físico. Ambas as entidades são sensores ou possuem sensores com os quais monitoram o ambiente físico, mensurando alguma grandeza física. Por questões de segurança, A deve autenticar B antes de qualquer comunicação. Isso é feito por meio de um protocolo de autenticação qualquer onde B apresenta sua identificação e A avalia se a mesma é autêntica. Considera-se que A e B interagem entre si somente quando estão fisicamente próximas. Assim, ambas podem utilizar algum evento físico como informação complementar para a identificação.

Também se considera a existência de uma entidade maliciosa M que realiza ataques contra A e B, com o objetivo de roubar informações ou comprometer serviços. Todavia, M não possui acesso ao ambiente físico onde A e B se encontram. Por definição, M é capaz de executar as seguintes ações:

- M pode expor qualquer informação secreta compartilhada por A e B, como uma senha ou chave criptográfica, usada para estabelecer a autenticação.
- M pode "bisbilhotar" (*eavesdrop*) a comunicação entre A e B, roubando qualquer identificação apresentada por B.
- M pode tentar ataques tipo *relay* contra A, personificando B.
- M pode tentar ataques tipo *replay* contra A, usando uma identificação legítima de B obtida em uma autenticação anterior.

Propõe-se que A e B podem fortalecer a autenticação usando informações de um evento físico que M desconhece. Desta forma, as chances de sucesso de M seriam reduzidas.

3.3. Como a Autenticação baseada em Eventos Físicos funciona

Considerando o modelo de ataque descrito, a autenticação baseada em eventos físicos constitui um método que explora o uso de eventos físicos existentes em sistemas que dispõem de funcionalidades de sensoriamento. Dadas as propriedades de imprevisibilidade e unicidade de um evento físico, espera-se que seu identificador também será único ou pelo menos difícil de se deduzir. O identificador de evento pode ser mensurado pelas entidades legítimas A e B, enquanto um atacante M terá dificuldades de reproduzí-lo. Note-se que um identificador de evento não identifica uma entidade em si, mas está associado à função exercida por essa entidade no sistema. O identificador de evento pode então ser combinado com informações como um número sequencial de identificação da entidade, por exemplo, de modo a prover um mecanismo de autenticação mais forte.

Pode-se afirmar que a autenticação proposta oferece as seguintes vantagens:

1. Assume-se que um evento físico é imprevisível e único, dadas as propriedades do sistema físico a ele associado. Consequentemente, espera-se que um identificador deste evento seja difícil de se adivinhar ou deduzir.
2. Uma vez que M não possui acesso ao ambiente físico do sistema, pode-se dizer que o identificador de evento é obtido por um canal exclusivo, protegido contra ataques *eavesdropping*.
3. O identificador de evento evidencia que A e B satisfazem as condições de co-alocação e simultaneidade, protegendo a autenticação contra ataques tipo *relay*.
4. Como o identificador de evento está associado à função de A e B no sistema, o mesmo pode ser combinado com outro mecanismo para implementar uma autenticação de dois fatores. Pode-se dizer também que o identificador de evento é comparável a uma informação dinâmica que uma entidade legitima conhece.
5. Pode-se implementar esta autenticação em qualquer sistema que disponha de funcionalidades de sensoriamento, sem a necessidade de qualquer infraestrutura ou *hardware* adicional.

3.4. Descrição do Mecanismo de Autenticação

Seja E um evento físico observado e mensurado por um conjunto de N sensores $S = \{s_1, ..., s_i, ..., s_N\}$, onde cada sinal de s_i é dado por um vetor de grandezas físicas $Q = \{q_{i,1}, ..., q_{i,k}, ..., q_{i,K}\}$ correspondentes à medição realizada por s_i em cada amostra $k = 1, 2, .., K$, sendo K o total de amostras. Suponha também um conjunto de M funções $P = \{p_1, ..., p_M\}$ que implementam fusão de dados sobre o sinal de s_i. O identificador ID_i associado ao sensor s_i é dado pela seguinte equação:

$$ID_i = p_1(s_i) \oplus ... \oplus p_k(s_i) \oplus ... \oplus p_M(s_i) \tag{1}$$

onde $\oplus$ é um operador de combinação definido em função da natureza de E e/ou P.

O conjunto de funções P tem um papel importante uma vez que será o responsável por extrair características de um sinal que efetivamente determinarão o identificador de evento. É intuitivo supor que determinadas funções apresentarão melhores resultados em diferentes tipos de eventos e grandezas físicas.

Embora obtidos de um mesmo evento E, não se pode esperar que os identificadores sejam idênticos. É preciso definir uma função de comparação C e um valor de limiar Th, de modo que ID_A e ID_B correspondem a um mesmo evento físico quando:

$$C(ID_A, ID_B) \leq Th \tag{2}$$

4. Estudo de Caso: Autenticação de Sensores em um Testes de Impacto

4.1. Instrumentação de um Teste de Impacto Veicular

Um teste de impacto veicular (*vehicle crash test*) é um teste destrutivo que avalia a segurança passiva de um veículo. São baseados em um conjunto de protocolos de teste que estabelecem diferentes configurações, condições de controle e monitoramento [Insurance Institute for Highway Safety 2014].

Uma visão geral da arquitetura de instrumentação de um teste de impacto é apresentado na Figura 1. Diferentes sensores, usualmente acelerômetros e células de carga,

são posicionados em diferentes pontos do veículo, bem como na barreira de colisão e também no ATD *(Anthropomorphic Test Dummy)*, boneco humanoide usado para avaliar as consequências do impacto sobre pessoas dentro do veículo. Os dados de sensoriamento são coletados por um DAS *(Data Acquisition System)* que digitaliza e armazena os sinais. Em alguns casos o DAS possui sensores integrados, como acontece com o ATD que é também um DAS na prática. As informações de configuração e calibração dos sensores são obtidas de um banco de dados representado como *SensorDB*. O DAS envia os dados a um centro de análise, referenciado como *Control*, onde estes serão verificados e processados. Todo o resultado dos testes é armazenado em um banco de dados *TestDB*.

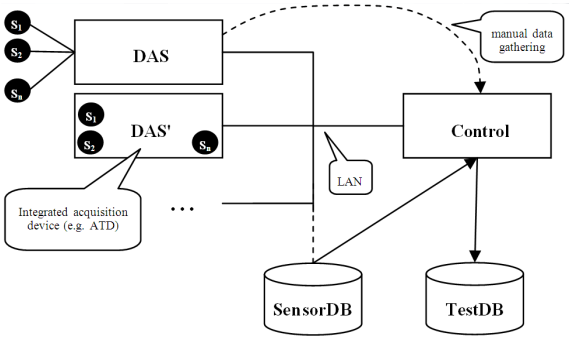


Figura 1. Visão geral da arquitetura de instrumentação de um teste de impacto.

4.2. Instanciando o Modelo de Ataque

Uma vez que os resultados de um teste afetam significativamente as decisões de consumo a respeito de um modelo de veículo, um fabricante malicioso pode ter interesse em modificar os resultados de seu próprio teste de modo a obter uma melhor avaliação de mercado. Ao mesmo tempo, este fabricante pode tentar prejudicar um concorrente, modificando os resultados de seu teste de modo a refletir uma pior avaliação. Ambos os casos estão relacionados à adulteração de dados de sensoriamento ou mesmo à substituição de sensores confiáveis por sensores corrompidos. Ambos os casos também implicam a possibilidade de existência de ataques internos, uma vez que os testes são usualmente conduzidos por laboratórios independentes, em ambientes onde os fabricantes não deveriam ter acesso.

Um exemplo está relacionado com a avaliação de risco de ferimento na cabeça do ATD. Dois indicadores são usualmente aceitos: o HIC (*Head Injury Criteria*) e o vetor resultante de aceleração máxima, ambos obtidos a partir de acelerômetros posicionados dentro da cabeça do ATD [Insurance Institute for Highway Safety 2014]. Um atacante pode modificar dados dos acelerômetros, comprometendo o cálculo destes indicadores e consequentemente modificando o resultado dos testes. Tal ataque poderia ser feito pela substituição do sensor, ou apenas do sinal processado pelo DAS por um sensor/sinal gerado em um teste diferente.

Desse modo, considera-se uma instância do modelo de ataque onde o atacante pode realizar as seguintes ações:

- Interceptar e modificar o sinal de um sensor legítimo antes que este seja recebido por *Control* (ataque tipo *relay*).
- Substituir o sinal de um sensor legítimo por outro sensor também legítimo, todavia obtido em um teste prévio (ataque tipo *replay*).

Em um ataque onde um sensor legítimo pode ser usado em um ambiente de teste manipulado para gerar resultados maliciosos, abordagens tradicionais de autenticação não são suficientes. Se um sensor legítimo é usado, nenhuma suspeita será levantada quanto à sua identificação. Neste caso, um método de autenticação que satisfaça as condições de co-alocação e simultaneidade pode oferecer maior confiabilidade.

4.3. Propondo uma Autenticação Baseada em Eventos Físicos

Como resposta para o modelo de ataque descrito, este trabalho propõe implementar uma autenticação baseada em eventos físicos para alguns sensores posicionados dentro do ATD durante um teste de impacto. O evento físico é a colisão em si. Mesmo quando um único protocolo de testes é considerado, cada colisão produz um resultado único devido à resposta cinética dos materiais utilizados na construção do veículo. Quando sensores descrevem um mesmo evento físico, proveem evidências de que foram utilizados no mesmo teste de impacto, satisfazendo as condições de co-alocação e simultaneidade.

O processo de autenticação ocorre da seguinte forma. Primeiramente define-se *Control* como a entidade responsável por autenticar um sensor antes de aceitar seus dados. Supõe-se também que cada sensor já dispõe de algum mecanismo para prover uma identificação (por exemplo, um código de fabricação), embora esse mecanismo não possa atestar qualquer informação sobre tempo e local onde os dados foram obtidos. Assim, a identificação do sensor deve ser complementada com um identificador de evento físico. Este identificador é calculado usando dados brutos de sensoriamento, sendo determinado simultaneamente ao evento. Uma vez obtido o identificador de evento, este é verificado por meio da função de comparação definida para tal. Para cada sinal, se a função de comparação satisfaz o critério de limiar estabelecido, o sensor é autenticado por *Control*.

4.4. Cenários de Autenticação

O protocolo de testes escolhido determina que a instrumentação do ATD inclua acelerômetros e/ou células de carga em seu corpo e membros. Durante a colisão, as partes do ATD sofrem a ação de diferentes forças co-relacionadas. As forças que agem na cabeça do ATD são também observadas em seu pescoço e peito. O termo *sensores vizinhos* será usado para se referir a grupos de sensores que compartilham esta propriedade.

Também é comum que protocolos de teste especifiquem o uso de sensores redundantes, prevenindo a perda de sinais em função de falhas na instrumentação durante a colisão. Este aspecto pode ser útil para a autenticação uma vez que sensores posicionados no mesmo local capturam praticamente as mesmas informações sobre o evento físico.

Neste estudo, são autenticados sensores posicionadas na cabeça e no pescoço do ATD. A razão desta escolha é que estes sensores são os mais significativos na avaliação dos resultados do teste de impacto. Diversos programas de avaliação de segurança passiva em veículos propõe o valor do HIC como decisivo na pontuação a ser atribuída ao veículo. Além disso, os sensores de cabeça e pescoço são sensores vizinhos, o que facilita determinar a similaridade entre seus sinais. Também foi escolhido um protocolo de testes que determina o uso de sensores redundantes na cabeça do ATD. Deste modo, é possível determinar dois cenários de teste distintos, a saber:

- *Cenário NS (Sensores Vizinhos)*: um identificador de evento resulta de sensores posicionados em uma mesma "vizinhança" dentro do veículo.

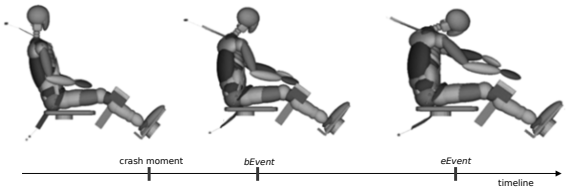


Figura 2. Movimento do ATD na linha de tempo da colisão

- *Cenário RS (Sensores Redundantes)*: um identificador de evento resulta de sensores redundantes.

4.5. A Colisão como Evento Físico

Os sensores da cabeça do ATD são acelerômetros triaxiais cuja grandeza física é dada em G's (sendo $G \approx 9,8m/s^2$). O sensor de pescoço é uma célula de carga triaxial que reporta medidas de força em Newtons (N). Ambas as grandezas são inter-comparáveis por meio da equação $F = ma$. Mesmo sem conhecer a massa m do veículo, é possível normalizar ambos os sinais. Além disso, ao invés de avaliar cada eixo do sinal de forma separada, opta-se pela resultante do vetor de aceleração/força, dada pela equação:

$$R_s = norm\left(\sqrt{X_s^2 + Y_s^2 + Z_s^2}\right) \tag{3}$$

onde $norm()$ é a função de normalização, R_s é a resultante do sinal de um sensor s e X_s, Y_s e Z_s são as componentes em cada eixo.

São definidas também as variáveis $bEvent$ e $eEvent$ como pontos na linha do tempo onde o evento físico tem início e fim, respectivamente (Figura 2). No exato instante da colisão, o ATD manterá sua inércia e continuará sua trajetória dentro do veículo por aproximadamente 50 ms. Considera-se como $bEvent$ o instante de tempo aproximado quando o peito do ATD desacelera gradualmente pela tensão do cinto de segurança. Por sua vez $eEvent$ deve coincidir com o pico de desaceleração do ATD, determinante no cálculo do HIC e do vetor resultante de aceleração máxima. Após $eEvent$, a trajetória do ATD torna-se imprevisível em função do impacto contra o *air-bag* do veículo, o que dificulta identificar e analisar a relação entre as forças resultantes do impacto.

4.6. O Identificador ID_i e a Função de Comparação C

Um identificador de evento ID_i pode ser obtido extraindo-se características relevantes do sinal de cada sensor. Isso é feito combinando-se dois métodos de fusão de dados. O primeiro é algoritmo de quantização proposto por [Liu et al. 2009] que combina Filtro de Médias Móveis (MAF) com passo variável como fator de compressão e discretização do sinal para eliminar operações de ponto flutuante. Este método resulta na redução do número de amostras e requer menor esforço computacional. O segundo é uma função de extração de informações projetada a partir de observações empíricas dos sinais de diferentes sensores posicionados na cabeça e pescoço do ATD. Foi verificado que os sinais de sensores usados em um mesmo teste apresentam um padrão similar em termos de frequências médias. Isto pode ser evidenciado nos valores de máximos e mínimos locais do sinal, quando quantizações com MAF usando diferentes tamanhos de janela são comparadas (Figura 3).

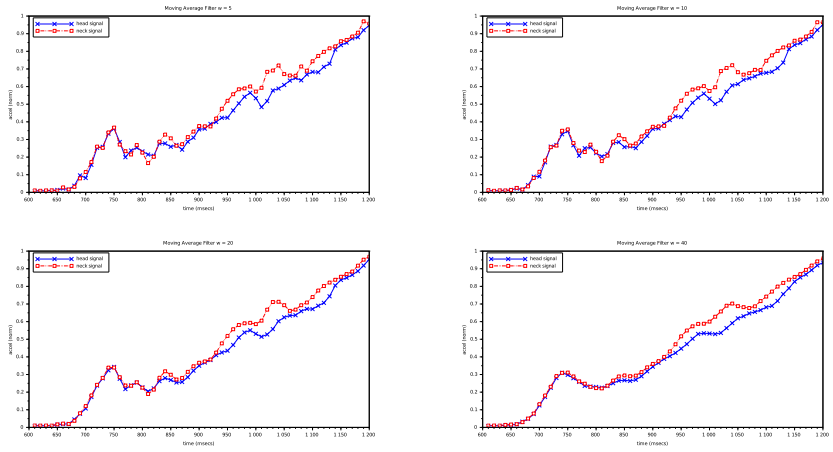


Figura 3. Aplicando MAF com diferentes tamanhos de janela w

Uma descrição formal do método usado para obtenção do identificador de evento é obtida instanciando-se a equação 1. Sejam $M = 3$ e $P = \{p_1, p_2, p_3\}$, tem-se:

$$ID_i = p_1 \oplus p_2 \oplus p_3 \tag{4}$$

onde $\oplus$ é definido como um operador de função composta.

Primeiramente define-se p_3 como o algoritmo de quantização de [Liu et al. 2009]:

$$p_3 = quant(R_s, w_i, step) \tag{5}$$

onde $quant(...)$ é uma função que aplica o algoritmo MAF em R_s com um tamanho de janela w_i e fator de compressão $step$.

Para extrair informação de frequências médias de um sinal R_s, são obtidos dois sinais quantizados diferenciados pelo tamanho da janela w_i usada pelo algoritmo MAF. Sejam w_a e w_b tamanhos diferentes de janela onde $w_a < w_b$, informação associada a frequências médias de R_s podem ser obtidas calculando-se p_2 como:

$$p_2 = p_3(R_s, w_b, step) - p_3(R_s, w_a, step) \tag{6}$$

Como último passo, a informação resultante de p_2 é normalizada e discretizada dentro de uma faixa de valores $range$, sendo que p_1 é definida como:

$$p_1 = int(norm(p_2) * range) \tag{7}$$

onde $int(...)$ é o truncamento de cada valor em p_2 e $range$ é a faixa de discretização.

Por sua vez, a função de comparação C é dada como uma medida de similaridade entre dois identificadores de um mesmo evento físico. Neste experimento foram obtidos bons resultados combinando-se Correlação de Pearson e a função Coerência usada por [Mayrhofer and Gellersen 2007]. Da mesma forma como desenvolvido para ID_i, a especificação formal de C foi obtida instanciando-se a equação 2 da seguinte forma:

$$C(ID_A, ID_B) = (1 - max(correl(ID_A, ID_B), 0) * (1 - coher(ID_A, ID_B)) \tag{8}$$

onde $correl(...)$ é a Correlação de Pearson e $coherence(...)$ é a função Coerência. Sobre a correlação é importante observar que valores negativos são considerados como zero.

5. Experimento Prático e Resultados

5.1. Descrição do Experimento

A autenticação é avaliada por meio de um experimento prático usando dados de testes de impacto da NHTSA [National Highway Traffic Safety Administration 2015]. Foram coletados 100 casos de teste mais recentes realizados com o protocolo de colisão frontal (*vehicle into barrier*) [National Highway Traffic Safety Administration 2012]. Os arquivos de dados disponibilizados pela NHTSA constituem séries temporais amostradas a uma frequência de 1 Khz durante um intervalo de 500 milissegundos (ms). Os dados de teste foram divididos em dois grupos: grupo de análise, onde diferentes métodos e ajustes foram testados de modo a se determinar os melhores resultados; e o grupo de validação, usado para se obter as métricas de desempenho das funções selecionadas.

5.2. Valores Atribuídos às Variáveis de Sintonia

O experimento utiliza as equações definidas na seção 4.6 para gerar os identificadores de evento, bem como a função de comparação. O método é sensitivo aos valores atribuídos para o intervalo de dados de análise determinado por $bEvent$ e $eEvent$, aos tamanhos de janela w_a e w_b usados no MAF, ao fator de compressão $step$ e ao fator de discretização $range$. Os respectivos valores e as justificativas para sua atribuição são discutidos a seguir.

Os valores de $bEvent$ e $eEvent$ foram escolhidos de modo a descartar dados amostrados antes dos primeiros 50 ms após a colisão e após a desaceleração máxima do ATD. $bEvent$ e $eEvent$ foram definidos testando-se valores nas faixas de 40 até 60 ms e 100 a 150 ms, respectivamente. Bons resultados foram obtidos com $bEvent$ em 60 ms e $eEvent$ em 120 ms. Tem-se assim um sinal R_s de 600 pontos de dados, para cada sensor.

As janelas w_a e w_b são variáveis críticas para a identificação do evento. Primeiramente w_a deve ser definido de modo a eliminar apenas frequências altas indesejáveis. Por sua vez, w_b é definido com o objetivo de preservar frequências baixas. Quando a equação 6 é aplicada, o resultado será essencialmente a informação de frequências médias do sinal. Os valores de w_a e w_b foram determinados empiricamente, testando-se valores na faixa entre 5 e 100 pontos de dados. Janelas de tamanho menor mostraram-se pouco efetivas enquanto janelas de tamanho maior eliminam praticamente toda a informação de frequência média. Os melhores resultados foram obtidos com $w_a \approx 10$ e $w_b \approx 40$.

O fator de compressão $step$ determina quanto do sinal de dados será comprimido. Um fator alto resulta em menos informação e consequente redução na acurácia do processo de autenticação. Bons resultados foram obtidos usando-se $step = 10$, que implica em 90% de compressão e redução dos 600 pontos de R_s a 60 pontos de dados.

Por fim, $range$ é definido de modo a se obter uma representação binária de ID_i. Atribuindo-se $range = 8$, os valores normalizados são distribuídos em valores discretos entre -7 e 8. Isto permite representar cada ponto de dados em ID_i como um valor hexadecimal entre `0x0` e `fx0`, ocupando 2 bytes. Uma vez que p_3 retorna um vetor de 60 pontos, tem-se um identificador de evento de 30 *bytes*, ou 240 *bits*.

5.3. Resultados do Cenário NS

Uma vez que a autenticação trabalha com sensores da cabeça e pescoço do ATD, considera-se que o atacante pode tentar corromper um desses sensores para comprometer os resultados do teste. Para o cenário NS duas situações são inicialmente descritas:

- **Ataque HxH**: O atacante compromete o sensor da cabeça do ATD usando um sinal legítimo obtido de um sensor de cabeça usado em um teste diferente.
- **Ataque NxN**: O atacante compromete o sensor do pescoço do ATD usando um sinal legítimo obtido de um sensor de pescoço usado em um teste diferente.

Com o objetivo de verificar a robustez do método de autenticação proposto, foram investigadas duas outras situações. Supõe-se que o atacante conhece o mecanismo de autenticação e portanto sabe que o identificador de evento obtido de um sensor de cabeça será comparado ao obtido pelo pescoço, ou vice-versa. Com a intenção de confundir o processo de autenticação, o atacante pode tentar substituir um sinal da cabeça do ATD por um sinal do pescoço. A ideia por trás desta estratégia é a possibilidade de que a comparação entre dois sinais do pescoço de um ATD, ainda que de eventos diferentes, apresente tanta similaridade quanto os sinais da cabeça e pescoço de um mesmo evento. Para testar um eventual ataque como este são acrescidas as seguintes situações:

- **Ataque HxN**: O atacante compromete o sensor da cabeça do ATD usando um sinal legítimo obtido de um sensor de pescoço usado em um teste diferente.
- **Ataque NxH**: O atacante compromete o sensor do pescoço do ATD usando um sinal legítimo obtido de um sensor de cabeça usado em um teste diferente.

A probabilidade de sucesso de tais ataques é verificada por meio do seguinte experimento. Para cada caso de teste $k = 1, 2, ..., K$, sendo K o total de casos analisados, determina-se o i-ésimo identificador de evento para os sensores da cabeça e pescoço do ATD, obtendo-se assim $ID_{i,H}$ e $ID_{i,N}$, respectivamente. Em seguida calcula-se $C_{leg,i} = C(ID_{i,H}, ID_{i,N})$ usando-se a equação 8. Simula-se então a tentativa de fraude do i-ésimo identificador de evento, substituindo-se os dados brutos do sensor sob ataque em cada situação descrita com os dados do j-ésimo evento, sendo $j = 1, 2, ..., K$ e $j \neq i$.

Espera-se que o algoritmo de autenticação proposto determine $C_{leg,i} \leq Th$, satisfazendo a equação 8. Em contrapartida, espera-se que cada situação de ataque simulado j resulte em $C_{X,j} > Th$, onde $X = \{HH, NN, HN, NH\}$. Casos de ataque duplicados não desconsiderados pois C é comutativa e portanto $C(ID_i, ID_j) = C(ID_j, ID_i)$.

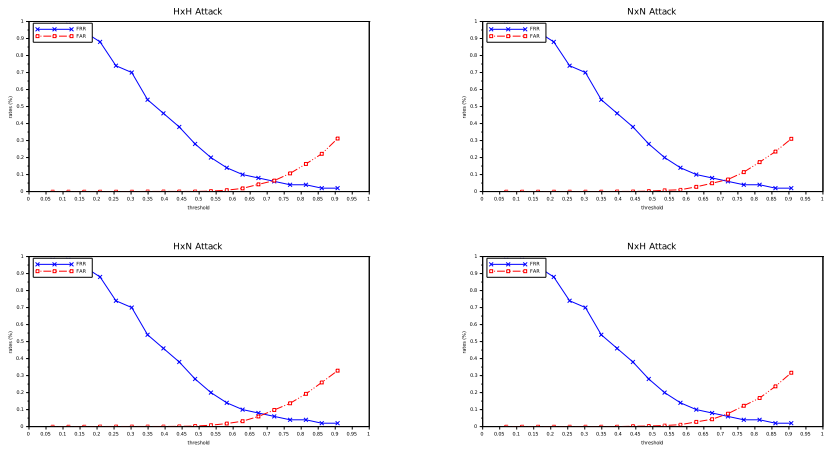


Figura 4. FRR e FAR para o Cenário NS, com diferentes valores de Th

A eficiência do método de autenticação proposto foi avaliada usando métricas comuns em sistemas biométricos, que são a FRR (*False Rejection Rate*) e FAR (*False*

Tabela 1. Taxas de desempenho da autenticação - Cenário NS com $Th = 0.5$

	HxH		**NxN**		**HxN**		**NxH**	
	OK	**NOK**	**OK**	**NOK**	**OK**	**NOK**	**OK**	**NOK**
Sensores legítimos	41	9	41	9	41	9	41	9
Sensores sob ataque	10	1215	9	1216	14	1211	7	1218
FRR	18%		18%		18%		18%	
FAR	0.8%		0.7%		1.1%		0.6%	
F-Measure	0.81		0.82		0.78		0.83	

Acceptance Rate). Primeiramente, usando o grupo de dados de análise, verificou-se como FRR e FAR se comportam em função do valor de limiar Th, conforme apresentado na Figura 4. Pode-se observar que o valor de erro CER (*Crossover Error Rate*) ocorre quando $0.65 < Th < 0.75$ em todas as situações de ataque. Uma vez que o problema tratado é a autenticação, pode-se afirmar que um menor valor de FAR é melhor do que um baixo valor de FRR. Se uma autenticação legítima é negada, a autenticidade do sensor pode ser verificada por um segundo método. Por outro lado, se uma tentativa de ataque é aceita pelo processo de autenticação, nenhuma suspeita será levantada e tem-se uma falha de segurança. Com base nestas considerações, definiu-se $Th = 0.5$ na expectativa de se obter um FRR superior a 10% mas ao mesmo tempo manter FAR abaixo de 1%.

Por fim o desempenho da autenticação é determinado usando-se o grupo de dados de validação. A Tabela 1 resume os resultados. Como esperado, FAR se mantém abaixo de 1% para todas as situações de ataque, exceto *HxN*. Como efeito colateral, tem-se FRR em 18%, o que pode ser considerado um resultado razoável pois compromete a usabilidade do método em razão de um excessivo número de sensores legítimos que serão considerados suspeitos de ataque. Conclui-se que as funções de geração do identificador de evento e de comparação necessitam ser melhorada, visando a redução do valor de FRR.

5.4. Resultados do Cenário RS

No cenário RS são usados sensores redundantes da cabeça do ATD. Dois sensores posicionados no mesmo local apresentam entre si uma alta correlação, facilitando a obtenção de um identificador. Em compensação, a autenticação de um cenário RS pode não ser factível se for considerado que um atacante pode substituir os sensores do ATD fisicamente, pois nesse caso ele pode substituir também o sensor redundante. Independente disso a autenticação com sensores redundantes pode ser útil em situações quando não se tem acesso físico aos sensores e o ataque consiste em se substituir os dados de sensoriamento obtidos pelo DAS, desde que a coleta de dados dos sensores principais e redundantes se dê por canais diferentes de acesso.

Repetindo o mesmo experimento descrito na seção anterior, o grupo de dados de análise foi usado para avaliar a variação de FRR e FAR em função do limiar Th. A diferença é que neste caso apenas a situação HxH se aplica por não haver sensores redundantes no pescoço. Os valores de FRR e FAR são exibidos na Figura 5. Pode-se observar que o CER ocorre durante um intervalo maior quando $0.3 < Th < 0.5$, onde tanto FAR quanto FRR apresente valores muito próximos de zero. Isto acontece devido a elevada similaridade entre os sinais dos sensores redundantes. Consequentemente definiu-se $Th = 0.4$ com a expectativa de valores praticamente ótimos para FRR e FAR.

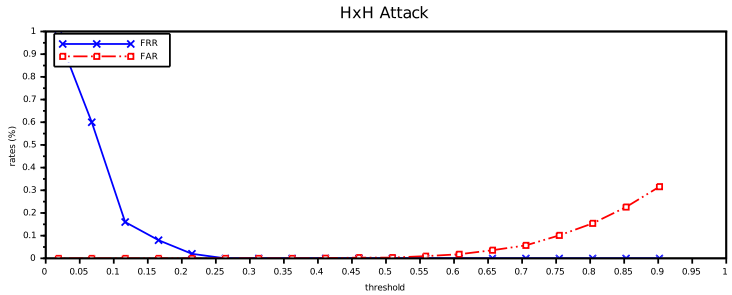


Figura 5. FRR e FAR para o Cenário RS, com diferentes valores de Th

Usando o grupo de dados de validação, foram calculadas as taxas de desempenho da autenticação para ao cenário RS, conforme a Tabela 2. Como esperado, ambos os valores de FRR e FAR ficam próximos de zero, mostrando que sensores redundantes permitem a autenticação baseada em eventos físicos com elevada eficiência.

Tabela 2. Taxas de desempenho da autenticação - Cenário RS com $Th = 0.4$

	HxH	
	OK	**NOK**
Sensores legítimos	50	0
Sensores sob ataque	1	1224
FRR	0%	
FAR	0.08%	
F-Measure	0.99	

6. Conclusão

Este artigo apresentou uma proposta de implementação de autenticação de sensores baseada em eventos físicos. Pode-se afirmar que esta pesquisa é importante dada a necessidade de mecanismos de autenticação em sistemas com sensoriamento. Além disso, o método proposto produz evidências de co-alocação e simultaneidade das entidades envolvidas, podendo ser implementado em qualquer sistema de sensoriamento sem a necessidade de infraestrutura adicional. Os argumentos são fortalecidos pelos resultados práticos obtidos no estudo de caso. Foi demonstrado que o evento da colisão é capturado por diferentes sensores que preservam informações comuns entre si. Dada a similaridade dos sinais, é possível implementar a autenticação atestando que os sensores operam no mesmo tempo e local, evitando assim ataques tipo *replay* e *relay*. Ambos os cenários NS e RS foram avaliados com resultados promissores.

Este estudo também abre diversas possibilidades em termos de pesquisas futuras. Primeiramente, a investigação quanto à possibilidade de se aplicar esta autenticação em soluções de IoT e CPS. Outra linha de futuros estudos será a natureza aleatória dos eventos físicos, visando descrever quais classes de eventos são mais apropriados para prover este tipo de autenticação. Por fim, a investigação de métodos e estratégias de fusão de dados e reconhecimento de padrões pode auxiliar na criação de um arcabouço de algoritmos para implementação deste tipo de autenticação em diferentes classes de sistemas físicos.

Referências

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols and Applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347 –2376.

Gafurov, D., Snekkenes, E., and Bours, P. (2007). Gait authentication and identification using wearable accelerometer sensor. *2007 IEEE Workshop on Automatic Identification Advanced Technologies - Proceedings*, pages 220–225.

Hsieh, W.-B. and Leu, J.-S. (2011). Design of a time and location based One-Time Password authentication scheme. In *Wireless Communications and Mobile Computing Conference IWCMC 2011 7th International*, pages 201–206.

Insurance Institute for Highway Safety (2014). Moderate Overlap Frontal Cashworthiness Evaluation. Technical Report September.

Liu, J., Zhong, L., Wickramasuriya, J., and Vasudevan, V. (2009). uWave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657–675.

Mathur, S., Reznik, A., Ye, C., Mukherjee, R., Rahman, A., Shah, Y., Trappe, W., and Mandayam, N. (2010). Exploiting the physical layer for enhanced security. *IEEE Wireless Communications*, 17(5):63–70.

Mayrhofer, R. and Gellersen, H. (2007). Shake Well Before Use: Authentication Based on Accelerometer Data. In *5th International Conference, PERVASIVE 2007*, pages 144–161.

Mitra, S., Wongpiromsarn, T., and Murray, R. M. (2013). Verifying Cyber-Physical Interactions in Safety-Critical Systems. *IEEE Security & Privacy*, 11(4):28–37.

National Highway Traffic Safety Administration (2012). Laboratory Test Procedure For New Car Assessment Program Frontal Impact Testing. Technical Report September.

National Highway Traffic Safety Administration (2015). NHTSA Vehicle Crash Test Database. http://www-nrd.nhtsa.dot.gov/database/veh/veh.htm. Accessed: 2015-05-05.

Priya, L. C. and Patil, S. D. (2014). A Survey on Sensor Authentication in Dynamic Wireless Sensor Networks. *International Journal of Computer Science and Information Technology Research*, 2(2):454–461.

Scannell, A., Varshavsky, A., Lamarca, A., and de Lara, E. (2009). Proximity-based authentication of mobile devices. *International Journal of Security and Networks*, 4(1/2):4–16.

Suh, G. E. and Devadas, S. (2007). Physical unclonable functions for device authentication and secret key generation. *Proceedings - Design Automation Conference*, pages 9–14.

Wu, F.-J., Chu, F.-I., and Tseng, Y.-C. (2011). Cyber-physical handshake. *ACM SIGCOMM Computer Communication Review*, 41(4):472.

Yampolskiy, R. V. and Govindaraju, V. (2008). Behavioural biometrics: a survey and classification. *International Journal of Biometrics*, 1(1):81–113.

Um controle de associações resistente a ataques Sybil para a disseminação segura de conteúdo da IoT*

Danilo Evangelista[1], Eduardo da Silva[1,2], Michele Nogueira[1], Aldri Santos[1]

[1]Núcleo de Redes Sem-Fio e Redes Avançadas (NR2) – UFPR

[2]Departamento de Informática Instituto Federal Catarinense – Araquari, SC – Brazil

eduardo@araquari.ifc.edu.br, {dfrevangelista,michele,aldri}@inf.ufpr.br

Abstract. *The Internet of Things (IoT) comprises a diversity of heterogeneous objects that collect and disseminate data for Internet applications. The content dissemination on these networks is subject to various malicious actions, such as the identities impersonation made by Sybil attack. Although there are several Sybil attack detection techniques, such as LSD, they are costly, disregard heterogeneous devices and attackers with stolen identities. This work presents a association control mechanism for IoT, called SA^{2}CI, which prevents the access Sybil attackers on content dissemination service. The SA^{2}CI uses elliptic curve cryptography (ECC), physical unclonable functions (PUF), and identities receipts. The ECC technique provides key distribution and stablishes a secure channel with low cost. The physical unclonable function enables the verification of a device identity. Then, a receipt identity, calculated by the a device PUF, ensures its legitimacy. The effectiveness and efficiency of SA^{2}CI were evaluated in IoT network using Network Simulator 3 (NS3).*

Resumo. *A Internet das coisas (IoT) compreenderá uma diversidade de objetos heterogêneos integrados que coletam e disseminam conteúdo com diferentes propósitos de aplicações. Logo, uma disseminação segura na IoT é essencial, visto que ela está sujeita a diversas ações maliciosas, como a personificação de identidades por ataques Sybil que buscam violar a confidencialidade do conteúdo disseminado. Contudo, as técnicas existentes de detecção de ataques Sybil desconsideram o uso de identidades roubadas e de dispositivos heterogêneos, e em geral são custosas. Este trabalho apresenta um mecanismo de controle de associações para IoT, chamado SA^{2}CI, que previne o acesso de atacantes Sybil ao serviço de disseminação de conteúdo. O SA^{2}CI emprega criptografia de curvas elípticas (ECC) que prover uma distribuição de chaves e criação de um canal seguro com baixo custo, aplica funções não clonáveis (PUF) na comprovação da identidade dos dispositivos, e recibos de identidade para garantir a legitimidade dos dispositivos. Uma avaliação feita no NS3 mostra a eficácia do SA^{2}CI no controle de associações diante de ataques Sybil.*

1. Introdução

A crescente necessidade de serviços personalizados e autônomos tem possibilitado o desenvolvimento da Internet das coisas (IoT). Uma IoT permite que objetos (coisas) como

*Artigo parcialmente apoiado pelo CNPq projetos nº 488004/2013-6, 486393/2013-5 e 310609/2014-3

geladeiras, máquinas de lavar, aparelhos de ar-condicionado e dispositivos computacionais estejam conectados com as pessoas em qualquer momento e lugar. Um serviço fundamental nas IoTs é a disseminação de conteúdos, que possibilita a implementação eficaz de outros serviços como o monitoramento do consumo de água, da temperatura de um cômodo, e o acompanhamento dos dados vitais de uma pessoa, em tempo real. Isso promove um maior conforto e bem estar às pessoas, propiciando a implementação de ambientes inteligentes, como casa, hospitais e rodovias [Perera et al. 2014].

Uma vez que a disseminação de conteúdo é base ao provimento serviços da IoT, ela demanda garantia da sua segurança contra ações maliciosas. Dentre as ações maliciosas numa IoT destaca-se a personificação de identidades realizada pelo ataque Sybil. Em resumo, um atacante Sybil usa identidades forjadas visando o acesso à rede, incluindo o acesso à disseminação dos dados [Wallgren et al. 2013]. Logo, um atacante autenticado numa rede IoT busca alcançar vantagens como o uso de recursos não autorizados, a obtenção de informações vitais, infligindo a confidencialidade e a privacidade dos usuários da rede. Portanto, a qualidade dos serviços suportados pela disseminação de conteúdo é afetada, comprometendo a segurança dos dados trafegados na rede.

Diversas técnicas são encontradas na literatura para detectar ataques Sybil, que são classificadas neste trabalho em três grupos: baseadas nas características da rede [Vamsi and Kant 2014], em criptografia [Park et al. 2013], e no relacionamento entre vizinhos [Quercia and Hailes 2010]. A técnica baseada nas características das redes considera aspectos dos nós, como a força do sinal recebido (RSS) e a mobilidade, para identificar um ataque. Apesar de considerar a restrição de recursos dos dispositivos, ela não é eficaz contra ataques Sybil [Evangelista et al. 2015]. Já a técnica baseada em criptografia emprega chaves simétricas e assimétricas para garantir a irretratabilidade das identidades de uma rede. No entanto, ela necessita de constantes atualizações dos novos pares de chaves, sobrecarregando a rede. Por fim, a técnica baseada no relacionamento entre os vizinhos considera as opiniões sobre um nó emitidas por seus vizinhos. Contudo, um nó malicioso que simule um comportamento legítimo pode ludibriar o sistema. Logo, surge a necessidade de uma solução eficaz contra a associação de atacantes à rede e que considere as restrições de recursos dos dispositivos da IoT.

Este trabalho apresenta um mecanismo chamado SA^2CI (***S**ybil **A**ttack **A**ssociation **C**ontrol for **I**oT*) que previne associações de atacantes Sybil na disseminação de conteúdo da IoT. O SA^2CI atua entre as camadas de rede e aplicação sendo um *middleware* para apoiar a disseminação segura de conteúdo. Ele emprega criptografia de curvas elípticas (ECC) para a distribuição de chaves seguras a um baixo custo computacional e a criação de canais seguros entre dispositivos heterogêneos. Com um canal seguro, o SA^2CI aplica funções não clonáveis (PUF), extraídas do hardware dos dispositivos, para a comprovação da sua identidade, e associa aos recibos de identidade garantindo a legitimidade dos dispositivos. O SA^2CI foi avaliado por meio de simulações e os resultados comprovam a sua resistência a ataques Sybil, assegurando a disseminação segura de conteúdo.

O restante do artigo está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados sobre detecção de ataques Sybil. A Seção 3 descreve mecanismo SA^2CI, suas fases de operação e seu funcionamento diante de ataques. Já a Seção 4 mostra uma avaliação do SA^2CI para suportar disseminação de conteúdo na IoT diante SA's e uma comparação com o mecanismo LSD. Por fim, a Seção 5 conclui o trabalho.

2. Trabalhos Relacionados

A necessidade de se garantir uma disseminação segura na Internet das coisas (IoT) torna-se providencial para o provimento dos serviços implementados por ela. As técnicas existentes de detecção de ataques Sybil encontradas na literatura geralmente são classificadas em **características das redes**, **criptografia**, e **relacionamento entre vizinhos**. Como a IoT requer soluções leves, dinâmicas, e eficientes, o uso destas técnicas no âmbito da IoT pode acarretar numa disseminação ineficaz. Isto acontece por que os trabalhos que empregam tais técnicas desconsideram estes requisitos.

Os trabalhos de detecção do ataque Sybil baseados nas características das redes em geral ressaltam a necessidade da solução ser leve e eficaz [Vamsi and Kant 2014]. Dentre os trabalhos encontrados na literatura, o mecanismo Lightweight Sybil Attack Detection (LSD) [Abbas et al. 2013], que emprega a técnica das características da rede, é uma solução leve e dinâmica. O LSD identifica o ataque Sybil através das características do RSS e da mobilidade dos dispositivos. Para realizar a detecção de um atacante, estas características são avaliadas por um conjunto de dispositivos da rede de modo à verificar o comportamento durante uma associação de um novo participante. Em seguida um dispositivo da rede armazena o RSS e a identidade numa tupla, e a identificação de um atacante ocorre quando esta identidade não condiz com o valor estimado pelos dispositivos através do seu RSS. Contudo, essa abordagem é vulnerável quando um atacante emprega identidades roubadas uma vez que ela ignora a irretratabilidades das identidades dos dispositivos da rede. Como o LSD adota a técnica de características das rede, ele requer de uma série de análises, e isto acarreta a uma alta taxa de falsos positivos.

Para uma solução garantir a disseminação segura diante de ataques Sybil na IoT é necessário que ela seja dinâmica, leve, escalar, confidencial e trate a irretratabilidade das identidades dos dispositivos. O estabelecimento de um canal seguro entre os componentes de uma rede propicia a disseminação de dados de forma confidencial, onde apenas os dispositivos em comum acordo de chaves tem acesso à informação. Dentre as técnicas existentes as quais asseguram um canal seguro, as cuvas elípticas [Mahalle et al. 2012, Chatzigiannakis et al. 2011] despontam como promissora para o uso na IoT, uma vez que ela demanda uma baixa quantidade energética para gerar pares de chaves, podendo ser empregada numa gama maior de dispositivos com ou sem restrições. Já a irretratabilidade dos dispositivos é alcançada por meio de técnicas de comprovação e garantia de identidades. Logo, o uso da função não clonável (PUF) [Choden Konigsmark et al. 2014, Zheng and Potkonjak 2014] associada com recibos de identidades [Wu et al. 2008] proporcionam a irretratabilidade das identidades dos dispositivos. O recibo de identidade permite identificar um dispositivo de forma única, sendo ele apenas necessário numa associação escalar a uma rede IoT. Assim, atendendo estes requisitos seria possível suportar uma disseminação segura diante de ataques Sybil.

3. Mecanismo contra Ataque Sybil

Para apoiar a disseminação de dados segura contra ataques Sybil é o proposto um mecanismo para controle de associações, chamado SA^2CI (*Sybil Attack Association Control for IoT*). O SA^2CI atua como um *middleware*, entra as camadas de rede e aplicação, e assegura que os dispositivos (nós) disseminem conteúdos de forma segura e resiliente à ataques Sybil. Ele usa informações da camada de rede, como o encaminhamento de dados

entre os seus nós, e da camada de aplicação, como a troca de chaves e o estabelecimento de segredos compartilhados. O SA^2CI assume nós com ou sem restrições de recursos.

3.1. Modelo da rede e do ataque

A rede IoT é composta por um conjunto de n nós, móveis ou fixos, denotado por $\mathcal{N} = \{N_1, N_2, ..., N_n\}$. Esses nós possuem um comportamento *legítimo* ($\mathcal{N}_L$) ou um comportamento Sybil ($\mathcal{N}_S$), sendo que $\mathcal{N}_L \subseteq \mathcal{N}$, $\mathcal{N}_S \subseteq \mathcal{N}$ e $\mathcal{N}_L \cup \mathcal{N}_S = \mathcal{N}$. Um nó ($\mathcal{N}_L$) não se torna um nó ($\mathcal{N}_S$) ao longo do tempo, enquanto que ($\mathcal{N}_S$) pode se passar por legítimo. Cada nó legítimo N_i possui apenas uma identidade, denotada por Id_i. O conjunto de todas as identidades do sistema é denotado por Id.

O nós que não possuem restrição de recursos são denotados por $\mathcal{N}_{KDC}$ enquanto $\mathcal{N}_{MOV}$ denota os nós com recursos limitados. A relação $\mathcal{N}_{KDC} \cap \mathcal{N}_{MOV}$ não existe, pois um dado nó $N_x \in \mathcal{N}_{KDC}$ não pode ser membro de $\mathcal{N}_{MOV}$, e vice-versa. Os nós do conjunto $\mathcal{N}_{KDC}$ são chamados de nós N_{KDC} e os demais são denominados nós N_{MOV}. O relacionamento entre um N_{KDC} e um N_{MOV} ocorre a partir de uma função sobrejetora $F : \mathcal{N}_{KDC} \rightarrow \mathcal{N}_{MOV}$, em que os elementos do subconjunto $\mathcal{N}_{MOV}$ estão associados a um elemento de $\mathcal{N}_{KDC}$. Por outro lado, não existe uma função $F : \mathcal{N}_{MOV} \rightarrow \mathcal{N}_{KDC}$, visto que os $\mathcal{N}_{KDC}$ não podem se associar aos $\mathcal{N}_{MOV}$ por conta da limitação de recurso destes.

O meio de transmissão é sem fio, baseado no padrão 802.15.4. A comunicação acontece a partir de um canal assíncrono sujeito à perda de pacotes devido à mobilidade. Os nós da rede, $\mathcal{N}_{MOV}$ e $\mathcal{N}_{KDC}$, disseminam um dado conteúdo para uma origem, sendo que a estratégia de disseminação não impõe restrições na capacidade de detecção do SA^2CI. Sem a perda da generalidade, para evitar que os nós do conjunto $\mathcal{N}_{MOV}$ gastem seus recursos utilizando o *flooding*, neste trabalho é adotado um modelo de disseminação baseado nas capacidades dos dispositivos [Le et al. 2012]. O raio de comunicação de cada nó varia entre 10 e 100 metros dependendo da capacidade de transmissão de sua antena. A disseminação dos dados ocorre através do envio de fluxos de dados de 64 *bytes*, partindo de uma ou mais origens até um destino.

Um serviço de disseminação de dados na IoT é vulnerável a diversos tipos de ataques. Entre esses, encontra-se o ataque Sybil, em que um nó adversário cria uma ou mais identidades com o objetivo de obter acesso não autorizados ao serviço. Como estas identidades falsas são apresentadas como legítimas, elas violam os princípios da confidencialidade, autenticidade e irretratabilidade, necessários para garantir a segurança da rede. Tais identidades falsas podem ser roubadas (Id_ρ) ou fabricadas (Id_φ), sendo que $Id_\rho \cup Id_\varphi$ compreende o conjunto das identidades forjadas pelos atacantes. Note que $Id_\rho \in Id$ pois foram roubadas de nós legítimos. Por outro lado, $Id_\varphi \notin Id$, visto que é uma identidade fabricada e não está associada a nenhum nó único. Contudo, ambas as identidades estão sob a custódia de um atacante, e o seu uso deve ser impedido pelo SA^2CI.

Neste trabalho, um nó Sybil pode apresentar duas formas de comportamento, o primeiro é chamado de *churn* e o segundo de múltiplas identidades. No comportamento *churn* um atacante deve possuir apenas uma identidade falsa, e ele pode entrar e sair muitas vezes da rede, de forma dinâmica, imprevisível, e arbitrária. Nesta conduta, um nó atacante busca promover o esgotamento dos recursos de um nó que realiza a autenticação, e também praticar uma força bruta na tentativa de forjar uma identidade legítima. Já no comportamento de múltiplas identidades, um dado nó atacante deve possuir diver-

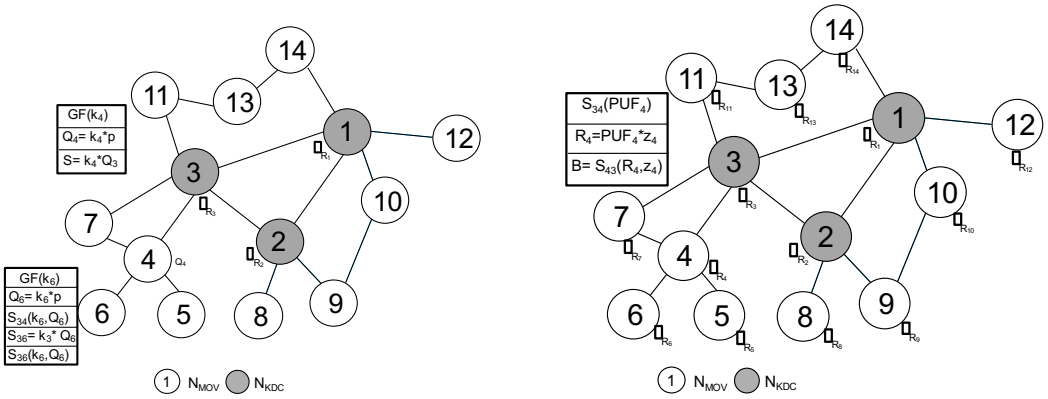
sas identidades, solicitando associação à rede. Um atacante com este comportamento se passa por mais de um nó legítimo apresentando identidades aos nós autenticadores com uma frequência baixa, e tentando reproduzir um dispositivo legítimo. Logo, ambos os comportamentos visam ludibriar um nó que executa a autenticação na rede.

3.2. Mecanismo de controle de associações para IoT - (SA^2CI)

Na fase de inicialização os nós sem restrição de recursos formam um KDC, de forma autônoma e distribuída. Para isso, eles estabelecem um curva elíptica e compartilham as informações dessa curva entre si. As curvas elípticas requerem um baixo custo computacional para a distribuição de chaves e possibilitam a criação de canais seguros, sendo atrativas para as IoTs [Guicheng and Zhen 2013]. Na fase de configuração, os nós KDC emitem as chaves públicas e privadas. Essas chaves são geradas para todos os nós, N_{KDC} e N_{MOV}. Em seguida, os N_{KDC} estabelecem chaves de sessão (simétrica) com os seus respectivos nós N_{MOV} a partir das chaves públicas e privadas e de pontos da curva elíptica. As chaves de sessão são utilizadas para a transferência de dados entre um nó N_{MOV} e o seu respectivo N_{KDC}. Ainda nesta fase, um nó computa sua PUF, envia ao N_{KDC}, que gera um recibo associado a esta PUF e à identidade do nó. Na fase de gerência, são realizadas as associações de nós ao serviço de disseminação, considerando a PUF e o recibo de um dado nó, bem como o seu comportamento.

3.2.1. Inicialização

A inicialização da rede tem como objetivo estabelecer uma curva elíptica entre os nós $\mathcal{N}_{KDC}$. Nesta fase, apenas os nós $\mathcal{N}_{KDC}$ atuam, trocando informações e estabelecendo uma curva E por meio de um canal seguro. Este canal seguro é necessário apenas nesta etapa. A curva elíptica é expressa pela equação de Weierstrass, $y^2 = x^3 + Ax + B$, em que A e B são constantes [Pinol et al. 2015]. Geralmente, A, B, x e y são números reais $\mathbb{R}$, complexos $\mathbb{C}$, racionais $\mathbb{Q}$, ou um corpo finito K. A curva E é um conjunto sobre dois pontos primos p e q de um corpo finito GF. O grupo $E(GF)$ corresponde ao grupo de pontos pertencente à curva E com coordenadas em GF. Um dado ponto $G \in E(GF)$ é um ponto da curva E e a sua ordem é o menor inteiro positivo k tal que $k.p = \infty$. Além disso, a ordem do ponto G sempre divide a ordem do grupo $E(GF)$, obtendo assim um subgrupo base para um conjunto de coordenadas projetivas que dão origem à curva E.



(a) Inicialização - Distrib. dos Pares de Chave

(b) Configuração - Criação do recibo

Figura 1. Operações de Inicialização e configuração do mecanismo

É considerado um sistema de coordenadas projetivas jacobianas, que possibilita a representação dos pontos de uma curva E em um espaço projetivo P_k [Pinol et al. 2015], evitando assim multiplicações no grupo $E(GF)$. Isto permite gerar uma curva com menos recursos computacionais quando comparado com o plano de coordenadas homogêneas. No sistema de coordenadas jacobiana um ponto (x, y) é representado por $(X : Y : Z) = (\lambda_x^2, \lambda_y^3, \lambda)$ para todo $\lambda \neq 0$. O ponto ∞ é dado por $(\lambda^2 : \lambda^3 : 0)$. Neste sistema, as coordenadas X e Y possuem peso 2 e 3. Assim, a curva E gerada pelos nós $\mathcal{N}_{KDC}$ é alcançada a partir da seguinte equação $Y^2 = X^3 + AXZ^4 + BZ^6$.

Na inicialização, Figura 1(a), uma curva E é obtida por meio de um acordo entre os nós $\mathcal{N}_{KDC}$. Para isso, os nós trocam informações sobre os valores x, y, z, a, b. Um dado nó $\mathcal{N}_{KDC}$ gera esta curva por meio da equação $Y^2 = X^3 + AXZ^4 + BZ^6$ já definida. Em seguida, ele compartilha os valores da curva E, isto é, x, y, z, a, b para os demais nós $\mathcal{N}_{KDC}$. Com a curva E compartilhada entre os nós $\mathcal{N}_{KDC}$, o próximo passo consiste em configurar os pares de chaves de $\mathcal{N}_{KDC}$ e $\mathcal{N}_{MOV}$, o que ocorre na configuração.

3.2.2. Configuração

A configuração da rede visa estabelecer os pares de chaves entre os nós $\mathcal{N}_{MOV}$ e os seus respectivos $\mathcal{N}_{KDC}$. Ambos os tipos de nós, ($\mathcal{N}_{MOV}$ e $\mathcal{N}_{KDC}$), participam dessa fase. Um nó pode ser configurado de forma direta ou indireta. Na primeira, um nó $\mathcal{N}_{KDC}$ realiza a configuração de um novo nó, enquanto que na segunda forma um $\mathcal{N}_{MOV}$ já associado na rede atua como intermediário entre um novo nó e o seu respectivo $\mathcal{N}_{KDC}$.

Após a criação e compartilhamento da curva elíptica E, os nós N_{KDC} geram os pares de chaves para seus respectivos nós $\mathcal{N}_{MOV}$. Inicialmente, cada nó $N_k \in \mathcal{N}_{KDC}$ gera a sua chave privada através de um ponto $K_{N_x} \in GF(p)$. A chave pública deste nó (Q_{N_x}) é computada por meio da sua chave privada multiplicada pelo ponto $p \in GF$ $Q_{N_x} = K_{N_x} \times p$. Em seguida, os nós $\mathcal{N}_{KDC}$ realizam o mesmo processo de gerar um par de chaves para cada nó $\mathcal{N}_{MOV}$ que esteja próximo da sua área de cobertura.

Então, cada nó N_{KDC} estabelece um segredo com cada um dos nós $\mathcal{N}_{MOV}$. Para cada nó $N_x \in \mathcal{N}_{MOV}$, o nó KDC correspondente calcula o segredo $S_x = Q_{N_x} \times K_{kdc}$. Após a criação do segredo compartilhado, cada nó $\mathcal{N}_{MOV}$ envia o seu código PUF$_x$ ao seu respectivo nó KDC, comprovando a sua identidade. A PUF do um dado nó é um código único extraído do hardware deste nó que permite a sua identificação. Este código é computado através da diferença de ciclos de *clocks* do processador, ou até das imperfeições geradas no processo de fabricação de um componente do hardware. Assim, cada nó $N_x \in \mathcal{N}_{MOV}$ envia sua PUF$_x$ cifrada com o segredo (S_x) ao seu respectivo nó KDC.

Ao receber a PUF do nó $N_x \in \mathcal{N}_{MOV}$, o nó N_{KDC} gera o recibo R_{N_x} para o nó N_x, a partir da PUF$_x$ e um ponto $z \in E$, obtendo $R_{N_x} = \text{PUF}_x \times z$. Em seguida, o nó N_{KDC} cifra o recibo R_{N_x} com o segredo compartilhado S_x e o envia juntamente com o ponto z da curva ao nó N_x. Ao final deste processo, cada nó N_x recebe o seu R_{N_x}. Com esse recibo, um nó pode obter acesso à rede IoT e aos seus serviços. Por fim, os nós KDCs atualizam a sua lista de recibos gerados, de modo que todos os demais nós KDCs possuam os recibos dos legítimos, permitindo a sua verificação a partir de qualquer requisição.

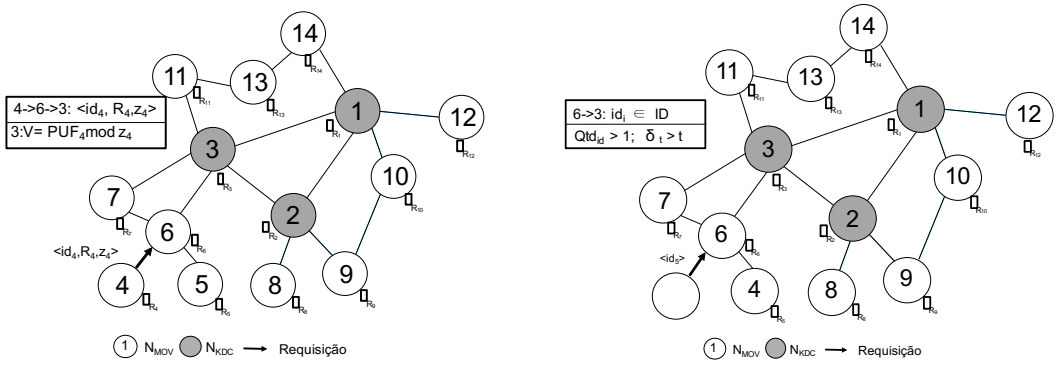
A Figura 1(b) ilustra o funcionamento da configuração dos nós $\mathcal{N}_{MOV}$ realizada pelos nós $\mathcal{N}_{KDC}$. No exemplo da Figura 1(a), N_3 gera um par de chaves privada K_4 e pública Q_4 para N_4. Com a chave pública de N_4, N_3 emite um segredo S_4 e compartilha

com N_4 para que eles possam trocar informações de forma segura. Este processo acontece para todo $N_i \in \mathcal{N}_{MOV}$. Em seguida, N_4 usa o segredo S_4 para cifrar a sua PUF_4 e enviá-la à N_3. Ao receber a PUF_4 de N_4, N_3 emite o recibo de identidade para esse nó (Figura 1(b)). Para gerar o recibo de N_4, o nó N_3 escolhe aleatoriamente um ponto de E no qual será multiplicado pela PUF_4, e obtém R_{N_4}. Logo após criar R_4, N_3 envia a garantia da identidade ao nó N_4, que pode utilizá-la para realizar uma requisição de acesso à rede, comprovando a legitimidade da sua identidade.

3.2.3. Gerência da disseminação

A fase de gerência da disseminação realiza o monitoramento das requisições de associações dos nós $\mathcal{N}_{MOV}$ e dos novos nós que desejam o acesso ao serviço de disseminação de dados. Uma associação à rede consiste de uma requisição de acesso, em que o nó solicitante envia a sua identidade e o seu recibo, no caso de reassociação, ou apenas o pedido de nova associação se for um novo nó.

Quando um nó que não foi configurado deseja acesso à rede, ele realiza uma nova associação. Numa nova associação, este nó (N_x) tem o seu comportamento avaliado. Para isso, tanto um $\mathcal{N}_{MOV}$ quanto o $\mathcal{N}_{KDC}$ monitoram o comportamento de N_x durante a sua associação à rede, verificando se o seu comportamento é ou não malicioso através das assinaturas maliciosas conhecidas. Tais assinaturas consistem, por exemplo, em entrar e sair várias vezes da rede mudando de identidade ou exibir múltiplas identidades. Uma nova associação enviada por N_x compreende de apenas um campo, a identidade $< Id_x >$ de N_x. Assim, caso não seja detectado um comportamento malicioso de N_x, este nó tem o seu acesso concedido, recebendo um recibo correspondente à identidade apresentada.



(a) Reassociação (b) Nova Associação

Figura 2. Monitoramento das associações da rede

A Figura 2 mostra o funcionamento desta etapa, em que os nós da rede autenticam as reassociações e as novas associações. Para uma reassociação, N_4, por exemplo, faz uma requisição ao nó autenticador, N_1, contendo uma tripla, a sua Id_4, o seu respectivo recibo R_{N_4}, e o ponto z_4 usado para gerar o seu recibo. O nó N_1 verifica se a identidade apresentada na requisição equivale ao recibo R_{N_4}. Esta verificação acontece através de $V = PUF_{N_4} \mod z_{N_4}$, que confirma a veracidade do recibo. Se o recibo apresentado na requisição for verdadeiro, N_1 concede o acesso a este nó, caso contrário o acesso não é autorizado. Já em uma nova associação, o nó N_5, por exemplo, não possui recibo nem seu comportamento é conhecido na rede. Diante disso, a sua conduta deve ser avaliada

pelo nó autenticador por meio de assinaturas maliciosas. Ao avaliar a conduta de N_5 (Figura 2), o nó N_1 pode detectar um comportamento malicioso desassociando este nó ou, caso contrário, ele será configurado na rede.

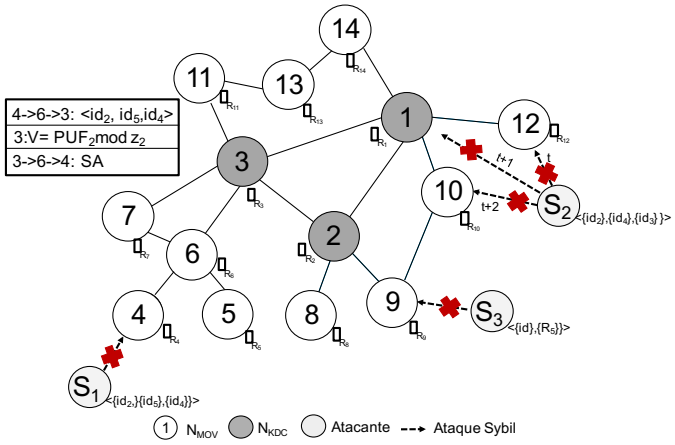


Figura 3. Detecção do SA

A detecção dos ataques Sybil, com comportamento de múltiplas identidades e *churn*, é ilustrada na Figura 3. Nela, o nó N_4 identifica um atacante com múltiplas identidades, S_1, no instante que ele exibe as identidades através da tripla $< id_2$,id_5,$id_4 >$ durante a associação à rede. Inicialmente, o nó N_4 averigua a conduta de S_1, verificando as assinaturas maliciosas conhecidas. Com a conduta legitima, o nó S_1 apresenta uma dada identidade falsa e inicia-se o processo de autenticação dessa identidade. O nó N_4 detecta a requisição maliciosa de S_1 neste momento, porque S_1 apresenta uma identidade que já possui um recibo associado e que não foi apresentado.

Um segundo atacante, S_2, apresenta o comportamento *churn* ao tentar o acesso à rede. Os nós N_{10} e N_{12} identificam a conduta de S_2 durante a avaliação da conduta, pois nos tempos t e $t+1$ o atacante S_2 realizou pedidos de associações num tempo menor do que a assinatura $\delta_{solicitacao} > t$. Caso S_2 apresente pedidos de associação num tempo maior do que assinatura $\delta_{solicitacao} \geq t$, a má conduta é identificada através da verificação da identidade e do seu respectivo recibo utilizando a função $Vreq$.

Um terceiro atacante, S_3, tenta o acesso ao serviço de disseminação por meio de uma identidade forjada id_5 associada com um recibo R_5 também forjado. O nó N_9 detecta o atacante de modo indireto. No pedido de acesso à rede, N_9 envia a tupla $< id_5, R_5 >$ ao nó KDC N_2 cifrada com o seu segredo $S_{9,2}$, para que ele verifique a veracidade de R_5. O nó N_2 executa a função $Vreq$, obtendo o resultado da legitimidade do recibo. Como esse recibo não condiz com a identidade forjada, o nó S_3 é desassociado da rede.

4. Avaliação

A eficácia do mecanismo SA^2CI sobre o controle de associações à disseminação de conteúdo da IoT diante de ataques Sybil foi avaliada e comparada com o mecanismo LSD (*Lightweight Sybil Attack Detection*) [Abbas et al. 2013]. Ambos os mecanismos foram implementados no simulador NS3 – versão 3.24, onde foram utilizados a biblioteca crypto++ – versão 5.63 para a implementação das curvas elipticas, e a classe *energy model* para o consumo energético, assim como a função PUF. Também foram implementados

os ataques Sybil com identidades fabricadas e roubadas, apresentando o comportamento churn e de múltiplas identidades.

Os mecanismos foram avaliados num cenário residencial como o descrito em [Le et al. 2012], onde assume-se que o modo de disseminação não impõe restrições na capacidade de detecção dos mecanismos, e na forma como os atacantes obtém as identidades. No cenário, os objetos, por exemplo, refrigerador, televisão, e *smartphone*, representam os nós da rede. Estes nós disseminam um fluxo de dados a um destino que os encaminham para aplicações na Internet, de modo a prover serviços em tempo real. Um fluxo de dados consiste no envio de uma mensagem de 127 bytes, isto é, *payload* e cabeçalhos que seguem o padrão das redes 6LoWPAN. A escolha dos nós origem e de um nó destino dos fluxos acontece de forma aleatória e os nós origem não podem ser o destino. Os nós atacantes visam o acesso à disseminação através do uso de identidades forjadas, com comportamentos churn e exibindo múltiplas identidades. As Tabelas 1 e 2 resumem os parâmetros usados na configuração da rede IoT e no ataque Sybil. Nesta avaliação, os valores obtidos nas duas primeiras métricas, $MTBF$ e $MTTR$ definidas a seguir, consistem de apenas uma rodada de simulação com o objetivo de observar o funcionamento dos mecanismos. Já nas demais métricas foram consideradas trinta repetições da simulação com um intervalo de confiança de 95%.

Tabela 1. Parâmetros dos Nós

Parâmetro	Valores
Área	$25m$x$25m$
Qtd. de nós	20,40,60
Raio de alcance	$10m$ (KDC) e $100m$ (MOV)
Mod. de mobilidade	Random Waypoint
Vel. dos nós	0,2m/s a 2m/s
Tempo de simulação	600 s

Tabela 2. Parâmetros da Rede e do Ataque

Parâmetro	Valores
Tipo do pacote	UDP
Protoc. de roteamento	RPL
Período transiente	40s
Protoc. de enlace	IEEE 802.15.4
Nós Sybil	10%
Quant. Múltiplas IDs	1 a 5

As métricas utilizadas na avaliação da eficácia do mecanismo SA^2CI são: ***Tempo médio entre falhas*** ($MTBF$), ***Tempo médio de reparo*** ($MTTR$), ***Taxa de Detecção*** (T_{det}), ***Acurácia*** (A_c), e ***Falsos Positivos*** (T_{fp}). Já ***Consumo Energético*** (CE) mede o custo da energia gasta pelo mecanismo. O $MTBF$ identifica o intervalo de tempo entre falhas do mecanismo na detecção de um ataque (Eq. 1). O $MTTR$ calcula o intervalo de tempo gasto para se recuperar uma falha causada por um atacante (Eq. 2). A T_{det} contabiliza os ataques Sybil identificados corretamente do total de ataques (Eq. 3). A métrica A_c indica a precisão da detecção, e corresponde ao total de detecção do ataque Sybil, det_{ni} mais o total de identificação correta dos nós legítimos da rede, det_{na}, dividido pela quantidade de requisições feitas à rede, T_{req}, (Eq. 4). A T_{fp} determina a quantidade de vezes que os mecanismos identificaram um ataque Sybil inexistente (Eq. 5).

$$MTBF = \frac{T}{d} \quad (1)\ MTTR = \frac{\sum_{i=1}^{d} T_i}{d} \quad (2)\ T_{det} = \frac{\sum det_{ni}}{Tatq} \quad (3)\ A_c = \frac{\sum det_{ni} + \sum det_{na}}{Treq} \ (4) \quad T_{fp} = \frac{\sum det_{ni}}{Treq_l} \ (5)$$

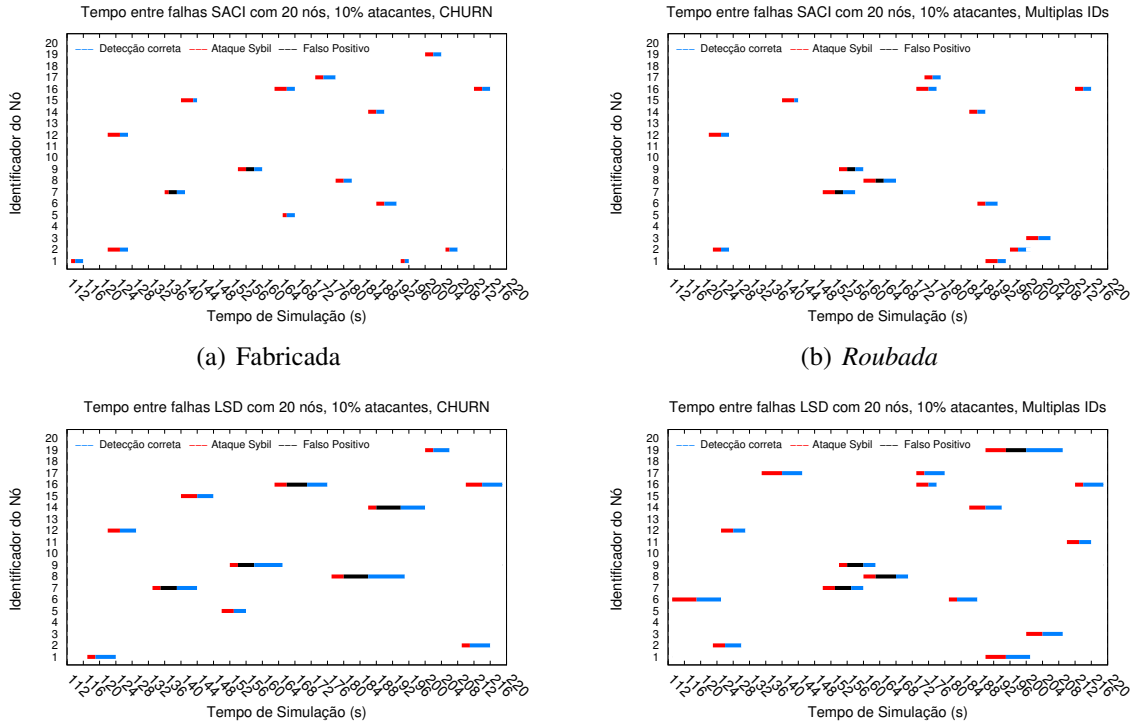
A métrica CE determina o consumo energético dos nós da rede com o mecanismo SA^2CI, sendo obtida através do somatório da energia inicial dos nós da rede, TE_i, subtraído do total restante de energia destes nós, TE_r, (Eq. 6).

$$CE = \sum(TE_i - TE_r) \quad (6)$$

4.1. Resultados

Os resultados inicialmente apresentados nesta seção mostram o funcionamento do SA^2CI e do LSD ao analisar seu desempenho em termos do tempo médio entre falhas e de recuperação diante de falhas num ambiente controlado onde foi realizada apenas uma rodada de simulação. As

falhas na detecção do ataque Sybil no SA^2CI e no LSD são mostradas nos gráficos da Figura 4. Estes gráficos descrevem um intervalo de tempo do funcionamento dos mecanismos em relação ao tempo total de simulação, onde os atacantes empregam identidades legitimas da rede. A faixa de tempo vermelha significa o período de tempo entre o inicio de um ataque e a sua percepção pelo mecanismo, já a preta representa o tempo de uma detecção errada até a percepção do diagnóstico equivocado, enquanto a azul equivale o intervalo de tempo entre uma detecção correta e a eliminação da falha. Nos gráficos, percebe-se que o SA^2CI detecta uma presença maliciosa, num curto espaço de tempo, em média 2 segundos. Já o LSD necessita de um tempo maior para identificar o ataque, uma vez que ele requer a mensuração do RSS pelos nós vizinhos durante a autenticação. Além disso, o LSD uma vez feita uma identificação errada, ele exige mais tempo para corrigir tal erro (faixa preta), para remover o ataque. Isto deve-se ao atraso na aferição dos valores de RSS e esses valores não serem precisos. No SA^2CI, os falsos positivos ocorreram em nós $\mathcal{N}_{MOV}$, que possuiam limitação de energia ou desconectaram-se devido à mobilidade.



(a) Fabricada (b) *Roubada*

(c) Fabricada (d) *Roubada*

Figura 4. Falhas do SA^2CI e LSD diante de ataques Sybil

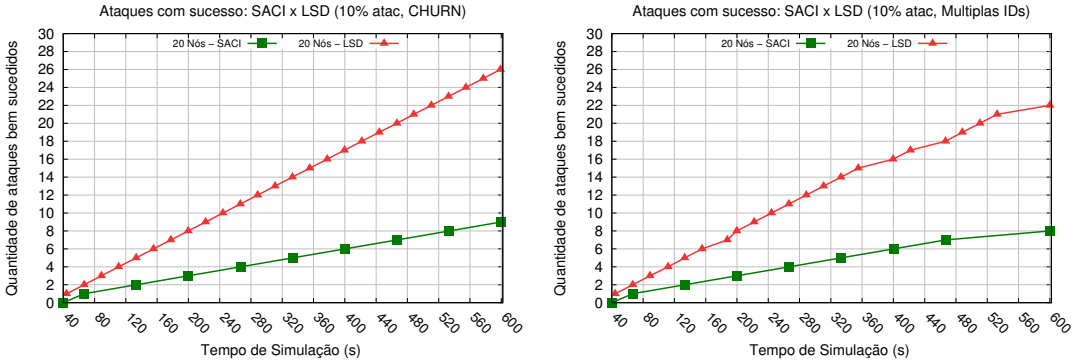
A Tabela 3 exibe o *MTBF* e o *MTTR* dos dois mecanismos na qual a frequência de falhas e o tempo de recuperação delas são menores no SA^2CI do que no LSD independente da quantidade de nós.

Tabela 3. Tempo entre falhas e recuperação do SA^2CI e LSD

Qtd. Nós	MTBF – SA^2CI	MTTR – SA^2CI	MTBF – LSD	MTTR – LSD
20	66.7 s	2.3 s	22.2 s	5.566 s
40	69.3 s	2.5 s	23.4 s	5.89 s
60	70.1 s	2.5 s	24.2 s	5.93 s

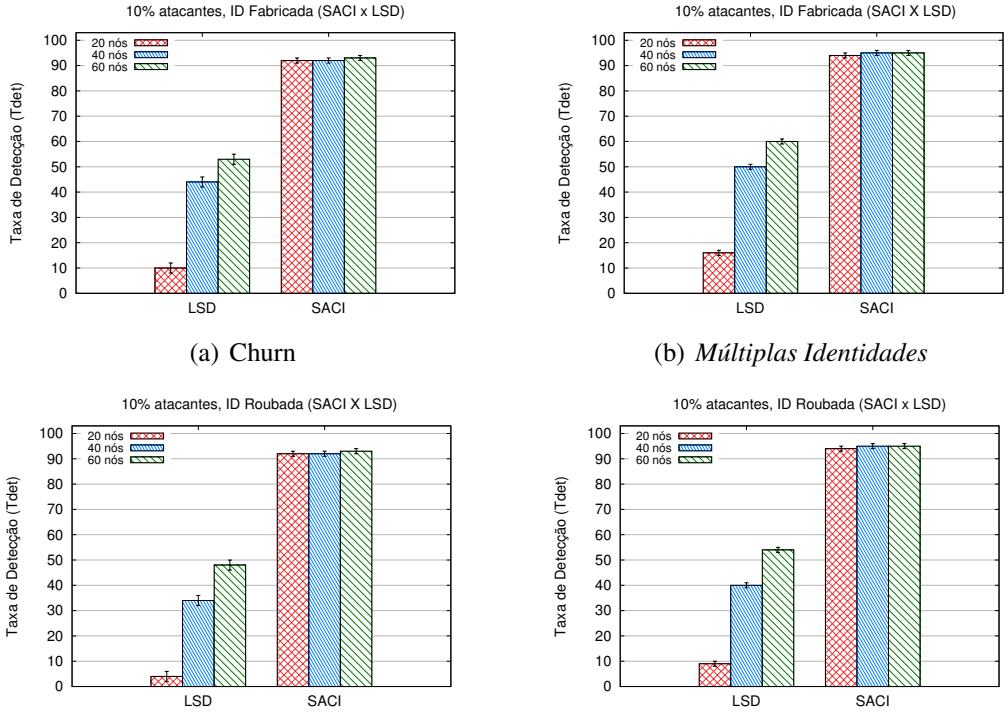
A resistência ao comportamento dos ataques Sybil é mostrada nos gráficos da Figura 5, onde a quantidade de ataques com sucesso é contabilizada ao longo do tempo. Percebe-se que

independente do comportamento, os ataques Sybil são mais difíceis de serem detectados pelo LSD. O gráfico 5(a) mostra que no SA^2CI apenas nove ataques obtiveram sucesso. Já no LSD vinte e nove investidas feitas pelos atacantes foram bem sucedidas. Os atacantes Sybil empregando múltiplas identidades foram menos efetivos do que aqueles com o comportamento churn, como mostrado no Gráfico 5(b). Percebe-se que o SA^2CI identifica melhor as ameaças independente do comportamento do atacante, enquanto o LSD possui uma maior oscilação.



(a) *Churn* (b) Múltiplas Identidades

Figura 5. Efetividade do Ataque Sybil no SA^2CI e no LSD



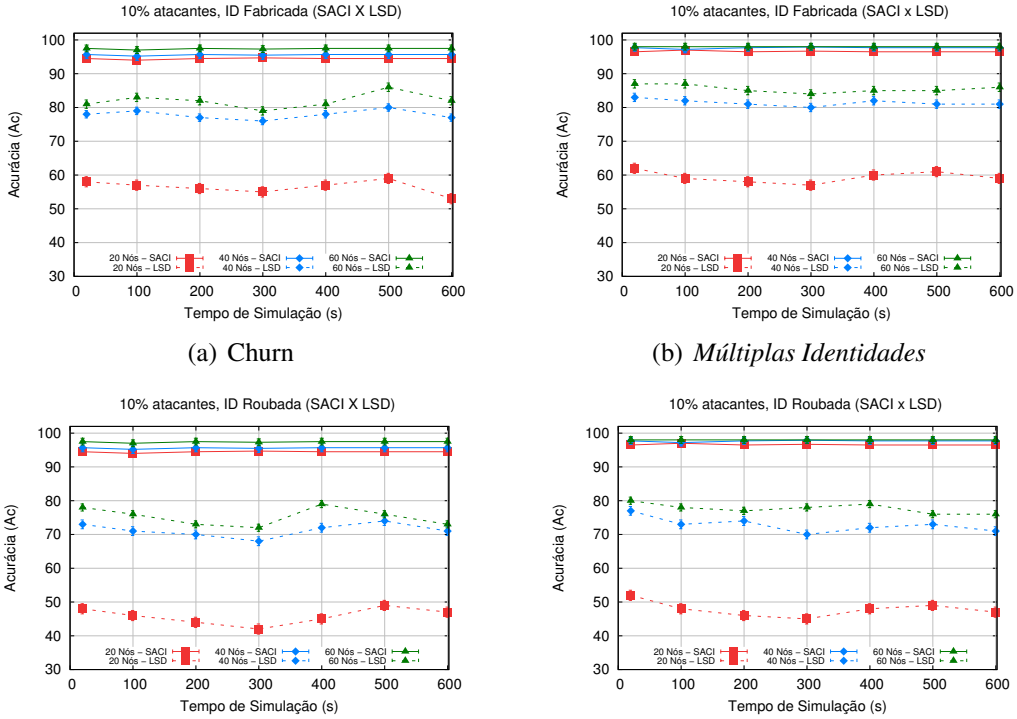
(a) Churn (b) *Múltiplas Identidades*

(c) Churn (d) *Múltiplas Identidades*

Figura 6. Comparativo entre taxa de detecção

Este bloco de resultados tem como objetivo a discursão dos resultados relacionados a métricas de segurança, isto é, T_{det}, A_c, e T_{fp}, aonde são considerados trinta repetições de simulação para cada métrica. Os gráficos da Figura 6 mostram a T_{det} do SA^2CI e do LSD diante de ataques Sybil com os comportamentos churn e múltiplas identidades. As características da rede empregadas pelo LSD limitam a detecção de atacantes, principalmente num ambiente móvel,

pois ele requer análises constantes do RSS do nó autenticado. O comportamento, o tipo de identidade, e a densidade da rede influenciam na detecção do LSD, onde este mecanismo apresentou o pior desempenho, 5% com 20 nós e identidades roubadas, possuindo uma baixa T_{det} pois ele desconsidera a irretratabilidade das identidade nós. Já a T_{det} do SA^2CI alcançou 92% por que o SA^2CI emprega recibos de identidade, e isto garante a irretratabilidade dos nós. Antes de cada associação de um nó à rede, a sua conduta é verificada, evitando que um atacante apresente uma identidade falsa independente do comportamento churn ou de múltiplas identidades. O uso de recibos possibilita a identificação de forma única de um nó, isto é, caso um atacante Sybil fabrique ou roube uma identidade, o recibo desta identidade não será o mesmo.



(a) Churn (b) *Múltiplas Identidades*

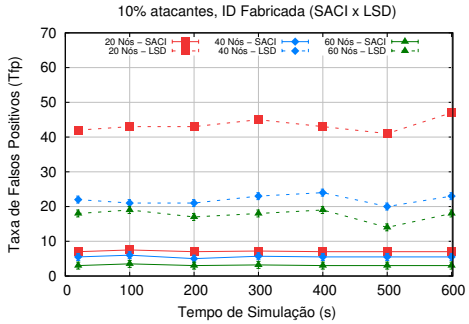
(c) Churn (d) *Múltiplas Identidades*

Figura 7. Comparativo entre as acurácias

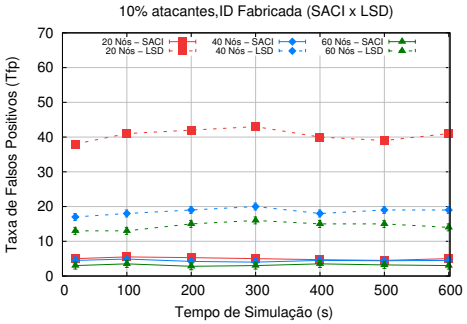
Os gráficos da Figura 7 mostram a A_c de ambos os mecanismos. O comportamento constante do SA^2CI ocorre em razão da técnica empregada necessitar apenas do recibo de identidades para realizar a detecção. Esta forma de detecção é diferente da técnica usada pelo LSD, que requer uma constante mensuração do RSS dos nós. Além disso, o comportamento de uma atacante durante uma associação interfere muito pouco na acurácia do SA^2CI, onde ela reduz 3% em cenários mais esparsos quando os ataques empregam o churn no lugar de múltiplas identidades. O LSD possui uma A_c próxima dos 90% no cenário mais denso, 60 nós. Contudo, em cenários menos densos a precisão tende a diminuir, visto que o LSD desconsidera a irretratabilidade das identidades dos nós e possui menos vizinhos para auxíliar na detecção.

Os gráficos da Figura 8 mostram as T_{fp} do SA^2CI e do LSD. No SA^2CI esta taxa varia entre 4% à 7% para todos os cenários, enquanto que no LSD esta variação é de 20% à 60%. Ambos os mecanismos possuem uma maior valor de T_{fp} quando um atacante utiliza o comportamento churn, devido a sua maneira de associações e desassociações à rede, e isto prejudica a distinção entre um nó atacante e um nó legítimo. O SA^2CI possui uma taxa de falsos positivos menor, sofrendo também menos oscilações do que o LSD, principalmente em cenários esparsos. Por outro lado, os ataques Sybil com o comportamento churn aumentam a taxa da falsos positivos de

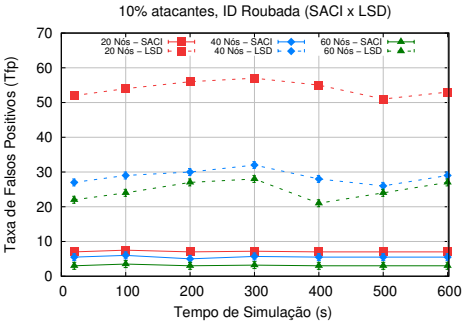
ambos os mecanismos em relação ao ataques de multiplas identidades.



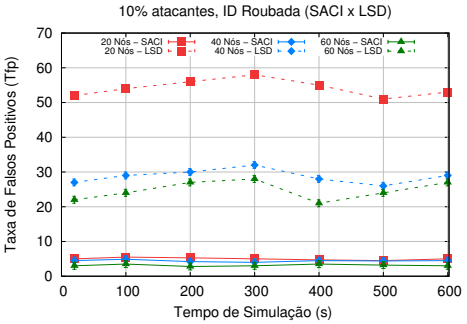
(a) Churn



(b) *Múltiplas Identidades*



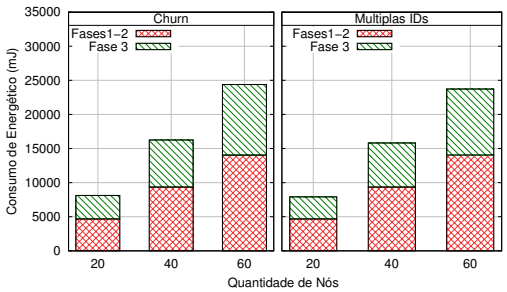
(c) Churn



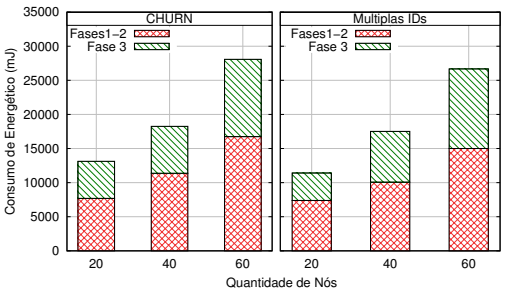
(d) *Múltiplas Identidades*

Figura 8. Comparativo entre os falsos positivos

Os gráficos da Figura 9 mostram o CE do SA^2CI na detecção de ataques Sybil, e compreende o custo da sua inicialização e configuração (fases 1 e 2), e a gerência das associações (fase 3). Percebe-se que os nós com comportamento churn demandam um gasto energético maior, pois o SA^2CI lida com as constantes associações e desassociações de um atacante. Em ambos os tipos de identidades empregadas pelo atacante Sybil, o SA^2CI obteve o mesmo consumo, e isto deve-se porque ele precisa apenas do recibo de identidade para determinar uma associação maliciosa.



(a) Fabricada



(b) *Roubada*

Figura 9. Consumo energético do SA^2CI diante de ataques Sybil

5. Conclusão

Este trabalho apresentou um mecanismo SA^2CI para o controle de associações resistente à ataques Sybil para a disseminação de conteúdo em redes IoT. Ele detecta ataques Sybil de forma escalar, distribuída e adaptativa. Além disso, O SA^2CI leva em conta a heterogeneidade computacional e a irretratabilidade das identidades dos dispositivos. O SA^2CI foi avaliado em um ambiente

doméstico e a sua eficácia foi comparada ao do LSD. As simulações demonstraram que ele previne associações de atacantes com comportamentos maliciosos e de identidades manipuladas em razão das técnicas empregadas. Futuros trabalhos consiste em avaliá-lo em domínios maiores.

Referências

Abbas, S., Merabti, M., Llewellyn-Jones, D., and Kifayat, K. Lightweight sybil attack detection in manets. *Systems Journal, 2013*, páginas:236–248.

Chatzigiannakis, I., Pyrgelis, A., Spirakis, P. G., and Stamatiou, Y. C. Elliptic curve based zero knowledge proofs and their applicability on resource constrained devices. In *8th International Conference on Mobile Adhoc and Sensor Systems (MASS), 2011*, páginas 715–720.

Choden Konigsmark, S., Hwang, L. K., Chen, D., and Wong, M. D. System-of-pufs: Multilevel security for embedded systems. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2014*, páginas 1–10.

Evangelista, D., dos Santos, A., and Nogueira, M. Avaliação das técnicas de detecção do ataque sybil na disseminação de conteúdo da internet das coisas. In *XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBseg), 2015*.

Guicheng, S. and Zhen, Y.. Application of elliptic curve cryptography in node authentication of internet of things. In *Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, 2013*.

Le, V.-D., Scholten, H., and Havinga, P. Unified routing for data dissemination in smart city networks. In *3rd International Conference on the Internet of Things, 2012*, páginas 175-182.

Mahalle, P. N., Anggorojati, B., Prasad, N. R., and Prasad, R. . Identity establishment and capability based access control (IECAC) scheme for internet of things. In *15th International Symposium on Wireless Personal Multimedia Communications (WPMC), 2012*, páginas 187–191.

Park, S., Aslam, B., Turgut, D., and Zou, C. C. Defense against sybil attack in the initial deployment stage of vehicular ad hoc network based on roadside unit support. *Security and Communication Networks, 2013*, páginas 523–538.

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. Sensing as a service model for smart cities supported by internet of things. *Transactions on Emerging Telecommunications Technologies*, páginas 81–93.

Pinol, O. P., Raza, S., Eriksson, J., and Voigt, T. Bsd-based elliptic curve cryptography for the open internet of things. In *7th International Conference New Technologies, Mobility and Security (NTMS), 2015*, páginas 1-5.

Quercia, D. and Hailes, S. Sybil attacks against mobile users: friends and foes to the rescue. In *INFOCOM, 2010*, páginas 1–5.

Vamsi, P. R. and Kant, K. A lightweight sybil attack detection framework for wireless sensor networks. In *Seventh International Conference on Contemporary Computing (IC3), 2014* páginas 387–393.

Wallgren, L., Raza, S., and Voigt, T. Routing attacks and countermeasures in the RPL-based Internet of things. *International Journal of Distributed Sensor Networks, 2013*.

Wu, C.-C., Chang, C.-C., and Lin, I.-C. . New sealed-bid electronic auction with fairness, security and efficiency. *Journal of Computer Science and Technology, 2008*, páginas 253–264.

Zheng, J. X. and Potkonjak, M. A digital puf-based ip protection architecture for network embedded systems. In *Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems*, páginas 255–256.

Privacy-Preserving Techniques in Smart Metering: An Overview

Pedro Barbosa[1], **Lucas Freitas**[1],
Andrey Brito[1], **Leandro Silva**[1]

[1]Computer and Systems Department, Federal University of Campina Grande, Brazil

pedroyossis@copin.ufcg.edu.br, jose.freitas@ccc.ufcg.edu.br,

andrey@dsc.ufcg.edu.br, leandro@ufcg.edu.br

Abstract. *Smart energy meters grant to power providers the capability to perform many advanced services, and several countries already started to deploy them. However, data analysis can raise privacy issues by inferring daily activities and appliance usages of consumers. Hence, there is a crucial need to deal with the problem of consumers' privacy in smart metering. Several approaches offer ways to provide privacy and preserve some of the benefits. In this paper, we list, experiment and evaluate five of these approaches based on three orthogonal technologies: noise addition, rechargeable batteries, and homomorphic encryption. We evaluate them based on the main needed attributes, such as complexity and accuracy, and conclude that there are many tradeoffs to be dealt with.*

1. Introduction

Governments and power providers across the world have recognized that the traditional grid, which has not significantly changed in the last decades, must be replaced by more efficient, flexible and intelligent energy-distribution networks, called Smart Grids. These are digitally monitored, self-healing energy systems that deliver electricity from generation sources, including distributed renewable sources, to points of consumption. They optimize power delivery and enable end-user energy management, minimizing power disruptions and transporting only the required amount of power. The result is a lower cost to the power provider and to the consumer, and a more reliable power generation, transmission and distribution.

Smart meters are devices that measure electricity consumption in real time and transmit this data to remote servers. They may represent a turning point in the energy industry and foster the development of new services and improvement of existing ones. In a typical smart metering architecture, the analysis of the collected data can help power providers to learn how to better manage the areas within their networks. Thus, helping to understand the business benefits of investing in Smart Grids. Figure 1 presents a smart metering system architecture.

Despite the benefits, smart meters raise concerns about the privacy of consumers. Electricity data may contain private sensitive information, such as which appliances are being used, if the house is empty, when people take a shower or shut down the television. Using advanced power signature analysis tools, such as the Non-Intrusive Appliance Load Monitoring (NIALM), it is possible to find out private information about

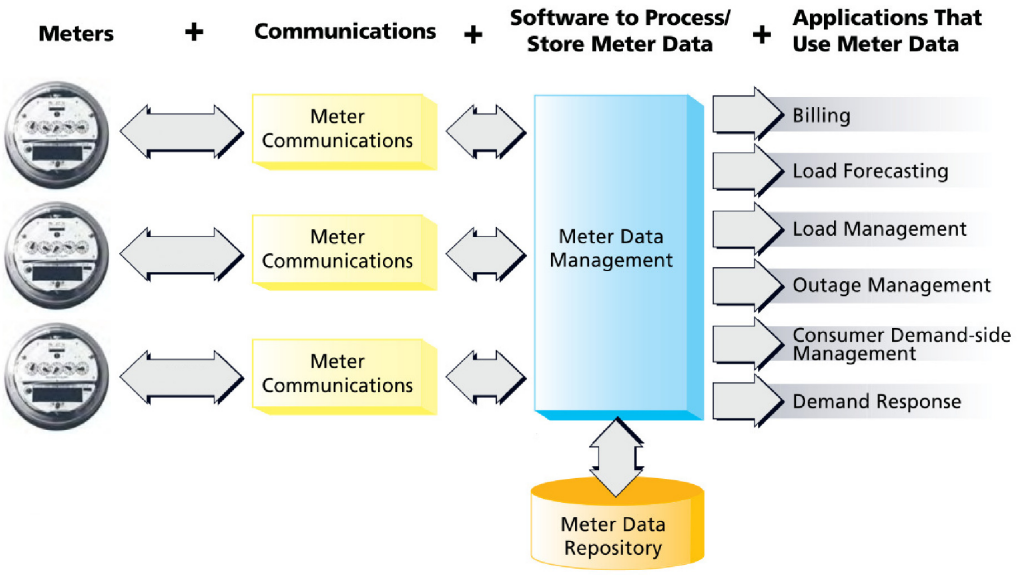


Figure 1. Smart metering system architecture and data usage applications [Waters 2006].

consumer's lifestyle. [Batra et al. 2014] designed some methodologies to identify the use of appliances from load profiles. If the load monitoring algorithm is running remotely, the consumers may not know that their behaviors are being monitored.

The information about a lifestyle of a person may be interesting to many businesses that can take advantages of it and offer their goods and services. In order to protect consumers from misuse of their data, and to prevent smart meters to become a new type of Big Brother, rules and regulations are needed [Boccuzzi 2010]. Sadly, such laws may take decades to be created and applied. While this still needs to happen, smart meters are already operational.

Taking these problems into account, there are a handful of solutions to protect privacy in smart metering. Many of them provide levels of privacy and still preserve several of the benefits of the smart metering deployment. Regarding some performance aspects such as computational complexity, some solutions are fast and lightweight, while others require heavy processing due to complex computations.

The solutions can be classified based on their approach to solve the privacy issues. There are techniques based on homomorphic encryption, the techniques that use rechargeable batteries, and the ones that make use of noise addition. In this paper, we review these techniques, evaluate and discuss their performance, according to a few relevant performance aspects. Besides computational complexity, we also evaluate other aspects, such as scalability, meters' independence, cost, environmental impact, and accuracy.

The rest of the paper is organized as follows. We summarize the problem statement in Section 2 and present the privacy techniques in Section 3. In Section 4 we present the conducted experiments and in Section 5 we discuss the obtained results. Finally, we conclude the paper in Section 6.

2. Problem Statement

There are many benefits of using a smart metering system. Some of them are: identify technical and non-technical losses (*e.g.*, power thefts), monitor energy quality scores, and optimize load forecasting. However, information of power consumption may be used for reasons not related to energy management, making the collection and distribution of such information dangerous to the privacy of consumers. The privacy issues are some of the main reasons why smart meters were still not deployed in many countries [Koehle 2012].

With the considerable amount of privacy-preserving techniques that can be adopted to ensure privacy in smart metering, the need to better understand and compare these solutions arises. To enable power providers to process smart meter measurements, while preventing them to access private data, cryptographic tools like homomorphic encryption may be used. With these approaches, before sending its measurement, each smart meter runs a cryptographic routine. The utility provider receives encrypted measurements, but can still perform useful computations and output the correct aggregate values, like the total consumption of a consumer during a billing period or the total consumption in the region in an instant of time. Thus, some benefits of using smart metering are still provided, while the consumer's privacy is maintained.

Techniques based on the usage of rechargeable batteries consist in using a battery between the smart metter and home appliances. Therefore, the disclosed information is the battery load profile, but not the daily activities and appliance usages of consumers.

Through noise addition techniques, individual measurements are masked by adding random numbers. This masking happens in a way that does not affect the outcome of the aggregating operations but hides the individual measurements.

In this paper, we address the following business problem: which privacy preserving solution in smart metering is more suited for a specific scenario? It is important for the development of smart grids that all involved parties understand the solutions, knowing what are their strengths and weaknesses. Some solutions might be simple for only one end (*e.g.*, in the smart meters), but not for the other (*e.g.*, power providers). Having an outline of solutions, descriptions, advantages and disadvantages is useful to assist the choice of techniques that are best fit for specific situations.

3. Privacy Techniques

Below we present some of the proposals for preserving privacy in smart metering. Since the main privacy issues are derived from individual data (like the consumption in a instant of time), these privacy preserving approaches tend to reveal aggregate data (like the total consumption through a billing period or the total consumption in a region) and hide individual data. The presented approaches follow a *Trust No One* philosophy, that is, the privacy techniques are applied in the smart meter, and trusted third parties are not required.

3.1. Noise Addition

Noise addition is a privacy preserving technique to mask the data. [Wang et al. 2012] propose an approach to mask the data adding random numbers from a GMM (Gaussian Mixture Models), whereas [Bohli et al. 2010] and [He et al. 2013] propose approaches to

mask the data using Gaussian noise. Noise addition is a promising and efficient technique, however, these mentioned aproaches do not have formal models to calculate the amount of noise that should be added to guarantee desired privacy and utility levels. In fact, [He et al. 2013] argue that for real world system design, a proper trade-off between privacy protection and accuracy should be considered.

[Barbosa et al. 2014] propose that for every measurement, the smart meter reads the consumption and adds a random number. Thus, after an aggregating operation (such as the calculation of the total consumption in a region or the total consumption of a consumer in the end of a billing period), the result will be:

$$\sum_{i=1}^{N} c_i \;\approx\; \sum_{i=1}^{N} (c_i \;+\; x_i)$$

where N is the total number of measurements, x_i is a random number generated from a probabilistic distribution and c_i is an individual consumption measurement.

The previous formalization can also be rewritten as follows:

$$\sum_{i=1}^{N} c_i \;=\; \sum_{i=1}^{N} (c_i \;+\; x_i) \;-\; e_o$$

where e_o is the obtained error by the addition of random numbers. Therefore, e_o is the sum of all added random values:

$$e_o \;=\; \sum_{i=1}^{N} x_i \;.$$

[Barbosa et al. 2015] developed many analytical models using probability theory for different distributions. Here we will consider the Laplace distribution. Let x_i be a random variable generated from this distribution. Its variance is $\sigma_x^2 \;=\; 2b^2$, where b is a scale parameter. Now, for a large N, the central limit theorem ensures that the obtained error for billing purpose follows a normal distribution with mean $\mu_{e_o} = 0$ and variance:

$$\sigma_{e_o}^2 \;=\; N^2(\sigma_x^2 \;/\; N) \;=\; N\sigma_x^2 \;= 2Nb^2 \;. \tag{1}$$

In other words, to have an obtained error between two accepted values (with high probability), we can use the following normal distribution:

$$e_o \;\sim\; N\left(0,\; 2Nb^2\right) \;.$$

As an example, Figure 2 shows a daily profile of a residential consumer.[1] There are 16 appliances in this consumption profile. However, the appliance with highest wattage and easier to identify is the laundry dryer.

[1]Combining appliance signatures we can generate arbitrary large populations and measurement frequency. Several databases of appliance signatures are available online (*e.g.*, Tracebase [Reinhardt et al. 2012]).

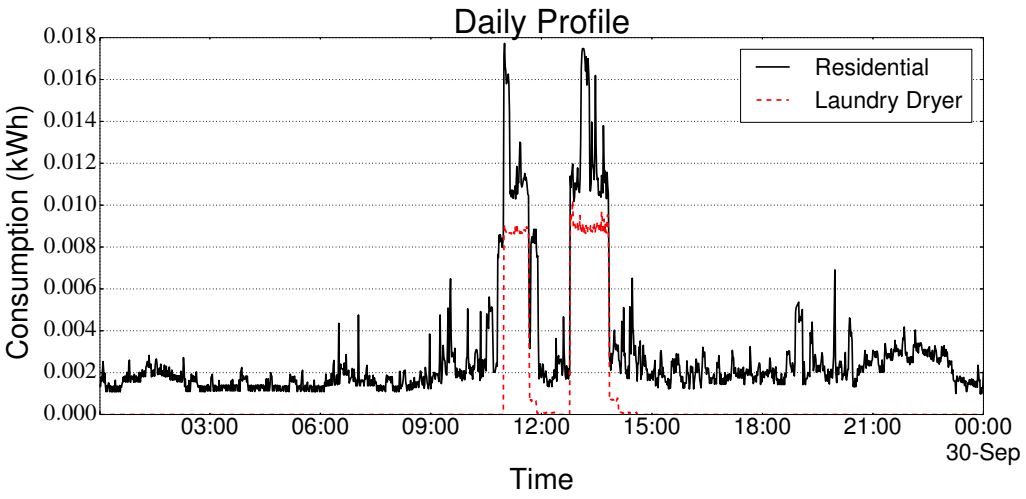


Figure 2. Residential (black solid) and Laundry Dryer (red dashed) daily profiles with measurements at each 1 minute.

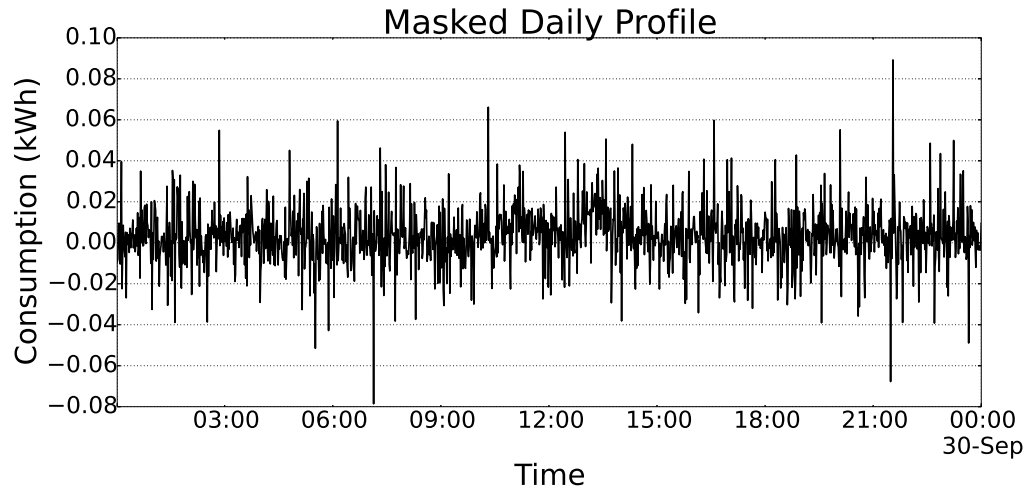


Figure 3. Residential masked daily profile with measurements at each 1 minute.

Assuming the billing period as one month, the total consumption of this consumer during the billing period (one month of 31 days) is 131.978 kWh. Considering a maximal allowed error of 5% for billing purpose, we have 6.5989 kWh. Thus, the variance for a high probability (*e.g.*, 0.98) of not exceeding this value is $\sigma^2_{e_o} = 8.04626$. Isolating the scale parameter b from Equation 1, we have (for measurements at each 1 minute, $N = 44,640$):

$$b = \sqrt{\sigma^2_{e_o} / (2N)} = 0.0094934 \, .$$

Figure 3 presents the daily masked profile using this Laplacian noise. Considering that at the end of the month the power provider sums the informed masked values by the consumer, it obtains a value of 133.7977 kWh. The real value is 131.978 kWh. The difference between these values is an error of 1.3788%, less than the maximum allowed error (5%). This error may be less if the consumer does not mask the measurements all the time.

Using this noise addition approach, it is possible to provide differential privacy [Dwork 2006] guarantees for appliance usages, making them indistinguishable in a consumption profile [Barbosa et al. 2016].

3.2. Rechargeable Batteries

Rechargeable batteries between appliances and smart meters can help to reduce the privacy issues as the appliance signatures are no longer legible [Backes and Meiser 2014, Kalogridis et al. 2010, McLaughlin et al. 2011, Zhao et al. 2014].

[McLaughlin et al. 2011] propose an approach called Non-Intrusive Load Leveling (NILL). The goal of a NILL system is to level the load profile to a constant *target load*, thus removing appliance signatures. When an appliance turns ON, it will exert a load beyond the target load. Thus, NILL will discharge the battery to partially supply the load created by the appliance, maintaining the target load. Similarly, if an appliance enters the OFF state, the load profile will decrease below the target load. These opportunities are used to charge the battery while restoring the target load.

The NILL system consists of two parts: a battery and a control system that regulates the battery's charge and discharge based on the present load and battery state. The controller attempts to maintain a steady state target load K_{SS}, but will go into one of two special states K_L or K_H if the battery needs to recover from a low or high state of charge.

The essence of NILL is described by the equation, $u(t) = d(t) + b(t)$, where $b(t)$ is the battery's rate of charge overtime, $d(t)$ is the actual load profile of the residence and $u(t)$ is the load under the influence of NILL as perceived by the smart meter and what is disclosed to the power provider. If $b(t) > 0$, the battery is charging, otherwise $b(t) < 0$ and the battery is discharging. Finally, $c(t)$ is used to represent the battery's state of charge, thus:

$$c(t) = \int_{t_0}^{t} b(t)dt + c(t_0) \ .$$

Therefore, $c(t)$ is monitored. If $c(t) < L$, where L is the lower safe limit on the battery's state of charge, then the battery needs to be recharged and the system goes to the K_L state. Similarly, if $c(t) > H$, the system goes to the K_H state and the battery is discharged.

3.3. Using a Modified ElGamal Encryption

The first homomorphic encryption solution that we consider is based on a modification in the ElGamal encryption, a cryptographic system that relies on the discrete logarithm problem. The ElGamal cryptosystem proceeds as follows:

- *Set up*: two large primes p and q such that $q \mid p - 1$ are chosen. Next, a generator g of the order q multiplicative subgroup G of $\mathbb{Z}_p^*$ is selected. Afterwards, g, p and q are published.
- *Key generation*: a secret key x is generated by setting its value as a random number $x \in_R \mathbb{Z}_q^*$. The corresponding public key is computed as $y = g^x$.
- *Encryption*: a message $m \in G$ is encrypted under public key y by taking a random number $r \in_R \mathbb{Z}_q^*$ and computing $c = g^r$ and $d = m \cdot y^r$. The ElGamal encryption of m under public key y, $E_y(m)$, is the tuple (c, d).

- *Decryption*: a ciphertext $E_y(m)$ is decrypted using the private key x by computing $m = d \cdot c^{-x}$.

Given messages m_1 and m_2, we can obtain an encryption of $m_1 \cdot m_2$ by computing:

$$\begin{aligned} E_y(m_1) \cdot E_y(m_2) &= (c_1 \cdot c_2, d_1 \cdot d_2) \\ &= (g^{r_1+r_2}, m_1 \cdot m_2 \cdot y^{r_1+r_2}) \\ &= E_y(m_1 \cdot m_2) \, . \end{aligned}$$

Hence, ElGamal is a multiplicative homomorphic cryptosystem.

To calculate the total consumption in a region, [Busom et al. 2015] propose a protocol which uses an additive ElGamal cryptosystem. Given $E_y(g^{m_1})$ and $E_y(g^{m_2})$, then, $E_y(g^{m_1}) \cdot E_y(g^{m_2}) = E_y(g^{m_1} \cdot g^{m_2}) = E_y(g^{m_1+m_2})$.

Initially, each smart meter possess the following values: a big prime number q and its generator g; a secret key x_i; a public key $y_i = g^{x_i}$. To encrypt the measurements, it is necessary a global public key $y = \prod_{i=1}^{N} y_i$.

Let m_i denote the measurement of a smart meter. To calculate the total consumption in the region, the following protocol is executed:

1. Each meter generates a random noise value $z_i \in \mathbb{Z}_q^*$ and computes a ciphertext as $C_i = E_y(g^{m_i+z_i}) = (c_i, d_i)$ which is sent to the aggregator (which can be the power provider).
2. The aggregator combines all the messages as $C = (\prod_{i=1}^{N} c_i, \prod_{i=1}^{N} d_i) = (c, d)$ and sends c to each meter.
3. Each meter computes $T_i = c^{x_i} \cdot g^{z_i}$ and sends the result to the aggregator. After that, each meter removes z_i from its memory.
4. Finally, the aggregator computes $D = d \cdot (\prod_{i=1}^{N} T_i)^{-1}$ and $log_g D = M = \sum_{i=1}^{N} m_i$, where M is the total consumption in the region.

Notice that, since M is a relatively small number, the discrete logarithm problem in step 4 can be solved in a short time. In step 2, the aggregator computes:

$$C = (\prod_{i=1}^{N} g^{r_i}, \prod_{i=1}^{N} g^{m_i+z_i} \cdot y^{r_i}) = (g^r, g^{M+z} \cdot y^r) = (c, d) \, ,$$

and in step 3, each meter computes:

$$T_i = c^{x_i} \cdot g^{z_i} = g^{r \cdot x_i} \cdot g^{z_i} = g^{x_i \cdot r} \cdot g^{z_i} = y_i^r \cdot g^{z_i} \, .$$

Therefore, the protocol works because in step 4 the aggregator computes:

$$D = d \cdot (\prod_{i=1}^{N} T_i)^{-1} = \frac{g^{M+z} \cdot y^r}{\prod_{i=1}^{N} (y_i^r \cdot g^{z_i})} = \frac{g^{M+z} \cdot y^r}{(\prod_{i=1}^{N} y_i^r) \cdot g^z} = \frac{g^{M+z} \cdot y^r}{g^z \cdot y^r} = g^M \, .$$

3.4. Using Paillier Encryption and Secret Sharing

A protocol based on Paillier encryption and *secret sharing* was proposed by [Garcia and Jacobs 2010]. The Paillier cryptosystem proceeds as follows:

- *Set up*: two large primes p and q are chosen, $n = p \cdot q$, and $\lambda = lcm(p-1, q-1)$. A random number $g \in_R \mathbb{Z}^*_{n^2}$ is chosen in such a way that $gcd(b, n) = 1$, where $b = L(g^\lambda \ mod \ n^2)$ and $L(u) = \frac{(u-1)}{n}$.
- *Key generation*: let μ be the modular multiplicative inverse of b modulo n, *i.e.*, $\mu = b^{-1} \ mod \ n$. Thus, the public key is $P_k = (n, g)$ and the private key is $S_k = (n, \lambda, \mu)$.
- *Encryption*: a message m is encrypted under public key P_k by taking a random number $r \in_R \mathbb{Z}^*_{n-1}$ and computing $E_{P_k}(m) = g^m \cdot r^n \ mod \ n^2$.
- *Decryption*: a ciphertext $c = E_{P_k}(m)$ is decrypted using the private key S_k by computing $m = L(c^\lambda \ mod \ n^2) \cdot \mu \ mod \ n$.

Given messages m_1 and m_2, we can obtain an encryption of $m_1 + m_2$ by computing:

$$\begin{aligned} E_{P_k}(m_1) \cdot E_{P_k}(m_2) &= g^{m_1} \cdot r_1^n \cdot g^{m_2} \cdot r_2^n \ mod \ n^2 \\ &= g^{m_1+m_2} \cdot (r_1 \cdot r_2)^n \ mod \ n^2 \\ &= E_{P_k}(m_1 + m_2) \ . \end{aligned}$$

Hence, Paillier is an additive homomorphic cryptosystem.

[Garcia and Jacobs 2010] propose that each smart meter possess a public key P_{ki} and a private key S_{ki}. Let m_i denote the measurement of the meter. To calculate the total consumption in the region, the following protocol is executed:

1. Each meter sends its public key to the aggregator.
2. The aggregator receives all public keys and shares them with all meters. Thus, each meter stays with its private key S_{ki} and all the public keys $\{P_{k1}, P_{k2}, \ldots, P_{kn}\}$.
3. Each meter calculates N secret sharings for its measurement m_i, in such a way that $m_i = \sum_{j=1}^{N} s_{ij}$. Then, the meter keeps s_{ii} privately and sends to the aggregator all the other secret sharings encrypted with the public keys of the other $N-1$ meters, *i.e.*, it sends $E_{P_{kj}}(s_{ij})$ for $j = 1, \ldots, i-1, i+1, \ldots, N$.
4. After receiving all the encrypted secret sharings, the aggregator multiplies the ones encrypted with the same public key. Due to the Pailier homomorphic property, for each meter i, it has $E_{P_{ki}}(m'_i) = \prod_{j \neq i}^{N} E_{P_{ki}}(s_{ji}) = E_{P_{ki}}(\sum_{j \neq i}^{N} s_{ji})$. Then, the aggregator sends $E_{P_{ki}}(m'_i)$ for each meter i.
5. Using its private key S_{ki}, each meter decrypts $E_{P_{ki}}(m'_i)$ and adds its s_{ii}, obtaining $\sum_{j=1}^{N} s_{ji}$. The meter then sends this value to the aggregator.
6. Finally, the aggregator can sum all the received values, obtaining the total consumption in the region $M = \sum_{i=1}^{N} \sum_{j=1}^{N} s_{ji}$.

In this approach, the total consumption in the region is computed and at the same time, neither the aggregator nor any other consumer has access to any real measurement from a consumer, for they can only access random shares. Since in step 6 the aggregator simply sum all the secret shares, the proof that the protocol works is straightforward.

3.5. Using a Modified Paillier Encryption

[Erkin and Tsudik 2012] propose a protocol based on a modification in the Paillier encryption. Starting, there is a single pair of Paillier keys (P_k and S_k) shared with all N meters. Let m_i denote the measurement of the meter. To calculate the total consumption in the region, the following protocol is executed:

1. Each meter generates $N-1$ random numbers, one for each other meter, and sends them using secure communication (*e.g.*, RSA encryption between meters). Thus, there is a total of $N \cdot (N-1)$ message exchanges in this step.
2. After receiving all the random numbers generated by the other meters, the meter computes $R_i = n + \sum_{j \neq i}^{N} r_{(i \to j)} - \sum_{j \neq i}^{N} r_{(j \to i)}$, where n is the Paillier modulo and $r_{(i \to j)}$ is the random number generated by the meter i for the meter j.
3. Following, the meter computes a hash h_t using the timestamp of the current measurement m_i. This hash must be coprime with the Paillier modulo n, *i.e.*, $gcd(h_t, n) = 1$. Since the timestamp is synchronized, the obtained hash is the same for all meters.
4. After computing R_i and h_t, the meter encrypts m_i using the following modified scheme of Paillier: $E_{P_k}(m_i) = g^{m_i} \cdot h_t^{R_i}$. Then, this encrypted measurement is disclosed to all other $N-1$ meters.
5. Finally, after receiving all the encrypted measurements of the other meters, the meter calculates $E_{P_k}(M) = \prod_{i=1}^{N} E_{P_k}(m_i) = E_{P_k}(\sum_{i=1}^{N} m_i)$. This is true due the homomorphic property.

Possessing $E_{P_k}(M)$, the meter can decrypt this value and then send the total consumption in the region M to the aggregator. This way, the total consumption is computed and privacy is preserved, for the meter does not have access to the other measurements in plaintext.

The protocol works because in step 5, the meter computes:

$$E_{P_k}(M) = g^{m_1+m_2+\cdots+m_N} \cdot h_t^{(\sum_{i=1}^{N} n)+(\sum_{i=1}^{N}\sum_{j\neq i}^{N} r_{(i\to j)})-(\sum_{i=1}^{N}\sum_{j\neq i}^{N} r_{(j\to i)})},$$

and

$$E_{P_k}(M) = g^{M} \cdot h_t^{N \cdot n}.$$

Considering that $r = h_t^N$, this configuration represents the original paillier cryptosystem.

4. Our Experiments

In order to analyze and compare solutions, we describe the performance aspects that are considered in our studies. It is important because, for example, solutions might excel at

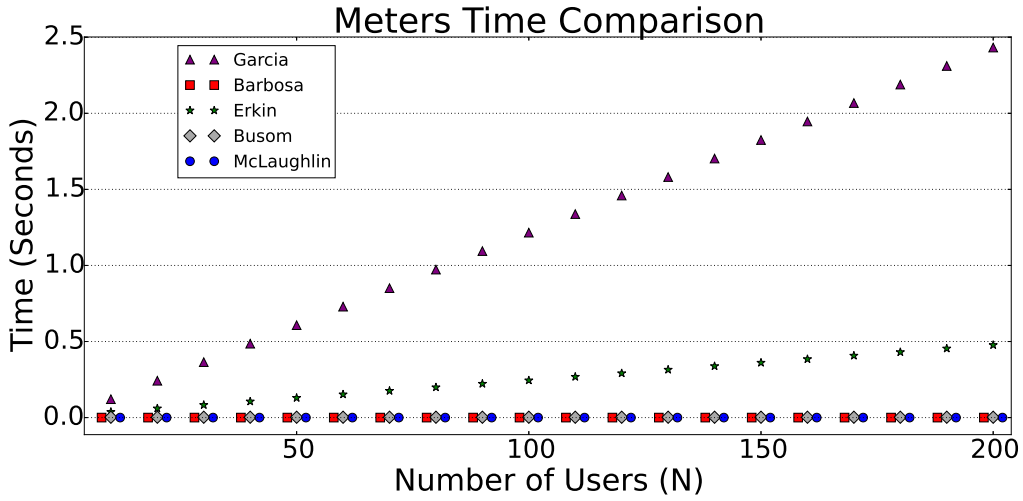


Figure 4. Processing time of smart meters in 5 different privacy preserving approaches.

computational complexity but their installation and usage has high costs and harms the environment. Thus, the need to consider different aspects of performance. In Section 5, we discuss the following aspects: computational complexity, scalability, meters' independence, cost, environmental impact and accuracy. Here, we present an experiment that aims to compare the response time (which is also related to computational complexity) of the approaches.

We implemented simulators[2] in *C* programming language. These simulators make use of a few functions in the *libgmp*[3], *libpaillier*[4] and *libcrypto*[5] libraries to implement algorithms that mimic the protocols described in this paper. The simulators were executed in a machine with 1.6 GHz Intel Core i5 processor, 6 GB of RAM memory and the Ubuntu 14.04 operating system.

In our simulations, using different configuration scenarios (number of meters, ranging from 1 to 200) to calculate the total consumption in the region, we measured the processing time of each meter (Figure 4) and the aggregator (Figure 5).

Each scenario was executed 10 times and the average values are being considered. This amount of repetitions were enough to get precise average values. Due to the very low obtained variations, the confidence intervals are being omitted here, except for the aggregation in the approach proposed by [Busom et al. 2015], which needs a trial and error mechanism to solve the discrete logarithm problem. The confidence intervals in these cases are of 95%.

From these measurements, we conclude that the noise addition approach and the one that uses rechargeable batteries presented very low response times, whereas the homomorphic encryption approaches presented considerable delays. It is also important to

[2]The source codes can be found at our GitHub repository (`https://git.lsd.ufcg.edu.br/pedroysb/privacy-performance-smart-metering/tree/master`).

[3]*libgmp*: `https://gmplib.org`

[4]*libpaillier*: `http://acsc.cs.utexas.edu/libpaillier`

[5]*libcrypto*: `https://www.openssl.org/docs/manmaster/crypto/crypto.html`

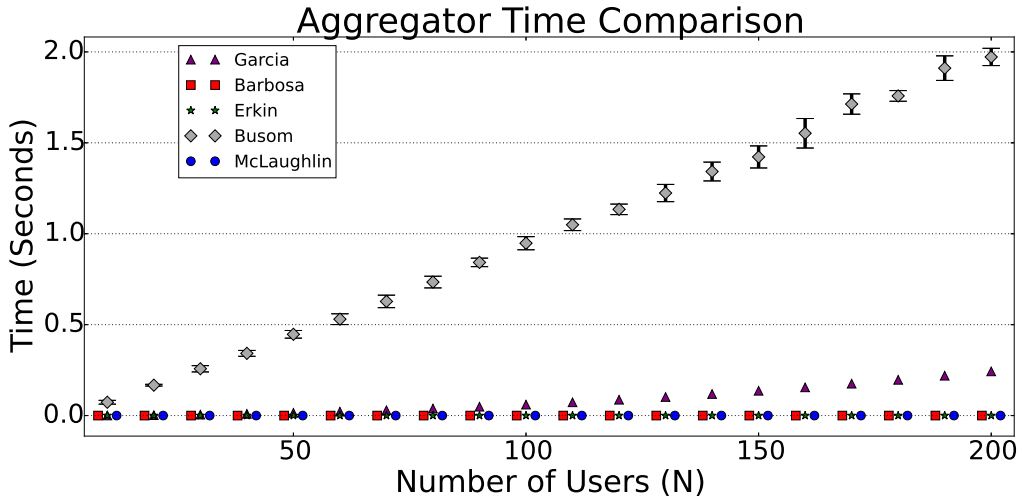


Figure 5. Processing time of the aggregator in 5 different privacy preserving approaches.

note that there are many message exchanges in the homomorphic encryption approaches, but we are not considering possible network delays in our experiments.

5. Discussions

We have presented several examples of privacy-preserving solutions to the computation of aggregate values. Now, we discuss the solutions regarding each of the performance aspects and taking into consideration the experimental results presented in Section 4. Table 1 presents a summary and comparison of approaches discussed.

Table 1. Comparison of noise addition (NA), rechargeable batteries (RB), and homomorphic encryption (HE) approaches.

	NA	RB	HE
Low complexity	✓	✓	
Scalability	✓	✓	
Meters' independence	✓	✓	
Low cost	✓		✓
Low environmental impact	✓		✓
Accuracy		✓	✓

5.1. Cost and Environmental Impact

As mentioned, usage of rechargeable batteries can help to diminish many privacy issues. Nevertheless, it is hard to ignore the environmental effects and the costs of using batteries. [McLaughlin et al. 2011] stipulate that a lead-acid battery of 50 Ah which operates at 12 V may cost approximately $100, and to achieve a typical residential nominal voltage of 120 V it is required 10 of such batteries (aprox. $1,000). The lifetime of each battery is approximately two years. Therefore, these solutions can not be considered low cost and cause a high environmental impact. Noise addition and homomorphic encryption approaches do not have these limitations.

5.2. Meters' Independence and Scalability

In order to exchange randomly generated numbers, keys and secret shares, the homomorphic encryption protocols require communication between meters and/or the power provider. Some solutions even require a large key distribution and certification. For this reason, in these approaches, the meters are not independent. This overhead can be the bottleneck of the smart metering system. Hence, actually optimizing communication at the meters is a hard task that has not been fully addressed in homomorphic encryption approaches. Additionally, if one meter fails in any message exchange, the aggregation becomes impossible because it requires computations using all distributed keys or secret shares. Therefore, these solutions may raise scalability issues when used in an area with a large number of meters. Noise addition and rechargeable batteries approaches do not have these limitations.

5.3. Accuracy

As mentioned, noise addition masks the data and introduces some error in an aggregation operation. This error is controlled and usually smaller than an acceptable value. Also, allowing the enabling or disabling of masking would make errors smaller in both dimensions: for billing, since a consumer would not mask all the time, and for load monitoring in a region, since not all consumers would mask in an instant of time. However, the noise addition privacy preserving approach is still not considered one hundred percent accurate. In the case of [Barbosa et al. 2015], the use of noise will introduce errors in the billing reports. These errors tend to be cancelled over time, but there is a probability that a higher value can occur. Rechargeable batteries and homomorphic encryption approaches do not have this limitation.

5.4. Computational Complexity

Low complexity is desired mainly because most of the deployed smart meters are low-cost microcontrollers and with limited computational resources. Through the analysis of the solutions' running time growth order, we have the complexities presented in Table 2. We consider that, regarding computational complexity, noise addition and rechargeable batteries stand in relation to the homomorphic encryption approaches.

Although estimation through asymptotic complexity is a good way to estimate computational complexity, we claim that experimental analysis is essential to present concrete results when comparing different proposals. As explored in Section 4, the experiments we conducted have shown that the noise addition and the rechargeable batteries approaches have considerably better performance regarding processing time.

6. Conclusions

Through the collection of energy consumption data, smart meters foster the development of new utilites and services. However, this same data can bring privacy issues for consumers, thus the need to study and develop privacy protection solutions. In this paper, we reviewed five solutions, one based on the use of noise addition, other based on the use of rechargeable batteries and three others based on homomorphic encryption schemes. We evaluated these approaches considering the main needed performance aspects and conclude that each solution has its advantages and disadvantages. As an example, by using rechargeable batteries the desired low levels of computational complexity is achieved, but on the other hand, they have a high cost and cause damage to the environment.

Table 2. Complexity analysis for different privacy preserving approaches in smart metering.

	NA		RB		MEE		PESS		MPE	
Operation	SM	AG	SM	AG	SM	AG	SM	AG	SM	AG
Encryption	-	-	-	-	$O(1)$	-	$O(N)$	-	$O(1)$	-
Decryption	-	-	-	-	-	$O(M)$	$O(1)$	-	$O(1)$	-
Transmission	$O(1)$	$O(N)$	$O(1)$	$O(N)$	$O(1)$	$O(N)$	$O(N)$	$O(N^2)$	$O(N)$	-
Sum	$O(1)$	$O(N)$	$O(1)$	$O(N)$	-	-	$O(1)$	$O(N)$	$O(N)$	-
Product	-	-	-	-	-	$O(N)$	-	$O(N^2)$	$O(N)$	-

Legend:
- **NA**: Noise Addition [Barbosa et al. 2015];
- **RB**: Rechargeable Batteries [McLaughlin et al. 2011];
- **MEE**: Modified ElGamal Encryption [Busom et al. 2015];
- **PESS**: Paillier Encryption and Secret Sharing [Garcia and Jacobs 2010];
- **MPE**: Modified Paillier Encryption [Erkin and Tsudik 2012];
- **SM**: Smart Meter;
- **AG**: Aggregator;
- **N**: Number of consumption measurements;
- **M**: Total (aggregate) consumption value.

Acknowledgement

This work was partially funded by the EU-BRA SecureCloud project (MCTI/RNP 3rd Coordinated Call) and by the Conselho Nacional de Desenvolvimento Científico e Tecnólogico (CNPq).

References

Backes, M. and Meiser, S. (2014). Differentially private smart metering with battery recharging. In *Data Privacy Management and Autonomous Spontaneous Security*, pages 194–212.

Barbosa, P., Brito, A., and Almeida, H. (2015). Defending against load monitoring in smart metering data through noise addition. In *Proc. of the 30th Annual ACM Symposium on Applied Computing (SAC)*, pages 2218–2224, Salamanca, Spain.

Barbosa, P., Brito, A., and Almeida, H. (2016). A technique to provide differential privacy for appliance usage in smart metering. *Information Sciences*, pages 355–367.

Barbosa, P., Brito, A., Almeida, H., and Clauß, S. (2014). Lightweight privacy for smart metering data by adding noise. In *Proc. of the 29th Annual ACM Symposium on Applied Computing (SAC)*, pages 531–538, Gyeongju, South Korea.

Batra, N., Kelly, J., Parson, O., Dutta, H., Knottenbelt, W., Rogers, A., Singh, A., and Srivastava, M. (2014). Nilmtk: An open source toolkit for non-intrusive load monitoring. In *5th International Conf. on Future Energy Systems (ACM e-Energy)*, Cambridge, UK.

Boccuzzi, C. (2010). Smart grid e o big brother energético. *Metering International América Latina*, 3:82–83.

Bohli, J., Sorge, C., and Ugus, O. (2010). A privacy model for smart metering. In *Proc. IEEE International Conf. Communications Workshops (ICC)*, pages 1–5, Cape Town, South Africa.

Busom, N., Petrlic, R., Sebé, F., Sorge, C., and Valls, M. (2015). Efficient smart metering based on homomorphic encryption. *Computer Communications*, pages 95–101.

Dwork, C. (2006). Differential privacy. In *Proc. of the 33rd International Colloquium on Automata, Languages and Programming, part II (ICALP)*, pages 1–12, Venice, Italy.

Erkin, Z. and Tsudik, G. (2012). Private computation of spatial and temporal power consumption with smart meters. In *Proc. of the 10th Int. Conf. on Applied Cryptography and Network Security (ACNS)*, pages 561–577.

Garcia, F. D. and Jacobs, B. (2010). Privacy-friendly energy-metering via homomorphic encryption. In *Security and Trust Management*, volume 6710, pages 226–238.

He, X., Zhang, X., and Kuo, C. C. J. (2013). A distortion-based approach to privacy-preserving metering in smart grids. *IEEE Access*, 1:67–78.

Kalogridis, G., Efthymiou, C., Denic, S. Z., Lewis, T. A., and Cepeda, R. (2010). Privacy for smart meters: towards undetectable appliance load signatures. In *IEEE 1st International Conf. on Smart Grid Communications (SmartGridComm)*, pages 232–237, Gaithersburg, USA.

Koehle, O. (2012). *Just say no to big brother's smart meters. The latest in bio-hazard technology*. ARC Reproductions.

McLaughlin, S., McDaniel, P., and Aiello, W. (2011). Protecting consumer privacy from electric load monitoring. In *Proc. of the 18th ACM Conf. on Computer and Communications Security (CCS)*, pages 87–98, Illinois, USA.

Reinhardt, A., Baumann, P., Burgstahler, D., Hollick, M., Chonov, H., Werner, M., and Steinmetz, R. (2012). On the accuracy of appliance identification based on distributed load metering data. In *Proc. of the 2nd IFIP Conf. on Sust. Internet and ICT for Sustainability*, pages 1–9.

Wang, S., Cui, L., Que, J., Choi, D.-H., Jiang, X., and Xie, L. (2012). A randomized response model for privacy preserving smart metering. *IEEE Trans. on Smart Grid*, 3:1317–1324.

Waters, G. (2006). Conquering advanced metering cost and risk. *Electric Energy T&D Magazine*, 10:22–25.

Zhao, J., Jung, T., Wang, Y., and Li, X. (2014). Achieving differential privacy of data disclosure in the smart grid. In *Proc. of IEEE INFOCOM*, pages 504–512.

AdC: um Mecanismo de Controle de Acesso para o Ciclo de Vida das Coisas Inteligentes

Antonio L. Maia Neto[1], Artur Luis Fernandes[1], Italo Cunha[1]
Michele Nogueira[2], Ivan Oliveira Nunes[1], Leonardo Cotta[1]
Nicolas Gentille[3], Antonio A. F. Loureiro[1], Diego F. Aranha[4]
Harsh Kupwade Patil[5], Leonardo B. Oliveira[1]

[1] UFMG [2]UFPR [3]LG Electronics [4]Unicamp [5]LGE Mobile Research
{lemosmaia,arturluis,cunha,ivanolive,leonardo.cotta,loureiro,leob}@dcc.ufmg.br
michele@inf.ufpr.br dfaranha@ic.unicamp.br {nicolas.gentille,harsh.patil}@lge.com

Abstract. *In this paper, we present AdC, a suite of protocols to incorporate authentication and access control for devices in the Internet of Things. AdC relies on cryptographic primitives from the state of the art to enforce an IoT-adequate access control scheme. AdC makes it possible to add new devices to Domestic domains in a wireless and authenticated way. AdC also allows devices from different domains to establish a trust relationship, making possible their inter-operation. To validate our solution, we have developed an AdC prototype for Android smartphones and evaluated the most resource-consuming cryptographic schemes over more constrained devices.*

Resumo. *Nesse artigo, apresentamos Autenticação das Coisas (AdC), um conjunto de protocolos para incorporar autenticação e controle de acesso para dispositivos na Internet das Coisas (IdC). O AdC emprega primitivas criptográficas do estado da arte para implementar um esquema de controle de acesso adequado à IdC. O AdC possibilita que novos dispositivos sejam adicionados à domínios Domésticos de maneira sem fio e autenticada e, além disso, permite que dispositivos de diferentes domínios estabeleçam uma relação de confiança, possibilitando a interoperação deles. Para validar nossa solução, nós desenvolvemos um protótipo do AdC para smartphones Android e avaliamos a execução dos esquemas criptográficos computacionalmente mais caros em dispositivos mais restritos.*

1. Introdução

A Internet das Coisas (IdC) [Ashton 2009] já é parte de nossas vidas. A ideia de um ambiente inteligente formado por elementos computacionais heterogêneos interconectados se tornou realidade. A IdC combina objetos físicos, sensores e atuadores, gerando sistemas ciber-físicos (*e.g.,* cidades, redes e casas inteligentes). Algumas projeções sugerem que, em breve, estaremos cercados por bilhões de dispositivos de IdC.[1]

A autenticação é primordial para a segurança [Stinson 2002]. Os mecanismos de autenticação permitem a distinção entre dados legítimos e forjados, assim como a determinação da origem de uma mensagem [Stinson 2002]. Para IdC, em particular, a autenticação permite controle de acesso, previne ataques *sybil*, e mitiga ataques de negação de serviço. Isto faz da autenticação uma propriedade chave em segurança para IdC [Stinson 2002].

Mesmo que a segurança em IdC tenha recebido atenção da comunidade científica (*e.g.,* [Markmann et al. 2015, Wangham et al. 2013]), as abordagens existentes não

[1] `http://www.gartner.com/newsroom/id/2636073`

atendem às necessidades de autenticação em IdC. Primeiramente, porque os esquemas tradicionais baseados em infraestrutura de chaves públicas (*Public Key Infrastructure* – PKI) e certificados causam sobrecarga significativa de CPU, memória, armazenamento, comunicação e gerenciamento, o que inviabiliza sua utilização em dispositivos de IdC [Stinson 2002]. Em segundo lugar, IdC geralmente requer que dispositivos de um certo domínio sejam capazes de interoperar, de forma segura, com dispositivos que pertençam a diferentes domínios. Acontece que grande parte dos esquemas de autenticação projetados para dispositivos com restrições de recursos (*e.g.,* [Perrig et al. 2001, Zhu et al. 2003, Liu et al. 2005, Oliveira and Dahab 2006, Silva et al. 2013]) assumem que os dispositivos pertencem a um domínio único e, portanto, não podem ser diretamente aplicados a IdC. Por fim, não há, no estado da arte, mecanismo que contemple autenticação durante todo o ciclo de vida de um dispositivo de IdC (*i.e.,* desde a manufatura até o descarte). Assim, há a necessidade de novas soluções, destinadas exclusivamente a atender os requisitos de autenticação para IdC.

Objetivo. Neste artigo, objetivamos projetar, desenvolver e avaliar um esquema de autenticação e controle de acesso para todo o ciclo de vida dos dispositivos de IdC. Nossa solução, Autenticação das Coisas (AdC), é composta por uma família de protocolos criptográficos que provê autenticação e controle de acesso a cada um dos estágios do ciclo de vida de um dispositivo, a saber: *Manufatura*, *Aquisição*, *Implantação*, *Operação* e *Descarte*.

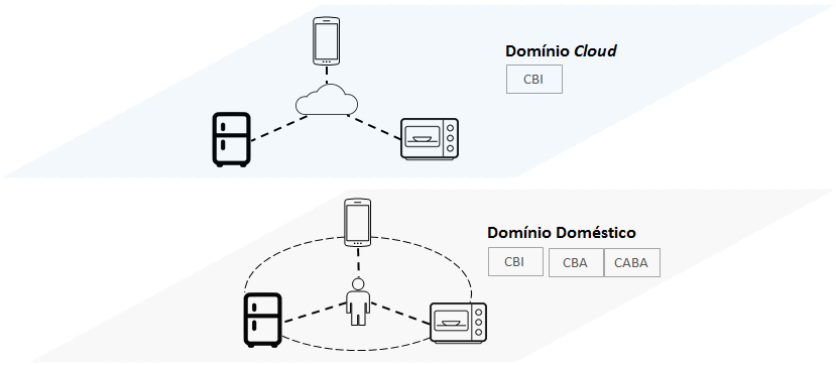


Figura 1. Arquitetura de dois domínios do AdC.

Contribuições. Este trabalho tem as seguintes contribuições:

1. AdC: uma família de protocolos para prover autenticação e controle de acesso durante todo o ciclo de vida de produtos de IdC.
2. Avaliação analítica e experimental do protocolo AdC, assim como a análise, via ferramenta de verificação automática, das propriedades de segurança do AdC. A avaliação experimental quantifica diversas métricas de desempenho em plataformas de hardware com restrição de recursos.
3. Os primeiros resultados experimentais de um esquema de assinatura baseada em atributos já publicados. Nossa implementação, baseada no trabalho de Maji, Prabhakaran e Rosulek [Maji et al. 2008], está disponível publicamente e é eficiente o suficiente para ser executada em dispositivos com restrição de recursos.

Organização. O restante deste artigo é estruturado da seguinte forma: a seção 2 apresenta o protocolo AdC e as premissas sob os quais foi projetado. A seção 3 descreve o desenvolvimento do AdC, com um foco especial à implementação e otimizações de primitivas

criptográficas. A seção 4 avalia a segurança do AdC e os requisitos computacionais necessários para sua implantação. Nós discutimos os trabalhos relacionados na seção 5 e concluímos na seção 6.

2. Autenticação das Coisas

Nesta seção apresentamos Autenticação das Coisas (AdC), uma família de protocolos de autenticação para todas as fases do ciclo de vida de um dispositivo de IdC. A seção 2.1 apresenta a visão geral do AdC, nas seções 2.2–2.7 descrevemos cada um dos protocolos que o compõe e, por fim, discutimos características complementares na seção 2.8.

Premissas. Para o projeto do protocolo assumimos que (i) todas as mensagens são enumeradas e contém o identificador do interlocutor; (ii) material criptográfico pode ser carregado de forma segura nos dispositivos durante o processo de manufatura (iii) PINs podem ser digitados de forma segura nos dispositivos dentro da casa do usuário; e (iv) primitivas criptográficas são ideais – *i.e.,* sem falhas – e podem ser utilizadas como caixas pretas.

2.1. Visão Geral

Problemas. Em IdC, questões elementares como a autenticação e controle de acesso ainda não estão solucionadas. A criptografia de chaves públicas tradicional é computacionalmente mais cara, não adequada para a maior parte dos dispositivos de IdC. Além disso, dada a heterogeneidade dos dispositivos de IdC e seus múltiplos domínios, esquemas de segurança desenvolvidos para outros cenários de redes sem fio e dispositivos com restrições de recursos (e.g., [Perrig et al. 2001, Zhu et al. 2003, Liu et al. 2005, Margi et al. 2009]) não podem ser diretamente aplicados a IdC. Da mesma forma, modelos tradicionais de controle de acesso não são aplicáveis em IdC [Yuan and Tong 2005]. Por exemplo, esquemas de controle de acesso mandatório, discricionário, e baseado em papéis são centrados no usuário e não consideram as relações dinâmicas entre o solicitante e o recurso. Além disso, em redes de larga escala como as de IdC, pode ser inviável gerir as listas de permissão de acesso aos recursos.

Objetivo. Nosso principal objetivo é projetar uma solução de autenticação e controle de acesso adequada para IdC. Neste contexto, consideramos eletrodomésticos inteligentes que interagem entre si, com os dispositivos pessoais dos usuários, com um servidor Doméstico e um servidor *Cloud*, que age como o fabricante dos dispositivos. Nossa solução deve atender às restrições de eficiência e segurança exigidas pelas aplicações de IdC.

Abordagem. Como usual, em nossa abordagem tratamos (i) a distribuição das chaves para a inicialização da segurança e (ii) o controle de acesso para gerenciar as permissões de execução das operações nos dispositivos de IdC. Entretanto, nossa implementação é inovadora: para a distribuição das chaves, aplicamos Criptografia Baseada em Identidade (CBI) [Shamir 1984, Sakai et al. 2000, Boneh and Franklin 2001], enquanto Criptografia Baseada em Atributos (CBA) [Goyal et al. 2006, Bethencourt et al. 2007] é aplicada para implementar criptograficamente o esquema de Controle de Acesso Baseado em Atributos (CABA) [Yuan and Tong 2005], que, por sua vez, gerencia o controle de acesso às operações nos dispositivos.

O principal desafio para adoção de CBI é o conhecido problema da custódia de chaves, onde o Gerador de Chaves Privadas (GCP) pode passar-se por qualquer usuário

no sistema. Para resolver este problema, concebemos uma arquitetura de dois domínios CBI isolados (figura 1): o domínio do fabricante, chamado *Cloud*, e o domínio local, a que chamamos Doméstico. Estes domínios definem as relações de confiança fabricante-dispositivo e (usuário-)(dispositivo-)dispositivo, respectivamente. Não há sobreposição dessas relações e, portanto, material criptográfico gerado no domínio *Cloud* não é válido no domínio Doméstico e vice-versa. Note-se que ainda existe custódia de chaves dentro de cada domínio isoladamente; entretanto, não é considerada mais um problema. A custódia do domínio *Cloud* não influencia as relações domésticas, já que as requisições originadas neste domínio são nulas no domínio Doméstico. Já a custódia de chaves no domínio Doméstico está inserida em um contexto onde o dispositivo que detém as chaves pertence aos mesmos usuários que o gerenciam, evidenciando uma forte relação de confiança.

Considerando-se o controle de acesso das operações nos dispositivos, o CABA simplifica as relações, substituindo permissões discricionárias por políticas baseadas nos atributos dos usuários, o que permite considerar características do recurso e informação de contexto. O CBA, por sua vez, fornece mecanismos criptográficos para adoção do CABA. Mais precisamente, um esquema de assinatura CBA (Assinatura Baseada em Atributos – ABA) pode ser usado para assegurar um subconjunto mínimo de atributos necessários para a execução de uma operação. Assim, as permissões de execução das operações dos dispositivos relacionam, através de políticas de alto nível CABA, os atributos CBA disponíveis no sistema.

Uma das principais características do AdC é que os dispositivos são implantados sem a necessidade de conexões cabeadas. Para tal, o AdC lança mão da proteção física das casas e de um dispositivo confiável (*e.g.,* o *smartphone* do administrador do domínio Doméstico) para servir de ponte durante a primeira comunicação entre o novo dispositivo e o servidor Doméstico e, assim, engendrar a segurança. Esta comunicação pode acontecer sob diferentes tecnologias de codificação e transmissão.

2.2. Protocolos Auxiliares

No AdC, o procedimento auxiliar *ChaveDeSessão* e os protocolos *AcordoDeChaves*, *Distribuição*, *Vinculação* e *Desvinculação* complementam os protocolos dos estágios principais (seções 2.3–2.7) com os seguintes objetivos: derivar chaves de sessão, acordar chaves par a par (*pairwise keys*), distribuir chaves privadas e vincular e desvincular um usuário de um dispositivo no domínio *Cloud*, respectivamente.

O procedimento ChaveDeSessão(k, i) recebe uma chave previamente compartilhada k e um contador i mantido atualizado entre os interlocutores para derivar chaves de sessão. Esta ideia é baseada no trabalho de [Perrig et al. 2001], onde uma função pseudoaleatória é usada para gerar novas chaves.

O protocolo AcordoDeChaves(A, B) é baseado no trabalho de [Sakai et al. 2000]. Ele permite que dois dispositivos (A e B) do mesmo domínio Doméstico troquem chaves par a par empregando emparelhamentos bilineares.

Já no protocolo Distribuição$(D, S, \mathcal{Z}, \mathrm{Y})$, o servidor S do domínio $\mathcal{Z}$ emite a chave privada $S^{\mathrm{Y}}_{D,\mathcal{Z}}$, relativa ao criptossistema Y, para o dispositivo D. A segurança desta operação é garantida pelo uso de uma chave de sessão derivada de uma chave par a par $k_{D,S}$ previamente compartilhada entre o dispositivo D e o servidor S.

Por fim, os protocolos Vinculação(D, U) e Desvinculação(D, U) vincula e des-

vincula, respectivamente, o dispositivo D e o usuário U no domínio *Cloud*.

Em cada um dos protocolos há uma troca de mensagens da forma desafio-resposta entre os interlocutores. Esta estratégia é adotada nos outros protocolos do AdC para assegurar validade (*freshness*) e evitar ataques de repetição.

2.3. *Manufatura*

Manufatura (figura 2) é executado durante a fabricação dos dispositivos. Neste estágio, o material criptográfico do domínio *Cloud* é carregado nos dispositivos. O servidor *Cloud* C gera a identidade e a chave privada do dispositivo D, $id_{D,\mathcal{C}}$ e $S^{\mathrm{I}}_{D,\mathcal{C}}$, respectivamente, no domínio *Cloud* (subscrito $\mathcal{C}$) e criptossistema CBI (sobrescrito I). Posteriormente, C gera uma chave par a par entre C e D ($k_{D,C}$), cria um contador c_D, e os envia ($\rightarrow$), juntamente com o restante do material criptográfico, para o dispositivo D (passo 3). Este envio é feito de forma segura via um canal físico (FIS), garantindo comunicação confidencial e autenticada. Ao final, C apaga $S^{\mathrm{I}}_{D,\mathcal{C}}$ e envia fisicamente ($\Rrightarrow$) D para ser comercializado por seu distribuidor T_D (passo 3).

MANUFATURA(D)
1. C : $id_{D,\mathcal{C}} \coloneqq serial\#\text{de } D,\ S^{\mathrm{I}}_{D,\mathcal{C}}$
2. $C \rightarrow D$: FIS($id_{D,\mathcal{C}}$,$S^{\mathrm{I}}_{D,\mathcal{C}}$,$k_{D,C}$,$\mathrm{c}_D$)
3. $C \Rrightarrow T_D$: D

Figura 2. Estágio *Manufatura*.

AQUISIÇÃO(D, U)
1. $U \rightarrow T_D$: TLS(\$)
2. $T_D \rightarrow C$: TLS(D | U)
3. $C \rightarrow U$: TLS(pin_D)
4. $T_D \Rrightarrow U$: D

Figura 3. Estágio *Aquisição*.

2.4. *Aquisição*

O protocolo *Aquisição* (figura 3) gere a comercialização do dispositivo D, através do distribuidor T_D, para o usuário U. Toda comunicação digital é feita de forma segura por TLS (passos 1, 2 e 3). O usuário U realiza o pedido e paga por D (passo 1). O distribuidor T_D, então, comunica a C a aquisição de D por U (passo 2). C, por sua vez, comunica a U o código de acesso pin_D (passo 3) e, por fim, T_D envia D para U (passo 4).

2.5. *Implantação*

O estágio de *Implantação* (figura 4) inicializa a segurança dos dispositivos em seus domínios Domésticos. Aqui, o usuário *root* (U_r) – provavelmente o dono da casa – e seu dispositivo pessoal D_{U_r} desempenham um papel fundamental. Mais precisamente, D_{U_r} atua como uma ponte de conexão confiável entre o dispositivo que é inserido no domínio Doméstico e o servidor Doméstico H.

Para inserir o novo dispositivo D, U_r digita o código de acesso pin_D em D (passo 1). Em seguida, D obtém de D_{U_r} (i) a identidade $id_{U_r,\mathcal{H}}$ de U_r no domínio Doméstico (subscrito $\mathcal{H}$); (ii) a chave pública $P^{\mathrm{I}}_{H,\mathcal{H}}$ de H, criptossistema CBI do domínio $\mathcal{H}$; (iii) e o contador do domínio $\mathrm{c}_{\mathbb{G}_{\mathrm{H}}}$ (passo 2) que vai prover *freshness* para *broadcasts* ($\Rightarrow$) originados de H. O dispositivo D, subsequentemente, gera, de forma aleatória, uma chave par a par $k_{D,H}$, efêmera entre D e H, a cifra usando a chave pública recém recebida $P^{\mathrm{I}}_{H,\mathcal{H}}$, e envia a cifra resultante ($\mathrm{CIP}(k_{D,H})_{P^{\mathrm{I}}_{H,\mathcal{H}}}$), juntamente com sua identidade $id_{D,\mathcal{H}}$ no domínio $\mathcal{H}$ e as operações suportadas ($info_D$) para D_{U_r} (passo 3). Então, U_r usa D_{U_r} para configurar os atributos $\mathbb{A}_D$ e predicados das operações $\mathbb{Y}_D$ de D (passo 4). Assumimos que a comunicação nesses passos iniciais é feita via canais seguros e adequados para o dispositivo D. Como este protocolo ocorre em um ambiente seguro (*e.g.,* casa do usuário), mecanismos de transmissão como digitar no teclado de um dispositivo ou escanear códigos

de barras poderiam ser cuidadosamente empregados para trocar informações de forma segura.

IMPLANTAÇÃO(D)

1.	$U_r \rightarrow D$:	FIS(pin_D)	8.	H :	$S^{\text{I}}_{D_U,\mathcal{H}}$, $S^{\text{A}}_{D_U,\mathcal{H}}$
2.	$D_{U_r} \rightarrow D$:	FIS($id_{U_r,\mathcal{H}}$,$P^{\text{I}}_{H,\mathcal{H}}$,$\text{c}_{\mathbb{G}_H}$)	9.	D :	DISTRIBUIÇÃO(D, H, $\mathcal{H}$, I)
3.	$D \rightarrow D_{U_r}$:	FIS($id_{D,\mathcal{H}}$,$info_D$, CIP($k_{D,H}$)$_{P^{\text{I}}_{H,\mathcal{H}}}$)	10.	D :	DISTRIBUIÇÃO(D, H, $\mathcal{H}$, A)
4.	$U_r \rightarrow D_{U_r}$:	FIS($\mathbb{A}_D$,$\mathbb{Y}_D$)	11.	$H \Rightarrow \mathbb{G}_H$:	$\mathbb{Y}_{\mathbb{G}_H}$, $\mathbb{A}_{\mathbb{G}_H}$,$\text{c}_{\mathbb{G}_H}$, $\text{SIG}_{S^{\text{I}}_{H,\mathcal{H}}}$
5.	$D_{U_r} \rightarrow H$:	$\text{n}_{D_{U_r}}$, imp_req	12.	D :	VINCULAÇÃO (D, U_r)
6.	$H \rightarrow D_{U_r}$:	n_H, MAC($\text{n}_{D_{U_r}}$)$_{k^i_{D_{U_r},H}}$	13.	$H \rightarrow D_{U_r}$:	imp_ack, MAC($\text{n}_{D_{U_r}}+1$)$_{k^i_{D_{U_r},H}}$
7.	$D_{U_r} \rightarrow H$:	imp, $id_{D,\mathcal{H}}$, $\mathbb{A}_D$, $\mathbb{Y}_D$, $info_D$, CIP($k_{D,H}$)$_{P^{\text{I}}_{H,\mathcal{H}}}$, SIG($\text{n}_H \mid \text{n}_{D_{U_r}}$)$_{S^{\text{I}}_{D_{U_r},\mathcal{H}}}$			

Figura 4. Estágio *Implantação*.

O protocolo prossegue como dispositivo root D_{U_r} requisitando (imp_req) a H, a implantação de D. (Note-se que neste ponto é iniciado um protocolo de desafio-resposta entre D_{U_r} e H, com o uso de números usados somente uma vez, *nonces*, $\text{n}_{D_{U_r}}$ e n_H, Código de Autenticação de Mensagens, MAC, e uma chave de sessão $k^i_{D_{U_r},H}$ entre D_{U_r} e H para assegurar *freshness* e evitar ataques de repetição.) Para esse fim, D_{U_r} encaminha a H o comando de implantação (imp), a informação recebida de D, as configurações inseridas por U_r e a cifra da chave $k_{D,H}$ recentemente gerada. Toda essa informação é enviada de forma autenticada, com a assinatura $\text{SIG}_{S^{\text{I}}_{D_{U_r},\mathcal{H}}}$ de D_{U_r}, utilizando sua chave privada $S^{\text{I}}_{D_{U_r},\mathcal{H}}$ de CBI no domínio $\mathcal{H}$, concluindo o passo 7. Aqui fica evidenciada a ponte de D_{U_r} para autenticação entre D e H. D_{U_r} "assina cegamente" [2] o conteúdo cifrado. Note-se que apesar da dependência de D_{U_r}, D não confia em D_{U_r} para trocar material criptográfico e, por isso, a chave $k_{D,H}$ é cifrada.

Seguindo o protocolo, o servidor Doméstico H deriva (passo 8) as chaves privadas de D no domínio $\mathcal{H}$, relativas a CBI $S^{\text{I}}_{D_U,\mathcal{H}}$ e CBA (sobrescrito A) $S^{\text{A}}_{D_U,\mathcal{H}}$, e as emite (passos 9 e 10). Esta emissão é protegida usando a chave recém compartilhada $k_{D,H}$. Para concluir a inclusão de D, H envia uma mensagem *broadcast* (passo 11), assinada com sua chave privada $S^{\text{I}}_{H,\mathcal{H}}$, para todos os dispositivos domésticos – incluindo o recém inserido D – contendo todo o conjunto de predicados ($\mathbb{Y}_{\mathbb{G}_H}$), atributos ($\mathbb{A}_{\mathbb{G}_H}$) e o contador atualizado do domínio ($\text{c}_{\mathbb{G}_H}$). Nesse ponto, D recebe acesso à Internet, *e.g.,* por meio do roteador Wi-Fi do domínio $\mathcal{H}$. O dispositivo D então se vincula ao usuário U_r no domínio *Cloud* (passo 12). Nós assumimos que D é um dispositivo doméstico, *e.g.,* uma geladeira ou TV, e é, portanto, vinculado a U_r. Se, por outro lado, D for o dispositivo pessoal de um usuário U, D é vinculado a U. O protocolo termina (passo 13) com o servidor H informando a D_{U_r} que D foi inserido com sucesso (imp_ack), concluindo o protocolo de desafio-resposta iniciado no passo 5.

2.6. *Operação*

Operação (figura 5) governa a operação cotidiana entre os dispositivos. Nesse protocolo, o usuário U requisita a execução da operação op no dispositivo B usando o dispositivo A (passo 1). O dispositivo A, então, encaminha a requisição (op_req) ao dispositivo B, que,

[2] Usamos aspas duplas pois assinatura cega se refere a uma construção criptográfica existente [Stinson 2002].

por sua vez, responde com o predicado da operação requisitada Υ_{op}, uma expressão booleana que relaciona atributos do sistema (passo 3). O dispositivo A prova que pode executar a operação com a assinatura da mensagem $\text{SIG}_{S^{\text{A}}_{A,\mathcal{H}}}$ usando sua chave privada CBA $S^{\text{A}}_{A,\mathcal{H}}$ do domínio $\mathcal{H}$. De fato, a chave utilizada para assinatura contém um subconjunto de atributos que deve satisfazer Υ_{op} (passo 4). Se B verifica com sucesso a assinatura, A tem permissões para executar op (passo 5). Note-se que *Operação* também faz uso de um protocolo desafio-resposta (passos 2 a 6), análogo àquele apresentado em *Implantação* (figura 4).

OPERAÇÃO(U, A, B, op)

1. U : usa A para executar op em B
2. $A \rightarrow B$: n_A, op_req
3. $B \rightarrow A$: $\text{n}_B, \Upsilon_{op}, \text{MAC}(\text{n}_A)_{k^i_{A,B}}$
4. $A \rightarrow B$: op, $\text{SIG}(\text{n}_B \mid \text{n}_A)_{S^{\text{A}}_{A,\mathcal{H}}}$
5. B : executa operação op
6. $B \rightarrow A$: op_ack, $\text{MAC}(\text{n}_A + 1)_{k^i_{A,B}}$

Figura 5. Estágio *Operação*.

DESCARTE(U, A, B)

1. A : {*Requisita Descarte*}
2. B : DESVINCULAÇÃO(B, U)
3. B : apaga $S^{\text{I}}_{B,\mathcal{C}}$, $S^{\text{I}}_{B,\mathcal{H}}$, $S^{\text{A}}_{B,\mathcal{H}}$ e $\mathbb{R}_B$
4. B : exibe '*descarte conluído*' na tela

Figura 6. Estágio *Descarte*.

2.7. *Descarte*

De maneira geral, *Descarte* é simplesmente uma operação especial, disponível em todos os dispositivos. Portanto, o protocolo (figura 6) é análogo ao *Operação* (figura 5). Para executar *Descarte*, um usuário U usa o dispositivo A para requisitar a operação de descarte do dispositivo B. A operação é executada se os atributos de A satisfazem o predicado relacionado à essa operação (passos 1–4 do protocolo *Operação* na figura 5). Aqui, assumimos que B pertence à U. Então, B se desvincula do seu dono U (passo 2) e apaga todas suas chaves (passo 3), de ambos os domínios *Cloud* ($S^{\text{I}}_{B,\mathcal{C}}$) e Doméstico ($S^{\text{I}}_{B,\mathcal{H}}$ e $S^{\text{A}}_{B,\mathcal{H}}$), além de todas as chaves simétricas compartilhadas com outros dispositivos (conjunto $\mathbb{R}_B$). O dispositivo B conclui o protocolo exibindo uma mensagem como '*descarte concluído*' em sua tela, que desempenha o papel de uma confirmação (passo 4). Após a execução deste protocolo o dispositivo pode ser descartado.

2.8. Aspectos Complementares

Nesta seção descrevemos algumas características complementares do AdC, em particular, como lidar com transferência de dono de um dispositivo, revogação de chaves e operação inter domínios.

Transferência de dono. Em AdC devemos contemplar a transferência de propriedade de um dispositivo. Esta tarefa é executada pelo protocolo *Transferência*, que é similar ao *Descarte* (figura 6) e as mesmas observações lá feitas são aplicáveis aqui. Contudo, as chaves relacionadas ao domínio *Cloud* não são apagadas. Para mimetizar uma confirmação, o dispositivo deve exibir uma mensagem da forma '*transferência concluída*' em sua tela. A partir daí, o antigo dono do dispositivo pode enviar ao novo dono um código de acesso e, então, entregar o dispositivo ao novo usuário.

Revogação de Chaves. Revogação não é o foco principal do AdC. Apesar disso, como mecanismo de revogação no AdC, nós seguimos o esquema proposto por [Boneh and Franklin 2001]. A estratégia consiste em associar as identidades (chaves públicas) com datas de validade. Assim, as chaves são instantaneamente revogadas assim que a validade expira. A granularidade da validade pode ser definida pelo servidor Doméstico ou *Cloud* (*e.g.,* renovação a cada dia).

Operação Inter Domínios. O AdC define o protocolo *AcordoDeChavesInterDomínio* para possibilitar a interoperação entre um dispositivo de um domínio Doméstico estrangeiro (*i.e.,* um visitante) e um dispositivo do domínio Doméstico local. O protocolo é baseado no trabalho de [McCullagh and Barreto 2005], onde, a partir de um acordo de parâmetros públicos dos respectivos criptossistemas CBI, é possível que dispositivos de diferentes domínios derivem uma mesma chave par a par. O protocolo tem como premissa que a chave pública do domínio Doméstico visitado seja obtida de forma autenticada. Com uma chave par a par estabelecida entre os dispositivos visitante e local, digamos A e B respectivamente, A pode requisitar operações em B. Para tal, A deve executar um protocolo similar ao *Operação* (figura 5). Como A não possui atributos no domínio visitado, o domínio de B deve definir um atributo 'visitante' que possa ser semanticamente atribuído aos dispositivos visitantes, *i.e.,* sem a existência de chaves criptográficas. Desta forma, B permite que A realize alguma operação apenas se o predicado de tal operação é satisfeito pelo atributo 'visitante', após um desafio-resposta baseado em MAC.

3. Desenvolvimento

Nesta seção descrevemos a arquitetura e implementação do protótipo do AdC, apresentando detalhes da camada criptográfica do protocolo.

Arquitetura. A nossa arquitetura é composta por (i) um servidor *Cloud* que estabelece o domínio *Cloud* (fabricante de dispositivos); (ii) múltiplos servidores Domésticos que refletem os diversos domínios Domésticos (*e.g.,* casas dos usuários); e (iii) dispositivos de IdC. O servidor *Cloud* pode fazer alocação de recursos (CPU, armazenamento, memória e banda) sob demanda, cenário típico de ambientes *cloud*. Os servidores Domésticos não necessitam da mesma escalabilidade do servidor *Cloud*, entretanto, assumimos que estão sempre disponíveis e tem capacidade suficiente para controlar centenas de dispositivos. Eles podem ser implantados em consoles de videogames, *desktops*, *gateways* de redes locais ou Módulos de Segurança em Hardware (*Hardware Security Modules* – HSMs). Por fim, os dispositivos se conectam a seus domínios *Cloud* e Doméstico e, em termos de plataforma, podem variar amplamente, inclusive na limitação de recursos.

Implementação. Os servidores *Cloud* e Doméstico são implementados em LAMP (Linux, Apache, MySQL e PHP), enquanto que a implementação dos dispositivos se dá em *smartphones* Android. Os servidores comunicam-se com os dispositivos usando o protocolo HTTP e invocam do PHP os esquemas criptográficos previamente compilados. Nos dispositivos Android, as interfaces de usuário são implementadas utilizando a interface nativa e a chamada das funções criptográficas é feita via Interface Nativa Java (*Java Native Interface* – JNI).

Criptografia. A camada de software criptográfico é construída em linguagem C sobre uma versão estendida da biblioteca criptográfica RELIC [Aranha and Gouvêa]. A implementação é compartilhada por todas as entidades e acessada através da chamada de métodos nativos. A RELIC foi escolhida pelo fato de suportar dispositivos com restrição de recursos (*e.g.,* processadores de 8 e 16 bits e 4KB de RAM) e, também, por implementar de forma eficiente diversos algoritmos criptográficos baseados em curvas elípticas em diferentes níveis de segurança. Nossa versão estendida da biblioteca inclui otimização por meio de código *assembly* elaborado especificamente para a arquitetura ARM.

Em relação ao ABA, esquema chave no mecanismo de controle de acesso do AdC,

nossa implementação é baseada no trabalho de [Maji et al. 2008]. Trata-se da primeira implementação aberta de um esquema de assinaturas para CBA. A implementação é dependente de Criptografia Baseada em Emparelhamentos (CBE) e de *Monotone Spam Programs* – (MSPs), que são matrizes usadas para representar os predicados do AdC como expressões booleanas.

A complexidade computacional dos algoritmos ABA mais executados no AdC, assinatura e verificação (ou sua versão probabilística), depende das operações de CBE, mais precisamente, do produto de emparelhamentos. Essa característica aumenta não apenas a complexidade de tempo mas também o consumo de memória. Para atacar este problema, otimizamos a RELIC para computar produtos de emparelhamentos de forma simultânea. A computação de emparelhamentos pode ser dividida em duas fases: o *laço de Miller*, que consiste de um algoritmo de quadrado e multiplicação e a exponenciação final. Quando um produto de emparelhamentos é computado simultaneamente (operação de multi-emparelhamento), os quadrados na extensão completa do corpo e a exponenciação final podem ser compartilhados para todos os emparelhamentos, mantendo uma variável única para acumular os resultados parciais do laço de Miller. Esta otimização evita o armazenamento de um elemento do grupo multiplicativo $\mathbb{G}_T$ e cerca de 50% de multiplicações de corpo finito por emparelhamento adicional computado no produto.

A nossa implementação do AdC combina, ainda, os seguintes protocolos e algoritmos criptográficos. (i) *Sakai-Ohgishi-Kasahara (SOK)* para acordo de chaves simétricas. (ii) *Boneh-Franklin (CBE)* para cifração CBI (adaptado para utilizar emparelhamentos assimétricos). (iii) *Bellare-Namprempre-Nevem (ABI)* para assinaturas CBI. (iv) *Maji-Prabhakaran-Rosulek (ABA)* como ABA resistente a conluio. (v) *keyed-Hash Message Authentication Code (HMAC)* para MAC (baseado em *hash* SHA256). (vi) *Advanced Encryption Standard (AES)* para cifração simétrica, modo CBC.

Parametrização. Os protocolos criptográficos do AdC foram implementados usando curvas equipadas com emparelhamentos bilineares e com grau de mergulho de 12, suficientes para os níveis de segurança de 80 e 128 bits. Em particular, nós adotamos as curvas [Barreto and Naehrig 2005], parametrizadas pelos inteiros $u = (2^{38} + 2^{32} + 2^5 + 1)$ no nível de 80 bits e $u = -(2^{62} + 2^{55} + 1)$ no nível de 128 bits. As curvas desta família são definidas por um número primo módulo $p = 36u^4 + 36u^3 + 24u^2 + 6u + 1$, ordem $n = 36u^4 + 36u^3 + 18u^2 + 6u + 1$ e suportam uma construção eficiente e otimal de emparelhamentos. Nesta instanciação, os elementos de grupos aditivos tem tamanho 40 e 80 bytes, e do grupo multiplicativo 240 bytes no nível de segurança de 80 bits; e, respectivamente, 64, 128 e 384 bytes no nível de segurança de 128 bits. Multiplicação por escalar de base fixa e variável foram implementadas usando método *comb* de tabela única e exponenciação em janela NAF, respectivamente. Os parâmetros para nível de segurança de 80 bits se destinam a sistemas legados e dispositivos com restrições de recursos, enquanto que parâmetros de segurança de 128 bits são os atuais candidatos a padronização e garantem segurança a longo prazo e eficiência.

4. Avaliação

Nesta seção avaliamos a segurança do AdC utilizando uma ferramenta de verificação automática de protocolos de segurança e mostramos que nossa solução é adequada para dispositivos embarcados com restrição de recursos.

4.1. Segurança

O AdC garante as seguintes propriedades de segurança: (i) autenticação, usando MACs e assinaturas digitais; (ii) confidencialidade, usando cifras; (iii) *freshness*, usando *nonces* e contadores; (iv) integridade, usando MACs, assinaturas digitais e funções de *hash*; e (iv) não-repúdio, usando assinaturas digitais.

Para assegurar a segurança do AdC utilizamos o *Scyther* [Cremers 2008] – uma ferramenta de verificação automática de protocolos de segurança. A ferramenta faz sua análise verificando se um dado protocolo é vulnerável a ataques que poderiam violar certas propriedades de segurança. O *Scyther* assume que as primitivas criptográficas são ideais do ponto de vista de segurança e encontra falhas decorrentes de trocas de mensagens nos protocolos. Todas as propriedades garantidas pelo AdC foram verificadas com sucesso pela ferramenta. O *Scyther* foi escolhido, principalmente, por questões de eficiência e suporte a diferentes classes de ataques [Cremers 2008].

4.2. Avaliação Analítica

Nós avaliamos os custos computacionais, quantificados em função do número de primitivas criptográficas de maior custo – emparelhamentos ($\hat{e}$) e multiplicações de pontos de curvas elípticas por escalar (p)[3] e a sobrecarga de comunicação do AdC, quantidade extra de bytes devido à criptografia. A tabela 1 sumariza os resultados.

Tabela 1. Custo computacional e sobrecarga de comunicação.

	Custo Computacional		Comunicação
Esquema	**Remetente**	**Destinatáio**	**Sobrecarga(bytes)**
ABA [Maji et al. 2008]	$(3+2l+2lt)\hat{e}$	$(2lt+1)p+(lt+t+3)\hat{e}$	$130+65l+129t$
ABA Probabilístico [Maji et al. 2008]	$(3+2l+2lt)\hat{e}$	$(2lt+t+2)p+(l+2)\hat{e}$	$130+65l+129t$
ABI [Cao et al. 2008]	$1p$	$3p$	133
EBI [Boneh and Franklin 2001]	$1\hat{e}+2p$	$1\hat{e}+1p$	129
SOK [Sakai et al. 2000]	$1\hat{e}$	$1\hat{e}$	65

O esquema criptográfico mais custoso é o ABA, cerne do mecanismo de controle de acesso e o mais frequentemente utilizado no AdC. Por isso, as próximas análises serão focadas no ABA. Seu custo computacional cresce com o tamanho das matrizes MSPs, que representam os predicados. Uma MSP tem dimensões $l \times t$, onde l é o número de atributos e t o número de operadores *AND* ($\wedge$) do predicado mais um.

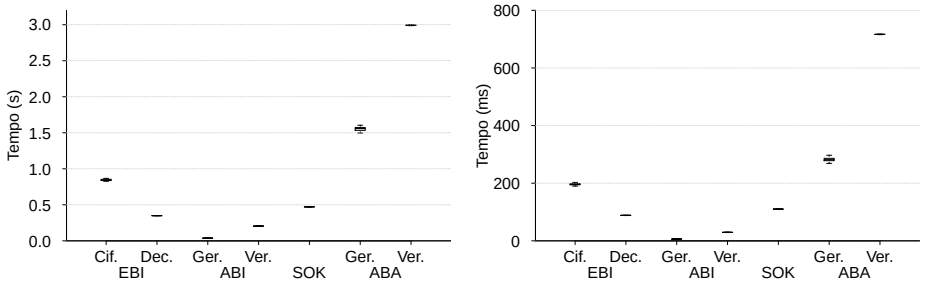
A sobrecarga de comunicação é computada considerando *nonces* de 16 bytes e requisições (req) e confirmações (ack) de 1 byte. No ABA, tal sobrecarga é função do tamanho das MSPs, mas ainda assim, é pequena o suficiente para caber em um ou poucos pacotes de rede.

4.3. Experimentos

Os esquemas criptográficos presentes no AdC foram avaliados em duas plataformas: um Arduino Due (processador ARM M3 de 32 bits e 84 MHz; 96 KB de RAM; 512 KB de memória *flash*) e um Intel Edison (processador Atom de 32 bits e 500 MHz; 1 GB de RAM; 4 GB de memória *flash*). As figuras 7(a) e 7(b) mostram os tempos de execução para: *Cif*ração, *Dec*ifração, *Ger*ação de assinatura, e *Ver*ificação de assinatura nas plataformas Due e Edison, respectivamente. Cada esquema foi executado 100 vezes. Aqui, exibimos os quartis com o 5º e o 95º percentis. Todos esquemas executam em tempo

[3] As outras operações (*e.g.,* primitivas simétricas) são executadas em tempo desprezível.

razoável.[4] Como esperado, o ABA é o esquema mais custoso. No Due são necessários 1,6s para gerar assinaturas e menos de 3,0s para verificá-las, usando predicados da forma $A \wedge B$. Já no Edison estes números chegam a cerca de 300ms e 750ms, respectivamente.



(a) Due (b) Edison

Figura 7. Tempos de execução.

Complementando a figura 7(a), a figura 8 mostra os tempos de execução para o ABA no Due, com predicados da forma $A \wedge B$, diferentes níveis de segurança (80 e 128 bits) e as verificações determinística e probabilística de assinatura. Os resultados mostram um compromisso entre nível de segurança e tempos de execução. Em particular, a verificação probabilística, no nível de 80 bits, permite que assinaturas sejam verificadas em cerca de 1,2s, tempo razoável para a maioria das aplicações interativas. Comparada à sua versão determinística, a abordagem probabilística troca determinismo por eficiência computacional, entretanto, a segurança do algoritmo não é prejudicada por essa troca, a menos de um fator desprezível.

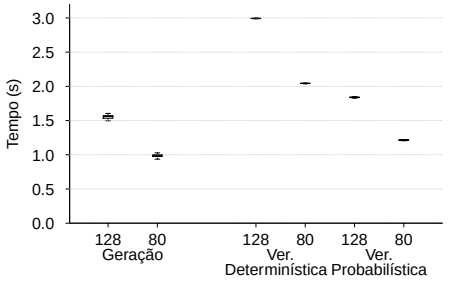


Figura 8. ABA no Due.

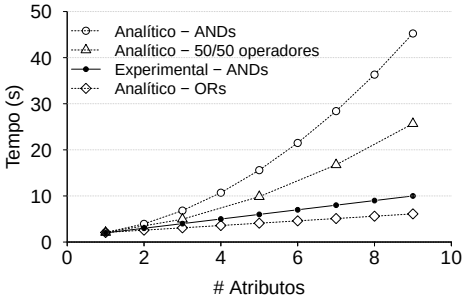


Figura 9. Ver. ABA no Due.

Para verificar como o AdC escala em cenários realísticos de IdC, avaliamos a relação entre tempo de computação e o tamanho dos predicados. A figura 9 mostra os tempos de verificação de assinatura para diferentes predicados. No eixo x representamos a variação no número de atributos. Cada curva corresponde a uma configuração do número de operadores *AND*. Note-se que algumas destas curvas correspondem a tempos de verificação estimados analiticamente. Elas são obtidas da combinação dos tempos de execução das operações de emparelhamento e multiplicação de pontos de curvas elípticas por escalar na plataforma Due (cerca de 355ms e 74,5ms, respectivamente) com os custos analíticos apresentados na tabela 1. A curva experimental da figura 9 mostra que o Due é capaz de verificar assinaturas com predicados complexos com até nove atributos e somente operadores *AND* em cerca de dez segundos. Para predicados mais usuais, entre um e três atributos, a verificação de uma assinatura é feita em menos de seis segundos.

[4]Esquemas simétricos (AES e MAC) foram omitidos pois executaram em menos de 2,5ms.

Pela figura 9 nota-se que os tempos de execução analíticos superestimam significativamente os tempos de execução. Há duas razões principais para isso. Primeiro, nossa implementação é otimizada para computar multi-emparelhamentos simultaneamente (seção 3). Segundo, os custos analíticos consideram que uma matriz MSP de coeficientes $t \times l$ não contém zeros. Na prática, contudo, as matrizes de coeficiente são esparsas, fato que reduz significativamente o número de operações.

Considerando-se o uso de memória do ABA, detectamos a máxima utilização da memória (valor de pico) como o maior tamanho de pilha alcançado durante a execução (a RELIC só aloca memória na pilha). Como esperado, o ABA é adequado mesmo para dispositivos com restrição como o Due, requerendo menos de 19 KB para um predicado com nove atributos e oito operadores *AND*. A utilização de memória deve ser idêntica em outros processadores de 32 bits e um pouco maior em processadores de 64 bits.

Em termos de armazenamento, a implementação da RELIC estendida mais o AdC ocuparam 147 KB da memória do Due, deixando espaço significativo para outras aplicações.

5. Trabalhos Relacionados

Embora a autenticação seja diferente, em termos de conceito e finalidade, de controle de acesso, os dois assuntos são intimamente relacionados. Trabalhos de segurança que discutem dispositivos com limitação de recursos normalmente sugerem o uso de autenticação para estabelecer um mecanismo de controle de acesso. Adiante, descrevemos brevemente trabalhos sobre (i) segurança para dispositivos com limitação de recursos e (ii) esquemas de autenticação e controle de acesso para dispositivos de IdC.

Segurança em dispositivos com limitação de recursos. Antes do advento de IdC, diversos trabalhos discutiram segurança em MANETS e redes de sensores em geral. Os estudos para MANETs (*e.g.,* [Zhou and Haas 1999, Venkatraman and Agrawal 2002]) não são aplicáveis a IdC porque, em sua maioria, consideram a existência de dispositivos com maior poder computacional que aqueles encontrados em IdC. Já os trabalhos destinados a redes de sensores (*e.g.,* [Perrig et al. 2001, Zhu et al. 2003, Liu et al. 2005, Oliveira et al. 2008, Patil and Szygenda 2012]) usualmente se baseiam em premissas que não se aplicam a IdC, logo, essas propostas não podem ser utilizadas. Por exemplo, em uma rede de sensores todos os dispositivos executam a mesma aplicação e são gerenciados por uma única entidade. Em IdC, por outro lado, os dispositivos executam diferentes aplicações e respondem a mais de uma autoridade. Nossa solução AdC provê autenticação e controle de acesso para IdC, atendendo aos requisitos e restrições específicos de IdC.

Autenticação e controle de acesso para IdC. Na literatura, os trabalhos que abordam autenticação e controle de acesso sob uma perspectiva de IdC propõe diferentes estratégias, *e.g.,* através de políticas de gerenciamento, verificação de dados, aplicação de CBI, tecnologias específicas para pareamento de dispositivos, definição de padrões de segurança, entre outros. A seguir descrevemos brevemente alguns trabalhos que dão uma visão geral do estado da arte.

Recentemente, [Liang et al. 2015] apresentaram *Safe Internet oF Things* – SIFT, uma plataforma de programação centrada em segurança para dispositivos conectados em ambientes de IdC. No SIFT, os usuários expressam suas intenções de alto nível em

aplicações de IdC e o esquema, então, decide quais dados e operações devem ser combinados para atender às necessidades do usuário. Para garantir segurança e conformidade, o esquema verifica se podem ocorrer conflitos ou violações de políticas dentro ou entre as aplicações, o que pode ser considerado como um mecanismo de controle de acesso. Estritamente falando, no entanto, tal trabalho tem foco na segurança dos usuários.

O trabalho de [Oliveira et al. 2009], por sua vez, aborda a autenticação da comunicação entre um dispositivo e múltiplos usuários. A proposta, chamada *Secure Tiny Web Services – Secure-TWS*, reporta a sobrecarga causada por estratégias de assinatura digital, mais especificamente, pelo esquema de Assinatura Digital Baseada em Curvas Elípticas (*Elliptic Curve Digital Signature Algorithm* – ECDSA) e o esquema de assinaturas curtas de Boneh-Lynn-Shacham (BLS).

[Mora-Afonso et al. 2013] propõe um esquema para fornecer segurança na comunicação de dispositivos domésticos. Os autores empregam CBI e *Wi-Fi*, assim como *Bluetooth* e NFC para autenticar as mensagens trocadas pelos dispositivos. O NFC é usado durante o pareamento por *Bluetooth* e a distribuição de chaves. Além disto, os autores apresentam protótipos para diversos dispositivos.

[Markmann et al. 2015] propõe um esquema de autenticação fim a fim para implantar uma federação de *gateways* em sub-redes de IdC. A estratégia proposta é, também, baseada em CBI, e, apesar de preliminar, traz contribuições significativas para a área.

Por fim, [Yavuz 2013] desenvolveu um esquema criptográfico destinado a dispositivos IdC chamado *Efficient and Tiny Authentication* – ETA. ETA é similar ao AdC no sentido de suportar autenticação e controle de acesso, entretanto, não considera todo o ciclo de vida dos dispositivos de IdC e, além disso, é baseada em dispositivos com abundância de recursos computacionais para executar operações custosas.

6. Conclusão

À medida que IdC se torna ubíqua, a troca de informações privadas, assim como a execução automática de operações nos ambientes computacionais se tornam fator crítico. Nestes cenários é imperativo que a comunicação seja autenticada e que haja meios seguros de controle de acesso às operações dos dispositivos. Este trabalho propõe AdC, uma família de protocolos que provê autenticação e controle de acesso a todo o ciclo de vida de dispositivos de IdC. O AdC possibilita autenticação forte e controle de acesso flexível através da combinação de primitivas criptográficas do estado da arte.

O AdC foi avaliado analítica e experimentalmente. Os resultados analíticos mostram que a sobrecarga imposta pelo AdC tanto para CPU quanto para comunicação, no pior caso, são gerenciáveis até mesmo em dispositivos com limitações de recursos. Nós também utilizamos uma ferramenta de verificação automática para validar as propriedades de segurança do AdC. Por fim, os resultados experimentais mostram que o AdC pode ser executado com baixo custo em dispositivos com restrições computacionais e com custo desprezível em dispositivos computacionalmente poderosos.

Referências

Aranha, D. F. and Gouvêa, C. P. L. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.

Ashton, K. (2009). That 'Internet of Things' Thing. *RFiD Journal*, 22:97–114.

Barreto, P. S. L. M. and Naehrig, M. (2005). Pairing-friendly Elliptic Curves of Prime Order. In *SAC*.

Bethencourt, J., Sahai, A., and Waters, B. (2007). Ciphertext-policy Attribute-based Encryption. In *S&P*.

Boneh, D. and Franklin, M. K. (2001). Identity-Based Encryption from the Weil Pairing. In *CRYPTO*.

Cao, X., Kou, W., Dang, L., and Zhao, B. (2008). IMBAS: Identity-based Multi-user Broadcast Authentication in Wireless Sensor Networks. *Computer Communications*, 31(4):659 – 667.

Cremers, C. (2008). The scyther tool: Verification, falsification, and analysis of security protocols. In *CAV*.

Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006). Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. In *CCS*.

Liang, C.-J. M., Karlsson, B. F., Lane, N. D., Zhao, F., Zhang, J., Pan, Z., Li, Z., and Yu, Y. (2015). SIFT: Building an Internet of Safe Things. In *IPSN*.

Liu, D., Ning, P., and Li, R. (2005). Establishing Pairwise Keys in Distributed Sensor Networks. *TISSEC*.

Maji, H. K., Prabhakaran, M., and Rosulek, M. (2008). Attribute-Based Signatures: Achieving Attribute-Privacy and Collusion-Resistance. *IACR Cryptology ePrint Archive*, 2008:328.

Margi, C., Simplicio, M., Barreto, P., and Carvalho, T. (2009). Segurança em redes de sensores sem fio. *Minicursos: SBSeg*.

Markmann, T., Schmidt, T. C., and Wählisch, M. (2015). Federated End-to-End Authentication for the Constrained Internet of Things Using IBC and ECC. In *SIGCOMM*.

McCullagh, N. and Barreto, P. S. L. M. (2005). A New Two-party Identity-based Authenticated Key Agreement. In *CT-RSA*.

Mora-Afonso, V., Caballero-Gil, P., and Molina-Gil, J. (2013). Strong Authentication on Smart Wireless Devices. In *FGTC*.

Oliveira, L. B. and Dahab, R. (2006). Pairing-based cryptography for sensor networks. In *NCA*.

Oliveira, L. B., Kansal, A., Priyantha, B., Goraczko, M., and Zhao, F. (2009). Secure-TWS: Authenticating Node to Multi-user Communication in Shared Sensor Networks. In *IPSN*.

Oliveira, L. B., Scott, M., Lopez, J., and Dahab, R. (2008). TinyPBC: Pairings for Authenticated Identity-Based Non-Interactive Key Distribution in Sensor Networks. In *INSS*.

Patil, H. K. and Szygenda, S. A. (2012). *Security for Wireless Sensor Networks Using Identity-Based Cryptography*. CRC Press.

Perrig, A., Szewczyk, R., Wen, V., Culler, D. E., and Tygar, J. D. (2001). SPINS: Security Protocols for Sensor Netowrks. In *MobiCom*.

Sakai, R., Ohgishi, K., and Kasahara, M. (2000). Cryptosystems Based on Pairing. In *SCIS*.

Shamir, A. (1984). Identity-based Cryptosystems and Signature Schemes. In *CRYPTO*.

Silva, B., da Silva Jr, D. C., Souza, E. M., Pereira, F., Teixeira, F., Wong, H. C., Nazaré, H., Maffra, I., Freire, J., Santos, W. F., et al. (2013). Segurança de software em sistemas embarcados: Ataques & defesas. *Minicursos: SBSeg*.

Stinson, D. (2002). *Cryptography: Theory and Practice*. CRC/C&H.

Venkatraman, L. and Agrawal, D. P. (2002). A novel authentication scheme for ad hoc networks. In *WCNC*.

Wangham, M. S., Domenech, M. C., and de Mello, E. R. (2013). Infraestrutura de autenticação e de autorização para internet das coisas. *Minicursos: SBSeg*.

Yavuz, A. A. (2013). ETA: Efficient and Tiny and Authentication for Heterogeneous Wireless Systems. In *WiSec*.

Yuan, E. and Tong, J. (2005). Attributed Based Access Control (ABAC) for Web Services. In *ICWS*.

Zhou, L. and Haas, Z. J. (1999). Securing Ad Hoc Networks. *IEEE Network*, 13(6):24–30.

Zhu, S., Setia, S., and Jajodia, S. (2003). LEAP: Efficient Security Mechanisms for Large-scale Distributed Sensor Networks. In *CCS*.

Metodologia de Detecção de *Malware* por Heurísticas Comportamentais

Neriberto Prado[1], **Ulisses Penteado**[1], **André Grégio**[2]

[1] Bluepex Security Solutions
Limeira – SP – Brasil

[2]Departamento de Informática (DInf)
Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

Abstract. *Malicious programs have evolved either in sophistication as in complexity level, increasing the rate of successful attacks against computer systems and their users. Such as any benign program, malicious ones need to interact with the underlying operating system so as to perform their intendend activities. Thus, there is a need for understanding which of the performed actions are related to infection procedures. These "suspicious" actions mold the malware's execution behavior, whose identification is decisive for detection. In this paper, we propose a methodology to detect malware based on behavioral heuristics. We also present tests and results of its application in actual samples.*

Resumo. *Programas maliciosos têm evoluído em sofisticação e complexidade, aumentando a incidência de ataques bem sucedidos contra sistemas computacionais e seus usuários. Como qualquer programa benigno, os programas maliciosos precisam interagir com o sistema operacional de forma a realizar as atividades pretendidas. Assim, faz-se necessário compreender quais das ações efetuadas estão envolvidas em processos de infecção. Tais ações "suspeitas" compõem o comportamento de execução do malware e sua identificação é crucial na detecção desses programas. Neste artigo, propõe-se uma metodologia para detecção de malware baseada em heurísticas comportamentais e apresenta-se os testes e resultados obtidos de sua aplicação em exemplares reais.*

1. Introdução

Programas maliciosos, também conhecidos como *malware*, são programas de computador que atuam de forma a subverter o sistema operacional ou a executar ações que beneficiem o atacante (por exemplo, roubo de credenciais da vítima). Dado que *malware*, em essência, é um tipo de programa como qualquer outro programa legítimo, é necessário conhecer que tipos de ações executadas por este podem estar envolvidas em processos de infecção. Tais ações, consideradas suspeitas, quando executadas de forma encadeada levam ao comprometimento da vítima, incorrendo em ataques contra a integridade, confidencialidade e/ou disponibilidade dos dados, do sistema ou do usuário-alvo em questão. A identificação de ações suspeitas e sua atribuição a um processo de infecção levam à detecção de uma execução, processo ou programa malicioso.

A detecção de programas maliciosos pode ser feita de forma estática ou dinâmica, isto é, por meio da análise do arquivo binário que representa o exemplar do *malware* ou da

execução de tal arquivo em um emulador (*engine*) ou ambiente controlado (*sandbox*). A análise estática é um procedimento que pode ser bastante custoso, devido ao uso de algoritmos de força-bruta (abordagem *greedy*) que tentam seguir todos os ramos em um fluxo de programa, exaurindo as possibilidades presentes em saltos e estruturas condicionais presentes nas instruções. De maneira mais simples, a análise estática pode prover informações importantes em uma pré-análise, seja através de dados extraídos do cabeçalho do *malware* (ofuscação, nomes de seções, entropia) ou por dados encontrados no conteúdo do binário (chamadas de funções efetuadas, porções de texto, endereços IP e de correio eletrônico, URLs, etc.). Entretanto, caso o *malware* sob análise esteja ofuscado por um empacotador ou cifrador, esta não terá sucesso a menos que o *packer* seja removido de antemão.

No caso da análise dinâmica, o exemplar de *malware* é efetivamente executado, ou seja, suas instruções são "acionadas" em um emulador por um número finito de ciclos ou em uma *sandbox* por alguns minutos, permitindo que a infecção ocorra. Dessa forma, o *malware* pode exibir seu comportamento e o mecanismo de monitoração pode obter as ações que são realmente feitas, evitando o custo de se tentar seguir todos os ramos presentes na análise estática e tornando possível a desofuscação do exemplar. A monitoração via análise dinâmica consiste em interceptar chamadas de sistema e de API em busca de ações suspeitas. Quando um certo conjunto de ações suspeitas é observado, é possível "marcar" o programa que originou essas ações e tomar alguma medida cabível, que varia entre emitir um alerta, colocá-lo em quarentena, desinfectar o sistema, entre outras. A observação das ações suspeitas depende da definição de heurísticas que representem comportamentos a serem correspondidos.

Neste artigo, introduz-se uma metodologia para detecção de programas maliciosos que se utiliza de heurísticas definidas com base em ações suspeitas, isto é, aquelas que são comumente executadas por *malware* ou têm o potencial de subverter os sistemas-alvo. No decorrer do trabalho, apresenta-se o modelo heurístico criado, as heurísticas propriamente ditas, um estudo de caso envolvendo a análise dinâmica de *malware* para extração dos comportamentos e os testes e resultados da detecção de programas maliciosos por meio da aplicação de aprendizado de máquina. Outras importantes contribuições deste trabalho incluem uma abordagem de detecção bastante simples em relação às disponíveis, bem como uma extensa revisão do estado da arte. O restante do artigo está organizado como segue: na Seção 2, são mostradas abordagens vistas na literatura para detecção de *malware* através de seu comportamento; na Seção 3, é apresentado o modelo heurístico proposto em detalhes, os comportamento definidos, suas descrições e riscos associados, bem como o algoritmo para detecção desses comportamentos em programas com base nas heurísticas propostas; na Seção 4, são mostrados os testes realizados para verificar a capacidade de detecção do modelo gerado e a discussão dos resultados obtidos; na Seção 5, são apresentadas as considerações finais deste trabalho.

2. Trabalhos Relacionados

[Christodorescu et al. 2005] descreveram um modelo de detecção de programas maliciosos baseado em *templates*. Estes são definidos para os seguintes comportamentos: *decryption-loop*—programas que se desofuscam/decifram em memória após execução—e *mass-mailing*—funcionalidade de envio de *spam* comumente associada a *worms*. A detecção por *templates* foi feita pela comparação dos grafos de controle de fluxo (CFG)

contendo as instruções obtidas do *malware* por *disassembling* via `IDA Pro`[1]. A abordagem almejou identificar principalmente *malware* polimórfico (capacidade de gerar mutantes), mas os testes foram limitados a três famílias de *malware* mutante da época e aos dois *templates* semânticos mencionados.

[Yin et al. 2007] apresentaram Panorama, um sistema para detecção automatizada de *malware* por meio de um processo de testes, monitoração e análise de programas em execução. A parte de testes foi feita pela inserção no sistema de dados sensíveis "marcados" não direcionados ao programa sob análise; a monitoração consistiu em identificar se o *malware* fez algum acesso a esses dados marcados e qual sua ação sobre eles (por exemplo, envio desse dado pela rede visando roubo de informações); a etapa de análise foi realizada sobre os grafos gerados do fluxo de dados marcados, aos quais foram aplicadas políticas definidas para detecção de comportamento malicioso. Essas políticas pretendiam detectar *malware* especificamente envolvido em vazamento de informações da vítima, incluindo senhas, texto, documentos, páginas Web e protocolos de rede.

[Ye et al. 2007] apresentaram o *Intelligent Malware Detection System* (IMDS), um sistema de classificação baseado em regras que age sobre as sequências de execução das APIs do Windows. O objetivo principal do IMDS foi o de detectar *malware* polimórfico ou desconhecido via associações encontradas em chamadas de API obtidas diretamente do binário em formato PE. Para tanto, calculou-se o suporte e a confiança relacionados ao encadeamento das APIs, atribuindo o rótulo "maligno" ou "benigno" ao exemplar que realizou tais chamadas. A interpretação do comportamento executado pelo conjunto de chamadas de API careceu de interpretação adicional. Os autores compararam os resultados obtidos na detecção via IMDS com alguns antivírus famosos e da técnica de classificação por associação *a priori* com outros algoritmos de aprendizado de máquina.

[Jacob et al. 2008] apresentaram a detecção comportamental de *malware* como um processo de inferência, uma vez que tentou-se identificar quais ações de um programa foram realizadas com intuito de atacar o sistema ou seus usuários. Os autores discutiram os problemas deste tipo de detecção que, embora permitisse a identificação de exemplares desconhecidos, requeria a definição de comportamentos capazes de separar adequadamente os programas benignos dos maliciosos. O processo de definição desses comportamentos foi complexo e envolveu a criação de diversos perfis, ou heurísticas, que detectassem tipos distintos de *malware*. Para abordar o problema, os autores fizeram um levantamento da literatura sobre o tema e criaram uma taxonomia voltada para a detecção de *malware* baseada em comportamento. A taxonomia proposta abrangeu a extração de dados via análise estática e dinâmica, atuação sobre instruções, funções, chamadas de sistemas, grafos de fluxo e abstrações algébricas, e detecção por algoritmos heurísticos, classificadores diversos, sistemas especialistas e verificação de modelos. Com isso, teve-se um panorama bastante completo do domínio entre 1994 e 2007.

[Griffin et al. 2009] apresentaram Hancock, um sistema para geração automática de assinaturas para detecção de *malware*. Hancock usava cadeias de caracteres de 48 bytes para identificar sequências de código maliciosas dentro de um arquivo, generalizando o processo de comparação por *hashes*, o qual em geral requer uma assinatura por exemplar. A fim de minimizar a quantidade de falsos-positivos, Hancock comparava as potenciais

[1] `https://www.hex-rays.com/products/ida/index.shtml`

assinaturas de 48 bytes com um modelo criado a partir de programas conhecidamente benignos, descartando aquelas que apresentam correspondência em tal modelo. Para gerar as assinaturas, o sistema dependia que o exemplar esteja desofuscado (a fim de que não fosse criada uma assinatura do *packer*) e que as instruções tivessem sido obtidas por *disassembling* (via `IDA Pro`). Testes do sistema resultaram em taxa de falsos-positivos próxima de zero, porém baixa cobertura de detecção, dada a ofuscação de exemplares e a variedade de famílias. Mostrou-se, com isso, que soluções para a geração automática de assinaturas, sem a intervenção humana e sem a execução do exemplar (análise dinâmica) ainda não são suficientemente maduras para lidar com o problema de detecção de *malware*

[Treadwell and Zhou 2009] propuseram uma abordagem para detecção por heurísticas cujo alvo é *malware* com *packers*. Esse tipo de exemplar, ao ser ofuscado, apresenta certos padrões em seu cabeçalho que podem prover pistas para sua identificação. Os autores se aproveitaram disto para criar detectores que atuavam antes do processo de desofuscação, impedindo assim a execução do programa malicioso. Com base em oito características extraídas estaticamente do cabeçalho PE de *malware*, foi calculado um nível de risco inicial baseado na presença dessas que, se alcançasse um determinado limiar, alertava o exemplar como suspeito.

[Bazrafshan et al. 2013] discutiram as limitações da detecção de *malware* por correspondência de padrões, isto é, assinaturas, bem como abordagens da literatura baseadas em heurísticas. Tais heurísticas podem ser baseadas em chamadas de funções de APIs, grafos de controle de fluxo, n-gramas, opcodes, ou apresentar características híbridas. Essas características foram utilizadas para se identificar o comportamento de programas maliciosos e benignos, aos quais são aplicados técnicas de aprendizado de máquina. Os autores expuseram as desvantagens das técnicas disponíveis na literatura, como grafos muito grandes, alto custo computacional, quantidade de regras criadas, conjuntos de treinamento pequenos, entre outras.

[Adkins et al. 2013] propuseram um algoritmo para detectar programas maliciosos por meio da descoberta de trechos de código reutilizados. Este algoritmo, denominado BBCP, opera em blocos básicos extraídos de binários, os quais são então processados de forma a gerar uma "assinatura". O objetivo foi o de se identificar variantes ou funções maliciosas compartilhadas entre executáveis. Os blocos básicos foram obtidos pelo `IDA Pro` e as instruções foram normalizadas e agrupadas em 4-gramas para a produção de *hashes*, os quais foram utilizados na comparação por similaridade (calculada pelo índice de Jaccard).

[Ahmadi et al. 2013] introduziram um método de detecção heurístico que se utilizava de mineração de padrões iterativos de chamadas de API. Os exemplares de *malware* foram executados em máquina virtual e as chamadas de API realizadas por estes foram monitoradas e armazenadas em traços comportamentais. Cada API foi mapeada para um identificador numérico único a ser utilizado em uma sequência de detecção. Cada sequência foi então representada por um conjunto de APIs chamadas em ordem e cuja frequência de ocorrência obedecia a um limiar determinado. A classificação de um programa como malicioso envolveu associar a ele uma ou mais sequências rotuladas como tal. Esse processo foi feito por meio do algoritmo *RandomForest*.

[Alazab 2015] se propôs a diferenciar programas maliciosos de benignos por meio

de assinaturas baseadas em chamadas de API extraídas através de *disassembling* pelo `IDA Pro`. Essa detecção foi feita calculando-se a distância entre sequências de chamadas de API e a frequência em que elas ocorreram em ambas as classes. Assim, foi possível obter a similaridade dos exemplares (maliciosos e benignos) avaliados, para os quais foi também aplicado o algoritmo KNN. Embora os testes, diferente de outros trabalhos, tenham sido feitos em Windows 7, a base de exemplares maliciosos era muito antiga para ser considerada relevante perante os exemplares atualmente observados.

[Prelipcean et al. 2015] apresentaram um método estatístico para detecção de comportamentos de execução específicos em programas. Para gerar os traços comportamentais, os autores executaram os exemplares (*malware* e programas "limpos") em ambiente controlado e obtêm suas chamadas de API. Conjuntos de chamadas presentes na sequência que compõe um traço, as quais podem levar a um comportamento malicioso, foram denominados "contextos". Verificou-se então a ocorrência dos contextos nos traços capturados e analisou-se a probabilidade de que tais contextos ocorressem em traços advindos de programas limpos. Dependendo da frequência de aparição dos contextos em exemplares desconhecidos e da probabilidade deles em traços limpos, tais exemplares puderam ser considerados benignos, suspeitos ou maliciosos. A taxa de detecção obtida com a aplicação do método foi próxima de 80%, com falsos-positivos menores que 2%.

[Khodamoradi et al. 2015] propuseram uma técnica para detecção de *malware* metamórfico baseada em assinaturas feitas a partir de estatísticas sobre contagem de instruções do programa analisado. As instruções foram obtidas por análise estática, sua frequência foi calculada e os opcodes foram utilizados como características para classificação por meio de seis algoritmos de árvores de decisão. O treinamento e a classificação feitos da maneira proposta foram rápidos, mas o método utilizado poderia ser facilmente subvertido por programas maliciosos evasivos que se utilizassem de substituição de instruções. Além disso, os exemplares utilizados na avaliação correspondem em sua maioria à *malware* antigo, os quais não são mais atuantes.

[Feng et al. 2015] apresentaram HRS, uma plataforma híbrida que fazia uso de modelos baseados em *hash*, regras e *Support Vector Machines* para detecção de *malware*. O primeiro modelo foi o tradicional utilizado, de comparar *hashes* com uma lista de exemplares conhecidamente maliciosos ou benignos, enquanto que os dois último pretendiam detectar exemplares desconhecidos por meio de heurísticas ou treinamento de classificadores gerados a partir de n-gramas advindos de *scanning* do binário. Os resultados do teste com as regras mostraram uma taxa de verdadeiros-positivos de 76.10%.

[Fan et al. 2016] introduziram uma plataforma para detecção de *malware* que se utilizava de mineração de dados para encontrar padrões em sequências de instruções extraídas de arquivos do tipo PE. O arquivo a ser analisado passava por um processo de *disassembling* seguido pelo mapeamento das instruções para um identificador numérico. Tais identificadores, na ordem em que aparecem, representavam um programa e foram filtrados de forma a se retirar instruções que não eram úteis para a detecção. De acordo com a frequência das instruções, calculou-se um peso para auxiliar na classificação entre maligno e benigno. Subsequências encontradas em *malware* foram utilizadas no processo de detecção, que se utilizou de uma variação do algoritmo "k-vizinhos mais próximos" (KNN) denominada por ANN, onde o parâmetro "k" foi escolhido automaticamente. Os resultados em comparação com outros classificadores baseados em distância foram mar-

ginalmente melhores.

Cabe ressaltar que quase a totalidade dos trabalhos relacionados é ou atrelado à análise estática com *IDA Pro*—dependente da qualidade do *disassembling* e prejudicados pela ofuscação presente em *malware* modernos—ou executado em ambientes virtualizados com o sistema operacional Windows XP, mesmo na literatura mais recente. O trabalho desenvolvido neste artigo se utiliza não só de heurísticas mais genéricas para definir comportamentos suspeitos, como executa exemplares mais atuais em sistema operacional Windows com kernel NT 6.x (Windows Vista até o atual 10)[2]. Por ser feito com base em análise dinâmica, evita-se que técnicas de ofuscação atrapalhem a extração de comportamento. Entretanto, como qualquer abordagem baseada na execução de *malware*, há limitações para o caso de exemplares que aplicam técnicas evasivas (anti-análise, anti-forense, `sleep`, etc.).

3. Modelo Heurístico

Uma heurística pode ser vista como um modelo, processo ou método composto por uma ou mais regras que visam encontrar a solução para um dado problema com base em informações altamente especializadas. O resultado da aplicação de um conjunto de heurísticas pode ser uma aproximação para um programa específico, por exemplo, quais são as ações executadas por um programa que determinam se este é malicioso ou não. O modelo heurístico proposto neste artigo visa agrupar características e informações que serão utilizadas para descrever uma ação suspeita genérica. Esse modelo servirá de base para a definição das ações suspeitas que, por sua vez, serão usadas como insumo para um algoritmo de análise comportamental de programas. O algoritmo recebe dados sobre a execução de um determinado programa monitorado (também chamado de "traço") e aplica regras de comparação a fim de identificar quais das heurísticas correspondem às ações observadas. Com isso, ao fim da execução do algoritmo de análise comportamental, obtém-se um vetor de comportamentos exibidos pelo programa monitorado. Tal vetor permite que se detecte se o programa é malicioso em um passo posterior.

O modelo heurístico desenvolvido trata-se de uma estrutura de dados e pode ser facilmente armazenado em um banco de dados para facilitar as atividades de gerenciamento (inserção, remoção e edição) destas. Para representar tal estrutura de dados, modela-se um conjunto de campos com tipos específicos (em forma de tabela) que possa especificar qualquer heurística definida, isto é, ações suspeitas genéricas. Assim, pode-se partir para o processo de detecção de comportamentos suspeitos exibidos por programas, visando identificar programas maliciosos ou programas que apresentam atividades que potencialmente danosas a um sistema "alvo" (máquina da vítima). A estrutura que especifica as características que uma dada heurística deve conter é ilustrada na Tabela 1.

3.1. Descrição das características

As descrições dadas a seguir detalham as características elicitadas (vistas na Tabela 1) para a definição geral de uma heurística de detecção comportamental.

- **Identificador da heurística**: número inteiro crescente que serve como identificador da heurística (e chave primária de uma possível tabela de heurísticas em um banco de dados);

[2]O estudo de caso aqui realizado utilizou-se do Windows 7.

Tabela 1. Modelagem dos campos (características) que representam uma heurística para identificação de comportamento suspeito.

Campo	Tipo
Identificador da heurística	número inteiro sequencial
Nome da heurística	rótulo composto separado por "_"
Descrição	explicação do comportamento exibido
Risco associado	número inteiro no intervalo [1..5]
Categoria	tipo geral do comportamento
Padrão para detecção	expressão regular

- **Nome da heurística**: o nome é composto pelo evento (ex.: modificação, roubo, evasão, etc.) e/ou comportamento exibido (ex.: persistência, exfiltração de credenciais, anti-debugging, etc.), alvo da ação do programa monitorado (ex.: Registro, sistema de arquivos, processos, etc.) e variação, representada por um número inteiro sequencial que indica variantes de comportamentos que podem ser enquadrados no mesmo nome. Esta característica foi inspirada na taxonomia de comportamentos suspeitos definida por [Grégio et al. 2015];
- **Descrição**: indica que tipo de ação suspeita o programa monitorado exibiu, permitindo que se tenha informações de mais alto nível sobre o comportamento apresentado e a extensão do dano causado por uma possível infecção;
- **Risco associado**: define o nível do risco relacionado à atividade monitorada, podendo servir para identificar uma ameaça ou compor um índice com o somatório dos riscos atribuídos a um determinado processo. Os valores de risco vão de 1 (menos grave) até 5 (mais crítico) e foram inspirados na tabela apresentada por [Grégio et al. 2016], na experiência dos autores em anos de análise de *malware* e na observação de que determinados comportamentos aparecem mais frequentemente associados a programas maliciosos;
- **Categoria**: define uma classe para explicitar o tipo de comportamento observado, permitindo o agrupamento de exemplares similares e a motivação por trás da referida ação;
- **Padrão para detecção**: consiste em uma string especialmente criada para servir como um meio rápido para se detectar a ocorrência de um comportamento suspeito. É a regra que aciona a heurística definida nos demais campos.

3.2. Detecção Comportamental

A detecção comportamental é feita pela definição de comportamentos suspeitos associados à execução de programas maliciosos, os quais são então mapeados cada qual para um padrão (expressão regular ou cadeia de caracteres representativa). Este, por sua vez, é usado para se criar um *template* heurístico cujo conjunto produzido é comparado com o traço comportamental de programas analisados dinamicamente. A ocorrência de um dado *template* ativa seu comportamento em um vetor associado ao programa em questão e, ao final dessa análise, tem-se todas as atividades suspeitas efetuadas por cada programa. Mais adiante, os vetores rotulados como associados a programas maliciosos ou benignos são utilizados no treinamento de um classificador. Os comportamentos definidos inicialmente, para os quais *templates* heurísticos foram gerados, são os seguintes:

1. **Evasão**: modificação da configuração de notificação das atualizações do sistema. (*Risco: 4*)
2. **Evasão**: modificação da configuração do *firewall* nativo do sistema. (*Risco: 4*)
3. **Evasão**: chama o comando `sc.exe` para modificar configurações ou aplicações do sistema. (*Risco: 3*)
4. **Evasão**: chama o comando `netsh.exe` para modificar configurações privilegiadas do sistema. (*Risco: 3*)
5. **Evasão**: chama um terminal para execução de comandos ou programas. (*Risco: 1*)
6. **Evasão**: remoção de evidências do comprometimento do sistema. (*Risco: 2*)
7. **Evasão**: desabilita a inicialização em modo de segurança. (*Risco: 3*)
8. **Subversão do sistema operacional**: altera chaves de Registro para modificar políticas de segurança vigentes. (*Risco: 5*)
9. **Subversão do sistema operacional**: altera chaves de Registro para instalar automaticamente novos controles ActiveX. (*Risco: 5*)
10. **Subversão do sistema operacional**: altera chaves de Registro para abrir automaticamente certos tipos de arquivo. (*Risco: 4*)
11. **Subversão do sistema operacional**: altera comandos ou opções nos menus de contexto de arquivos do Windows no Registro. (*Risco: 3*)
12. **Subversão do sistema operacional**: carrega um *driver* na vítima, que pode ser um *rootkit*. (*Risco: 5*)
13. **Subversão do sistema operacional**: tenta executar arquivo `.exe`, criando novo processo. (*Risco: 2*)
14. **Subversão do navegador**: altera configurações de zonas de segurança do Internet Explorer no Registro. (*Risco: 4*)
15. **Subversão do navegador**: insere barras de ferramentas ou *Browser Helper Objects* no Internet Explorer. (*Risco: 4*)
16. **Subversão do navegador**: altera configurações de servidor de proxy do Internet Explorer no Registro. (*Risco: 4*)
17. **Subversão do navegador**: altera o arquivo de *hosts* para evitar o acesso a *sites* legítimos ou redirecionar para *sites* maliciosos. (*Risco: 5*)
18. **Subversão do navegador**: carrega um arquivo do tipo PAC (*Proxy Auto Configuration*) para redirecionar a navegação do usuário, o que pode ser usado para roubar credenciais de Internet Banking. (*Risco: 4*)
19. **Subversão das configurações de Internet**: altera zonas de segurança do Internet Explorer no Registro. (*Risco: 4*)
20. **Persistência**: modifica o Registro ou sistema de arquivos para executar na inicialização do sistema. (*Risco: 3*)
21. **Persistência**: altera chaves de Registros do WinLogon para iniciar automaticamente com o Windows. (*Risco: 4*)
22. **Persistência**: altera arquivos .INI ou Registros equivalentes para iniciar automaticamente com o Windows. (*Risco: 4*)
23. **Manutenção**: verifica conectividade com a rede externa. (*Risco: 1*)
24. **Manutenção**: cria um objeto de sincronização do tipo `mutex`. (*Risco: 1*)
25. **Disfarce**: modifica arquivos legítimos para se disfarçar e executar ações suspeitas. (*Risco: 5*)

É importante notar que nenhuma dessas atividades sozinhas garante que o programa em execução é malicioso. Entretanto, quando analisadas sob a ótica da execução de um programa sabidamente malicioso ou quando consideradas em conjunto, pode-se inferir um maior grau de suspeição a tal programa e observá-lo mais atentamente.

Um exemplo de *template* criado seguindo o modelo proposto pode ser visto na Tabela 2. Essa heurística representa o comportamento da persistência, isto é, a capacidade de um programa executar após reinicialização ou desligamento da máquina. Programas maliciosos fazem uso desse comportamento para "sobreviverem" na vítima (em vez de residirem na memória somente quando executados da primeira vez). Pode-se observar que a ativação de uma dada heurística é passível de ocorrer por meio de mais de um padrão para detecção.

Tabela 2. Exemplo de aplicação do modelo heurístico na criação de um detector do comportamento de persistência via modificação de chaves pré-definidas do Registro.

Campo	Tipo
Identificador da heurística	004
Nome da heurística	persistencia_registro
Descrição	modifica o Registro para executar na inicialização do sistema
Risco associado	3
Categoria	persistência
Padrão para detecção	".*\Software\Microsoft\Windows NT\CurrentVersion\Run", ..., ".*\Software\Microsoft\Windows\CurrentVersion\RunOnce", ...

3.3. Algoritmo de Identificação de Comportamento Suspeito

Uma vez criados, os *templates* são utilizados para encontrar instâncias desses comportamentos de acordo com um algoritmo definido. O Algoritmo 1 ilustra os passos efetuados com a finalidade de se identificar comportamentos suspeitos em programas monitorados por análise dinâmica.

4. Testes e Resultados

Os testes foram realizados com 2.959 amostras divididas em 2.394 maliciosas e 565 benignas.rotuladas como maliciosas foram coletadas no decorrer de 2015 e são provenientes do laboratório de análise de *malware* da Bluepex. As amostras rotuladas como benignas incluem programas nativos do sistema operacional Windows, aplicações disponíveis para *download* em *sites* Web e programas do *SysInternals*[3]. Todas os programas foram submetidos para análise dinâmica em ambiente de testes implantado para validação dos *templates* heurísticos com base no sistema *Cuckoo Sandbox* [Guarnieri 2013]. A escolha desse sistema deve-se à facilidade de se customizá-lo para que, dadas as heurísticas implementadas, se verifique se tal comportamento é exibido pelo programa analisado. A configuração do *Cuckoo* foi feita sobre o emulador QEMU [Bellard 2005] com sistema

[3]https://technet.microsoft.com/en-us/sysinternals/bb545021.aspx

Algorithm 1: Identificação de comportamento suspeito

ENTRADA: Traço obtido da monitoração da execução de um programa ou chamadas de API capturadas em tempo real.
SAÍDA: Comportamento encontrado e pontuação de risco associada.
Seja $A = \{a_1, a_2, ..., a_N\}$ o conjunto de ações presentes em um traço de análise dinâmica ou interceptadas por um monitor de chamadas de API.
Seja $H = \{h_1, h_2, ..., h_M\}$ o conjunto de heurísticas definidas que representam ações suspeitas
risco $\leftarrow 0$
comportamentos $\leftarrow$ vetor cujas posições representam cada $h_i \in H$ e são inicializadas com 0
for $a_i \in A$ **do**
    for $h_i \in H$ **do**
        if $h_i \subset a_i$ **then**
            comportamentos$[h_i] \leftarrow 1$
            risco $\leftarrow$ *risco* $+h_i.risco$
return *comportamentos*,*risco*

operacional Windows 7. Cada exemplar permaneceu em execução por no máximo quatro minutos e gerou resultados que passaram por uma etapa de pós-processamento, a fim de se identificar os comportamentos exibidos. Os *templates* heurísticos aplicados aos traços de análise dinâmica produziram os resultados mostrados na Tabela 3.

É interessante notar que as heurísticas são ativadas tanto em *malware* como benignos. Isso já é esperado, pois um programa malicioso nada mais é do que um programa como outro qualquer, porém com intenção de efetivar um "ataque". O que muda é o conjunto de comportamentos exibidos e sua frequência para ambos os tipos de programas, o que fica bem explícito na tabela. As duas primeiras linhas da tabela, que possuem os valores mais altos para ambas as classes de programas, são importantes para a discussão da aplicação de heurísticas comportamentais na detecção de *malware*: nota-se que não se pode afirmar que a criação de um novo processo é uma atitude maliciosa, mas a maioria dos exemplares de *malware* o faz; os programas legítimos, por sua vez, não costumam criar novos processos diferentes dos seus próprios após execução inicial. Da mesma forma, enquanto que a criação de um objeto de sincronização (mutex) não é incomum, uma análise mais detalhada, por exemplo, obtendo-se seu nome, pode revelar uma determinada família de *malware*, pois estes geralmente fazem uso de *mutexes* para evitar reinfecções e *crashes*.

Sete dos 25 comportamentos definidos não foram identificados no conjunto de exemplares utilizado nos testes. Isso pode ser ocasionado por diversos fatores: nenhum programa realmente apresentou tal comportamento, o tempo de execução foi insuficiente, a monitoração é limitada, entre outros. Porém, a ausência de um dado comportamento não afeta a proposta deste trabalho, uma vez que as heurísticas podem ser implementadas em outros sistemas de análise dinâmica ou como um *engine* próprio. Para fins de validação desta proposta, é suficiente mostrar que há possibilidade de se diferenciar programas maliciosos de benignos por meio da detecção de seu comportamento. Assim, calculou-se

Tabela 3. Porcentagem de exemplares maliciosos e benignos que ativaram heurísticas comportamentais. O número entre parênteses nas linhas da coluna "Heurística" representa o comportamento apresentado na lista da Seção 3.2.

Heurística	Malware	Benigno
Subversão do SO (13)	87,3%	15,6%
Manutenção (24)	77,2%	17,5%
Subversão do navegador (16)	44,7%	1,4%
Evasão (7)	19,5%	1,2%
Persistência (20)	6,0%	1,6%
Subversão do SO (12)	4,6%	1,1%
Evasão (4)	4,5%	0,2%
Evasão (5)	2,3%	0,4%
Subversão do navegador (15)	1,6%	0,2%
Evasão (3)	1,6%	0,2%
Subversão do SO (10)	1,6%	0,2%
Manutenção (23)	0,8%	✗
Subversão do SO (8)	0,5%	0,2%
Persistência (22)	0,5%	✗
Evasão (6)	0,5%	✗
Subversão do navegador (17)	0,2%	✗
Subversão do SO (11)	0,1%	0,4%
Subversão do SO (9)	✗	0,2%

a média do valor de risco encontrado nos programas maliciosos e benignos, obtendo-se 5,77 para os primeiros e 0,75 para os últimos. Cabe ressaltar que os programas pertencentes ao *SysInternals* (parte do subconjunto dos benignos) causaram aumento na ativação das heurísticas para este grupo, pois executam atividades privilegiadas (como carregar *drivers*) ou fazem muitas alterações em Registros e no sistema de arquivos que podem ser consideradas "administrativas" (pela própria natureza dessas ferramentas), mas que também são comumente observadas em *malware*.

Conforme mencionado, a aplicação do algoritmo de detecção heurística a cada um dos traços de execução dos programa monitorado foi utilizada para se gerar um vetor binário de comportamentos, no qual os índices representam comportamentos e seus valores ("0" ou "1") a ausência ou presença, respectivamente, do comportamento em questão. A Listagem 1 ilustra um vetor de comportamentos associado a um programa malicioso e outro a um programa benigno[4].

Listagem 1. Exemplos de vetores comportamentais para exemplar malicioso (rótulo 'm') na linha 1 e benigno ('g') na linha 2.

```
1 0 ,0 ,0 ,1 ,0 ,0 ,1 ,0 ,0 ,0 ,1 ,0 ,0 ,1 ,0 ,0 ,0 ,1 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,m
2 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,g
```

Os vetores rotulados servem de entrada para um classificador baseado no Teorema de Bayes, que por sua vez faz uso do conceito de probabilidade condicional: $P(A \mid B)$

[4]O arquivo CSV com os vetores utilizados neste artigo, bem como a lista de MD5 dos exemplares estão disponíveis em: `http://www.inf.ufpr.br/gregio/SBSeg2016/`.

é a probabilidade de que um evento "A" ocorra dado que o evento "B" já ocorreu. É possível derivar o Teorema de Bayes a partir de probabilidade condicional e probabilidade conjunta, chegando à seguinte fórmula [Downey 2012]:

$$P(A \mid B) = \frac{P(A)\,P(B \mid A)}{P(B)}$$

O classificador Bayesiano costuma unir rapidez na geração do modelo com bom nível de precisão, sendo capaz de atuar em casos nos quais há alta dimensionalidade [John and Langley 1995]. Para os testes realizados, usou-se a implementação do algoritmo *Naïve Bayes* disponível no *framework* de mineração de dados WEKA [Hall et al. 2009]. O parâmetro passado para treinamento e teste foi a validação cruzada utilizando 10 pastas, resultando em 88,44% de exemplares corretamente classificados. Consequentemente, 11,56% deles foram incorretamente classificados, em parte devido às já mencionadas ferramentas do *SysInternals*. A Tabela 4 mostra os resultados da classificação de maneira mais detalhada, considerando as duas classes de programas presentes no conjunto de dados.

Tabela 4. Resultados da classificação dos programas em *malware* ou benigno, onde TP corresponde aos verdadeiros-positivos e FP aos falsos-positivos.

TP	FP	Precisão	*Recall*	*F-Measure*	Área ROC	Classe
90,1%	18,4%	95,4%	90,1%	0,927	0,933	*Malware*
81,6%	9,9%	66,0%	81,6%	0,729	0,933	Benigno

A avaliação dos resultados da tabela mostram que, enquanto 90,1% dos programas maliciosos foram corretamente classificados como tal, 18,4% dos programas benignos foram considerados *malware*. Esse resultado pode ter um impacto grande no uso de um sistema de detecção automatizado por usuários. No entanto, deve-se considerar que os vetores usados para classificação foram gerados com base na execução do programa em uma *sandbox*, o que traz diversas implicações (como a taxa de interações com os programas benignos, bem diferente do uso legítimo feito por um ser humano) que podem não se tornar um problema na vida real. Mesmo assim, é um ponto importante a se levar em conta na implementação deste tipo de abordagem para detectores em tempo real, o que está fora do escopo do presente artigo. Se considerarmos exclusivamente as habilidades de detecção de programas maliciosos, a qualidade (calculada através da *F-Measure*) do classificador foi boa: 92,7%. Esse resultado também é corroborado pela área sob a curva ROC, onde quanto mais próximo de 1 for o valor, mais exatidão se tem.

5. Conclusão

Neste artigo apresentou-se o estado da arte em detecção de *malware* baseada em heurísticas comportamentais, discutindo-se as limitações das abordagens existentes na literatura. Foi proposto um modelo de detecção heurística dependente de comportamentos de alto nível, os quais descrevem ações feitas por um programa durante o processo de infecção da vítima. Foram definidos comportamentos comumente exibidos por *malware* em execução, aos quais foram atribuídos valores de risco. O cálculo do risco associado a programas maliciosos e legítimos mostrou que há uma diferença considerável entre ambos, podendo

servir no futuro para o estabelecimento de limiares dinâmicos que auxiliem na emissão de alertas quanto a execução de programas suspeitos. Introduziu-se também um algoritmo que faz uso de *templates* heurísticos a fim de identificar comportamentos pré-definidos durante a execução de programas. A avaliação de programas reais, rotulados como maliciosos ou benignos, serviu para a geração de vetores binários explicitando a presença ou ausência da observação de cada um dos comportamentos definidos. Tais vetores foram utilizados como entrada em um algoritmo de classificação Bayesiana, que obteve taxa de acertos da ordem de 88,4% no conjunto de dados disponível (exemplares coletados em 2015). Os resultados mostram que é viável aplicar heurísticas comportamentais de alto nível para a detecção de *malware* em tempo real, seja em um *engine* de monitoração *stand-alone* ou como complementação de antivírus por assinaturas. Trabalhos futuros incluem o desenvolvimento de um protótipo que atue em tempo real, de forma a se medir o impacto no desempenho—atenuado pelo fato de que, uma vez que o modelo está treinado, a classificação Bayesiana é praticamente instantânea e que os antivírus já realizam a monitoração das chamadas de sistema e/ou de APIs em nível privilegiado, bastando-se implementar algum tipo de dicionário de comportamentos que gere os vetores em memória—e a consideração de outros "níveis" de avaliação de comportamento, como nomes de arquivos, valores de chaves de Registro e objetos do tipo `mutex` pré-definidos, bem como o encadeamento de comportamentos. Vale ressaltar que a abordagem proposta, embora simples e efetiva, necessita ser trabalhada de forma a minimizar drasticamente a quantidade de falsos-positivos de *malware*, isto é, a detecção incorreta de programas benignos como maliciosos, a fim de que possa ser utilizada em modo *stand-alone* e sem a supervisão de uma terceira parte (usuário, mecanismo de segurança ou profissional).

Referências

Adkins, F., Jones, L., Carlisle, M., and Upchurch, J. (2013). Heuristic malware detection via basic block comparison. In *Malicious and Unwanted Software: "The Americas"(MALWARE), 2013 8th International Conference on*, pages 11–18.

Ahmadi, M., Sami, A., Rahimi, H., and Yadegari, B. (2013). Feature. *Computer Fraud & Security*, 2013(8):11–19.

Alazab, M. (2015). Profiling and classifying the behavior of malicious codes. *J. Syst. Softw.*, 100(C):91–102.

Bazrafshan, Z., Hashemi, H., Fard, S. M. H., and Hamzeh, A. (2013). A survey on heuristic malware detection techniques. In *Information and Knowledge Technology (IKT), 2013 5th Conference on*, pages 113–120.

Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA. USENIX Association.

Christodorescu, M., Jha, S., Seshia, S. A., Song, D., and Bryant, R. E. (2005). Semantics-aware malware detection. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, SP '05, pages 32–46, Washington, DC, USA. IEEE Computer Society.

Downey, A. B. (2012). *Think Bayes: Bayesian Statistics Made Simple*. Green Tea Press.

Fan, Y., Ye, Y., and Chen, L. (2016). Malicious sequential pattern mining for automatic malware detection. *Expert Syst. Appl.*, 52(C):16–25.

Feng, Z., Xiong, S., Cao, D., Deng, X., Wang, X., Yang, Y., Zhou, X., Huang, Y., and Wu, G. (2015). Hrs: A hybrid framework for malware detection. In *Proceedings of the 2015 ACM International Workshop on International Workshop on Security and Privacy Analytics*, IWSPA '15, pages 19–26, New York, NY, USA. ACM.

Grégio, A., Bonacin, R., de Marchi, A. C., Nabuco, O. F., and de Geus, P. L. (2016). An ontology of suspicious software behavior. *Applied Ontology*, 11(1):29–49.

Griffin, K., Schneider, S., Hu, X., and Chiueh, T.-C. (2009). Automatic generation of string signatures for malware detection. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, RAID '09, pages 101–120, Berlin, Heidelberg. Springer-Verlag.

Grégio, A. R. A., Afonso, V. M., Filho, D. S. F., Geus, P. L. d., and Jino, M. (2015). Toward a taxonomy of malware behaviors. *The Computer Journal*, 58(10):2758–2777.

Guarnieri, C. (2013). Cuckoo sandbox. `http://www.cuckoosandbox.org/`. Acesso em junho/2016.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. (2009). The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18.

Jacob, G., Debar, H., and Filiol, E. (2008). Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology*, 4(3):251–266.

John, G. H. and Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In *11^{th} Conference on Uncertainty in Artificial Intelligence*, pages 338–345.

Khodamoradi, P., Fazlali, M., Mardukhi, F., and Nosrati, M. (2015). Heuristic metamorphic malware detection based on statistics of assembly instructions using classification algorithms. In *Computer Architecture and Digital Systems (CADS), 2015 18th CSI International Symposium on*, pages 1–6.

Prelipcean, D. B., Popescu, A. S., and Gavrilut, D. T. (2015). Improving malware detection response time with behavior-based statistical analysis techniques. In *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 232–239.

Treadwell, S. and Zhou, M. (2009). A heuristic approach for detection of obfuscated malware. In *Proceedings of the 2009 IEEE International Conference on Intelligence and Security Informatics*, ISI'09, pages 291–299, Piscataway, NJ, USA. IEEE Press.

Ye, Y., Wang, D., Li, T., and Ye, D. (2007). Imds: Intelligent malware detection system. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 1043–1047, New York, NY, USA. ACM.

Yin, H., Song, D., Egele, M., Kruegel, C., and Kirda, E. (2007). Panorama: Capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 116–127, New York, NY, USA. ACM.

A comparison of encryption tools for disk data storage from digital forensics point of view

Vitor Hugo Galhardo Moia, Marco Aurélio A. Henriques

[1]School of Electrical and Computer Engineering (FEEC)
University of Campinas (UNICAMP)
Campinas, SP, Brasil 13083-852

vhgmoia,marco@dca.fee.unicamp.br

Abstract. *A closer look from a digital forensics point of view may help us to find and understand vulnerabilities in cryptographic tools to determine which ones are the safest and provide the best features. In this paper, we analyze different ways of using cryptography to protect data on hard drives, discuss some vulnerabilities and present particular features of disk encryption tools. Then we analyze the security of some tools according to a set of criteria and evaluate the level of expertise required to use them. We conclude that the encryption subject impacts the scope of tools' vulnerabilities and full disk encryption is the safest and easiest option. Moreover, we highlight some free and easy to use tools that can protect users data properly if basic precautions are taken.*

1. Introduction

Digital Forensics aims to solve crimes committed with computers and electronic devices where evidences may reside on and can be used against criminals in court. The development of anti-forensic (AF) tools [Kessler 2007] with the purpose of thwarting investigations has increased in the past years. Cryptography, the most efficient AF tool, is a strong measure to protect users data against unauthorized access. In the past, when adversaries used encryption to protect their data, forensics experts could recover useful information to assist in investigations, due to some traces left in the operating system, known vulnerabilities in the tools used, weak choice of passwords by users etc.

However, the development of strong encryption tools and their integration into the operating systems have introduced new challenges for forensic analysis. Some methods encrypt the whole disk, even the operating system files and unallocated sectors on disks, prompting a password when the device is initialized before the machine boots, preventing unauthorized access to data. Normally, the only and not viable way to get users data in such case is by means of brute-force or dictionary attacks on the password. This way, the use of encryption tools became one of the major challenges of current digital forensic investigators, and its use is increasing even more since these tools are becoming easy and simple to use.

Nonetheless, not all hope is lost for forensic examiners when encryption is used. A lot of tools do not provide full disk encryption, and traces of useful information and sometimes the decryption keys may be found in the device. Also, there could be known vulnerabilities that can be exploited to gain access to data or, in some cases, the assistance of system administrators or IT. personnel in providing the recovery keys of the system.

In this scenario, if the forensics examiners can recover information, adversaries can too. Better ways to protect users privacy are necessary. However, with so many tools available, how can users distinguish a secure from an insecure tool? Which ones are in fact able to protect their privacy? Which features are the most appropriate? A closer look at these tools from a forensic perspective allow us to identify vulnerabilities and understand the provided features, in order to answer this sort of questions. In this paper, we analyze different ways of using cryptography to protect data stored on hard drivers, discuss some vulnerabilities from a forensic point of view and present particular features of disk encryption tools. Then, we compare some encryption tools according to a set of criteria to analyze security aspects, followed by an evaluation with respect to the minimum level of expertise required to use different features of the tools. We expect that our evaluation will help users choosing the best tools according to their needs.

2. Encryption methods for data at rest

The data encryption processes can be classified according to its subject, such as File Encryption (FE), Virtual Disk Encryption (VDE) and Full Disk Encryption (FDE) [C. 2010]. In this section, we will explain each one of these methods and give examples of some related tools. We will also present and discuss some vulnerabilities in these three methods and particular features of FDE tools.

2.1. Classification of encryption methods

1. **File Encryption (FE):** Method where individual files are encrypted, such as images, audio, videos, documents, personal backups, and so on. Examples of encryption tools available are: GNU Privacy Guard (GPG) [GNU 1999], AxCrypt [AxCrypt 2001], 7-Zip [Pavlov 1999], among others.
2. **Virtual Disk Encryption (VDE):** In this mode users can create an encrypted container, which is a single file responsible for holding as many files and/or folders as they wish. Getting access to its content requires a passphrase to unlock it, which after being correctly supplied, mounts a virtual disk of the container where users can easily store and read data from it. This is a type of on-the-fly encryption, which means that data is automatically encrypted or decrypted as it is loaded in the container or saved elsewhere. Examples of VED tools are: GNU Privacy Guard (GPG), VeraCrypt [IDRIX 2014], CipherShed [PCM 2014], R-Crypto [R-Tools 2007], The Linux Unified Key Setup (LUKS) [Guardian 2014].
3. **Full Disk Encryption (FDE):** This is a more sophisticated level of encryption, where the whole disk, including the operating system and all its files along with users data is encrypted. Unallocated spaces of the hard drive are also encrypted. Usually, before booting a encrypted system there is an interface to collect the right decryption key, preventing access of those who do not posses the credentials. Examples of such tools are: Bitlocker [Microsoft 2006], VeraCrypt, CipherShed, DiskCryptor [NTLDR 2007], FileVault2 [Apple 2011], The Linux Unified Key Setup (LUKS), among others.

2.2. Vulnerabilities to be exploited

Usually, when encryption is used to protect data, there is not much what forensic examiners can do to break keys and recover useful information. Current cryptographic tools

are well designed and reviewed, presenting no vulnerabilities (at least not known by most people). However, depending on the type of encryption applied, some methods can get users' cryptographic keys or obtain some useful information. Also, there are other ways to get users data without attacking the algorithms.

When **File Encryption (FE)** and **Virtual Disk Encryption (VDE)** tools are employed, there is a broader scope of attacks that can succeed in some cases, allowing forensic examiners recover useful information despite the use of the strongest algorithms. In such cases, Casey E. and Stellatos G.J. [Casey and Stellatos 2008] point out that plaintext versions of some encrypted files may be obtained in temporary files, in the process of creating or editing documents, or even in a spool file created while printing data. Besides, deleted files that have not been overwritten on hard drivers are also a possible source of useful information. The authors highlight that keyword searching could be an effective mechanism for finding relevant information in plaintext, looking for dates, phrases, and other known characteristics. As a matter of fact, forensic examiners may also be able to recover encryption keys from passphrases stored on hard drives or removable media (thumb drivers, external hard drivers etc), since many users prefer to write them down to avoid forgetting them.

Mrdovic S. and Huseinovic [Mrdovic and Huseinovic 2011] indicates that encryption keys can also be recovered from hibernation files, responsible for storing the system memory content when the computer hibernates (or enters in a power-saving mode). The RAM may hold sensitive information from a running computer, such as decryption keys, passphrases or other relevant information, which will be written in these files. Analyzing them allows forensic examiners getting the keys to decrypt data or important information to a case. The same concept applies to the memory dump method, analyzed by Balogh S. and Pondelík M. [Balogh and Pondelik 2011], where the examiners obtain a dump of the memory content to work with in the search of useful data.

Another possible way to look for traces of sensitive data are through paging files or bad sectors on disks [IDRIX 2015]. The former, also called swap file, is related to data that do not fit in memory and are stored on disk unencrypted. The latter refers to bad sectors of a disk that once may have stored useful information and now are marked as bad sectors due to problems in the writing/reading processes. In this case, a new sector is allocated to store the content of the bad one. This means that data in these sectors can not be overwritten, causing security implications due the old data that still remains.

In cases where **FDE** tools are employed, these approaches described are not suitable for recovering useful information. In such cases, the hard drive is fully encrypted preventing forensic examiners to access any operating system files. This way, new methods for overcoming this technology are necessary. However, it is important to highlight that as long as the computer is on and the passphrase was supplied, forensic examiners can access everything, such as personal data, memory content, operating system files etc, since all data will be decrypted. FDE tools usually decrypt the cryptography keys once the right credentials are supplied and keep them on memory to decrypt other content on the hard drives as necessary. On the other hand, if the computer is off, recovering useful information becomes harder and, sometimes, an impossible task. One way is through a brute-force or dictionary attack in order to guess users' passphrase. In this scenario, the adversary needs to locate the sector(s) storing the decryption keys on the hard drive, ex-

tract them and perform the attack, trying lots of combinations until it matches. For poorly chosen passphrases, this task should not take a long time. But if users choose a strong passphrase, the attack fails since it would require a large amount of time. Zhang L. et al. [Zhang et al. 2014] show how to proceed in such attack, where they studied the TrueCrypt system (version 7.1a), a cryptographic tool precursor of VeraCrypt and CipherShed. They also cite the possibility of using a GPU (Graphics Processing Unit) to accelerate the process. A similar work is done by Kornblum J. D. [Kornblum 2009], but the author explore the BitLocker system, providing useful information for forensic examiners while dealing with this encryption tool. Other way of attacking some FDE systems is by the recovery password feature available in some softwares [Zhang et al. 2014], [Kornblum 2009]. In case of BitLocker, users can save the recovery key in their Microsoft account. If so, this could be a way of attacking the system. Adversaries first steal user's credentials and then have access to the recovery key.

Another vulnerability to be exploited independent of the encryption subject, is the users behaviour. In general, users do not take the necessary precautions regarding the right management of the cryptography keys neither create a strong passphrase to protect their keys. They usually chose easy to remember passwords, which are not large enough and encompasses common words from the dictionary, names of places, celebrities or family, last names, dates etc, vulnerable to brute-force or dictionary attacks. This may be a consequence of the fear and impact of losing their passwords, since it prevents users from accessing their data or using their computers (specially when FDE tools are used). Social engineering is another kind of attack involving users that most tools are vulnerable.

Weak implementations of the cryptographic algorithms are also a vulnerability to be exploited. With the source code released to the community, this risk tends to be mitigated since there will be many eyes evaluating the code. However, developers are always subject to making mistakes, as the ones in the Drown Attack [Aviram et al. 2016] and the Heartbleed Bug [Codenomicon 2014]. For the non-open source applications, the risks are even greater with addition of doubts about backdoors implanted to compromise the whole security by those who know them.

In some contexts, the forensic experts may have support from a server administrator or the I.T. team from a company in which some of its employees are under investigation. Such support could be supplying a recovery password, for example, allowing the decryption of the whole data.

Finally, Halderman J. A. et al. [Halderman et al. 2009] describe another alternative to attack encrypted systems, using a technique called Cold-Boot Attack. It is similar to the memory dump technique but with the difference that it uses cooling techniques to retain information on memory after the computer is turned off. In some cases, this technique could be a solution to recover the decryption keys of FDE systems.

Table 1 summarizes the vulnerabilities according to encryption subject, where the mark (✓) represents the presence of the vulnerability in that mode of encryption.

2.3. Particular features of FDE tools

FDE tools present some particular features due to their requirements that need to be understood, to allow users making a good choice of a tool that fits their needs. In the following subsections, we will present and explain these features.

Table 1. Vulnerabilities according to the encryption subject

Encryption Subject	Vulnerabilities									
	Plaintext version of old files	Hibernation files	Memory Dump	Paging files	Bad sectors on disks	Brute-force or Dictionary Attacks	Social Engineering	Weak implementation	Support from others	Cold Boot Attack
FE	✓	✓	✓	✓	✓	✓	✓	✓		✓
VDE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
FDE					✓	✓	✓	✓	✓	✓

2.3.1. The XTS mode for disk Encryption

When it comes to data encryption on storage devices, the XTS (**X**EX **T**weakable Block Cipher with Ciphertext **S**tealing) mode is the one recommended by NIST [Dworkin 2010]. It was designed specifically for encryption on storage devices with fixed length data units, and it is a modified version of the XEX (XOR Encrypt XOR) tweakable block cipher, with the difference of using two independent keys instead of one.

This mode of operation takes advantage of the hard drives structure, where the disks are partitioned into circular paths (the tracks) and these are divided into fixed-sized (typically 512 bytes) logical sectors, where information is stored. The encrypted data is placed in a sector including the cryptographic key and any necessary metadata, which usually correspondes to the sector number and block number in that sector. This is how the XTS stores the encrypted data, protecting better than other modes against ciphertext manipulation and cut-and-paste attacks [Martin 2010].

Martin [Martin 2010] explains how this mode of operation works. Let us consider a block cipher E encrypting a message M with a key K, producing a ciphertext C ($C = E_K(M)$). A tweakable block cipher operates on M, K, and a tweak T using E to result in C ($C = E_K(T, M)$). The tweak is similar to an initialization vector (IV) used in standard operation modes as CBC (Cipher Block Chaining), but it does not require to be random or to remain secret. Its purpose is to provide variability in the ciphertext. Also, a tweak block cipher has to remain secure even if the adversary knows the tweak.

In the XTS mode there are two tweaks, one corresponding to the sector number (j) and the other to the block number in that sector (i). It calculates a ciphertext C from a message M and the two keys K_1 (primary key) and K_2 (secondary key) as follows:

$$E_{K_1,K_2}(i, j, M) = E_{K1}(M \oplus \Delta) \oplus \Delta,$$

where $\Delta = \alpha^{\mathrm{i}} \bigotimes E_{K2}(j)$, $\bigotimes$ is polynomial multiplication modulo $x^{128} + x^7 + x^2 + 1$ and α a constant primitive element in $GF(2^{128})$. This process is illustrated in Fig. 1, where a message M is broken in n equal size blocks and encrypted with K_1 and K_2, resulting in the ciphertexts C_1 to C_n. The value of i goes through 1 to m, where m is the number of blocks that fits in a single sector. The decryption process can be seen in Fig. 2.

The use of XTS as an operation mode for disk encryption has some advantages besides being secure against common attacks in this context. The first one is regarding error propagation when a ciphertext block gets damaged, affecting the decryption of that particular block only. Another advantage is the use of parallelism in the processes of encryption and decryption to accelerate the process.

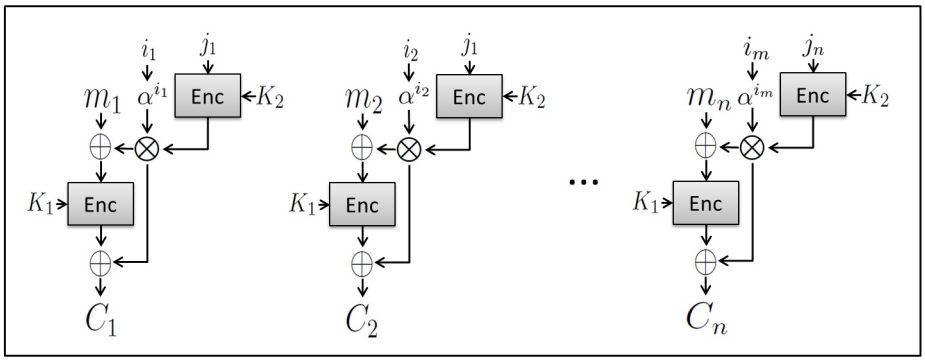


Figure 1. Encryption process in XTS mode

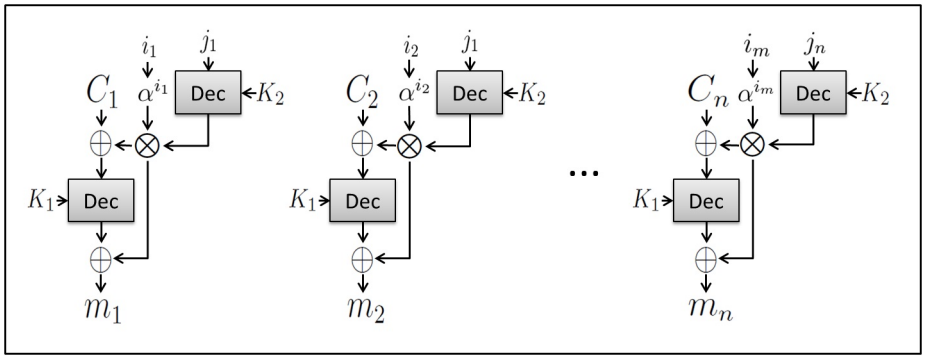


Figure 2. Decryption process in XTS mode

2.3.2. Plausible Deniability

As highlighted by Lowman S. [Lowman 2010], plausible deniability is a feature available in some encryption tools which allows users to deny the use of cryptography. They may blame the presence of ciphertext on others, saying that a malicious software infected their machines and encrypted everything or that no encrypted data exists at all. Without the proper encryption key, it is difficult for an adversary to prove that plaintext data exists. Lowman S. says that "it is difficult to determine whether a disk contains encrypted versus random data".

When plausible deniability is employed, users create an encrypted message consisting of two keys and two messages, one representing the real information and the other a fake one. In case of coercion, users can decrypt the latter message with the second key, revealing the fake data. That way, the real information is protected and the adversary convinced that users have no futher information. Without the knowledge of the right key, one cannot prove that more data exist [Canetti et al. 1997].

2.3.3. Use of Trusted Platform Module

A Trusted Platform Module (TPM) is a secure cryptoprocessor embedded in the motherboard with the purpose of authenticating hardware devices. Encryption tools can store cryptographic keys inside the TPM. When the system is booting, there is a process that verifies the integrity of the hardware and operating system in that machine. In case

the verification succeeds, the encryption keys are used by the cryptoprocessor and the whole disk is decrypted, allowing the booting process to continue. However, using this technology implies tying the volume to that particular device, making decryption difficult, if not impossible, to suceed if the volume is removed and placed in another device [Kornblum 2009]. In case the TPM or motherboard have problems, users will not be able to access their data, unless they have a recovery key. This is one of the disadvantages of using the TPM, since it creates a single point of failure.

3. Encryption Tools comparison

In this section we present a comparison between some encryption tools available nowadays. We selected a small group of well known tools that implement the three cryptography usages in a disk storage system (FE, VDE and FDE) and analyzed them according to some criteria in order to show users the better tools according to their needs. We will also present an analysis about the level of expertise required from users in the tools used, evaluating the minimum amount one needs in order to make good use of each processes analyzed, according to our expertise and experience with these tools.

3.1. Security aspects

In the following items we will present the criteria used to evaluate the encryption tools and a brief explanation of each one.

- **Algorithms:** Represents the algorithm(s) available for encrypting data. More than one algorithm may be available for users' choice, and some tools allow users to use them in cascade, for example, where data is encrypted more than once.
- **Key recovery:** This feature allows users to recover their keys in case they lose or forget their password. This could be a good way to avoid the burden of losing all data in such case. However, adversaries can use this artifice to break the encryption. Also, some tools allow system administrators to have access to the recovery keys, which in some cases could represent as a threat to users.
- **Algorithm operation mode:** It is already known that some operation modes are more secure than others. In our analysis, we are considering tools that are used for encrypting data on hard drives, which require a different treatment, for example, the reuse of IV, forbidden for some ciphers. For this reason, some encryption modes should be avoided, as the most common cipher CBC and other well known stream ciphers (CTR, CFB, OFB), due to problems pointed out by Fruhwirth C. [Fruhwirth 2005].
- **Open source code:** According to Kerckhoff's principle, only the key should constitute the secret information. This stands in contrast with the concept of 'security by obscurity', where the security is achieved by keeping the algorithm secret. Having a code open to the community can bring several advantages to the software. For example, the public scrutiny, where the code will receive an extensive study by the community and will be tested against several attacks. Also, if the security relies on the algorithm secrecy, methods as reverse engineering represent a big threat and can compromise the software. Thus, this characteristic is desired in a secure tool and helps eliminating vulnerabilities.
- **Encryption subject:** This feature refers to the classification of the tool regarding its encryption subject, as discussed previously: File Encryption (FE), Virtual Disk

Encryption (VDE) and Full Disk Encryption (FDE). It is important to highlight that some tools can operate in more than one mode.

- **Multi-factor authentication:** This feature is related to the use of other factors in addition to the passphrase to authenticate users and decrypt their data. For example, VeraCrypt uses the concept of keyfiles, which, besides user's password, demands the presence of a certain file to derive a key and decrypt users data.
- **Operating system:** This characteristic represents all the available choices of operating systems for using the encryption tool.
- **Portable software:** This option allows users to execute a software without installing it on the computer. Gupta D. and Mehtre B. M [Gupta and Mehtre 2013] highlight that usually this kind of applications does not leave files or any data on host systems. Also, they do not use the registry or configuration files, and can be stored on removable storage devices. This means that the traces left behind by their use will be limited, making forensics work harder.
- **Plausible deniability:** Users can use this technique to avoid adversaries know about the use of cryptography to protect information.
- **File Attributes Encryption:** Encrypting data attributes, such as filename, creation and last modifications dates, ownership, among others, is also important feature for preserving users' privacy. Usually, a file is named according to its content in order to make easier its identification later. However, this could help an adversary in several ways, for example, identifying the most valuable files that worth stealing and focusing the attack on them.
- **Security level:** This criteria evaluates the size of the keys available in the algorithms. The larger the keys, the bigger its search space will be, making brute-force or dictionary attacks harder to succeed.
- **Hidden containers:** Some tools offer a similar feature to plausible deniability, with the difference of allowing users to create a hidden area in the hard drive (instead of a fake content) which will be revealed only with a specific key.
- **TPM:** Use of a Trusted Platform Module to manage the cryptographic keys and validate the hardware. Ties the volume to the device.
- **Key Stretching:** Use of an algorithm for key stretching and number of iterations performed in the process. The purpose of this feature is to slow down attacks such as brute-force or dictionary, in which the adversary tries to figure out the right password and steal users' keys. A small number of iterations make no effect on adversaries, and a big one can impact on the application usability and annoy users. It is important to find a balance between these two options.

3.2. Levels of expertise

We also made an analysis of the tools according to the level of expertise required from users to accomplish certain tasks while using them. We made an estimation based on our point of view and experience using them. We created a categorization level to classify users based on the work of the National Institutes of Health [OHR 2009], presented next.

- **Level 0** - No knowledge on data security field at all.
- **Level 1 - Beginner:** Someone with a common knowledge or an understanding of basic techniques and concepts of data security. Able to perform simple tasks on computers, as managing files and folders, installing basic programs etc. However, they often need some help to deal with more complex issues.

- **Level 2 - Intermediate:** Users with a better and broader understanding of cryptography and computer systems. They have knowledge on some security algorithms, related tools, and are able to complete simple tasks by themselves. Some assistance is needed from time to time.
- **Level 3 - Advanced:** Someone with a degree in areas related to computers, with a solid knowledge about security and operating systems. He/She can perform tasks associated to the fields without assistance and will be recognized as the one to ask questions to. He/She is able to implement tools and algorithms on the field.
- **Level 4 - Expert:** People with a degree or deep knowledge on security, able to implement, develop and evaluate algorithms and tools to find out vulnerabilities.

We considered the following items for evaluating the tools during their use:

- **Installation process:** We installed each tool with default setting options (when possible), which is usually the simplest way. We evaluate how hard is for users to install them based on their knowledge, choosing between the options, and so on.
- **Configuration:** Once the tools are installed, some of them need to be configured in order to encrypt a disk partition or the whole disk. We evaluate how hard is for users perform this task.
- **Encryption/ Decryption:** Some tools require an interface to allow users to encrypt/decrypt their data, while others do it automatically. In the latter case, this operation is not considered at all. We evaluated these processes considering how intuitive and simple they are.
- **Data manipulation:** In this item, we evaluated the process of authentication and data access, considering how intuitive and simple it is.
- **Changing default settings:** We also evaluated the process of changing the default configuration. For example, some tools allow users to use multiple algorithms (AES, 3DES, Twofish, Blowfish etc), select different key sizes (128, 192, 256 bits), and modes of operation (ECB, CBC, XTS). Users are required to change the configuration manually. We evaluate how hard is for them to perform this task, or, for example, change their passphrase.
- **Enabling additional features:** This item evaluated how hard is for users to enable certain features available in the tools, such as the second factor of authentication, use of hidden containers etc.
- **Uninstallation process:** This is the last item evaluated. We took into consideration the removal process, considering how hard is to decrypted all users' data before removing the tool.

Table 2. Comparison of encryption tools regarding to security aspects

Tools	Algorithms	Key recovery	Alg. operation modes	Open source	Encryption subjects	Multi factor authentication	Operating systems	Portable software	Plausible deniability	File Attributes Encryption	Security Levels	Hidden containers	TPM use	Key Stretching
AxCrypt	AES / RSA	X	CTR	✓	FE	X	Windows	✓	X	X	128 or 256 / 4096 bits	X	X	PBKDF2 - 10000 iterations or more
GnuPG	AES*, Cast5, 3DES, IDEA, Blowfish, TwoFish, Camellia / RSA *, DSA	X	CFB	✓	FE	X	Windows, Linux and Mac OS	✓	X	X	128 or 256 / 1024-4096 bits	X	X	X
7-Zip	AES	X	CBC	✓	FE	X	Windows, Linux and Mac OS	✓	X	Only filenames	256 bits	X	X	SHA-256 - 262144 iterations
R-Crypto	AES, DES, 3DES	X	?	X	VDE	X	Windows	X	X	✓	56-256 bits	X	X	X
DiskCryptor	AES, Serpent, Twofish (and combinations)	X	XTS	✓	VDE and FDE	✓	Windows	X	✓***	✓	256 bits	X	X	PBKDF2 - 1000 iterations
LUKS	AES*, Twofish, Serpent, Cast5, Cast6	✓**	*XTS, ECB, CBC (plain or ESSIV)	✓	VDE and FDE	✓	Linux	X	X	✓	256 bits	X	X	PBKDF2 - 1000 iterations
VeraCrypt	AES, Serpent, Twofish (and combinations)	X	XTS	✓	VDE and FDE	✓	Windows, Linux and Mac OS	✓	✓	✓	256 bit	✓	X	PBKDF2 - X iterations****
CipherShed	AES, Serpent, Twofish (and combinations)	X	XTS	✓	VDE and FDE	✓	Windows, Linux and Mac OS	✓	✓	✓	256 bits	✓	X	PBKDF2 - (1000 - 2000) iterations
BitLocker	AES	✓	CBC modified	X	FDE	✓	Windows	X	X	✓	128 or 256 bits	X	✓	X
FileVault 2	AES	✓	XTS	X	FDE	X	Mac OS	X	X	✓	128 bits	X	X	PBKDF2 - 41000 iterations

Observations: The mark (✓) represent the tools that implement the feature, while the X the ones that do not. The interrogation point (?) is used to mark the features that we could not find information related to their use or not. All options marked with the special character * are default choices. The options marked with ** allows users to add additional passphrases or keyfiles for the authentication. The ones with *** do not support plausible deniability but allow users to install the boot-loader on an external device, being possible to claim that the hard drive was wiped clean since there is no boot-loader neither identifiable information proving that it has been encrypted. The mark **** represents the number of iterations of VeraCrypt, consisting in two ways: system partition, with 200000, 327661 or PIM x 2048 iterations, and containers, with 655331, 500000 or 15000+(PIMx1000) iterations, where PIM (Personal Iterations Multiplier) is a field for users having more control over the number of iterations.

3.3. Comparisons

We choose a set of tools according to works that points out the best ones to encrypt data [Henry 2015] [Manes 2015] [Sharma 2016], considering the operating system (Macintosh, Windows and Linux) and the subject of encryption (FE, VDE and FDE). The tools chosen are still maintained and available for free use. Table 2 presents the comparison, evaluating security aspects. Another comparison evaluated the level of expertise required to use the tools' different features (Table 3).

Table 3. Minimum level of expertise required for different features of the tools

Tools	Installation	Data manipulation	Configuration	Encryption/ Decryption	Changing default setting	Enabling additional features	Uninstallation
AxCrypt	1	2	-	2	1	2	1
GnuPG (Windows version)	1	2	-	2	1	-	1
GnuPG (Linux version)	-	2	2	3	3	-	2
7-Zip (Windows version)	1	2	-	2	-	-	1
R-Crypto	1	1	2	-	2	-	1
DiskCryptor	1	1	2	-	1	1	1
LUKS (VDE)	2	1	1	-	2	3	2
LUKS (FDE)	1	1	1	-	2	3	2
VeraCrypt (Windows version)	1	1	2	-	1	1	1
CipherShed (Windows version)	1	1	2	-	1	1	1
BitLocker	-	1	2	-	1	1	1
FileVault 2	1	1	1	-	1	-	1

4. Discussion

Our security analysis shows users how to choose the tools that best fit their needs. FDE method is the least vulnerable encryption mode, but this is true only when computers are off. Once users turn on their devices and enter their passwords, all data will be decrypted.

The analysis allowed us to observe some good trends. First, most VDE and FDE tools adopt the XTS encryption mode, recommended for disk data storage with no vulnerabilities known so far. Besides that, we can see that 70% of the tools are open source, a tendency we hope will increase since it is a necessary requirement with respect to secure software. The file attributes encryption is also a matter taken into consideration and available in all VDE and FDE tools. The maximum security level is also available for all tools except FileVault 2, which works with AES 128 bits keys.

However, there are some security aspects to improve. The first one is related to the multi-factor authentication, offered by only 50% of the tools. As it can minimize the impacts of choosing weak passwords, it adds an extra protection layer to the system, and should be more explored. Also, only half of tools have a portable version, which besides being easier to run (does not require installation), does not leave traces in the machine. This and some other features as plausible deniability (30%), hidden containers (20%) and TPM (10%), also need to be more explored, since they offer additional security services that should at least be available for users. Also, we point out that more FE tools should

adopt the file attributes encryption, because only one tool of this kind use it (but protecting only filenames). The key stretching algorithm is another point for improvement. Although 70% of the tools adopt this feature, 30% use it ineffectively (small number of iterations). Besides, there are new algorithms such as Argon2, Lyra2 etc., that could better protect users since the PBKDF is becoming obsolete by these password hashing algorithms.

The key recovery feature is a point to be explored by adversaries, being addressed by most tools (70%). We noticed that BitLocker enables users to store the recovery key in their Microsoft account. This can reduce the security, since users tend to create shorter and easier to remember passwords for this kind of account. This way, adversaries can focus on breaking users' accounts to get the recovery keys. However, for some users this could release them from the burden of losing their keys.

From our analysis and criteria, the most secure tool is VeraCrypt (satisfies all criteria except the use of TPM). CipherShed has a similar result, but its key stretching algorithm has a smaller number of iterations. On the other hand, the least secure one is R-Cript. However, we do emphasize that even when the best tools are used, it is necessary that users take some precautions, such as using strong passwords (upper and lower case letters, numbers, special characters etc, with a long length - 20 characters or more are recommended) and managing them appropriately. Another precaution is related to malicious applications on the device. None of the tools analyzed was designed to protect users data in a hostile environment, and in such cases, the whole security could be compromised. This threat is not in the scope of this paper and could be addressed in a future work.

In the analysis of the minimum level of expertise needed to use the tools, we see that they are becoming easier to use. What once was a restricted tool for military and experts only, has now reached average users, allowing them to enjoy the benefits of privacy with little effort and no costs. We have observed that tools designed for Linux systems are more complex and demand more effort or knowledge from users, as the case of GnuPG and LUKS. For Macintosh and Windows, we have easier options, as FileVault 2, 7-Zip, and BitLocker, with low level of expertise required. This statement is sustained by the comparison of GnuPG in Windows and Linux version, where in the former we have a lower required level of expertise in most activities. In our analysis, FileVault 2 is the easiest tool compared to the others, while GnuPG (Linux) is the hardest to use.

It is important to emphasize that some tools, once configured, do not require extra effort to encrypt data. Most VDE or FDE tools work in this way, requiring only that users store their files in a specific place (container or partition) or anywhere in the current operating system to encrypt them. Cryptography has little impact on users, working in the background. On the other hand, most FE tools require the use of an interface to perform the encryption. Also, some tools are already installed in the operating system (BitLocker in specific versions of Windows and GnuPG for Linux), making them easier to use.

5. Conclusion

With so many encryption tools available today, due to the wide spread of simple and easy to use cryptographic tools, users face a challenge of choosing the ones that best protect their data. An analysis of these tools from a forensics point of view was done with the purpose of pointing out strengths and weakness to help users differentiate them. We presented different ways of using cryptography to protect data on the disk, pointing

out some vulnerabilities. We showed that FE and VDE methods have a broader scope of attacks in which one can get useful information about users. On the other hand, the FDE mode is the safest one as long as the computer is off. Only those who know the password can decrypt data and access the system. We also presented particular features of disk encryption tools and evaluated a set of them regarding security aspects, showing important features that can help users to choose the tools that best fit their needs. The other evaluation was related to the minimum level of expertise required to use different features during tool operation. We concluded that there are free and easy to use tools that can protect users data when basic precautions are taken. As future work, we intend to extend the number of tools and items analyzed, evaluating criteria as the whole key management process, the PRNG (Pseudo Random Number Generator) algorithms and the performance in each one of the tools.

Acknowledgment

The authors would like to thank the FORTE project and CAPES (23038.007604/2014-69) for the partial financial support.

References

Apple (2011). Use FileVault to encrypt the startup disk on your Mac. `https://support.apple.com/en-us/HT204837`. Accessed 2016 jun 07.

Aviram, N., Schinzel, et al. (2016). Drown: Breaking TLS using SSLv2. `https://drownattack.com/drown-attack-paper.pdf`. Accessed 2016 jun 07.

AxCrypt (2001). AxCrypt. `http://www.axcrypt.net/`. Accessed 2016 jun 07.

Balogh, Š. and Pondelik, M. (2011). Capturing encryption keys for digital analysis. In *IDAACS, 2011 IEEE 6th International Conf. on*, volume 2, pages 759–763. IEEE.

C., S. (2010). Laptop/table encryption. `https://answers.uchicago.edu/page.php?id=15736`. Accessed 2016 May 09.

Canetti, R., Dwork, C., Naor, M., and Ostrovsky, R. (1997). Deniable encryption. In *Advances in Cryptology-CRYPTO'97*, pages 90–104. Springer.

Casey, E. and Stellatos, G. J. (2008). The impact of full disk encryption on digital forensics. *ACM SIGOPS Operating Systems Review*, 42(3):93–98.

Codenomicon (2014). Drown: Breaking tls using sslv2. `http://heartbleed.com/`. Accessed 2016 jun 07.

Dworkin, M. (2010). Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices. In *NIST Special Publication*.

Fruhwirth, C. (2005). New methods in hard disk encryption. `http://clemens.endorphin.org/nmihde/nmihde-letter-os.pdf`. Accessed 2016 Jun 07.

GNU, P. (1999). The GNU Privacy Guard. `https://www.gnupg.org/`. Accessed 2016 jun 07.

Guardian, P. (2014). Luks: Linux Unified key Setup. `https://gitlab.com/cryptsetup/cryptsetup`. Accessed 2016 jun 07.

Gupta, D. and Mehtre, B. M. (2013). Recent trends in collection of software forensics artifacts: Issues and challenges. In *Security in Computing and Communications*, pages 303–312. Springer.

Halderman, J. A., Schoen, S. D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J. A., Feldman, A. J., Appelbaum, J., and Felten, E. W. (2009). Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98.

Henry, A. (2015). Five best file encryption tools. `http://lifehacker.com/five-best-file-encryption-tools-5677725`. Accessed 2016 jun 08.

IDRIX (2014). Veracrypt. `https://veracrypt.codeplex.com/`. Accessed 2016 jun 07.

IDRIX (2015). Veracrypt user's guide. `http://cyberside.net.ee/veracrypt/VeraCrypt%20User%20Guide.pdf`. Accessed 2016 Jun 01.

Kessler, G. C. (2007). Anti-forensics and the digital investigator. In *Australian Digital Forensics Conference*, page 7. Accessed 2016 Apr 22.

Kornblum, J. D. (2009). Implementing bitlocker drive encryption for forensic analysis. *digital investigation*, 5(3):75–84.

Lowman, S. (2010). The effect of file and disk encryption on computer forensics. Accessed 2016 Apr 22.

Manes, C. (2015). The top 24 free tools for data encryption. `http://www.gfi.com/blog/the-top-24-free-tools-for-data-encryption/`. Accessed 2016 jun 08.

Martin, L. (2010). Xts: A mode of aes for encrypting hard disks. *IEEE Security & Privacy*, (3):68–69.

Microsoft (2006). Bitlocker. `https://technet.microsoft.com/en-us/windows/bitlocker-and-bitlocker-to-go`. Accessed 2016 jun 07.

Mrdovic, S. and Huseinovic, A. (2011). Forensic analysis of encrypted volumes using hibernation file. In *TELFOR, 2011 19th*, pages 1277–1280. IEEE.

NTLDR (2007). Diskcryptor: Open source partition encryption solution. `https://diskcryptor.net/wiki/Main_Page`. Accessed 2016 jun 07.

OHR (2009). Proficiency scale. `https://hr.od.nih.gov/workingatnih/competencies/proficiencyscale`. Accessed 2016 Jun 01.

Pavlov, I. (1999). 7-zip. `http://www.7-zip.org/`. Accessed 2016 jun 07.

PCM, P. M. C. (2014). Ciphershed: Secure Encryption Software. `https://www.ciphershed.org/`. Accessed 2016 jun 07.

R-Tools, T. I. (2007). R-crypto: Data security for windows. `http://www.r-tt.com/data_security_software/`. Accessed 2016 jun 07.

Sharma, R. (2016). 7 Best Encryption Software for Windows. `http://beebom.com/best-encryption-software-windows/`. Accessed 2016 jun 08.

Zhang, L., Zhou, Y., and Fan, J. (2014). The forensic analysis of encrypted truecrypt volumes. In *PIC, 2014 International Conference on*, pages 405–409. IEEE.

Classificação de Fragmentos de Arquivos com Técnica de Aprendizagem de Máquina baseada em Árvores de Decisão

Juliano K. M. Oya[1], Bruno W. P. Hoelz[2]

[1] Perito Criminal, Instituto de Criminalística
Polícia Civil do Distrito Federal, Brasília-DF, Brasil

[2] Perito Criminal, Instituto Nacional de Criminalística
Polícia Federal, Brasília-DF, Brasil

juliano.oya@pcdf.df.gov.br, werneck.bwph@pdf.gov.br

Abstract. *The classification of file fragments is an important problem in computer forensics. This paper describes a flexible method for classifying file fragments using machine learning techniques. We used evidence files from real forensic cases to generate the training and testing fragments. From 12,153 evidence files of 21 different types, we generated and selected over a million of fragments with 1, 2 and 4 kilobytes of size. For each fragment, we extracted 45 attributes, which were subjected to machine learning techniques based on decision trees and, as a result, we obtained an average hit percentage of 98.78% for binary classifiers and 86.05% for multinomial classifiers.*

Resumo. *A classificação de fragmentos de arquivos é um importante problema na computação forense. Este artigo descreve um método flexível para classificar fragmentos de arquivos através de técnicas de aprendizagem de máquina. Foram utilizados arquivos de evidências de casos periciais reais para gerar os fragmentos de treinamento e teste. A partir de um total de 12.153 arquivos de evidências, de 21 tipos diferentes, foram gerados e selecionados mais de um milhão fragmentos de tamanhos de 1, 2 e 4 kilobytes. De cada fragmento foram extraídos 45 atributos, os quais foram submetidos a técnicas de aprendizagem de máquina baseadas em árvores de decisão e, como resultado, obteve-se um percentual de acerto médio de 98,78% para classificadores binários e de 86,05% para classificadores multinomiais.*

1. Introdução

A classificação de fragmentos é uma importante atividade na decodificação de fragmentos de memória, análise de fragmentos de arquivos em tráfegos de rede, detecção de *software* maliciosos, entre outras [Roussev e Quates 2013]. No caso específico da computação forense, a classificação de fragmentos é uma atividade essencial e indispensável para o propósito de recuperação de arquivos armazenados em mídias sem informações sobre o seu sistema de arquivos [Fitzgerald et al. 2012].

O problema da classificação de fragmentos consiste em determinar o tipo do arquivo ao qual pertence o fragmento. Em situações em que o fragmento é parte do cabeçalho ou do final do arquivo, a classificação é uma tarefa mais simples, visto que a maioria dos arquivos armazenam assinaturas ou estruturas de dados que indicam o seu tipo nas áreas iniciais ou finais do arquivo. Entretanto, em situações em que os

fragmentos são partes do meio do arquivo, onde há pouca ou nenhuma informação que identifiquem o seu tipo, a classificação torna-se uma tarefa desafiadora.

A técnica mais utilizada para a classificação e recuperação de arquivos – inclusive fragmentos de arquivos – é por meio da busca por assinaturas conhecidas, as quais ficam localizadas no início e no final de um arquivo. Entretanto, essa técnica tem como pressupostos que o arquivo possui o cabeçalho e o final do arquivo e, ainda, que o arquivo foi alocado de forma contínua na mídia de armazenamento. Como mostrado por Cohen (2007), nem sempre esses pressupostos estão presentes nas mídias de armazenamento.

A fim de verificar a aplicabilidade da técnica de recuperação de arquivos baseadas em assinaturas, foi realizado um teste sobre um conjunto de fragmentos de arquivos sem as informações de cabeçalho e de final do arquivo (o mesmo conjunto de fragmentos, como será apresentado posteriormente, foi utilizado nos experimentos deste trabalho). Para tanto, foi utilizado o arquivo de perfis da ferramenta *Foremost* (2016) – com 80 assinaturas de 40 tipos de arquivos – para classificar os fragmentos de acordo com as assinaturas encontradas dentro dos fragmentos. O resultado, apresentado na Tabela 1, são percentuais de acertos de 0,54%, 0,92% e 1,25% para a classificação dos fragmentos, o que mostra que a técnica não pode ser aplicada na recuperação de fragmentos sem as informações de cabeçalho e de final do arquivo.

Tabela 1. Resultado da classificação baseada em assinatura dos fragmentos.

	FRAGMENTOS DE 1 KBYTES	FRAGMENTOS DE 2 KBYTES	FRAGMENTOS DE 4 KBYTES
Número de fragmentos	1.215.000	1.215.000	1.210.160
Número de acertos	6.661	11.284	15.245
Número de erros	1.208.339	1.203.716	1.194.915
Não classificados	816.267	696.807	530.402
Percentual de acertos	0,54%	0,92%	1,25%

Alternativamente ao uso de assinaturas na classificação de fragmentos, vários trabalhos foram realizados com a aplicação de técnicas de aprendizagem de máquina para resolver o problema de classificação de fragmentos [Veenman 2007, Calhoun e Coles 2008, Axelsson 2010, Conti et al. 2010, Li et al. 2010, Fitzgerald et al. 2012].

A aprendizagem de máquina, aplicada na classificação de fragmentos de arquivos, consiste em construir um modelo computacional a partir de dados de atributos de fragmentos cujos tipos são conhecidos e, posteriormente, utilizar o modelo construído para predizer ou classificar novos fragmentos cujos tipos são desconhecidos.

Nos trabalhos citados, de modo geral, a aplicação de técnicas de aprendizagem de máquina segue os seguintes passos: coleta de arquivos, fragmentação dos arquivos, extração dos atributos dos fragmentos, consolidação dos atributos em dados de treinamento e validação, treinamento do modelo de classificação e, por fim, a avaliação do modelo.

A proposta apresentada neste trabalho inova ao considerar os atributos utilizados por outros pesquisadores e, ainda, novos atributos extraídos a partir dos fragmentos; ao utilizar uma base de dados de 2.830.472 fragmentos extraídos de arquivos de 21 tipos diferentes; ao utilizar a aprendizagem de máquina baseada em árvores de decisão, cuja precisão média na classificação dos fragmentos se mostrou promissora e superior aos outros trabalhos.

O restante do trabalho está organizado em 5 seções. Na seção 2 são analisados os principais pontos dos trabalhos realizados na mesma área de pesquisa. Na seção 3 é descrita a classificação de fragmentos baseada em árvores de decisão. Na seção 4 é descrita a metodologia de trabalho para a realização do experimento. Na seção 5 é descrito o experimento realizado, o qual é dividido em subseções onde são relatadas a seleção dos arquivos, a extração dos fragmentos, a extração dos atributos, a classificação dos fragmentos e o resultado obtido. Na seção 6, por fim, são apresentadas a conclusão e a indicação de trabalhos futuros.

2. Trabalhos relacionados

Os trabalhos de Veenman (2007), Calhoun e Coles (2008), Axelsson (2010), Conti et al. (2010), Li et al. (2010) e Fitzgerald et al. (2012) também exploraram a aplicação de técnicas de aprendizagem de máquina com o objetivo de classificar fragmentos de arquivos. Em geral, os experimentos realizados nesses trabalhos variam em número e tamanho dos fragmentos utilizados como base de treinamento e validação, número de tipos de arquivos utilizados na classificação, números de atributos extraídos dos fragmentos e, por fim, a técnica de aprendizagem de máquina utilizada.

No experimento realizado por Veenman (2007), foram utilizados um conjunto de 3.000 a 20.000 fragmentos por tipo de arquivos, dentre 11 tipos distintos. Foram utilizados fragmentos de 4 *Kbytes* de tamanho, dos quais foram extraídos os atributos de histograma de 1-grama, entropia de *Shannon* de 1-grama e a complexidade de *Kolmogorov*. O autor utiliza um discriminante linear para classificar os fragmentos e obtêm uma precisão média do classificador de 45%.

Já Calhoun e Coles (2008) alcançaram uma precisão média de 88%, que fora obtida limitando-se a 4 tipos de arquivos (`JPG`, `BMP`, `GIF` e `PDF`), os quais são analisados aos pares, através da aplicação de um classificador de regressão linear. O autor extraiu, dos fragmentos, atributos estatísticos (média, desvio padrão e correlação de valores), frequências de caracteres *ASCII*, entropia de *Shannon* de 1-grama e, ainda, combinou os valores desses atributos.

O trabalho realizado por Axelsson (2010) utilizou o maior número de tipos de arquivos (28 tipos distintos), entretanto, obteve a menor precisão média com o classificador *k-nearest-neighbors* (34%). O autor utilizou 31.541 fragmentos de 512 *bytes*, dos quais extraiu a medida NDC (*Normalised Compression Distance*), que é baseada no cálculo de complexidade de *Kolmogorov*.

O trabalho realizado por Conti et al. (2010) utiliza o classificador *k-nearest-neighbors* aplicado sobre os atributos de medidas estatísticas, entropia de *Shannon* de 2-gramas, peso de *Hamming*, Qui-quadrado e média aritmética, os quais foram extraídos de 14.000 fragmentos de 1 *Kbyte*. Os autores obtiveram uma precisão média de 96%, entretanto, em vez de utilizar tipos específicos de arquivos, os autores utilizaram grupos bastante abrangentes no classificador: randômico/compactado/criptografado, codificado em *Base64*, codificado em *Uuencoding* (*Unix-to-Unix encoding*), código de máquina, texto e *Bitmap*.

Através da aplicação de SVM (*Support Vector Machine*), Li et al. (2010) obtiveram uma precisão média de 81,5%. Para seu experimento, os autores extraíram o histograma de 1-grama de fragmentos de 4 *Kbytes* de 3.600 arquivos. Utilizaram, no entanto, apenas 5 tipos de arquivos (`JPEG`, `DLL`, `EXE`, `MP3` e `PDF`).

Dentre os trabalhos apresentados, os autores Fitzgerald et al. (2012) realizaram o experimento com o maior número de atributos extraídos dos fragmentos de 24 tipos de arquivos. Para cada fragmento foram extraídos: histograma 1-grama, histograma 2-gramas, medidas estatísticas, entropia de *Shannon* 1-grama, entropia de *Shannon* 2-gramas, peso de *Hamming*, média aritmética, taxa de compressão do arquivo, média da distância entre *bytes* consecutivos e maior sequência contígua de *bytes* repetidos. Apesar da grande quantidade de atributos considerados, a precisão média de 47,5% pode ser considerada baixa.

3. Classificação de fragmentos baseada em árvores de decisão

Os algoritmos baseados em árvores de decisão são destinados tanto para a regressão quanto para a classificação. Tais algoritmos envolvem a segmentação recursiva dos dados em subconjuntos, um para cada valor do atributo. Para cada árvore de decisão é definida um conjunto de regras, que definem um caminho a ser percorrido na árvore até a classificação do fragmento.

Os dados para a aprendizagem dos algoritmos podem ser ajustados, de modo a produzir dois modelos de classificadores:

1. *Classificador binomial ou binário*: o classificador resultante será capaz de distinguir dois tipos. Esse modelo foi utilizado por Calhoun e Coles (2008), que realizaram a classificação de pares de tipos na forma `TYPE` vs. `NOT-TYPE`.
2. *Classificador multinomial ou multiclasse*: o classificador resultante será capaz de distinguir múltiplos, ou seja, um conjunto de dois ou mais tipos. Esse modelo foi utilizado por Fitzgerald et al. (2012).

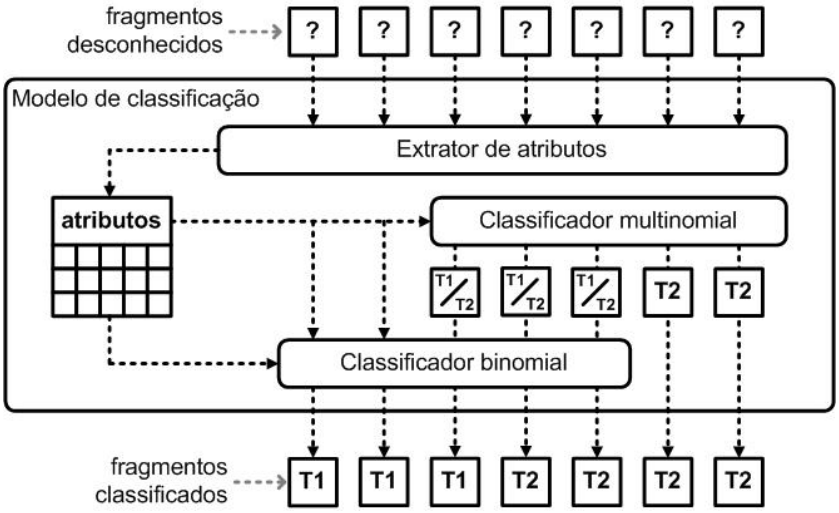


Figura 1. Aplicação dos modelos de classificadores binomiais e multinomiais.

Comumente os classificadores binomiais e multinomiais são utilizados de forma isolada. Um cenário possível é a utilização combinada dos dois modelos de classificadores, como ilustrado na Figura 1, na qual um classificador mais especializado é aplicado sempre que se há dúvidas a respeito do tipo de um fragmento previamente processado por um classificado mais genérico. A seção seguinte descreve o método de trabalho para se obter os classificadores binomiais e multinomiais.

4. Método de trabalho

O método de trabalho consiste em 4 fases principais, quais sejam: (1) seleção dos arquivos, (2) extração dos fragmentos dos arquivos, (3) extração dos atributos dos fragmentos e (4) treinamento e validação do classificador.

No fluxograma apresentado na Figura 2 são representadas as fases (1) e (2). Nela pode-se observar como o conjunto inicial de arquivos é filtrado, quando são retirados os arquivos idênticos, menores que 10 *Kbytes* e maiores que 100 *Mbytes*.

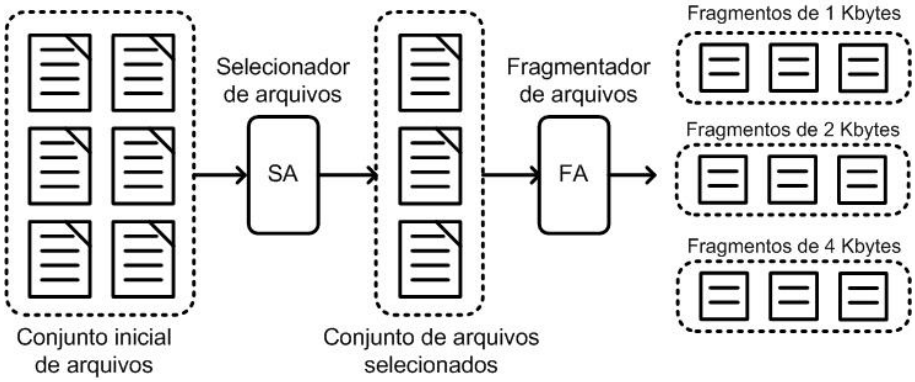


Figura 2. Visão geral das fases seleção e fragmentação de arquivos.

Já no fluxograma apresentado na Figura 3 é representada a fase (3), quando são extraídos um conjunto de atributos dos fragmentos dos arquivos. Os símbolos `SA`, `FA`, `EA` (`EA1`, `EA2`, ..., `EAn`) e `CD` são módulos do protótipo desenvolvido para automatizar as atividades das fases (1), (2), (3) e (4).

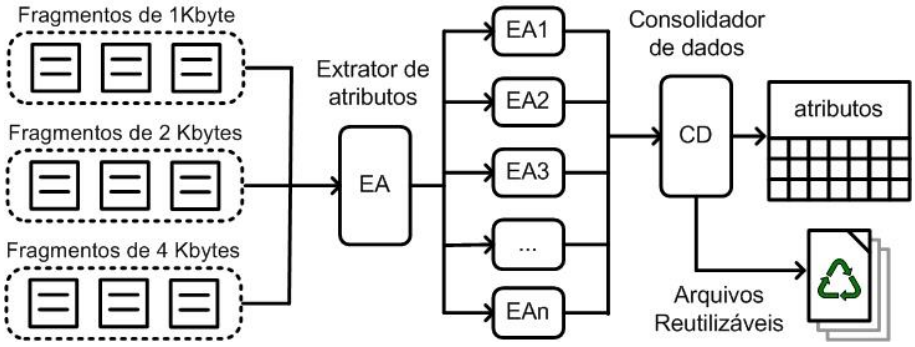


Figura 3. Visão geral da fase de extração dos atributos dos fragmentos.

Ainda na Figura 3, pode-se observar que foram implementados módulos específicos para extrair cada atributo (são os subitens denominados `EA1`, `EA2`, ..., `EAn`). Ao final dessa fase, os dados são consolidados e salvos em um conjunto de arquivos.

A fase (4) do método de trabalho consiste em transformar o conjunto de dados dos atributos em conjuntos binomiais e multinomiais, ambos para o aprendizado supervisionado do algoritmo de classificação baseado em árvores de decisão. Por fim, o resultado é um conjunto reutilizável de modelos de classificadores binomiais e multinomiais. Na Figura 4 é ilustrada essa fase.

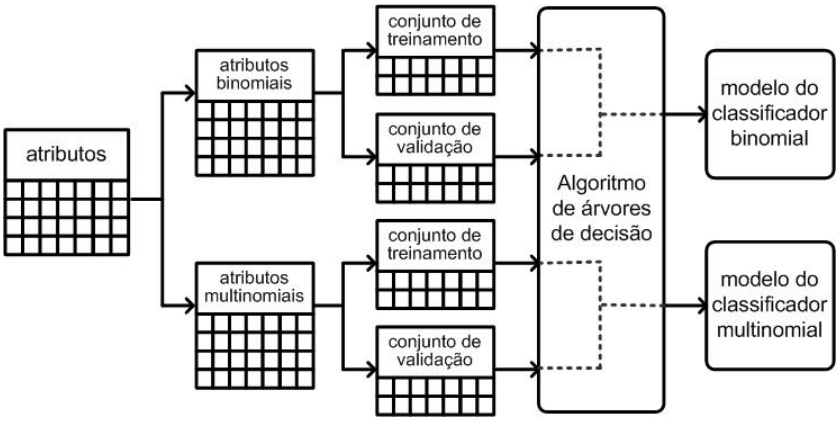


Figura 4. Visão geral da fase de treinamento e validação dos classificadores.

5. Experimentos

Utilizando-se do método de trabalho apresentado anteriormente, foram realizados vários experimentos com dados de fragmentações de 1, 2 e 4 *Kbytes*. Todo o processo, que envolve a seleção de arquivos, a extração de fragmentos e a extração de atributos do fragmento, foi automatizado através de um protótipo desenvolvido em linguagem *Java*. Os dados coletados foram exportados para a ferramenta *Weka* [Waikato 2016a], na qual foi aplicado o algoritmo de classificação baseado em árvores de decisão. Nas seções seguintes é descrita cada fase do processo.

5.1. Desenvolvimento do protótipo

Um protótipo em linguagem *Java* foi desenvolvido como prova de conceito. A implementação automatiza todas as fases do método de trabalho, ou seja, realiza a seleção dos arquivos, a extração dos fragmentos, a extração dos atributos e a consolidação final dos dados. A automatização das atividades se mostrou essencial para a realização do experimento, dada a grande quantidade de informações e de passos para processar e analisar os dados.

Para cada atributo foi desenvolvida uma classe específica que o extrai (ressalte-se que o protótipo pode ser facilmente adaptado para extrair novos atributos). Dessa forma, foram implementados algoritmos específicos para extrair os atributos descritos na Tabela 4 da seção 5.4. Ainda, a fim de criar a compatibilidade com o projeto *Weka*, o protótipo realiza a conversão automática dos dados dos atributos extraídos para o formato `ARFF` [Waikato 2016b].

5.2. Seleção do algoritmo de classificação

Foi realizado um experimento com um conjunto reduzido de 21.000 registros de dados de fragmentos de 1, 2 e 4 *Kbytes* (ou seja, cerca de 2% da base total utilizada no experimento principal). Sobre esses registros, foram executados os algoritmos de classificação baseados em árvores de decisão *Decision Stump*, *J48*, *REPTree*, *Random Forest* e *Random Tree* [Waikato 2016c], a fim de avaliar o percentual de acertos, o tempo de execução e o tamanho físico do modelo do classificador. O resultado é apresentado na Figura 5, onde a média dos valores de fragmentos de 1, 2 e 4 *Kbytes* dos parâmetros avaliados são postos em uma escala de 0 a 1.

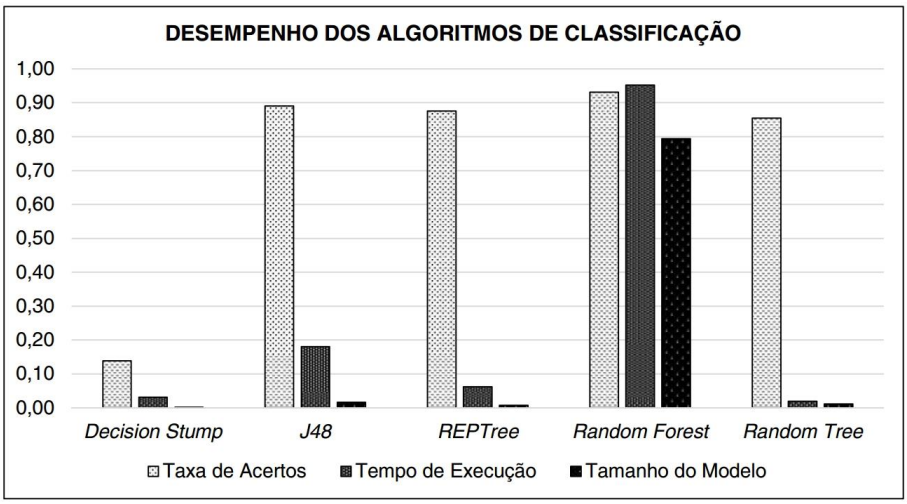


Figura 5. Avaliação de desempenho dos algoritmos de classificação.

Dessa forma, o algoritmo *J48* foi selecionado para a realização do experimento principal, por apresentar taxa de acertos acima da média e bons tempo de execução e tamanho físico do modelo de classificação – em detrimento, por exemplo, do algoritmo *Random Forest*, que apesar de apresentar a melhor taxa de acertos, teve o pior tempo de processamento e originou um modelo com tamanhos físicos extremamente grandes.

5.2. Seleção dos arquivos

Foram selecionados cerca de 500 relatórios de evidências periciais de casos reais, cujas investigações ocorreram nos anos de 2014, 2015 e 2016. Dos relatórios, foram extraídos os arquivos de evidências vinculados a eventos de natureza criminal e, dentre esses arquivos, foram selecionados 12.153 arquivos de 21 tipos diferentes. Na Tabela 2 são descritos os tipos selecionados para o experimento.

Tabela 2. Descrição dos tipos de arquivos selecionados para o experimento.

TIPO	DESCRIÇÃO DO TIPO
AVI	Arquivos de vídeos AVI (*Audio Video Interleave*).
BMP	Arquivos de imagens *Bitmap*.
DLL	Arquivo de bibliotecas dinâmicas DLL (*Dynamic-Link Library*).
DOC	Arquivos de documentos do programa *Microsoft Word*.
DOCX	Arquivos de documentos do programa *Microsoft Word* com formato XML.
DWG	Arquivos do programa *Autocad*.
EML	Arquivos de mensagens de *e-mails* com codificação *Base64*.
EXE	Arquivos de programas executáveis.
HTML	Páginas *web* no formato HTML (*Hiper Text Markup Language*).
JAR	Arquivos de programas ou bibliotecas em *Java*.
JPG	Arquivos de imagem JPEG (*Joint Photographic Experts Group*).
MP3	Arquivos de áudio MP3 (*MPEG-1/2 Audio Layer 3*).
MP4	Arquivos de vídeos MP4 (*MPEG-4 Part 14*).
PDF	Arquivos de documentos do programa *Adobe Acrobat Reader*.
RTF	Arquivos de documentos RTF (*Rich Text Format*).
SQLL	Arquivos de banco de dados *SQLite*.
TIF	Arquivos de imagens TIFF (*Tagged Image File Format*).
TXT	Arquivos de texto simples.
WMV	Arquivos de vídeos WMV (*Windows Media Player*).
XML	Arquivos de documentos XML (*Extensible Markup Language*).
ZIP	Arquivos de documentos compactado no formato *Winzip*.

Os tipos selecionados foram aqueles mais bem representados dentro do conjunto total dos arquivos. Dessa forma, foram selecionados os tipos que possuíam a maior quantidade de arquivos e a maior quantidade de espaço ocupado em disco. O primeiro critério visa obter a maior variedade possível de arquivos para o experimento; já o segundo visa obter uma quantidade expressiva de fragmentos.

A medida que os arquivos eram selecionados, também eram calculados os valores do *hash* MD5 dos arquivos, a fim de evitar que a amostra possuísse arquivos idênticos. Portanto, os arquivos que possuíam o mesmo valor de *hash* de um arquivo já selecionado eram excluídos.

Foram excluídos, ainda, os arquivos menores que 10 *Kbytes*, os quais não gerariam fragmentos de 4 *Kbytes* úteis para o experimento, uma vez que os fragmentos inicial e final dos arquivos (que podem conter dados de assinaturas do tipo de arquivo) são descartados; e também foram excluídos os arquivos maiores que 100 *Mbytes*, os quais produziriam uma grande quantidade não desejada de fragmentos do mesmo arquivo. O resultado final do processo de seleção de arquivos é apresentado na Tabela 3.

Tabela 3. Quantitativos dos arquivos selecionados para o experimento.

TIPO	TAMANHO (KB)	% DO TOTAL	QUANTIDADE	% DE ARQUIVOS
AVI	195.233	4,76%	12	0,10%
BMP	195.311	4,76%	199	1,64%
DLL	195.313	4,76%	300	2,47%
DOC	195.312	4,76%	901	7,41%
DOCX	195.307	4,76%	716	5,89%
DWG	195.309	4,76%	269	2,21%
EML	195.310	4,76%	198	1,63%
EXE	195.306	4,76%	72	0,59%
HTML	195.312	4,76%	2.424	19,95%
JAR	195.312	4,76%	275	2,26%
JPG	195.193	4,76%	265	2,18%
MP3	195.306	4,76%	136	1,12%
MP4	195.211	4,76%	88	0,72%
PDF	195.214	4,76%	358	2,95%
RTF	195.298	4,76%	2.020	16,62%
SQLL	195.312	4,76%	401	3,30%
TIF	194.737	4,75%	33	0,27%
TXT	195.310	4,76%	1.320	10,86%
WMV	195.308	4,76%	29	0,24%
XML	195.312	4,76%	1.886	15,52%
ZIP	195.308	4,76%	251	2,07%

5.3. Extração dos fragmentos dos arquivos

A partir dos 12.153 arquivos selecionados foram extraídos os fragmentos de 3 tamanhos distintos: 1, 2 e 4 *Kbytes*. Os fragmentos que faziam parte do cabeçalho ou do final dos arquivos foram excluídos, a fim de eliminar qualquer vestígio da assinatura dos arquivos. O resultado final foi a seleção de 2.830.472 fragmentos, divididos da seguinte forma: 945.000 fragmentos de 1 *Kbytes*, 945.000 fragmentos de 2 *Kbytes* e 940.472 fragmentos de 4 *Kbytes*.

Os fragmentos foram selecionados de forma homogênea entre os tipos de arquivos. Para a maioria dos tipos, foram selecionados 45.000 fragmentos por tipo. A exceção ocorreu para os tipos HTML, RTF e XML, para os quais foram selecionados 42.578, 43.783 e 44.111 fragmentos de 4 *Kbytes*, respectivamente.

5.4. Extração dos atributos dos fragmentos

A partir dos fragmentos selecionados, foram extraídos os principais atributos utilizados nos experimentos realizados por Veenman (2007), Calhoun e Coles (2008), Axelsson (2010), Conti et al. (2010), Li et al. (2010), Fitzgerald et al. (2012). Foram extraídos, ainda, atributos de frequência de caracteres *Base85*, *Monte Carlo Pi*, correlação serial, frequência de *bytes* `x00`, `xFF` e `xF8`, entropia de *Shannon* de 3-gramas, histograma de 3-gramas, identificador de língua, identificador de linguagem de programação e identificador de delimitadores de *tags*. Na Tabela 4 são descritos, brevemente, os principais atributos extraídos de cada fragmento do experimento.

Tabela 4. Descrição dos atributos extraídos de cada fragmento.

NOME DADO AO ATRIBUTO	DESCRIÇÃO DO ATRIBUTO
`ascii-percentual`	Percentual de caracteres `ASCII` imprimíveis.
`ascii-freq`	Frequência de *bytes* no fragmento com o valor *v*, onde `x00` ≤ *v* < `x20`.
`ascii-low-freq`	Frequência de *bytes* no fragmento com o valor *v*, onde `x20` ≤ *v* ≤ `x7E`.
`ascii-high-freq`	Frequência de *bytes* no fragmento com o valor *v*, onde `x80` ≤ *v*.
`base64-percentual`	Percentual de caracteres com codificação *Base64*.
`base85-percentual`	Percentual de caracteres com codificação *Base85*.
`byte-mean`	Média dos valores dos *bytes* do fragmento.
`byte-sd`	Desvio padrão dos valores dos *bytes* do fragmento.
`byte-cornext`	Correlação dos valores dos *bytes* nas posições *n* e *n+1*.
`chi-square`	Calcula o grau de aleatoriedade dos *bytes* do fragmento.
`npl-dist-mean`	Calculo média da distância entre os valores de dois *bytes* consecutivos.
`npl-long-seq`	Cálculo da sequência mais longa de *bytes* repetidos.
`monte-carlo-pi`	Cálculo do valor *Monte Carlo para Pi* utilizando-se cada sequência sucessiva de 6 *bytes* como coordenadas *X* e *Y*.
`serial-correlation`	Calcula o grau de dependência de cada *byte* com o *byte* anterior.
`pattern-[FF,F8] -[1,2,3]gram`	Contagem do número de *bytes* `xFF` e `xF8`, inclusive sequencias de 1, 2 e 3 gramas (ou seja, sequências de 1, 2 e 3 *bytes* iguais a `xFF` e `xF8`).
`hamming-weight`	Calcula o total de *bits* 1dividido pelo total de *bits* do fragmento.
`histo-[1,2,3]gram-modes`	Soma dos 4 maiores valores do histograma de combinações de 1, 2 e 3 *bytes*.
`histo-[1,2,3]gram-sd`	Desvio padrão dos valores do histograma de combinações de 1, 2 e 3 *bytes*.
`histo-[1,2,3]gram-cornext`	Correlação da frequência dos valore *m* e *m+1* do histograma de combinações de 1, 2 e 3 *bytes*.
`lang-detect`	Detecção da língua utilizada no texto (ex: português, inglês, espanhol, japonês, etc.)
`program-lang`	Detecção da linguagem de programação ou de marcação utilizada. (ex: Java, C, Python, HTML, etc).
`entropy-[1,2,3]gram`	Cálculo do grau de entropia de 1, 2 e 3 gramas (ou seja, combinações de 1, 2 e 3 *bytes*).
`tag-score`	Contagem do número de *bytes* com valores `x28`, `x29`, `x3C`, `x3E`, `x5B`, `x5D`, `x7B`, `x7D`.
`zero-[1,2]gram-percentual`	Percentual de *bytes* `x00` no fragmento, inclusive sequencias de 1 e 2 gramas.
`zip-compression-score`	Taxa de compressão do fragmento.

A partir dessa fase do experimento, cada fragmento passa a ser representado pelo seu conjunto de atributos.

5.5. Treinamento e validação do classificador

O conjunto de atributos foram consolidados e, em seguida, convertidos para o formato `ARFF`, o qual é o formato de arquivo utilizado pelo projeto *Weka*. Por meio da ferramenta *Weka* foi aplicado o algoritmo de árvores de decisão *J48*.

O algoritmo *J48*, cuja implementação é baseada no algoritmo *C4.5* [Quinlan 1993], percorre todos os atributos de modo a identificar aquele apresenta o maior ganho de informação (ou seja, maior contribuição para o resultado na árvore). Após identificar esse atributo, ele é definido como um nó na árvore (ou a raiz, caso seja a primeira iteração) [Bell 2014].

Neste trabalho, foi realizada a classificação binomial na forma `TYPE` vs. `NOT-TYPE` e, dessa forma, o algoritmo de classificação foi executado 21 vezes, uma vez para cada tipo, na qual a cada execução o fragmento é classificado como de um tipo específico ou como diferente do tipo específico. Também foi realizada a classificação multinomial, na qual o classificador resultante será capaz de distinguir qualquer um dos 21 tipos.

Na Figura 6 é apresentada, como exemplo, a árvore de decisão extraída após o processamento do algoritmo *J48* para a classificação binomial dos fragmentos de 1 *Kbyte* do tipo `EML`. Cada linha segue o formato `[regra-de-decisão]: [classificação]([total-de-instâncias]/[erro-de-classificação])`. No final da figura pode-se observar o percentual de classificações corretas e incorretas e a matriz de confusão do classificador.

```
base64-percentual <= 0.998047
|   ascii-percentual <= 0.999023: NOT-EML (774069.0)
|   ascii-percentual > 0.999023
|   |   histo-2gram-sd <= 0.071029
|   |   |   ascii-freq <= 0.987305
|   |   |   |   histo-2gram-modes <= 0.042198
|   |   |   |   |   ascii-freq <= 0.979492
|   |   |   |   |   |   base64-percentual <= 0.843137
|   |   |   |   |   |   |   tag-score-3C <= 0.055172: EML (22.0/2.0)
|   |   |   |   |   |   |   tag-score-3C > 0.055172:  NOT-EML  (2.0)
|   |   |   |   |   |   base64-percentual > 0.843137: NOT-EML (23.0)
|   |   |   |   |   ascii-freq > 0.979492: EML (35.0)
|   |   |   |   histo-2gram-modes > 0.042198
|   |   |   |   |   tag-score-3E <= 0.042471
|   |   |   |   |   |   histo-3gram-modes <= 0.038348
|   |   |   |   |   |   |   entropy-2gram <= 0.824984
|   |   |   |   |   |   |   |   lang-detect = ??: NOT-EML (0.0)

... linhas removidas ...

Correctly Classified Instances      944817      99.9806 %
Incorrectly Classified Instances       183      0.0194 %

... linhas removidas ...

Confusion Matrix:
      a      b   <-- classified as
  44880    120 |     a = EML
     63 899937 |     b = NOT-EML
```

Figura 6. Árvore de decisão produzida pelo algoritmo *J48* sobre fragmentos do tipo `EML`.

5.6. Resultados

Ao final do processamento do algoritmo *J48*, foram obtidos 63 classificadores binomiais (específicos para cada um dos 21 tipos e para cada tamanho de fragmento de 1, 2 e 4 *Kbytes*) e 3 classificadores multinomiais (específicos para cada um dos tamanhos de 1, 2 e 4 *Kbytes*). As taxas de precisão foram consolidadas e são apresentadas nas Tabelas 5 e 6.

Tabela 5. Resultado da classificação binomial do algoritmo *J48*.

TAXA DE CLASSIFICAÇÃO CORRETA CLASSIFICAÇÃO BINOMIAL			
TIPO	**FRAG. DE 1 KBYTES**	**FRAG. DE 2 KBYTES**	**FRAG. DE 4 KBYTES**
AVI	98,73%	98,88%	99,12%
BMP	99,72%	99,83%	99,88%
DLL	99,22%	99,20%	98,87%
DOC	97,58%	97,31%	97,19%
DOCX	99,70%	99,60%	99,44%
DWG	99,79%	99,56%	99,60%
EML	99,98%	99,99%	99,93%
EXE	96,14%	97,46%	97,93%
HTM	99,90%	99,94%	99,95%
JAR	97,69%	98,39%	98,67%
JPG	96,68%	96,84%	97,40%
MP3	99,14%	99,59%	99,76%
MP4	96,64%	97,75%	98,11%
PDF	96,33%	97,63%	98,40%
RTF	99,79%	99,85%	99,92%
SQLITE	99,67%	99,69%	99,68%
TIF	99,41%	99,71%	99,56%
TXT	99,56%	99,69%	99,84%
WMV	98,54%	98,89%	98,15%
XML	99,89%	99,94%	99,86%
ZIP	95,88%	95,70%	96,26%
média	98,57%	98,83%	98,93%

Tabela 6. Resultado da classificação multinomial do algoritmo *J48*.

TAXA DE CLASSIFICAÇÃO CORRETA CLASSIFICAÇÃO MULTINOMIAL			
TIPO	**FRAG. DE 1 KBYTES**	**FRAG. DE 2 KBYTES**	**FRAG. DE 4 KBYTES**
AVI	87,10%	88,60%	91,40%
BMP	97,60%	98,20%	98,70%
DLL	90,60%	89,70%	86,10%
DOC	97,00%	96,00%	93,80%
DOCX	69,10%	68,70%	69,40%
DWG	98,10%	95,20%	95,60%
EML	99,80%	99,90%	99,20%
EXE	58,10%	78,00%	79,90%
HTM	98,90%	99,40%	99,60%
JAR	69,20%	79,30%	83,20%
JPG	64,70%	68,00%	73,50%
MP3	90,50%	95,00%	97,10%
MP4	58,10%	72,60%	79,30%
PDF	56,40%	70,40%	79,80%
RTF	97,10%	97,80%	98,80%
SQLITE	96,10%	96,40%	96,60%
TIF	92,30%	96,50%	95,30%
TXT	96,60%	98,10%	98,80%
WMV	81,80%	85,90%	79,60%
XML	98,90%	99,30%	98,80%
ZIP	50,30%	49,00%	56,70%
média	83,25%	86,76%	88,15%

Os classificadores binomiais apresentaram melhores percentuais de acertos (uma média de 98,78% de classificações corretas). Foi observado, ainda, que a medida que os fragmentos aumentam de tamanho, há um aumento na taxa de precisão.

Já os classificadores multinomiais, apresentaram uma queda em seus percentuais de classificação (uma média de 86,05% de classificações corretas). No caso específico do classificador multinomial, como era esperado, há uma redução no percentual de acertos do modelo, em troca da flexibilidade de se ter um único modelo capaz de classificar todos os tipos.

A seguir, na Tabela 7, é apresentada a matriz de confusão do classificador multinomial para fragmentos de 4 *Kbytes*. A diagonal da tabela indica os índices de classificação positiva, ou seja, um fragmento do tipo sendo classificado corretamente.

Dessa forma, os dados com a taxa de classificação correta para os fragmentos de 4 *Kbytes*, os quais foram inicialmente apresentados na Tabela 6, são posicionados na diagonal da matriz da Tabela 7. Ademais, para cada item da diagonal, na mesma linha ou coluna, são apresentados os percentuais de classificações incorretas de cada tipo.

Tabela 7. Matriz de confusão da classificação multinomial do algoritmo *J48* (4 *Kbytes*).

	classificado como																				
	AVI	BMP	DLL	DOCX	DOC	DWG	EML	EXE	HTML	JAR	JPG	MP3	MP4	PDF	RTF	SQLI	TIF	TXT	WMV	XML	ZIP
AVI	91,4	0,0	0,2	0,1	0,3	0,2	0,0	0,2	0,0	0,1	0,1	0,7	2,6	0,2	0,0	0,0	0,3	0,1	2,8	0,0	0,6
BMP	0,0	98,7	0,2	0,1	0,1	0,1	0,0	0,0	0,0	0,1	0,0	0,0	0,1	0,1	0,0	0,1	0,2	0,1	0,0	0,0	0,0
DLL	0,2	0,3	86,1	0,3	1,6	1,2	0,0	3,5	0,0	0,8	0,3	0,1	0,7	0,7	0,1	1,3	0,1	0,8	0,7	0,1	1,1
DOCX	0,1	0,1	0,3	93,8	1,9	0,1	0,0	0,1	0,0	0,2	0,7	0,1	0,2	0,9	0,0	0,4	0,0	0,0	0,2	0,0	1,0
DOC	0,2	0,1	1,5	1,6	69,4	0,3	0,0	0,7	0,0	0,9	14,9	0,1	0,7	2,3	0,1	0,6	0,3	0,1	1,1	0,0	4,9
DWG	0,2	0,1	0,9	0,2	0,3	95,6	0,0	0,2	0,0	0,1	0,1	0,0	0,2	0,3	0,0	0,4	0,1	0,2	0,7	0,0	0,2
EML	0,0	0,0	0,0	0,0	0,0	0,0	99,2	0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,7	0,0
EXE	0,2	0,1	4,0	0,1	0,8	0,3	0,0	79,9	0,0	1,7	0,0	0,1	2,7	0,5	0,0	0,3	0,1	0,4	0,3	0,0	8,5
HTML	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,0	99,6	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2	0,0
JAR	0,2	0,1	0,8	0,3	0,9	0,2	0,0	2,5	0,0	83,2	0,2	0,0	0,8	1,8	0,0	0,2	0,6	0,0	1,1	0,1	7,0
JPG	0,1	0,0	0,3	0,7	14,7	0,2	0,0	0,0	0,0	0,2	73,5	0,0	0,1	5,2	0,0	0,0	0,3	0,1	0,4	0,0	4,1
MP3	0,7	0,0	0,0	0,0	0,1	0,1	0,0	0,0	0,0	0,1	0,0	97,1	0,9	0,1	0,0	0,0	0,0	0,1	0,6	0,0	0,1
MP4	2,8	0,0	0,8	0,2	0,8	0,3	0,0	3,8	0,0	0,7	0,1	1,0	79,3	0,7	0,0	0,0	0,1	0,1	6,1	0,0	3,0
PDF	0,2	0,1	0,7	1,1	2,8	0,4	0,0	1,1	0,0	2,1	5,9	0,1	0,7	79,8	0,1	0,1	0,4	0,0	0,8	0,0	3,4
RTF	0,0	0,1	0,1	0,0	0,2	0,1	0,0	0,1	0,0	0,1	0,0	0,0	0,1	0,1	98,8	0,0	0,0	0,0	0,0	0,0	0,2
SQLI	0,1	0,1	1,0	0,5	0,5	0,4	0,0	0,2	0,0	0,1	0,1	0,0	0,1	0,1	0,0	96,6	0,0	0,0	0,2	0,0	0,1
TIF	0,4	0,1	0,2	0,0	0,4	0,1	0,0	0,1	0,0	0,6	0,2	0,0	0,2	0,3	0,0	0,0	95,3	0,0	1,2	0,0	0,9
TXT	0,0	0,0	0,2	0,0	0,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,2	0,0	0,0	0,1	0,0	98,8	0,1	0,1	0,0
WMV	3,2	0,0	0,7	0,2	1,3	0,7	0,0	0,4	0,0	1,1	0,4	0,6	6,5	0,7	0,1	0,2	1,4	0,1	79,6	0,0	3,0
XML	0,0	0,0	0,1	0,0	0,0	0,0	0,7	0,0	0,2	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,0	98,8	0,0
ZIP	0,9	0,1	1,3	1,2	5,6	0,3	0,0	10,9	0,0	6,9	4,9	0,1	3,4	3,5	0,1	0,1	1,0	0,0	2,9	0,0	56,7

Ainda na Tabela 7, foram destacados os maiores percentuais de erros de classificação dos fragmentos. Esses são casos que requerem maiores estudos para introdução de novos atributos que auxiliem nessa classificação. Por exemplo, no caso específico da classificação dos fragmentos dos tipos `JPG`, 14,7% dos fragmentos foram classificados como fragmentos do tipo `DOC` e 14,9% de fragmentos do tipo `DOC` foram considerados como fragmentos `JPG`. Esse erro de classificação pode ser atribuído a presença de imagens `JPG` nos fragmentos do tipo `DOC`.

Da mesma forma, observa-se que o tipo `ZIP` tem a pior taxa de acerto (56,7%). A matriz mostra que 10,9% dos fragmentos `ZIP` foram classificados como fragmentos do tipo `EXE`. No sentido oposto, 8,5% dos fragmentos `EXE` foram classificados como fragmentos do tipo `ZIP`. Isso decorre do fato de que vários outros tipos de arquivos utilizam compressão semelhante àquelas aplicadas nos arquivos `ZIP`, tais como `JAR`, `DOCX` e executáveis comprimidos com *packers* (`EXE`).

Os casos de falso-positivos e falso-negativos têm impactos diretos sobre o processo posterior de recuperação do arquivo. No primeiro, o componente de recuperação tentaria recuperar um fragmento inválido; já no segundo, o componente de recuperação poderia ignorar incorretamente o fragmento válido. Esse cenário indica a necessidade de classificadores específicos como os classificadores binários apresentados anteriormente. Note-se que vários classificadores binários podem ser aplicados a um mesmo fragmento. Se mais de um classificador indicar que aquele fragmento é do seu tipo, o componente de recuperação deverá testar as duas hipóteses. O importante, nesse caso, é que, pelo menos, um dos classificadores gere uma classificação positiva correta, apesar dos demais falsos-positivos.

6. Conclusão

Neste trabalho foi explorado o uso de técnicas de aprendizagem de máquina baseadas em árvores de decisão aplicadas no problema de classificação de fragmentos para posterior recuperação do arquivo. Em cenários nos quais inexistem informações sobre o sistema de arquivos e, ainda, inexistem o cabeçalho e o final do arquivo, a classificação e a recuperação de arquivos se tornam atividades desafiadoras.

Foi empregado o algoritmo de classificação *J48*, considerado um classificador estável e de amplo uso na área de aprendizagem de máquina, sobre um conjunto de fragmentos extraídos de arquivos de 21 tipos distintos. Como resultado, obteve-se um conjunto de classificadores binomiais especializados para 21 tipos de arquivos e um conjunto de classificadores multinomiais.

O resultado do experimento mostrou que mesmo quando inexistentes a informação do cabeçalho, do final do arquivo e do sistema de arquivos, é possível classificar um fragmento com percentuais de acertos de 98,78% (com classificadores binomiais) e de 86,05% (com classificadores multinomiais). Logo, considera-se que a utilização de tal técnica pode trazer grandes ganhos para o exame pericial ao complementar as demais formas de recuperação de dados.

Devido ao caráter experimental do trabalho, muito ainda precisa ser feito a respeito da (1) análise dos atributos utilizados no experimento, (2) proporção dos tipos de fragmentos do experimento e (3) tratamento de fragmentos de tipos desconhecidos.

Assim, deve-se analisar a necessidade de normalizar ou mesmo discretizar alguns atributos, assim como analisar o custo e o impacto de cada um. Para determinados modelos de classificação, pode-se eliminar alguns desses atributos sem que haja prejuízos significativos aos resultados do modelo.

Pode-se equilibrar a amostragem de registros para o modelo de classificação binária. O experimento atual foi realizado com uma proporção de 4,76% contra 95,24% dos fragmentos, de tal forma que algumas árvores de classificação binária se tornaram "especialistas" em identificar fragmentos que não pertencem ao tipo. Experimentos com uma quantidade reduzida de registros mostrou que, após o ajuste para uma proporção de 50% contra 50%, os modelos de classificação têm uma redução média de 5% na taxa de acerto do algoritmo.

Nesse experimento não é dado um tratamento para os fragmentos de tipos desconhecidos no classificador multinomial e, portanto, inexiste uma classe "outros". Então, se um fragmento de um tipo desconhecido (ex.: `GIF`) for apresentado, o classificador necessariamente vai indicar um dos 21 tipos esperados, gerando um falso-positivo para aquele tipo.

Agradecimentos. Os autores agradecem o apoio da Secretaria Nacional de Segurança Pública (SENASP), do Departamento de Engenharia Elétrica da Universidade de Brasília, da Fundação de Peritos em Criminalística Ilaraine Acácio Arce (FPCIAA), da Associação Brasiliense de Peritos Criminais (ABPC), da Diretoria Técnico-Científica da Polícia Federal e da FINEP (Convênio 01.12.0433.01, Projeto: Defesa Nacional e Segurança Pública) na realização deste trabalho.

Referências Bibliográficas

Axelsson, S. (2010) "The normalized compression distance as a file fragment classifier". Proceedings of the 2010 Digital Forensics Research Conference (DFRWS).

Bell, J. (2014) "Machine Learning: Hands-on for developers and technical professionals". John Wiley & Sons.

Calhoun, W., Coles, D. (2008) "Predicting the types of file fragments". Proceedings of the 2008 Digital Forensics Research Conference (DFRWS).

Cohen, M. I. (2007) "Advanced carving techniques". In Digital Investigation: The International Journal of Digital Forensics & Incident, volume 4, pages 119-128.

Conti, G., Bratus, S., Sangster, B., Ragsdale, R., et al. (2010) "Automated mapping of large binary objects using primitive fragment type classification". Proceedings of the 2010 Digital Forensics Research Conference (DFRWS).

Garfinkel, S. L. (2007) "Carving contiguous and fragmented files with fast object validation". Journal Digital Investigation, volume 4, pages 2-12.

Fitzgerald, S., Mathews, G., Morris, C. and Zhulyn, O. (2012) "Using NLP techniques for file fragment classification". Journal Digital Investigation, volume 9.

Foremost (2016). Disponível em: <http://foremost.sourceforge.net/>.

Li, Q., Ong, A., Suganthan, P., Thing, V. (2010) "A novel support vector machine approach to high entropy data fragment classification". Proceedings of the South African Information Security Multi-Conference.

Penrose, P., Macfarlane, R., and Buchanan, W. J. (2013) "Approaches to the classification of high entropy file fragments". Journal Digital Investigation, volume 10, issue 4, pages 372-384.

Quinlan, R. (1993) "C4.5: Programs for Machine Learning". Morgan Kaufmann Publishers.

Roussev, V. and Quates, C. (2013) "File fragment encoding classification-An empirical approach". Journal Digital Investigation, volume 10, pages S69-S77.

Veenman, CJ. (2007) "Statistical disk cluster classification for file carving". In Proceedings of the IEEE 3rd international symposium on information assurance and security, IEEE Computer Society, pages 393–8.

Waikato (2016a) "Weka: Waikato Environment for Knowledge Analysis Tool". Disponível em: <http://www.cs.waikato.ac.nz/~ml/>.

Waikato (2016b) "Weka: Attribute-Relation File Format". Disponível em: <http://weka.wikispaces.com/ARFF>.

Waikato (2016c) "Weka: Decision Tree Algorithms". Disponível em: <http://weka.sourceforge.net/doc.stable/weka/classifiers/trees/package-summary.html>.

TARP Fingerprinting: Um Mecanismo de Browser Fingerprinting Baseado em HTML5 Resistente a Contramedidas

Pablo Ximenes[1,3], Márcio Correia[3], Patrícia Mello[2], Fernando Carvalho[3], Miguel Franklin[3], Rossana Andrade[3]

[1]Governo do Estado do Ceará – Empresa de Tecnologia da Informação do Ceará (ETICE) – Fortaleza – CE – Brasil

[2]Instituto Atlântico – Fortaleza – CE – Brasil

[3]Universidade Federal do Ceará (UFC) – Mestrado e Doutorado em Ciência da Computação (MDCC) – Fortaleza – CE – Brasil

pablo@ximen.es, marcio@ufc.br, patriciatmello@gmail.com, fcarvalho@ufc.br,miguel@lia.ufc.br,rossana@ufc.br

Resumo. *Este artigo apresenta um novo mecanismo de browser fingerprinting baseado em HTML5 que é resistente a contramedidas. Mais especificamente, apresentamos o mecanismo TARP Fingerprinting †, que permite a geração de uma assinatura identificadora de um navegador de internet (browser) através do uso do elemento <canvas> presente na API HTML5. Diferentemente do modelo tradicional de Canvas Fingerprinting, o mecanismo apresentado é resistente às técnicas usadas como contramedidas ao Canvas Fingerprinting, especialmente àquelas que bloqueiam a técnica adulterando o comportamento normal do elemento <canvas>. O mecanismo proposto atinge esse objetivo através da modulação da entropia das assinaturas geradas pelo elemento <canvas>, que passam a funcionar também como uma espécie de "marca d'água" ou "lacre de garantia". Ademais, apresentamos um dos dois únicos estudos existentes atualmente que medem a entropia da técnica de Canvas Fingerprinting em larga escala (>64.000 amostras) com resultados de entropia de Shannon expressivos.*

1. Introdução

O mecanismo conhecido como *cookies* foi inventado por Lou Montulli em 1994 e tinha o propósito original de rastrear as atividades de um usuário em um website [PEN2000]. Cookies são pequenas *strings* de texto recebidas pelo navegador ao fazer uma conexão HTTP e associadas ao sítio web visitado, de forma automática e com mínima participação do usuário. Assim, ao retornar ao sítio web visitado anteriormente, o navegador automaticamente envia, junto com sua requisição, a *string* do cookie gravada outrora. Essa string serve para identificar o navegador web em sucessivas requisições e permite que todas essas requisições sejam associadas umas às outras em um verdadeiro sistema de rastreamento HTTP.

Contudo, o modelo de cookies possui uma fragilidade gritante: o usuário sempre poderá apagar o cookie (ou mesmo bloqueá-lo). Essa deficiência permite a existência de diversas ferramentas e condutas de usuário que inibem o rastreamento alcançado pelo uso de cookies. Assim, embora ainda amplamente utilizados, os cookies estão perdendo gradualmente sua eficiência. Seja por conta de condutas mais astutas de usuários preocupados com a privacidade, seja pelo uso de ferramentas que diminuem a capacidade

de rastreamento oferecida pelos cookies. Isso se torna um problema em especial para o segmento de publicidade e propaganda online. Esta indústria, para poder entregar anúncios com contexto de real interesse a seus usuários, utiliza técnicas que dependem enormemente do rastreamento trazido pela técnica. Inviabilizar esse rastreamento pode significar anúncios menos eficazes. Outro segmento que teria muito a perder com a inviabilização dos cookies é o segmento de proteção à fraude (em especial o de proteção à fraude bancária) que depende do rastreamento de usuários para identificar comportamentos fraudulentos [NIK2013].

Por conta disso, outras formas de rastreamento de clientes WEB se fizeram necessárias. Estas alternativas necessariamente tiveram que se basear em elementos que identifiquem o browser, mas que não pudessem ser facilmente destruídos pelo usuário, como em uma espécie de "impressão digital" (ou fingerprinting) do navegador [ACA2014]. Dessa forma, diversas técnicas de browser fingerprinting foram propostas [UPA2015]. Em especial, o Canvas Fingerprinting [MOW2012] é destas técnicas a que mais obteve sucesso [ACA2014]. Ele é baseado na coleta de informações gráficas de renderização do HTML5 presente no navegador para gerar uma assinatura.

Contudo, o Canvas Fingerprinting, bem como as demais técnicas de Browser Fingerprinting propostas, depende de que o código gerador da assinatura seja executado, em grande parte, no lado do cliente, onde não existem controles de segurança que garantam que este código não foi adulterado. Isso significa dizer que o comportamento de um determinado navegador pode ser alterado de forma a forjar uma assinatura. Isso tem se tornado cada vez mais simples com a existência de navegadores que permitem a instalação de plug-ins como o FireFox e o Google Chrome [WAN2012][LIU2012]. De fato, já existem inclusive alguns plug-ins no mercado que prometem proteção contra Canvas Fingerprinting [APP2016][KKA2016]. Eles simplesmente alteram a resposta do código de coleta da assinatura para enviar dados forjados ou simplesmente lixo.

Para contornar esse problema, propomos uma variação do Canvas Fingerprinting resistente a contramedidas. Nossa técnica, chamada de *TARP Fingerprinting*[†], se vale da modulação da entropia presente na geração de imagens na técnica do Canvas Fingerprinting. O mecanismo proposto explora a incapacidade de diferenciar imagens com alta entropia de imagens com baixa entropia por parte das técnicas de contramedida ao Canvas Fingerprinting. Ao invés de usar o *hash* de uma única imagem, como no Canvas Fingerprinting tradicional, nosso mecanismo usa o conteúdo de múltiplas imagens, variando a entropia entre elas, de forma a conseguir identificar uma resposta que tente forjar a assinatura. [1]

Este artigo está organizado da seguinte maneira. A seção 2 trata de uma discussão das técnicas de fingerprinting de browser. A seção 3 fala especificamente do Canvas Fingerprinting. A seção 4 discute as contramedidas existentes contra o Canvas Fingerprinting. A seção 5 detalha nossa proposta. A seção 6 fala dos resultados obtidos, demonstrando protótipo e experimentos realizados. Por fim, a seção 7 tece algumas conclusões.

[†] O nome "TARP Fingerprinting" possui uma relevância simbólica. Enquanto "*canvas*" é a tradução inglesa de "lona", a palavra "*tarp*" (abreviação de *tarpaulin*) é a expressão inglesa usada para designar a lona que foi tratada quimicamente para se tornar à prova d'água.

2. Browser Fingerprinting

De acordo com [UPA2015], as técnicas de browser fingerprinting podem ser categorizadas em 4 grupos principais. São eles: o Browser Specific Fingerprinting; o Cross Browser Fingerprinting; o Javascript Engine Fingerprinting; e o Canvas Fingerprinting.

Ao longo desta seção, detalharemos cada uma destas categorias, à exceção do Canvas Fingerprinting, que será tratado de forma exclusiva na seção seguinte, dada a sua importância basal para o modelo proposto.

2.1. Browser Specific Fingerprinting

Trata-se da primeira leva de técnicas usadas para identificar navegadores web. Ela é baseada na coleta de strings semiestatizas presentes de forma específica nos diversos navegadores. Ela faz isso através do uso de funcionalidades específicas do Browser como o Java e o Flash. Exemplos destas strings são a resolução de tela, as fontes instaladas, plug-ins instalados, a string User-Agent, dentre outros. A exemplo desta categoria, podemos citar o projeto Panopticlick [ECK2010] que foi o precursor das demais técnicas de Browser Fingerprinting aqui discutidas [ACA2014]. Nele foi demonstrado que navegadores com Flash ou Java instalados poderiam ser distinguidos dos demais de forma individual mesmo sem o uso de cookies [ECK2010].

2.2. Cross Browser Fingerprinting

Essa categoria tenta se valer de técnicas que não sejam exclusivas de um navegador específico. Por exemplo, uma subcategoria dela é o Fingerprinting de Acelerômetro. Neste exemplo, os erros de calibração do acelerômetro são utilizados como estratégia para identificar individualmente o navegador [UPA2015]. Em contraste com a categoria anterior, a Cross Browser Fingerprinting não depende de um plug-in ou nada específico do browser, que pode ou não estar instalado, para obter os dados geradores da assinatura. Poderia, por exemplo, obter a mesma lista de fontes instaladas como na técnica de Browser Specific, mas ao invés de usar o Flash ou Java, usaria o Javascript, que é uma funcionalidade presente em todos os navegadores. Essa característica de usar estratégias que funcionem em qualquer navegador é o que define a categoria.

2.3. Javascript Engine Fingerprinting

Esta categoria se vale de testes de conformidade aplicados ao Java Script para obter assinaturas. Basicamente ela realiza uma série de testes de conformidade, se valendo muitas vezes de uma suíte específica de testes para isso. Cada navegador irá "passar" em alguns testes e falhar em outros. Essa relação de testes aprovados e testes reprovados comporá uma assinatura que indicará, na maioria dos casos, o tipo e versão do navegador, de forma muito mais confiável que a análise da string "USER-AGENT" [UPA2015]. Exemplo desta categoria é o trabalho realizado em [MUL2013]. Nele, os autores realizam uma série de testes de conformidade Javascript através da ferramenta Sputnik [SPU2016] de forma a gerar uma assinatura.

3. Canvas Fingerprinting

O Canvas Fingerprinting é uma técnica relativamente recente, proposta em 2012 por Mowery e Shacham [MOW2012]. É a forma de Browser Fingerprinting mais usada na

Internet atualmente [ACA2014]. Basicamente, a estratégia é desenhar uma imagem (com texto e cenas WebGL) usando comandos gráficos do HTML5 através da *tag* <canvas> para, posteriormente, fazer a captura da imagem desenhada. A imagem capturada é usada para construir a assinatura, já que suas nuances a podem deixar exclusiva para cada navegador. Essa assinatura é, em seguida, remetida ao servidor que realiza o rastreamento. Nesta seção discutiremos em detalhes essa técnica.

3.1. Detalhes de Implementação

O Canvas Fingerprinting é implementado através do uso do elemento <canvas> presente na API do HTML5. Na prática, essa técnica é aplicada através de um código em Javascript que é executado no navegador ao visitar um site. O código abaixo demonstra a simplicidade de implementação desta técnica em Javascript:

```
<script type="text/javascript"> var canvas = document.getElementById("drawing"); var context = canvas.getContext("2d"); context.font = "18pt Arial"; context.textBaseline = "top"; context.fillText("Hello, user.", 2, 2); </script>
```

Esse código é um exemplo de renderização gráfica de um texto (no caso a palavra inglesa "drawing"). O resultado é uma imagem contendo o referido texto encapsulada em um objeto <canvas> que não precisa ser necessariamente exibida na tela do usuário. O objeto <canvas>, por sua vez, possui o método toDataURL(). Quando executado com argumento "image/png", por exemplo, esse método retorna o conteúdo gráfico integral do canvas na forma de uma imagem PNG codifica em base64. O que ocorre é que cada navegador particular tem uma forma diferente de "renderizar" essa imagem. Conforme o gráfico torna-se mais complexo, mais única e exclusiva é essa forma. Essa complexidade pode ser trabalhada de forma a gerar imagens que sejam exclusivas para cada navegador e que não variem nele mesmo quando produzidas a partir de sites diferentes. Isso faz com que a imagem bem trabalhada, gerada por um Canvas, seja um verdadeiro identificador, uma verdadeira impressão digital.

Essa imagem, por sua vez, passa a ser tratada pelo script como elemento primordial na geração da assinatura. A proposta original do Canvas Fingerprinting sugere que essa imagem não seja remetida de volta por completo ao site que a solicitou. Em vez disso, os autores propões que um hash (MD5, por exemplo) seja gerada ainda dentro do browser em Javascript, de forma que apenas esse hash seja enviado de volta. Isso economizaria largura de banda na transmissão dos dados da assinatura.

3.2. Entropia

A forma mais comum de se medir a efetividade do Canvas fingerprinting é o cálculo da entropia de Shannon [MOW2012][LAP2016]. Ela é denotada pela fórmula: $E = -\sum_{i=1}^{n} \rho(x_i) log_2\, \rho(x_i)$. Em última análise, essa fórmula representa o número de bits de entropia gerados por um dado modelo de assinatura.

Em [MOW2012], os próprios propositores da técnica de canvas fingerprinting só foram capazes de realizar um experimento pequeno, com apenas 294 amostras, chegando a uma entropia de Shannon de menos 6 bits, o que é baixo. Dada a limitação deste experimento, os autores argumentam que esse valor subestima o potencial da técnica.

De fato, até recentemente a mensuração em larga escala da entropia da técnica de Canvas Fingerprinting ainda era um problema em aberto. Contudo, [LAP2016] apresentou um amplo estudo de várias técnicas de *fingerprinting* (incluindo também o

Canvas Fingerprinting) em maio de 2016. Nele, o site *amiuniq.org* aplicava várias técnicas de fingerprinting aos seus visitantes, de forma que a coleta de assinaturas vem sendo realizada desde 2014. Todas as técnicas aplicadas tiveram sua entropia medida em um conjunto de mais de 100 mil amostras. O objetivo dos autores foi realizar um estudo nas mesmas proporções do projeto Panopticlick [ECK2010], mas incluindo novas técnicas como a de *canvas*. Como veremos a seguir, o nosso trabalho apresenta contribuição similar, apresentando também um estudo da entropia do canvas fingerprinting com grande número de amostras.

3.3. Problemas

O grande problema do Canvas Fingerprinting é ser altamente dependente do cliente na geração de assinaturas, onde pouco se faz no lado do servidor para conter contramedidas. De fato, o trabalho seminal inclusive propôs que a técnica fosse implementada através da execução de um script Javascript no browser do usuário que devolvesse ao servidor apenas o *hash* da imagem gerada. Essa estratégia, apesar de trazer benefícios no tocante a desempenho, mascara um grande problema de segurança. Ao delegar a execução do algoritmo para o cliente, não existem garantias de que o código remetido pelo servidor é o mesmo que foi executado. De fato, em uma perspectiva onde o cliente é adversário, o algoritmo certamente será adulterado de forma a torna-lo ineficaz. O Canvas Fingerprinting não é resistente a adulteração, justamente por isso. Ademais, qualquer técnica que tente ao menos identificar que houve adulteração torna-se inviável, já que a única informação recebida pelo servidor é o *hash* da imagem, um dado sumarizado que carrega pouca informação a respeito da execução do algoritmo em si.

Essa discussão não é apenas teórica, já que ferramentas de bloqueio do Canvas Fingerprinting que exploram essa fragilidade têm surgido [KKA2016][APP2016]. De fato, por conta de sua popularidade, a técnica acaba por tornar-se grande foco no desenvolvimento de contramedidas.

4. Contramedidas ao Canvas Fingerprinting

Os mecanismos de rastreamento de usuários na web geram uma preocupação de privacidade que estimulou a criação de técnicas de proteção. Da mesma forma que novas propostas de rastreamento surgem, diversas contramedidas contra essas mesmas propostas são criadas [CAR2014]. Isso se torna mais evidente no caso do Canvas Fingerprinting devido a sua popularidade.

De acordo com [KHA2014], podemos classificar as contramedidas que tentam proteger contra o Browser Fingerprinting em 3 categorias principais: as baseadas em aleatoriedade, as baseadas em filtragem e as baseadas em forja. Discutiremos como cada uma delas se aplica ao Canvas Fingerprinting no decorrer da seção.

4.1. Contramedidas Baseadas em Aleatoriedade

Este tipo de contramedida modifica aleatoriamente as propriedades do navegador web de forma a enganar os geradores de assinaturas. Como as técnicas de fingerprinting geram a assinatura com base nessas propriedades, esse tipo de contramedida faria com que as assinaturas geradas se tornassem também aleatórias, inviabilizando qualquer rastreamento.

No caso do Canvas Fingerprinting, essa contramedida é acionada quando o navegador identifica a captura (para posterior cálculo de hash) do conteúdo de imagem renderizado. Isso pode ser feito através da identificação do uso do método toDataURL, responsável pela "captura" da imagem gerada. Nesse caso, ao invés de permitir que o hash da imagem original fosse enviado, o navegador substituiria o conteúdo do objeto por uma imagem aleatória. Assim, o hash calculado seria dessa imagem aleatória, gerando assinaturas diferentes para toda nova requisição. Os plug-ins CanvasBlocker para Firefox [KKA2016] e o CanvasFingerprintBlock [APP2016] para Chrome, por exemplo, podem ser configurados para fazer justamente isso.

4.2. Contramedidas Baseadas em Filtragem

Esse tipo de contramedida simplesmente desabilita a propriedade ou recurso usado para gerar a assinatura, ou a deixa funcionando, mas retornando resultados vazios. Isso impende a coleta de dados necessários à geração da assinatura. No caso do Canvas Fingerprinting, isso seria realizado através da completa desativação do elemento <canvas> do HTML5.

4.3. Contramedidas Baseadas em Forja

Esse tipo de contramedida mantém ativos os elementos ou propriedades necessárias à composição da assinatura, mas faz com que seus resultados sejam forjados. Por exemplo, ao invés de retornar a *string* identificadora do browser (user agente) real, o navegador retornaria uma versão falsa, possivelmente forjando um outro navegador.

No caso do Canvas Fingerprinting, essa contramedida novamente terá seu fluxo relacionado à identificação da captura do conteúdo de imagem renderizado. Novamente, isso pode ser feito através da identificação do uso do método toDataURL. Nesse caso, ao invés de permitir que o hash da imagem original fosse enviado, o navegador substituiria o conteúdo do objeto por uma imagem forjada, representando outro navegador ou mesmo inválida. Assim, o hash calculado seria dessa imagem forjada, gerando uma assinatura falsa, diferente da assinatura do próprio usuário. Os plug-ins CanvasBlocker (Firefox) e o CanvasFingerprintBlock (Chrome) também podem ser configurados para fazer isso.

4.3. Efeitos Colaterais das Contramedidas

Todas as formas de contramedida discutidas podem ter como efeito colateral o comprometimento das funcionalidades do navegador, já que a ferramenta que operacionaliza a contramedida não possui inteligência para distinguir o uso legítimo das propriedades do navegador que são modificadas. Contudo, no caso do Canvas Fingerprinting, esse efeito é menor. O plug-in CanvasBlocker, por exemplo, pode ser configurado para superar essa limitação através de whitelists. Dessa forma, o próprio usuário poderá "treinar" a ferramenta para que ela saiba quais são os sites que fazem uso legítimo e quais usam os elementos para rastreamento.

5. TARP Fingerprinting

O Tarp Fingerprinting funciona de forma similar ao Canvas Fingerprinting no tocante ao uso do elemento <canvas> do HTML5. Da mesma forma que seu antecessor, ele trabalha com a renderização de elementos gráficos no browser que são capturados para a geração

da assinatura. Contudo, diferem em aspectos essenciais que representam uma evolução entre os modelos.

A proposta do TARP Fingerprinting é estender o Canvas Fingerprinting de forma que se torne resistente às contramedidas discutidas na seção anterior. Em termos gerais, modificamos o modelo original de forma que ele seja capaz de perceber quando alguma dessas contramedidas está sendo posta em prática. Dessa forma, ele poderá simplesmente descartar a assinatura falsa ou negar funcionalidade ao usuário que se recuse a fornecer sua assinatura. Isso é realizado mudando-se o algoritmo original para que trabalhe com múltiplos canvas (no mínimo dois) ao invés de apenas um. Parte dessas imagens servirá para compor a assinatura semelhante à maneira original, enquanto que a outra parte servirá como uma espécie de marca d'água, um verdadeiro "lacre de garantia" que, se violado, indicará que houve o uso de contramedida. Outra mudança importante é que o Tarp Fingerprinting não trabalha com *hashes* gerados pelo cliente. No lugar de acreditar no hash gerado pelo browser, o mecanismo proposto remete ao servidor todas as imagens geradas em sua integralidade.

Ao longo dessa seção discutiremos os detalhes de nossa proposta.

5.1. Substituição do modelo de Hash calculado no cliente

A diferença entre as duas técnicas começa com o fato de que o modelo Tarp não confia nos hashes gerados pelo browser como ocorre no modelo proposto em [MOW2012]. Nossa proposta exige que qualquer imagem gerada seja enviada ao servidor para inspeção em sua integralidade. Isso por si só já confere uma capacidade inicial de analisar a existência de contramedidas. Um *hash* é uma representação resumida de um conjunto de dados. Analisar dados através de seu *hash* é equivalente a analisar uma obra literária inteira somente através do título. Ou seja, teremos muito poucos elementos para verificar se houve qualquer adulteração da obra original se analisarmos somente o título. Quando temos acesso ao conteúdo completo, diversas estratégias podem ser aplicadas para aferir se houve a aplicação de alguma contramedida.

5.2. Grupos de Imagens

A segunda diferença entre os dois modelos é que o Tarp Fingerprinting passa a trabalhar com dois grupos de imagens ao invés de uma única imagem. É importante notar que cada um destes grupos pode ser representado por apenas uma imagem ou por várias imagens.

Como veremos a seguir, esses grupos serão usados para compor a assinatura e para identificar uso ou não de contramedida.

5.3. Imagens de Assinatura

O primeiro grupo é usado para compor a assinatura do browser, de forma similar ao Canvas Fingerprinting, mas com mais potencialidades. O fato de que estamos trabalhando com a imagem integral (ao invés de apenas seu *hash)* dá a liberdade à nossa proposta de aplicar qualquer transformação às imagens recebidas deste grupo para gerar a assinatura. Poderíamos, inclusive, usar todos os bits de seu conteúdo integral.

Esse grupo de imagens é modelado para possuir alta entropia, ou seja, ter características que fazer sua renderização não se repetir entre navegadores diferentes.

5.4. Imagens de Marca D'água

O segundo grupo, por sua vez, gera imagens que funcionarão como uma espécie de marca d'água no sentido de lacre de garantia. Serão imagens de referência que devem se repetir com bastante frequência entre navegadores de forma que sua adulteração seja imediatamente percebida. Assim, essas imagens possuirão baixa entropia.

5.5. Processamento da Assinatura

A assinatura do browser no modelo Tarp Fingerprinting se processa da seguinte maneira.

Inicialmente as imagens dos dois grupos são geradas através do elemento <canvas> por um script Javascript executado no navegador. Essas imagens são capturadas pelo método toDataURL() e remetidas ao servidor. No servidor as imagens são analisadas.

Inicialmente, o servidor computa a assinatura do browser com base nas imagens do grupo de assinatura. Isso pode ser feita de várias formas, inclusive por meio do cálculo de *hash* de tais imagens. Como estamos fazendo isso dentro do servidor, que é um ambiente confiável de computação, não temos as mesmas limitações do Canvas Fingerprinting nesse aspecto.

A segunda análise feita pelo servidor se dá com o grupo de imagens de marca d'água. O objetivo é saber se houve alguma modificação nessas imagens. Como tais imagens foram construídas de forma que permaneçam sempre as mesmas (com todos os seus bits iguais) ou com poucas variações entre os vários navegadores diferentes (baixa entropia), uma alteração nelas pode ser facilmente percebida. O sistema coletaria todas as variações da imagem de referência e construiria padrão de referência de suas versões. Como a entropia dessas imagens é baixa, existirão pouquíssimas variações. A imagem que seja diferente das variações mais comuns, ou seja, que esteja fora do padrão de referência, será tratada como adulterada. Isso permite que o modelo proposto identifique que houve o uso de uma contramedida.

Como discutido na sessão anterior, as contramedidas funcionam através da modificação do elemento <canvas> de forma que retorne resultados forjados ou até mesmo nenhum resultado. Como tais contramedidas não são capazes de identificar quais são os elementos <canvas> que devem ser modificados, elas, em via de regra, alteram indiscriminadamente todos os elementos <canvas> presentes no script. Fazem isso com objetivo de se proteger contra o rastreamento. Ao fazer isso justamente nas imagens do grupo de marca d'água, estas contramedidas estão, indiretamente, ativando um alerta de sua existência. Ora, as imagens de marca d'água são construídas de forma que sempre permaneçam as mesmas ou com pouquíssimas variações. Receber uma imagem de marca d'água que foge completamente à norma indica que o browser foi modificado para adulterar o comportamento do elemento <canvas>, o que significa o uso de contramedidas.

Dessa forma, o modelo proposto poderá considerar a assinatura recebida como falsa e, a partir daí, requisitar outra assinatura ou mesmo simplesmente negar serviço ao navegador infrator.

5.6. Arquitetura

A figura 1 ilustra a arquitetura da proposta separando-a em navegador web e servidor.

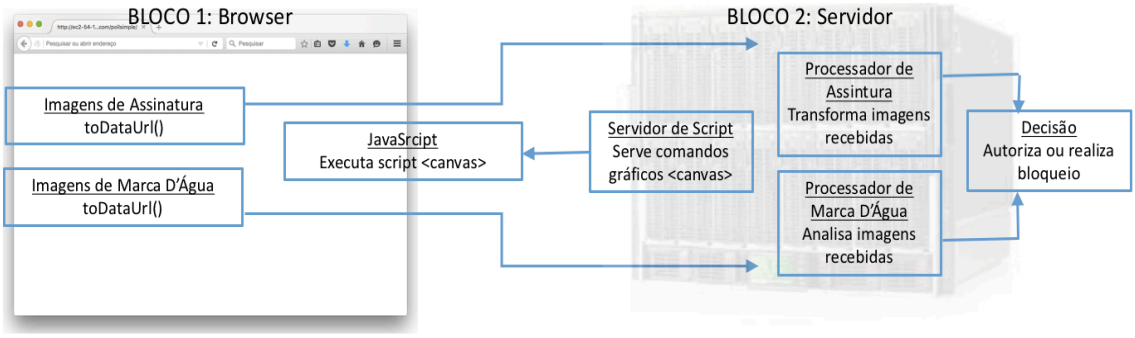


Figura 1. Arquitetura do Mecanismo Proposto

No navegador são geradas as imagens para cada um dos grupos através de script Javascript advindo do servidor. Posteriormente, cada grupo de imagens é enviado ao servidor para processamento. O processador de assinatura é responsável por transformar as imagens recebidas em uma assinatura. O processador de marca d'água é responsável por analisar as imagens recebidas para aferir se houve uso de contramedida. Os resultados dos dois processadores alimentam um módulo central de tomada de decisão que pode, inclusive, retroalimentar o mecanismo do Tarp Fingerprinting de forma que aperfeiçoe a geração de assinaturas.

5.7. Análise de Segurança

A premissa principal do mecanismo proposto é que as contramedidas aplicadas para tentar barrar a coleta de assinatura baseada no elemento <canvas> não conseguem distinguir a geração de imagens com alta entropia da geração de imagens com baixa entropia. Essa premissa, em última análise, significa que um mecanismo automático de contramedida não é capaz de obter o contexto necessário do código fonte Javascript de forma a saber qual imagem pode ou não pode ser alterada.

Contudo, para entendermos as implicações de segurança do modelo proposto, temos que definir o cenário ao qual estas se aplicam. No cenário que consideramos, temos um usuário web usando seu navegador municiado de algum plug-in de proteção contra o Canvas Fingerprinting. Nesse cenário, levando-se em conta o estado da arte, o software de contramedida será incapaz de distinguir qual <canvas> poderá ser alterado sem "disparar o alarme".

Apesar disso, devemos considerar o cenário onde o software de contramedida é customizado com ajuda do ser humano de forma estática. O programador do plug-in de proteção poderia, por exemplo, analisar uma implementação do Tarp Fingerprinting e modificar seu plug-in de forma a adicionar estaticamente a informação de qual <canvas> pode ser alterado sem acionar o mecanismo de detecção. Esse vetor de ataque pode ser facilmente debelado se programarmos o Tarp Fingerprinting para modificar os padrões de geração das imagens de marca d'água de forma dinâmica. Assim, o mecanismo proposto variaria as imagens de marca d'água de forma que não fiquem estáticas. Como a entropia de tais imagens é baixa, rapidamente o sistema se adaptaria para construir um novo padrão de referência. Isso pode ser feito através do módulo de decisão, que retroalimenta o sistema em conjunto com o módulo servidor de script. Para isso, geraríamos instâncias que incluíssem imagens já dentro do padrão de referência existente, juntamente com novas imagens diferentes, até que um padrão de referência alternativo

seja construído. Assim, a contramedida nunca poderá ser adaptada para identificar imagens de marca d'água de forma estática.

6. Experimento

Para aferir o modelo proposto foi necessário avaliar, principalmente, a entropia das imagens geradas. Nosso objetivo era verificar se poderíamos gerar objetos <canvas> com imagens de alta entropia e de baixa entropia de forma simultânea para um mesmo browser. Para tanto, desenvolvemos um protótipo que implementa a proposta e aplicamos este protótipo a um conjunto de usuários web. Com isso pudemos melhor balizar nossas conclusões. Discutiremos o protótipo desenvolvido bem como o experimento realizado ao longo desta seção.

6.1. Objetivos

Modelamos um conjunto de questionamentos que funcionam como marco hipotético do experimento. Todos eles trabalham na mesma conjuntura, onde lotes distintos de instruções gráficas HTML5 com o elemento <canvas> são processadas pelo universo atual de navegadores web que, por sua vez, geram as respectivas imagens, uma para cada lote de instruções. Dessa forma, queremos responder aos seguintes questionamentos. Existe variação de entropia de assinaturas baseadas nessas imagens, quando tomadas individualmente? É possível gerar uma assinatura baseada na combinação de duas imagens que possua entropia maior do que as duas assinaturas baseadas em cada uma dessas imagens individualmente? É possível gerar ao mesmo tempo assinaturas de baixa entropia e de alta entropia com essas imagens?

6.2. Protótipo

Desenvolvemos um protótipo do mecanismo proposto para aferir a proposta com base nos objetivos apresentados. Nele, usamos três grupos distintos de instruções gráficas HTML5 do tipo <canvas>.

O primeiro grupo de instruções (grupo L) foi modelado de forma que, individualmente, produza assinaturas de baixa entropia. Ele foi construído a partir da sobreposição de duas formas gráficas simples.

O segundo grupo de instruções (grupo H) foi modelado de forma que produza assinaturas de alta entropia. Ele foi construído a partir de uma simplificação da implementação da biblioteca *fingeterprint2.js* [VAL2016]. Nesta biblioteca, são implementadas várias técnicas de Browser Fingerprinting. Escolhemos essa implementação em particular pois ela usa uma interessante diversidade de variações gráficas do elemento <canvas> HTML5 na geração da imagem. Essas variações gráficas têm o objetivo exclusivo de conferir maior entropia à assinatura gerada a partir da imagem. Contudo, o autor não dá nenhuma informação de qual a entropia gerada por suas técnicas. Assim, o que fizemos para o grupo H foi isolar somente a implementação do Canvas Fingerprinting de [VAL2016].

Finalmente, o terceiro grupo de instruções (grupo M) é uma modificação das instruções de grupo H, feita de forma a incrementar a entropia das assinaturas geradas. O que fizemos para o grupo M foi duplicar todas as técnicas de incremento de entropia usadas nas variações gráficas do grupo H, bem como inserir em codificação gráfica, na

própria imagem, texto indicando a presença de certas propriedades (ex.: “canvas winding”). Um exemplo do grupo de instruções M pode ser visto na figura 2.

Figura 2. Exemplo de Imagem Gerada pelo Protótipo

Assim, o protótipo do modelo TARP gera a imagem de baixa entropia através da execução das instruções do grupo L e gera imagens de alta entropia através da execução das instruções do grupo H e do grupo M. Para diminuir o consumo de dados de armazenamento em memória, as assinaturas tratadas pelo modelo proposto são o cálculo do *hash* MD5 do arquivo binário de cada uma das imagens geradas. Dessa forma, o protótipo gera uma única assinatura de baixa entropia (que poderá ser usada como marca d’água) e duas assinaturas de alta entropia (usadas para compor um identificador distinto) por instância.

6.3. Coleta de Dados

Para avaliar com precisão a entropia gerada pelas instâncias do protótipo, desenhamos um teste de campo. Nele, instalamos o protótipo no sítio web de uma grande universidade brasileira de forma que todas as visitas gerassem uma instância do Tarp Fingerprinting da forma descrita anteriormente (imagens L, H e M).

A instalação se deu na forma da inclusão de um script em Javascript responsável por gerar as instâncias do Tarp e realizar toda a coleta de dados necessária. Todos os dados coletados estavam em consonância com os padrões de privacidade da Universidade em questão. Além dos dados tradicionais de uma transação web, foram coletadas as três imagens geradas em seu conteúdo binário integral (e não apenas o hash) para cada visita.

Nossa expectativa foi a de conseguir uma massa de dados grande o suficiente para avaliar a entropia gerada pelos grupos L, H e M com objetivo de responder os questionamentos do item 6.1.

6.4. Parâmetros de Avaliação

Para aferir os resultados obtidos usamos dois principais parâmetros de avaliação. O primeiro parâmetro é a entropia de Shannon, conforme detalhada anteriormente, já que este foi o mesmo parâmetro utilizado em [MOW2012] quando introduzida a técnica de Canvas Fingerprinting. O segundo parâmetro de avaliação é a entropia ponderada, técnica usada em [LAP2016]. A entropia ponderada é representada pela divisão da entropia de Shannon pela entropia do pior caso. A entropia do pior caso é aquela onde todas as amostras são diferentes entre si. Esse cálculo resulta em um fator que é independente do tamanho da amostra e pode ser usado para comparar amostras de tamanhos diferentes. Ademais, para melhor alinhar nossos resultados aos parâmetros de [LAP2016], também calculamos os parâmetros “assinaturas distintas” e “assinaturas únicas”. As “assinaturas distintas” são o total de assinaturas diferentes que são coletadas durante o levantamento de dados. As “assinaturas únicas” é a porção das “assinaturas distintas” que foi gerada

uma única vez, nunca se repetindo. Dessa forma, usaremos os resultados de [MOW2012] e [LAP2016] como balizadores para nossos próprios resultados.

6.5. Resultados

A coleta de dados foi realizada por um período de 11 dias. As vistas recorrentes eram controladas pelo protótipo através de cookies e HTML5 *local storage* de forma a computar o visitante e não a visita. Após isolados os visitantes únicos, fomos capazes de coletar um total de 64.086 assinaturas. Uma assinatura equivale a uma instância do *Tarp Fingerprinting* gerada por um visitante único. A tabela 1 sumariza todos os resultados obtidos em conjunto com as duas últimas linhas que indicam os mesmos parâmetros, quando disponíveis, dos trabalhos [MOW2012] e [LAP2016].

Tabela 1. Sumário dos Resultados Obtidos

Instruções Canvas	total de amostras	Entropia de Shannon	Entropia Ponderada	Assinaturas Distintas	Assinaturas Únicas
Tarp L	64086	2,43	0,152	45	11
Tarp H	64086	7,68	0,481	2449	1111
Tarp LH	64086	7,68	0,481	2449	1111
Tarp M	64086	7,86	0,492	2550	1091
Tarp LM	64086	7,86	0,492	2550	1091
Tarp HM	64086	7,91	0,495	2801	1317
Tarp LHM	64086	7,91	0,495	2801	1317
[MOW2012]	**294**	**5,73**	**N/A**	**N/A**	**N/A**
[LAP2016]	**118934**	**N/A**	**0,491**	**8375**	**5533**

As linhas Tarp L, Tarp H, e Tarp M indicam os resultados obtidos pela distribuição de assinaturas (hash MD5) das imagens geradas pelos grupos de instruções gráficas HTML5 <canvas> L, H e M respectivamente, ao longo de toda a amostra. A linhas compostas (LH, LM, HM, LHM) representam a aplicação dos mesmos cálculos a composições de hashes. Em HM, por exemplo, no lugar de ver isoladamente o *hash* de H como assinatura ou o *hash* de M como assinatura, usamos as dois *hashes* concatenados como assinatura da amostra.

6.6. Análise

Com o objetivo de responder aos questionamentos de nossos objetivos do item 6.1, faremos uma breve análise dos resultados obtidos.

Iniciaremos pelo primeiro questionamento com a análise das entropias de Shannon geradas por L, H e M. Como vemos, as entropias são, respectivamente, 2,43, 7,68 e 7,86. Como cada esquema produz assinaturas de entropia diferente, oscilando para mais e para menos, temos demonstrado que é sim possível variar a entropia como questionado no primeiro quesito. Isso, em última instância, significa dizer que é possível se fazer ajustes finos nos comandos gráficos do elemento <canvas> de forma a aumentar ou diminuir a entropia da assinatura gerada.

Em relação ao segundo questionamento, podemos analisar a composição das assinaturas de H e de M. Como podemos perceber, as assinaturas de M e H são, cada uma, inferiores à composição HM, que é de 7,91. Isso significa dizer que é possível modelar gráficos distintos com o elemento <canvas> que recebem componentes entrópicos não sobrepostos. Veja que o mesmo não ocorre quando associamos L e M. A entropia de LM é exatamente a mesma da de M isoladamente, significando que os elementos entrópicos de L já estão contidos em M. Vemos, assim, que a resposta ao quesito é positiva.

A respeito do terceiro questionamento, podemos analisar os extremos entrópicos que são L, com entropia de 2,43, e HM com entropia de 7,91. Vemos que L possui uma entropia bem baixa, pois é menos da metade do resultado de [MOW2012] e menos de um terço do resultado de [LAP2016]. Por sua vez, HM mostra alta entropia, sendo superior a [MOW2012] em mais de dois pontos em entropia de Shannon e superior a [LAP2016] em 0,004 pontos de entropia ponderada. Isso significa dizer que é possível gerar, pelo mesmo processo, imagens variam pouco entre as amostras e imagens que variam muito entre as amostras. A imagem com pouca variação deve se manter razoavelmente constante entre as amostras. No caso de L existem apenas 45 variações distintas entre todas as mais de 64 mil amostras. Logo, usar essa composição como marca d'água se configura viável, pois uma tentativa de burlar o esquema de *fingerprinting* modificaria também o resultado de L, gerando uma assinatura diferente das 45 levantadas pelo experimento. Isso automaticamente geraria um alerta de que determinada coleta de assinatura foi alvo de contramedida. Adicionalmente, tudo isso é feito simultaneamente à geração de outra assinatura com entropia alta, utilizável para rastreamento, segundo os melhores padrões da literatura. Dessa forma, a resposta ao questionamento 3 é positiva o que, por conseguinte, confirma a própria viabilidade da ideia por detrás de nossa proposta.

Outro ponto interessante é a similaridade de resultados quando comparamos nossos achados com os de [LAP2016]. Apesar de nosso estudo ter sido capaz de alcançar um incremento na entropia ponderada (0,004) em relação a [LAP2016], os dois trabalhos apontam para características semelhantes do Canvas Fingerprinting em termos de entropia. Contudo, se levarmos em conta que o estudo de [LAP2016] contém vícios estatísticos no sentido de favorecer usuários mais preocupados com privacidade, conforme narrado por seus autores, nossos resultados aparentam ser mais precisos. Outra vantagem de nossa proposta em relação a [LAP2016] é o tempo de levantamento de amostras. Enquanto nosso experimento foi capaz de levantar uma massa relevante de dados ao longo de dias, o experimento de [LAP2016] levou vários meses.

7. Conclusão

Neste artigo foi apresentada uma proposta de browser fingerprinting resistente a contramedidas que tentem evitar o rastreamento trazido pelo mecanismo. Trata-se de uma variação do Canvas Fingerprinting resistente a contramedidas que forjem ou falsifiquem a assinatura. Nossa técnica, chamada de TARP Fingerprinting, se vale da variação da entropia presente na geração de imagens da técnica do Canvas Fingerprinting. O mecanismo proposto explora a incapacidade de diferenciar imagens com alta entropia de imagens com baixa entropia por parte das técnicas de contramedida ao Canvas Fingerprinting. Ao invés de usar uma única imagem, como no modelo tradicional, nosso mecanismo usa o conteúdo integral de múltiplas imagens, variando a entropia, de forma a conseguir identificar um ataque que tente falsificar a assinatura. Ademais, apresentamos um dos dois únicos estudos existentes atualmente que medem a entropia da técnica de Canvas Fingerprinting em larga escala.

Referências

[NIK2013] Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., & Vigna, G. (2013, May). Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In Security and privacy (SP), 2013 IEEE symposium on (pp. 541-555). IEEE.

[UPA2015] Upathilake, R., Li, Y., & Matrawy, A. (2015, July). A classification of web browser fingerprinting techniques. In New Technologies, Mobility and Security (NTMS), 2015 7th International Conference on (pp. 1-5). IEEE.

[MOW2012] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in HTML5," in Proceedings of Web 2.0 Security & Privacy. IEEE, 2012.

[LAP2016] Pierre Laperdrix, Walter Rudametkin, Benoit Baudry. Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints. 37th IEEE Symposium on Se- curity and Privacy (S&P 2016), May 2016, San Jose, United States.

[ECK2010] P. Eckersley, "How Unique Is Your Browser?" in 10th Privacy Enhanc- ing Technologies Symposium, 2010.

[MUL2013] M. Mulazzani, P. Reschl, M. Huber, M. Leithner, S. Schrittwieser, and E. Weippl, "Fast and Reliable Browser Identification with JavaScript Engine Fingerprinting," in Proceedings of Web 2.0 Security & Privacy. IEEE, 2013.

[SPU2016] Sputnik. Disponível em https://code.google.com/p/sputniktests/. Acessado em Maio de 2016.

[APP2016] ApPodroMe.net "CanvasFingerprintBlock: Protect your privacy. Prevent webpages from tracking you by your browser's HTML canvas fingerprint". Disponível em https://chrome.google.com/webstore/ Acessado em maio de 2016.

[KKA2016] kkapsner, "Plugin CanvasBlocker para FireFox". Disponível em https://github.com/kkapsner/CanvasBlocker/. Acessado em Maio de 2016.

[ACA2014] Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., & Diaz, C. (2014, November). The web never forgets: Persistent tracking mechanisms in the wild. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (pp. 674-689). ACM.

[LIU2012] Liu, L., Zhang, X., Yan, G., & Chen, S. (2012, February). Chrome Extensions: Threat Analysis and Countermeasures. In NDSS.

[WAN2012] Wang, J., Li, X., Liu, X., Dong, X., Wang, J., Liang, Z., & Feng, Z. (2012). An empirical study of dangerous behaviors in firefox extensions. In Information Security (pp. 188-203). Springer Berlin Heidelberg.

[PEN2000] W. Peng and J. Cisna, "Http cookies - a promising technology," Online Information Review, pp. 150–153, 2000.

[VAL2016] Valve, "fingerprintjs2: Modern & flexible browser fingerprinting library". Disponível em https://github.com/Valve/fingerprintjs2. Acessado em Maio de 2016

[KHA2014] Amin Faiz Khademi. (2014). Browser Fingerprinting: Analysis, Detection, and Prevention at Runtime. Master Thesis. Queen's University, Ontario, Canada.

[CAR2014] CARNEIRO, Brito; FEITOSA, Eduardo Luzeiro. Device Fingerprinting: Conceitos e Técnicas, Exemplos e Contramedidas. In: Minicursos do XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais — SBSeg 2014

IBEMCS: IDS Baseado em Eventos Multi-Contexto para SCADA

Anderson Mussel D'Aquino[13], Luiz Fernando Rust da Costa Carmo[12], Luci Pirmez[1], Claudio Miceli[1]

[1]Programa de Pós-Graduação em Informática - Universidade Federal do Rio de Janeiro – Cidade Universitária – 21.941-901 – Rio de Janeiro – RJ – Brasil

[2]Instituto Nacional de Metrologia, Normalização e Qualidade Industrial – 25.250-020 – Duque de Caxias – RJ – Brasil

[3]Petrobras Transportes S.A. – Transpetro – 20.091-060 – Rio de Janeiro – RJ – Brasil

anderson.aquino@petrobras.com.br,
{rust,luci,claudiofarias}@nce.ufrj.br

Abstract. *Nowadays security mechanisms used for intrusion detection in industrial automation environment use either ICT (Information and Communication Technologies) or OT (Operation Technologies) data. This work proposes an IDS that integrates ICT and OT in order to identify the ICT and OT chain of events that leaded to industrial process flaw. Measurements show that runtime and hit are linearly proportional to the data volume processed allowing a prior planning of this framework to the work environment.*

Resumo. *Atualmente os mecanismos de segurança utilizados para detecção de intrusão em ambientes de automação industrial utilizam dados exclusivos TIC (Tecnologia da Informação e Comunicação) ou TO (Tecnologia da Operação). Este trabalho propõe um IDS (Intrusion Detection System) que integra informações TIC e TO para identificação das cadeias de eventos TIC e TO que ocasionaram a falha no processo industrial. Medições demonstram que o tempo de resposta e a taxa de acerto são linearmente proporcionais ao volume de dados processados permitindo um planejamento prévio da arquitetura ao ambiente de trabalho.*

1. Introdução

Sistemas SCADA (*Supervisory Control and Data Acquisition*) são utilizados para operar processos industriais automatizados e exercem um papel fundamental no controle das infraestruturas críticas, tais como: geração e transmissão de energia, saneamento, usinas nucleares, sistemas bélicos, dentre outros. Uma falha em tais estruturas pode causar desde graves impactos econômicos até sério comprometimento da soberania nacional. Devido à esta importância, muita pesquisa tem sido feita para ampliação dos mecanismos de segurança aplicados a sistemas SCADA. Esta pesquisa está concentrada na construção de IDS (*Intrusion Dectection System*) customizados para ambientes SCADA. As propostas destes IDS tendem a criar uma segregação nos dados processados e trabalham ou com dados puramente de automação ou com dados TIC, sem levar em consideração a relação existente entre o funcionamento do processo industrial controlado pelo sistema SCADA e a infraestrutura TIC.

A operação dos sistemas SCADA é feita por operadores, que são técnicos de automação com conhecimento sobre o processo industrial e treinados para utilizar as funções do sistema SCADA para garantir a execução do processo industrial, identificar anomalias e atuar de forma a levar o sistema SCADA à condição normal de operação. Ao conjunto formado pelo sistema SCADA, operadores, CLP (Controlador Lógico Programável) e instrumentação, chamamos de TO (Tecnologia da Operação).

A utilização de elementos da infraestrutura TIC é feita em todos os níveis da arquitetura SCADA. No centro de controle as funções do sistema SCADA são hospedadas em computadores e sistemas operacionais comerciais. Nos sítios remotos, os CLPs possuem interface Ethernet. A comunicação entre o centro de controle e os sites remotos é feita através de tecnologia IP (*Internet Protocol*).

Toda esta infraestrutura TIC é operada através de um NOC (*Network Operation Center*) por técnicos que não possuem conhecimento sobre a lógica do processo automatizado e utilizam ferramentas TIC que não interpretam os dados e eventos do processo automatizado. Assim como a operação da infraestrutura TIC é segregada da operação TO, a pesquisa científica para mecanismos de segurança de sistemas SCADA também adota este mesmo critério de segregação e como consequência os trabalhos propostos focam na proteção exclusiva TIC ou TO.

Esta segregação de informações entre os contextos TIC e TO gera uma lacuna por não permitir a correlação de anomalias do processo automatizado com anomalias da infraestrutura TIC. Desta forma um atacante pode conseguir infiltrar-se na infraestrutura TIC explorando uma falha do mecanismo de segurança TIC, causar um desvio de funcionalidade no processo automatizado burlando o mecanismo de segurança TO e perpetrar um ataque uma vez que não há troca de informações entre as equipes e ferramentas TIC e TO que possa identificar o ataque. Este tipo de ataque é praticado por atacantes com interesse em controlar o processo industrial. A infraestrutura TIC não é o alvo do ataque e sim o vetor utilizado para atingir o processo industrial, estuda-lo mapear suas falhas e planejar a melhor maneira de impedir o processo industrial de atingir seus objetivos durante o maior período de tempo sem ser percebido. Exemplo clássico de ataque deste tipo é o Stuxnet, que foi largamente estudado, como pode ser visto em [Falliere et al. 2011].

Este trabalho propõe um IDS para proteção do processo industrial controlado por um sistema SCADA, denominado IBEMCS (IDS Baseado em Eventos Multi-Contexto para SCADA), que estende as propostas atuais, permitindo a integração entre IDS que processam informações de contexto TIC e IDS que processam informações de contexto TO como forma ampliar o espectro de detecção destes IDS isolados através da geração de alarmes que identifiquem a anomalia TO e os eventos TIC responsáveis por tal anomalia. Este alarme, que une informações TO e TIC, permite uma análise forense mais ágil por identificar o momento e a forma como a infraestrutura TIC causou a anomalia TO. Análise de dados simulados demonstram que tanto o tempo de reposta do IDS proposto quanto a taxa de acerto são linearmente proporcionais ao volume de dados processados, permitindo um planejamento de capacidade do IDS para o ambiente em que será utilizado.

Este trabalho está organizado como segue. Na seção 2, são apresentados os trabalhos recentes relacionados a IDS para sistema SCADA. Na seção 3 é apresentada a

proposta detalhada. Na seção 4 é apresentado um estudo de caso. Na seção 5 são apresentados os experimentos e simulações realizados e por fim, na seção 6, a conclusão.

2. Trabalhos Relacionados

A grande maioria das propostas atuais para construção de IDS para sistemas SCADA utilizam dados de contextos mutuamente exclusivos TIC ou TO.

Dentre os trabalhos que utilizam dados exclusivos TIC, podemos citar [Sayegh et al. 2014] que utilizam um método estatístico para determinar o padrão de comunicação baseado nos metadados contidos nos níveis 2 e 3 de rede, [Maglaras et al. 2014] que utilizam OCSVM para traçar o perfil de comunicação dos protocolos de rede e [Ponomarev et al. 2016] que fazem um estudo de 10 classificadores estatísticos diferentes para mapeamento do padrão de trafego entre o sistema SCADA e CLP. Todos os trabalhos mencionados propuseram seus IDS através de máquina de aprendizado para identificação do padrão de comunicação de rede do sistema SCADA e geração de alarmes caso a comunicação desvie do padrão esperado.

Ainda utilizando informações exclusivas TIC podemos citar os trabalhos que utilizam colaboração de agentes distribuídos como forma de diversificação e intensificação dos mecanismos de detecção de anomalia. [Shosha et al. 2011] propõem uma arquitetura de agentes distribuída para coleta de eventos. Cada perímetro tem um concentrador de eventos local e existe um concentrador de eventos global para coordenação de todos os agentes locais. São designados agentes específicos para cada protocolo de aplicação utilizado na arquitetura SCADA. [Schuster et al. 2012] também utilizam o conceito de colaboração de agentes distribuídos, no entanto utilizam uma arquitetura descentralizada e sem agentes especializados, que gera maior trafego e causa uma maior dependência da rede. [Cruz et al. 2014] ultrapassam a fronteira dos eventos de rede providos pelos NIDS e integram os eventos de host providos pelo HIDS permitindo análise de eventos do sistema operacional, aplicações, antivírus e etc expandindo o campo de visão do IDS e permitindo uma análise de anomalia mais profunda.

Todos os trabalhos citados focam na proteção da infraestrutura TIC, diferentemente do IBEMCS que foca na proteção do processo industrial e monitora a infraestrutura TIC para impedir que um ataque consiga controlar o processo industrial.

Já no contexto TO, os trabalhos propostos utilizam mecanismos de aprendizagem para traçar o perfil de utilização do sistema SCADA através de eventos TO e identificar anomalias para geração de alarmes TO.

[Goldenberg et al. 2013] propõem uma máquina de aprendizado para mapeamento dos estados de um sistema SCADA através de um autômato finito. Esta máquina de aprendizado é utilizada para detecção de ataques através de anomalias. [Fovino et al. 2012] propõem um mecanismo para detecção de anomalias de processo através da distância Manhattan entre o estado do sistema SCADA e seus estados críticos pré mapeados. A distância entre estados é definida pela quantidade de variáveis que diferem entre os estados. [Almalawi et al. 2016] propõem uma máquina de aprendizado baseado no algoritmo de regressão *k-nearest neighbourhood* para mapeamento dos

estados normais do sistema SCADA e detecção de anomalia para geração de alarmes. As anomalias são estados que não se enquadram no padrão estabelecido na fase de aprendizado do algoritmo. [Carcano et al. 2011] propõem uma linguagem para descrever os estados do sistema SCADA e a partir desta formalização, identificar os estados críticos e monitorar a evolução dos estados do sistema SCADA para geração de alarmes quando tal evolução direcionar para um estado crítico.

Além de nenhum dos trabalhos citados focados em eventos TO definir um formato padrão que descreva um alarme de forma a permitir integração com sistemas externos, nenhum deles correlaciona a falha TO com a causa proveniente da infraestrutura TIC dificultando ou impossibilitando a adoção de medidas para prevenir e impedir o problema.

3. Proposta

A proposta deste trabalho é a criação de um IDS capaz de garantir a segurança da execução de um processo industrial contra ataques que explorem vulnerabilidades na infraestrutura TIC capazes controlar o processo industrial de forma adversa do planejado. Este IDS utiliza o mecanismo de assinatura para a identificação do ataque através do mapeamento entre as anomalias TO e a sequência de eventos TIC que causaram tal anomalia.

Uma das vantagens da identificação de anomalias através de assinaturas é a facilidade de propagação da assinatura, depois que a anomalia tem seu comportamento estudado e mapeado, devido a integração entre bases de dados públicas de vulnerabilidade, como por exemplo (CVE), (NVD) (ambas com fomento do governo norte americano), (Quickdraw SCADA IDS), e ferramentas como Snort, que são referências de tais bases de dados de vulnerabilidade na criação e disponibilização de suas respectivas assinaturas. Outra vantagem é a alta taxa de acerto, uma vez que as assinaturas são criadas para identificar comportamentos específicos da anomalia. Mecanismos de detecção capazes identificar ataques que exploram vulnerabilidades *zero-day* sofrem de problemas com taxa de acerto além de não conseguirem compartilhar informações publicamente assim como as assinaturas.

O funcionamento do IDS proposto utiliza uma base de dados de assinaturas construída por especialistas com conhecimento da infraestrutura TIC e do processo industrial que se deseja proteger, a partir de informações coletadas sobre anomalias ocorridas na planta industrial sob controle ou de bases de dados públicas, como as bases de dados TIC citadas acima, e bases de dados TO, como RISI. Cada assinatura desta base de assinaturas define uma associação entre eventos TIC e TO provenientes de IDS TIC e TO externos. As assinaturas são do tipo *condição→alarme* onde a condição representa um conjunto de restrições, baseadas nos eventos TIC e TO, necessárias para a geração de um alarme. O IDS proposto tem duas entradas de dados: Uma para recebimento de eventos TIC e outra para eventos TO. Diferente dos eventos TIC, que contam com padrões consagrados para sua representação, não há padrão para representação dos eventos TO, desta forma é proposto um formato para representação destes eventos assim como um componente conversor que pode ser utilizado caso a entrada de dados não esteja no padrão proposto.

Propomos o formato do evento TO como um conjunto de pares no formato *{chave=valor}*, assim como os eventos tradicionais TIC, como pode ser visto na Figura 1 que é parte de um evento gerado pelo IDS (snort) através da aplicação da Regra snort 1. Neste formato, *chave* identifica univocamente um campo dentro de um evento e *valor* é a informação associada à chave.

```
"Evento TIC" : {
        "dport-icode" : 102,
        "source-ip" : "192.168.251.1",
        "EventoTIC_ID" : 1201502,
        "sport-itype" : 53151,
        "event-second" : 1466369924,
        "destination-ip" : "192.168.251.2",
        "signature-revision" : 0,
        "event-microsecond" : 69146,
        "blocked" : 0
        }
```

Figura 1. Representação JSON do evento TIC detectado pela Regra snort 1

No formato do evento TO proposto, três campos são definidos como obrigatórios: *ID* que é um identificador utilizado para diferenciar os eventos TO entre si, *Mensagem* que é um campo texto com uma mensagem descritiva do evento e *Timestamp* que registra o momento da geração do evento. Além destes três campos obrigatórios, o evento TO contém outros campos, preenchidos pelo IDS TO, que transmitem informações sobre o estado do sistema SCADA relacionadas ao evento. A estrutura de um evento TO pode ser visualizada na Tabela 1.

Tabela 1. Visão de um Evento TO.

ID	Tipo numérico. Chave primária.
Mensagem	Tipo texto. Descrição do evento.
Timestamp	Tipo data/hora. Registro do momento do evento.
Campo 1	Valor proveniente do IDS TO.
Campo 2	Valor proveniente do IDS TO.
⋮	⋮
Campo N	Valor proveniente do IDS TO.

A arquitetura conta ainda com um sistema de decisão, que é responsável por orquestrar o funcionamento dos demais componentes para junção das informações TIC e TO e posterior geração do alarme.

3.1. Arquitetura lógica

A Figura 2 ilustra a arquitetura lógica do IDS, que é composta de um conjunto de componentes e uma base de dados. Os componentes são: **Conversor de Eventos TO**, **Gestor de Eventos TO**, **Gestor de Eventos TIC**, **Gestor de Decisão**, **Correlacionador** e **Gestor de Alarmes**. A base de dados é a **Base de Assinaturas**.

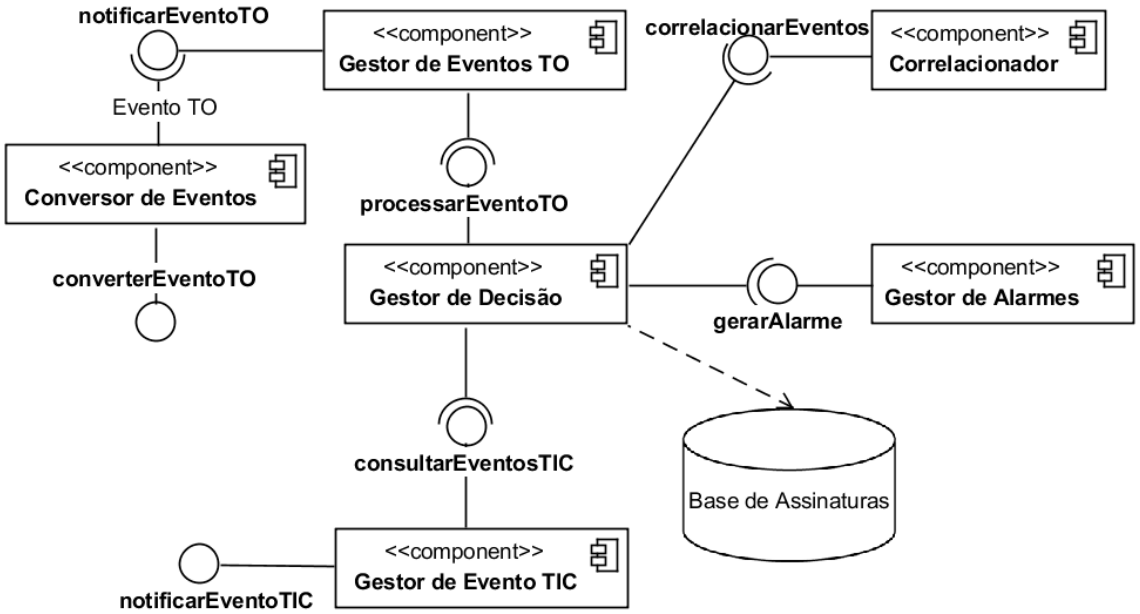


Figura 2. Arquitetura IBEMCS.

3.2. Base de Assinaturas

O IDS proposto utiliza o mecanismo tradicional de assinaturas onde os alarmes são gerados quando um conjunto de restrições é satisfeito. Como um exemplo de regra podemos citar a Regra snort 1 para identificação e registro de um evento de acesso de rede a um CLP na porta 102 TCP.

```
log tcp any any -> 192.168.251.2 102 ( sid:1201502; msg: "Siemens Step7 conexão de manutenção estabelecida"; )
```

Regra snort 1. Regra para geração de alarme TIC

As assinaturas podem ser entendidas como a expressão lógica *condição→alarme* armazenadas na **Base de Assinaturas** que são consultadas pelo componente **Gestor de Decisão**. Os elementos da condição necessários para geração do alarme são:

- **EventoTIC$_{ID}$**: Este campo representa o ID do evento TIC gerado pelo IDS TIC que deve ser condicionado ao Evento TO para geração do alarme.
- **EventoTO$_{ID}$**: Este campo representa o ID do evento TO gerado pelo IDS TO que deve ser condicionado ao Evento TIC para geração do alarme. Diferentemente dos Eventos TIC, não há padrão estabelecido para representação dos eventos TO, portanto, adotaremos o formato proposto neste trabalho.
- **TimeWindow**: Este campo representa uma restrição de localidade temporal entre os eventos TO e TIC mapeados na condição da assinatura. *TimeWindow* é um número do tipo real que representa a máxima diferença de tempo, em segundos, decorrido entre a geração dos eventos TO e TIC para validação de associação entre esses eventos. Este campo é fornecido pelo componente **Gestor de Decisão** como filtro para o componente **Gestor de Eventos TIC** durante a chamada do método *consultarEventosTIC*.
- **ValidFunc**: Este campo representa uma restrição de localidade espacial entre os eventos TO e TIC mapeados na condição da assinatura. *ValidFunc* é uma função

de validação que deve ser programada em linguagem de programação pelo especialista que desenvolveu a assinatura com a finalidade de garantir que os eventos TIC e TO estão relacionados aos mesmos elementos de rede da planta industrial. O componente **Correlacionador** avalia a função de validação com os dados dos eventos TO e TIC fornecidos pelo componente **Gestor de Decisão**.

O parâmetro *TimeWindow* influencia diretamente no poder de processamento necessário para a utilização do IDS, pois controla o volume de dados que o IDS irá processar em cada requisição de correlaciomento de eventos. O ajuste desde parâmetro depende de estudo do especialista pela implantação do IDS, que deve levar em consideração o perfil de vazão de geração de eventos da planta industrial para o correto dimensionamento dos recursos para implantação do IDS.

ValidFunc é responsável pela validação da correlação entre os eventos através de informações contidas nos próprios eventos. A complexidade de tais validações influencia no tempo de processamento do IDS.

O alarme da assinatura conterá as informações TIC e TO necessárias que permitam aos técnicos de segurança analisarem e tratarem o alarme. Essas informações são únicas de cada alarme e especificadas pelo especialista responsável pelo planejamento do alarme.

Uma assinatura pode ser descrita como $(EventoTIC_{ID}, EventoTO_{ID}, TimeWindow, ValidFunc) \rightarrow (CampoTIC_1, ..., CampoTIC_N, CampoTO_1, ..., CampoTO_N)$, onde $CampoTIC_i$ é um dos campos contidos no evento TIC mapeado na condição da assinatura do alarme e $CampoTO_i$ é um dos campos contidos no evento TO mapeado na condição da assinatura do alarme.

3.3. Conversor de Eventos TO

O funcionamento do IBEMCS depende de uma entrada de dados padronizada, tanto de eventos TIC quanto de eventos TO. Este componente é um padrão de projeto *Adapter* para uso do IDS TO que não seja capaz de gerar eventos no formato Evento TO. Este componente define as interfaces que devem ser implementadas para geração de eventos em formato evento TO. A implementação destas interfaces é especifica para cada sistema que forneça dados para este componente.

3.4. Gestor de Eventos TO

Este componente é responsável por gerenciar o recebimento concorrente de eventos TO e acionar o componente Gestor de Decisão para iniciar o processo de análise de dados para geração do alarme.

3.5. Gestor de Decisão

Este componente é responsável pelo recebimento da entrada de dados, Eventos TO, e obtenção de dados necessários para a correlação de eventos realizado pelo componente **Correlacionador**. Seu funcionamento pode ser descrito como a seguinte sequência de ações, visualizada na Figura 3:

1. Recebimento do Evento TO;

2. Consulta à **Base de Assinaturas** para obtenção das assinaturas relacionadas ao Evento TO recebido;
3. Consulta ao componente **Gestor de Eventos TIC** para obtenção da lista de Eventos TIC que deve ser processada pelo componente **Correlacionador**. Como filtro de pesquisa, o componente **Gestor de Eventos TIC** recebe os parâmetros *TimeWindow* e $EventoTIC_{ID}$ presentes na assinatura obtida no passo 2;
4. Acionamento do componente **Correlacionador**, informando o Evento TO obtido no passo 1, a assinatura obtida no passo 2 e a lista de Eventos TIC obtida no passo 3.

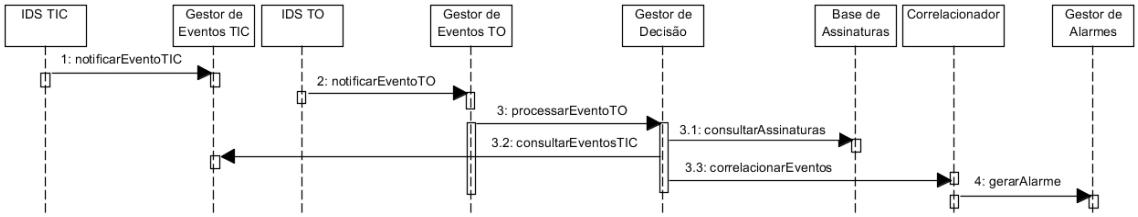


Figura 3. Diagrama de sequência do IBEMCS.

3.6. Gestor de Eventos TIC

Este componente é responsável por responder às solicitações do componente **Gestor de Decisão** para obtenção de Eventos TIC segundo os filtros presentes nas assinaturas da **Base de Assinaturas**. A consulta é feita através da interface *consultarEventosTIC* cujos parâmetros são *TimeWindow* e $EventoTIC_{ID}$ que indicam respectivamente a janela de tempo a ser considerada para pesquisa e o identificador do Evento TIC que de ser pesquisado.

3.7. Correlacionador

A função deste componente é avaliar os dados contidos nos Evento TO e Evento TIC para validar a geração do alarme. Esta avaliação é feita através da função de validação *ValidFunc* contida na assinatura do alarme. O objetivo desta função é garantir que o Evento TIC que está sendo avaliado para geração do alarme de fato afetou os componentes de automação envolvidos na geração do Evento TO que está sendo analisado. Caso este teste não seja feito poderíamos gerar um alarme a partir de eventos TO e TIC que aconteceram em momentos próximos, mas em localidades diferentes, como por exemplo o Evento TO acontecido no Brasil e o Evento TIC acontecido na Espanha. Caso a topologia de rede do sistema SCADA sendo monitorado permita que eventos ocorridos em regiões geográficas distintas tornem válida a geração de um alarme, a função de validação *ValidFunc* deve processar dados adicionais de Evento TO e/ou Evento TIC para tornar tal lógica válida.

Como exemplo, podemos definir uma função de validação espacial chamada *ValidarIP*, definida no Algoritmo 1, que realiza a validação espacial entre os eventos TIC e TO baseado nas informações de rede do dispositivo que gerou o evento.

Algoritmo 1. Algoritmo para validação da localidade espacial entre os Eventos TO e TIC

```
1. Função ValidarIP(EventoTIC, EventoTO)
```



```
2.    Se EventoTIC.IP == EventoTO.CLPAddress
3.       Então
4.          retorne Verdadeiro
5.       Senão
6.          retorne Falso
```

3.8. Gestor de Alarmes

Este componente é responsável pela geração de um alarme como descrito na assinatura cadastrada na **Base de Assinaturas**. O componente **Correlacionador**, após executar todas as validações necessárias para geração do alarme, invoca o componente **Gestor de Alarmes** fornecendo a assinatura e os Eventos TIC e TO que foram correlacionados para geração do alarme. O componente **Gestor de Alarmes** identifica no componente alarme da assinatura, os campos dos eventos TIC e TO que devem ser preenchidos no alarme.

4. Estudo de caso

A **Figura 4** ilustra parte de um processo industrial controlado por um sistema SCADA e executado no centro de controle de uma Transportadora de Gás Natural. Neste exemplo estão representados o centro de controle, onde está situado o sistema SCADA, e um dos muitos sítios remotos onde ocorre parte do processo industrial. O sítio representa um trecho de um gasoduto com três válvulas controladas por um mesmo CLP. O trecho representado no exemplo não contém nenhuma junção ou derivação, portanto os valores medidos de pressão e vazão nos sensores das três válvulas devem coincidir.

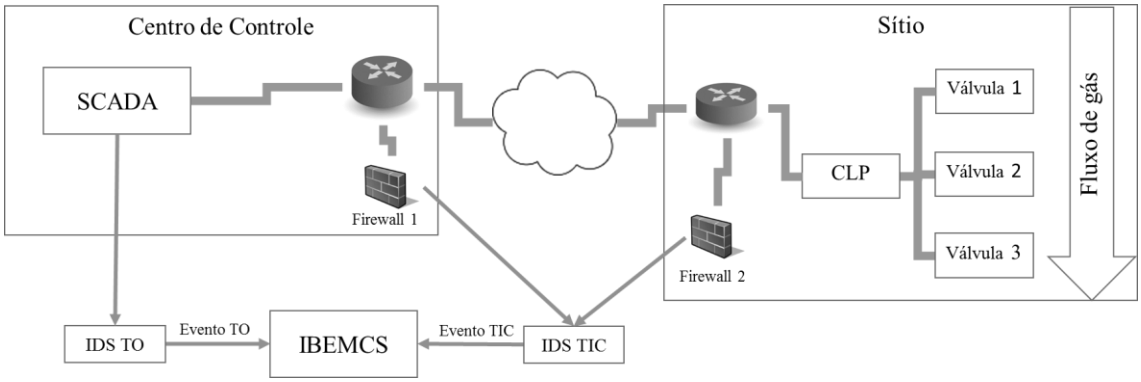


Figura 4: Estudo de caso em um processo industrial.

O acesso ao CLP é feito remotamente tanto pelo sistema SCADA para realização das tarefas de supervisão e controle executadas no centro de controle com finalidade de operação quanto por sistemas de suporte executados pela equipe de manutenção no próprio sítio. Os dados a seguir são baseados em dados de supervisão de uma Transportadora de Gás Natural.

Em um instante t_0 a visão do operador, no centro de controle, das variáveis pressão e vazão das três válvulas pode ser resumida pela Tabela 2.

Tabela 2. Situação normal de operação.

	Pressão (Kgf/cm^2)	Vazão (m^3/dia)
Válvula 1	77,7	2.721.700
Válvula 2	77,7	2.721.700
Válvula 3	77,7	2.721.700

Em um instante $t_2>t_0$ há uma alteração no estado do processo automatizado e o operador a passa visualizar os valores de pressão e vazão representados na Tabela 3.

Tabela 3. Situação anormal de operação.

	Pressão (Kgf/cm^2)	Vazão (m^3/dia)
Válvula 1	77,7	2.721.700
Válvula 2	71,3	2.962.683
Válvula 3	77,7	2.721.700

Um Evento TO é gerado e registrado no IBEMCS devido ao estado anormal do sistema SCADA. A configuração deste Evento TO pode ser visualizada na Tabela 4.

Tabela 4. Evento TO que representa estado SCADA anormal.

Campo	Valor
ID	1001
Mensagem	Inconsistência de valor em linha.
Timestamp	01/01/2015 17:23:31.100
V1Pressao	77,7
V2Pressao	71,3
V3Pressao	77,7
V1Vazao	2.721.700
V2Vazao	2.962.683
V3Vazao	2.721.700
CLPAddress	172.15.52.1

A alteração dos valores de pressão e vazão na válvula 2 é um comportamento não esperado e requer análise do operador TO. Diante de tal anomalia, o operador deve tomar uma ação para corrigir o estado do sistema SCADA, que pode ser:

- Enviar comando ao CLP para equilibrar as três válvulas;
- Acionar a equipe de manutenção do campo para inspecionar a válvula 2 e o CLP;
- Acionar a equipe de Engenharia de automação para inspecionar o sistema SCADA.

A anomalia é tratada sob o ponto de vista exclusivo de automação. As equipes envolvidas foram de automação e os dados coletados para análise da anomalia são de automação.

Em um instante $t_1<t_2$, $t_1>t_0$ acontece um evento percebido pela equipe TIC. Um acesso remoto ao CLP através da VPN de manutenção é realizado e registrado nas ferramentas de monitoração da infraestrutura TIC. Este acesso, no entanto, não chama atenção da equipe de segurança TIC pois é uma operação prevista e tem seu controle de acesso liberado.

O IDS TIC está planejado para registrar os eventos de manutenção no CLP e por isso gera um Evento TIC que é registrado no IBEMCS. A configuração deste Evento TIC pode ser visualizada na Tabela 5.

Tabela 5. Evento TIC de manutenção no CLP.

Campo	Valor
ID	4365
Mensagem	Acesso TCP na porta de manutenção.
Timestamp	01/01/2015 17:21:57.085
IP	172.15.52.1
Porta	502

A análise dos eventos ocorridos no processo automatizado e na infraestrutura TIC de forma isolada impossibilita a criação de nexo causal entre tais eventos e cria uma fragilidade que pode ser explorada de forma maliciosa contra o processo automatizado.

O IBEMCS provê os meios para preencher esta lacuna de comunicação entre os técnicos de automação e os técnicos de TIC através da geração de alarmes que contenham informações TIC e TO correlacionadas à anomalia.

Neste caso em estudo, para que a anomalia seja identificada, a Assinatura 1 deve ser cadastrada na Base de Assinaturas do IBEMCS. A condição da assinatura correlaciona o Evento TIC de ID 4365, acesso ao CLP, com o Evento TO de ID 1001, inconsistência de valores das três válvulas em sequência, em uma janela de tempo de 120s. Para validação espacial, utilizamos a função *ValidarIP* descrita no Algoritmo 1.

Assinatura 1. Assinatura para geração de alarme que correlaciona inconsistência de medição nas válvulas e acesso de manutenção no CLP.

(4365, 1001, 120, ValidarIP)→(EventoTIC.ID, EventoTIC.Mensagem, EventoTIC.IP, EventoTIC.Porta, EventoTO.ID, EventoTO.Mensagem, EventoTO. Timestamp, EventoTO.V2Pressao, EventoTO.V2Vazao)

5. Dados experimentais

No âmbito do presente trabalho, o IDS proposto foi implementado na linguagem Python. Para criação do ambiente de simulação foi desenvolvido um aplicativo auxiliar cuja finalidade é controlar a quantidade de eventos disponibilizados para processamento pelo IDS. Este aplicativo é multithread e a quantidade de threads é controlada em cada ensaio para controlar a vazão de dados desejada. Cada thread registra um número fixo de 4 eventos no IBEMCS, sendo 3 eventos TIC e 1 evento TO.

Os experimentos foram conduzidos com uma variação do número de threads de 1 a 400, em intervalos de 10, refletindo em uma variação na quantidade eventos simultâneos de 4 a 1600, em intervalos de 40. Cada experimento foi repetido 30 vezes, para obter um intervalo de confiança de 95%.

Os eventos TIC considerados para análise são eventos de rede gerados pelo IDS Snort, como resultado da aplicação da Regra snort 1. O evento TO selecionado foi o evento descrito na Tabela 4 do estudo de caso apresentado. A ferramenta (Elasticsearch) é utilizada para gerenciar a base de dados de eventos que foi populada com 6 milhões de registros em um período de 30 dias durante a realização dos testes. Não houve descarte de eventos.

As métricas utilizadas na avaliação são **Tempo de Resposta** e **Taxa de Acerto**. Tempo de resposta é definido como o tempo decorrido desde o recebimento do evento TO pelo IBEMCS até a geração do alarme. Taxa de Acerto é definido como a relação entre a quantidade de alarmes gerados (verdadeiros positivos) e a quantidade de anomalias processadas.

O IBEMCS, o aplicativo auxiliar gerador de eventos e a base de dados (Elasticseach) são executados em uma máquina virtual Ubuntu Linux 14.04 com dois processadores 2.30Ghz e 2.0GB RAM.

5.1. Tempo de Resposta

O estudo do tempo de resposta dos sistemas de segurança é fundamental para determinar a adequação de tais sistemas aos cenários onde serão utilizados. Este experimento foi

conduzido para estudar a evolução do tempo do IDS proposto em função do volume de dados processados.

Foram coletados dados para *TimeWindow* de 2,5s, 5,0s e 7,5s [Aramaki 2015]. O parâmetro *TimeWindow* influencia na quantidade de eventos TIC que será correlacionada ao evento TO que gerou o processo de análise.

A análise dos dados mostra que os tempos médio e máximo de resposta (Figura 5 e Figura 6, respectivamente) são lineares em função do volume de dados processados e não houve influência perceptível de *TimeWindow* nos cenários apresentados.

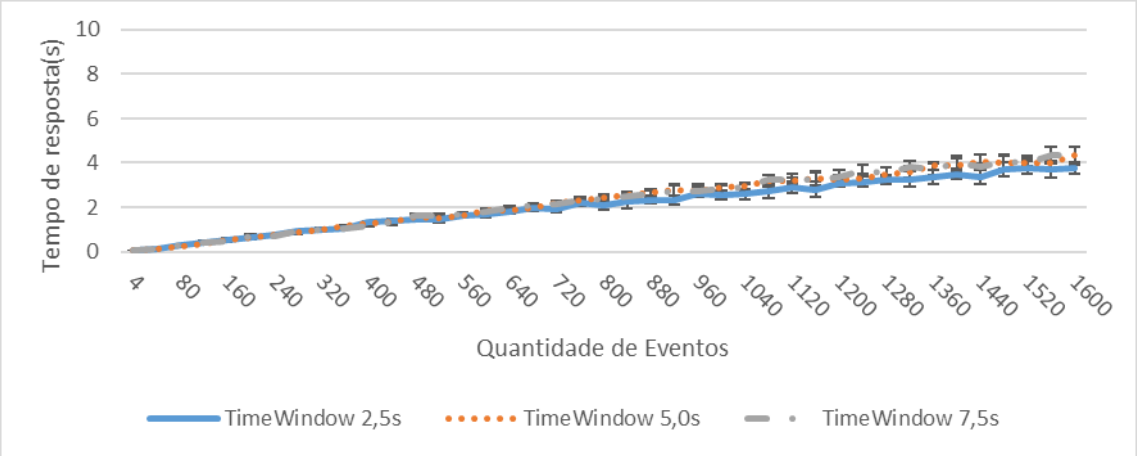


Figura 5. Tempo médio de resposta em função do número de eventos

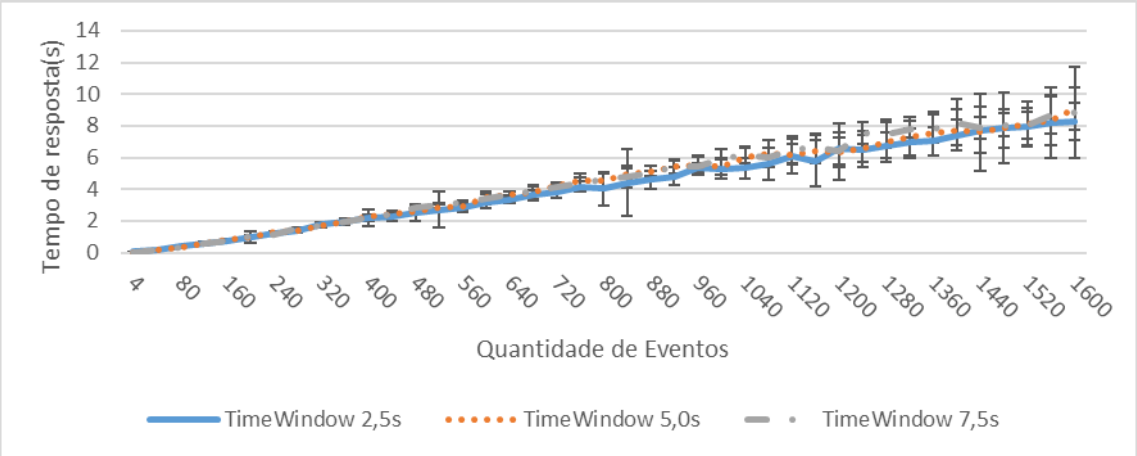


Figura 6. Tempo máximo de resposta em função do número de eventos

5.2. Taxa de Acerto

A Figura 7 mostra o gráfico da taxa de acerto do IBEMCS em função do número de eventos processados. Através do gráfico é possível perceber que a quantidade de eventos tem influência direta sobre a taxa de acerto. O aumento do número de eventos causa um aumento no tempo de resposta do IBEMCS. Este aumento no tempo de resposta tem impacto direto na taxa de acerto pois causa um deslocamento da janela de tempo considerada para correlacionamento de eventos fazendo com que os eventos TIC

percam a condição de associação aos eventos TO determinados pelo parâmetro *TimeWindow*.

O número de eventos que começa a afetar a taxa de acerto é 400, 1000 e 1400 para *TimeWindow* de 2,5s, 5,0s e 7,5s respectivamente, pois nestes limites o tempo máximo de resposta aproxima-se do valor *TimeWindow* tendo início o declínio da taxa de acerto do IBEMCS. A partir destes limites o aumento do número de eventos tem efeito inversamente proporcional sobre a taxa de acerto.

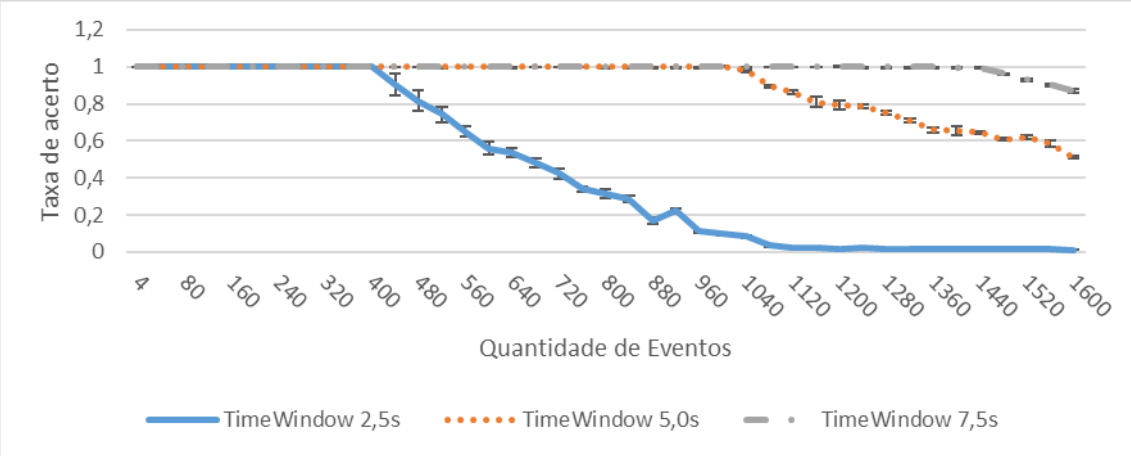


Figura 7. Taxa de acertos em função do número de eventos

6. Conclusão

A segurança das infraestruturas críticas depende dos mecanismos de segurança empregados para controle da lógica do processo automatizado e também da infraestrutura TIC. Atualmente existe uma lacuna nos mecanismos de segurança propostos para infraestruturas críticas. Esta lacuna é causada pela visão mutuamente exclusiva entre TIC e TO e impede correlacionar eventos TIC com TO como forma de enriquecimento de informações para aumento da acurácia dos mecanismos de detecção de intrusão. Este trabalho propõe um IDS que permite o correlacionamento de informações TIC e TO para construção de mecanismos de detecção de intrusão. O estudo do desempenho da arquitetura proposta demonstra que ela é fácil de ser ajustar a capacidade do sistema a ser monitorado pois o tempo de resposta é uma função linear do volume de informação demandada o que permite planejar também a taxa de acerto do mecanismo pois esta pode ser determinada em função do volume de informações processadas.

7. Referências

Almalawi, A., Fahad, A., Tari, Z., Alamri, A., Alghamdi, R., & Zomaya, A. Y. (2016). An Efficient Data-Driven Clustering Technique to Detect Attacks in SCADA Systems. IEEE Transactions on Information Forensics and Security, 11(5), 893–906. doi:10.1109/TIFS.2015.2512522

Carcano, A., Coletta, A., Guglielmi, M., Masera, M., Fovino, I. N., & Trombetta, A. (2011). A Multidimensional Critical State Analysis for Detecting Intrusions in

SCADA Systems. Industrial Informatics, IEEE Transactions on, 7(2), 179–186. doi:10.1109/TII.2010.2099234

Goldenberg, N., & Wool, A. (2013). Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems. International Journal of Critical Infrastructure Protection, 6(2), 63–75. doi:10.1016/j.ijcip.2013.05.001

Maglaras, L. A., & Jiang, J. (2014). Intrusion detection in SCADA systems using machine learning techniques. In 2014 Science and Information Conference (pp. 626–631). doi:10.1109/SAI.2014.6918252

Nai Fovino, I., Coletta, A., Carcano, A., & Masera, M. (2012). Critical state-based filtering system for securing SCADA network protocols. IEEE Transactions on Industrial Electronics, 59(10), 3943–3950. doi:10.1109/TIE.2011.2181132

Sayegh, N., Elhajj, I. H., Kayssi, A., & Chehab, A. (2014). SCADA Intrusion Detection System based on temporal behavior of frequent patterns. MELECON 2014 - 2014 17th IEEE Mediterranean Electrotechnical Conference, (April), 432–438. doi:10.1109/MELCON.2014.6820573

Schuster, F., & Paul, A. (2012). A distributed intrusion detection system for industrial automation networks. In Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012) (pp. 1–4). doi:10.1109/ETFA.2012.6489703

Shosha, A. F., Gladyshev, P., Wu, S. S., & Liu, C. C. (2011). Detecting cyber intrusions in SCADA networks using multi-agent collaboration. In 2011 16th International Conference on Intelligent System Applications to Power Systems, ISAP 2011 (pp. 1–7). doi:10.1109/ISAP.2011.6082170

Tiago Cruz, Jorge Proença, Paulo Simões, Matthieu Aubigny, Moussa Ouedraogo, Antonio Graziano, L. Y. (2014). Improving Cyber Security Awareness on Industrial Control Systems: The CockpitCI Approach. 13th European Conference on Cyber Warfare and Security The University of Piraeus Greece 3-4 July 2014, (July), 326.

Snort.org. (2016). Snort. Retrieved June 24, 2016, from https://www.snort.org/

Elasticsearch. (2016). Retrieved June 24, 2016, from https://www.elastic.co/

Quickdraw SCADA IDS (2016). Retrieved September 12, 2016, http://www.digitalbond.com/tools/quickdraw/

Aramaki, T. L., Vellasco, M. M. B. R., & Barbosa, C. R. H. (2015). A neural network approach for leak detection and localization in liquid pipelines. In *8th Brazilian Congress on Metrology* (pp. 2–5). Bento Gonçalves.

NVD (2016). Retrieved September 12, 2016, https://nvd.nist.gov/

CVE (2016). Retrieved September 12, 2016, https://cve.mitre.org/

RISI (2016). Retrieved September 12, 2016, http://www.risidata.com/

Falliere, N., Murchu, L. O., & Chien, E. (2011). W32.Stuxnet Dossier. Symantec-Security Response, Version 1.(February 2011), 1–69. http://doi.org/20 September 2015

Ataques Furtivos em Sistemas de Controle Físicos Cibernéticos

Alan Oliveira de Sá[1,2], **Luiz F. Rust da Costa Carmo**[1,3], **Raphael C. S. Machado**[3]

[1]Programa de Pós-Graduação em Informática - Instituto Tércio Pacitti / IM,
Universidade Federal do Rio de Janeiro, 21.941-901, RJ – Brasil

[2]Centro de Instrução Almirante Wandenkolk – Marinha do Brasil,
Ilha das Enxadas, Baía de Guanabara – Rio de Janeiro – RJ – Brasil

[3]Instituto Nacional de Metrologia, Qualidade e Tecnologia (Inmetro)
Av. Nossa Senhora das Graças, 50, Xerém, Duque de Caxias, 25.250-020, RJ – Brasil

alan.oliveira.sa@gmail.com, {lfrust,rcmachado}@inmetro.gov.br

Abstract. *The advantages of using communication networks to interconnect controllers and physical plants motivate the increasing number of Networked Control Systems, in industrial and critical infrastructure facilities. However, this integration also exposes such control systems to new threats, typical of the cyber domain. In this context, studies have been conduced, aiming to explore vulnerabilities and propose security solutions for cyber-physical systems. In this paper, it is proposed a covert attack for system degradation, which is planned based on the intelligence gathered by another attack, herein proposed, referred as System Identification attack. The simulation results demonstrate that the joint operation of the two attacks is capable to affect, in a covert and accurate way, the physical behavior of a system.*

Resumo. *As vantagens do uso de redes de comunicação para interconectar controladores e plantas físicas tem motivado o crescente número de Sistemas de Controle em Rede, na indústria e em infraestruturas críticas. Entretanto, esta integração expõe tais sistemas a novas ameaças, típicas do domínio cibernético. Neste contexto, estudos têm sido realizados com o objetivo de explorar as vulnerabilidades e propor soluções de segurança para sistemas físico-cibernéticos. Neste artigo é proposto um ataque furtivo de degradação de serviço o qual é planejado com base nos dados colhidos por um outro ataque, ora proposto, denominado de System Identification. Os resultados de simulação demonstram que a operação conjunta dos dois ataques é capaz de afetar de forma furtiva e acurada o comportamento físico de um sistema.*

1. Introdução

A integração de sistemas usados para controlar processos físicos por meio de redes de comunicação visa atribuir a tais sistemas melhores capacidades operacionais e gerenciais, bem como reduzir custos. Em face destas vantagens, existe a tendência de um crescente número de processos industriais e sistemas de infraestruturas críticas controlados por Sistemas de Controle em Rede, ou *Networked Control Systems* (NCS) [Farooqui et al. 2014], também referidos como *Network-Based Control Systems* (NBCS) [Long et al. 2005]. Um NCS, conforme apresentado na Figura 1, consiste de uma planta física, descrita por uma função de transferência $G(z)$, um controlador, o qual executa uma função de controle

$C(z)$, e uma rede de comunicação que interconecta ambos os dispositivos para a transmissão de sinais de controle e de realimentação. Os sinais de controle são transmitidos do controlador para os atuadores da planta. Os sinais de realimentação são transmitidos dos sensores da planta para o controlador.

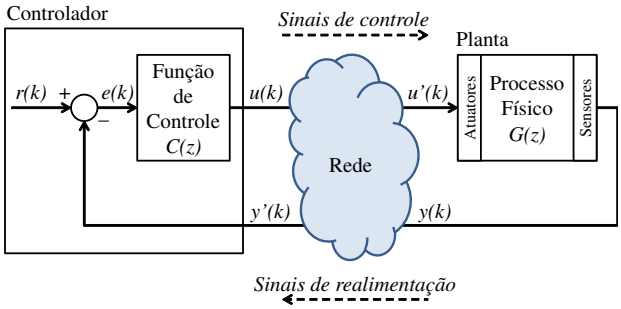


Figura 1. Sistema de Controle em Rede (ou NCS)

Ao mesmo tempo em que traz uma série de vantagens, a integração de controladores e plantas físicas em malha fechada por meio de redes de comunicação também expõe tais sistemas a novas ameaças, típicas do domínio cibernético. Neste contexto, estudos vêm sendo realizados, com o objetivo de caracterizar vulnerabilidades e propor soluções de segurança em NCSs.

Uma possível forma de atacar um NCS se dá pela intervenção em seu *software*, *i.e.* por meio de alterações na configuração ou mesmo no código executado pelo controlador, seguindo estratégia similar àquela utilizada pelo *worm* Stuxnet [Langner 2011]. Outra maneira possível para um atacante afetar um NCS é por meio de interferências no seu processo de comunicação. Basicamente, um atacante pode interferir nos sinais de controle e/ou de realimentação de três diferentes modos: induzindo *jitter* (atrasos variáveis), causando a perda de pacotes de dados, ou mesmo injetando dados falsos na comunicação.

No presente trabalho, desenvolvemos um ataque onde são injetados dados falsos no processo de comunicação de um NCS, demonstrando a possibilidade de degradação do serviço realizado por uma planta por meio de alterações sutis em seu comportamento físico. Esta intervenção tem por objetivo reduzir a eficiência da planta ou mesmo lhe causar danos em médio/longo prazo. Cabe ressaltar que uma intervenção descontrolada no NCS pode levar a uma avaria imediata da planta, ou mesmo causar alterações de grande proporção em seu funcionamento, o que pode resultar na descoberta do ataque e no eventual insucesso da operação. Sendo assim, as alterações impelidas pelo ataque ora proposto são dimensionadas para que a mudança de comportamento da planta seja fisicamente de difícil percepção, motivo pelo qual classificamos o ataque como fisicamente furtivo.

Para garantir que o ataque a um NCS seja fisicamente furtivo, o atacante deve planejar sua ofensiva com base em um conhecimento acurado sobre a dinâmica do sistema, caso contrário, as consequências do ataque podem ser imprevisíveis. Uma forma de adquirir tal conhecimento é por meio de operações de inteligência convencionais, desempenhadas para colher informações sobre o projeto e a dinâmica do NCS. Outra forma de obter informações sobre o sistema a ser atacado é por meio de o que classificamos neste trabalho como ataques de *Cyber-Physical Intelligence*. Neste sentido, também propomos no presente artigo um ataque de identificação de sistemas, ou *System Identification*, que visa obter informações sobre a função de transferência $G(z)$ da planta e da função de controle $C(z)$ do controlador. Este ataque é baseado no Algoritmo de Busca por Retro-

cesso, ou *Backtracking Search Optimization algorithm* (BSA) [Civicioglu 2013]. Note que, tanto o ataque de degradação de serviço por meio da injeção de dados, quanto o ataque *System Identification* requerem acesso aos sinais transmitidos no NCS, o que pode ser dificultado por técnicas de criptografia. Entretanto, não se pode negligenciar a possibilidade de acesso a tais dados por meio de ataques de criptoanálise ou mesmo de força bruta.

O presente trabalho motivou a formalização de uma série de conceitos relacionados a furtividade e inteligência no contexto da segurança físico-cibernética. Sendo assim, uma contribuição complementar do artigo é a proposição de uma nomenclatura que abarque toda uma nova classe de ataques aos sistemas físico-cibernéticos. A taxonomia proposta estabelece uma nova abordagem quanto à furtividade de ataques a sistemas físico-cibernéticos, os quais devem ser analisados sob dois aspectos simultaneamente: o aspecto físico e o aspecto cibernético.

É digno de nota que o objetivo deste trabalho não é facilitar ataques furtivos de degradação de serviço em sistemas de controle físico-cibernéticos. O objetivo deste trabalho é demonstrar o grau de acurácia que pode ser obtido neste tipo de ataque, sobretudo quando apoiado por ataques de *System Identification*, e, portanto, encorajar a pesquisa de contramedidas para tais ataques. O restante do artigo é organizado da seguinte forma: Primeiramente, na Seção 2, são apresentados alguns trabalhos relacionados. Em seguida, na Seção 3, é proposta uma taxonomia referente aos ataques físico-cibernéticos em malhas de controle de NCSs. Na Seção 4, é feita a descrição de um ataque do tipo *System Identification*. Na Seção 5, é definido um ataque furtivo de degradação de serviço. Na Seção 6, são apresentados os resultados obtidos em simulações de ataques furtivos de degradação de serviço, apoiados por ataques *System Identification*. Finalmente, na Seção 7, são apresentadas algumas conclusões e possibilidades de trabalhos futuros.

2. Trabalhos Relacionados

A possibilidade de ataques físico-cibernéticos se tornou uma realidade após o lançamento do *worm* Stuxnet [Langner 2011] e tem motivado pesquisas concernentes à segurança de NCSs. Nesta seção são apresentados alguns trabalhos relacionados ao assunto.

Em [Long et al. 2005] os autores propõem dois modelos de fila para avaliar o impacto do *jitter* e da perda de pacotes em um NCS sob ataque. O ataque não é planejado com base em um conhecimento prévio sobre os modelos do controlador e da planta. Sendo assim, para afetar o comportamento físico dos sistema, o atacante inunda a rede com um tráfego adicional, causando *jitter* e perda de pacotes de forma arbitrária. Nesta tática, o excesso de pacotes na rede pode reduzir a furtividade do ataque, permitindo a adoção de contramedidas tais como a filtragem de pacotes ou o bloqueio do tráfego malicioso na sua origem [Long et al. 2005]. Adicionalmente, a ação arbitrária sobre um modelo desconhecido pode levar o sistema a comportamentos físicos extremos, o que não é desejável se for almejado um ataque furtivo.

Em [Farooqui et al. 2014], os autores apresentam uma plataforma de testes para sistemas SCADA (*Supervisory Control and Data Acquisition*). Os mesmos demonstram um ataque onde são enviados dados falsos para o controlador e para o atuador do NCS. No artigo, os dados falsos injetados durante a comunicação têm valores randômicos e visam fazer com que um motor DC perca a sua estabilidade. Este tipo de ataque não demanda

um conhecimento prévio sobre o NCS. Em contrapartida, o efeito físico desejado e a furtividade não podem ser garantidos em virtude das consequências imprevisíveis que podem surgir da aplicação de sinais aleatórios em um sistema cujo modelo é desconhecido.

Mais recentemente, em [Teixeira et al. 2015], os autores fornecem um quadro geral contendo a análise de uma grande variedade de métodos de ataque em NCSs. Em sua classificação, os mesmos estabelecem que ataques furtivos em NCSs requerem um alto nível de conhecimento sobre o sistema atacado. Exemplos de ataques furtivos são apresentados em [Smith 2011, Smith 2015]. Nestes trabalhos os ataques são desempenhados por um *man-in-the-middle* (MitM), onde o atacante necessita injetar dados tanto no enlace de controle quanto no de realimentação, bem como conhecer o modelo da planta que está sendo controlada. A furtividade destes ataques, que depende da diferença entre o modelo real da planta e o modelo utilizado pelo atacante, é analisada do ponto de vista dos sinais que chegam para o controlador, sem abordar se os efeitos físicos causados na planta são perceptíveis, ou se são furtivos perante um observador humano.

Nos trabalhos [Teixeira et al. 2015, Smith 2011, Smith 2015], onde é requerido um conhecimento sobre o modelo do NCS atacado, não é descrito como este este conhecimento é obtido pelo atacante. Considera-se apenas que o modelo é previamente conhecido para subsidiar o planejamento do ataque. A ação conjunta, ora proposta, de um ataque furtivo de degradação de serviço, apoiado por um ataque *System Identification*, visa preencher este hiato, demonstrando como os dados do NCS podem ser obtidos e como um ataque furtivo pode se beneficiar disto. A Tabela 1 apresenta uma síntese das características dos ataques apresentados nesta seção.

Tabela 1. Síntese dos ataques mencionados

Ataque	Método de ataque	Conhecimento sobre o modelo	Como o modelo é obtido
Long, *et al.* [Long et al. 2005]	Indução de *jitter* e perda de pacotes	Nenhum	N/A
Stuxnet *worm* [Langner 2011]	Modificações no código do PLC	Sim	Experimentos em um sistema real
Farooqui, *et al.* [Farooqui et al. 2014]	Injeção de dados	Nenhum	N/A
Smith [Smith 2011, Smith 2015]	Injeção de dados	Sim	Não descrito
Teixeira [Teixeira et al. 2015]	Perda de pacotes	Nenhum	N/A
	Injeção de dados	Sim	Não descrito

3. Taxonomia

Nesta Seção é apresentada uma taxonomia relativa aos possíveis ataques a sistemas de controle físico-cibernéticos. Na Seção 3.1, os ataques são brevemente descritos e classificados de acordo com a forma como agem no NCS. Na Seção 3.2, é proposta uma nova abordagem para a análise da furtividade de ataques à sistemas físico-cibernéticos.

3.1. Classificação dos ataques

Ataques a sistemas físico-cibernéticos podem atuar tanto nos seus dispositivos – *i.e.* no controlador, atuadores e sensores da planta – quanto em seus sistemas de comunicação, afetando os sinais de controle e de realimentação. Como premissa, devemos considerar que o *serviço* que se pretende atacar/proteger em tal sistema é o trabalho executado pelo processo físico, controlado por um NCS.

Considerando a definição supracitada de serviço em NCSs, os ataques podem ser classificados em três categorias distintas, como apresentado na Figura 2:

- *Denial-of-Service* (DoS) [Hussain et al. 2003]: em um NCS, os ataques DoS compreendem todos os tipos de ataques físico-cibernéticos que neguem a operação do processo físico, interrompendo a execução do serviço que a planta controlada se propõe a fazer. O ataque resulta, por exemplo, em comportamentos que podem desligar a planta ou mesmo destruí-la em um curto prazo.
- *Service Degradation* (SD): os ataques do tipo SD consistem em intervenções maliciosas que são executadas na malha de controle visando reduzir a eficiência do serviço, *i.e.* a eficiência do processo físico, ou mesmo reduzir o tempo médio entre falhas, ou *mean time between failure* (MTBF), da planta em médio/longo prazo.
- *Cyber-physical Intelligence* (CPI): o conceito de *Cyber-physical Intelligence*, aqui proposto, é diferente do conceito onde sistemas físico-cibernéticos são integrados com sistemas inteligentes [Ramos et al. 2011]. Na presente taxonomia, os ataques do tipo CPI compreendem as ações que são desempenhadas na malha de controle do NCS com o objetivo de colher informações sobre a operação do sistema e/ou sobre o seu projeto. Estes ataques têm por objetivo adquirir as informações necessárias para o planejamento de ataques furtivos e controlados, ou mesmo para subsidiar ações de *replay* [Langner 2011].

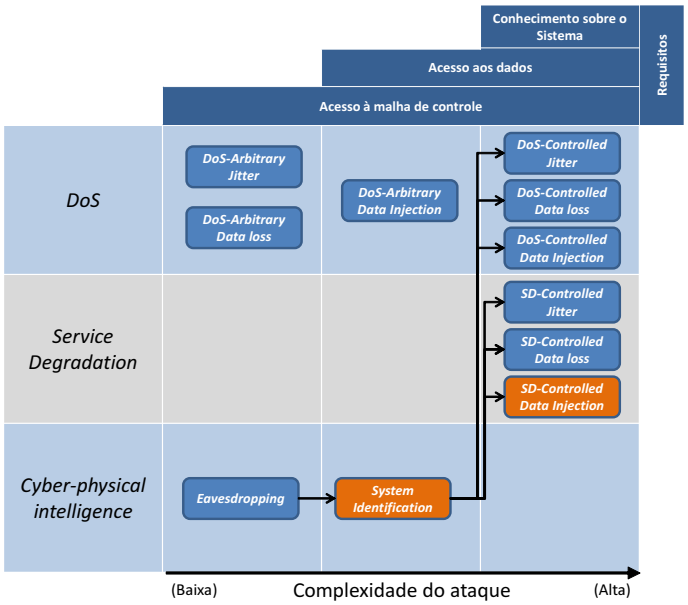


Figura 2. Classificação e requisitos dos ataques físico-cibernéticos atuantes na malha de controle de um NCS.

Na Figura 2, são apresentados seis tipos de ataques DoS, bem como os seus respectivos requisitos. Destes seis tipos de ataque, os menos complexos são os arbitrários:

- *DoS-Arbitrary Jitter*: neste tipo de ataque, o atraso dos sinais de controle e /ou realimentação é alterado arbitrariamente, sem um conhecimento prévio do modelo do NCS, com o objetivo de levar o sistema a uma instabilidade ou a uma condição que cause a interrupção do processo físico. Este ataque requer somente o acesso à malha de controle, uma vez que o mesmo pode se dar pelo simples consumo de recursos do sistema, tal como a banda dos enlaces de comunicação, ou mesmo recursos computacionais dos equipamentos que fazem parte da malha de controle.

- *DoS-Arbitrary Data Loss*: neste tipo de ataque, o atacante impede que os dados cheguem aos atuadores e/ou controladores. O atacante efetua um *jamming* arbitrário nos sinais de comunicação, sem um conhecimento prévio do modelo do NCS, levando o sistema à instabilidade ou a uma condição que cause a interrupção do processo físico. Cabe ressaltar que alguns ataques do tipo *DoS-Arbitrary Jitter* podem evoluir para um ataque *DoS-Arbitrary Data Loss*, caso atrasos de maior proporção venham a causar a perda de pacotes. Assim como no ataque *DoS-Arbitrary Jitter*, este ataque só requer o acesso à malha de controle do NCS.
- *DoS-Arbitrary Data Injection*: nestes ataques, o atacante envia dados falsos e arbitrários ao controlador, como se estes tivessem sido enviados pelos sensores, e/ou para os atuadores, como se tivessem sido enviados pelo controlador. Os dados são injetados na malha de controle do NCS sem o conhecimento prévio de seu modelo. Este ataque é mais complexo que os ataques *DoS-Arbitrary Jitter* e *DoS-Arbitrary Data Loss*, uma vez que requer o acesso aos dados que fluem na malha de controle do NCS.

Os ataques do tipo *DoS-Controlled – DoS-Controlled Jitter*, *DoS-Controlled Data Loss* e *DoS-Controlled Data Injection* – apresentados na Figura 2, interferem na malha de controle do NCS da mesma forma que seus respectivos ataques *DoS-Arbitrary*. A diferença entre um ataque *DoS-Controlled* e um ataque *DoS-Arbitrary* é que, no primeiro, a interferência causada pelo atacante é precisamente planejada e executada, visando alcançar com exatidão o comportamento desejado que leva o sistema à interrupção do serviço físico, de uma forma mais eficiente. Assim, para alcançar tal eficiência, um ataque *DoS-Controlled* requer um conhecimento acurado do modelo do NCS, *i.e.* das funções de transferência da planta e do controlador, as quais devem ser analisadas para o planejamento do ataque.

Referente aos ataques SD, devemos considerar três diferentes tipos de ataque – *SD-Controlled Jitter*, *SD-Controlled Data Loss* e *SD-Controlled Data Injection* – conforme apresentado na Figura 2. A diferença entre um ataque *SD-Controlled* e um ataque *DoS-Controlled* é que o primeiro não tem a intenção de interromper o processo físico em um curto prazo. O ataque visa manter o processo funcionando com a eficiência reduzida ou, por vezes, causar a deterioração física e gradual dos dispositivos controlados. Para que isto ocorra, os ataques *SD-Controlled* requerem um conhecimento prévio e acurado sobre o NCS. Caso contrário o ataque pode, por razões não previstas, evoluir para um ataque DoS, causando a interrupção do processo físico.

O conhecimento sobre o sistema, requerido tanto nos ataques *DoS-Controlled* e *SD-Controlled*, pode ser obtido por meio de ataques CPI, conforme apresentado na Figura 2. O primeiro, e mais simples, ataque CPI é o *eavesdropping* [Khatri et al. 2015], que consiste em simplesmente capturar os sinais de controle e de realimentação transmitidos. O segundo ataque CPI, proposto neste artigo, é o *System Identification*, o qual visa obter informações sobre a função de transferência da planta e a função de controle do controlador por meio da análise dos sinais que trafegam na rede. Os ataques CPI por si só não impactam no funcionamento do NCS, mas são uma poderosa ferramenta para planejar ataques *DoS-Controlled* e *SD-Controlled* eficientes.

3.2. Furtividade Cibernética vs. Física

A furtividade de um ataque corresponde à sua capacidade de não ser percebido ou detectado. No caso de ataques físico-cibernéticos em NCSs, a furtividade deve ser analisada

simultaneamente em dois domínios diferentes: o domínio cibernético; e o domínio físico. Neste sentido, é apresentada nesta seção a definição de o que é um ataque *cibernetica-mente furtivo* e o que é um ataque *fisicamente furtivo*:

- Ataques ciberneticamente furtivos: são ataques que têm baixa probabilidade de serem detectados por algoritmos que monitoram os softwares, a comunicação e os dados do sistema, ou por sistemas que monitoram a dinâmica da planta.
- Ataques fisicamente furtivos: são ataques que causam efeitos físicos que não são facilmente percebidos ou identificados por um observador humano. O ataque modifica sutilmente alguns comportamentos do sistema de forma a afetar fisicamente a planta, mas o efeito não é facilmente percebido ou, eventualmente, pode ser entendido como uma consequência cuja causa seja outra, diferente de um ataque.

4. Ataque de Identificação de Sistema

O ataque de Identificação de Sistemas, ou *System identification*, aqui apresentado visa estimar os coeficientes da função de transferência da planta $G(z)$ e da função de controle $C(z)$ do controlador. Ambas as funções são de sistemas Lineares e Invariantes no Tempo (LIT). O ataque usa o Algoritmo de Busca por Retrocesso, ou *Backtracking Search Algorithm* (BSA), proposto em [Civicioglu 2013] e resumidamente descrito em [de Sá et al. 2016], como metaheurística para minimizar a função de aptidão apresentada nesta Seção.

O BSA é um algoritmo evolucionário que utiliza informações obtidas por gerações – ou iterações – passadas para buscar soluções em problemas de otimização. O algoritmo possui dois parâmetros que são empiricamente ajustados: o tamanho da sua população P; e η, descrito em [de Sá et al. 2016], que estabelece a amplitude do deslocamento dos indivíduos de P. O parâmetro η deve ser ajustado visando atribuir ao algoritmo tanto uma boa capacidade exploração, quanto de refinamento da busca.

Se a entrada $i(k)$ e a saída $o(k)$ de um dispositivo real do NCS são conhecidas, seu modelo interno pode ser inferido aplicando a entrada conhecida $i(k)$ em um modelo estimado, que deve ser ajustado até que a sua saída estimada $\hat{o}(k)$ convirja para $o(k)$. Neste sentido, o BSA é usado para ajustar iterativamente o modelo estimado, minimizando uma função de aptidão específica, até que o modelo estimado convirja para o modelo real do dispositivo do NCS, o qual pode ser um controlador ou uma planta.

Para estabelecer a função de aptidão, devemos primeiramente considerar o sistema LIT genérico, cuja função de transferência $Q(z)$ pode ser representada por (1):

$$Q(z) = \frac{O(z)}{I(z)} = \frac{a_n z^n + a_{n-1} z^{n-1} + ... + a_1 z^1 + a_0}{z^m + b_{m-1} z^{m-1} + ... + b_1 z^1 + b_0}, \tag{1}$$

onde $I(z)$ é a entrada do sistema, $O(z)$ é a sua saída, n e m correspondem a ordem do numerador e do denominador, respectivamente, e $[a_n, a_{n-1}, ...a_1, a_0]$ e $[b_{m-1}, b_{m-2}, ...b_1, b_0]$ são os coeficientes do numerador e do denominador, respectivamente, os quais pretende-se estimar com o presente algoritmo de Identificação de Sistemas. Consideremos ainda que $i(k)$ e $o(k)$ representam as amostras da entrada e da saída do sistema, respectivamente, onde $I(z) = \mathcal{Z}[i(k)]$, $O(z) = \mathcal{Z}[o(k)]$, k é o número da amostra e $\mathcal{Z}$ representa a operação da transformada Z.

Neste ataque de identificação de sistemas, $i(k)$ e $o(k)$ são primeiramente capturados por um ataque do tipo *eavesdropping* [Khatri et al. 2015], por exemplo, durante um

período T. Para lidar com eventuais perdas de amostras, que podem não ser recebidas pelo atacante durante T, o algoritmo retém o valor da última amostra recebida, conforme (2), onde $x(k)$ pode ser tanto $i(k)$ quanto $o(k)$.

$$x(k) = \begin{cases} x(k-1) & \text{se a amostra } k \text{ é perdida;} \\ \\ x(k) & \text{senão.} \end{cases} \tag{2}$$

Em seguida, após capturar $i(k)$ e $o(k)$, o sinal $i(k)$ é aplicado à entrada de um modelo estimado, descrito por função de transferência cujos coeficientes $[a_{n,j}, a_{n-1,j}, ...a_{1,j}, a_{0,j}, b_{m-1,j}, b_{m-2,j}, ...b_{1,j}, b_{0,j}]$ são as coordenadas de um indivíduo j do BSA. A aplicação de $i(k)$ ao modelo estimado resulta em um sinal de saída $\hat{o}_j(k)$. Após obter $\hat{o}_j(k)$, a função de aptidão f_j do indivíduo j é calculada comparando a saída $o(k)$, capturada no dispositivo atacado, com a saída do modelo estimado $\hat{o}_j(k)$, de acordo com (3):

$$f_j = \frac{\sum_{k=0}^{N} (o(k) - \hat{o}_j(k))^2}{N}, \tag{3}$$

onde N é o número de amostras que existem durante o período de monitoração T. Note que, se o atacante não perder nenhuma amostra de $i(k)$ e $o(k)$ durante T, então $\min f_j = 0$ quando $[a_{n,j}, a_{n-1,j}, ...a_{1,j}, a_{0,j}, b_{m-1,j}, b_{m-2,j}, ...b_{1,j}, b_{0,j}] = [a_n, a_{n-1}, ...a_1, a_0, b_{m-1}, b_{m-2}, ...b_1, b_0]$, *i.e.* quando o modelo estimado converge para o modelo real.

É possível estabelecer uma analogia entre este ataque de identificação de sistemas e o ataque de criptoanálise do tipo *known plaintext*, onde $i(k)$ e $o(k)$ correspondem aos textos simples e cifrado, respectivamente, o formato da função de transferência genérica $Q(z)$ corresponde ao algoritmo de criptografia e os coeficientes reais de $Q(z)$ correspondem à chave criptográfica.

5. Ataque Furtivo para Degradação do Serviço

Com base na taxonomia apresentada na Seção 3.1, o ataque descrito nesta Seção é classificado como do tipo *SD-Controlled Data Injection*. Seu propósito é reduzir o MTBF da planta e/ou reduzir a eficiência do processo físico que a mesma executa, através da inserção de dados falsos na malha de controle. Ao mesmo tempo, o atacante deseja que o ataque atenda ao requisito de ser fisicamente furtivo, *i.e.* com um efeito físico de difícil percepção por um observador humano, ou entendido como uma consequência cuja causa não seja um ataque – conforme definido na seção Seção 3.2.

Uma das maneiras de degradar um serviço físico é por meio da indução de um *overshoot* durante o regime transitório da planta. *Overshoots*, ou picos no regime transitório, podem causar estresse e, eventualmente, danos à sistemas físicos como por exemplo sistemas mecânicos, químicos e eletromecânicos [El-Sharkawi and Huang 1989, Tran et al. 2007]. Adicionalmente, por ocorrerem em curto espaço de tempo, os *overshoots* são de difícil percepção pelo observador humano. Outra forma de degradar o serviço é causar um erro estacionário constante na planta, ou seja, fazer com que a saída da mesma tenha um erro constante quando $t \to \infty$. Erros estacionários de pequena proporção, além de serem de difícil percepção pelo observador humano, podem reduzir a eficiência do processo físico e, eventualmente, estressar e danificar o sistema em médio/longo prazo.

Neste ataque, para alcançar qualquer um dos dois efeitos citados, *i.e.* um *overshoot* ou um erro estacionário constante, o atacante intervém no processo de comunicação do NCS a fim de injetar, de forma controlada, dados falsos no sistema. Para tal, o atacante atua como um MitM que executa uma função de ataque $M(z)$, conforme apresentado na Figura 3, onde $U'(z) = M(z)U(z)$, $U(z) = \mathcal{Z}[u(k)]$ e $U'(z) = \mathcal{Z}[u'(k)]$. A função $M(z)$ é projetada com base nos dados da planta e do controlador, obtidos no ataque do tipo *System Identification* descrito na Seção 4. A eficácia do ataque, portanto, depende do projeto de $M(z)$, que por sua vez depende da acurácia do ataque de *System Identification*. Cabe ressaltar que, apesar de na Figura 3 o MitM atuar nos sinais de controle, é possível, também, que o mesmo atue nos sinais de realimentação do NCS. O MitM pode ser estabelecido tanto em redes cabeadas quanto, eventualmente, em redes sem fio conforme em [Hwang et al. 2008].

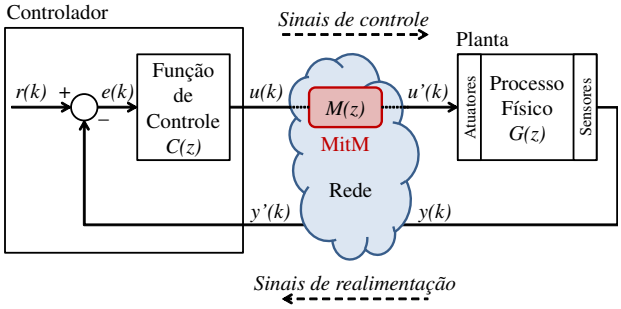


Figura 3. Ataque MitM

6. Resultados

Nesta seção são apresentados os resultados obtidos em simulações que combinam ataques do tipo *System Identification* com ataques *SD-Controlled* fisicamente furtivos. Na Seção 6.1, é apresentado o modelo do sistema atacado. Na Seção 6.2 são apresentados os resultados obtidos pelo ataque do tipo *System Identification*. Na Seção 6.3 são apresentados os resultados obtidos com simulações de ataques do tipo *SD-Controlled Data Injection*, fisicamente furtivos, planejados com base nos dados do ataque de *System Identification*.

6.1. Modelo do Sistema

O NCS atacado tem a mesma arquitetura do NCS apresentado na Figura 1, e consiste em um controlador Proporcional-Integral (PI) que controla a velocidade de rotação de um motor DC. A função de controle $C(z)$ e a função de transferência $G(z)$ do motor DC foram extraídas de [Long et al. 2005]. Tais equações são representadas por (4):

$$C(z) = \frac{c_1 z - c_2}{z - 1} \qquad G(z) = \frac{g_1 z + g_2}{z^2 - g_3 z + g_4} \tag{4}$$

onde $c_1 = 0,1701$, $c_2 = -0,1673$, $g_1 = 0,3379$, $g_2 = 0,2793$, $g_3 = -1,5462$ e $g_4 = 0,5646$. A taxa de amostragem do sistema é 50 amostras/s e o *set point* $r(k)$ é uma função degrau unitário. O atraso na rede não é considerado nestas simulações.

6.2. Resultados da Identificação do Sistema

Nesta Seção, o desempenho do algoritmo de Identificação de Sistemas é avaliado por meio um conjunto de simulações realizadas no MATLAB. A ferramenta SIMULINK foi

utilizada para calcular a saída $\hat{o}_j$ dos modelos estimados, cujos coeficientes são as coordenadas de um indivíduo j do BSA.

A estrutura das equações representadas por (4) são previamente conhecidas pelo atacante, uma vez que, como premissa, este sabe que o alvo é um NCS que controla um motor DC por meio de um controlador PI. Nestas simulações, o objetivo do ataque de *System Identification* é descobrir g_1, g_2, g_3, g_4, c_1 e c_2, levando em consideração cenários em que o atacante eventualmente perde amostras durante a coleta dos sinais de controle e de realimentação.

Toda vez que o motor DC é ligado, os sinais de controle e de realimentação são capturados pelo atacante durante um período $T = 2s$. no momento em que o motor é ligado, todas as condições iniciais são consideradas 0. Os coeficientes de $G(z)$, $[g_1, g_2, g_3, g_4]$, e os coeficientes de $C(z)$, $[c_1, c_2]$, são calculados separadamente considerando que, apesar da malha fechada, $G(z)$ e $C(z)$ são funções independentes. Para estimar $[g_1, g_2, g_3, g_4]$, o atacante considera que o sinal de controle é a entrada e que o sinal de realimentação é a saída da planta. Já para estimar $[c_1, c_2]$, o atacante considera que o sinal e realimentação é a entrada e que o sinal de controle é a saída do controlador.

Para simular a perda de amostras, são consideradas quatro taxas de perda l diferentes: 0, 0,05, 0,1 e 0,2. Assim, uma amostra é perdida pelo atacante toda vez que $l < \mathcal{P}$, onde $\mathcal{P} \sim U(0, 1)$ e U é uma distribuição uniforme. Para cada taxa de perda são executadas 100 simulações diferentes.

No BSA, a população utilizada contém 100 indivíduos e η, empiricamente ajustado, é 1. Para estimar os coeficientes do controlador $[c_1, c_2]$, são executadas 600 iterações do algoritmo. Já para estimar os coeficientes da planta $[g_1, g_2, g_3, g_4]$, o número de iterações é aumentado para 800, devido ao maior número de dimensões do espaço de busca neste caso. Os limites de cada dimensão do espaço de busca são $[-10, 10]$.

A Figura 4 apresenta a média de 100 valores estimados para g_1, g_2, g_3, g_4, c_1 e c_2, com um Intervalo de Confiança (IC) de 95%, considerando diferentes taxas de perda de amostras. Os valores reais dos coeficientes de $C(z)$ e $G(z)$ também são representados na Figura 4. Note que a amplitude das escalas das Figuras 4(a), 4(b), 4(c) e 4(d) é diferente da amplitude das escalas das Figuras 4(e) e 4(f), em virtude dos menores IC de c_1 e c_2. Adicionalmente, algumas estatísticas referentes aos resultados obtidos são apresentadas na Tabela 2.

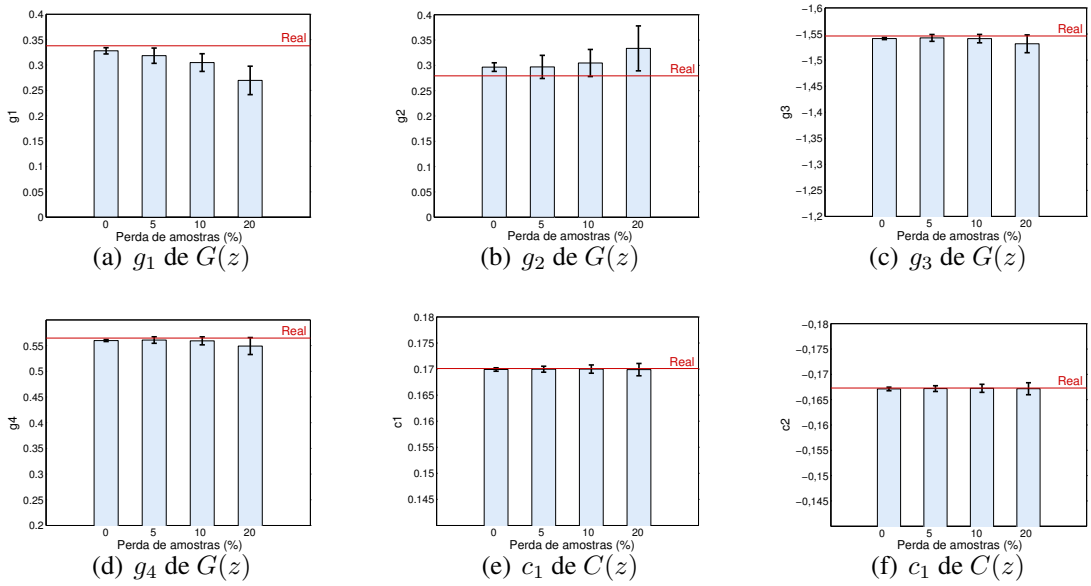
Tabela 2. Estatísticas dos resultados com diferentes perdas de amostras

Perda de amostras	Média						Desvio Padrão					
	g_1	g_2	g_3	g_4	c_1	c_2	g_1	g_2	g_3	g_4	c_1	c_2
0%	0.32793	0.29652	-1.54121	0.55983	0.16991	-0.16712	0.03097	0.04288	0.00986	0.00944	0.00167	0.00178
5%	0.31835	0.29689	-1.54251	0.56085	0.16997	-0.16719	0.07572	0.11523	0.03322	0.03194	0.00287	0.00287
10%	0.30473	0.30461	-1.54110	0.55925	0.16999	-0.16724	0.08781	0.13483	0.04076	0.03922	0.00397	0.00399
20%	0.26963	0.33352	-1.53119	0.54916	0.16989	-0.16716	0.14120	0.22378	0.08596	0.08313	0.00596	0.00598
Perda de amostras	Assimetria(*)						Curtose					
	g_1	g_2	g_3	g_4	c_1	c_2	g_1	g_2	g_3	g_4	c_1	c_2
0%	-1.21214	1.23278	1.75298	-1.73202	-0.64331	0.79458	0.18846	0.19433	0.21259	0.21218	0.15119	0.16472
5%	-2.34607	1.64875	1.35284	-1.41346	-0.42288	0.36037	0.08094	0.10527	0.09412	0.09802	0.02540	0.03118
10%	-2.52938	1.97711	1.18018	-1.26045	-0.23379	0.13377	0.16833	0.17123	0.25041	0.24811	0.24361	0.23429
20%	-3.24122	1.75186	1.68335	-1.71055	-0.40055	0.37927	0.21292	0.21127	0.25054	0.24932	0.23883	0.24441

(*) Calculado de acordo com o 2^{o} coeficiente de assimetria de Pearson.

De acordo com a Tabela 2 as distribuições de g_1, g_2, g_3 e g_4 possuem forte assimetria, enquanto as distribuições de c_1 e c_2 têm assimetria moderada. Em relação a curtose, as distribuições de todos os coeficientes de $G(z)$ e $C(z)$ são leptocúrticas. Entretanto, analisando a Tabela 2, não é possível identificar uma clara tendência de aumento ou diminuição de assimetria e curtose em face do aumento da perda de amostras.

Na Figura 4, é possível constatar que em todos os casos os ICs tendem a crescer com o aumento da perda de amostras. O mesmo ocorre com os desvios padrão apresentados na Tabela 2. Referente aos coeficientes de $G(z)$, a Figura 4 mostra que a diferença entre a média e o valor real de g_1, g_2, g_3 e g_4 também tende a crescer com o aumento da perda de amostras. Cabe ressaltar que o desempenho do algoritmo no cálculo de g_3 e g_4 é melhor do que no cálculo de g_1 e g_2, tanto no que diz respeito a média quanto a amplitude do IC. Atribuímos este comportamento à maior sensibilidade que a saída de $G(z)$ tem às variações de seus polos do que às variações de seus zeros. Isto significa que, neste problema, f_j cresce mais com os erros de g_3 e g_4 do que com os erros de g_1 e g_2, fazendo com que a população do BSA convirja de forma mais acurada em g_3 e g_4.



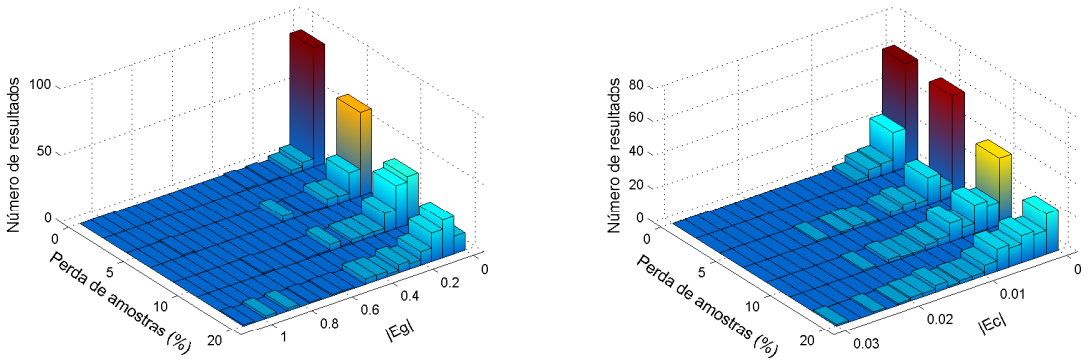
(a) g_1 de $G(z)$ (b) g_2 de $G(z)$ (c) g_3 de $G(z)$

(d) g_4 de $G(z)$ (e) c_1 de $C(z)$ (f) c_1 de $C(z)$

Figura 4. Média, com IC de 95%, dos coeficientes estimados de $G(z)$ e $C(z)$, em face de diferentes taxas de perda de amostras.

Na Figura 4 é possível também verificar que a acurácia obtida no cálculo dos coeficientes de $C(z)$ é melhor do que a acurácia dos coeficientes de $G(z)$, para todas as taxas de perda de amostras. As médias de c_1 e c_2 são mais próximas dos seus valores reais, com um menor IC. De fato, o processo de otimização é mais eficiente no cálculo dos coeficientes de $C(z)$ devido ao menor tamanho do espaço de busca, que possui apenas duas dimensões ao invés das quatro existentes no problema de $G(z)$.

Como uma forma adicional de avaliar o desempenho do algoritmo, foram calculados $|E_g| = |\mathcal{G}_r - \mathcal{G}_e|$ e $|E_c| = |\mathcal{C}_r - \mathcal{C}_e|$ que sintetizam o erro de estimativa dos coeficientes de $G(z)$ e $C(z)$, respectivamente, para cada solução encontrada. $\mathcal{G}_r$ e $\mathcal{G}_e$ são vetores contendo os coeficientes reais e estimados de $G(z)$, respectivamente. Já $\mathcal{C}_r$ e $\mathcal{C}_e$ são vetores contendo os coeficientes reais e estimados de $C(z)$, respectivamente. Os histogramas de $|E_g|$ e $|E_c|$ são apresentados na Figura 5, considerando as diferentes taxas de perda de amostras mencionadas. Os histogramas mostram graficamente que $|E_g|$ e $|E_c|$, que correspondem ao módulo do erro dos coeficientes estimados de $G(z)$ e $C(z)$, respectivamente, tendem a apresentar valores maiores à medida que a perda de amostras aumenta. Isto também pode ser confirmado pelo aumento do desvio padrão dos coeficientes de $G(z)$ e $C(z)$ apresentados na Tabela 2. Entretanto, de acordo com a Figura 5, a moda destes erros permanecem próximas de zero em todos os casos de perda avaliados.

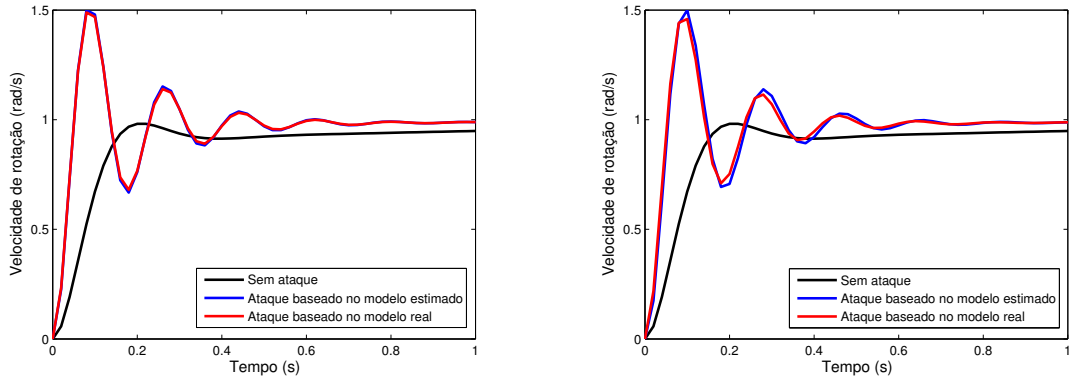


(a) Distribuição dos $|E_g|$ (b) Distribuição dos $|E_c|$

Figura 5. Histogramas de $|E_g|$ e $|E_c|$ em face das diferentes perdas de amostras

6.3. Resultados do Ataque de Degradação do Serviço

Nesta seção são apresentados os resultados obtidos em simulações de ataque do tipo *SD-Controlled Data Injection*, realizados por um MitM atuando no enlace de controle do NCS, conforme na Figura (3). Os ataques foram simulados no MATLAB, com o objetivo de avaliar a acurácia de ataques planejados com base nos resultados da Seção 6.2, obtidos pelo ataque de *System Identification*. Foram realizados dois conjuntos de ataques. O primeiro, visa causar um *overshoot* de 50% na velocidade de rotação do motor. O segundo, visa causar um erro estacionário de -10% na velocidade de rotação do motor em regime permanente.



(a) Ataque baseado nos dados obtidos sem perda de amostras (b) Ataque baseado nos dados obtidos com 20% de perda de amostras

Figura 6. Resposta do sistema a ataques planejados com o propósito de causar um *overshoot* de 50% da velocidade de rotação do motor.

No ataque visando o *overshoot*, a função executada pelo atacante é $M(z) = \mathcal{K}_o$. Por meio da análise do lugar das raízes, traçado com base nos modelos levantados, o atacante ajusta o valor de $\mathcal{K}_o$ para que o sistema se torne subamortecido com um pico de velocidade de rotação 50% maior do que a velocidade em regime permanente. Os valores de $\mathcal{K}_o$ foram ajustados com base nas médias dos coeficientes levantados na Seção 3.2. A Tabela 3 apresenta os valores de $\mathcal{K}_o$, estimados considerando as diferentes situações de perda de amostras no ataque de *System Identification*, bem como os *overshoots* obtidos com os respectivos $\mathcal{K}_o$ no modelo real. Na Figura 6 é possível comparar a resposta do sistema sem ataque com a resposta ao ataque visando o *overshoot* de 50%. É possível verificar, ainda, que o ataque ao modelo real apresenta, no domínio do tempo, uma resposta bem próxima ao ataque projetado com base no modelo obtido pelo ataque de *System*

Identification, tanto no caso em que o sistema foi identificado com 0% de perdas, quanto no pior caso considerado, com 20% de perdas. Cabe ressaltar que todas as respostas apresentadas na Figura 6 convergem para 1 rad/s.

No ataque cujo propósito é causar um erro estacionário de -10% na velocidade de rotação do motor, o atacante executa a função (5):

$$M(z) = \frac{\mathcal{K}_{Ess}(z-1)}{z-0,94}, \tag{5}$$

onde $\mathcal{K}_{Ess}$ é ajustado com base nos dados de identificação do sistema, considerando cada condição de perda de amostras. O pólo de $M(z)$ é adicionado com o objetivo de permitir que ocorra um erro estacionário no sistema. O zero de $M(z)$ visa formatar o lugar das raízes a fim de que haja um $\mathcal{K}_{Ess}$ estável que leve o sistema a um erro estacionário de -10%. A Tabela 3 apresenta os valores de $\mathcal{K}_{Ess}$ adotados considerando as diferentes situações de perda de amostras no ataque de *System Identification*, bem como os respectivos erros estacionários alcançados no modelo real.

Tabela 3. Valores de $\mathcal{K}_o$, $\mathcal{K}_{Ess}$ e resultados obtidos com os ataques

	Perda de amostras no ataque *System Identification*			
	0 %	5 %	10 %	20 %
$\mathcal{K}_o$	4,0451	4,0745	4,0828	3,796
Overshoot no modelo real	48,90 %	49,43 %	49,57 %	45,94 %
$\mathcal{K}_{Ess}$	5,7471	5,7803	5,8140	5,8823
Erro estacionário no modelo real	-10%	-10%	$-9,9\%$	$-9,8\%$

De acordo com os dados na Tabela 3, é possível afirmar que os ataques *SD-Controlled Data Injection*, projetados com base nos dados colhidos pelo ataque *System Identification*, foram capazes de modificar de forma acurada a resposta do sistema físico, considerando todas as condições de perda avaliadas. No pior caso, *i.e.* com 20% de perda de amostras, o *overshoot* foi de $45,94\%$ e o erro estacionário foi de $-9,8\%$, bem próximos dos valores desejados de 50% e -10%, respectivamente. Tal acurácia, permite que a resposta do sistema se mantenha controlada e próxima a um comportamento pré-definido como fisicamente furtivo para o sistema em questão.

7. Conclusões

Este trabalho propõe um ataque fisicamente furtivo de degradação de serviço, cujo desempenho depende do conhecimento sobre a planta atacada e seu controlador. Para adquirir tal conhecimento, é proposto um ataque de *System Identification*, baseado no algoritmo BSA. A eficácia do ataque de *System Identification* é demonstrada e o seu desempenho é estatisticamente analisado em face de diferentes taxas de perda de amostra. Os resultados alcançados nos ataques fisicamente furtivos de degradação de serviço, dimensionados com base nos dados levantados pelo *System Identification*, demonstram o elevado grau de acurácia que pode ser obtido com a combinação dos ataques. No pior caso, *i.e.* com 20% de perda de amostras durante a identificação do sistema, o atacante foi capaz de causar na planta um *overshoot* de $45,94\%$ e um erro estacionário de $-9,8\%$, bem próximos dos valores desejados de 50% e -10%, respectivamente. Em ambas as ações fisicamente furtivas, a acurácia do ataque garante que estas não evoluam para alterações de comportamento fisicamente mais perceptíveis.

Como trabalho futuro, encorajamos a pesquisa de técnicas capazes de evitar, ou dificultar, ataques fisicamente furtivos planejados com dados obtidos por ataques *System Identification*. Neste sentido, planejamos investigar contramedidas que possam dificultar a obtenção de informações sobre os sistemas de controle físico-cibernéticos, as quais são essenciais para o planejamento de ataques furtivos e controlados.

Referências

Civicioglu, P. (2013). Backtracking search optimization algorithm for numerical optimization problems. *Applied Mathematics and Computation*, 219(15):8121–8144.

de Sá, A. O., Nedjah, N., and de Macedo Mourelle, L. (2016). Distributed efficient localization in swarm robotic systems using swarm intelligence algorithms. *Neurocomputing*, 172:322–336.

El-Sharkawi, M. and Huang, C. (1989). Variable structure tracking of dc motor for high performance applications. *Energy Conversion, IEEE Transactions on*, 4(4):643–650.

Farooqui, A. A., Zaidi, S. S. H., Memon, A. Y., and Qazi, S. (2014). Cyber security backdrop: A scada testbed. In *Computing, Communications and IT Applications Conference (ComComAp), 2014 IEEE*, pages 98–103. IEEE.

Hussain, A., Heidemann, J., and Papadopoulos, C. (2003). A framework for classifying denial of service attacks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 99–110. ACM.

Hwang, H., Jung, G., Sohn, K., and Park, S. (2008). A study on mitm (man in the middle) vulnerability in wireless network using 802.1 x and eap. In *Information Science and Security, 2008. ICISS. International Conference on*, pages 164–170. IEEE.

Khatri, S., Sharma, P., Chaudhary, P., and Bijalwan, A. (2015). A taxonomy of physical layer attacks in manet. *International Journal of Computer Applications*, 117(22).

Langner, R. (2011). Stuxnet: Dissecting a cyberwarfare weapon. *Security & Privacy, IEEE*, 9(3):49–51.

Long, M., Wu, C.-H., and Hung, J. Y. (2005). Denial of service attacks on network-based control systems: impact and mitigation. *Industrial Informatics, IEEE Transactions on*, 1(2):85–96.

Ramos, C., Vale, Z., and Faria, L. (2011). Cyber-physical intelligence in the context of power systems. In *Future Generation Information Technology*, pages 19–29. Springer.

Smith, R. (2011). A decoupled feedback structure for covertly appropriating networked control systems. In *Proceedings of the 18th IFAC World Congress 2011*, volume 18. IFAC-PapersOnLine.

Smith, R. S. (2015). Covert misappropriation of networked control systems: Presenting a feedback structure. *Control Systems, IEEE*, 35(1):82–92.

Teixeira, A., Shames, I., Sandberg, H., and Johansson, K. H. (2015). A secure control framework for resource-limited adversaries. *Automatica*, 51:135–148.

Tran, T., Ha, Q. P., and Nguyen, H. T. (2007). Robust non-overshoot time responses using cascade sliding mode-pid control. *Journal of Advanced Computational Intelligence and Intelligent Informatics*.

A framework for searching encrypted databases

Pedro Geraldo M. R. Alves, Diego F. Aranha

Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Av. Albert Einstein, 1251 – CEP 13083-852, Campinas/SP – Brazil

{pedro.alves, dfaranha}@ic.unicamp.br

Abstract. *Cloud computing is a ubiquitous paradigm. It has been responsible for a fundamental change in the way distributed computing is performed. The possibility to outsource the installation, maintenance and scalability of servers, added to competitive prices, makes this platform highly attractive to the industry. Despite this, privacy guarantees are still insufficient for data processed in the cloud, since the data owner has no real control over the processing hardware. This work proposes a framework for database encryption that preserves data secrecy on an untrusted environment and retains searching and updating capabilities. It employs order-revealing encryption to provide selection with time complexity in* $\Theta(\log n)$*, and homomorphic encryption to enable computation over ciphertexts. When compared to the current state of the art, our approach provides higher security and flexibility. A proof-of-concept implementation on top of MongoDB is offered and presents an 11-factor overhead for a selection query over an encrypted database.*

1. Introduction

Cloud computing is a ubiquitous paradigm. From mobile to scientific computing, the industry increasingly embraces cloud services and take advantage of their potential to improve availability and reduce operational costs [Hoffa et al. 2008, Dinh et al. 2013]. Despite this, the cloud cannot be blindly trusted since malicious parties may have full access to the server and consequently to data. This includes external entities exploiting vulnerabilities, governmental institutions requesting information, and even curious cloud administrators. The data owner has no real control over the processing hardware and therefore cannot guarantee the secrecy of data [Xiao and Xiao 2013].

The risk of confidentiality breach caused by insecure use of cloud computing is real. As examples, we can mention PRISM, a surveillance program from the USA government that forced cloud companies to provide user data [Greenwald and MacAskill 2013, Weber 2014]; the Ashley Madison case, when personal data of millions of users was leaked by malicious parties exploiting security vulnerabilities [Thomsen 2015]; and the Yahoo data breach, possibly the largest known and affecting about 500 million accounts [BBC News 2016]. These security issues contribute to a growing crisis of confidence and leads domestic and corporate users to slow down the adoption of the cloud. There are estimates that the disclosures about PRISM may imply in a reduction of financial income from U$ 35 billion to U$ 180 billion for the cloud computing market in 2016 [Miller 2014]. This could be avoided if data was encrypted by the user and kept this way all the time, staying secret to the application and the cloud.

The problem of using standard encryption in an entire database is that it eliminates the capability to select records or evaluate arbitrary functions without the cryptographic keys. This reduces the cloud to a complex and huge storage service, discarding several

advantages it offers. Searchable encryption enables the cloud to manipulate encrypted data on behalf of a client without learning information. Hence, it solves both of aforementioned problems, keeping confidentiality in regard to the cloud but retaining some of its interesting features.

This work follows the state of the art and proposes directives to the modeling of a searchable encrypted database [Popa et al. 2011, Bösch et al. 2014]. We define the main primitives of a relational algebra necessary to keep the database functional, while adding enhanced privacy-preserving properties. A set of cryptographic tools is used to construct each of these primitives. It is composed by order-revealing encryption to enable data selection, homomorphic encryption for evaluation of arbitrary functions, and a standard symmetric scheme to protect and add flexibility to the handling of general data. In particular, our proposal of a selection primitive achieves time complexity of $\Theta(\log n)$. Moreover, we provide a security analysis and performance evaluation to estimate the impact on execution time and space consumption, and a conceptual implementation that validates the framework. It works on top of MongoDB, a popular document-based database, and is implemented as a wrapper over its Python driver. Its source code was made available to the community under a GNU GPLv3 license [Alves 2016].

When compared to CryptDB [Popa et al. 2011], our proposal provides stronger security since it is able to keep data confidentiality even in the case of a compromise of the database and application servers. Since CryptDB delegates to the application server the capability to derive users' cryptographic keys, it is not able to provide such security guarantees. Futhermore, our work is database-agnostic. This way, it is not limited to SQL but can be applied on different key-value-like databases.

2. Building blocks

Next, we revisit basic security notions and special properties that make a cryptosystem suitable to a certain context.

2.1. Security notions

Ciphertext indistinguishability is an important property that can be used to analyze the security of a cryptosystem. Two scenarios are considered, when an adversary has and does not have access to an oracle that provides decryption capabilities. Usually these scenarios are evaluated through a game in which an adversary tries to acquire knowledge about ciphertexts generated by a challenger [Bellare et al. 1998].

Indistinguishability under chosen plaintext attack - IND-CPA. In the IND-CPA game the challenger generates a pair (PK,SK) of cryptographic keys, makes PK public and keeps SK secret. An adversary has as objective to recognize a ciphertext created from a randomly chosen message from a known two-element message set. A polynomially bounded number of operations is allowed, including encryption (but not decryption), over PK and the ciphertexts. A cryptosystem is indistinguishable under chosen plaintext attack if no adversary is able to achieve the objective with non-negligible probability.

Indistinguishability under chosen ciphertext attack/adaptive chosen ciphertext attack - IND-CCA and IND-CCA2 This type of indistinguishability differs from IND-CPA due

to the adversary having access to a decryption oracle. In this game the challenge is again to recognize a ciphertext as described before, but now the adversary is able to use decryption results. This new game has two versions, non-adaptive and adaptive. In the non-adaptive version, IND-CCA, the adversary may use the decryption oracle until it receives the challenge ciphertext. On the other hand, in the adaptive version he is allowed to use the decryption oracle even after that event. For obvious reasons, the adversary cannot send the challenge ciphertext to the decryption oracle. A cryptosystem is indistinguishable under chosen ciphertext attack/adaptive chosen ciphertext attack if no adversary is able to achieve the objective with non-negligible probability.

Indistinguishability under an ordered chosen plaintext attack - IND-OCPA Introduced by Boldyreva et al., this notion supposes that an adversary is capable of retrieving two sequences of ciphertexts resulting of the encryption of any two sequences of messages [Boldyreva et al. 2009]. Furthermore, he knows that both sequences have identical ordering. The objective of this adversary is to distinguish between these ciphertexts. A cryptosystem is indistinguishable under an ordered chosen plaintext attack if no adversary is able to achieve the objective with non-negligible probability.

2.2. Order-revealing encryption (ORE)

Order-revealing encryption schemes are characterized by having, in addition to the usual set of cryptographic functions like *keygen* and *encrypt*, a function capable of comparing ciphertexts and returning the order of the original plaintext, as shown by Definition 1.

Definition 1 (Order-revealing encryption) *Let E be an encryption function, C be a comparison function, and m_1 and m_2 be plaintexts from the message space. The pair (E, C) is defined as an encryption scheme with the order-revealing property if:*

$$C(E(m_1), E(m_2)) = \begin{cases} \text{LOWER}, & \text{if } m_1 < m_2, \\ \text{EQUAL}, & \text{if } m_1 = m_2, \\ \text{GREATER}, & \text{otherwise.} \end{cases}$$

This is a generalization of order-preserving encryption (OPE), that fixes C to a simple numerical comparison [Boneh et al. 2015].

Security As argued by Lewi and Wu, the "best-possible" notion of security for ORE is IND-OCPA, what means that it is possible to achieve indistinguishability of ciphertexts. This property implies a much stronger security guarantee than OPE schemes can have [Lewi and Wu 2016]. Furthermore, differently from OPE, ORE is not inherently deterministic [Kolesnikov and Shikfa 2012]. For example, Chenette et al. propose an ORE scheme that applies a pseudo-random function over an OPE scheme, while Lewi and Wu propose an ORE scheme completely built upon symmetric primitives, capable of limiting the use of the comparison function and reducing the leakage inherent to this routine [Chenette et al. 2016, Lewi and Wu 2016]. Nevertheless, any scheme that reveals numerical order of plaintexts through ciphertexts is vulnerable to inference attacks and frequency analysis, as those described by Naveed et al. over relational databases encrypted using deterministic and OPE schemes [Naveed et al. 2015]. Although ORE does not completely discard the possibility of such attacks, it offers stronger defenses.

2.3. Homomorphic encryption (HE)

Homomorphic encryption schemes have the property of conserving some plaintext structure during the encryption process, allowing the evaluation of certain functions over ciphertexts and obtaining, after decryption, a result equivalent to the same computation applied over plaintexts. Definition 2 presents this property in a more formal way.

Definition 2 (Homomorphic encryption) *Let E and D be a pair of encryption and decryption functions, and m_1 and m_2 be plaintexts. The pair (E, D) forms an encryption scheme with the homomorphic property for some operator $\diamond$ if and only if the following holds:*

$$E(m_1) \circ E(m_2) \equiv E(m_1 \diamond m_2).$$

The operation $\circ$ in the ciphertext domain is equivalent to $\diamond$ in the plaintext domain.

Homomorphic cryptosystems are classified according to the supported operations and their limitations. *Partially homomorphic encryption* schemes (PHE) hold on Definition 2 for either unlimited addition or multiplication operations, while *fully homomorphic encryption* schemes (FHE) support both addition and multiplication operations.

Security In terms of security, homomorphic encryption schemes achieve at most IND-CCA-1 [Bellare et al. 1998]. This is a natural consequence of the design requirements, since these cryptosystems allow any entity to manipulate ciphertexts. Most of current proposals, however, reach at most IND-CPA and stay secure against attackers without access to a decryption oracle [Loftus et al. 2012].

3. Searchable encryption

In this section, the problem of searching over encrypted data is formally defined. We present two state-of-the-art solutions to this problem, namely the CryptDB and Arx database systems.

3.1. The problem

Suppose a scenario where Alice stores a set of documents with an untrusted entity Bob. She would like to keep this data encrypted because, as defined, Bob is not trustworthy. Alice also would like to occasionally retrieve a subset of documents accordingly to a predicate without revealing any sensitive information to Bob. Thus, sharing the decryption key is not an option. The problem lies in the fact that communication between Alice and Bob may (and probably will) be constrained. Hence, a naive solution consisting of Bob sending all documents to Alice and letting her decrypt and select whatever she wants may not be feasible. Alice must then implement some mechanism to protect her encrypted data so that Bob will be able to identify the desired documents without knowing their contents or the selection criteria [Song et al. 2000].

An approach that Alice can take is to create an encrypted index as in Definition 3.

Definition 3 (Searching on an encrypted index) *Suppose a database $\mathcal{DB} = (m_1, \ldots, m_n)$ and a list $\mathcal{W} = (W_1, \ldots, W_m)$ of sets of keywords such that W_i contains keywords for m_i. The following routines are needed to build and search on an encrypted index:*

BUILDINDEX$_K(\mathcal{DB}, \mathcal{W})$**:** *The list $\mathcal{W}$ is encrypted using a searchable scheme under a key K and results in a searchable encrypted index $\mathcal{I}$. This process may not be reversible (e.g., if a hash function is used). The routine outputs $\mathcal{I}$.*

TRAPDOOR$_K(\mathcal{F})$**:** *This function receives a predicate $\mathcal{F}$ and outputs a trapdoor $\mathcal{T}$. The latter is defined as the information needed to search $\mathcal{I}$ and find records that satisfy $\mathcal{F}$.*

SEARCH$_{\mathcal{I}}(\mathcal{T})$**:** *It iterates through $\mathcal{I}$ applying T and outputs every record that returns* TRUE *for the input trapdoor.*

This way, Alice is able to keep her data stored with Bob and remain capable of selecting subsets of it without leaking information [Bösch et al. 2014].

3.2. CryptDB

CryptDB is a software layer that provides capabilities to store data in a remote database and query over it without revealing sensitive information to the database management system (DBMS). It introduces a proxy layer responsible to encrypt and adjust queries to the database and decrypt the outcome [Popa et al. 2011].

The context in which CryptDB stands is a typical structure of database-backed applications, consisting of a DBMS server and a separate application server. To query a database, a predicate is generated by the application and processed by the proxy before it is sent to the DBMS server. The user interacts exclusively with the application server and is responsible for keeping the password secret. This password is provided on login to the proxy (via application) that derives all the keys required to interact with the database. When the user logs out, it is expected that the proxy deletes their keys.

Data encryption is done through "onions". These are defined as layers of encryption that are combined to provide different functionalities. Modeling a database involves evaluating the meaning of each attribute and predicting the operations it must support. In particular, keyword-searching as described in Definition 3 is implemented as proposed in Song's work [Song et al. 2000].

The authors address two types of threats with CryptDB: curious database administrators who try to snoop and acquire information about client's data; and an adversary that gains complete control of application and DBMS servers. The first threat is achieved through the encryption of stored data and the ability to query it without any decryption or knowledge about its content. The second threat, as the authors claims, applies only to logged-out clients. In the described scenario, the cryptographic keys relative to data in the database are handled by the application server. Thus, if the application server is compromised, all the keys it possesses at that moment (that are expected to be only from logged-in users) are leaked to the attacker.

3.3. Arx

Arx is a database system alternative to CryptDB [Poddar et al. 2016]. It targets much stronger security properties and claims to protect the database with the same level of regular AES-based encryption[1], achieving IND-CPA security. This is a direct consequence of the

[1]The Advanced Encryption Standard (AES) is a well-established symmetric block cipher enabling high performance implementation in hardware and software [Daemen and Rijmen 1999].

almost exclusively use of AES to construct selection operators, even on range queries. This not only brings strong security but also good performance, due to efficiency of symmetric primitives, sometimes even benefiting from hardware implementations. The building blocks used for searching follow those described previously. Futhermore they apply a different AES key for each keyword when generating the trapdoor.

At its core, Arx introduces two database indexes, ARX-RANGE for range and order-by-limit queries and ARX-EQ for equality queries, both built on top of AES. The former uses an obfuscation strategy to protect the data it compares and does not leak any information, while enabling searches in logarithmic time. The latter embeds a counter into each repeating value. This ensures that the encryption of both are different, protecting it against frequency analysis. Using a small token provided by the client, the database is able to expand it in many search tokens and return all the occurrences desired, allowing an index to be built over encrypted data.

The context in which Arx stands is similar to CryptDB. However, the authors consider the data owner as the application itself. This way, it simplifies the security measures and considers the responsibility to keep the application server secure outside of its scope.

4. Proposed framework

The goal of the proposed framework is to develop a database model capable of storing encrypted records and applying relational algebra primitives on it without the knowledge of any cryptographic keys or the need for decryption. A trade-off between performance and security is desirable, however we completely discard deterministic encryption whenever is possible for security reasons. The applicability of this framework goes beyond SQL databases. Besides the relational algebra hereby used to describe the framework, it can be extended to key-value, document-oriented, full text and several other databases classes that keeps the same attribute structure.

The three main operations needed to build a useful database are insertion, selection and update. Once data is loaded, being able to select only those pieces that correspond to an arbitrary predicate is the fundamental block to construct more complex operations, as grouping and equality joins. This functionality is fundamental when there is a physical separation between the database and the data owner, otherwise high demand for bandwidth is incurred to transmit large fractions of the database records. Furthermore, real data is frequently mutable and thus the database must support updates to remain useful.

We define as *secure* a system model that guarantees that the data owner is the only entity capable of revealing data, which can be achieved by his exclusive possession of the cryptographic keys. Thus, a fundamental aspect of our proposal is the scenario in which the database and the application server handle data with minimum knowledge.

Lastly, the framework does not ensure integrity, freshness or completeness of results returned to the application or the user, since an adversary that compromises the database in some way can delete or alter records. We consider this threat to be outside the scope of this framework.

4.1. Classes of attributes

Records in an encrypted database are composed by attributes. These consist of a name and a value, that can be an integer, float, string or even a binary blob. Values of attributes are

classified according to their purpose:

static An immutable value only used for storage. It is not expected to be evaluated with any function, so there is no special requirement for the encryption.

index Used for building a single or multivalued searchable index. It should enable one to verify if an arbitrary term is contained in a set without the need to acquire knowledge of their content.

computable A mutable value. It supports the evaluation with arithmetic circuits and ensures obtaining, after decryption, a result equivalent to the same circuit applied over plaintexts.

The implementation of each attribute must satisfy the requirements without leaking any vital information beyond those related directly with the attribute objective (i.e.: order for *index* attributes). Since the name of an attribute reveals information, it may need to be protected as well. However, the acknowledgement of an attribute is done using its name, so even anonymous attributes must be traceable in a query. An option for anonymizing the attribute name is to treat it as an *index*.

The aforementioned cryptosystems are natural suggestions to be applied within these classes. Since *static* is a class for storage only and has no other requirements, any scheme with appropriate security levels and performance may be used, as AES. On the other hand, *index* and *computable* attributes are immediate applications of ORE and HE schemes. Particularly, the latter defines the HE scheme according to the required operations. Attributes that require only one operation can be implemented with a PHE scheme, which provides good performance; while those that requires addition and multiplication must use FHE and deal with the performance issues.

Definition 4 (Secure ORE) *Let E and C be, respectively, an encryption and a comparison function. The pair (E, C) forms an encryption scheme with the order-revealing property defined as "secure" if and only if it satisfies Definition 1; the encryption of a message m can be written as $E(m) = (c_L, c_R) = (E_L(m), E_R(m))$, where E_L and E_R are complementary encryption functions; and the comparison between two ciphertexts c_1 and c_2 is done by $C(c_{L1}, c_{R2})$. This way, C may be applied without the complete knowledge of the ciphertexts.*

In order to build a secure and efficient *index*, an ORE scheme that corresponds to Definition 4 should be used. We define the search framework as in Definition 5.

Definition 5 (Encrypted search framework) *Let S be a set of words, sk a secret key, and an ORE scheme (*ENC, CMP*) that satisfies Definition 4. The operations required for an encrypted search over S are defined as follows:*

BUILDINDEX$_{sk}(S)$ *Output the set*

$$S^* = \{c_R \mid (c_L, c_R) = \text{ENC}_{sk}(w), \forall w \in S\}.$$

TRAPDOOR$_{sk}(w)$ *Output the trapdoor*

$$T_w = (c_L \mid (c_L, c_R) = \text{ENC}_{sk}(w)).$$

SEARCH$_{S^*,\,r}(T_w)$ *To select all records in* S^* *with the relation* $r \in \{$LOWER, EQUAL, GREATER$\}$ *to a word* w*, one computes the trapdoor* T_w *and iterates through* S^* *looking for the records* $w^* \in S^*$ *that satisfy*

$$\textsc{Cmp}\,(T_w, w^*) = r.$$

The set $\hat{S}$ *with all the elements in* S^* *that satisfy this equation is returned.*

4.2. Database operations

A relational algebra for database operations can be built over six main operations to query on related sets of data: selection, projection, rename, Cartesian product, union and difference [Codd 1983]. They are defined as follows.

1. **Selection** (σ): The selection of entries in the database is done exclusively using *index* attributes.
   (a) *index*: The user wants all records with the relationship r when compared to an arbitrary word w in an *index* attribute, where $r \in \{$LOWER, EQUAL, GREATER$\}$. The trapdoor $T_w = Trapdoor_{sk}(w)$ is sent to the server that executes SEARCH for the desired r.
2. **Projection** (π): Attribute names may or may not be encrypted.
   (a) *encrypted*: If encrypted, a deterministic scheme is used or they are treated as *index* values.
      i. *deterministic scheme*: The user defines a set A of attribute names and computes $A^* = \{Enc(a) \mid a \in A\}$ using the encryption function related to the deterministic scheme chosen. A^* is sent to the server that returns only the records with names in it.
      ii. *index*: The user defines a set A of attribute names, computes $A^* = \{Trapdoor_{sk}(a) \mid a \in A\}$ and sends it to the server, that selects the projected attributes through the operation SEARCH.
   (b) *unencrypted*: Unencrypted attribute names may be sent to and selected by the server using a standard algorithm.
3. **Rename** (ρ): Attribute names may or may not be encrypted.
   (a) *encrypted*: If encrypted, a deterministic scheme is used or they are treated as *index* values.
      i. *deterministic scheme*: The user picks an attribute A and the new name B and computes $(A^*, B^*) = (Enc(A), Enc(B))$ using the encryption function related to the deterministic scheme chosen. (A^*, B^*) is sent to the server that applies a standard algorithm.
      ii. *index*: The user picks an attribute A and the new name B and computes $(A^*, B^*) = (Trapdoor_{sk}(A), (c_R \mid (c_L, c_R) = \text{Enc}_{sk}(B)))$. (A^*, B^*) is sent to the server, that selects attributes related to A^* as EQUAL through the operation SEARCH and renames the result to B^*.
   (b) *unencrypted*: Unencrypted attribute names may be renamed by the server using a standard algorithm.
4. **Cartesian product** ($\times$): The Cartesian product of two datasets encrypted with the same keys is executed using a standard algorithm.

5. **Difference** ($-$): The difference between two datasets A and B encrypted with the same keys is defined as $A - B = \sigma_{\text{not in B}}(A)$, that means a selection in A of all elements not contained in B.
6. **Union** ($\cup$): The union of two datasets encrypted with the same keys is defined as $A \cup B = A + (B - A)$, where $+$ is the usual set operator.

In addition, three other important operations must be defined to complete the set of operators:

7. **Insert**: Encrypted data is provided and added to the database using a standard algorithm.
8. **Intersect** ($\cap$): The intersection of two sets A and B encrypted with the same keys is defined as $A \cap B = \sigma_{\text{in B}}(A)$.
9. **Update**: An update operation is defined as a selection followed by the evaluation of a *computable* attribute by a supported homomorphic operation.

This set of operators enables operating over an encrypted database without the knowledge of cryptographic keys or acquiring sensitive information from user queries.

4.3. Security analysis

We assume the scenario in which the data owner has exclusive possession of cryptographic keys. This way, insertions to the database must be locally encrypted before being sent to the server. The database or the application never deals with plaintext data. Our framework thus has the advantage over CryptDB of preserving privacy even in the outcome of a compromised database or application server.

Despite being conceptual similar to OPE, ORE is able to address several security limitations of it. ORE does not necessarily generate ciphertexts that reveal their order by design, but allows someone to protect this information and only reveals it through specific functions. ORE is able to achieve the IND-OCPA security notion and adds randomization to ciphertexts. Those characteristics make it much safer against inference attacks [Naveed et al. 2015]. The proposal of Lewi and Wu goes even beyond that and is capable of limiting the use of the comparison function [Lewi and Wu 2016]. Their scheme generates a ciphertext that can be decomposed into left and right components such that a comparison between two ciphertexts requires only a left component of one ciphertext and the right component of the other. This way, the authors argue that robustness against such attacks is ensured since the database dump may only contain the right component, that is encrypted using semantically-secure encryption.

Nonetheless, an eavesdropper is capable of recognizing repeated queries by observing the outcome of a selection. This weakness may still be used for inference attacks, that can breach confidentiality from related attributes. This issue can get worse if the trapdoor is deterministic, when there is no other solution than implementing a key refreshment algorithm. Besides that, the knowledge of the numerical order between every pair of elements in a sequence may leak information depending on the application. This problem manifests itself in our proposal on the σ primitive if it uses a weak index structure, like a naive sequential index. A balanced-tree-based structure, on the other hand, obscures the numerical order of elements in different branches. This way, an attacker is capable of recovering the order of up to $O(\log n)$ database elements and somewhat infer about the others, in a database with n elements.

Finally, BUILDINDEX is not able to hide the quantity of records that share the same index. This way, one is able to make inferences about those by the number of records. There is also no built-in protection for the number of entries in the database. A workaround is to fix the size of each *static* attribute value and round the quantity of records in the database using padding. This approach increases secrecy but also the storage overhead.

4.4. Performance analysis

The application of ORE as the main approach to build a database index provides an extremely important contribution to selection queries. SEARCH does not require walking through all the records testing a trapdoor, but only a logarithmic-smaller subset of it when implemented over an optimal index structure, as an AVL tree or B-tree based structure [Sedgewick 1983]. This characteristic is highlighted on union, intersection and difference operations, that work by comparing and selecting elements in different groups. Moreover, current proposals in the state of the art of ORE enjoys the good performance provided by symmetric primitives and does not require more expensive approaches such as public-key cryptography [Chenette et al. 2016, Lewi and Wu 2016, Boneh et al. 2015]. In particular, although fully homomorphic cryptosystems promises to fulfill this task, it is still prohibitively expensive for real-world deployments [Boneh et al. 2013].

Space consumption is also affected. Ciphertexts are computed as a combination of the plaintext with random data. This way, a non-trivial expansion rate is expected. Differently from speed overheads which are affected by a single attribute type, all attributes suffer with the expansion rate of encryption.

4.5. Capabilities and limitations

Our framework is capable of providing an always-encrypted database that preserves secrecy as long as the data owner keeps the cryptographic keys secure. One is able to select records through *index* and apply arbitrary operations on attributes defined as *computable*. Furthermore, it increases the security of data but maintaining the computational complexity of standard relational primitives, achieving a fair trade-off between security and performance.

Although the framework has no constraints about attributes classified as both *index* and *computable*, there is no known encryption scheme in the literature capable of satisfying all the requirements. This way, the relational model of the database must be as precise as possible when assigning attributes to each class, specially because the costs of a model refactor can be prohibitive.

Some scenarios appear to be more compatible with an encrypted database as described than others. An e-mail service, for example, can be trivially adapted. The e-mails received by a user are stored in encrypted form as *static* and some heuristic is applied on its content to generate a set of keywords to be used on BUILDINDEX. This heuristic may use all unique words in the e-mail, for example. The sender address may be an important value for querying as well, so it may be stored as an *index*. To optimize common queries, a secondary collection of records may be instantiated with, for example, counters. The quantity of e-mails received from a particular sender, how often a term appears or how many messages are received in a time frame. Storing this metadata information in a secondary data collection avoids some of the high costs of searching in the main dataset.

However, our proposal fails when the user wants to search for something that was not previously expected. For example, regular expressions. Suppose a query that searches

for all the sentences that start with “Attack” and end with “dawn”, or all the e-mails on the domain “gmail.com”. If these patterns were not foreseen when the keyword index was built, then no one will be able to correctly execute this selection without the decryption of the entire database. Since the format of the strings is lost on encryption, this kind of search is impossible on our proposal.

5. Implementation

A proof-of-concept implementation of the proposed framework was developed aiming at the popular document-based database MongoDB [Chodorow and Dirolf 2010] and made available to the community under a GNU GPLv3 license [Alves 2016]. It is written in Python and works as a wrapper on its driver. This way, it becomes database-agnostic and limits the database server to dealing with encrypted data. Table 1 provides the schemes used for each attribute class, the parameter size and its security level.

Table 1. Chosen cryptosystems for each attribute presented in Section 4.

Attribute	Cryptosystem	Parameters	Security level
static	AES	128 bits	128 bits
index	Lewi-Wu	128 bits	128 bits
computable - add	Paillier	3072 bits	128 bits
computable - mul	ElGamal	3072 bits	128 bits

Lewi-Wu’s ORE scheme relies on symmetric primitives and achieves IND-OCPA. The authors claim that this is more secure than all existing OPE and ORE schemes which are practical [Lewi and Wu 2016]. Finally, Paillier and ElGamal are known public-key schemes. Both achieve IND-CPA and are based in the integer factorization and discrete logarithm problems, respectively. Paillier supports homomorphic addition, while ElGamal provides homomorphic multiplication. Both are defined as PHE schemes [Paillier 1999, ElGamal 1985]. The implementation of AES was provided by *pycrypto* toolkit [Litzenberger 2016], while Lewi-Wu, Paillier and ElGamal were implemented with our own code.

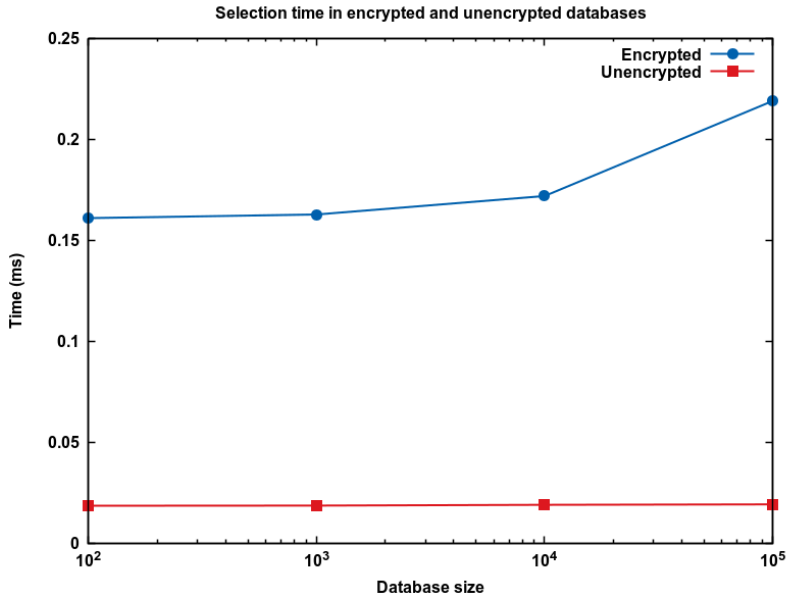
Table 2. Attribute structure of elements in the synthetic dataset.

Name	Value type	Class
e-mail	string	*static*
firstname	string	*static*
surname	string	*static*
country	string	*static*
age	integer	*index*
text	string	*static*

To measure the computational costs of managing an encrypted database, four synthetic variable-sized datasets were generated with the structure described in Table 2. Each one was loaded in encrypted and unencrypted form to a stock MongoDB database using insertion element-by-element by a Python script. An AVL tree was used as index for the documents through the attribute “age”. While it was possible to index the unencrypted database natively, it was not so simple with the encrypted version. MongoDB is not friendly to custom index structures or comparators, so we decided to construct the structure with Python

code and then insert it into the database using pointers based on MongoDB's native identity codes. This way, walking through the index tree depends on a database-external operation at Python-side, calling MongoDB's FIND method to localize documents related to left/right pointers starting from the tree root. Encrypting this document structure took 2.13ms in a Intel Xeon E5-2630 CPU at 2.60GHz. As can be seen in Figure 1, the performance overhead on queries in the encrypted database goes from 7 to 11 times.

Figure 1. Time required to perform a selection query on encrypted and unencrypted databases in the worst case scenario for an AVL tree index. The measures are the average of 100 independent executions.



Besides the need to walk through the AVL tree using database-external operations, these results are comparable to those related to Arx [Poddar et al. 2016] and one magnitude higher than CryptDB [Popa et al. 2011]. This way, it is expected that an efficient implementation, capable of executing searches completely inside MongoDB, will present an expressive speedup compared with the state of the art.

6. Conclusion

We presented the problem of searching in encrypted data and a proposal of a framework that guides the modeling of a database with support to this functionality. This is achieved by combining different cryptographic concepts and using different cryptosystems to satisfy the requirements of each attribute, like order-revealing encryption and homomorphic encryption. Over this approach, six main relational algebra operations were redefined to support encrypted data: selection, projection, rename, Cartesian product, union and difference. An overview of the security provided is discussed, as well as an analysis about the impact in database performance. We present a proof-of-concept implementation in Python over the document-based database MongoDB. A selection query on the worst case scenario was up to 11 times slower on the encrypted database. In comparison with CryptDB our proposal provides higher security, since it delegates exclusively to the data owner the responsibility of encrypting and decrypting data. This way, privacy holds even in a scenario of database or application compromised.

As future work, we intend to focus on performance and pursuit an efficient implementation of this framework.

References

[Alves 2016] Alves, P. (2016). A proof-of-concept searchable encryption backend for mongodb. `https://github.com/pdroalves/encrypted-mongodb`. Last accessed: 09/08/2016.

[BBC News 2016] BBC News (2016). Yahoo 'state' hackers stole data from 500 million users. Last accessed: 23/09/2016.

[Bellare et al. 1998] Bellare, M., Desai, A., Pointcheval, D., and Rogaway, P. (1998). *Advances in Cryptology — CRYPTO '98: 18th Annual International Cryptology Conference Santa Barbara, California, USA August 23–27, 1998 Proceedings*, chapter Relations among notions of security for public-key encryption schemes, pages 26–45. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Boldyreva et al. 2009] Boldyreva, A., Chenette, N., Lee, Y., and O'Neill, A. (2009). Order-preserving symmetric encryption. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5479 LNCS:224–241.

[Boneh et al. 2013] Boneh, D., Gentry, C., Halevi, S., Wang, F., and Wu, D. J. (2013). Private database queries using somewhat homomorphic encryption. In *Proceedings of the 11th International Conference on Applied Cryptography and Network Security*, ACNS'13, pages 102–118, Berlin, Heidelberg. Springer-Verlag.

[Boneh et al. 2015] Boneh, D., Lewi, K., Raykova, M., Sahai, A., Zhandry, M., and Zimmerman, J. (2015). *Semantically Secure Order-Revealing Encryption: Multi-input Functional Encryption Without Obfuscation*, pages 563–594. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Bösch et al. 2014] Bösch, C., Hartel, P., Jonker, W., and Peter, A. (2014). A survey of provably secure searchable encryption. *ACM Comput. Surv.*, 47(2):18:1–18:51.

[Chenette et al. 2016] Chenette, N., Lewi, K., Weis, S. A., and Wu, D. J. (2016). Practical order-revealing encryption with limited leakage. In FSE.

[Chodorow and Dirolf 2010] Chodorow, K. and Dirolf, M. (2010). *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 1st edition.

[Codd 1983] Codd, E. F. (1983). A relational model of data for large shared data banks. *Commun. ACM*, 26(6):64–69.

[Daemen and Rijmen 1999] Daemen, J. and Rijmen, V. (1999). AES Proposal: Rijndael.

[Dinh et al. 2013] Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13(18):1587–1611.

[ElGamal 1985] ElGamal, T. (1985). A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In Blakley, G. and Chaum, D., editors, *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer Berlin Heidelberg.

[Greenwald and MacAskill 2013] Greenwald, G. and MacAskill, E. (2013). NSA Prism program taps in to user data of Apple, Google and others.

[Hoffa et al. 2008] Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., and Good, J. (2008). On the Use of Cloud Computing for Scientific Workflows. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 640–645.

[Kolesnikov and Shikfa 2012] Kolesnikov, V. and Shikfa, A. (2012). On the limits of privacy provided by Order-Preserving Encryption. *Bell Labs Technical Journal*.

[Lewi and Wu 2016] Lewi, K. and Wu, D. J. (2016). Order-revealing encryption: New constructions, applications, and lower bounds. Cryptology ePrint Archive, Report 2016/612.

[Litzenberger 2016] Litzenberger, D. (2016). Python Cryptography Toolkit. `http://www.pycrypto.org/`. Last accessed: 07/03/2016.

[Loftus et al. 2012] Loftus, J., May, A., Smart, N. P., and Vercauteren, F. (2012). On CCA-Secure Somewhat Homomorphic Encryption. In *Proceedings of the 18th International Conference on Selected Areas in Cryptography*, SAC'11, pages 55–72, Berlin, Heidelberg. Springer-Verlag.

[Miller 2014] Miller, C. C. (2014). Revelations of N.S.A. spying cost U.S. tech companies. *The New York Times*. Last accessed: 02/04/2016.

[Naveed et al. 2015] Naveed, M., Kamara, S., and Wright, C. V. (2015). Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 644–655, New York, NY, USA. ACM.

[Paillier 1999] Paillier, P. (1999). Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In Stern, J., editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer Berlin Heidelberg.

[Poddar et al. 2016] Poddar, R., Boelter, T., and Popa, R. A. (2016). Arx: A strongly encrypted database system. Cryptology ePrint Archive, Report 2016/591.

[Popa et al. 2011] Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. (2011). Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 85–100, New York, NY, USA. ACM.

[Sedgewick 1983] Sedgewick, R. (1983). *Algorithms*, chapter 15, page 199. Addison-Wesley.

[Song et al. 2000] Song, D. X., Wagner, D., Perrig, A., and Perrig, A. (2000). Practical techniques for searches on encrypted data. *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, pages 44–55.

[Thomsen 2015] Thomsen, S. (2015). Extramarital affair website Ashley Madison has been hacked and attackers are threatening to leak data online. Last accessed: 25/05/2016.

[Weber 2014] Weber, H. (2014). How the NSA & FBI made Facebook the perfect mass surveillance tool. Venture Beat. Published on 05/15/2014.

[Xiao and Xiao 2013] Xiao, Z. and Xiao, Y. (2013). Security and Privacy in Cloud Computing. *IEEE Communications Surveys Tutorials*, 15(2):843–859.

Detecção de Ataque DDoS Flash Crowd Realizando Análise Comportamental de Solicitações Web

Samuel Lautert Jardim[1] **Raul Ceretta Nunes**[1,2] **Marcelo Colomé**[1]

[1]Programa de Pós-Graduação em Computação [2]Departamento de Computação Aplicada

Universidade Federal de Santa Maria, Av. Roraima, 1000, Camobi, Santa Maria, RS

samuel@saopedronet.com.br, ceretta@inf.ufsm.br,
marcelocolome@inf.ufsm.br

Abstract. *A Distributed Denial of Service (DDoS) attack is a threat to Internet good functioning. To make undetectable attacks, attackers (botmasters) are exploring the application layer as alternative to get similarity with benign network traffic. A DDoS Flash Crowd attack generates similar traffic to Flash Crowds events (outbreak of unexpected visits). Attack detection tools need to differentiate DDoS attack flows from flash crowd flows. This paper proposes a DDoS detection method based on the analysis of user request interactivity. The method differs a human user from a bot (malware) modeling the interactive behavior by analyzing the number of requests and the time between them (interactivity rate). The experiments demonstrate the detection effectiveness of the method and that the applicability of analysis of expected interactivity pattern as a detection mechanism.*

Resumo. *Um Ataques de Negação de Serviço Distribuído (DDoS) é uma ameaça para o bom funcionamento da Internet. Ataques na camada de aplicação, como DDoS Flash Crowd, vêm se consolidando como alternativa para botmasters tornarem seus ataques ainda mais indetectáveis, dado a similaridade com tráfego de rede benigno do tipo Flash Crowd (surto de visitas inesperadas). Ferramentas de detecção de ataques necessitam diferenciar um tráfego flash crowd de um tráfego com ataque DDoS. Este trabalho propõe um método de detecção baseado na observação do padrão de interatividade nas solicitações dos usuários. O método difere um usuário humano de um bot (programa malicioso) modelando o comportamento através do número de solicitações e do tempo entre elas (taxa de interatividade). Os experimentos demonstram a eficácia do método na detecção, comprovando que o padrão de interatividade esperado pode ser aplicado como mecanismo de detecção.*

1. Introdução

Inúmeras formas de ataques são conduzidas por grupos maliciosos para tornar falho ou inoperante um *website* ou serviço específico, podendo afetar uma cadeia de pessoas que usufruem do serviço de maneira direta ou indireta. Neste sentido, são utilizados muitos

métodos de ataques aos computadores servidores, onde se destaca o ataque de Negação de Serviço Distribuído (DDoS) [Yu et al. 2012][Dantas 2013][Dhingra e Sachdeva 2014].

Para a realização de ataques DDoS são utilizadas redes denominadas *botnets*, formadas por computadores distribuídos geograficamente e comprometidos com softwares maliciosos (*bots*) comandados por criminosos (*botmasters*), sem o consentimento e conhecimento dos proprietários dos computadores (hosts infectados). Segundo [Feily et al. 2009] o ataque possui dois métodos principais. O primeiro se aproveita de uma vulnerabilidade e envia pacotes mal formados ludibriando um protocolo ou aplicação. Já o segundo tem o objetivo de interromper a conectividade do usuário utilizando o esgotamento da largura de banda, capacidade dos roteadores ou recursos de rede, sendo conhecido como ataque por inundação de rede; ou também, esgotamento dos recursos do servidor, este conhecido como ataque de inundação de aplicação. Com o objetivo de dificultar a detecção de *botnets*, os *botmasters* utilizam-se de técnicas anti forenses para disfarçar peculiaridades que possam diferenciar traços de um ataque dos dados legítimos. Ofuscação de código, criptografia, comunicação *peer-to-peer* e imitação de eventos *Flash Crowd* são exemplos destas técnicas [Yu et al. 2012].

Um evento *Flash Crowd* corresponde a um pico de acesso legítimo a dado *website*, tal como pode ocorrer em sites de notícias durante grandes eventos. Este tipo de evento costuma gerar degradação de desempenho ou indisponibilidade com consequente insatisfação de usuários. Um ataque DDoS *Flash Crowd* é um tipo de ataque que procura imitar um comportamento *Flash Crowd*, ou seja, procura tornar o serviço indisponível a partir de rajadas de tráfego legítimo [Xie e Yu 2009][Ke et al. 2009] [Prasad et al. 2013]. Tanto *Flash Crowd* quanto DDoS *Flash Crowd* podem ser detectados analisando as características estatísticas do tráfego [Thapngam et al. 2011]. O desafio é adotar uma abordagem capaz de realizar a diferenciação adequada entre o comportamento de um *Flash Crowd* (gerado por humanos) e o de um ataque (gerado por *bots*) [Xie e Yu 2009][Yu et al. 2012] [Prasad et al. 2013].

O cálculo da correlação de *Pearson* aplicado às séries temporais do número de solicitações HTTP [Xu et al. 2012] tem sido o método mais aplicado para diferir o comportamento humano de um *bot* no caso de ataques DDoS *Flash Crowd*. Este método assume que os traços gerados pelos *bots* possuem entre sí uma correlação maior que os traços gerados por humanos [Thapngam et al. 2011]. Entretanto, cada página web possui um número de solicitações secundárias aleatórias, podendo fazer com que este método apresente muitos falsos positivos. Para minimizar o problema, Thapngam et al. (2011) propuseram um método no qual calculam a correlação de *Pearson* da taxa de chegada de pacotes dos *hosts* ao longo do tempo. Porém, na medida em que um servidor web estiver trabalhando com alta carga o comportamento da taxa de pacotes dos *hosts* pode se assemelhar bastante em função do *throughput* minimizado para cada *host*, o que pode comprometer o método. De maneira semelhante, mas considerando não mais a correlação de Person, Yu et al. (2012) perceberam a existência de maior similaridade nos fluxos de rede dos ataques DDoS *Flash Crowd* do que nos fluxos dos usuários humanos, possibilitando a detecção por análise da similaridade. No entanto, como reconhecido pelos próprios autores, a similaridade pode ser facilmente ludibriada pelos *botmasters* criando vários grupos de padrões de solicitações no código fonte, terminando com a alta similaridade de um grupo grande de *hosts* e tornando a similaridade entre os *hosts* da rede heterogênea.

O presente trabalho apresenta um método para diferenciação entre humano e *bot* baseado na análise comportamental da interação nas solicitações HTTP(S) de cada *host* cliente. O comportamento é modelado através do número de solicitações e do tempo entre solicitações, resultando na análise da taxa de interatividade. O método de diferenciação contrasta os comportamentos médios esperados para humanos com os de *bots*. O trabalho inicialmente discute a relação entre ataques DDoS na camada de aplicação e eventos *Flash Crowd* (seção 2), bem como o comportamento interativo nas solicitações HTTP(S) (seção 3). O método proposto e os resultados experimentais são apresentados nas seções 4 e 5, respectivamente. Os trabalhos relacionados são apresentados na seção 6 e as conclusões na seção 7.

2. Flash Crowds e Ataque DDoS na Camada de Aplicação

Tradicionalmente, ataques de negação de serviço exploram vulnerabilidades nos protocolos de rede visando o esgotamento de seus recursos. Diferentemente, ataques DDoS na camada de aplicação tem por objetivo esgotar os recursos do servidor (sockets, CPU, memória, banco de dados, largura de banda de I/O) [Ke et al. 2009][Xie e Yu 2009][Walfish et al. 2010]. A motivação para esta mudança de foco dos atacantes decorre da melhor eficácia dos mecanismos de segurança em redes e da crescente evolução da tecnologia, que permitiu crescimento substancial da largura de banda das conexões de internet (*throughput* das interconexões).

Na camada de aplicação o tráfego anormal pode ser classificado em quatro tipos [Saravanan et al. 2013]: (1) *insistência no pedido* - o atacante concentra o ataque em uma página inicial ou uma página momentaneamente muito acessada, gerando maior fluxo no tráfego que converge para um ou mais pontos; (2) *pedido recursivo* - o tráfego é direcionado para várias páginas web (*urls*), fazendo o endereço fonte dos acessos convergir para um grupo de pontos. Porém as metas de tráfego tornam-se dispersas; (3) *repetidas cargas de trabalho* - tráfego que resulta de um grupo de *bots* que envia repetidas solicitações, por exemplo, acesso a uma imagem grande ou operações que exijam grande carga no banco de dados; e (4) *Flash Crowd* - causa um surto de visitas inesperadas, geralmente em um anúncio de novo serviço ou download de software livre, os endereços fontes são dispersos, as solicitações são legítimas e as metas convergem para um ou dois pontos. O tráfego anormal resultante de comportamento *Flash Crowd* é similar ao de um ataque DDoS na camada de aplicação, tanto na maneira que os dados são transmitidos quanto na troca de mensagens de comunicação entre cliente e servidor. Ambos são volumosos, apresentam rajadas e são instáveis, causando interrupções no serviço [Kandula et al. 2005][Ke et al. 2009][Dhingra and Sachdeva 2014].

Diferenças entre fluxos *Flash Crowd* e ataques DDoS no nível de aplicação (que imitam *flash crowds*) aparecem na análise comportamental dos acessos benignos e malignos, como por exemplo, na distribuição da origem dos endereços IPs e aumento e diminuição da velocidade de tráfego. De maneira geral, o ataque tem a intenção de manter o tráfego intenso de dados com o servidor por um período maior, enquanto que um usuário legítimo realiza a solicitação e encerra ou pausa sua interatividade com o servidor. Observa-se assim que o sucesso na detecção de DDoS *Flash Crowd* pode ser função da interpretação de "intenção" assinada no tráfego e que pode permitir a distinção entre os fenômenos.

A diferença no tipo de tráfego também decorre da característica do *flash crowd*, no qual o aumento na quantidade de usuários e acessos se dá de forma brusca e geralmente em resposta a um evento especial num dado período, tal como na publicação de notícias de grande repercussão, lançamento de filmes, músicas, *softwares*, entre outros. No caso de acesso legítimo, no momento que o usuário recebe a informação desejada do servidor ele tende a prosseguir solicitando outra página ou finaliza seu ciclo no site. Porém em um ataque DDoS *Flash Crowd* o objetivo explícito é sobrecarregar o servidor com solicitações legítimas, mas com más intenções, para causar uma grande lentidão ou, preferencialmente, tornar o servidor inacessível para usuários benignos. Na prática, *botmasters* tendem a forçar seus *bots* a gerar muito mais solicitações web do que um usuário legítimo [Yu et al. 2012]. Em [Dhingra e Sachdeva 2014] observa-se que o tráfego de dados do usuário benigno em um *flash crowd* é variável e tende a formar uma onda em "zigue zague". Esta tendência é em resposta ao comportamento humano de solicitar uma página e após alguns instantes solicitar outra. Outra observação pertinente é que uma eventual lentidão do servidor em função do alto volume de solicitações pode fazer com que o usuário desista e após alguns instantes insista no pedido.

Observa-se, pelo exposto, que eventos *Flash Crowd* e DDoS na camada de aplicação, ou DDoS *Flash Crowd*, compartilham características similares, mas que a diferenciação pode ser realizada via análise comportamental do tráfego.

3. Comportamento Interativo de um Humano e de um Bot

Assumindo como hipótese que o comportamento do evento *Flash Crowd* e do ataque DDoS *Flash Crowd* pode ser analisado e classificado em função do padrão de interatividade em requisições web e do número de requisições, esta seção detalha os comportamentos básicos destes eventos.

A interatividade nas solicitações de um humano junto ao servidor web apresenta um comportamento que varia de acordo com cada ação tomada pelo usuário em seu computador. Inicialmente, o usuário solicita o acesso a um *website* pelo seu *browser*, o que gera a abertura de conexão entre os *hosts* cliente/servidor utilizando o protocolo TCP. Via protocolo HTTP ou HTTPS, neste texto representados por HTTP(S), após a conexão, é realizada a solicitação da *url* primária e, posteriormente, das secundárias que estão incorporadas no corpo do código. Como observado em [Oikonomou e Mirkovic 2009][Xu et al. 2012][Thapngam et al. 2011][Pan et al. 2014], durante esta fase de solicitações legítimas o comportamento pode ser definido como de *interação ativa*, ou seja, quando estão sendo enviadas solicitações HTTP(S) legítimas para o servidor. Dependendo do tamanho dos arquivos a serem carregados e da largura de banda do cliente/servidor, os dados começam a ser transmitidos logo após as solicitações e a transmissão pode perdurar por vários segundos até sua conclusão. Entretanto, este tempo de transmissão até o término do *download* não deve comprometer a observação do comportamento interativo do usuário, pois a troca de dados, mesmo que ocorra de maneira intercalada com as solicitações HTTP(S) do usuário, não é realizada via protocolo HTTP(S). Logo, a troca de dados não interfere na frequência de mensagens das solicitações web. Numa situação de normalidade, após o recebimento dos dados pelo *browser* é esperado que o usuário realize a apreciação das informações apresentadas e leve um tempo variável até a próxima solicitação. Em outras palavras, numa *interação ativa* é esperado que o usuário não realize solicitações em elevada frequência [Hwang et al. 2005][Xie and Yu 2009][Oikonomou and Mirkovic 2009][Thapngam et al. 2011]. Por outro lado, o comportamento de um

usuário maligno, realizando um ataque genérico DDoS *Flash Crowd* de acesso ao conteúdo do *website*, corresponde às solicitações praticamente interruptas.

Ao modelar a interatividade como uma variável aleatória discreta, com valor 0 indicando períodos sem interatividade e 1 indicando período com interatividade, é observado que o comportamento esperado do usuário humano apresenta tempo variável entre novas solicitações, dado a existência de momentos de apreciação das informações (sem interatividade). Por outro lado, o comportamento esperado de um atacante (representado por um *bot*) apresenta maior interatividade. A Figura 1 ilustra os comportamentos típicos e esperados de um humano e de um atacante na notação discreta. Note que a interatividade, mensurada de forma binária (com interatividade e sem interatividade), o número de solicitações (representados pelos traços horizontais) e o tempo entre os estados com interatividade (t), quando analisados por unidade de tempo descrevem um comportamento observável que também permite diferir um usuário humano de um robô de ataque (*bot*).

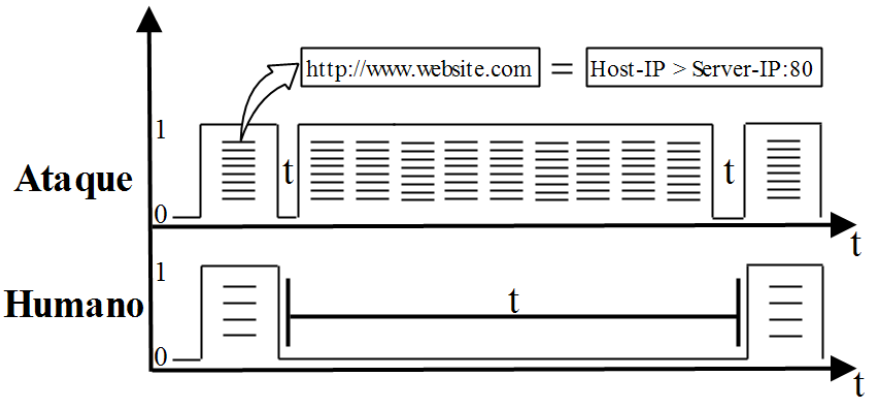


Figura 1. Padrão de interatividade de humano e de atacante.

4. Método para Detecção de Ataques DDoS *Flash Crowd*

Esta seção explora as características apresentadas na seção 3 e apresenta um novo método para detecção de ataques DDoS *Flash Crowd*. O método assume como hipótese básica que a análise do comportamento de um usuário web humano pode se distinguir de comportamentos de usuários web automatizados (*bots*), possibilitando a diferenciação entre *Flash Crowds* e ataques DDoS *Flash Crowd*.

Esta seção está organizada como segue. A seção 4.1 descreve as variáveis utilizadas para modelar o comportamento de humanos e de *bot*s no método proposto e a seção 4.2 apresenta o novo método proposto para diferenciar o comportamento humano do de um *bot* e distinguir entre um *Flash Crowd* e um DDoS *Flash Crowd*.

4.1 Modelagem Comportamental

Para distinguir um comportamento malicioso de um comportamento benigno três variáveis de interesse são modeladas: o número de solicitações HTTP(S) (*NumS*); a taxa de interatividade do usuário (*TxIn*); e o tempo entre estados de interatividade (*Teei*). A seguir, o comportamento de cada uma destas variáveis é discutido e modelado com vistas à diferenciação entre um ataque e um tráfego legítimo.

4.1.1 Número de Solicitações HTTP(S)

Conforme seção 3, e também observado em [Xu et al. 2012][Singh et al. 2015]**,** em ataques DDoS na camada de aplicação o número de solicitações HTTP(S) (*NumS*), aquelas com porta de destino 80 ou 443, tende ser maior para um *host* fonte do tipo *bot* do que para um *host* fonte tradicional (não *bot*), dado as intenções distintas de cada um. Com isso o número de solicitações de um *host* atacante (*NumSH_bot*) tende a ser maior que o de um *host* operado por humano (*NumSH_humano*) em um mesmo período Δ*t*, conforme já ilustrado na Figura 1. Desta forma, para um dado período Δ*t* assume-se o que segue.

$$NumSH\_bot > NumSH\_humano$$

A observação do número de solicitações dos *hosts* permite assim a análise parcial da distinção entre comportamento humano e de um *bot*. Esta análise pode ser realizada em função da intensidade de *NumS*. Deste modo, o método de detecção proposto realiza duas medidas:

(*i*) a média de solicitações do traço analisado no período Δ*t*, dado por $\overline{NumSG}$; e
(*ii*) a média de solicitações por *host* no mesmo período, dado por $\overline{NumSH}$.

O intervalo de tempo Δ*t* é denominado período de atividade observado (*Pat*). Desta forma, em termos do número de solicitações, em um *Pat* é possível avaliar o quão distante um *host* sob análise ($\overline{NumSH}$) encontra-se da média do traço ($\overline{NumSG}$). Salienta-se que para um traço livre de ataque, as duas médias tendem ser próximas.

4.1.2 Taxa de Interatividade

A taxa de interatividade (*TxIn*) de um dado usuário, ou seja, de sua interação ativa, pode ser obtida através da análise da ocorrência ou não de solicitações por período de tempo. Dado o período de atividade observado (*Pat*), correspondente a uma janela de observação de tamanho Δ*t*, discretizada em períodos Δ*t'*, onde Δ*t* >= 2.Δ*t'*, a taxa de interatividade corresponde a frequência de ocorrência de solicitações nos períodos de tempo Δ*t'* dentro período Δ*t*.

A verificação de interatividade de um dado usuário (*In*) é realizada observando em *Pat* se há ou não solicitações do *host* (*NumSH*) para o servidor a cada período Δ*t'*. O resultado é um vetor de interatividade do host (*VIn*) contendo valores 0s e 1s, onde 0 indica que não existiu solicitação/interação e 1 indica que existiu. O tamanho do vetor depende da granularidade desejada para o *Pat*, ou seja, da janela temporal de observação Δ*t* e da janela temporal considerada no computo da amostra Δ*t'*.

Desta forma, a taxa de interatividade de um usuário (*TxIn*) deriva da quantidade de períodos com interatividade em um *Pat*. De acordo com a seção 3, a interatividade do *bot* (*In_bot*) tenderá a ocupar todo o período *Pat* e ser maior que a média observada dos comportamentos humanos $\overline{TxIn\_humano}$, ou seja,

$$TxIn\_bot > \overline{TxIn\_humano}.$$

De maneira similar, a interatividade do humano tenderá a um comportamento de baixa interatividade no período observado, tendendo ao padrão médio dos traços observados em período sem ataques $\overline{TxIn\_humano} \cong \overline{TxIn\_traço}$. Salienta-se que em períodos de *Flash crowd* o comportamento humano não é alterado.

4.1.3 Tempo entre Estados Interativos

Entre dois estados interativos existe um intervalo de tempo de não interatividade do usuário. O Tempo entre Estados Interativos (*Teei*), correspondente ao intervalo de tempo de não interatividade, inicia no final de um estado interativo (última solicitação observada de uma interação) e finda na próxima solicitação (início de um estado interativo). Dado a característica de navegação humana apresentadas na seção 3, num traço legítimo o *Teei* médio para um dado host, ou $\overline{Teei}$, tende ser maior do que para *bots*. Como observado em [Xu et al. 2012][Dhingra and Sachdeva 2014][Singh et al. 2015], os períodos de não interatividade para *bots* tendem a zero segundos, e, de acordo com [Hwang 2004][Hwang et al. 2005], para humanos tendem a valores maiores do que 10 segundos. Logo tem-se que

$$\overline{Teeı\_humano} \gg \overline{Teeı\_bot}.$$

4.2 O Método de Diferenciação

Para que a distinção comportamental entre humanos e *bot*s possa ser avaliada, possibilitando diferenciar um *Flash Crowd* de um DDoS *Flash Crowd*, o método proposto, chamado DVM, explora uma nova técnica baseada na análise das diferenças aos valores médios (*DVM*).

Conforme apresentado na seção 4.1, três variáveis de interesse são modeladas (*NumS*, *TxIn* e *Teei*). Cada uma delas pode expressar o comportamento médio de um dado *host* ($\overline{NumSH}$, $\overline{TxInH}$ e $\overline{TeeıH}$) e, juntas, o comportamento médio do traço sob análise ($\overline{NumSG}$, $\overline{TxInG}$ e $\overline{TeeıG}$) . Para um dado período de atividade amostrado (*Pat*), o cálculo das diferenças aos valores médios tem o intuito de explicitar a distância comportamental de um host frente aos comportamentos esperados. À distância, expressa por host (*DVMH*) é função dos percentuais diferenciais relativos ao número de solicitações, taxa de interatividade e tempo entre estados interativos, tal como segue.

$$(DVMH)=\left[\left(\frac{\text{NumSH-NumSG}}{\text{NumSG}}.100\right)+\left(\frac{\text{TxInH-TxInG}}{\text{TxInG}}.100\right)+\left(\frac{\text{TeeiH-TeeiG}}{\text{TeeiG}}.100\right).(\text{-}1)\right]$$

O valor *DVMH* assume magnitude positiva e negativa, e quando zero indica equivalência comportamental do host com a do traço envolvendo todos os hosts. Quanto mais negativo, mais distante o host está de um comportamento maligno. Por outro lado, quando positivo, quanto maior sua magnitude maior é a chance de representar um comportamento malicioso. Na formulação, o cálculo percentual do $\overline{TeeıH}$ é subtraído pelo fato do seu valor desejado ser o maior possível. Quanto maior mais próximo das características humanas, conforme discutido na seção 4.1.3, diferentemente de $\overline{NumSH}$ e $\overline{TxInH}$ que quanto menor mais próximo do humano.

Para avaliar se os valores de *DMVH* indicam ataque é aplicado um limiar (*threshold*). O método considera um limiar adaptativo calculado com base no *threshold* universal por nível [Donoho 1995], tal como aplicado em [Dalmazo et al. 2009]. O limiar adaptativo é dado por

$$L = \sqrt{2.logn(\sigma^2)}$$

na respectiva *Pat*. A Figura 2 ilustra o método de diferenciação.

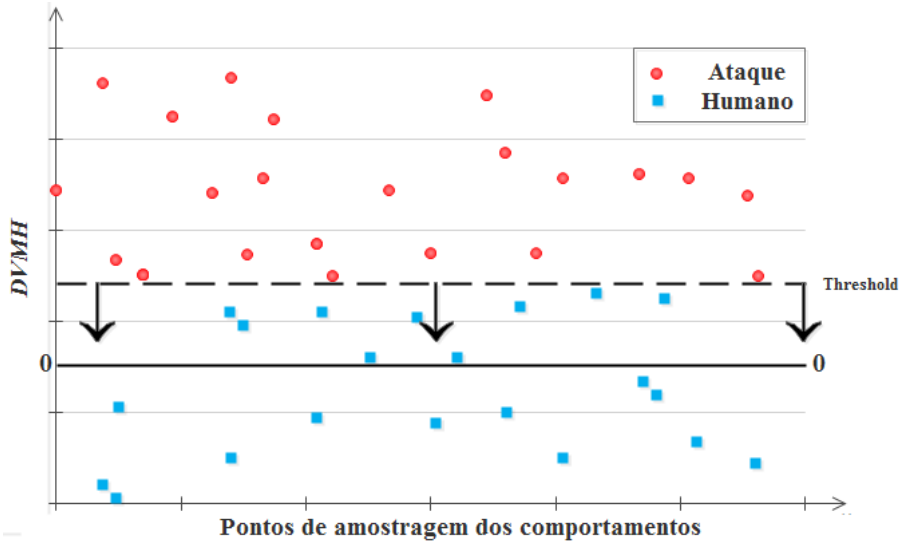


Figura 2. Diferenciação do comportamento humano e de ataque

De acordo com a Figura 2, o método de diferenciação do comportamento humano e de atacantes *bot* adota como parâmetro uma função de corte nas diferenças aos valores médios, que calculadas por *host* permite distinguir bots e identificar ataques DDoS mesmo em momentos de ampliação de tráfego *Flash Crowd*.

5. Resultados Experimentais

A obtenção de amostras reais de tráfego malicioso do tipo DDoS *Flash Crowd* é um desafio. Entretanto, a validação experimental pode também trabalhar com geração de ataques artificiais [Lucena and Moura 2010][Xie and Yu 2009] inseridos ao tráfego de fundo sem ataque. Neste trabalho, para tráfego de fundo são utilizados dados das bases de tráfego público CAIDA [CAIDA 2016] e WITS [Waikato 2016]. A ferramenta utilizada para geração de tráfego anômalo foi a DDoSIM v0.2 [DDoSIM 2016], que também foi utilizada nos trabalhos de [Ye and Zheng 2011][Bekeneva et al. 2015], e permite a simulação de diferentes rajadas de acessos HTTP válidos provenientes de distintos IPs. O tráfego de fundo WITS já foi utilizado em [Lu et al. 2007][Behal and Kumar 2016] e o CAIDA em [Hwang et al. 2005][Xie and Yu 2009][Babu et al. 2011][Righi and Nunes 2015][Bhuyan et al. 2015][David and Thomas 2015].

Inicialmente demonstra-se que o modelo proposto na seção 4 pode modelar o comportamento sem ataques (de humanos), medindo valores médios (*NumSG, TxInG, TeeiG*) de traços reais. Para tal, foram medidos valores médios para os tráfegos de fundo WITS ISPDSL II 2010 06/Jan 15:30h-15:40h e CAIDA 2015 19/Fev 13:00h-13:10h com *Pat* definido em 60s. Salienta-se que os tráfegos das bases são bastante distintos, ou seja, possuem diferença de *throughput*, número de pacotes, tipo de aplicação, dentre outros. Por exemplo, o número de *hosts* analisados nos trechos citados foi de 7.343 na base WITS e 1.989.553 na base CAIDA. Os valores médios obtidos (média para todos os *Pats*) nos dois traços foi: CAIDA apresentou 13,98 segundos e WITS 18,37 segundos para *TeeiG* e para *NumSG* e *TxInG* CAIDA resultou em 32,10 e 3,80 e WITS 33,71 e 4,55, respectivamente. A regularidade e proximidade dos resultados demonstra capacidade do método para modelar o comportamento humano, permitindo a discrimição para não humanos (*bots*). Além disto, este método de análise contribui para a formação do perfil

humano na navegabilidade web, encontrado anteriormente somente por [Jena et al. 2003], que afirma que o humano possui o tempo médio entre interações (*TeeiG*) de 15 segundos.

Para a obtenção dos traços com *Flash Crowd* foram utilizados traços das bases CAIDA e WITS. Os traços foram modificados utilizando como base o resultado do trabalho de [Pan et al. 2014], no qual, apresenta a diferença comportamental do *host* em momento de normalidade e eventos *Flash Crowd.* Os autores apontam que estes traços são 2,35 mais intensos do que sem *Flash Crowd* e que no evento sob análise somente 62,3% dos *hosts* mudam seu comportamento. Desta forma, para a elaboração dos traços com *Flash Crowd* foi realizado incremento de 2,35 em *NumSH* e *TxInH* e decremento em *TeeiH* em 62,3% *hosts*.

Os traços com DDoS *Flash Crowd* usaram o mesmo tráfego de fundo, sendo os ataques elaborados de forma sintética com a ferramenta DDoSIM. A geração do traço anômalo considerou *delay* de 4000 milisegundos entre pacotes *SYN* e 254 *hosts* atacantes, resultando em um comportamento médio de 2000 solicitações em um Δt de 300 segundos. Tal configuração está alinhada ao que foi utilizado no experimento de [Xu et al. 2012].

Para ilustrar que o DVMH é uma variável que permite distinguir *Flash Crowd* de DDoS *Flash Crowd*, plota-se na Figura 3 o DVMH para uma amostra de trecho (a) sem anomalias e com *Flash Crowd* e (b) sem anomalias e com DDoS *Flash Crowd*, onde metade da amostra é sem anomalia e a outra com anomalias. O traço usado é do projeto WITS denominado ISPDLS II 2010 do dia 06/Jan com 60 minutos. Os comportamentos anormais são indicados nos retângulos pontilhados. A análise comparativa da Figura 3(a) e 3(b) permite identificar visualmente o efeito dos cálculos do método proposto, demonstrando que a diferença aos valores médios dos *host*s (DVMH) apresenta variações significativas entre traços com *Flash Crowd* e com DDoS *Flash Crowd*. A diferença no número de cálculos decorre no maior número de *hosts* solicitantes em momentos de ataques DDoS *Flash Crowd*.

Para demonstrar a eficiência do método DVM, foi implementado o algoritmo de [Xu et al. 2012] que visa detectar ataques DDoS *Flash Crowd* realizando a contagem de solicitações HTTP. O método de Xu foi então aplicado aos mesmos traços, sendo calculado o valor de Correlação do Coeficiente de *Pearson* (PCC) a cada Δt de 5 minutos, conforme os autores sugerem, e definido um R de 280, que é o número de *hosts* com maior número de solicitações dentro do Δt, no qual, são utilizados para determinar o PCC do Δt.

Os resultados obtidos com os dados CAIDA e WITS modificados, para conter *flash crowd* de acordo com os indicadores de [Pan et al. 2014] (envolvendo 62,3% dos hosts) e para conter ataques DDoS *flawh crowd* sintéticos gerados pela ferramenta DDoSIM (254 ataques), são sintetizados na Tabela 1. Para tráfegos sem ataques, mas contendo comportamentos anômalos do tipo *flash crowd*, o método DVM, quando comparado com o método de Xu et al., reduziu significativamente a taxa de falsos positivos (de 22,05% para 14,01%). Redução ainda mais significativa foi observada para tráfegos com ataques DDoS (de 31,07% para 8,36¨%). O método também apresentou melhor acurácia, reduzindo significativamente o número de falsos positivos (de 23,90% para 1,97%). Salienta-se que embora o método ainda apresente taxa de falsos positivos relativamente alta (14%), claramente demonstra eficiência na diferenciação entre *Flash Crowd* e DDoS *Flash Crowd* e demonstrando-se uma importante ferramenta para detecção efetiva de ataques DDoS *Flash Crowds* (98,03% de efetividade).

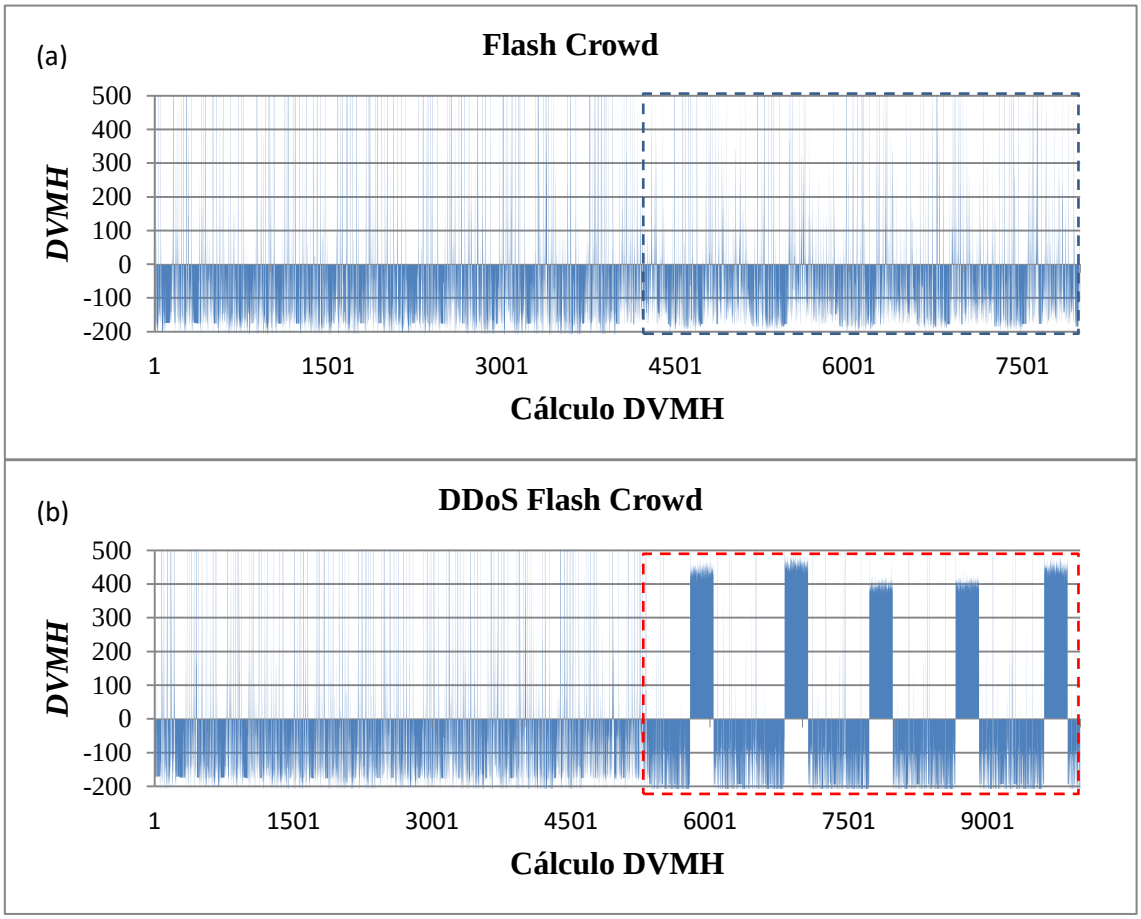


Figura 3. DVMH de tráfego (a) com Flash Crowd e (b) com DDoS Flash Crowd. Traço do projeto WITS denominado ISPDLS II 2010 do dia 06/Jan.

Tabela 1. Resultados da análise de eficiência do método DVM

Método	Base de dados CAIDA e WITS	Ataques	Detecção	Falso Positivo	Falso Negativo
DVM	Tráfegos com *Flash Crowd*	0	-	14,01%	-
Xu et al.		0	-	22,05%	-
DVM	Tráfegos com DDoS *Flash Crowd*	254	98,03%	8,36%	1,97%
Xu et al.		254	76,10%	31,07%	23,90%

Como observado em [Valcy et al. 2011], a técnica NAT (*Network Address Translator*) ainda é bastante utilizada, ou seja, vários usuários podem utilizar um mesmo endereço IP para se comunicar com o servidor. Embora o uso de NAT não tenha sido alvo dos experimentos, ressalta-se que eventual aumento no número de falsos positivos causado pelo uso de NAT pode ser facilmente contornado. Nos experimentos um usuário

foi identificado pelos atributos *host_fonte*, *host_destino* e *porta_destino* (80/443). A simples adoção de mais um parâmetro de identificação dos hosts de usuários, tal como a porta fonte (diferentes hosts atrás do NAT usam diferentes portas para retorno de informações do servidor), pode descriminar os hosts usando NAT.

6. Trabalhos Relacionados

Alguns métodos foram propostos para possibilitar a detecção de ataques DDoS *Flash Crowd*. [Xu et al. 2012] apresenta um algoritmo que utiliza o Coeficiente de Correlação de *Pearson* para analisar o grau de atividade dos usuários contabilizado pelo número de solicitações HTTP. Os autores assumem que após um início de ataque os *bots* não param de enviar solicitações com o propósito de negar o serviço, gerando alto volume de solicitações HTTP e que os humanos terão um comportamento de altos e baixos, oscilando ao longo da análise. Os autores assumem assim que a correlação dos *bots* deve ser mais alta que a dos humanos, permitindo a descriminação dos comportamentos. Entretanto, analisar somente a correlação entre os graus de atividades (número de requisições) pode, ao longo do tempo, acarretar em muitos falsos positivos, principalmente pelo fato do número de solicitações na requisição de uma página web variar muito. Cada requisição possuirá uma quantidade aleatória de *urls* incorporada no seu corpo (*body*), fazendo com que o comportamento do grau de atividade não seja influenciado somente pelo usuário, mas sim pela quantidade de solicitações de cada página, comprometendo a análise discriminatória.

O trabalho de Thapngam et al. (2011) assume que os humanos não possuem uma previsibilidade no comportamento, enquanto os *bots*, por realizarem as solicitações de forma automatizada a partir de um algoritmo, possuem características que se repetem em um período curto de tempo. Desta forma, os autores apresentam um método baseado na análise da repetição da taxa de transmissão de pacotes. A análise avalia a Correlação de *Pearson* entre a taxa de chegada de pacotes x tempo e a autocorrelação da taxa calculada. A limitação desta técnica é seu foco apenas na taxa de chegada de pacotes. Os autores omitiram a quantidade de falsos positivos.

Segundo [Yu et al. 2012] os fluxos de ataques são mais similares entre si do que os fluxos de acessos benignos. Desta forma, os autores apresentam uma técnica que consiste na discriminação utilizando o coeficiente de correlação de fluxos e o cálculo de similaridade entre os fluxos suspeitos. A descoberta que os fluxos em um ataque são bastante semelhantes é um grande avanço, porém, como também discutido no trabalho, o atacante tendo conhecimento da técnica pode customizar o ataque para imitar um acesso benigno, deixando a semelhança entre os fluxos muito equivalentes. Logo, a detecção utilizando a similaridade dos fluxos pode não ser suficiente para construção de um método eficiente.

Neste trabalho a distinção entre comportamento humano e de um *bot* explora uma análise conjunta de outras características, não só da quantidade de solicitações e similaridade entre fluxos. Diferencia-se dos demais por explorar a característica de interatividade e tempo entre estados interativos por método distinto da similaridade entre fluxos.

7. Conclusão

Para distinguir um ataque DDoS *Flash Crowd* de um *Flash Crowd* legítimo este artigo apresenta um método que modela o comportamento de humanos e *bot*s por três características (número de solicitações, tempo entre estados ativos e taxa de interatividade) e realiza uma análise de suas diferenças aos valores médios. Os experimentos demonstram que o método gera resultados satisfatórios na diferenciação entre o comportamento de humanos e de atacantes automatizados (*bots*). O resultado é um novo método para elaborar modelos comportamentais para detecção de ataques DDoS *Flash Crowd*.

Por detectar comportamentos anômalos proveniente de *bot*s, o método pode ser usado também para fornecimento de uma lista de *hosts potencialmente maliciosos* ou para apoiar técnicas de Web QoS [Fang et al. 2015], que permitem priorizar o tráfego para *hosts* que não estejam incluídos em listas de suspeitos. O método também pode ser aplicado em conjunto com métodos de redirecionamento para uma página de *CAPTCHA* [Singh and De 2015], para verificação que não se trata de um *bot*. Diante disto, além da diferenciação entre *Flash Crowd* e DDoS *Flash Crowd*, o método proposto pode ser usado para compor soluções que visam manter a operabilidade de sistemas web minimizando a exploração por *bots*. Trabalhos futuros irão avaliar o desempenho do método em termos de consumo de recursos e tempo de resposta, bem como variantes que permitam contornar falsos positivos decorrentes de um grande número de usuários legítimos operando por meio de NAT/Proxy.

Referências

Babu, G. P.; Jayavani, V.; Mohan Rao, C. P. V. N. J. (2011). Anomaly Detection On User Browsing Behaviors Using Hidden Semi-Markov Model. *Int. Journal of Computer Science and Information Technologies*, v. 2, n. 3, p. 1197-1201

Behal, S. and Kumar, K. (2016). Trends in Validation of DDoS Research. *In: Int. Conf. On Computational Modeling and Security*. Procedia Computer Science n. 85, Elsevier, p. 7-15. doi: 10.1016/j.procs.2016.05.170

Bekeneva, Y., Shipilov, N., Borisenko, K. and Shorov, A. (2015). Simulation of DDoS-attacks and protection mechanisms against them. In: *IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW)*.

Bhuyan, M. H.; Kalwar, A.; Goswami, A.;Bhattacharyya, D. K.; Kalita, J. K. (2015) Low-Rate and High-Rate Distributed DoS Attack Detection Using Partial Rank Correlation. In: *Int. Conf. On Communication Systems and Network Technologies*, IEEE, doi:10.1109/CSNT.2015.24

CAIDA Anonymized Internet Traces (2015). http://www.caida.org/data/overview/, [acessado em Junho de 2016].

Dalmazo, B. L.; Perlin, T.; Nunes, R. C.; Kozakevicius, A. J. (2009) Filtros de Alarmes de Anomalias através de Wavelets. In: *Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais* – SBSeg, Campinas/SP. Porto Alegre: Sociedade Brasileira de Computação, 2009. v. 1. p. 85-98.

Dantas, R. S. A. (2013). Diferenciação de Ataques DDoS e Flash Crowds. Rio de Janeiro: Instituto Militar de Engenharia. 75p. (Dissertação).

David, D. and Thomas, C. (2015). DDoS Attack Detection using Fast Entropy Approach on Flow-Based Network Traffic. *In: Int. Conf. On Big Data and Cloud Computing*. Procedia Computer Science n. 50, Elsevier, p. 30-36. doi: 10.1016/j.procs.2015.04.007

DDoSIM (2016). https://stormsecurity.wordpress.com/2009/03/03/application-layer-ddos-simulator/, [acessado em Junho de 2016].

Dhingra, A. and Sachdeva, M. (2014). Recent Flash Events : A Study. *International Conference on Communication, Computing & Systems (ICCCS–2014)*, p. 94–99.

Donoho, D. L. (1995). De-noising by soft-thresholding. *IEEE Transactions on Information Theory*, *41*, 613–627.

Fang, W.; Sun, J.; Wu, X.; Palade, V. (2015). Adaptive Web QoS controller based on online system identification using quantum-behaved particle swarm optimization. *Soft Computing*. v.19, n. 6, p. 1715-1725. doi:10.1007/s00500-014-1359-9

Feily, M., Shahrestani, A. and Ramadass, S. (2009). A survey of botnet and botnet detection. *Proceedings - 2009 3rd International Conference on Emerging Security Information, Systems and Technologies, SECURARE 2009*, p. 268–273.

Hwang, F. (2004). Análise dos Efeitos Gerados pelo Comportamento das Aplicações e pelo Perfil das Redes na Característica Auto-Similar do Tráfego Internet. São Paulo: Universidade Estadual de Campinas. 167p. (Dissertação)

Hwang, F., Bianchi, G. R. and Lee Luan Ling (2005). Impacto Gerado pelo Comportamento das Aplicações (Web, FTP e E-mail) e pelo Perfil das Redes na Característica Auto-Similar. *IEEE Latin America Transactions*, v.3, n.4, p.356–361.

Jena, A. K., Popescu, A. and Nilsson, A. A. (2003). Modeling and evaluation of internet applications. *2014 IEEE International Conference on Internet of Things(iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*. IEEE. p. 531–540.

Kandula, S., Katabi, D., Jacob, M. and Berger, A. (2005). Botz-4-sale: Surviving organized DDoS attacks that mimic flash crowds. *2nd Symposium on Networked Systems Design and Implementation (NSDI)*, p.287–300.

Ke, L., Wanlei, Z., Ping, L., Jing, H. and Jianwen, L. (2009). Distinguishing DDoS Attacks from Flash Crowds Using Probability Metrics. In: *Proc. of the International Conference on Network and System Security* - NSS '09, p.09-17.

Lu, K.; Wu, D.; Fan, J.; Todorovic, S.; Nucci, A. (2007). Robust and efficient detection of DDoS attacks for large-scale internet. *In: Computer Networks*, n. 51, Elsevier, p. 5036-5056. doi:10.1016/j.comnet.2007.08.008

Lucena, S. C. and Moura, A. S. (2010). Estimativa de Holt-Winters para Detecção de Ataques em Redes WAN. *X Simpósio Brasileiro Em Segurança Da Informação E De Sistemas Computacionais*, v. d, p. 157–170.

Oikonomou, G. and Mirkovic, J. (2009). Modeling Human Behavior for Defense against Flash-Crowd Attacks. *ICC'09 Proceedings of the 2009 IEEE international conference on Communications*, n. 0430228, p. 625–630.

Pan, J., Hu, H. and Liu, Y. (2014). Human behavior during Flash Crowd in web surfing. *Physica A: Statistical Mechanics and its Applications*, v. 413, p. 212–219.

Prasad, K. M., Reddy, A. R. M. and Rao, K. V (2013). Discriminating DDoS Attack traffic from Flash Crowds on Internet Threat Monitors (ITM) Using Entropy variations. *African Journal of Computing & ICT*, v. 6, n. 2, p. 53–62.

Righi, M. A., Nunes, R. C. (2015). Detecção de DDoS Através da Análise da Recorrência Baseada na Extração de Características Dinâmicas. *XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, p. 327–330.

Saravanan, R., Shanmuganathan, S. and Palanichamy, Y. (2013). Behavior-based detection of application layer distributed denial of service attacks. *Turkish Journal of Electrical Engineering & Computer Sciences*, p. 1–14.

Singh, B., Kumar, K. and Bhandari, A. (2015). Simulation Study of Application Layer DDoS Attack. *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, p. 893–898.

Thapngam, T., Yu, S., Zhou, W. and Beliakov, G. (2011). Discriminating DDoS Attack Traffic from Flash Crowd through Packet Arrival Patterns. *Computer Communications Workshops* (*INFOCOM*), 2011 *IEEE Conference*, p. 952–957.

Valcy, I., Barreto, L. P., Bezzera, J. (2011). Tratamento Automatizado de Incidentes de Segurança da Informação em Redes de Campus. *XI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, p. 29–42.

Waikato (2016). Waikato Internet Traffic Storage - WITS. http://wand.net.nz/wits/, [acessado em Junho de 2016].

Walfish, M.; Vutukuru, M.; Balakrishnan, H.; Karger, D.; Shenker, S. (2010) DDoS defense by offence. ACM Trans. on Computer System, v. 28, n. 1, Article 3 (August 2010), 54p.

Xie, Y. and Yu, S. Z. (2009). Monitoring the application-layer DDoS sttacks for popular websites. *IEEE/ACM Trans on Networking*, Feb., v. 17, n. 1, p. 15–25.

Xu, C., Du, C. and Kong, X. (2012). An Application Layer DDoS Real-Time Detection Method in Flash Crowd. *International Association of Computer Science & Information Technology (IACSIT)*, v. 30, p. 68–73.

Ye, C. and Zheng, K. (2011). Detection of application layer distributed denial of service. *2011 International Conference on Computer Science and Network Technology Detection*, IEEE, p. 310–314.

Yu, S., Zhou, W. and Member, S. (2012). Discriminating DDoS Attacks from Flash Crowds Using Flow Correlation Coefficient. *Parallel and Distributed Systems, IEEE Transactions*, v. 23, n. 6, p. 1073–1080.

Speeding up Elliptic Curve Cryptography on the P-384 Curve

Armando Faz-Hernández, Julio López*

Institute of Computing, University of Campinas.
1251 Albert Einstein, Cidade Universitária. Campinas, Brazil.

{armfazh,jlopez}@ic.unicamp.br

Abstract. *The P-384 is one of the standardized elliptic curves by ANSI and NIST. This curve provides a 192-bit security level and is used in the computation of digital signatures and key-agreement protocols. Although several publicly-available cryptographic libraries support the P-384 curve, they have a poor performance. In this work, we present software techniques for accelerating cryptographic operations using the P-384 curve; first, we use the latest vector instructions of Intel processors to implement the prime field arithmetic; second, we devise a parallel scheduling of the complete formulas for point addition law. As a result, on Skylake micro-architecture, our software implementation is 15% and 40% faster than the OpenSSL library for computing ECDSA signatures and the ECDH protocol, respectively.*

Resumo. *A P-384 é uma das curvas elípticas padronizadas pelo ANSI e o NIST. Ela fornece um nível de segurança de 192 bits e é usada tanto na computação de assinaturas digitais como nos protocolos de acordo de chaves. Embora várias bibliotecas criptográficas disponíveis publicamente suportam a P-384, elas possuem um baixo desempenho. Neste trabalho, apresentamos técnicas de implementação em* software *para acelerar operações criptográficas usando a curva P-384; primeiro, usamos as mais novas instruções vetoriais dos processadores Intel para implementar a aritmética de corpo primo; depois, propomos um escalonamento paralelo das fórmulas completas para calcular a lei de adição de pontos. Como resultado, na microarquitetura* Skylake*, a nossa implementação em* software *é 15% e 40% mais rápida do que a biblioteca OpenSSL para calcular assinaturas ECDSA e o protocolo ECDH, respectivamente.*

1. Introduction

The elliptic curve cryptography (ECC) is well-known for providing secure algorithms that are faster and use shorter key sizes in comparison with practical cryptosystems like the RSA [Rivest et al. 1978]. In July 1999, the National Institute of Standards and Technology (NIST) recommended a set of elliptic curves and parameters to be used by federal government applications [NIST 1999]. This recommendation considers a set of 15 elliptic curves that covers five security levels: 80, 112, 128, 192, and 256 bits. Five of these curves are defined over prime fields and are known as P-192, P-224, P-256, P-384, and P-521. Afterward, the American National Standards Institute (ANSI) standardized the use of NIST's curves [ANSI 1999]; and this had an impact on the widespread use of ECC for establishing secure communications. For instance, a scan performed in 2014 over around

*The authors gratefully acknowledge the support from the Intel Labs University Research Office.

Table 1. Set of algorithms classified according to the security levels defined by Suite B. Entries marked with ⋆ denote algorithms considered for legacy support.

Primitive	Algorithm	Suite B Security Level	
		SECRET	TOP SECRET
Data Encryption	AES	AES-128	AES-256
Hash Function	SHA2	SHA-256	SHA-384
Key Agreement	ECDH	P-256	P-384
	DH⋆	2,048-bit modulus	3,072-bit modulus
Digital Signature	ECDSA	P-256	P-384
	DSA⋆	2,048-bit modulus	3,072-bit modulus
	RSA⋆	2,048-bit modulus	3,072-bit modulus

30.2 millions of Internet hosts revealed that 7.2% of them support a cipher suite based on elliptic curves [Bos et al. 2014]. The authors also showed that 98% of these hosts support the P-256 curve, 80% support the P-384 curve, and only 17% support the P-521 curve. The use and application of ECC are nowadays a suitable alternative for implementing public-key cryptography systems.

The Committee on National Security Systems (CNSS) in the United States selected a set of standardized security protocols and cryptographic algorithms to define a cipher suite called *Suite B* [CNSS 2012]. The Suite B defines two security levels: the SECRET level and the TOP SECRET level, detailed in Table 1. The most remarkable of Suite B is the choice of elliptic curves as the preferred cryptosystem over other public-key algorithms such as RSA and DSA. Recently in 2015, the CNSS announced an update concerning the usage of the Suite B [CNSS 2015]. The announcement was motivated by the effect of quantum computing on information assurance; in particular, the weakness of ECC against an attack by a quantum computer [Proos and Zalka 2003]. At this time, the latest quantum computers do not represent a threat to the ECC instances approved in standards. Thus, before making a transition to quantum-resistant algorithms, the CNSS restricted the use of the Suite B to work only at the TOP SECRET level.

In some computational systems, like those dependent on a cryptographic hardware infrastructure, upgrading to the TOP SECRET level could incur into a serious investment for updating the whole infrastructure in cases where the higher security level is not supported. However, in systems using a cryptographic software infrastructure, this upgrade can be performed by setting the appropriate parameters in the software libraries. Unfortunately, a loss of performance can occur, since some cryptographic libraries have inefficient implementations for the higher security levels, like in the case of the P-384 curve. Consequently, the transition to the TOP SECRET level enhances security but downgrades on performance.

Regarding software implementations, the increasing support for parallel computing in the latest micro-architectures favors algorithms that can be partitioned into a series of independent tasks. Thus, it is necessary to provide new algorithms, or adapt the existent ones, to take advantage of modern processors. In that sense, it is crucial to devise formulations that increase the parallelism degree of algorithms that support ECC.

In this work, we focus on a high-performance software implementation of the P-384 curve using advanced software techniques and efficient algorithms for the secure execution of cryptographic operations. The following is the synopsis of our contributions. In a lower level, we optimize the execution of prime field arithmetic by using vector instruction sets. At a higher level, we devise a scheduling of the field operations employed in the *complete* formulas for calculating point additions; this scheduling allowed that point additions be executed in parallel with low communication between the processing units. To determine the impact of the techniques proposed, we apply them to the implementation of the Elliptic Curve Diffie-Hellman protocol (ECDH) [ANSI 2001] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [Vanstone et al. 1992]. Finally, we compare the performance of our implementation against some publicly-available cryptographic libraries.

Now, we detail some software optimizations targeting the NIST's curves. Käsper presented a fast implementation of the P-224 curve using a redundant representation for implementing the prime field arithmetic [Käsper 2012]. Gueron and Krasnov showed optimizations for the P-256 curve by computing the field multiplication efficiently using Montgomery-friendly primes [Gueron and Krasnov 2014]. Granger and Scott described a novel technique for accelerating multiplications in the field $\mathbb{F}_{2^{521}-1}$, which they used for the implementation of the P-521 curve; this technique can also be extended to Crandall's numbers [Granger and Scott 2015]. Unfortunately, from these prime field arithmetic optimizations, only the redundant representation applies to the prime modulus used by the P-384 curve.

2. Elliptic Curve Cryptography

An elliptic curve in short Weierstrass form over $\mathbb{F}_p$ is defined by $E_{a,b}(\mathbb{F}_p)\colon y^2 = x^3 + ax + b$, where $a, b \in \mathbb{F}_p$ and $-16(4a^3 + 27b^2) \neq 0$. The set of points belonging to this curve forms a commutative group E of cardinality ℓ where $\mathcal{O}$ is the identity element. The group law is denoted by $+$; thus, for any $P, Q \in E$, the operation $P + Q$ is called as *point addition*; in the case that $P = Q$, then the operation $2P = P + P$ is called as *point doubling*. The inverse of a point $P = (x, y)$ is calculated as $-P = (x, -y)$.

One of the most critical operations in ECC is the *point multiplication*. Given a point $P \in E$ and an integer k, the point multiplication is defined as the scalar product $kP = P + P + \cdots + P$, i.e. the point obtained after P was added to itself k times. The most efficient algorithms to calculate point multiplication have a computational complexity of $O(\log k)$; refer to [Hankerson et al. 2003] for a complete reference about point multiplication algorithms.

Around 1986, Miller [Miller 1986] and Koblitz [Koblitz 1987] suggested the use of elliptic curves for constructing public-key cryptosystems under the assumption that the *elliptic curve discrete logarithm problem* (ECDLP) is computationally intractable. This problem is stated as follows: given a generator point G of E and a point $P \in E$, the ECDLP consists on finding a number $k \in \mathbb{Z}_\ell$ such that $P = kG$. The best-known algorithm to solve ECDLP has a computational complexity of $O(\sqrt{\ell})$ [Pollard 1975]. Hence, key sizes on ECC are shorter than on RSA for attaining the same security level.

The elliptic curve Diffie-Hellman protocol (ECDH) is used to establish a shared secret between two entities securely. Assume that Alice and Bob want to agree on a

Domain parameter generation

Given: m, the security parameter.
Return: $D = \{p, E, G, \ell, h\}$, where p is a prime number such that $p \approx 2^{2m}$, E is the group induced by $E_{a,b}(\mathbb{F}_p)$, $G \in E$ is a generator point of order ℓ, and h is a hash function producing $2m$ bits.

Key Generation

Given: The domain parameters D.

1. Choose $\mathbf{sk} \xleftarrow{\$} \mathbb{Z}_\ell^*$.
2. Calculate $\mathbf{pk} \leftarrow \mathbf{sk}G$.

Return: $(\mathbf{sk}, \mathbf{pk})$.

Signature generation

Given: The domain parameters D, a message M and a secret key $\mathbf{sk}$.

1. Choose $k \xleftarrow{\$} \mathbb{Z}_\ell^*$.
2. Calculate $(R_x, R_y) \leftarrow kG$.
3. Calculate $r \leftarrow R_x \bmod \ell$.
4. If $r = 0$ then go to Step 1.
5. Set $s \leftarrow k^{-1}(h(M) + \mathbf{sk} \times r) \bmod \ell$.
6. If $s = 0$ then go to Step 1.

Return: (r, s).

Verification

Given: The domain parameters D, a message M, a signature (r, s) and a public key $\mathbf{pk}$.

1. Set $u_1 \leftarrow s^{-1} \times h(M) \bmod \ell$.
2. Set $u_2 \leftarrow s^{-1} \times r \bmod \ell$.
3. Calculate $(x, y) \leftarrow u_1 G + u_2 \mathbf{pk}$.
4. If $x \equiv r \bmod \ell$, set $v \leftarrow$ Accept; otherwise, set $v \leftarrow$ Reject.

Return: v.

Figure 1. Set of algorithms for the ECDSA scheme.

shared secret S. Then, both Alice and Bob create a pair of keys, $(\mathrm{s}_i, \mathrm{P}_i)$, where the secret key is randomly chosen as $\mathrm{s}_i \xleftarrow{\$} \mathbb{Z}_\ell^*$, and the public key is calculated as $\mathrm{P}_i \leftarrow \mathrm{s}_i G$ for $i \in \{A, B\}$. After that, they communicate their public keys to each other. To calculate the shared secret S, both multiply their private key by the public key received; thus, $S = \mathrm{s}_A \mathrm{P}_B = \mathrm{s}_B \mathrm{P}_A$. The main application of this protocol is for generating secret keys for symmetric data encryption. The ECDH protocol was standardized by ANSI [ANSI 2001] and by NIST [Barker et al. 2007].

The elliptic curve digital signature algorithm (ECDSA) is an adaptation of the digital signature algorithm (DSA) [NIST 2000] that replaced the use of the group $\mathbb{Z}_q$ by the elliptic curve group E. Figure 1 shows the set of algorithms of the ECDSA. The ANSI standardized the ECDSA [ANSI 1999]; this had a strong influence on other agencies, such as IEEE [IEEE 2000] and SECG [Brown 2009] that also approved its use.

3. Prime Field Arithmetic

3.1. Targeting the Machine Instruction Set

The latest processors strive to boost the performance of applications by improving on both the instruction-level parallelism (ILP) and the data-level parallelism (DLP). Widening the execution engine of a processor enhances the ILP; this means that by increasing the number of functional units will allow that several units execute instructions. For increasing DLP, some micro-architectures have included a vector unit able to execute one instruction over a set of words; this unit operates in concordance with the parallel paradigm known as Single Instruction Multiple Data (SIMD). We focus on the use of vector instructions to increase the performance of field arithmetic operations.

In 1999, the MMX was the first vector instruction set integrated into the Intel processors; the MMX supported operations over vector registers of 64 bits. Since then, the vector unit evolved into a more complex unit supporting both 128- and 256-bit vector instruction sets; on Intel processors these instruction sets are known as SSE, AVX and AVX2 [Intel Corporation 2011]. The AVX2 unit has 16 vector registers of 256 bits and also provides instructions for computing operations over words of 64 bits. The first micro-architecture supporting AVX2 was codenamed as *Haswell* and was released in 2014. Nowadays, both the *Broadwell* and the *Skylake* micro-architectures also have support for 256-bit vector instructions. All these micro-architectures contain two functional units for integer arithmetic instructions, three units for logic instructions, one unit for shift and multiplication instructions (two units in the case of Skylake) and one unit for byte permutations. A complete list of AVX2 instructions is available at [Intel Corporation 2011].

The most critical instruction in the implementation of prime field arithmetic is the integer multiplier. In AVX2, the `VPMULUDQ` instruction computes four simultaneous multiplications of 32-bit words producing four 64-bit products. This instruction takes five clock cycles; meanwhile, the 64-bit native multiplier, provided by the instructions `MULQ` and `MULX`, takes three clock cycles. This fact presents a trade-off between parallel computation and latency.

3.2. Field Element Representation Suitable for Vector Instructions

The prime modulus used by P-384 is $p_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$; this number belongs to the Generalized Mersenne numbers [Solinas 1999] which were designed to perform the modular reduction efficiently.

The implementation of operations in $\mathbb{F}_{p_{384}}$ requires operating over integers of 384 bits. Usually, field elements are split in digits, having a size equal to word size of the machine. One downside of this method appears on the implementation of modular additions, which propagate carry bits from the first to the last digit; this propagation introduces a dependency chain on computations. It has been observed that large dependency chains reduce the throughput of a calculation significantly. As a corollary, it follows that reducing such dependencies is crucial to improve the performance.

In that sense, we express field elements using a *redundant* representation with the aim of reducing carry dependencies [Käsper 2012]. In this representation, a field element is split into digits of smaller size that the word size of the instruction set; for example, an element of $\mathbb{F}_{p_{384}}$ can be represented by 14 digits of 28 bits using a 32-bit instruction set or by seven digits of 55 bits using a 64-bit instruction set. This representation guarantees that the addition of two elements will not overflow the registers, in that case, the carry-bit propagation can be postponed. Moreover, a determined number of additions can be processed without any carry propagation, which enables a parallel processing of digits and leading to an immediate application of vector instructions.

We represent a field element using 14 digits of 28 bits. Operating with digit size less than 32 is justified due to the `VPMULUDQ` instruction multiplies words of 32 bits producing 64-bit products; these products can be operated using 64-bit integer arithmetic, which is supported by AVX2. We considered to use 13 digits of 30 bits; however, digits will not have enough room to store carry bits. The same argument applies to digits of 29 bits. In the other hand, using less than 28 bits per digit increases the number of digits and

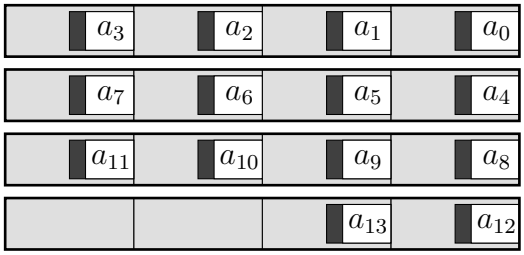


Figure 2. We store the digits a_i into four 256-bit vector registers; each vector register is composed of four 64-bit words. For computing additions, digits will have enough room (dark shading space) for storing carry bits.

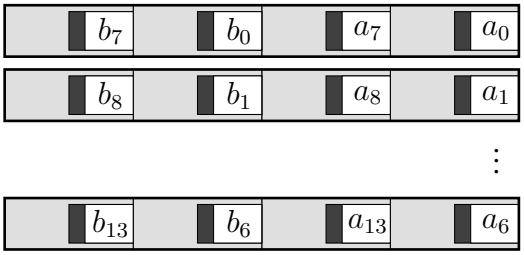


Figure 3. We store the digits of A and B into seven 256-bit vector registers. This arrangement is intended to reduce the use of permutation instructions since they move words across the 128-bit boundary increasing the execution latency.

slows field operations.

An element $a \in \mathbb{F}_{p_{384}}$ is then split into 14 digits a_i such that the following relation holds $a \equiv \sum_{i=0}^{13} 2^{28i} a_i \bmod p_{384}$. We denote by A the ordered sequence of digits $(a_0, \ldots, a_{13})$, for which a digit d in the n-th position of the sequence represents the number $2^{28n}d$. Figure 2 shows how we store one sequence of digits into four 256-bit registers.

3.3. Implementation of Arithmetic Operations

The addition $c = a+b \in \mathbb{F}_{p_{384}}$ is computed entirely in parallel; we proceed by performing $c_i = a_i + b_i$ for all $i \in [0, 14)$; using the representation shown in Figure 2 there are required only four vector addition instructions (`VPADDQ`). For the case of the subtraction $d = a - b \in \mathbb{F}_{p_{384}}$, we restrict digits to be always positive; hence, we compute $d_i = a_i - b_i + 2p_i$ for all $i \in [0, 14)$, where p_i is the redundant representation of p_{384}.

Now, we show how to compute the multiplication $a \times b \in \mathbb{F}_{p_{384}}$. Our technique is different from other implementations that process first the integer multiplication and then the modular reduction; instead, we perform modular reduction during the processing of integer multiplication. We express $c = a \times b$ in terms of an auxiliary function π as follows $c = \sum_{i=0}^{13} \pi^i(a) b_i$; where π is defined as $\pi(x) = 2^{28}x \bmod p_{384}$. Algorithm 1 shows the processing of modular multiplication using π. Lines 1 and 4 of Algorithm 1 calculate component-wise additions and multiplications over sequences of 14 digits; therefore, we can schedule 14 operations without dependency between them. This fact is crucial for improving performance; particularly, the processor can issue `VPMULUDQ` instructions into the pipeline every clock cycle since products do not present any dependency that could stall the pipeline.

The π function, used in line 3 of Algorithm 1, is computed by Algorithm 2 as follows. Given an input sequence A, every digit of A must be multiplied by 2^{28}, this can be done by changing a digit from the n-th position to the $(n+1)$-th position in the output sequence (line 1 of Algorithm 2). Now, notice that the a'_{14} represents the number $L = 2^{392}a_{13}$, which is congruent to $r = (2^{136} + 2^{104} - 2^{40} + 2^8)a_{13} \bmod p_{384}$. Thus, we add r to the digits of the output sequence; this addition must be processed ensuring that the digits do not be greater than 2^{32} since $\pi(A)$ will be the input of another round of

Algorithm 1 Modular multiplication in terms of the π function.

Input: $A = (a_0, \ldots, a_{13})$ and $B = (b_0, \ldots, b_{13})$, be sequences of the digits of a and b, respectively.

Output: $C = (c_0, \ldots, c_{13})$, be a sequence of digits such that $a \times b \equiv \sum_{i=0}^{13} 2^{28i} c_i \bmod p_{384}$.

1: $C \leftarrow A \cdot b_0$
2: **for** $i \leftarrow 1$ **to** 13 **do**
3: $\quad A \leftarrow \pi(A)$ *(Alg. 2.)*
4: $\quad C \leftarrow C + A \cdot b_i$
5: **end for**
6: **return** C

Algorithm 2 The π function.

Input: $A = (a_0, \ldots, a_{13})$, be a sequence of the digits of a.

Output: $(a'_0, \ldots, a'_{13})$, be a sequence of digits such that $2^{28}a \equiv \sum_{i=0}^{13} 2^{28i} a'_i \bmod p_{384}$.

1: $a'_{i+1} \leftarrow a_i$ for all $i \in [0, 14)$
2: $L \leftarrow a'_{14}$
3: $a'_0 \leftarrow 2^8(L \bmod 2^{20})$
4: $a'_1 \leftarrow a'_1 + \lfloor L/2^{20} \rfloor - 2^{12}(L \bmod 2^{16})$
5: $a'_2 \leftarrow a'_2 - \lfloor L/2^{16} \rfloor$
6: $a'_3 \leftarrow a'_3 + 2^{20}(L \bmod 2^8)$
7: $a'_4 \leftarrow a'_4 + \lfloor L/2^8 \rfloor + 2^{24}(L \bmod 2^4)$
8: $a'_5 \leftarrow a'_5 + \lfloor L/2^4 \rfloor$
9: **return** $(a'_0, \ldots, a'_{13})$

multiplications. We guarantee this property by adding the correspondent portions of bits of a_{13} into the digits of the output sequence as it is described in lines 2-8 of Algorithm 2. The operations of Algorithm 2 are implemented using only logic and shift instructions; moreover, whenever the shift displacement is multiple of 8, we use byte permutations instead of shifts since permutations are executed by another functional unit. Therefore, by taking advantage of the out-of-order execution, instructions for π are executed while the long-latency multiplication instructions are processed increasing the ILP of this workload.

4. Elliptic Curve Arithmetic

The elliptic curve addition law is calculated using operations in the prime field such as additions, multiplications, and squares that we denote by **A**, **M**, and **S**, respectively. Adding points in affine coordinates implies the use of field inversions, which in general are computationally expensive.

An alternative to avoid inversions consists on representing points using Jacobian coordinates, i.e. a point $P = (x, y)$ is represented by (X, Y, Z) such that $x = X/Z^2$ and $y = Y/Z^3$. This coordinate system presents the most efficient formulas to compute the addition law for curves in the short Weierstrass form. Specifically, the NIST's curves fix $a = -3$. Thus, the point addition requires 12**M**, 4**S**, and 7**A**. Meanwhile, the point doubling uses 4**M**, 4**S**, and 8**A** [Hankerson et al. 2003]. The formulas using Jacobian coordinates are not *complete*; this means that for some particular cases, formulas compute addition law incorrectly. Implementations using incomplete formulas must verify the presence of such special cases and handle them in a proper way. Otherwise, some vulnerabilities can be exploited [Izu and Takagi 2002]. For that reason, the use of complete formulas is recommended.

Renes et al. [Renes et al. 2016] introduced an optimization on the evaluation of complete formulas presented by Bosma and Lenstra [Bosma and Lenstra 1995]. This approach represents points in projective coordinates; i.e. a point $P = (x, y)$ is represented by (X, Y, Z) such that $x = X/Z$ and $y = Y/Z$. For the case of $a = -3$, the point ad-

Algorithm 3 Parallel scheduling of point addition for $a = -3$ using complete formulas.

Input: $P = (X_1, Y_1, Z_1)$, $Q = (X_2, Y_2, Z_2)$, and the coefficient b from $E_{-3,b}(\mathbb{F}_p)$.
Output: $P + Q = (X_3, Y_3, Z_3)$.

	LEFT	RIGHT		LEFT	RIGHT
1:	$l_0 \leftarrow X_1 + Y_1$	$r_0 \leftarrow X_2 + Y_2$	12:	$l_6 \leftarrow l_5 - r_0$	$r_6 \leftarrow r_5$
2:	$l_1 \leftarrow X_1 + Z_1$	$r_1 \leftarrow Y_1 + Z_1$	13:	$l_7 \leftarrow 3l_6$	$r_7 \leftarrow 3r_6$
3:	$l_2 \leftarrow X_2 + Z_2$	$r_2 \leftarrow Y_2 + Z_2$	14:	$l_8 \leftarrow l_0 - r_0$	$r_8 \leftarrow q_0 - r_7$
4:	$p_0 \leftarrow X_1 \times X_2$	$q_0 \leftarrow Y_1 \times Y_2$	15:	$l_9 \leftarrow l_8$	$r_9 \leftarrow q_0 + r_7$
5:	$p_1 \leftarrow l_0 \times r_0$	$q_1 \leftarrow Z_1 \times Z_2$	16:	$l_{10} \leftarrow p_0 + q_0$	$r_{10} \leftarrow q_0 + q_1$
6:	$p_2 \leftarrow l_1 \times l_2$	$q_2 \leftarrow r_1 \times r_2$	17:	$l_{11} \leftarrow p_1 - l_{10}$	$r_{11} \leftarrow q_2 - r_{10}$
7:	$l_0 \leftarrow 3p_0$	$r_0 \leftarrow 3q_1$	18:	$p_4 \leftarrow l_{11} \times r_9$	$q_4 \leftarrow r_{11} \times l_7$
8:	$l_3 \leftarrow p_0 + q_1$	$\emptyset$	19:	$p_5 \leftarrow r_9 \times r_8$	$q_5 \leftarrow l_7 \times l_9$
9:	$l_4 \leftarrow p_2 - l_3$	$\emptyset$	20:	$p_6 \leftarrow l_{11} \times l_9$	$q_6 \leftarrow r_{11} \times r_8$
10:	$p_3 \leftarrow b \times l_4$	$q_3 \leftarrow b \times q_1$	21:	$X_3 \leftarrow p_4 - q_4$	$Y_3 \leftarrow p_5 + q_5$
11:	$l_5 \leftarrow p_3 - p_0$	$r_5 \leftarrow l_4 - q_3$	22:	$Z_3 \leftarrow p_6 + q_6$	$\emptyset$
23:	**return** (X_3, Y_3, Z_3)				

Algorithm 4 Parallel scheduling of point doubling for $a = -3$ using complete formulas.

Input: $P = (X_1, Y_1, Z_1)$, and the coefficient b from $E_{-3,b}(\mathbb{F}_p)$.
Output: $2P = (X_2, Y_2, Z_2)$.

	LEFT	RIGHT		LEFT	RIGHT
1:	$p_0 \leftarrow X_1 \times X_1$	$q_0 \leftarrow Y_1 \times Y_1$	9:	$l_4 \leftarrow l_0 - l_2$	$r_4 \leftarrow r_1 - r_2$
2:	$p_1 \leftarrow X_1 \times Y_1$	$q_1 \leftarrow Y_1 \times Z_1$	10:	$l_5 \leftarrow 3l_4$	$r_5 \leftarrow r_4$
3:	$p_2 \leftarrow 2X_1 \times Z_1$	$q_2 \leftarrow Z_1 \times Z_1$	11:	$p_4 \leftarrow l_3 \times l_5$	$q_4 \leftarrow r_3 \times r_5$
4:	$p_3 \leftarrow b \times p_2$	$q_3 \leftarrow b \times q_2$	12:	$p_5 \leftarrow p_1 \times r_5$	$q_5 \leftarrow q_1 \times l_5$
5:	$l_0 \leftarrow p_3 - p_0$	$r_0 \leftarrow q_3 - p_2$	13:	$\emptyset$	$q_6 \leftarrow q_1 \times r_1$
6:	$l_1 \leftarrow 3p_0$	$r_1 \leftarrow q_0$	14:	$X_2 \leftarrow 2(p_5 - q_5)$	$Y_2 \leftarrow p_4 + q_4$
7:	$l_2 \leftarrow 3q_2$	$r_2 \leftarrow 3r_0$	15:	$Z_2 \leftarrow 8q_6$	
8:	$l_3 \leftarrow l_1 - l_2$	$r_3 \leftarrow r_1 + r_2$			
16:	**return** (X_2, Y_2, Z_2)				

dition uses 12**M**, 2$\mathbf{M}_b$, and 29**A**, while the point doubling requires 8**M**, 2$\mathbf{M}_b$, 3**S**, and 21**A**; where $\mathbf{M}_b$ denotes the multiplication by the coefficient b of the elliptic curve. For point additions, Renes' formulas are as efficient as the Jacobian ones; however, there is a considerable amount of field additions.

4.1. Parallel Scheduling of Complete Addition Formulas

By analyzing the Renes' formulas, we notice that is possible to schedule field operations in such a way that two independent operations be executed in parallel. Our proposed schedule is listed in Algorithm 3 for point addition and in Algorithm 4 for point doubling. Assuming the use of two execution units, say LEFT and RIGHT units, the scheduling proceeds as follows. The LEFT unit will compute variables named p_i whenever the operation is a field multiplication; otherwise, they are called l_i. Analogously, the RIGHT unit will compute variables q_i for field multiplications, and r_i for the rest of the operations. Every line of Algorithm 3 (and Algorithm 4) computes pairs (p_i, q_i) or (l_i, r_i) using in most of the cases the same type of field operation. A line with $\emptyset$ symbol denotes no computation in this unit. The communication between units occurs when the LEFT unit uses variables previously computed by the RIGHT unit, and vice versa; we arranged field operations in such a way that minimizes the communication required.

We take advantage of the AVX2 vector unit to process field operations simultaneously since the 256-bit vector unit can also behave like two independent vector units of 128 bits. Hence, by making a slight modification in the distribution presented in Figure 2, we store two elements $a, b \in \mathbb{F}_{p_{384}}$ by packaging the digits a_i and b_i in seven 256-bit vector registers as Figure 3 depicts. The techniques for arithmetic operations presented in Section 3.3 are straightforwardly applicable to this representation.

The benefit of using the parallel scheduling proposed is notorious since it reduces by half the amount of operations required to compute point additions. For example, Algorithm 3 takes $6\widehat{\mathbf{M}}$, $1\widehat{\mathbf{M}}_b$, and $15\widehat{\mathbf{A}}$, where $\widehat{\cdot}$ denotes field operations executed in parallel. The communication between units is accomplished by performing permutations of the 128-bit parts that conform a 256-bit vector register using either the `VPERMQ` or the `VPERM2I128` instruction.

5. Elliptic Curve Point Multiplication

We focus on the optimization of three special cases of point multiplication. First, the *variable-point* multiplication, kP, where k represents a secret value, and P is an arbitrary point. Second, the *fixed-point* multiplication, kP, is processed under the assumption that the point P is previously known and fixed, and k is a secret value . Third, the *double-point* multiplication, $kP + lQ$, where P is a fixed point, and Q is an arbitrary point.

5.1. Variable-Point Multiplication

We compute variable-point multiplication using a deterministic execution pattern. First, we implemented the algorithm employed by Bos et al. [Bos et al. 2015, Alg. 1] and reproduced in Algorithm 5. For this case, we set $\omega = 6$, which implies to calculate a series of five point doublings followed by one point addition. Since the fixed-window recoding works only for odd numbers, we recode either k (when odd) or $\ell - k$ (when even), and at the end of the algorithm, we must choose between kP or $-kP$. We remark that both selections require running in constant time; hence, we use Algorithm 7 for meeting this requirement.

In line 8 of Algorithm 5, we perform queries, denoted by ϕ, to the look-up table according to the digits k'_i; to do that securely, we read the entire table and conditionally select the appropriate entry. The secure query (Algorithm 6) uses two primitives to operate: conditional move operation (Algorithm 8) and selection operation (Algorithm 7). We implement these functions using logic arithmetic over vector registers; this avoids that running time depends on the values of the operands reducing the success chances of a timing attack.

5.2. Fixed-Point Multiplication

In this case, the point multiplication can be accelerated by generating a look-up table containing some precomputed points. Since it is allowed to produce a large look-up table, this scenario exhibits a trade-off between speed and memory footprint. In our implementation, the fixed-point multiplication algorithm read the entire look-up table; hence, it would be desirable that a large part of the table fit on L1-D (data memory cache), which has a capacity of 32 KiB in the Intel Core i7 processors.

Algorithm 5 Variable-point multiplication using a deterministic pattern.

Input: P, k, and ω, where $P \in E$, k is a positive integer lesser than ℓ, and ω is a positive integer.
Output: Q, a point such that $Q = kP$.

1: $t \leftarrow \lceil \log_2(\ell)/(\omega - 1) \rceil$
2: $T = \{(2i + 1)P \mid \text{for } i \in [0, 2^{\omega-2})\}$
3: $k' \leftarrow \texttt{Select}(\ell - k, k, k \bmod 2, 0)$
4: $(k'_0, \ldots, k'_{t-1}) \leftarrow \texttt{Recoding}(k', \omega)$
5: $Q \leftarrow \mathcal{O}$
6: **for** $i \leftarrow t - 1$ **to** 0 **do**
7: $\quad Q \leftarrow 2^{\omega-1}Q$
8: $\quad Q \leftarrow Q + \phi(T, k'_i)$ *(Alg. 6.)*
9: **end for**
10: $Q_y \leftarrow \texttt{Select}(-Q_y, Q_y, k \bmod 2, 0)$
11: **return** Q

Algorithm 6 Secure look-up table query ϕ.

Input: T and v, where T is a look-up table storing d points; and v is an integer stored into a n-bit word.
Output: R, a point in projective coordinates such that $R = T_v$.

1: $(x, y) \leftarrow (0, 0)$
2: $(V, S) \leftarrow \left(\lfloor \frac{v+1}{2} \rfloor, v \gg (n - 1)\right)$
3: **for** $i \leftarrow 1$ **to** d **do**
4: $\quad (x_T, y_T) \leftarrow T_i$
5: $\quad x \leftarrow x \oplus \texttt{CMove}(x_T, i, V)$ *(Alg. 8.)*
6: $\quad y \leftarrow y \oplus \texttt{CMove}(y_T, i, V)$ *(Alg. 8.)*
7: **end for**
8: $y \leftarrow \texttt{Select}(1, 0, V, 0)$ *(Alg. 7.)*
9: $z \leftarrow \texttt{Select}(0, 1, V, 0)$ *(Alg. 7.)*
10: $y \leftarrow \texttt{Select}(y, -y, S, 0)$ *(Alg. 7.)*
11: **return** $R \leftarrow (x, y, z)$

Algorithm 7 `Select`: Secure selection operation.

Input: A, B, x, and y, where A and B are two k-bit integer numbers; and, x and y are two n-bit integers.
Output: If $x = y$, returns A; otherwise, returns B.

1: $b \leftarrow (-(x \wedge y)) \gg (n - 1)$
2: $M \leftarrow (\underbrace{bbb\cdots}_{n \text{ times}})_2$
3: **return** $(\neg M \wedge A) \oplus (M \wedge B)$

Algorithm 8 `CMove`: Secure conditional move operation.

Input: A, x, and y, where A is a k-bit integer number; and, x and y are two n-bit integers.
Output: If $x = y$, returns A; otherwise, returns 0.

1: $b \leftarrow (-(x \wedge y)) \gg (n - 1)$
2: $M \leftarrow (\underbrace{bbb\cdots}_{n \text{ times}})_2$
3: **return** $\neg M \wedge A$

Initially, we precompute a look-up table with entries $T_i = \{j2^{4i}P\}$ for all even $i \in [0, 96)$ and all $0 \leq j \leq 8$. For computing kP, first, we split scalar k in 96 signed digits of four bits, such that $k = \sum_{i=0}^{95} k_i 2^{4i}$. Then, kP is obtained as $kP = Q_0 + 2^4 Q_1$, where $Q_0 = \sum_{i=0}^{95} \phi(T_i, k_i)$ for $i \equiv 0 \bmod 2$, and $Q_1 = \sum_{i=0}^{95} \phi(T_{i-1}, k_i)$ for $i \equiv 1 \bmod 2$. This computation requires in total 96 point additions and 4 point doublings; whereas regarding memory footprint, the look-up table stores 384 points in affine coordinates accounting for 36 KiB of read-only memory; what is close to the size of L1-D. Like in the case of variable-point multiplication, the queries are securely processed by Algorithm 6.

5.3. Double-Point Multiplication

The $kP + lQ$ operation can be computed using the interleaved algorithm together with ω-NAF representation [Hankerson et al. 2003, Alg. 3.51]. This algorithm encodes both scalars k and l into ω-NAF expansions [Solinas 2000]. Indeed, the value of the window-ω can be different for each scalar; thus, let ω_k and ω_l be the window values selected for the scalar k and l, respectively. The value ω_k impacts significantly on both the running time

and the memory footprint of the implementation, whereas the ω_l value impacts only on the running time. In our implementation, we choose $\omega_k = 7$ and $\omega_l = 5$ since these values minimize the running time of the double-point multiplication.

6. Benchmarking Results

To evaluate the impact on the performance of our optimizations, we measure the time to compute the shared secret in the ECDH protocol, the signature generation, and the verification operation of the ECDSA scheme. Then, we make a comparison against other publicly-available cryptographic libraries. We choose libraries mainly written in C-language that offer support to the P-384 curve, and we end up with the following:

- OpenSSL (v1.0.2h) is the state-of-the-art on open-source cryptographic implementations; this library is in continuous development containing a vast number of optimizations [The OpenSSL Project 2003].
- Nettle (v3.2) is a low-level cryptographic library that provides a back-end functionality to the GnuTLS library [Niels Möller 2001].
- mbed TLS (v2.2.1), previously known as PolarSSL, is a multi-platform library with the aim to ease the development of cryptographic algorithms [Bakker 2008].
- Relic toolkit (v0.4.1) is a modern library that supports a broad range of cryptographic algorithms. Measurements were done using the default configurations, and setting GMP as the arithmetic backend [Aranha and Gouvêa 2009].
- BoringSSL (commit `fe47ba2f`) is a fork of the OpenSSL library modified with the aim to cover the requirements needed by Google's products [Google 2015].

Table 2 shows the performance comparison obtained. As can be seen, the cryptographic libraries selected offer similar performance for the P-348 curve, except the mbed TLS library, which is by far the slowest. In all scenarios, our software implementation outperforms the timings of the other libraries. We want to note that these libraries still compute the point addition using the Jacobian formulas, which present special cases that must be handled in constant time, and this could add a considerable performance penalty.

The authors of complete point addition formulas presented a proof-of-concept implementation of their formulas using the OpenSSL library [Renes et al. 2016]; they observed a slow-down factor of $1.41\times$ for computing the ECDH protocol. We state that this factor can be reduced by using the implementation techniques presented in this work. For example, we benchmark the prime field arithmetic operations and notice that our implementation computes modular additions $3\times$ faster than the OpenSSL library, which is a relevant result since the complete formulas require lots of prime field additions; thus, our implementation reduces the overhead caused by the complete formulas.

Regarding differences in the performance observed on the Haswell and the Skylake micro-architectures, our implementation had better performance improvement that the other libraries. For example, in the shared secret computation, the running time of our software is around 10% faster when is executed on Skylake, whereas the other libraries are accelerated only in 5%. A plausible explanation of this situation is that the improvements in the latest micro-architectures have a higher impact on vector instructions rather than the native scalar instructions. Therefore, it is expected that vectorized code will have a greater performance in the forthcoming micro-architectures.

Table 2. Performance of ECC protocols using the P-384 curve. Entries represent 10^6 clock cycles measured on a Haswell (Core i7-4770) and on a Skylake (Core i7-6700K) micro-architectures. All libraries were compiled using the GNU Compiler Collection (v5.3.1) and executed with the Intel Turbo Boost and the Intel Hyper-Threading technologies disabled.

Cryptographic library	Haswell			Skylake		
	ECDH	ECDSA		ECDH	ECDSA	
	Shared secret	Signature	Verification	Shared secret	Signature	Verification
mbed TLS	18.70	7.15	26.49	17.90	6.72	25.25
BoringSSL	3.60	3.78	4.47	3.43	3.67	4.33
Relic toolkit	1.79	0.89	2.40	1.52	0.77	2.06
Nettle	—	0.77	2.07	—	0.62	1.57
OpenSSL	2.12	0.65	2.60	2.03	0.62	2.49
This work	1.25	0.56	1.31	1.11	0.53	1.11

7. Concluding Remarks

Given the recent discovery of the efficient evaluation of the complete formulas for point addition, we contributed with a new parallel scheduling of field operations that allows computing two independent operations simultaneously. This technique works either for hardware or software implementations. Moreover, the scheduling is not restricted to the P-384 curve, since by applying slight modifications, it applies to all prime elliptic curves in short Weierstrass form.

Additionally, we presented software techniques for implementing the arithmetic operations of $\mathbb{F}_{p_{384}}$ using vector instructions. Essentially, we packed pairs of field elements into 256-bit vector registers, and we arranged vector instructions to perform two field operations at the same time. These operations served as building blocks for the implementation of the parallel scheduling of the complete formulas for point addition. In fact, we optimized our implementation to make an efficient use of the execution engine for micro-architectures supporting the AVX2 instruction set.

The widespread use of ECC can be hampered by low-performance implementations provided by some cryptographic libraries. In this work, the benchmark results revealed that the use of parallel execution units improves the performance of ECC. Specifically, we observed that the combination of the parallel scheduling proposed together with the efficient computation of field arithmetic allowed to obtain a significant reduction in the running time of the ECDSA scheme and the ECDH protocol using the P-384 curve. Finally, we expect that the techniques presented in this work contribute to improving the performance of elliptic curves on processors that support vector instructions.

Acknowledgments. We thank the anonymous reviewers for their helpful comments.

References

[ANSI 1999] ANSI (1999). ANS X9.62 Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).

[ANSI 2001] ANSI (2001). ANS X9.63 Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography.

[Aranha and Gouvêa 2009] Aranha, D. F. and Gouvêa, C. P. L. (2009). RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit.

[Bakker 2008] Bakker, P. (2008). mbed TLS. (v2.3). https://tls.mbed.org/.

[Barker et al. 2007] Barker, E. B., Johnson, D., and Smid, M. E. (2007). SP 800-56A. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised).

[Bos et al. 2015] Bos, J. W., Costello, C., Longa, P., and Naehrig, M. (2015). Selecting elliptic curves for cryptography: an efficiency and security analysis. *Journal of Cryptographic Engineering*, pages 1–28.

[Bos et al. 2014] Bos, J. W., Halderman, J. A., Heninger, N., Moore, J., Naehrig, M., and Wustrow, E. (2014). *Elliptic Curve Cryptography in Practice*, pages 157–175. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Bosma and Lenstra 1995] Bosma, W. and Lenstra, H. (1995). Complete Systems of Two Addition Laws for Elliptic Curves. *Journal of Number Theory*, 53(2):229–240.

[Brown 2009] Brown, D. R. L. (2009). SEC 1: Elliptic Curve Cryptography. http://www.secg.org/sec1-v2.pdf.

[CNSS 2012] CNSS (2012). National Information Assurance Policy on the Use of Public Standards for the Secure Sharing of Information Among National Security Systems. CNSSP Policy 15. https://www.cnss.gov/CNSS/issuances/Policies.cfm.

[CNSS 2015] CNSS (2015). Use of Public Standards for the Secure Sharing of Information among National Security Systems. CNSS Advisory Memorandum 02-15. https://www.cnss.gov/CNSS/issuances/Memoranda.cfm.

[Google 2015] Google (2015). BoringSSL. https://boringssl.googlesource.com/boringssl/+/fe47ba2fc5512436696f745b5756d08c7d8ceb0b.

[Granger and Scott 2015] Granger, R. and Scott, M. (2015). Faster ECC over $\mathbb{F}_{2^{521}-1}$. In Katz, J., editor, *Public-Key Cryptography – PKC 2015: 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 – April 1, 2015, Proceedings*, pages 539–553, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Gueron and Krasnov 2014] Gueron, S. and Krasnov, V. (2014). Fast prime field elliptic-curve cryptography with 256-bit primes. *Journal of Cryptographic Engineering*, pages 1–11.

[Hankerson et al. 2003] Hankerson, D., Menezes, A. J., and Vanstone, S. (2003). *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[IEEE 2000] IEEE (2000). IEEE Standard Specifications for Public-Key Cryptography. Number 1363, pages 1–228. IEEE Std 1363-2000.

[Intel Corporation 2011] Intel Corporation (2011). Intel® Advanced Vector Extensions Programming Reference. Technical report. https://software.intel.com/sites/default/files/m/f/7/c/36945.

[Izu and Takagi 2002] Izu, T. and Takagi, T. (2002). Exceptional Procedure Attack on Elliptic Curve Cryptosystems. In Desmedt, Y. G., editor, *Public Key Cryptography — PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings*, pages 224–239, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Käsper 2012] Käsper, E. (2012). Fast Elliptic Curve Cryptography in OpenSSL. In Danezis, G., Dietrich, S., and Sako, K., editors, *Financial Cryptography and Data Security*, volume 7126 of *Lecture Notes in Computer Science*, pages 27–39. Springer Berlin Heidelberg.

[Koblitz 1987] Koblitz, N. (1987). Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209.

[Miller 1986] Miller, V. S. (1986). Use of Elliptic Curves in Cryptography. In Williams, H. C., editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer Berlin Heidelberg.

[Niels Möller 2001] Niels Möller (2001). Nettle. `http://www.lysator.liu.se/~nisse/nettle`.

[NIST 1999] NIST (1999). Recommended elliptic curves for federal government use.

[NIST 2000] NIST (2000). Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-2. `http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf`.

[Pollard 1975] Pollard, J. (1975). A monte carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334.

[Proos and Zalka 2003] Proos, J. and Zalka, C. (2003). Shor's Discrete Logarithm Quantum Algorithm for Elliptic Curves. *Quantum Information & Computation*, 3(4):317–344.

[Renes et al. 2016] Renes, J., Costello, C., and Batina, L. (2016). Complete Addition Formulas for Prime Order Elliptic Curves. In Fischlin, M. and Coron, J.-S., editors, *Advances in Cryptology – EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 403–428, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Rivest et al. 1978] Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

[Solinas 2000] Solinas, J. (2000). Efficient Arithmetic on Koblitz Curves. *Designs, Codes and Cryptography*, 19(2-3):195–249.

[Solinas 1999] Solinas, J. A. (1999). Generalized Mersenne Numbers. Technical Report CORR 99-39, Center of Applied Cryptographic Research (CACR).

[The OpenSSL Project 2003] The OpenSSL Project (2003). OpenSSL: The Open Source toolkit for SSL/TLS. `http://www.openssl.org`.

[Vanstone et al. 1992] Vanstone, S., Rivest, R. L., Hellman, M. E., Anderson, J. C., and Lyons, J. W. (1992). Responses to NIST's Proposal. *Communications of the ACM*, 35(7):41–54. (John Anderson communicated Vanstone's proposal).

Aplicação do Algoritmo de Colônia de Formigas para Redução do Tamanho de Chaves Públicas em Criptografia Completamente Homomórfica

Joffre Gavinho Filho[1], Jonice Oliveira[1], Claudio Miceli[1], Gabriel Pereira da Silva[2]

[1]Programa de Pós-Graduação em Informática

{joffreufrj, jonice, cmicelifarias}@gmail.com

[2]Departamento de Ciência da Computação

gabriel.silva@ufrj.br

Universidade Federal do Rio de Janeiro

Abstract. *The fully homomorphic encryption, one of the areas post-quantum cryptography, is one of techniques for processing and manipulating encrypted data without deciphering them. Used on platforms where traditional encryption may not have the desired security, such as cloud computing case. Several methods have been proposed to facilitate the use of this technique. However, for its effective use, there is the need to generate parameters that result in keys with extremely large sizes. This article aims to reduce the size of public keys using the optimization algorithm ant colony to ratify the calibration of the variants of algorithms already present in the literature.*

Resumo. A *criptografia completamente homomórfica, uma das áreas da criptografia pós-quântica, é uma das técnicas para o processamento e manipulação de dados criptografados sem decifrá-los. Utilizada em plataformas onde a criptografia tradicional pode não ter a segurança desejada, como o caso da computação em nuvem. Diversos métodos têm sido propostos para viabilizar o uso dessa técnica. Porém, para a sua efetiva utilização, há a necessidade da geração de parâmetros que resultam em chaves com tamanhos extremamente grandes. Este artigo tem como objetivo reduzir o tamanho das chaves públicas usando o algoritmo de otimização por colônia de formigas para ratificação da calibração das variantes de algoritmos já presentes na literatura.*

1. Introdução

A Computação em Nuvem (*Cloud Computing*) – definida como uma plataforma para a utilização de uma estrutura com capacidades de armazenamento e processamento compartilhados e interligados por meio da Internet [Sousa 2009] – tem se tornado uma das soluções utilizadas na redução de custos para atender às crescentes demandas em informática nos tempos modernos.

Dentre todas as metas de projetos necessárias à viabilização da Computação em Nuvem, destacamos a provisão de segurança que, basicamente, tem a Criptografia como um dos métodos mais utilizados na determinação de confidencialidade e sigilo das informações manipuladas pela nuvem [NIST 1978]. Porém, os algoritmos de criptografia convencionais possuem uma característica em comum que inserem um

ponto de vulnerabilidade extremamente sensível no contexto de segurança na nuvem, a denominada "não-maleabilidade" [Coron 2011]. Este conceito requer que todos os dados devam ser descriptografados antes de serem processados, principalmente em ambiente compartilhado, isto é, em cada um dos servidores, componente da nuvem, onde haja necessidade de manipulação ou processamento dos dados, há a necessidade de se transformá-lo em texto claro para que se possa validar ou mesmo comparar os dados que estão codificados. Tendo-se então a necessidade de acesso direto aos dados originais. Em suma, o principal problema com ambiente compartilhado e a criptografia, reside no processamento dos dados, porque na criptografia tradicional, os dados não podem ser alterados enquanto criptografados.

Na Criptografia Homomórfica – que faz parte da chamada criptografia pós-quântica: ramo da criptografia que estuda as classes de algoritmos criptográficos resistentes à criptoanálise baseada na computação quântica [Daniel 2009] – os dados encriptados são manipulados em sua forma bruta, isto é, sem a necessidade de decifrá-los. Os modelos de encriptação homomórfica (EH) são baseados em somas e multiplicações de funções, isto é, além dos convencionais algoritmos de encriptação e desencriptação [Stalling 2007], há também um algoritmo de avaliação, que toma como entrada uma mensagem encriptada E(m) e retorna uma criptografia de f(E(m)) [Gentry 2009]. Os esquemas EH podem ser basicamente classificados em dois tipos: o primeiro são os esquemas, chamados de parcialmente homomórficos, que, além de operações de criptografia e decriptografia também executam operações de soma ou multiplicação, que tomam como entrada mensagens criptografadas m_1 e m_2 e regressam a criptografia de $m_1 + m_2$ ou $m_1 * m_2$, respectivamente. Se um esquema EH suporta tanto adição quanto multiplicação, ele também pode avaliar qualquer circuito aritmético em dados criptografados [Gentry 2009] e, portanto, podemos dizer que é um esquema de Criptografia Completamente Homomórfica (CCH), ou seja, se o $E(m)$ é a codificação de uma mensagem m, um modelo de criptografia é totalmente homomórfico se: $f(m) \rightarrow E(m_1 + m_2) = E(m_1) + E(m_2)$; e, $f(m) \rightarrow E(m_1 * m_2) = E(m_1) * E(m_2)$ [Coron 2011]. Usando esse tipo de regime, qualquer circuito pode receber uma avaliação homomórfica, permitindo a construção de programas que podem ser executados com as criptografias de suas entradas para produzir uma saída criptografada. Como esses programas não decodificam as informações, eles podem ser utilizados por terceiros não confiáveis sem revelar a sua entrada e estado interno. Por exemplo, podem-se somar dois números criptografados e, a menos que se possa decriptografar o resultado, não há como se descobrir o valor dos números originais individualmente [Gavinho 2015].

O grande problema dos métodos propostos para os sistemas completamente homomórficos é que os seus tempos de execução, bem como o tamanho dos parâmetros utilizados, especialmente os das chaves públicas, crescem ciclicamente a cada iteração em complexidade na ordem de $O(\lambda^{10})$ [Coron 2011]. Onde λ é o parâmetro de segurança de todo o regime, e que define o tamanho, em bits de comprimento, das chaves geradas. Melhorias dos processos estão sendo propostas. Como é o caso [Coron 2011] que propôs um esquema de chaves públicas DGHV [Dijk 2010] totalmente homomórfico, e que reduz o tamanho dessas chaves geradas para aproximadamente $O(\lambda^{7})$ [Coron 2011], e em [Coron 2012] onde tal redução chega a $O(\lambda^{5})$. Apesar de novas técnicas terem sido propostas [Mukherjee 2016], foi verificado que [Mukherjee 2016], os tamanhos de redução das chaves públicas têm sido pouco expressivos quando comparados a [Coron 2012], justificativa esta na qual tomamos como base para a

utilização especifica das primitivas de criptografia homomórfica de [Coron 2012] como fundamentos deste trabalho.

O trabalho que aqui apresentamos visa aplicar a meta-heurística de Otimização por Colônia de Formigas (*Ant Colony Optimization* - ACO), como forma de demonstrar as técnicas de redução de chaves públicas utilizadas por [Coron 2012], com os objetivos principais de: especificar, validar e ratificar a calibração dos parâmetros do regime das variantes de [Coron 2012] realizada por Gavinho em: [Gavinho 2015] e [Gavinho 2016], no qual o autor demonstra por meio da utilização de algoritmos genéticos (AG), a redução em cerca de 600 KB no tamanho da Chave Pública de 18 MB da compressão feita por [Coron 2012], mantendo-se, mesmo assim, os mesmos níveis de segurança alcançados por Coron [Coron 2012].

Este artigo está organizado da seguinte forma: na Seção 2 são descritos os mecanismos de Criptografia Completamente Homomórfica e os métodos de compactação de chaves públicas, bem como os trabalhos relacionados; na Seção 3, a proposta de avaliação e otimização é apresentada, bem como os experimentos realizados e as análises dos resultados obtidos; e finalmente, na Seção 4 são tecidas as conclusões finais e apresentadas as propostas de trabalhos futuros.

2. Trabalhos Relacionados

Nesta seção, descrevemos um breve histórico da criação e desenvolvimento das técnicas homomórficas, bem como os métodos e trabalhos propostos para a redução de chaves no processo de Criptografia Completamente Homomórfica.

2.1. Criptografia Completamente Homomórfica (CCH)

Os criptógrafos reconheceram há muitos anos a necessidade de um algoritmo de criptografia que permitisse a computação arbitrária em dados criptografados. A primeira proposta teórica prática foi feita em 1978 por Rivest, Adleman e Dertouzos [Rivest 1978], quando os autores sugeriram a construção de homomorfismos secretos - *privacy homomorphisms* [Rivest 1978] - como uma técnica para proporcionar um forma de proteção para a computação em dados sensíveis. O esquema proposto era parcialmente homomófico, i.e., usava apenas a propriedade multiplicativa do método e não a aditiva, além de não oferecer proteção contra ataques de texto simples escolhido (*Chosen Plaintext Attack - CPA*) [Morris 1993], não possuindo por isso segurança semântica. Caracterizamos, em criptografia, um esquema como semanticamente seguro se: dado um texto cifrado de qualquer mensagem, e o tamanho dessa mensagem, não há nenhum algoritmo probabilístico em tempo polinomial (*Probabilistic Polynomial-Time Algorithm - PPTA*), [Michael 1980], que possa determinar qualquer informação da mensagem com significado maior do que a probabilidade de uma escolha aleatória. Isto é: o conhecimento da mensagem cifrada e o tamanho da mensagem original, não revela nenhuma informação sobre esta mensagem, bem como, nenhum dado pode ser extraído com facilidade a partir do texto cifrado.

Após a bem sucedida construção de homomorfismos secretos - *privacy homomorphisms* [Rivest 1978], a comunidade científica começou a procurar implementações diversas para esta teoria – algoritmos capazes de executar a chamada criptografia completament homomórfica. Muitas tentativas sem sucesso foram realizadas, tanto que o problema permaneceu sem solução até pouco anos atrás – 2009 –

quando, Craig Gentry [Gentry 2009], demonstrou uma solução sugerindo a utilização de reticulados ideais na construção de um sistema de encriptação completamente homomórfico. Apesar do sucesso na solução, a proposta de Craig Gentry não foi suficientemente eficaz para ser usado na prática, devido à complexidade das avaliações de multiplicações e o tamanho excessivamente grande das chaves públicas geradas no processo.

O primeiro grande desafio em criptografia completamente homomórfica é torná-la eficientemente prática. Enquanto a construção original de [Gentry 2009] é vista como impraticável, as últimas construções e os esforços de implementação melhoraram sensivelmente a eficiência da criptografia completamente homomórfica. As propostas iníciais de implementações com foco na proposta original de Gentry e suas variações [Gentry 2010] melhoraram os gargalos de eficiência. Implementações posteriores fizeram uso de avanços recentes nos algoritmos [Dijk 2010] e das técnicas algébricas [Brakerski 2012] procurando melhorar a eficiência concreta dos resultados em tais esquemas Porém, o grande desafio está baseado no excessivo tamanho das chaves públicas criadas por esses métodos, bem como a necessidade de uma estrutura de armazenamento robusta. Em Dijk, Gentry, Halevi e Vaikuntanathan (DGHV) [Dijk 2010], por exemplo, o tamanho da chave pública se encontra na ordem de $O(\lambda^{10})$ bits e o tamanho da chave privada cerca de $O(\lambda^2)$ – sendo λ o parâmetro segurança de todo os esquemas. Coron, Mandal e Naccache [Coron 2011] propuseram uma modificação na geração de chaves, a fim de reduzir o tamanho da chave pública para $O(\lambda^7)$. A modificação consiste na utilização de formas quadráticas dos elementos da chave pública, em vez de formas lineares, como é feito no esquema DGHV. A ideia de Coron [Coron 2011] é armazenar apenas um pequeno subconjunto da chave pública e, quando necessário, gerar a chave pública completa multiplicativamente combinada aos elementos do subconjunto das chaves secretas. Esta proposta mantém a segurança semântica, e se baseia no problema do máximo divisor comum aproximado parcial (*Partial Approximate Greatest Common Divisor*) [Coron 2011], que consiste em retirar o erro dos primeiros conjuntos de chaves públicas criadas durante o processo. Uma segunda variante do método de otimização proposta em [Coron 2012], reduz ainda mais o tamanho das chaves prara $O(\lambda^5)$. Este esquema tem o mesmo problema básico de segurança do máximo divisor comum aproximado (*Approximate Greatest Common Divisor*) [DGHV 2010], isto é, o problema da Segurança Circular [Boneh 2008], que, dado um esquema de criptografia *E*, dizemos que *E* tem segurança circular se for seguro criptografar a chave privada com sua própria chave pública, esquema este que é a base do regime de segurança do DGHV, descrito a seguir.

2.1.1. O Esquema DGHV

Usando apenas álgebra modular ao longo de um anel de inteiros, Dijk, Gentry, Halevi e Vaikuntanathan propuseram um esquema totalmente homomórfico denominado DGHV [Dijk 2010]. Esquema que se provou ter uma menor complexidade matemática quando comparada com os regimes baseados em reticulados ideais de Gentry [Gentry 2009]. Coron, com base neste mesmo padrão, propôs suas duas variantes do método DGHV [Coron 2011] e [Coron 2012], conseguindo a otimização do custo computacional de certas primitivas, e a diminuição do tamanho do esquema de chaves públicas originais utilizando várias técnicas de redução e de compressão de chaves.

Basicamente, o esquema DGHV [Dijk 2010], é composto de cinco algoritmos – também chamados de primitivas: *E*(*KeyGen*, *Encrypt*, *Decrypt, Recrypt, Evaluate*), A primitiva *Keygen* é o esquema responsável por gerar o par de chaves (públicas e privadas); *Encrypt* é responsável por gerar o texto cifrado; *Decrypt* é responsável por decifrar o texto encriptado; *Recrypt* é utilizada para a criptografia da criptografia, isto é, a criptografia em níveis; e, *Evaluate* que realiza o processamento em tuplas de bits codificados em um circuito lógico, e que retorna o equivalente criptografado a este circuito como se fosse aplicado aos dados originais, isto é, faz a manipulação homomórfica nos bits codificados, sem a necessidade de decriptografá-los.

2.1.2. O Método de Coron.

O primeiro trabalho de Coron, denominado de sua primeira variante [Coron 2011], incluiu a adição de novos parâmetros quadráticos ao esquema de primitivas DGHV, armazenando assim apenas um pequeno conjunto de valores relacionados com a chave pública e, em seguida, gerando uma chave pública completa em tempo de execução. Usando esta técnica, Coron demonstrou a redução do tamanho da chave pública da ordem de $O(\lambda^{10})$ para a de $O(\lambda^{7})$, [Coron 2011], quando comparados ao DGHV.

Em sua segunda variante Coron [Coron 2012], demonstrou a redução do comprimento das chaves públicas produzidas para a ordem de $O(\lambda^{5})$. A principal inovação proposta por Coron neste esquema é que, em vez de armazenar todos os elementos-chave da criptografia, ele só armazena o valor de correção em relação a um gerador de números aleatórios. Assim, os dados a serem armazenados são menores, e, havendo necessidade dos dados, estes são recuperados "*on-the-fly*" pelas primitivas *Encrypt*, *Recrypt*, *Decrypt* e *Expand* (nova primitiva para expandir os blocos de dados reduzidos). Além disso, é descrita uma técnica de troca de módulos, que permite este regime, sem usar a estrutura proposta de *boostraping* descrito por Brakerski, Gentry e Vaikuntanathan, (BGV) [Brakerski 2012].

O regime inicial de inteiros usados por Coron como base para o seu trabalho, bem como para a criação de sua segunda variante, é fundamentada no trabalho de Gentry [Gentry 2009]. Este definiu o DGHV com base em um conjunto de inteiros, $x_i = p.q_i + r_i$, p $0 \leq i \leq \tau$, sendo λ o parâmetro de segurança em que se baseia todo o esquema. Os seguintes parâmetros devem ser utilizados para compor o esquema de Criptografia Homomórfica Reduzida (CHR) [Brakerski 2012], que, em seguida, deve ser reforçada para gerar o CCH de inteiros [Gavinho 2016]:

- γ é o comprimento em bits de x_i.
- η é o comprimento em bits da chave secreta p.
- ρ é o comprimento em bits do ruído r_i.
- τ é o número da chave pública de x_i.
- ρ' é um parâmetro de ruído secundário usado para criptografar, [Coron 2011].

O esquema é definido com as seguintes restrições:

- $\rho = \omega\ (log\lambda)$, necessário para a proteção contra ataques de força bruta.
- $\eta \geq \rho.\Theta\ (\lambda log 2\lambda)$, necessário para a execução de operações homomórficos de avaliação.
- $\gamma = \omega\ (\eta 2.log\lambda)$, necessário para a proteção a ataques com base no problema do MDC.
- $\tau \geq \gamma + \omega\ (log\lambda)$, necessário para reduzir a abordagem do MDC Aproximado.
- $\rho' = \rho + \omega\ (log\lambda)$, necessário para o parâmetro de ruído secundário.

2.1.3. A Otimização do Tamanho das Chaves Públicas de Gavinho.

O processo de otimização proposto por [Gavinho 2015] e [Gavinho 2016] na segunda variante de Coron, [Coron 2012], para a compactação e redução de chaves, consiste basicamente na calibragem dos valores dos parâmetros usados nas primitivas criptográficas através do Algoritmo Genético (AG), que, segundo [Lacerda 1999], são métodos de otimização e busca, inspirados nos mecanismos de evolução de populações de organismos.

Quanto à simulação, o esquema proposto por [Gavinho 2015] [Gavinho 2016], tanto para a criptografia parcialmente, como para a completamente homomórfica, foi implementado no software matemático Matlab/Simulink©, bem como todas as primitivas criptográficas descritas em [Coron 2012]. Coron implementou toda a sua proposta para o esquema DGHV utilizando o software matemático SAGE (Sistema para Experimentação de Álgebra e Geometria) [William 2012]. Como base comparativa, todas as métricas e primitivas de [Coron 2012], foram reimplementadas, e simuladas por [Gavinho 2016] no Matlab/Simulink©, com a efetiva calibração do simulador para que todas as métricas e tempos de execução repetissem fielmente os métodos em estudo. Os testes foram realizados, e os resultados analisados e comparados com os resultados obtidos pelos autores em seus trabalhos, tendo como resposta a igualdade de soluções que corroboraram com os resultados apresentados pelos artigos originais.

Como métricas para os valores dos parâmetros de segurança, foram utilizados os definidos por [Coron 2011]: *Toy* possuindo um lambda (λ) equivalente a 42 bits de segurança, *Small* equivalente a 52 bits, *Medium* equivalente a 62 bits e *Large* equivalente a 72 bits. Sendo λ o parâmetro que define o nível de segurança atribuído diretamente ao par de chaves gerados pela primitiva de geração de chaves *KeyGen,* além de definir todo o nível de segurança do método.

Como método comparativo, [Gavinho 2015] se baseou no proposto pela literatura especializada que, normalmente, utiliza a medida do tempo de execução de cada uma das primitivas criptográficas (os algoritmos) a fim de quantificar e avaliar o desempenho de cada uma delas [Gavinho 2016]. As primitivas são executadas de maneira repetitiva – todos os testes seguem uma distribuição normal e possuem intervalo de confiança de 95% – e tem o seu tempo de execução contabilizado por software. Em posse do tempo contabilizado e do número de vezes de execução das primitivas aplica-se uma média aritmética simples sobre os mesmos, obtendo assim o tempo médio de execução das respectivas primitivas que pode ser utilizado de maneira comparativa entre variadas implementações e variados esquemas homomórficos [Gavinho 2016].

Para otimização do método de [Coron 2012], [Gavinho 2015] utilizou uma Função Utilidade Aditiva (FUA) [Keeney 1976] com a finalidade de explorar e dar "pesos" ao Espaço de Soluções $[\lambda - 2, \lambda + 3]$, isto é, $39< \lambda<46$, $49< \lambda<56$, $59< \lambda<66$ e $69< \lambda<76$, sendo a FUA definida por: $S_{H\text{-}i} = T\lambda_{\text{-}1}P_1 + T\lambda_{\text{-}2}P_2 + T\lambda_{\text{-}3}P_3 + T\lambda_{\text{-}4}P_4 + T\lambda_{5}P_5 + T\lambda_{\text{-}6}P_6$, onde $S_{H\text{-}i}$ é a função Utilidade (Seleção); $T\lambda_{\text{-}i}$ é o peso do parâmetro λ_i sendo $\sum \lambda_{i\ =}\ 1$, e; P_i equivale ao elemento do espaço de soluções $[\lambda - 2, \lambda + 3]$ a ser determinado, por meio do seu peso, qual a sua influência real no processo de criptografia.

Os métodos de avaliações, calibrações e análises, propostas em [Gavinho 2015] [Gavinho 2016], iniciam-se com o processo de geração de valores ponderados para os pesos dos parâmetros λ, por meio do Algoritmo Genético (AG), e iniciados com a produção de uma massa de dados de 500 MB sem formatação. Essa massa de dados é gerada seguindo as métricas das primitivas de [Coron 2011] e reproduzem o estado original de testes de suas variantes [Coron 2011]. Esta massa é inicialmente usada para duas finalidades: *i*) a calibração dos módulos de forma análoga às primitivas de [Coron 2011], principalmente do algoritmo de avaliação do tempo de execução das primitivas; e, *ii*) ser usada como dados de treinamento para o agoritmo genético (AG). Foram executadas diversas simulações para cada um dos tamanhos originais dos parâmetros de segurança λ: *Toy* (42 bits), *Small* (52 bits), *Medium* (62 bits) e *Large* (72 bits), seguindo a proposta de [Coron 2011]. Após esta fase inicial dos módulos de calibração e formação dos algoritmos genéticos, a fase evolucionária do processo é iniciada. Sendo processada em um total de 100 gerações, onde as fases de seleção (método da roleta), cruzamento (em dois pontos) e mutação [Lacerda 1999] são executadas repetidamente, com a finalidade da convergência do algoritmo para um valor central do tamanho do parâmetro de segurança λ.

Tabela 1. Valores dos parâmetros obtidos com a variação dos tamanhos do parâmetro λ com os repectivos tempos de execução das primitivas criptográficas e os tamanhos finais das Chaves Públicas.

λ	ρ	η	μ	γ x 10^6	Θ	*KeyGen*	*Encryp*	*Decrypt*	*Expand*	*Recrypt*	*Evaluate*	*pk size*
40	12	318	54	0,03	131	00:00:05	00:00:04	00:00:00	00:00:01	00:00:29	00:00:17	346
41	**13**	**325**	**55**	**0,04**	**157**	**00:00:05**	**00:00:05**	**00:00:00**	**00:00:01**	**00:00:39**	**00:00:19**	**350**
42	14	336	56	0,06	195	00:00:06	00:00:05	00:00:00	00:00:01	00:00:41	00:00:20	354
43	15	341	56	0,10	229	00:00:06	00:00:05	00:00:00	00:00:01	00:00:41	00:00:20	358
44	16	348	57	0,14	263	00:00:17	00:00:06	00:00:00	00:00:03	00:01:00	00:00:35	362
45	17	355	58	0,22	330	00:00:21	00:00:16	00:00:00	00:00:05	00:01:30	00:00:43	688
50	18	368	63	0,54	600	00:00:49	00:00:53	00:00:00	00:00:14	00:04:15	00:01:50	1.356
51	**19**	**379**	**64**	**0,62**	**668**	**00:00:59**	**00:00:59**	**00:00:00**	**00:00:14**	**00:04:40**	**00:02:19**	**1.523**
52	20	390	65	0,70	735	00:01:00	00:01:00	00:00:00	00:00:15	00:04:50	00:02:30	1.690
53	21	395	65	64,40	872	00:03:00	00:03:50	00:00:00	00:00:19	00:05:51	00:02:40	680.451
54	22	400	66	128,10	1.009	00:07:00	00:07:10	00:00:00	00:00:22	00:06:55	00:03:15	1.698.593
55	23	404	67	255,50	1.283	00:08:00	00:08:00	00:00:00	00:00:30	00:13:10	00:04:30	2.716.734
60	24	416	71	765,20	2.378	00:27:00	00:20:00	00:00:01	00:02:30	00:48:00	00:11:50	6.419.067
61	**25**	**423**	72	**892,60**	**2.651**	**00:27:50**	**00:20:45**	**00:00:01**	**00:03:00**	**00:50:00**	**00:13:20**	**7.406.356**
62	26	438	73	1.020,00	2.925	00:28.00	00:21:00	00:00:01	00:03:10	00:51:00	00:15:10	7.900.000
63	27	443	73	1.092,00	3.099	00:31:00	00:22:12	00:00:01	00:05:20	00:01:00	00:30:15	8.151.398
64	28	449	74	1.166,00	3.272	00:58:00	00:24:10	00:00:02	00:07:30	00:02:10	00:01:00	8.528.496
65	29	456	75	1.315,00	3.619	00:02:00	00:28:00	00:00:03	00:11:10	00:03:23	00:03:12	8.905.594
70	30	464	79	1.906,00	5.006	00:09:10	00:03:35	00:00:04	00:50:00	00:10:10	00:10:30	16.342.969
71	**32**	**473**	**80**	**2.054,00**	**5.353**	**00:09:55**	**00:05:12**	**00:00:05**	**00:50:00**	**00:11:00**	**00:11:45**	**17.368.750**
72	34	492	82	2.200,00	5.700	00:10:00	00:07:15	00:00:05	00:51:00	00:11:34	00:12:00	18.000.000
73	36	503	84	2.260,00	6.104	00:12:00	00:11:10	00:00:05	00:51:00	00:12:20	00:13:30	18.631.250
74	39	522	87	2.510,00	6.412	00:17:00	00:15:00	00:00:06	00:55:00	00:13:00	00:14:05	19.657.031
75	41	541	89	3.000,00	6.921	00:21:00	00:21:00	00:00:07	00:58:00	00:15:05	00:15:00	21.708.594

Em cada uma das gerações do algoritmo, 1000 rodadas foram realizadas para cada valor de λ. Variando-se na ordem de números inteiros [λ - 2, λ + 3]. Totalizando para cada parâmetro: *Toy*, *Small*, *Medium* e *Large*, 6.000 rodadas. Um total de 24.000 rodadas para cada geração. Por fim todo o processo evolutivo do AG necessitou de $24x10^6$ rodadas. Podemos observar na Tabela 1, quando da execução dos parâmetros de segurança λ, os tamanhos obtidos para cada primitiva, bem como os seus tempos de execução. O resultado final da convergência do AG para o $T\lambda_{-i}$ foi: $S_{H\text{-}i} = \mathbf{10}P_1 + \mathbf{25}P_2 + \mathbf{23}P_3 + \mathbf{17}P_4 + \mathbf{14}P_5 + \mathbf{11}P_6$, sendo o maior peso dado a $\mathbf{P_2}$ (λ=41, λ=51, λ=61 e λ=71).

Durante todas as rodadas de simulações, testes idênticos aos de [Coron 2011] são realizados com valores calculados para parâmetros λ na verificação da segurança,

incluindo a segurança semântica do método. Foi verificado que os valores dos parâmetros λ convergiram para os observados na Tabela 1. Valores que, além de serem uma unidade de magnitude menores do que os parâmetros definidos por [Coron 2011], possuem uma redução substancial nos tempos de execução em cada umas das primitivas criptográficas do mecanismo. Mantendo mesmo assim a segurança semântica em todo o processo. Apesar de terem sido calculados outros valores menores, bem como menores tempos de execução das primitivas criptográficas, observado na Tabela 1, tais valores, como por exemplo: λ = 40, 50, 60 ou 70, quando utilizados no método não provêem segurança semântica, não sendo adequado o seu uso. Na Tabela 2 , observamos os valores comparativos das reduções dos tempos de execução e das diferenças dos tamanhos finais das chaves públicas.

Tabela 2. Valores comparativos das reduções dos tempos de execução e das diferenças dos tamanhos finais das chaves públicas.

λ	ρ	η	β	γ	Θ	*KeyGen*	*Encrypt*	*Decrypt*	*Expand*	*Recypt*	*Evaluate*	*#Pk*
42-41	1,00	11,00	1,00	0,02	38,00	00:00:00:133	00:00:00:015	00:00:00:000	00:00:00:007	00:00:00:984	00:00:00:004	4
52-51	1,00	7,00	1,00	0,08	67,50	00:00:00:600	00:00:00:070	00:00:00:000	00:00:00:054	00:00:18:987	00:00:00:069	167
62-61	1,00	6,00	1,00	157,00	273,75	00:00:05:900	00:00:00:020	00:00:00:000	00:00:00:944	00:06:34:000	00:00:01:064	1.288.289
72-71	1,00	7,00	1,00	22,00	346,88	00:00:39:000	00:00:00:319	00:00:00:000	00:00:27:945	00:12:26:000	00:00:03:988	631.250

2.1.4. A Otimização por Colônia de Formigas.

A Otimização por Colônia de Formigas (ACO, do inglês *Ant Colony Optimization*) é uma meta-heurística baseada em população e inspirada no comportamento de busca de alimento (comportamento forrageiro) das formigas. O Algoritmo formulado na década de 1990 por Marco Dorigo [Dorigo 1994], e que têm evoluído desde sua publicação [Dorigo e Stutzle 2004], relacionada às habilidades das formigas em encontrar o caminho mais curto entre o ninho e o alimento. Esta busca é efetuada através da exploração das trilhas de feromônio, substância química depositada pelas formigas durante seu percurso, e no comportamento denominado estimergia (em inglês *stigmergy*), que é o tipo de comunicação indireta entre as formigas, na qual elas rastreiam os melhores caminhos. Segundo [Castro 2006], estimergia é o método de comunicação em que os indivíduos de um sistema se comunicam entre si pela modificação do ambiente local. Devido a este comportamento cooperativo e eficaz de busca elas vão construindo melhores alternativas no caminho para encontrar o alimento.

Podemos observar na Figura 1 – baseada nos experimentos das pontes binárias realizados por Deneubourg [Dorigo 2004] – a varredura do terreno, bem como a formação da trilha de feromônio. Na letra (a), uma situação de demonstração do cenário inicial, composto do ninho (colônia) das formigas, da fonte de alimentos e de três caminhos alternativos até a fonte de alimento: [01], [02] e [03], sendo os caminhos [01] e [03] de mesmo comprimento e maiores que o caminho [02]. Em (b), as formigas podem ser observadas em sua busca aleatória por alimentos no terreno, movendo-se randomicamente, ou seja, realizam buscas exploratórias por possíveis soluções. Na letra (c), houve a detecção da fonte de alimento, e inicia-se então o transporte do mesmo para o ninho, como o caminho [02] é o menor entre os três, as formigas que escolheram esse percurso, chegam primeiro à colônia, tendo como consequência um maior acúmulo de feromônio em [02]; finalmente em (d), observamos a convergência da maior parte das formigas para o menor caminho, [02]; pode-se observar também, que, apesar da descoberta do menor percurso ninho-alimento, ainda há formigas procurando, aleatoriamente, caminho alternativos.

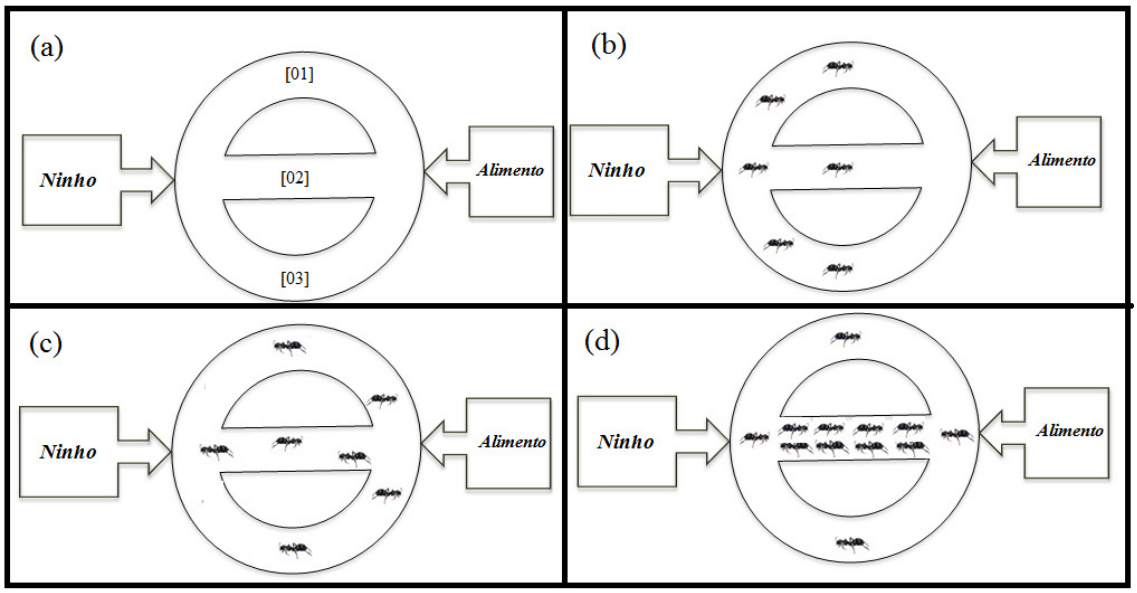


Figura 1. Busca por alimento: (a) condição inicial da colônia; (b) inicio da varredura no terreno; (c) descoberta do alimento; e, (d) formação da trilha de feromônio.

Este comportamento foi então simulado em algoritmos de otimização, conhecidos como ACO (do inglês *Ant Colony Optimization*-ACO). Estes algoritmos buscam melhores soluções nas trilhas em que se encontra a maior quantidade de feromônio, com o controle de seu depósito e evaporação. Realizam-se também atualizações locais e globais de feromônio, melhorando assim a busca de resultados e alternativas por caminhos não trilhados. As formigas artificiais constroem soluções de forma probabilística utilizando duas informações: (*i*) a trilha de feromônio (artificial) que muda dinamicamente durante a execução do programa de modo a refletir a experiência já adquirida durante a busca, e; (*ii*) a informação heurística específica do problema a ser resolvido. [Dorigo 2004] apresenta a correspondência entre o que acontece na natureza e o algoritmo ACO, correspondência representada na Tabela 2.

Tabela 2 - Correspondência entre a natureza e o ACO.

Natureza	ACO
Possíveis caminhos entre o ninho e o alimento	Conjunto de possíveis soluções
Caminho mais curto	Solução ótima
Ação via comunicação por feromônio	Procedimento de otimização

A construção da solução pela formiga artificial *k* começa de uma posição *i* , na Figura 2 → n_i, movendo-se e escolhendo probabilisticamente a posição vizinha *j* (entre os vizinhos factíveis), isto é uma das arestas A_i.

A probabilidade da formiga *k* que está no ninho de escolher a aresta $ij(A_i)$ para acessar o alimento f_i é dada pela regra [Dorigo 2004]:

$$p_{ij}^{k} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{i \in N_i^k}(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}} \text{ , se } j \in N_i^k \text{ ; senão } p_{ij}^{k} = 0.$$

onde: τ_{ij}^{k} → é a quantidade de feromônio associada à aresta *ij*; α e β são parâmetros para determinar a influência do feromônio e da informação heurística, respectivamente; N_i^k

são os caminhos possíveis (arestas *ij*) associado à *k*-ésima formiga; η_{ij} é a informação heurística ou valor heurístico que é definida em função das características do problema.

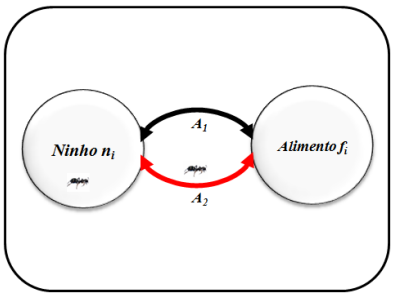


Figura 2. Diagrama formado pelo ninho n_i, fonte de alimento f_i e possíveis caminhos (arestas A_i) para a formiga *k* acessar a fonte de alimento f_i.

A atualização de feromônio, dada por: $\tau_{ij}^{k} \leftarrow \tau_{ij}^{k} + \Delta\tau^{k}$, ocorre no retorno do circuito da *k*-ésima formiga, depositando uma quantidade $\Delta\tau^{k}$ de feromônio nas arestas visitadas. O valor de $\Delta\tau^{k}$ é assumido constante para todas as formigas. Esta variação de feromônio faz com que os caminhos mais curtos sejam alcançados pela intensificação da quantidade de feromônio nos melhores trechos.

A evaporação do nível de feromônio nos caminhos tende a permitir uma convergência das formigas para um valor subótimo, ou seja, uma solução nas proximidades do ótimo. Como a quantidade de feromônio diminui, isto leva à exploração de caminhos alternativos durante o processo de busca. A evaporação limita o nível máximo de feromônio em cada trilha, evitando com isso, uma estagnação da solução em um valor ótimo local. A evaporação respeita a equação: $\tau_{ij}^{k} \leftarrow (1-\rho)\,\tau_{ij}^{k}$, $\forall(i,j) \in A_i; \rho \in [0,1]$, sendo ρ um parâmetro da taxa de evaporação de feromônio do ACO.

O algoritmo ACO resume-se em três procedimentos, [Dorigo 2004]: (*i*) construção das soluções com as formigas; (*ii*) atualização de feromônio, e; (*iii*) ações *daemon*. O primeiro procedimento é a construção das soluções pelas formigas com base em um grafo (estrutura formada por vértices e arestas) [Dorigo 2004]. As formigas movem-se entre os componentes do grafo, variáveis do problema, estabelecendo uma conexão entre eles. Este movimento é determinado estocasticamente e localmente pelas informações das trilhas de feromônio e pela informação heurística, η, a qual é utilizada para melhorar a eficiência do algoritmo sendo definida em função das características do problema. O segundo procedimento está relacionado ao depósito ou à evaporação do feromônio durante a construção na busca da solução. Quanto maior a quantidade de feromônio maior a probabilidade de uma mesma conexão ou componente ser usado, reforçando sua trilha. Por outro lado, a diminuição faz com que se busquem novas regiões ainda não consideradas, podendo ser regiões próximas do ótimo. O terceiro procedimento, ações *daemon*, diz respeito a rotinas que venham a melhorar a busca em determinado local, ações de busca local, ou um conjunto de ações globais que possibilitem tomar decisões positivas. O termo *daemon*, que é originário da linguagem computacional, significa uma rotina ou um programa criado para realizar determinada tarefa padrão, com fins específicos, a ser executado.

3. Proposta da Avaliação da Redução de Chaves.

Nossa proposta tem como diretriz a aplicação da ACO (*Ant Colony Optimization*), no método definido por [Coron 2012] como forma de verificação, ratificação e validação das otimizações dos métodos propostos por Gavinho em: [Gavinho 2015] e [Gavinho 2016]. Nesses trabalhos o autor demonstra, por meio da utilização de Algoritmos Genéticos (AG), a redução em cerca de 600 KB no tamanho da Chave Pública de 18 MB proposta e definida por [Coron 2012], mantendo-se, mesmo assim, os mesmos níveis de segurança dos alcançados por [Coron 2012] em seu método. Detalhamos nosso trabalho nesta seção.

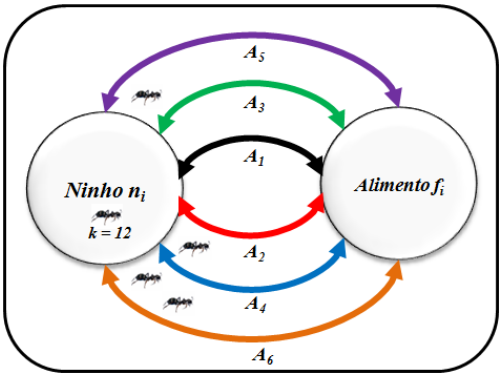


Figura 3. Diagrama formado pelo ninho n_i, fonte de alimento f_i e seis caminhos (A_i)

Em nossa proposta, os esquemas homomórficos de [Gentry 2009], [Dijk 2010], [Coron 2011], [Coron 2012], a otimização por AG de [Gavinho 2015] e [Gavinho 2016], bem como o método artificial de ACO (*Ant Colony Optimization*), foram implementados na ferramenta matemática Matlab/Simulink©. Bem como os três procedimentos de ACO:

```
1  Inicio
2    var i=0, C_i=0, t=0, nr_estagios=0, nr_ciclos=0,
3        τ=0; η_l=0; ρ =0, ω=0, λ=0, γ=0, α=0, τ_l=0, η=0, Θ=0, ρ' = 0,
4        nr_estagios=100, total_Ci=6,
5        Para i=1 ate nr_estagios Faça
6            Processar gerar_ACO(i);
7            Para i=1 ate total_C_i Faça
8                Processar inicializacao(ACO);
9                Processar solucao(p);
10
11                   p_ij^k = (τ_ij)^α (η_j)^β / Σ_{i∈N_i^k} (τ_ij)^α (η_j)^β
12
13               Processar atualizar_feromonio(τ);
14                   τ_ij^k ← τ_ij^k + Δτ^k
15               Processar evaporacao_feromonio(τ);
16                   τ_ij^k ← (1−ρ) τ_ij^k
17               C_i = C_i + 1;
18           FimPara
19       i = i + 1;
20 Fim
21 gerar_ACO ()
```

Figura 4. Pseudocódigo - ACO.

(*i*) Construção das soluções com as formigas → nosso esquema de solução, seguindo o definido por [Dorigo 2004] inicia-se com varredura aleatória de doze (12) (*m*) formigas artificiais (*k*), em um primeiro estágio (E_1), (num total de 30 execuções em 1000 estágios - E_i) partindo do ninho (n_i), com escolha, com mesma probabilidade → p_{ij}^{k} = 16,67%, entre uma das seis arestas (A_i), observados na Figura 3 e seguindo o pseudocódigo da Figura 4.

Cada aresta A_i é definida com o valor do parâmetro λ sendo $x - 2 < \lambda < x + 3$, para cada um dos quatro (04) níveis de segurança de λ (*Toy* → x= 42, *Small* → x= 52, *Medium* → x= 62 e *Large* → x= 72) – tamanhos definidos por [Coron 2011]. (*ii*) atualização de feromônio → Inicialmente todas as arestas A_i possuem a mesma quantidade de feromônio $\tau_{ij}=\tau_{Ai}=0{,}01$. A atualização de feromônio, é dada por: $\tau_{ij}^{k} \leftarrow \tau_{ij}^{k} + \Delta\tau^{k}$, sendo $\tau^{k} = 0{,}001$ [Dorigo 2004]. A evaporação respeita a equação $\tau_{ij}^{k} \leftarrow (1-\rho)\ \tau_{ij}^{k}$ com $\rho \in [0,1]$, que varia randomicamente adaptando-se as perdas "naturais" como "chuva" e "calor" (artificiais) [Castro 2004]. A valoração da taxa de depósito de feromônio α é a de evaporação são diretamente proporcionais aos níveis de segurança semântica e a da blindagem ao ataque do MDC aproximado [Coron 2011], verificações essas testadas a cada iteração do algoritmo.

(*iii*) Ações *daemon* [Dorigo 2004] → os valores da base heurística são associados à segurança ao ataque do aniversário [Gentry 2010], e da mesma forma que o processo para a atualização do feromônio, é realizado o processamento a cada iteração do método.

Tabela 3. Tabela dos resultados obtidos pelas heurísticas: (a) AG; (b) ACO.

$S_{H\text{-}1}$	$S_{H\text{-}2}$	$S_{H\text{-}3}$	$S_{H\text{-}4}$
$T\lambda_{-1} = 0{,}10$ $P_1 = 40$	$T\lambda_{-1} = 0{,}10$ $P_1 = 50$	$T\lambda_{-1} = 0{,}10$ $P_1 = 60$	$T\lambda_{-1} = 0{,}10$ $P_1 = 70$
$T\lambda_{-2} = 0{,}25$ $P_2 = 41$	$T\lambda_{-2} = 0{,}24$ $P_2 = 51$	$T\lambda_{-2} = 0{,}25$ $P_2 = 61$	$T\lambda_{-2} = 0{,}25$ $P_2 = 71$
$T\lambda_{-3} = 0{,}23$ $P_3 = 42$	$T\lambda_{-3} = 0{,}22$ $P_3 = 52$	$T\lambda_{-3} = 0{,}22$ $P_3 = 62$	$T\lambda_{-3} = 0{,}23$ $P_3 = 72$
$T\lambda_{-4} = 0{,}17$ $P_4 = 43$	$T\lambda_{-4} = 0{,}17$ $P_4 = 53$	$T\lambda_{-4} = 0{,}17$ $P_4 = 63$	$T\lambda_{-4} = 0{,}17$ $P_4 = 73$
$T\lambda_{-5} = 0{,}14$ $P_5 = 44$	$T\lambda_{-5} = 0{,}15$ $P_5 = 54$	$T\lambda_{-5} = 0{,}14$ $P_5 = 64$	$T\lambda_{-5} = 0{,}14$ $P_5 = 74$
$T\lambda_{-6} = 0{,}11$ $P_6 = 45$	$T\lambda_{-6} = 0{,}12$ $P_6 = 55$	$T\lambda_{-6} = 0{,}12$ $P_6 = 65$	$T\lambda_{-6} = 0{,}11$ $P_6 = 75$

(a)

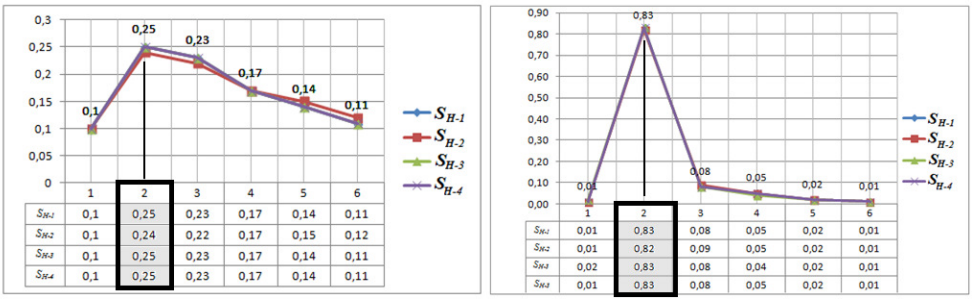
$S_{H\text{-}1}$	$S_{H\text{-}2}$	$S_{H\text{-}3}$	$S_{H\text{-}4}$
$T\lambda_{-1} = 0{,}01$ $P_1 = 40$	$T\lambda_{-1} = 0{,}01$ $P_1 = 50$	$T\lambda_{-1} = 0{,}02$ $P_1 = 60$	$T\lambda_{-1} = 0{,}01$ $P_1 = 70$
$T\lambda_{-2} = 0{,}83$ $P_2 = 41$	$T\lambda_{-2} = 0{,}82$ $P_2 = 51$	$T\lambda_{-2} = 0{,}83$ $P_2 = 61$	$T\lambda_{-2} = 0{,}83$ $P_2 = 71$
$T\lambda_{-3} = 0{,}08$ $P_3 = 42$	$T\lambda_{-3} = 0{,}09$ $P_3 = 52$	$T\lambda_{-3} = 0{,}08$ $P_3 = 62$	$T\lambda_{-3} = 0{,}08$ $P_3 = 72$
$T\lambda_{-4} = 0{,}05$ $P_4 = 43$	$T\lambda_{-4} = 0{,}05$ $P_4 = 53$	$T\lambda_{-4} = 0{,}04$ $P_4 = 63$	$T\lambda_{-4} = 0{,}05$ $P_4 = 73$
$T\lambda_{-5} = 0{,}02$ $P_5 = 44$	$T\lambda_{-5} = 0{,}02$ $P_5 = 54$	$T\lambda_{-5} = 0{,}02$ $P_5 = 64$	$T\lambda_{-5} = 0{,}02$ $P_5 = 74$
$T\lambda_{-6} = 0{,}01$ $P_6 = 45$	$T\lambda_{-6} = 0{,}01$ $P_6 = 55$	$T\lambda_{-6} = 0{,}01$ $P_6 = 65$	$T\lambda_{-6} = 0{,}01$ $P_6 = 75$

(b)

Podemos observar na Tabela 3 os resultados obtidos pelas heurística e meta-heurística em estudo: (a) Valores obtidos pelo algoritmo genético [Gavinho 2016]; (b) Valores obtidos pelo ACO em nossa proposta**.** Os valores dos pesos dados pelo AG, Tabela 3 (a) convergem, em seu valor máximo para $T\lambda_{-2} = 0.25$, e sendo $T\lambda_1 < T\lambda_6 < T\lambda_5 < T\lambda_{-4} < T\lambda_3$. Tendo a convergência do ACO, Tabela 3 (b), direcionando-se ao valor de $T\lambda_2 = 0{,}83$, como percurso de maior concentração de feromônio artificial.

Representamos a Tabela 3 em forma de gráficos com a finalidade de maior visualização dos resultados, observado na Figura 5. A Otimização por Colônia de Formigas selecionou como o "melhor caminho" (melhor solução), exatamente os tamanhos do parâmetro λ iguais a: *Toy* = 41, *Small* = 51, *Medium* = 61 e *Large* =71,

com valoração média de 0.83, Figura 5 (b). Valores esses verificados com as suas atribuições de peso pelo AG, Figura 5 (a).



(a) (b)

Figura 5. Gráficos de convergência: (a) do algoritmo genético; (b) ACO.

Ratificando e validando com isso o trabalho de [Gavinho 2015] onde foi verificado que, com a redução de um bit em cada nível de segurança do parâmetro λ, Tabela 4, (*Toy* = 41, *Small* = 51, *Medium* = 61 e *Large* =71) mantém-se os mesmos níveis de segurança quando comparados aos tamanhos definidos por [Coron 2012], bem como a redução dos tempos de execução do Algoritmo Criptográfico Totalmente Homomórfico.

Tabela 4. Tamanhos reduzidos dos Parâmetros *λ* com Segurança Semântica

Parâmetro	*Toy*	*Small*	*Medium*	*Large*
λ	41	51	61	71

4. Conclusão e Trabalhos Futuros.

Neste trabalho foi utilizada a Otimização por Colônia de Formigas para validar e ratificar as demonstrações das variantes de algoritmos presentes na literatura.Tais variantes utilizaram a heurística dos Algoritmos Genéticos para demonstrar, por meio de otimização e calibração, a redução dos tamanhos das chaves públicas e do tempo de execução das variantes das primitivas de Coron. Mantendo, mesmo assim, a segurança semântica do mecanismo e, alcançando, em consequência disso, a redução dos tempos de execução de todo o processo. Demonstramos que as trilhas de feromônio concentraram-se, em sua maior quantidade, exatamente nos tamanhos de chaves determinados pelo algorítimo em estudo. Como trabalho futuro, procuraremos melhorar a técnica utilizando a Otimização por Colônia de Formigas na proposta de Coron em sua segunda variante, por meio da: definição, calibração e fixação do valor da semente utilizada para a geração de chaves públicas, que no trabalho original de Coron é escolhida por meio de um gerador de números pseudo-aleatórios.

Referências

Boneh, D., Halevi, S., Hamburg, M., et al. “Circular-secure encryption from decision diffie-hellman”. In: Advances in Cryptology–CRYPTO 2008,Springer, 2008.

Buchmann, Johannes A. “Introdução a Criptografia”. Ed. Berkeley, São Paulo, 2002.

Brakerski, Z., Gentry, C., Vaikuntanathan, V. “Fully homomorphic encryption without bootstrapping”, ITCS 2012, 2012.

Coron, J., Naccache, D., Tibouchi, M. Optimization of Fully homomorphic Encryption. Cryptology ePrint Archive, Report 2011/440, 2012.

Castro, L. N. de. Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications. Chapman & Hall/CRC, 2006.

Coron, J., Mandal, A., Naccache, D., et al. "Fully homomorphic encryption over the integers with shorter public keys", Advances in Cryptology–, pp. 487–504, 2011.

CSA Security Guidance for Critical Areas of Focus in Cloud Computing –v2.1. Cloud Security Alliance, 2009.

Daniel J. Bernstein. Introduction to post-quantum cryptography. In Post-Quantum Cryptography. 2009.

Dijk., M. Van, Gentry, C., Halevi, S. e Vaikuntanathan, V., Fully homomorphic encryption over the integers. *In H. Gilbert* (Ed.), EUROCRYPT 2010.

Dorigo, M.; Gambardella, L. M., Ant Colony System: A cooperative learning approach to the traveling salesman problem. IEEE TEC 1997.

Dorigo, M.; Stutzle, T. Ant Colony Optimization. Massachusetts Institute of Technology. Cambridge, 2004.

Gavinho, Joffre Filho; Micelli, C; Pereira, G. "Compressão e Otimização de Chaves Públicas usando Algoritmo Genético em Criptografia Completamente Homomórfica"- XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSEG 2015 - Florianópolis/SC, Brasil - Novembro 09-12, 2015.

Gavinho, Joffre Filho; Micelli, C; Pereira, G. " A Public Key Compression Method for Fully Homomorphic Encryption using Genetic Algorithms!" 19th International Conference on Information Fusion – Heildeberg - Alemanha - 2016.

Gentry, C. "Fully homomorphic encryption using ideal lattices". In: Proceedings of the 41st annual ACM Symposium on Theory of computing, pp. 169–178. ACM, 2009.

Keeney, R.L. & RAIFFA, H. Decisions with Multiple Objectives: Preferences and Value Tradeoffs , John Wiley & Sons, Nova Iorque, 1976.

Lacerda, E.G.M e Carvalho, A.C.P.L. "Introdução aos Algoritmos Genéticos", In: Sistemas inteligentes: aplicações a recursos hídricos e ciências ambientais. Editado por Galvão, C.O., Valença, M.J.S. Ed. Universidade/UFRGS: ABRH, 1999.

Michael O. Rabin. Probabilistic algorithm for testing primality. Journal of Number Theory , 12(1):128 – 138, 1980.

Morris, Christopher , "Navy Ultra's Poor Relations", in Hinsley, F.H.; Stripp, Alan, *Codebreakers: The inside story of Bletchley Park*, : Oxford University Press, 1993

NIST- National institute of standards and technology. Cyber security Framework Development Overview.NIST's Role in Implementing Executive Order 7213636, Improving Critical Infrastructure Cybersecurity, Presentation to ISPAB, 2013.

Mukherjee P, Wichs D. Two round MPC from LWE via multi-key FHE. IACR Cryptology ePrint Archive , 2015. To appear in Proceedings of Eurocrypt 2016.

Rivest R. L., L Adleman, and M L Dertouzos. On data banks and privacy homomorphisms, in r. a. demillo et al. In Eds.), FSC. Academic Press, 1978.

Smart, N.; Vercauteren, F. Fully homomorphic encryption with relatively small key and ciphertext sizes. Cryptology ePrint Archive, Report 2009/571, 2009.

Stalling, Willian, Criptografia e Segurança de Redes: Princípios e Práticas 4. Ed. Prentice Hall Brasil, pag 17-36, 2007.

Sousa, F. R. C.; Moreira, L. O.; Machado, J. C. Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios. Fortaleza, 2009.

William S. SAGE: A Computer System for Algebra and Geometry Experimentation, 2012.

A Non-Probabilistic Time-Storage Trade-off for Unsalted Hashes

Frederico Schardong[1], **Daniel Formolo**[2]

[1] Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brazil

[2]Universidade do Vale do Rio dos Sinos (UNISINOS)
São Leopoldo - RS - Brazil

fschardong@inf.ufrgs.br, danielformolo@unisinos.br

Abstract. *This work proposes a new cryptanalytic non-probabilistic trade-off for unsalted hashes. It presents the main cryptanalytic trade-offs, making a comparison with the proposed method. Although the number of hash operations to recover an element is high compared with the traditional methods, the new method has the advantage of guaranteed success on the recovery of hashes, minimal and sequential disk read operations, unlike the existing probabilistic trade-offs.*

1. Introduction

Hash functions are widely used to protect sensitive information such as passwords. Unlike symmetric cryptography where a key is needed to recover the content encrypted by the same key, hash functions are not reversible. This fundamental difference allows the result of a hash function to be stored in an insecure environment, provided that the hash function is considered safe, i.e., does not possess known attacks or flaws. However, if the hash function's input is considered predictable, an attacker that eventually obtain such hash can compare it to hashes that she possesses and know their original content. If any hash matches then the attacker can use it against the target system and easily pretend to be the legitimate user, if the system has no additional authentication mechanism [Stalings 2005].

Concatenating either fixed or random data to the information being hashed before submitting to the hash function provides additional entropy as an attacker that possesses any hash will have to generate the salt value in addition to the password itself. Although salting can mostly overcome trade-off attacks, this technique is often forgotten by developers and analysts. Hash functions such as bcrypt [Provos and Mazieres 1999] and scrypt [Percival and Josefsson 2015] invalidate brute-force and trade-off attacks by internally using salt and large amounts of memory and processing power to compute a single hash. Even though solutions for trade-off attacks are known to the cryptanalytic community for a long time, many developers and analysts are not aware of them and use insecure hash functions with no extra care.

Using a single hash function to protect user passwords might compromise the security of cryptosystems. Such implementations are vulnerable to many attacks: brute-force, dictionary of passwords/words and more recently rainbow table. Both dictionary and rainbow table attacks require some pre-computation so the attack can be applied: a list of words need to be generated and either their hashes calculated or the rainbow table

computed. Currently, the processing power and capacity of storage are large. Even so, these resources are heavily used by attacks, so the balance in the usage of these resources adapting to situations and hardware conditions are essential for breaking cryptographic keys in an acceptable time. This paper presents a new cryptanalytic non-probabilistic trade-off and compares its results to the two most significant cryptanalytic trade-offs. The next section provides a brief description of the traditional methods. Section 3 describes Hellman's tables, the first method to be compared, while Section 4 presents rainbow table, the second method to be compared. Section 5 describes the delta encoding technique and Section 6 the proposed algorithm. Section 7 presents the results of experiments and performance comparison with the other two existing algorithms. Finally, section 8 concludes the study.

2. Traditional Attacks

The two main methods for breaking hashes are brute force and dictionary. Both attacks aim to discover the input value applied to a known hash function $H(x)$, which generated a hash h, also known. Each of the two approaches has their characteristics regarding the use of computational resources. To better understand these features and how traditional and modern methods balance the use of processing and memory the concept of trade-off was created.

[Hellman 1980] defines trade-off as: "if there are N possible solutions to search over, the time-memory trade-off allows the solution to be found in T operations (time) with M words of memory, provided the time-memory product TM equals N". Hellman's trade-off definition allows two extremes: $T = 1$ and $M = N$, or $M = 1$ and $T = N$. Although only one hash operation is needed in the former, the amount of memory required is equal to the dictionary attack (N). The latter case defines the brute force attack, where almost no memory is used ($M = 1$) and the number of hash calculations is equal to the number of possible solutions ($T = N$).

Given a hash h, one wants to discover the plaintext which generated h. Through the brute force attack, a set of plain words is chosen then one or more computers strive to produce this set and simultaneously apply the same hash function $H(x)$ to each word. Each hash generated by the function $H(x)$ is compared with the hash h provided, if they are equal then the plaintext that generated the hash h is the same plaintext that generated the hash $H(x)$ (ignoring the low probability of a collision). If no generated hash is equal to the hash h provided, then the plaintext that resulted in h is not part of the set of plain words chosen at the start of algorithm execution.

The main difference between dictionary and brute force attacks is that hashes are stored along with their respective plaintext in the dictionary attack. This means that the elements from the set of plain words are submitted to the hash function $H(x)$ only once. This dictionary creation process is called pre-processing or pre-computation, and its cost is usually negligible as it is executed only once. After its creation, the dictionary is sorted by hashes so that it is possible to find any element in up to $log_2(N)$ operations, where N is the number of elements in the dictionary. [Sleator and Tarjan 1985].

3. Hellman's Method

To demonstrate that valuable information should not be entrusted to systems with short keys, [Hellman 1980] proposed a trade-off able to reverse an encrypted element of a system with N elements using $N^{2/3}$ operations and $N^{2/3}$ words of memory. At the time it was created, Hellman was able to reduce the cost of breaking DES on block mode from $5'000$ USD dollars using brute force to 10 USD dollars using his method (not considering the cost of the precomputation required to create the tables).

Given a set of N elements to be stored in this system, m elements are randomly selected from the set $\{1, 2, ..., N\}$, creating starting points $SP_1, SP_2, ..., SP_m$ obtaining:

$$X_{i0} = SP_i \qquad 1 \leq i \leq m \tag{1}$$

Then computes:

$$X_{ij} = f(X_{ij-1}) \qquad 1 \leq j \leq t \tag{2}$$

t being the number of rounds that the function $f(x)$ is executed. After t executions of function $f(x)$ the last element of a column i will be:

$$EP_i = f^t(SP_i) \tag{3}$$

Function $f(x)$ is the result of some reduction function $r(x)$ applied to an encrypted element, thus $f(x) = r(H(x))$. The reduction function $r(x)$ must reduce the output of a hash function or symmetric encryption in such a way that the result is a random element of the same set N.

To reduce the amount of memory all the intermediate points are removed and $\{SP_i, EP_i\}_{i=1}^{m}$ are organized by the end points (EP) then stored in a table. Multiple tables can be created where each table must use a different reduction function to avoid collisions.

Given an encrypted element or some hash K to be searched, $Y_1 = r(K)$ is computed and searched in the end points of the table. If it is not found, it means that the plaintext which generated K is no $X_{i,t}$. So, $Y_2 = f(Y_1)$ is computed and searched in the table, if not found then the plaintext is not in position $X_{i,t-1}$. Repeating the previous step, $Y_3 = f(Y_2)$ is computed and another search is performed on the table. This process is repeated until some EP_i is found or until $X_{i,0}$ is reached.

Assuming EP_i is equal to Y_3, then the plaintext that generated K is at the position $X_{i,t-2}$. However, since all the intermediate points were discarded, the entire line (starting on SP_i) needs to be recreated, so the element in position $X_{i,t-2}$ can be found and encrypted to test if it is equal to K. If different, a false alarm has been found and the algorithm execution continues. A false alarm is a consequence of the probabilistic nature of this method. As the result of encryption or hash function is shortened, the probability of false alarms (collisions) may be significant depending on the settings used.

Hellman states that if the function f is modeled as a function that randomly maps elements of N to itself then the probability of success of a table is limited by:

$$P(S) \geq (1/N) \sum_{i=1}^{m} \sum_{j=0}^{t-1} [(N - it)/N]^{j+1} \tag{4}$$

The mathematical proof of this theorem is found on Hellman's paper [Hellman 1980]. He also states that if $mt^2 = N$ then the resolution of equation 4 results in a probability of success of at least 80%, and the expected number of false alarms per table is:

$$E(F) \leq mt(t+1)/2N \tag{5}$$

The use of Hellman's method to break the 56-bit Data Encryption Standard, or DES, is less complex than applying brute force on a system with 38-bit key [Hellman 1980].

4. Rainbow Table

[Oechslin 2003] proposes to change Hellman's table structure, making it use different reduction functions rather than just one. In Hellman's method each shortening round uses the same function $r(x)$, while in rainbow table each round uses a shortening function $r_i(x)$ different. First round always uses shortening function $r_1(x)$, the second $r_2(x)$ and so on until the last round t use the reduction function $r_{t-1}(x)$.

To recover a value K from a rainbow table, $r_{t-1}(K)$ is applied first. If the result is not present in the end points of the table then $r_{t-2}(f_{t-1}(K))$ is applied, if this result is not present in the end points of the table then the next step is to apply $r_{t-3}(f_{t-2}(f_{t-1}(K)))$ and so on. Retrieving an element in a rainbow table need at most $t(t^2 - 1)/2$ operations whereas searching on the original table need at most t^2 operations, considering that both tables have t columns.

Oechslin created $4'666$ Hellman's tables with dimensions of $m_h = 8'192$, $t_h = 4'666$ and a rainbow table with dimensions $t_r = 4'666$, $m = m_h * t_h = 38'223'872$ both using the same set $N = 2^{37}$ of 8 byte hashes. In his implementation the probability of success was measured by searching 500 random hashes on each scheme and calculated to be $78,8\%$ on the rainbow table and $75,8\%$ on Hellman's method. Search for elements in rainbow tables was 7 times faster with an average of $9,3$ million hash calculations, against $67,2$ million on Hellman's tables.

[Zhang et al. 2010] propose to divide rainbow table into smaller pieces and distribute them among computers in a client-server relationship such as each customer receives a different part of the table and can contribute to search keys, turning the problem from two dimensions $\{M, T\}$ into a three-dimensional problem $\{M, T, R\}$. The protocol that coordinates the work between each client and the server consists of synchronous calls that do not represent a significant cost. Implementation confirms the theoretical analysis made by the authors that attack time decreases linearly with the increase of clients (resources).

5. Delta Encoding

Delta encoding is a data compression technique that consists in calculating the difference between two objects, whether they are files, numbers, strings or something else. There are

numerous applications that benefit from this simple technique (complexity $O(n)$) such as incremental backup [Burns and Long 1997], deduplication systems [Paulo and Pereira 2014] and HTTP frame compression [Mogul et al. 1997].

According to [Mogul et al. 1997] video compression standard MPEG-1 already used the technique of calculating the difference between objects. The MPEG-1 occasionally sends an entirely new frame, and next frames are just information to reconstruct what has changed compared to previous frames.

Naturally the smaller difference between the data, the higher is the performance of the delta encoding. For example, the set $\alpha = \{2, 5, 7, 4, -3, 8, 23, 9, 4\}$ results in $\alpha_d = \{2, 3, 2, -3, -7, 11, 15, -14, -5\}$ such as $\alpha_d = D(\alpha)$ where $D(x)$ is a function that calculates the delta encoding of x. The α set needs at least 54 bits to be stored as the largest element (23) occupies 6 bits (5 bits for the number plus the sign indicator), so $9 * 6 = 54$ bits. On the other hand, the delta encoded set (α_d) needs only 45 bits.

6. Proposed Algorithm

In this section, the operation, benefits, and limitations of the proposed algorithm are described in detail. This algorithm consists of applying a round of delta encoding then sorting the resulting elements to remove all duplicate. Through this technique, a new trade-off is created as more duplicate elements are removed, more space is saved, however, more processing is required to recover the original set. The resulting ordered set may undergo another round of delta encoding or other compression technique chosen by the analyst as, for example, bitmap [Lemire et al. 2010].

6.1. Delta Encoding of Complexity $O(n^2)$

Given the set $\alpha = \{3, 12, 20, 21, 30\}$ the result of $\alpha_d = D(\alpha)$ is the set $\{3, 9, 8, 1, 9\}$. As explained earlier, it is possible to reassemble α with complexity $O(n)$. Ordering α_d in ascending order through any $S(x)$ function results in $\alpha_s = S(\alpha_d)$ such as $\alpha_s = \{1, 3, 8, 9, 9\}$. Removing duplicate numbers from α_s results in $\beta = \{1, 3, 8, 9\}$ which can again be subjected to the delta encoding function $D(\{1, 3, 8, 9\})$ creating the set $\beta_d = \{1, 2, 5, 1\}$. Set β_d must start with the first element of α, resulting in $\beta_d = \{3, 1, 2, 5, 1\}$. Set β_d can be saved in only 15 bits as each element occupies 3 bits while the original set needs 25 bits, disregarding any compression algorithm.

Recovery of set β from β_d can be performed with complexity $O(n)$, requiring 4 operations (ignoring the first element of β_d as it is the first element of α), obtaining $\beta = \{1, 3, 8, 9\}$. After adding the first element of β_d to β it is obtained $\beta = \{3, 1, 3, 8, 9\}$. Recovery of set α consists in a problem of higher complexity. One way to recover α would be to select the first element of β, which is the first element of α, iterate the set β and at each iteration add the first element of α to each element of β. Thus, in some interaction, the second element of α is found. When the next element of α is found then the search resumes, summing all elements of β to the last element of the set α found. Therefore, all elements of the set α will be found, and the set will be completely recovered.

However, there must be a way to validate whether an element of α was found or not. Evidently, such a method should not be based on the elements of the original set but on some heuristic.

One solution to this problem is to create a function $m(x)$ capable of determining whether an element belongs to α or not. Theorem 1 details the operation of the algorithm using such function $m(x)$, below the theorem, there is a proof confirming that the method described can fully recover α.

Theorem 1 *Given a set α of ordered elements and applying the reduction operations $D()$, ordering $S()$, removal and reduction $D()$, in that order, it is possible to retrieve the original α set if, after the last operation the set β_d contains, in addition to its elements, the first element of the original set α.*

Axiom 1 *Applying the functions sorting and removal to any set α_d, the resulting set β maintains a representative of each element of the set α_d and the first element of the set α.*

Axiom 2 *Given the first element of the set β, the other elements of this set are recoverable from β_d in a simple and obvious way.*

Proof.

1. Base: The element α_1 is in the set β.
2. Step: Based on axioms 1 and 2, given that α_n is known, $\alpha_{n+1} = \alpha_n + k$, being $\alpha_n + k$ the lowest element resulting from $m(\alpha_n + k) = true$, for all $k \in \beta$.
3. Restriction: The set α is sorted and does not have duplicate elements.

Creating a function $m(x)$ capable of determining whether an element is part of the set α or not can be tough. One possible way of solving this problem is separating the elements of the original set α into smaller subsets through some heuristic, such as a function that maps each element to some subset at random. Such $m(x)$ function should always map an element x for a subset α_a, never to another.

Considering that the function $m(x)$ maps elements of a set $\alpha = \{3, 12, 20, 21, 30\}$ into 2 subsets $\alpha_a = \{3, 12, 21\}$ and $\alpha_b = \{20, 30\}$. Applying the algorithm described above in the subset α_a it is obtained $\alpha_{ad} = \{3, 9, 9\}$, $\alpha_{as} = \{3, 9, 9\}$, $\beta_a = \{3, 9\}$ and finally $\beta_{ad} = \{3, 6\}$. Note that the first element of β_{ad} is equal to the first element of α_a, thus it has not been duplicated.

As noted earlier, it is trivial to recover β_a, but the recovery of α_a requires some heuristic defined in the function $m(x)$. The first element of β_a is always the first element of α_a, sum up the last element of the original set found (in the first round is $\alpha_a[0]$) with first of the list, 3, obtaining 6. Applying $m(6)$ it turns out that 6 is not part of the set α_a. The last element of the original set found is then added to the second element of the set β_a, obtaining 12. Applying $m(12)$ returns the subset a, indicating that 12 belongs to the original set α_a. As a new element of the original set was found, all elements of β_a must be added again to the last element of the set α_a found (12) so the third element of the original set can be found. This procedure must be repeated until all elements of the set α_a are recovered.

Disregarding the complexity of function $m(x)$ it is apparent that the complexity of the algorithm which recovers the subsets created from the result of $m(x)$ is $O(n^2)$. The complexity of the generation of subsets is negligible because they are created only once, while the recovery is a task that tends to be repeated multiple times.

6.1.1. Considerations regarding the Delta Encoding of Complexity $O(n^2)$

The most important part of the delta encoding algorithm of complexity $O(n^2)$ is the heuristic function that determines the distribution of elements of a set into smaller subsets. This function should be as close as possible to a random function.

Only ordered sets may be used in this algorithm. Another limitation is that it is unable to retrieve sets with duplicate numbers because through the recovery process it is not possible to determine how many times an element exists in the original set, thereby any duplicate element will be lost.

6.2. Cryptanalytic Non-Probabilistic Trade-off

Using the delta encoding of complexity $O(n^2)$ algorithm described in the previous section, one can represent numerical sets with many elements in a compact form at the cost of using more processing power. However, only the proposed algorithm can not be considered a trade-off because it is not possible to vary between memory and processing.

One way to make it a trade-off is to change the function $m(x)$ such that a given set of elements can be divided into subsets of different size. In this way, more subsets result in subsets with fewer elements and consequently little processing and lots of memory needed to store them (tending to $M = N$ and $T = 1$). However, fewer subsets result in subsets with more elements, namely very little memory but considerable processing (tending to $T = N$ and $M = 1$). Few subsets lead to low memory use because applying the first step of the algorithm previously described results in many duplicate deltas, which are later removed. On the other hand, many subsets with few elements result in sparse subsets, resulting in less duplicate deltas. Consequently, the total space of the tables will be higher. One can associate the subsets of this method with the tables of Hellman's method.

Since the goal of this algorithm is to store passwords and their hashes, the order of elements is irrelevant. This makes it possible to use delta encoding of complexity $O(n^2)$. Consider α as a set of passwords to be stored. This set must first be sorted and then submitted to the delta encoding algorithm of complexity $O(n^2)$, where $m(x)$ is a reduction function $R(x)$ of the hash function $H(x)$ to be reversed.

It is important to reduce the output of the function $H(x)$ through some reduction function $r(x)$ so that it produces an amount of subsets that satisfies the desired trade-off. For example, a 128-bit hash function would result in 2^{128} subsets if no reduction is used. If the number of elements in the set α is less than 2^{128} then the trade-off would be a dictionary attack ($M = N$ and $T = 1$).

The subsets of α must be stored along with the identification of which hash and reduction algorithms were used as function $m(x)$.

After all subsets are generated and stored, it is trivial to recover some password only providing its hash, considering that it is in the initial set α. To reverse any hash h, one has to reduce it to find the subset where its plaintext is stored. After identifying the subset apply the recovery algorithm proposed in the previous section using $r(H(x))$ as $m(x)$. When some element i belonging to the subset $m_{r(h)}$ is found then compare $H(i) = h$, if it is true then i is the plaintext of the hash h produced by function $H(x)$. If the plaintext i

Table 1. Comparison of a set α using a reduction of MD5 as $m(x)$

Plaintext	$H(x)$	$r(H(x))$
1	**c4ca4238a0b923820dcc509a6f75849b**	**c**
2	**c81e728d9d4c2f636f067f89cc14862c**	**c**
3	eccbc87e4b5ce2fe28308fd9f2a7baf3	e
4	a87ff679a2f3e71d9181a67b7542122c	a
5	e4da3b7fbbce2345d7772b0674a318d5	e
6	1679091c5a880faf6fb5e6087eb1b2dc	1
7	8f14e45fceea167a5a36dedd4bea2543	8
8	**c9f0f895fb98ab9159f51fd0297e236d**	**c**
9	45c48cce2e2d7fbdea1afc51c7c6ad26	4
10	d3d9446802a44259755d38e6d163e820	d
11	6512bd43d9caa6e02c990b0a82652dca	6
12	**c20ad4d76fe97759aa27a0c99bff6710**	**c**

is part of the set α then it is guaranteed to be recovered.

The algorithm described is exemplified in Table 1. The Plaintext column represents the set α to be stored, the function $H(x)$ is MD5 [Rivest 1992] and the reduction function $r(x)$ reduces the output of $H(x)$ to its 4 more significant bits or 1 hex character as shown in the third column. In this example, 4 of the 12 elements of the set α were reduced to the address c: $1, 2, 8, 12$ which after being subjected to delta encoding algorithm of complexity $O(n^2)$ result in: $1, 4, 6$. The address c is composed of the elements $1, 4, 6$.

As a result, the algorithm generates a table relating each $r(H(x))$ to their respective set of elements. In the example of Table 1, $r(H(x)) = c$ points to set $1, 4, 6$. This relationship allows to recover the plaintexts $1, 2, 8, 12$. From this table and a hash y intercepted, a cryptanalyst can recover the plaintext related to y applying $r(y)$ and accessing the list of values encoded with delta encoding algorithm of complexity $O(n^2)$.

As $m(x)$ is a hash function used to separate the set α in subsets then the elements shall be part of an arithmetic progression. If the distance between the elements is not constant, the recovery phase may mistakenly create elements that the hash function will consider as part of the subset being recovered. Despite this limitation, hash functions satisfy the conditions imposed on how the function $m(x)$ should work. Notwithstanding, this restriction is not problematic for this type of application as a cryptanalyst will typically use it with large data sets, often encompassing all possibilities. Such combination would be an arithmetic progression with 1 bit interval between each element.

Considering that previously to the second round of delta encoding the elements are unique and sorted, other compression techniques can be used instead of the second series of delta encoding. A straightforward and efficient technique when the elements are not far between and not repeated is the bitmap [Lemire et al. 2010]. In a bitmap numbers are saved in the respective bits, for example, the sequence $1, 4, 7, 8$ would be saved as 010010011 where each bit set to 1 represents a number from the sequence. In this case,

the first bit representing the number 0 is set to 0, i.e., the number 0 isn't part of the sequence. However, the second bit, which represents the number 1 is set, i.e., 1 is part of the sequence. The same goes for the next bits, as only the fifth, eighth and ninth bit are set then only these numbers are part of the sequence. Using this technique only 9 bits were needed to represent a sequence of 4 elements. If the sequence ended with an enormous number, for example, $1'500$ instead of 8 then it would take $1'501$ bits to represent them. In such case, the bitmap is clearly not advantageous.

In this study, both delta encoding and bitmap were used as the last round of the delta encoding of complexity $O(n^2)$. When few subsets were used the bitmap technique was more efficient.

6.3. Theoretical Analysis

As explained earlier, the proposed delta encoding algorithm that recovers α from β has temporal complexity of $O(n^2)$. The pseudo-algorithm 1 deepens even further how the recovery of α from β works.

As pointed out by Oechslin, the number of iterations to recover an element in a rainbow table is at most $t(t^2-1)/2$, while at most t^2 operations are needed to retrieve an element on Hellman's method [Oechslin 2003], i.e. both have temporal complexity $O(n^2)$.

Algorithm 1 Pseudo-algorithm to recover $alpha$ from $beta$

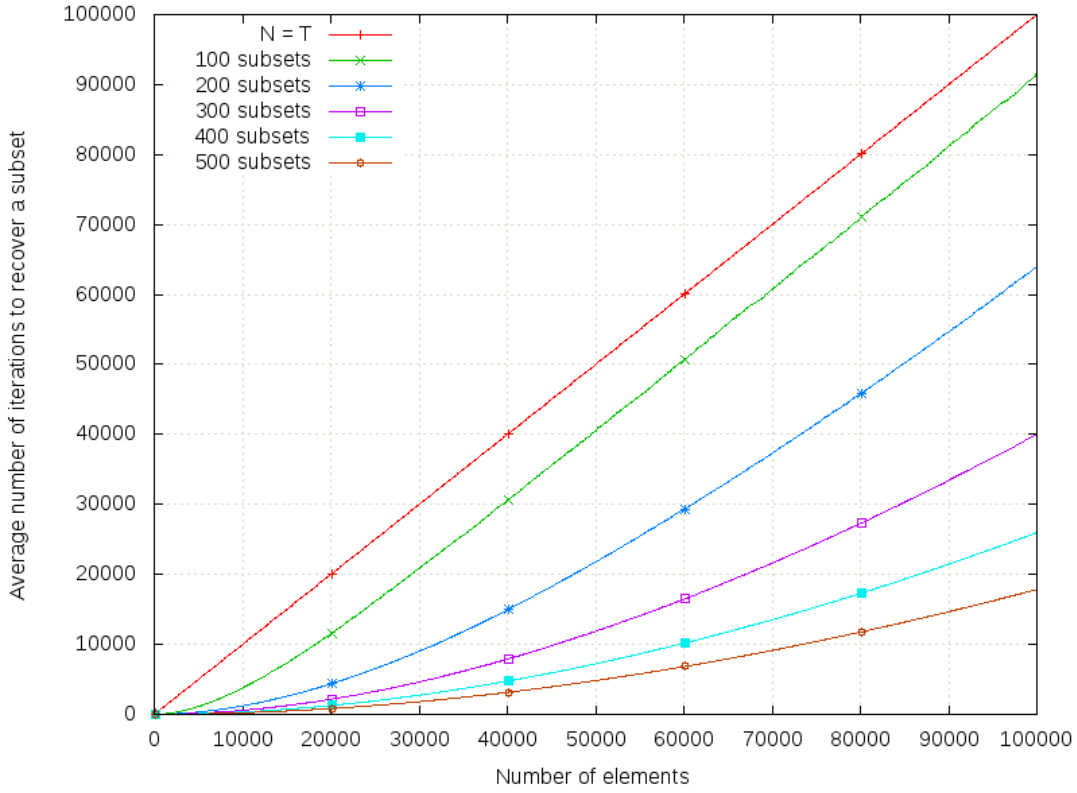
```
1:  procedure RECOVER α(β)
2:      α[0] ← β[0]
3:      address ← m(α[0])
4:      i ← 1
5:      for all a in β do
6:          for all b in β do
7:              if address = m(a + b)  &  (a + b) > α[i − 1] then
8:                  α[i] ← a + b
9:                  i ← i + 1
10:                 break
11:             end if
12:         end for
13:     end for
14:     return α
15: end procedure
```

Disconsidering the complexity of the function $m(x)$ it is evident that the complexity of the pseudo-algorithm 1 is $O(n^2)$. The complexity of the function $m(x)$ is not considered in the analysis because it can be implemented in several ways. If using a hash function as implemented in this study, its complexity ($O(1)$) is negligible [Sabharwal and Bratia 1997].

7. Experimental Results and Comparisons

To analyze the spatial and temporal behavior of the algorithm, numerical sets ranging between $1'000$ and $100'000$ elements were generated and applied to the cryptanalytic non-

Figure 1. Average number of iterations to recover a subset



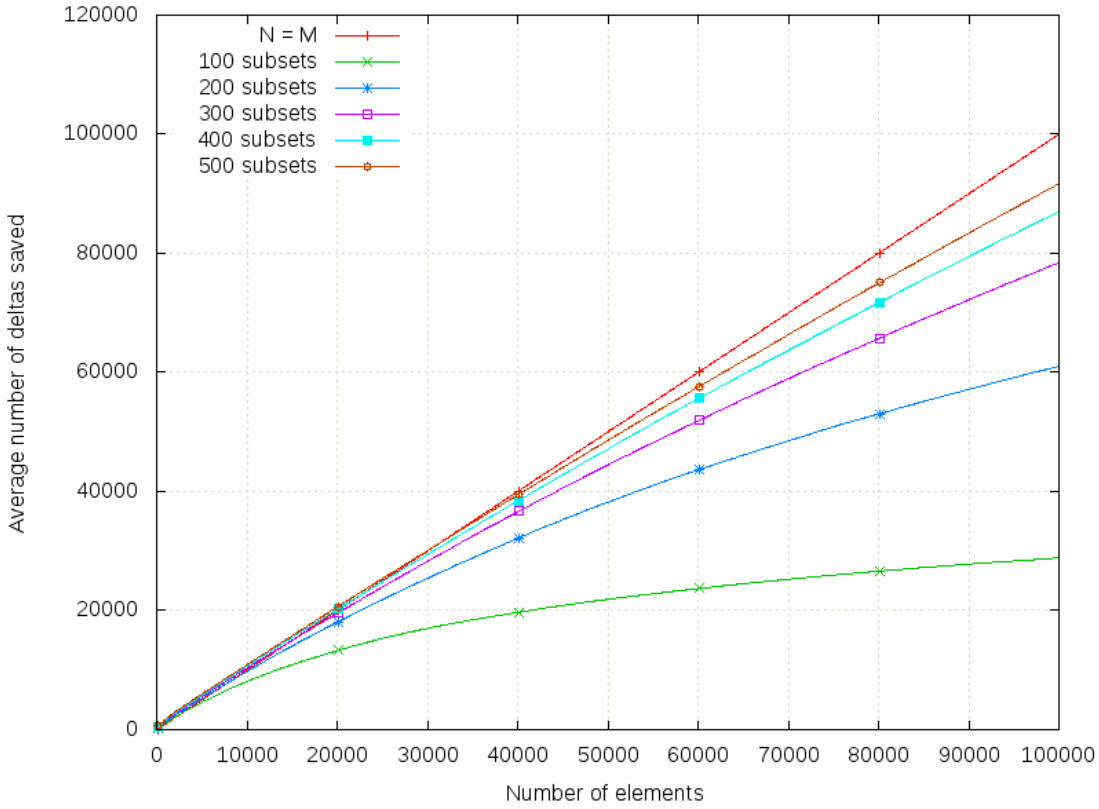
probabilistic trade-off using multiple subsets (100, 200, 300, 400 and 500). The input data generated by the experiments are irrelevant; any random sequence gives the same results. The graphs of Figures 1 and 2 aim to demonstrate the spatial and temporal behavior of the algorithm, respectively. Both graphs represent the same data sets and subsets. The X axis of both graphs represent the total number of elements while the lines represent each subset. Lines $N = T$ and $N = M$, however, do not represent any subset but the extremes of the trade-off for informational purposes only. The Y axis of the graph of Figure 1 is the average number of iterations (i.e. hash operations) to recover a subset while Y axis of the graph of Figure 2 represents the average number of deltas saved to disk.

Figure 1 shows that dividing the elements in many subsets (300, 400 and 500) drastically reduces the required number of hash operations (Y-axis) to retrieve the elements of a subset. The fewer subsets are used, i.e., the more elements there are in a subset, more hash operations will be needed to recover it, approximating the trade-off to $N = T$.

When fewer subsets are used, greater is the number of elements in each subset. Consequently, the first round of delta encoding result in more duplicate elements than using more subsets (and therefore fewer elements per subset). Since duplicate deltas are later removed, less memory is required to save all subsets as evidenced in graph of Figure 2. However, if many subsets are used, then few duplicate deltas will be generated causing more deltas to be written to disk, approximating the trade-off to $N = M$.

It is possible to decrease the size of saved deltas through a data compression algorithm. The ZPAQ algorithm [Mahoney 2013] was able to reduce the size of subsets

Figure 2. Average number of deltas saved in disk



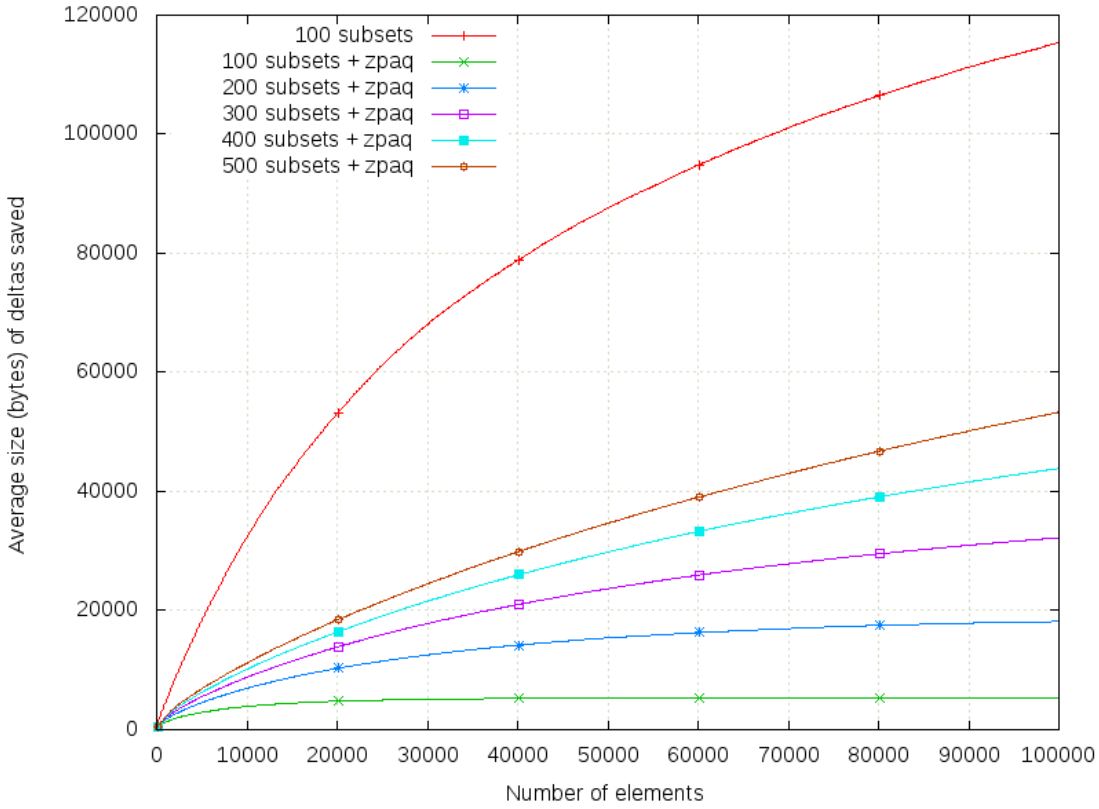
such that compressed deltas occupy less space than when arranged in 100 subsets without compression, as can be seen in the graph of Figure 3.

Oechslin created $23'330$ Hellman's tables with dimensions of $m = 7'501$ lines and $t = 4'666$ columns and 5 rainbow tables with dimensions $t_r = 4'666$ rows and $m = 35'000'000$ columns both using the same set N of 2^{37} different hashes of 8 bytes each. The probability of finding a hash of this group was estimated at 99.9% both in rainbow table and in Hellman's tables. Oechslin recovered 500 random elements in the rainbow table and found out that it took on average of 7.4 million hash operations to recover an element versus 90.3 million using original tables. The two sets of tables were designed to occupy 1.4 GB each.

Following the criterion of using at most 1.4 GB, tables of the proposed algorithm were generated to cover the same amount of 2^{37} different hashes of 8 bytes each. 27 thousand subsets were generated, and 500 elements were randomly recovered using on average 4 billion hash operations to retrieve each element. If compression is available, as discussed previously, the number of subsets to fulfill 1.4 GB would be 60 thousand subsets. In this scenario each of the 500 random numbers need on average 3.5 billion hash operations to be recovered.

Unlike probabilistic methods in which the elements are distributed randomly among all lines of all tables, the proposed algorithm randomly distributes the elements between the tables progressively and steadily, starting from the first element to the last. Resulting in each table being composed of random elements arranged in ascending order, thus

Figure 3. Average size of subsets after being compressed with ZPAQ



recovering an element will depend on its position in the table. If an element is at the beginning of a table few hash operations are required to retrieve it, but if it is at the end of the table many hash operations will be necessary.

In the scenario of 27 thousand tables, the number of saved elements in each subset is on average equivalent to 3.09% of the total elements before applying the delta encoding algorithm of complexity $O(n^2)$. On the other hand, the scenario of 60 thousand tables they were equivalent to on average 11.03%. Thus, with less duplicate elements less hash operations were required to recover each element. In both 27 and 60 thousand tables scenarios, the last delta encoding operation was replaced by bitmap because on average all sets possessed elements close enough and in sufficient quantity for the technique to be more advantageous than the second round of delta encoding. Only when using more than about 550 thousand subsets the second delta encoding was more efficient than the bitmap technique.

Table 2 summarizes this comparison, showing that although the average number of hash operations required by the proposed method is far superior to Hellman's and Oechslin's methods, the algorithm needs to make fewer read operations on disk to retrieve an element since only the subset to which it belongs must be loaded from disk into RAM at the beginning of execution. Meanwhile, Hellman's and Oechslin's methods need to either load all the tables at once or do random searches on disk when applying the reduction functions to the hash being recovered as it is necessary to search in all lines of all tables based on the result of the reductions. Depending on the architecture in which attacks are

Table 2. Comparison between the Hellman's and Oechslin's methods with the proposed algorithm

	Hellman	Rainbow Table	Proposed Method	Proposed Method with Compression
Input Range	2^{37}	2^{37}	2^{37}	2^{37}
Size	1.4 GB	1.4 GB	1.4 GB	1.4 GB
Hash Operations	90.3 M	7.4 M	4 B	3.5 B
Advantages	• Few hash operations		• Little disk access • Disk access is serial • Success is guaranteed • Easily parallelizable	

conducted and settings chosen for the trade-off, the average execution time of probabilistic methods quoted may be higher than the proposed method.

8. Conclusion

Hash functions are widely used to store passwords securely, but the simple application of a hash function may not be enough to protect them. Different attacks can be applied in hashes such as brute force, where passwords are generated, their hashes calculated and compared with the hashes being reverted. If any hash matches, the password that originated this hash is returned. Another possibility is to pre-compute a range of passwords and save them in a structure called dictionary and then search in this database for hashes to be reverted. In brute force attack nothing is stored, all hashes are re-calculated each time the attack is executed. In the dictionary attack, the password range is calculated and hashed only once, without the need to perform repeated hash operations when an attack is applied. On the other hand, hashes are fully saved to disk, consuming a lot of disk space. The brute force attack is one extreme of the trade-off: $M = 1$ and $N = T$ while the dictionary attack is the other extreme: $M = T$ and $N = 1$.

Hellman proposed a probabilistic method able to recover N elements with $N^{2/3}$ operations using $N^{2/3}$ memory words. Oechslin changed Hellman's method to use a different reduction function for each column of the table and obtained practical results of about 12 times fewer hash operations to retrieve an element in comparison to Hellman's method. Unlike Hellman's and Oechslin's methods, the algorithm proposed in this paper is not probabilistic, ensuring that any element of the range used will be found. This algorithm uses the delta encoding technique of complexity $O(n^2)$, also proposed and analyzed in this paper.

Despite the fact that the average number of hash operations required to retrieve each element of the proposed algorithm is much higher than the results obtained by Hellman and Ochslin, the proposed method only requires to read the subset on disk on which the hash being sought belongs, while Hellman's and Oechslin's methods need to do random reads on their tables. In practice, the high number of hash operations required by the proposed method is somewhat offset by the low amount of disk access. The gain obtained

with low disk access relies heavily on the type of hardware where the algorithm runs, so future studies are needed to evaluate whether in practice the proposed algorithm is more efficient at run time than current methods and which hardware architectures it can obtain speed advantages. In conclusion, the proposed algorithm has advantages over the current methods but also has disadvantages that can be circumvented with future improvements, demonstrating that it has potential for use if improved.

References

Burns, R. C. and Long, D. D. (1997). Efficient distributed backup with delta compression. In *Proceedings of the fifth workshop on I/O in parallel and distributed systems*, pages 27–36. ACM.

Hellman, M. E. (1980). A cryptanalytic time-memory trade-off. *Information Theory, IEEE Transactions on*, 26(4):401–406.

Lemire, D., Kaser, O., and Aouiche, K. (2010). Sorting improves word-aligned bitmap indexes. *Data & Knowledge Engineering*, 69(1):3–28.

Mahoney, M. (2013). The zpaq open standard format for highly compressed data - level 2. `http://mattmahoney.net/dc/zpaq202.pdf`. [Online; accessed 10-May-2015].

Mogul, J. C., Douglis, F., Feldmann, A., and Krishnamurthy, B. (1997). Potential benefits of delta encoding and data compression for http. In *ACM SIGCOMM Computer Communication Review*, volume 27, pages 181–194. ACM.

Oechslin, P. (2003). Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology-CRYPTO 2003*, pages 617–630. Springer.

Paulo, J. and Pereira, J. (2014). A survey and classification of storage deduplication systems. *ACM Computing Surveys (CSUR)*, 47(1):11.

Percival, C. and Josefsson, S. (2015). The scrypt password-based key derivation function.

Provos, N. and Mazieres, D. (1999). A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*, pages 81–91.

Rivest, R. (1992). The md5 message-digest algorithm. `https://www.ietf.org/rfc/rfc1321.txt`. [Online; accessed 02-May-2015].

Sabharwal, C. L. and Bratia, S. K. (1997). Image databases and near-perfect hash table. *Pattern Recognition*, 30(11):1867–1876.

Sleator, D. D. and Tarjan, R. E. (1985). Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686.

Stalings, W. (2005). *Cryptography and Network Security*. Upper Saddle River, NJ: Prentice Hall.

Zhang, W., Zhang, M., Liu, Y., and Wang, R. (2010). A new time-memory-resource trade-off method for password recovery. In *Communications and Intelligence Information Security (ICCIIS), 2010 International Conference on*, pages 75–79. IEEE.

Efficient Software Implementations of Fantomas

Rafael J. Cruz[1], **Diego F. Aranha**[1]

[1]Laboratory of Security and Applied Cryptography (LASCA)

Institute of Computing (IC) – University of Campinas (Unicamp)

Av. Albert Einstein, 1251 – Campinas/SP – Brazil

{raju,dfaranha}@lasca.ic.unicamp.br

Abstract. *We present a series of software implementations of the Fantomas block cipher in resource-constrained ARM devices like the Cortex-M3 and Cortex-M4; and more powerful processors such as the ARM Cortex-A15 and modern Intel platforms. Our implementations span a broad range of characteristics: 32-bit and 64-bit versions, side-channel resistant and vectorized code for NEON and SSE instructions. Our implementations of the algorithm improve the state of the art substantially, both in terms of efficiency or compactness, by making use of novel algorithmic techniques and features specific to the target platform. In particular, our unprotected 32-bit implementation achieves speedups from 35% to 66% in the ARM Cortex-M architecture, while consuming considerably less code size. The vectorized implementations improve performance over the state of the art by 40% in the ARM Cortex-A15 and 50% in the Core i7 Ivy Bridge, setting new speed records for the implementation of the block cipher.*

1. Introduction

Lightweight cryptography for embedded systems has been a very active field of research in the last years, and it recently gained renewed interest with the emergence of the Internet of Things. Cryptographic primitives can indeed mitigate or even solve several problems faced by connected devices collecting and exchanging sensitive information through an open network. Many innovative encryption algorithms were proposed to maximize performance in resource-constrained devices and to provide lighter alternatives to AES [Daemen and Rijmen 2002], without compromising security. Some remarkable examples are the NSA-designed SPECK and SIMON block cipher families [Beaulieu et al. 2013], the PRINCE block cipher [Borghoff et al. 2012], and more recently the Low-power Encryption Algorithm (LEA) [Hong et al. 2014]. These lightweight designs follow multiple constructions, such as Feistel, Substitution-Permutation and ARX networks, posing distinct trade-offs in terms of efficiency, compactness and resistance against different attacks. While these algorithms are still considered secure according to the latest cryptanalytic results, their corresponding implementations may be susceptible to attacks based on information leakage.

Side-channel analysis is a growing and important issue for security in cryptography, specially in embedded devices. These attacks are based on information leaked during computation through side channels such as execution time, energy consumption, acoustic

and electromagnetic emanations. When successful, they help the adversary to identify and recover secret data from observations captured from implementations of cryptography, overcoming the much higher computational cost of cryptanalysis or exhaustive search in the key space. Secret data may be a long-term private key, an ephemeral session key or partial information about the internal state of a primitive, including bits of the plaintext or round keys. The attacks may be based on a small number of observations, such as Branch Prediction [Aciiçmez et al. 2007] or Simple Power Attacks (SPA); or require traces from many consecutive observations, as in the case of Differential Power Attacks (DPA) [Kocher et al. 1999]. Resistance to side-channel attacks has been considered as an additional security requirement for low-cost ciphers, because the lightweight devices implementing them may be physically accessible to the attacker. Algorithms with side-channel resistance guarantees embedded in the construction itself have been thus favored in the scientific literature, bringing attention to ciphers like PICARO [Piret et al. 2012] and Fantomas [Grosso et al. 2015c].

The LS-Design paradigm [Grosso et al. 2015c] was created with side-channel resistance in mind, because it allows the designer to construct lightweight algorithms friendly to efficient implementation of side-channel countermeasures. LS-Design ciphers typically combine a bitsliced substitution layer with a linear diffusion layer implemented with precomputed tables, both amenable to masking techniques with controlled overhead. Masking schemes were initially proposed in 2003 [Ishai et al. 2003] in the context of protecting circuits against probing, but it has been later extended to much more complex operations, even achieving provable security guarantees [Rivain and Prouff 2010]. Masked implementations have the interesting property that the entire computation is performed over *shared secrets*, decorrelating any potential side-channel leakages from the actual data being encrypted or the real cryptographic keys. From this point of view, masking can be seen as a collection of perturbation techniques to introduce external random noise in the encryption or decryption processes, acting as countermeasure against several types of side-channel attacks.

One of the first and most famous instances of the LS-Design construction is the Fantomas block cipher. This work presents several efficient, compact, portable and secure (in the sense of side-channel resistance) implementations of Fantomas. In terms of performance, a number of optimizations are described to save execution time or code size, several of them easily adaptable to other LS-Designs, such as the CAESAR candidate SCREAMv3 [Grosso et al. 2015b]. In terms of security, constant-time and masked implementations are discussed. The constant-time implementation protects execution against timing attacks [Kocher 1996] and avoids precomputed tables vulnerable against cache latency attacks [Bernstein 2004, Bonneau and Mironov 2006]; and the masked implementation illustrates several current challenges of the research field. The constant-time implementation was validated using the FlowTracker static analysis tool [Silva et al. 2016].

This paper is organized as follows. Section 2 introduces the masking implementation strategy, LS-Designs and the Fantomas block cipher. Section 3 discusses multiple implementations of the algorithm, targeting different platforms. Section 4 presents experimental results and Section 5 concludes the paper.

2. Preliminaries

In this section, we introduce the concept of masking for protecting implementations against side-channel attacks and describe the LS-design construction instantiated by the Fantomas block cipher.

2.1. Masking Scheme

Masking is one of the most investigated countermeasures against side-channel cryptanalysis. In the context of block ciphers, masking aims to protect sensitive data, such as plaintext during encryption, or the ciphertext during decryption. Because information computed in these processes will be later transformed into the algorithm outputs, all intermediary states must be protected at all time. The masked state of m with $d+1$ *shared secrets* is given by $m = \bigoplus_{i=0}^{d} m_i = m_0 \oplus m_1 \oplus \ldots \oplus m_d$, where each m_i is a shared secret and all shared secrets form together a masked secret. From this definition, we can collect some observations on ciphers employing operations in finite field $\mathbb{F}_2$:

1. Every linear operation over a masked secret m is equivalent to applying the same operation over shared secrets of m:
$$L(m) \equiv L(m_0 \oplus m_1 \oplus \ldots \oplus m_d) \equiv L(m_0) \oplus L(m_1) \oplus \ldots \oplus L(m_d)$$
2. A `NOT` operation over a masked secret can be computed as:
$$\neg m \equiv \neg m_0 \oplus m_1 \oplus \ldots \oplus m_d$$
3. A `XOR` operation between masked secrets $a = \bigoplus_{i=0}^{d} a_i$ and $b = \bigoplus_{i=0}^{d} b_i$ can be seen as:
$$a \oplus b \equiv \bigoplus_{i=0}^{d} a_i \oplus \bigoplus_{i=0}^{d} b_i \equiv \bigoplus_{i=0}^{d} (a_i \oplus b_i)$$
4. An `AND` operation between two masked secrets $a = \bigoplus_{i=0}^{d} a_i$ and $b = \bigoplus_{i=0}^{d} b_i$ is more complicated and can be computed as in Algorithm 1.

Algorithm 1 Non linear operation `AND` performed on two masked secrets a and b

Require: Shares (a_i) and (b_i) satisfying $\oplus_{i=0}^{d} a_i = a$ and $\oplus_{i=0}^{d} b_i = b$.
Ensure: Shares (c_i) satisfying $\oplus_{i=0}^{d} c_i = a \wedge b$

for i **from** 0 **to** d **do**
  $r_{i,i} \leftarrow 0$;
  for j **from** $i+1$ **to** d **do**
    $r_{i,j} \leftarrow random()$;
    $r_{j,i} \leftarrow (r_{i,j} \oplus (a_i \wedge b_j)) \oplus (a_j \wedge b_i)$;
  end for
end for
for i **from** 0 **to** d **do**
  $c_i \leftarrow a_i \wedge b_i$;
  for j from 0 to d **do**
    $c_i \leftarrow c_i \oplus r_{i,j}$;
  end for
end for

These observations allow any algorithm employing binary field arithmetic to be implemented in a masked way. An important challenge in masked implementations can be seen in line 4 of the algorithm, in the form of random number generation. By considering that every share a_i is a *unity*, every masked AND requires $\frac{(d+1)^2-(d+1)}{2}$ unities of random data and additional space of $(d+1)^2$ to store a matrix containing all possible combinations of shares.

2.2. LS-Designs and Fantomas

LS-Designs were conceived to address side-channel threats, by combining the advantages of bitslicing-capable ciphers with easy support to regular and masked software implementations. Algorithm 2 presents a generic specification for an LS-Design, illustrating its simplicity and regularity. Instances of a LS-Design cipher are characterized by the choice of bitsliced S-boxes S, an L-box matrix L acting as the diffusion layer, a number of rounds N_r and round constants $C(r)$. In the original LS-Design paper, two ciphers were instantiated and analyzed: Robin, a faster involutive instance that later succumbed to invariant subspace attacks [Leander et al. 2015]; and the non-involutive candidate Fantomas.

Algorithm 2 LS-Design construction encrypting plaintext P with key K.

1: $x \leftarrow P \oplus K$ ▷ x represents an $s \times l$-bit matrix
2: **for** $0 \leq r < N_r$ **do**
3: **for** $0 \leq i < l$ **do** ▷ S-box layer
4: $x[i, \star] = S[x[i, \star]]$
5: **end for**
6: **for** $0 \leq j < s$ **do** ▷ L-box layer
7: $x[\star, j] = L[x[\star, j]]$
8: **end for**
9: $x \leftarrow x \oplus K \oplus C(r)$ ▷ Key and round constant addition
10: **end for**
11: **return** x

Fantomas employs the 3/5-bit S-boxes from the 3-round MISTY cipher [Canteaut et al. 2016], as presented in detail on Algorithm 3. An important consideration taken by the original authors of the cipher is the number of AND operations in the choice of S-boxes. As discussed in Section 2.1, masked implementations of the algorithm must rely on Algorithm 1 when computing ANDs. For security of the masking countermeasure, a lower bound on the number of ANDs is the size of the S-boxes. Because Fantomas employs S-boxes of 8-bit granularity, the S-boxes must contain at least 8 AND operations to be appropriate for masking. There is some security margin in this design decision because Fantomas employs 11 AND operations between elements of the cipher state. The L-box is presented in Figure 1 and its computation can be seen as a vector-matrix product in $\mathbb{F}_2$, as illustrated in the picture.

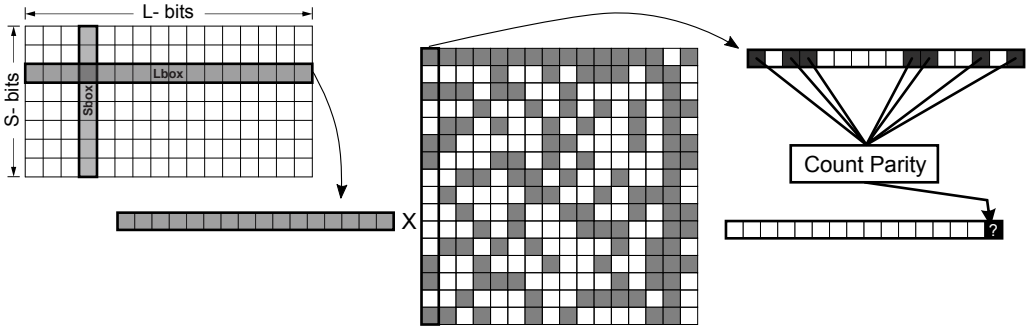


Figure 1. Linear layer of Fantomas. The L-box matrix has gray cells for `1` bits and white cells for `0` bits.

Algorithm 3 MISTY 3/5 bits S-boxes on state $x = \{X_0, X_1, \ldots, X_7\}$

.

▷ S5
$X_2 \leftarrow X_2 \oplus (X_0 \wedge X_1)$;
$X_1 \leftarrow X_1 \oplus X_2$;
$X_3 \leftarrow X_3 \oplus (X_0 \wedge X_4)$;
$X_2 \leftarrow X_2 \oplus X_3$;
$X_0 \leftarrow X_0 \oplus (X_1 \wedge X_3)$;
$X_4 \leftarrow X_4 \oplus X_1$;
$X_1 \leftarrow X_1 \oplus (X_2 \wedge X_4)$;
$X_1 \leftarrow X_1 \oplus X_0$;
▷ Extend-Xor
$X_0 \leftarrow X_0 \oplus X_5$;
$X_1 \leftarrow X_1 \oplus X_6$;
$X_2 \leftarrow X_2 \oplus X_7$;
▷ Key
$X_3 \leftarrow \neg X_3$;
$X_4 \leftarrow \neg X_4$;
▷ S3: 3-bit Keccak S-box
$t_0 \leftarrow X_5, t_1 \leftarrow X_6, t_2 \leftarrow X_7$;
$X_5 \leftarrow X_5 \oplus ((\neg t_1) \wedge t_2)$;
$X_6 \leftarrow X_6 \oplus ((\neg t_2) \wedge t_0)$;
$X_7 \leftarrow X_7 \oplus ((\neg t_0) \wedge t_1)$;
▷ Truncate-Xor
$X_5 \leftarrow X_5 \oplus X_0$;
$X_6 \leftarrow X_6 \oplus X_1$;
$X_7 \leftarrow X_7 \oplus X_2$;
▷ S5
$X_2 \leftarrow X_2 \oplus (X_0 \wedge X_1)$;
$X_1 \leftarrow X_1 \oplus X_2$;
$X_3 \leftarrow X_3 \oplus (X_0 \wedge X_4)$;
$X_2 \leftarrow X_2 \oplus X_3$;
$X_0 \leftarrow X_0 \oplus (X_1 \wedge X_3)$;
$X_4 \leftarrow X_4 \oplus X_1$;
$X_1 \leftarrow X_1 \oplus (X_2 \wedge X_4)$;
$X_1 \leftarrow X_1 \oplus X_0$;

3. Implementation

In this section, we present the multiple implementations of the Fantomas block cipher performed by this work. We discuss portable implementations for 32-bit and 64-bit processors, mostly targeting ARM platforms, and additional code vectorized for SSE/NEON instructions. Strategies for masked implementation are discussed later, before experimental results are presented.

3.1. 32-bit implementation

We have implemented two 32-bit variants of the cipher: a constant-time version protected against timings attacks and an unprotected one. Both versions require S/L-boxes which operate over 16-bit chunks and other operations over 32-bit data, such as key addition. Therefore, a portable and efficient implementation must simultaneously support

the two data types in one concise structure. Following the C99 standard, we prevented breaking strict aliasing point rules by representing the internal state as a union combining pointers to the data types as in Figure 2.

```
typedef union {
    uint32_t u32;
    uint16_t u16[2];
} U32_t;
```

Figure 2. Union to respect the strict aliasing rule: two different pointers cannot reference the same memory area.

The implementations still take aligned byte vectors as input and conveniently converts them to 32-bit pointers when needed. The S-boxes must then be computed using the union structure. Some operations over 16-bit chunks could be combined in 32-bit operations, but this was avoided to prevent unaligned loads and stores. Their bitsliced structure already provides the constant time property due to bitslicing, so no additional countermeasures were needed for secure implementation of the substitution layer.

The diffusion layer is performance-critical and presents more obstacles to side-channel resistance, since it is implemented through table lookups on the L-box. The unprotected version employs two 256-position half-word precomputed tables, while the protected version implements the operation online by performing a vector-matrix binary multiplication, where two 16-bit words are processed at the same time. A small code portion illustrating the unprotected L-box can be found in Figure 3, where `state` stores the 128-bit state, `LBoxH` transforms the 8 most significant bits and `LboxL` transforms the 8 less significant bits for all $j \in \{0, 1, 2, 3\}$. Observe that the table lookups are vulnerable to adversarial influence over the memory hierarchy in processors equipped with cache memory [Bernstein 2004, Bonneau and Mironov 2006].

```
/* Unprotected L-box version */
state[j].u16[0] = LBoxH[state[j].u16[0]>>8] ^
                  LBoxL[state[j].u16[0] & 0xff];
state[j].u16[1] = LBoxH[state[j].u16[1]>>8] ^
                  LBoxL[state[j].u16[1] & 0xff];
```

Figure 3. Unprotected L-Box using the internal states of the union. L-BoxH contains the higher 8-bit of the linear transformation and L-BoxL other less significant 8-bit.

The protected implementation is a little more involved. The code portion in Figure 4 illustrates part of it, where `x` stores the 32 bits to be transformed by the L-box in 16-bit pairs and `y` contains the `s`-th duplicate line of the binary matrix representing the linear transformation. This function computes the dot product of the two 32-bit vectors in $\mathbb{F}_2$, and calculates the parity of each 16-bit result, processing two transformations at the same time.

The key addition of Fantomas works by accumulating the key in the internal state using 32-bit XOR operations, as in Figure 5.


```
static inline uint32_t ProdLBox(uint32_t x, uint32_t y, uint8_t s) {
    x &= y;
    x ^= x >> 8;
    x ^= x >> 4;
    x ^= x >> 2;
    x ^= x >> 1;
    return (x & 0x00010001) << s;
}
```

Figure 4. Multiplying the s-th row of the matrix L containing the value $y = (y_s, y_s)$ by the value $x = (x_a, x_b)$ where the result is the s-th value of $(x \cdot L)_s = (x_a \cdot y_s, x_b \cdot y_s)$.

```
for(j=0; j < 4; j++)
    state[j].u32 ^= key_32[j];
```

Figure 5. Key Addition of Fantomas using the 32-bit state of the union.

3.2. 64-bit implementation

Two variants of the cipher were also implemented for 64-bit architectures. A modified union structure combines 16-bit and 64-bit words. The S-boxes must again be implemented over the union without breaking alignment and causing performance penalties. The unprotected L-box follows the same structure as the corresponding 32-bit implementation. Function `ProdLBox` was transformed to operate over 64 bits with simple modifications to the input and output types and a duplicated bit mask `0x0001000100010001` in the last operation, allowing computation of 4 simultaneous evaluations of the L-box. We also implemented a 64-bit version using the `POPCNT` instruction that proved much less efficient.

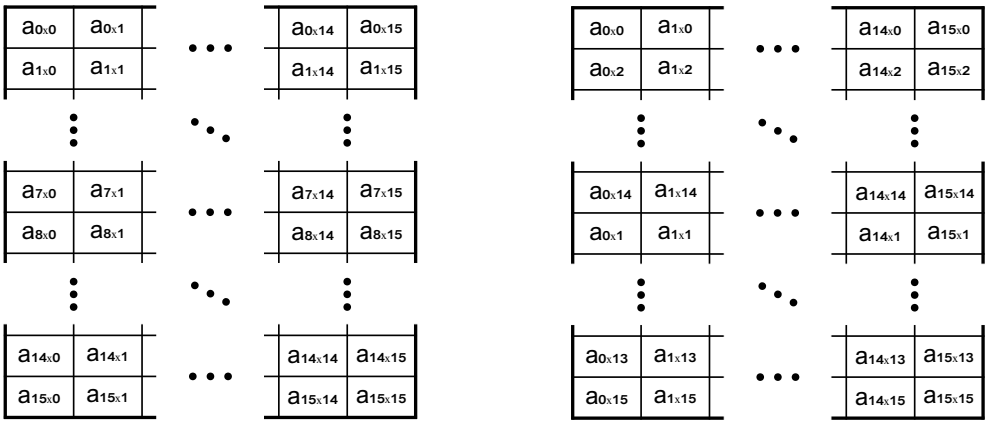
3.3. Vectorized implementation

We first discuss what structures inside a typical LS-Design are friendly to vectorization, before the vector implementations for ARM (using NEON instructions) or Intel platforms (equipped with SSE instructions) are described. The S-boxes are computed in a bitsliced way, facilitating vectorization as long as the S-layer can compute over at least 8 blocks simultaneously, applying the same operation over each 16-bit chunk from the same block. The L-box presents a higher obstacle, because memory accesses should be reduced to increase arithmetic density. There are two clear ways of implementing the L-box with high arithmetic density: the first one is to perform an explicit vector-matrix multiplication over $\mathbb{F}_2$ as in the constant-time 32/64-bit implementation; or employing byte shuffling instructions (such as Intel `PSHUFB`) for table lookups inside registers. Byte shuffling instructions take 128-bit registers filled with bytes $r_a = a_0, a_1, \ldots, a_{16}$ and $r_b = b_0, b_1, \ldots, b_{15}$ and replace r_a with the permutation $a_{b_0}, a_{b_1}, \ldots, a_{b_{15}}$. A powerful use of this instruction is to perform 16 simultaneous lookups in a 16-byte lookup table, computing a mapping from 4-bit sets to 8-bit values. This can be easily done by storing the lookup table in r_a and the lookup indexes in r_b. These two approaches were implemented and the latter was clearly faster due to higher occupancy of the vector registers. For portability over Intel and ARM, the table lookups were implemented using the GCC intrinsic `__builtin_shuffle()` for byte shuffling. Because the Fantomas block size

has 128 bits (or 16 bytes), the best choice for minimum number of blocks is 16. When operating over 16 blocks simultaneously, the individual bytes can be stored and transposed in a matrix to guarantee that every vector register has the same i-th byte of each block.

Observe that picking a certain number of simultaneous blocks to operate has impact on the choice of block cipher operating mode, since parallelism must be supported in both encryption and decryption. The traditional CBC mode of operation imposes a serialization of encryption, thus vectorized implementations should stick to the CTR mode of operation, where only the counters are encrypted/decrypted and later added to the plaintext/ciphertext, respectively. We also implemented a single-block vectorized Fantomas for benchmarking the CBC mode of operation, using the same substitution layer described in Algorithm 3.

The L-box is a linear transformation, thus the 16 bits can be broken in smaller pieces. Hence the L-box in Figure 1 can be split in 4-bit chunks and the table reduced to 4 tables of 16 positions storing 16-bit values. To make use of the shuffle instructions mapping 4-bit sets to 8 bits, the splitting must divide the most significant bytes from the least significant bytes and the entire table is stored in 8 vector registers of 128 bits. The single-block CBC version operates separately in the most significant bytes and least significant bytes, and combines them together at the end. The 16-block CTR version is a little more complex. First, it is necessary to expand the CTR counter for the 16 simultaneous blocks. After expansion, the counters must be transposed and stored in a different order. Counter updates can be done by propagating carries using vector comparisons. The expanded counter is computed from the original counter as in Figure 6a, and the state must be transposed and stored as in Figure 6b below.



(a) Initial state of the counter **(b) Final state of the counter**

Figure 6. Counter transformation for the vectorized CTR implementation.

The organization in Figure 6 must be kept through the whole process, because then the substitution layer can be performed in the first 8 blocks and then on the final 8 blocks. The linear layer is similar to the single-block version and the splitting is not required, since the least significant bytes are stored in the first 8 blocks in Figure 6b and the most significant bytes in the remaining 8 blocks in the same Figure. The SSE versions of Fantomas are publicly available for independent benchmarking and reproducibility[1].

[1] `https://github.com/rafajunio/fantomas-x86`

3.4. Masked implementation

The masked implementation needs only large modifications in the S-boxes, because every operation computed in Algorithm 3 must now be replaced by the operations specified in Section 2.1. Countermeasures are still needed for encryption and decryption, because the linear layer comes immediately before the last key addition in the encryption and comes immediately after the first key addition in the decryption. A cache latency attack would disclose the internal state in these positions and, with knowledge of the ciphertext, an attacker could mount a critical key recovery attack.

Two functions are essential for preprocessing the blocks before masked encryption and decryption can be performed. These functions convert a plaintext block to a masked block and the converse, respectively. The first function must generate $d - 1$ randomized blocks and combine these blocks with the original by means of `XOR` operations to generate the last block. The second function must combine all masked blocks with `XOR` operations after encryption and decryption are processed. There are two different ways to compute key addition. The key can be added directly to the masked state and the key can also be masked for addition in the masked state.

A substantial amount of random bits is required to generate the masked blocks and to compute the masked `AND` described in Algorithm 1. We implemented random number generation through the standardized HASH_DRBG [Barker and Kelsey 2012] instantiated with the SHA-256 hash function. This choice proved to be faster than reading bytes from `/dev/urandom` by a 10-factor. Even if faster, generating random bits still impose a massive performance penalty and represents around 97% of the execution time in the masked implementation.

4. Experimental results

Our implementations were benchmarked in five different platforms:

- **Cortex-M3:** Arduino Due powered by an Atmel SAM3X8E ARM Cortex-M3 84MHz CPU. The compiler provided by the latest version of the Arduino Development Kit, GCC 4.8.4, was used with flags `-O3 -fno-schedule-insns -nostdlib -mcpu=cortex-m3 -mthumb`. Execution time was measured by converting the output of the `micros()` function in Arduino for measuring microseconds to cycles through simple multiplication by the nominal frequency.
- **Cortex-M4:** Teensy 3.2 board containing a MK20DX256VLH7 Cortex-M4 72MHz processor. The same compiler used in the Cortex-M3 platform was used, but with flags `-O3 -fno-schedule-insns -nostdlib -mcpu=cortex-m4 -mthumb`. Execution time was measured through a native cycle counting register and some Assembly code.
- **Cortex-A15:** ODROID-XU4 board containing a Samsung Exynos5422 Cortex-A15 2Ghz and Cortex-A7 octa-core CPUs. We installed the official distribution of Arch Linux for the board, which comes equipped with GCC 6.1.1 for ARM, using the flags `-O3 -fno-schedule-insns -mcpu=cortex-a15 -mthumb -march=native`. Execution time was measured by enabling reading from the Cycle CouNT register (CCNT) from the Performance Monitor Unit (PMU) in user level.

- **Cortex-A53:** ODROID-C2 board containing an Amlogic ARM Cortex-A53(ARMv8) 2Ghz quad-core CPUs. We installed Arch Linux and employed GCC 6.1.1 cross-compiled for ARM with flags `-O3 -fno-schedule-insns -mcpu=cortex-a53 -mthumb -march=native`. Execution time was also measured through the PMU enabled by loading a special kernel module.
- **Core i7 Ivy Bridge:** Intel Core i7-3632Q 2.20GHz CPU. GCC 6.1.1 was again used with flags `-O3 -fno-schedule-insns -mssse3 -msse march=native`. The RDTSC register was used for cycle counting.

Table 1 presents results for the 32- and 64-bit portable implementations; and the NEON and SSE vectorized implementations of Fantomas. All measurements take into account the time to encrypt and decrypt using the operating modes CBC and CTR. The constant-time implementations in C receive the CT abbreviation. The vector implementations are intrinsically constant time by operating over registers only. Cycle counts were computed by encrypting *and* decrypting a 100 times the same message of length 1024 bytes and dividing the result by twice the number of bytes. The final result represents the average time to encrypt *or* decrypt a single byte using a specific implementation.

Table 1. Execution time and code size (ROM bytes) for Fantomas benchmarked in Cortex-M3/M4/A15/A53 ARM and Core i7 x86 Ivy Bridge. Figures present average cycles for encrypting or decrypting a single byte (CPB) in CBC/CTR mode, possibly using a constant-time implementation (CT) and vectorized implementation in NEON/SSE.

	Cortex-M3		Cortex-M4		Cortex-A15		Cortex-A53		I7 Ivy Bridge	
	Cycles per byte (CPB)	Code size (Bytes)	Cycles per byte (CPB)	Code size (Bytes)	Cycles per byte (CPB)	Code size (Bytes)	Cycles per byte (CPB)	Code size (Bytes)	Cycles per byte (CPB)	Code size (Bytes)
Fantomas 32 (CBC)	**177.44**	**3202**	143.62	3276	51.91	3900	87.65	3764	68.73	4040
Fantomas 32 (CTR)	**171.88**	**1916**	136.92	1934	50.71	2284	83.03	2364	67.06	2449
Fantomas 32 CT (CBC)	614.82	1858	489.13	1848	443.13	3084	232.73	3292	194.13	3680
Fantomas 32 CT (CTR)	629.59	1272	491.65	1262	443.44	1884	228.16	2148	194.13	2257
Fantomas 64 (CBC)	174.64	3198	142.88	3278	52.27	3628	80.87	3452	58.33	3736
Fantomas 64 (CTR)	168.81	1914	133.79	1934	50.66	2208	76.03	2200	56.03	2348
Fantomas 64 CT (CBC)	1920.06	4826	1559.42	4814	966.21	9372	226.54	2984	149.47	3395
Fantomas 64 CT (CTR)	1920.47	2738	1555.54	2734	961.77	5084	222.13	1984	127.95	2209
Fantomas NEON/SSE (CBC)	–	–	–	–	65.10	1844	62.04	1340	59.71	1490
Fantomas NEON/SSE (CTR)	–	–	–	–	63.81	1264	59.50	1160	46.49	1164
Fantomas16 NEON/SSE (CTR)	–	–	–	–	**16.07**	6846	17.93	3884	**5.95**	6139
Related work										
Fantomas 32 (CBC) Fast[1]	274.21	4620	–	–	–	–	–	–	–	–
Fantomas 32 (CTR) Fast[1]	220.13	2088	–	–	–	–	–	–		–
Fantomas 32 (CBC) Compact[1]	370.79	2916	–	–	–	–	–	–	–	–
Fantomas 32 (CTR) Compact[1]	520.94	1384	–	–	–	–	–	–	–	–
Fantomas16 NEON/SSE (CTR)[2]	–	–	–	–	26.60*	–	–	–	12.00*	–

[1] [Dinu et al. 2015] [2] [Grosso et al. 2015a] * adjusted timings

4.1. Discussion

The table contains plenty of interesting results to be discussed. Constant time implementations with uniform access to memory receive a massive performance penalty. In the Cortex-M, the 32-bit CBC/CTR constant time implementation of Fantomas proved to be almost twice as compact, although more than 3 times slower than the unprotected version. If the main objective is to obtain a smaller code fingerprint and/or resistance against timing-based side-channel attacks, this implementation can still be a good choice. Observe that Cortex-A processors and even some Cortex-M4 microcontrollers have cache memory, so it is also important to measure the performance impact of protecting the implementations against cache-timing leakage. The 64-bit implementations of Fantomas showed little difference in comparison to the 32-bit code in all platforms, with the i7 Ivy Bridge processor as the only exception with a 22% speedup. The Cortex-A53 is a great platform for the 32-bit constant time implementation, producing an almost twice faster implementation than the Cortex-A15. On the other hand, the Cortex-A53 is at the low-end of the 64-bit ARM processors, so better performance for the 64-bit implementation might be expected from higher-end processors. Our implementations were also tailored for ARM processors and enjoy the benefits of the second-operand barrel shifter as much as possible. Although not comparable, we note that cycle counts for 32-bit Fantomas were 25% lower in the Cortex-A15 than a Desktop machine equiped with the Ivy Bridge processor.

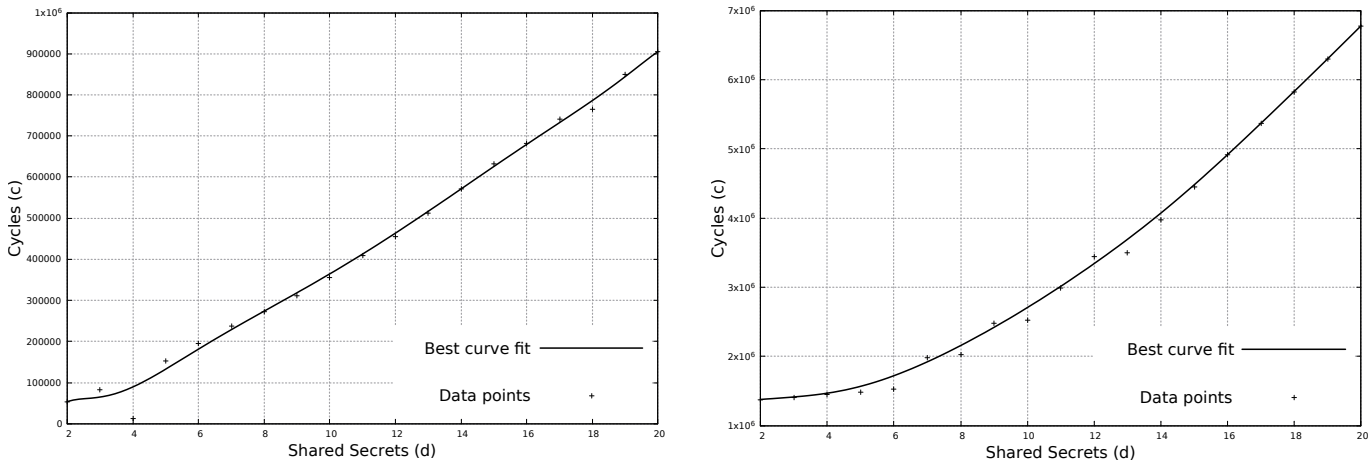
Code sizes generally grow from the Cortex-M3 to the Ivy Bridge. In the case of 64-bit code, there is a small code reduction starting in the Cortex-A53. This can be explained by the fact that the Cortex-M3/M4/A15 do not have 64-bit instructions, which means that 64-bit operations insert pairs of 32-bit instructions, resulting in a larger footprint. The code size for the Fantomas16 NEON (CTR) implementation in the Cortex-A53 was also surprising, producing almost twice more compact binaries than the same NEON version in the Cortex-A15 and the similar SSE version in the Core i7. There is a clear space-time trade-off in this implementation. It is the largest implementation in terms of code size, but also the fastest among all platforms supporting vector instructions.

Cycle counts for the masked implementation of Fantomas are presented in Figure 7. A clear quadratic trend for the performance degradation can be observed in the graph, as expected. Two versions were implemented: masked key addition and conventional key addition. The two versions take such a close execution time that performance figures are essentially the same, so we opted to only present the masked key addition version together with the best curve fit. This happens because generating random bytes for the masked `AND` operations in the S-boxes consumes approximate 97% of the execution time. Additionally, the data point for a single shared secret is lower than the general trend. The reason for this is mostly the lower requirement of random bytes, although simple calls to the random number generation already cause a substantial performance penalty.

4.2. Comparison to related work

There are two main related works that established the previous state of the art in the context of this work. The most recent is the massive implementation effort from the FELICS framework [Dinu et al. 2015] to compare lightweight block ciphers performance-wise in representative platforms of 8, 16 and 32 bits. The project website [2] also contains

[2] `https://www.cryptolux.org/index.php/FELICS`



(a) Without random bit generator. **(b) With DRBG_HASH using SHA256.**

Figure 7. Cycle counts for encrypting and decrypting with the masked implementation of Fantomas using masked key addition in the Cortex-A15 platform, as a function of the number d of shares.

results for some stream ciphers and block ciphers underlying MAC constructions. The target 32-bit platform considered in their work is the same Cortex-M3 present in the Arduino Due and two scenarios are taken into consideration. Scenario 1 considers consecutive encryption and decryption of 128 bytes in CBC mode. In the paper, the best implementation according to their Figure of Merit (FOM) takes 70,197 cycles using 4620 bytes of ROM (or 274.21 CPB in Table 1). The website has more recent numbers for an implementation capable of encrypting and decrypting in 94,921 cycles which consumes 2916 bytes of ROM (or 370.79 CPB in Table 1). Our implementation is 35.4% and 52.2% faster than their implementations, respectively, and competitive in terms of code size with the more compact implementation. In Scenario 2, FELICS reports a range of figures for unprotected Fantomas when encrypting 128 bits in CTR mode, ranging from most compact implementation to best execution time. The most compact takes 8335 cycles and 1384 bytes of ROM (520.94 CPB), the most efficient takes 3522 cycles and 2088 bytes of ROM (220.13 CPB) and a good trade-off is found at 4550 cycles and 2184 bytes of code size (284.38 CPB). After the proper conversions, our implementation improves these figures by 66.5%, 20.9% and 37.7%, respectively, by spending only 1916 bytes of ROM. As a reference point, FELICS reports much higher latencies for standardized block ciphers such as AES under different operating modes (73,868 cycles for encrypting and decrypting in CBC mode in the Cortex-M3, for example).

The second related work is the presentation for the SCREAMv3 candidate in the CAESAR competition [Grosso et al. 2015a]. In the slides, numbers for a 16-block vector implementation of Fantomas are also reported. We could not reproduce the numbers presented in the table due to unavailability of the Fantomas code, and benchmarking the publicly-available SCREAMv3 code gave rather different results. In private contact with the authors, we discovered that their benchmarking code takes the outputs of the `gettimeofday()` function for time measurement, a less precise approach than using cycle counts measured directly. Additionally, it is not clear if their numbers were taken in a machine with Turbo Boost enabled, as it is well known to distort benchmarking

data [Bernstein and Lange 2016]. By using the difference observed when benchmarking the SCREAMv3 reference code, we adjusted the timings in the CAESAR slides for architectures ARM Cortex-A15 by a factor of 1.5; and Intel Core i7 Ivy Bridge by a factor of 2, both observed in Table 1. After a simple comparison, we observed an approximate performance gain of 40% and 50% of our implementation when compared to the adjusted timings in the ARM and Intel platforms, respectively.

5. Conclusion

We presented several serial and vectorized software implementations of the Fantomas block cipher, producing more efficient and compact implementations in the ARM and x86 target platforms. Two approaches for side-channel resistance were implemented: constant time and masking. The constant time approach for implementing the L-box is of independent interest, as it can also be easily extended to other LS-Design ciphers. The masked implementation illustrates the computational cost of powerful side-channel countermeasures. Even if Fantomas was conceived to be easily masked in a protected implementation, the performance penalty can be as high as a factor of 40 when compared to a constant time implementation. Highly-efficient random number generation is a paramount research target for enabling masked implementations in software to perform well in the target platforms.

Acknowledgements

The authors acknowledge support from LG Electronics Inc. during the development of this research, under the project "*Efficient and Secure Cryptography for IoT*".

References

Aciiçmez, O., Koç, c. K., and Seifert, J.-P. (2007). On the power of simple branch prediction analysis. In *ASIACCS*, pages 312–320. ACM.

Barker, E. and Kelsey, J. (2012). NIST SP 800-90A – Recommendation for Random Number Generation Using Deterministic Random Bit Generators.

Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., and Wingers, L. (2013). The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404. `http://eprint.iacr.org/2013/404`.

Bernstein, D. J. (2004). Cache-timing attacks on AES. URL: `http://cr.yp.to/papers.html#cachetiming`.

Bernstein, D. J. and Lange, T. (2016). eBACS: ECRYPT Benchmarking of Cryptographic Systems. `http://bench.cr.yp.to`.

Bonneau, J. and Mironov, I. (2006). *Cache-Collision Timing Attacks Against AES*, pages 201–215. Springer Berlin Heidelberg, Berlin, Heidelberg.

Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E. B., Knezevic, M., Knudsen, L. R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S. S., and Yalçın, T. (2012). *PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications*, pages 208–225. Springer Berlin Heidelberg, Berlin, Heidelberg.

Canteaut, A., Duval, S., and Leurent, G. (2016). *Construction of Lightweight S-Boxes Using Feistel and MISTY Structures*, pages 373–393. Springer International Publishing, Cham.

Daemen, J. and Rijmen, V. (2002). The - advanced encryption standard.

Dinu, D., Corre, Y. L., Khovratovich, D., Perrin, L., Großschädl, J., and Biryukov, A. (2015). Triathlon of lightweight block ciphers for the internet of things. Cryptology ePrint Archive, Report 2015/209. `http://eprint.iacr.org/`.

Grosso, V., Laurent, G., Standaert, F., Varici, K., Durvaux, F., Gaspar, L., and Kerckhof, S. (2015a). CAESAR candidate SCREAM Side-Channel Resistant Authenticated Encryption with Masking. `http://2014.diac.cr.yp.to/slides/leurent-scream.pdf`.

Grosso, V., Laurent, G., Standaert, F., Varici, K., Durvaux, F., Gaspar, L., and Kerckhof, S. (2015b). SCREAM Side-Channel Resistant Authenticated Encryption with Masking. `https://competitions.cr.yp.to/round2/screamv3.pdf`.

Grosso, V., Leurent, G., Standaert, F.-X., and Varıcı, K. (2015c). *LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations*, pages 18–37. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hong, D., Lee, J.-K., Kim, D.-C., Kwon, D., Ryu, K. H., and Lee, D.-G. (2014). *LEA: A 128-Bit Block Cipher for Fast Encryption on Common Processors*, pages 3–27. Springer International Publishing, Cham.

Ishai, Y., Sahai, A., and Wagner, D. (2003). *Private Circuits: Securing Hardware against Probing Attacks*, pages 463–481. Springer Berlin Heidelberg, Berlin, Heidelberg.

Kocher, P. C. (1996). Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer.

Kocher, P. C., Jaffe, J., and Jun, B. (1999). Differential power analysis. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer.

Leander, G., Minaud, B., and Rønjom, S. (2015). A Generic Approach to Invariant Subspace Attacks: Cryptanalysis of Robin, iSCREAM and Zorro. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 254–283. Springer.

Piret, G., Roche, T., and Carlet, C. (2012). *PICARO – A Block Cipher Allowing Efficient Higher-Order Side-Channel Resistance*, pages 311–328. Springer Berlin Heidelberg, Berlin, Heidelberg.

Rivain, M. and Prouff, E. (2010). *Provably Secure Higher-Order Masking of AES*, pages 413–427. Springer Berlin Heidelberg, Berlin, Heidelberg.

Silva, B. R., Pereira, F. M. Q., and Aranha, D. F. (2016). Sparse representation of implicit flows with applications to side-channel detection. In *25th International Conference on Compiler Construction (CC)*, pages 110–120. ACM.

Uma versão não-interativa do k-NN sobre dados cifrados

Hilder V. L. Pereira, Diego F. Aranha

Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Av. Albert Einstein, 1251 – CEP 13083-852, Campinas/SP – Brazil

hilder@lasca.ic.unicamp.br, dfaranha@ic.unicamp.br

Abstract. *Machine learning tasks typically require large amounts of sensitive data to be shared, which is notoriously intrusive in terms of privacy. Outsourcing this computation to the cloud requires the server to be trusted, introducing a non-realistic security assumption and high risk of abuse or data breaches. In this paper, we propose a non-interactive privacy-preserving version of the k-NN classifier, combining order-preserving encryption and homomorphic encryption. According to our experiments, the privacy-preserving variant achieves the same accuracy as the conventional k-NN classifier, but considerably impacts the original performance. The performance penalty is however still viable for practical use when the security properties provided by the approach are examined in detail. In particular, the cloud server does not need to be trusted beyond correct execution of the protocol and computes the algorithm over encrypted data and encrypted classes, never learning the real dataset values, the number of classes, the query vectors or their classification.*

Resumo. *Tarefas de aprendizagem de máquina normalmente exigem que grandes quantidades de dados sensíveis sejam compartilhados, o que é notoriamente intrusivo em termos de privacidade. Terceirizar esta computação para a nuvem requer que o servidor seja confiável, introduzindo um requisito de segurança não realista. Neste trabalho, propomos uma versão do classificador k-NN que pode ser executada na nuvem sobre dados cifrados de uma forma não interativa, combinando cifração que preserva a ordem e criptografia homomórfica. De acordo com nossos experimentos, a versão sobre dados cifrados alcança a mesma precisão que a convencional, mas com impactos consideráveis no desempenho original. Contudo, a penalidade de desempenho não é proibitiva e a solução continua viável para uso prático, quando as propriedades de segurança adicionais fornecidas são examinadas em detalhe. Em particular, o servidor em nuvem não precisa ser confiável para além da execução correta do protocolo e, como todos os dados recebidos são cifrados, o servidor não aprende os valores do conjunto de dados, o número de classes, os vetores a serem classificados nem as classes a eles atribuídas.*

1. Introdução

Com corporações e governos cada vez mais intrusivos na suas coletas de dados e esforços de vigilância, e também com os recorrentes vazamentos de dados observados nos últimos anos, o paradigma de computação em nuvem enfrenta um novo desafio para permanecer como o modelo de computação vigente. Os baixos custos operacionais e a alta disponibilidade de capacidade de armazenamento e de poder computacional podem não parecer tão

atraentes depois que os riscos de terceirizar a computação e o armazenamento de dados são considerados, especialmente no caso de aplicações sensíveis. Por exemplo, não há garantias formais de que o provedor dos serviços em nuvem não está se comportando de forma abusiva ou intrusiva, ou até mesmo que a infra-estrutura se encontra devidamente protegida contra ataques externos. Diferentes regimes jurídicos e influência governamental introduzem outras complicações para o problema. O impacto financeiro a longo prazo da atual crise de confiança na computação em nuvem é estimado entre 35 e 180 bilhões de dólares nos EUA apenas em 2016 [Miller 2014].

Uma solução proposta para amenizar estas questões consiste na *computação sobre os dados cifrados*. Neste modelo, os dados são cifrados com esquemas que preservam algumas propriedades dos textos claros (originalmente chamados de *homomorfismos privados* [Rivest et al. 1978]), o que permite que algumas operações ainda sejam realizadas no domínio cifrado. Construções que fornecem esse recurso e suportam um número arbitrário de operações (adições e multiplicações) são classificados como *criptografia totalmente homomórfica* e, geralmente, introduzem uma penalidade de desempenho tão grande que torna suas aplicações práticas inviáveis. A *criptografia parcialmente homomórfica ou homomórfica em nível* impõe um limite no número e nos tipos de operações que podem ser realizadas sobre dados cifrados, mas apresenta desempenho muito superior. No entanto, exigem uma reformulação dos algoritmos a serem executados homomorficamente para que estes satisfaçam as restrições impostas por tais esquemas criptográficos.

Tradicionalmente, computação sobre dados cifrados já havia sido aplicada à contagem de votos de forma secreta em eleições eletrônicas, sem a necessidade de revelar os votos que sendo somados [Hirt and Sako 2000], mas esquemas de criptografia homomórfica modernos podem permitir uma série de aplicações e serviços interessantes nos anos futros, como processamento do genoma de forma privativa ou coleta de dados de saúde e diagnósticos que preservam a privacidade dos pacientes [Naehrig et al. 2011]. Mineração de dados [Lindell and Pinkas 2009] e aprendizagem de máquina [Dowlin et al. 2016] são aplicações naturais, pois são serviços já oferecidos na nuvem por empresas como Amazon[1], Google[2] e Microsoft[3]. Tais serviços envolvem quantidades consideráveis de dados a serem compartilhados e manipulados em plataformas não confiáveis.

Alguns trabalhos anteriores na literatura têm tratado do problema de executar algoritmos de classificação sobre dados cifrados. O problema é fundamental em si mesmo e uma solução para ele pode ser usada como sub-rotina de vários algoritmos de aprendizagem de máquina. Um algoritmo de classificação utiliza dados já classificados manualmente (conjunto de treinamento) para inferir a qual classe de dados uma nova amostra pertence. Tal amostra é chamada de vetor de consulta. [Graepel et al. 2013] adaptaram algoritmos de classificação simples para trabalhar sobre dados cifrados usando criptografia homomórfica, enquanto [Bost et al. 2014] consideraram classificadores mais complexos, como árvores de decisão e inferência Bayesiana para aplicações médicas. Outros autores projetaram protocolos para realizar agrupamentos (*clustering*) sobre dados cifrados [Jha et al. 2005] baseando-se em soluções da computação multi-parte (portanto, interativas).

[1] `https://aws.amazon.com/machine-learning/`
[2] `https://cloud.google.com/products/machine-learning/`
[3] `https://azure.microsoft.com/en-us/services/machine-learning/`

A principal contribuição deste trabalho é uma versão do classificador k-NN que preserva a privacidade, sendo executado sobre dados cifrados usando criptografia homomórfica e criptografia que preserva a ordem. Os protocolos são não-interativos, ou seja, os clientes e a nuvem não interagem durante a classificação, e, portanto, são ideais para o cenário de computação em nuvem, no qual o armazenamento e o processamento são delegados de um agente com poucos recursos computacionais para servidores poderosos. Os resultados experimentais demonstram que não há perdas significativas de precisão e a análise de segurança é feita no modelo semi-honesto. Até onde vai o conhecimento dos autores, esta é a primeira versão não-interativa e compatível com computação em nuvem do classificador k-NN sobre dados cifrados.

Este artigo é organizado da seguinte forma: na Seção 2, o classificador k-NN convencional é revisitado e o problema de classificação que preserva a privacidade é definido. Na Seção 3, os conceitos de criptografia homomórfica e de criptografia que preserva a ordem são apresentados. Os algoritmos propostos são discutidos na Seção 4. A Seção 5 apresenta os resultados experimentais comparando, sobre seis diferentes bases de dados, a versão tradicional com as versão que preserva a privacidade. Na Seção 6, as propriedades de segurança oferecidas pelo esquema são discutidas brevemente. Os trabalhos relacionados são discutidos na Seção 7 e as conclusões se encontram na última seção.

2. Definição do problema

Nesta seção, define-se o problema de classificação que o k-NN foi projetado para resolver e discute-se uma variante desse problema levando em conta a privacidade e o modelo de processamento centralizado na nuvem.

2.1. O classificador *k*-NN

O classificador *k-Nearest Neighbor* (k -NN), traduzido normalmente como k vizinhos mais próximos, é um método supervisionado não-paramétrico que serve para classificar uma instância com base nas classes de seus vizinhos mais próximos [Alpaydin 2004, Altman 1992]. Cada instância é representada por um vetor em $\mathbb{R}^p$ (ou seja, tem p atributos) e um rótulo associado, chamado de classe da instância. Um vetor de consulta é uma instância que ainda não tem um rótulo e que deve ser classificada pelo k-NN usando as n instâncias já classificadas.

O classificador k-NN tem um parâmetro k, que é inteiro positivo que representa o número de vizinhos mais próximos considerados ao classificar uma nova instância. Também há uma função d de $\mathbb{R}^p \times \mathbb{R}^p$ para $\mathbb{R}$, que determina a distância entre os duas instâncias (a distância euclidiana é frequentemente utilizada).

Para classificar uma nova instância $x \in \mathbb{R}^p$, o classificador k-NN executa o procedimento a seguir:

1. Encontra os k vizinhos mais próximos entre todas as instâncias classificadas $u_1, u_2, ..., u_n$, ou seja, as k instâncias cujas distâncias $d(u_i, x)$ são as menores.
2. Seleciona a classe mais frequente entre as classes dos k vizinhos mais próximos e a atribui à x.

Há também uma variante do k-NN conhecido como k-NN *ponderado pela distância*, ou simplesmente k-NN *com pesos*. Nesta versão, ao invés de simplesmente

atribuir a classe mais frequente entre os k vizinhos mais próximos, o inverso da distância é usado como o peso do voto de cada um desses k vizinhos.

2.2. k-NN privativo na nuvem

O problema *k-NN não interativo que preserva a privacidade* é definido como o problema de processar o k-NN em uma plataforma não confiável sem a ajuda do proprietário dos dados e sem revelar os valores das instâncias, suas classes, os vetores a serem classificados e as classes atribuídas para eles. Por plataforma não confiável, entende-se uma terceira parte que retém os dados de alguma forma, mas não é confiável sob o ponto de vista do proprietário dos dados.

O cenário considerado neste trabalho supõe um modelo de cliente e servidor, no qual o proprietário dos dados é o cliente e o servidor é um provedor de serviços em nuvem. O cliente pretende armazenar os dados na nuvem e processá-los de uma forma *não-interativa*, isto é, a nuvem deve se comunicar com o cliente apenas para receber os dados e os vetores a serem classificados, mas não deve se comunicar com o cliente durante o processamento. Também assume-se que o cliente tem recursos computacionais restritos, como processamento e espaço de armazenamento.

O outro cenário possível é o de processamento distribuído, no qual o cliente interage com a nuvem (ou com as outras partes envolvidas) recebendo e processando dados durante a fase de treinamento ou de classificação. Vale ressaltar que o modelo não interativo é mais conveniente para o cliente e é o modelo esperado quando se consideram serviços em nuvem.

3. Conceitos Fundamentais

Nesta seção, os dois esquemas de criptografia usados para garantir a propriedade de preservação da privacidade do classificador proposto são apresentados.

3.1. Criptografia que preserva a ordem

Criptografia que preserva a ordem (OPE[4]) é um esquema de criptografia simétrico e determinístico, cuja função de cifração preserva a ordenação numérica dos textos claros [Boldyreva et al. 2009]. Em outras palavras, dados dois textos claros m_1 e m_2, e seus textos cifrados correspondentes, c_1 e c_2, cifrados com um esquema OPE, a seguinte implicação é verdadeira:

$$m_1 \leq m_2 \Rightarrow c_1 \leq c_2.$$

Como esse tipo de esquema trabalha sobre conjuntos finitos de valores numéricos, é suficiente descrevê-los usando o conjunto $D = \{1, 2, ..., M\}$ como o espaço de textos claros (ou domínio) e o conjunto $R = \{1, 2, ..., N\}$ como o espaço de criptogramas (ou contradomínio). Então, o esquema OPE usado aqui é parametrizado por dois inteiros positivos M e N, que representam o número possível de textos claros e de textos cifrados, respectivamente. O esquema é apresentado a seguir:

$\mathrm{OPE.keyGen}(M, N)$ é um algoritmo não determinístico que recebe os parâmetros M e N com $N > M$ e devolve a chave secreta $\mathcal{K}$.

[4]Do inglês *Order-preserving encryption*

OPE. $\mathrm{enc}(\mathcal{K}, m)$ é um algoritmo que cifra um valor $m \in D$ usando a chave $\mathcal{K}$ e devolve um criptograma $c \in R$.

OPE. $\mathrm{dec}(\mathcal{K}, c)$ é um algoritmo que decifra um criptograma $c \in R$ usando a chave $\mathcal{K}$ e devolve o texto claro correspondente $m \in D$.

Para um vetor $w = (w_1, ..., w_p)$, define-se a cifração componente a componente:

$$\text{OPE. enc}(\mathcal{K}, w) = (\text{OPE. enc}(\mathcal{K}, w_1), ..., \text{OPE. enc}(\mathcal{K}, w_p)).$$

Define-se da mesma forma a decifração de um vetor.

O espaço formado pelos vetores cifrados é chamado de *espaço cifrado*. Se um esquema OPE é usado para cifrar vetores, então a ordem é mantida em cada eixo. Portanto, é muito provável que cada vetor mantenha a sua vizinhança no espaço cifrado e esse fato é usado para fazer com que a nuvem consiga encontrar os vizinhos mais próximos de um vetor cifrado.

3.2. Criptografia Homomórfica

Um esquema criptográfico é homomórfico para uma operação $\star$ se é equivalente executar esta operação sobre textos claros ou sobre mensagens cifradas. Por exemplo, considerando a operação de adição, a soma de dois criptogramas cifrados por um esquema homomórfico gera uma terceira mensagem cifrada que corresponde à soma dos dois textos claros correspondentes.

O número de operações que podem ser realizadas *em sequência* sobre textos cifrados é geralmente limitada. Depois de aplicar mais operações do que o limite suportado pelo esquema, o texto cifrado não pode ser corretamente decifrado. Por exemplo, se o esquema é homomórfico para a multiplicação e suporta apenas dois produtos em sequência, então, dadas as mensagens cifradas c_1, c_2, c_3, e c_4, é possível avaliar produtos da forma $c_i \cdot c_j$ e $c_i \cdot c_j \cdot c_l$ (para $i, j, l \in \{1, 2, 3, 4\}$), mas não $c_1 \cdot c_2 \cdot c_3 \cdot c_4$, pois este último emprega três produtos em sequência.

Tais esquemas de criptografia são chamados de *parcialmente homomórficos* (SHE[5]). Esquemas totalmente homomórficos (FHE[6]) atuais não têm tais limitações, mas introduzem uma penalidade de desempenho tão grande que torna o seu uso proibitivo na prática [Naehrig et al. 2011].

Neste trabalho, o esquema SHE utilizado foi o YASHE, que pode ser descrito em termos das seguintes funções:

HE. $\mathrm{keyGen}(\mathbf{X})$ é o algoritmo que recebe o conjunto de parâmetros X e devolve a chave secreta sk e a chave pública pk.

HE. $\mathrm{enc}(\mathrm{pk}, m)$ é o algoritmo que cifra uma mensagem m usando a chave pública pk e devolve um texto cifrado c.

HE. $\mathrm{dec}(\mathrm{sk}, c)$ é o algoritmo que decifra um criptograma c usando a chave secreta sk e devolve o texto claro m.

HE. $\mathrm{add}(c_1, c_2)$ é o algoritmo que soma dois textos cifrados c_1 e c_2, devolvendo um terceiro texto cifrado que corresponde à cifração da soma $m_1 + m_2$.

[5]Do inglês *Somewhat Homomorphic Encryption*.

[6]Do inglês *Fully Homomorphic Encryption*.

$\mathrm{HE.prod}(c_1, c_2)$ é o algoritmo que multiplica dois textos cifrados c_1 e c_2 e devolve um terceiro criptograma que corresponde à cifração do produto $m_1 \cdot m_2$.

O algoritmo $\mathrm{HE.keyGen}$ recebe um parâmetro de segurança λ; dois inteiros t e n que determinam o espaço de mensagens, i.e. o anel $R = \mathbb{Z}_t[x]/\langle\Phi_n(x)\rangle$, onde $\Phi_n(x)$ é o n-ésimo polinômio ciclotomico; duas distribuições de probabilidade discretas e limitadas χ_e e χ_k sobre R; e dois inteiros positivos q e w. O anel R pode ser visto como um conjunto de polinômios de grau menor ou igual ao grau de $\Phi_n(x)$ e com coeficientes em $\mathbb{Z}_t = \{0, 1, ..., t-1\}$. As distribuições χ_e e χ_k são usadas para amostragem de ruídos (polinômios com coeficientes pequenos) que são usados na geração de chave e na cifração. A distribuição Gaussiana discreta é normalmente escolhida para a χ_e e uma distribuição sobre polinômios com coeficientes pequenos (normalmente, coeficientes em $\{-1, 0, 1\}$) é escolhida para χ_k. Há também um parâmetro implícito L, a profundidade multiplicativa suportada pelo esquema. Ao fixar L e t, pode-se derivar os outros parâmetros, pois eles têm que respeitar algumas restrições para garantir a corretude e segurança do esquema.

4. k-NN não-interativo sobre dados cifrados

A versão privativa proposta para o k-NN pode ser dividida em quatro rotinas principais: inicialização, requisição, processamento e resposta.

- Codificação: essa função recebe a classe de uma instância $l \in \mathbb{N}$ e devolve o monômio x^l.
- Inicialização: o cliente tem n vetores $u_1, .., u_n \in \mathbb{R}^p$ e suas respectivas classes $l_1, ..., l_p \in \mathbb{N}$. Ou seja, cada vetor u_i é rotulado com um inteiro não negativo l_i. Esta sub-rotina cifra os vetores u_i usando o esquema OPE e codifica cada classe como um monômio para então cifrar usando o esquema de HE, conforme mostrado no Algoritmo 1. Note que esta etapa de inicialização é executada pelo cliente.

Algorithm 1 Inicialização

Entrada: $(u_1, .., u_n) \in \mathbb{R}^{n \times p}$, $(l_1, ..., l_n) \in \mathbb{N}^n$

$K = \mathrm{OPE.keyGen}(M, N, s)$
$(\mathrm{sk}, \mathrm{pk}) = \mathrm{HE.keyGen}(\text{ parâmetros })$
for $i = 1$ **to** n **do**
  $v_i = \mathrm{OPE.enc}(K, u_i)$
  $p(x) = \mathrm{encode}(l_i)$
  $c_i = \mathrm{HE.enc}(\mathrm{pk}, p(x))$
end for
Enviar $(v_1, v_2, .., v_n)$ e $(c_1, c_2, .., c_n)$ para o servidor na nuvem.

- Requisição: o cliente tem um vetor $w \in \mathbb{R}^p$ para ser classificado. Este vetor é cifrado utilizando o esquema OPE e então submetido para um servidor na nuvem.
- Processamento: A nuvem classifica um vetor cifrado y usando os vetores cifrados $v_1, v_2, .., v_n$ e suas respectivas classes (também cifradas) executando o Algoritmo 3. A função `indexesMinDistances` devolve os índices das k menores distâncias. Por exemplo, se o primeiro, o terceiro e o sétimo vetores fossem os

Algorithm 2 Requisição

Entrada: $w \in \mathbb{R}^p$

$y = \text{OPE.enc}(w)$
Escolher algum $k \in \mathbb{N}^*$
Enviar y e k para o servidor na nuvem.

três vizinhos mais próximos de y, então $d_1, d_3,$ e d_7 seriam as menores distâncias e essa função devolveria $(1, 3, 7)$.

Algorithm 3 Processamento

Entrada: vetor cifrado y, $k \in \mathbb{N}^*$

for $i = 1$ to n **do**
  $d_i = ||v_i - y||$
end for
$(i_1, .., i_k) =$ indexesMinDistances$(d_1, .., d_n)$
$class_y = c_{i_1}$
for $j = 2$ to k **do**
  $l = i_j$
  $class_y = \text{HE.add}(class_y, c_l)$
end for
Retornar $class_y$

- Resposta: o cliente recebe um texto cifrado $class_y$, decifra-o e extrai a classe do vetor y:

Algorithm 4 Resposta

Entrada: $class_y$

$c(x) = \text{HE.dec}(\text{sk}, class_y)$
$a_i =$ maxCoefficient$(c(x))$
Atribuir a classe i

A função funciona corretamente porque quando o servidor soma as classes cifradas no Algoritmo 4, está de fato gerando um polinômio que diz quantas vezes cada classe apareceu entre os k vetores mais próximos, pois a i-ésima classe é representada por x^i e a soma das classes resulta em um polinômio da forma $a_1x^1 + a_2x^2 + ... + a_sx^s$, onde cada coeficiente a_i representa o número de vezes que a i-ésima classe apareceu. Além disso, como as classes são cifradas usando um esquema de criptografia homomórfica, a nuvem pode somar classes cifradas e o criptograma resultante será decifrado para a soma das classes, como esperado.

5. Resultados experimentais

A versão privativa proposta para o k-NN foi implementada usando o esquema OPE apresentado em [Boldyreva et al. 2009] e o esquema homomórfico chamado

Tabela 1. Conjuntos de dados usados, número de instâncias (amostras no conjunto) e de atributos (variáveis de interesse).

CONJUNTO DE DADOS	INSTÂNCIAS	ATRIBUTOS
IRIS	150	4
WINE	178	13
CLIMATE MODEL (CM)	540	18
CREDIT APPROVAL (CA)	690	15
ABALONE	4177	8
WALL-FOLLOWING ROBOT (WFR)	5456	24

YASHE [Bos et al. 2013]. Ficam indicadas ao leitor as implementações públicas disponíveis em [Lepoint and Naehrig 2014] e [Dowlin et al. 2015]. A versão privativa foi avaliada usando conjuntos de dados do repositório de bases de dados para aprendizagem de máquina, da UCI[7]. Os conjuntos de dados são descritos na Tabela 1.

Todos os testes foram executados em uma máquina equipada com um processador Intel Xeon 2.6GHz, 30GB de memória RAM e o sistema operacional GNU/Linux Debian 8.0. É importante ressaltar que o consumo de memória ficou abaixo de 1GB durante toda a coleta dos resultados experimentais. A versão proposta do k-NN privativo foi implementada em C++ e compilada usando o GCC 4.9.2 fornecido pelo sistema operacional. A *flag* de otimização `-O3` foi usada. Para comparação, o k-NN implementado na biblioteca *Scikit Learn* do Python[8] foi usado como o k-NN convencional.

Os experimentos consistiram no processamento dos conjuntos de dados usando o k-NN convencional e a versão sobre dados cifrados, e na comparação das acurácias de classificação resultantes. Os resultados estão resumidos na Tabela 2. A tabela tem cinco colunas que representam o número de vizinhos mais próximos considerado ($k \in \{1, 3, 5, 7, 9\}$). A fim de verificar como os parâmetros do esquema OPE podem influenciar na precisão do k-NN, os conjuntos de dados foram cifrados usando vários pares de valores (M, N) (lembre-se que M e N determinam os tamanhos dos espaços de texto claro e de texto cifrado, respectivamente). Como não foram observadas diferenças significativas para as diversas combinações de parâmetros, são apresentados apenas os os resultados das execuções usando $(M, N) = (2^{32}, 2^{40})$. Como esperado, a versão que preserva a privacidade conservou a precisão do classificador original para quase todas as amostras.

O criptosistema YASHE foi instanciado com $\lambda = 80$, $w = 64$ e fixando parâmetros L e t para gerar n e q como descrito em [Lepoint and Naehrig 2014]. Como não é necessário realizar multiplicações homomórficas t deve ser um número maior do que o número máximo de vizinhos a ser considerado, é preciso escolher $t > k$ para lidar com o caso em que os k vizinhos mais próximos têm a mesma classe. Nesta situação, o valor 1 será somado ao mesmo coeficiente de polinômio a ser devolvido como classe pelo Algoritmo 3, fazendo com que o coeficiente se torne zero, já que as operações no anel R são tomadas módulo t. Portanto, foram escolhidos os parâmetros do YASHE como $L = 0$ e $t = 10$, o que resultou em $n = 541$ e $q = 33554467$ (um primo com 25 bits).

[7]UCI Repository: `https://archive.ics.uci.edu/ml/`
[8]`http://scikit-learn.org`

Tabela 2. Comparação das precisões entre a versão convencional do k-NN (CONV) e a versão privativa (PRIV) instanciada usando $M = 2^{32}$ e $N = 2^{40}$ como parâmetros do esquema OPE. Nenhum perda significativa de precisão foi observada na versão privativa.

	$k=1$		$k=3$		$k=5$		$k=7$		$k=9$	
BASE	CONV	PRIV	CONV	PRIV	CONV	PRIV	CONV	PRIV	CONV	PRIV
IRIS	0.960	0.960	0.980	0.980	0.960	0.961	0.960	0.960	0.960	0.970
WINE	0.847	0.830	0.796	0.796	0.779	0.779	0.796	0.780	0.745	0.730
CM	0.895	0.896	0.928	0.928	0.934	0.934	0.923	0.923	0.917	0.913
CA	0.633	0.619	0.685	0.685	0.680	0.680	0.746	0.746	0.746	0.737
WFR	0.883	0.882	0.875	0.875	0.868	0.869	0.855	0.855	0.837	0.837
ABALONE	0.591	0.583	0.612	0.618	0.621	0.620	0.628	0.630	0.628	0.627

Tabela 3. Comparação dos tempos de execução, em milissegundos, para classificar uma instância usando $k = 3$

BASE	CONVENCIONAL	PRIVATIVO
IRIS	0.020	1.30838
WINE	0.017	1.85090
CLIMATE	0.032	6.92744
CREDIT	0.044	7.66866
ABALONE	0.168	57.0700
WFR	0.180	83.0032

Uma comparação dos tempos de execução para classificar uma única amostra, usando $k = 3$, é mostrada na Tabela 3. Ressaltamos que mudanças no valor de k têm pouco efeito sobre os tempos de execução. Os conjuntos de dados foram divididos em um conjunto de treinamento contendo $\frac{2}{3}$ dos dados e um conjunto teste com o $\frac{1}{3}$ restante. Depois, a versão privativa foi executada 10 vezes e o tempo médio para classificar o conjunto foi computado. Foi realizada a mesma experiência na implementação convencional.

Para as bases testadas, a versão convencional do k-NN foi no máximo cerca de 400 vezes mais rápida que a versão sobre dados cifrados. Mesmo assim, as soluções propostas neste trabalho ainda podem ser consideradas eficientes se comparadas com soluções interativas, pois os custos de comunicação entre nuvem e cliente podem introduzir latências muito maiores dos que os tempos de execução observados.

A Tabela 4 apresenta os tempos de execução para cifrar cada conjunto de dados, incluindo conjuntos de treinamento e de teste. Ou seja, esses tempos correspondem à execução do procedimento de inicialização e consulta, sem considerar o custo de enviar os vetores para a nuvem (última operação do Algoritmo 2). Cifrar os conjuntos de teste é muitas vezes mais rápido do que cifrar o conjunto de treinamento porque, neste passo, não é necessário usar o esquema HE, que é mais lento do que o OPE.

Os vetores cifrados são representados por vetores de inteiros cujas entradas têm $\log_2(N)$ bits e, devido à nossa escolha de parâmetros, as classes cifradas são representadas por polinômios de grau até 540, com cada coeficiente tendo 16 bits. O comprimento,

Tabela 4. Comparação dos tempos de execução em segundos para cifrar os conjuntos de dados.

BASE	TESTE	TREINAMENTO
IRIS	0.214	0.037
WINE	0.621	0.154
CLIMATE	11.27	24.91
CREDIT	0.553	1.91
ABALONE	1.36	7.43
WFR	6.618	16.56

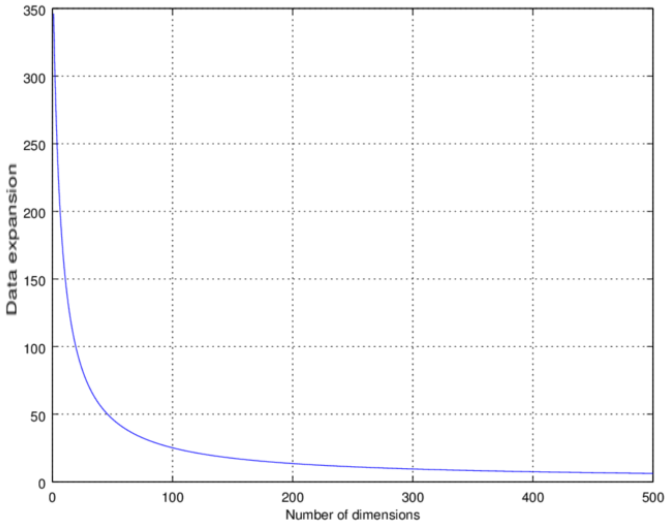


Figura 1. Expansão de dados como uma função do número de dimensões considerando os parâmetros M e N fixos

em bits, das entradas dos vetores originais (textos claros) é $\log_2(M)$ e as classes são representados por números inteiros. Portanto, considerando um conjunto de dados com n vetores P-dimensionais, a expansão de dados, definida como o tamanho máximo em bits dos dados cifrados sobre o tamanho máximo em bits de dados em texto claro é:

$$\frac{np\log_2 N + n540 \cdot 16}{np\log M + n32} = \frac{p\log_2 N + 8640}{p\log M + 32}$$

Note que o número de instâncias não afeta a expansão para armazenamento de dados e que os valores M e N têm pouco impacto na expansão, pois contribuem em escala logarítmica. Conforme o número de dimensões cresce, o quociente vai se aproximando de um. A Figura 1 mostra o efeito de variar p considerando $M = 2^{32}$ e $N = 2^{64}$ fixos.

6. Análise de Segurança

Assume-se neste trabalho que o servidor na nuvem satisfaz o modelo comumente chamado de *honesto, mas curioso* [Graepel et al. 2013], o que significa que o servidor

deve seguir o protocolo e executar o k-NN como esperado, devolvendo a resposta correta, mas pode tentar aprender informações durante a execução ou mais tarde tentar extrair informações dos criptogramas armazenados.

O servidor que executa o k-NN não aprende quais são os valores de qualquer coordenada de qualquer um dos vetores que ele recebe, incluindo os vetores de consulta, e também não aprende as classes associadas aos vetores. Como o esquema de criptografia homomórfica usado para cifrar as classes é não determinístico, o servidor não consegue determinar sequer quantas classes diferentes existem entre as classes cifradas. Como operações sobre mensagens cifradas resultam em outro texto cifrado bem formado, a classe atribuída ao vetor de consulta também não pode ser descoberta pelo servidor de nuvem. Por outro lado, a nuvem sabe o número de vetores e quantas entradas cada vetor tem por simples exame da quantidade de criptogramas recebidos.

Apesar disso, o esquema determinístico OPE introduz uma desvantagem: se a nuvem possui informações sobre a semântica das dimensões (qual variável é representada por cada componente dos vetores), e se também possui um conjunto de dados que está fortemente correlacionada com os dados cifrados, pode ser possível fazer ataques de inferência com base em análise de frequência [Naveed et al. 2015]. Por isso, nossas soluções são apropriadas apenas para a execução do k-NN sobre bases de dados privadas, sem informação pública sobre a distribuição dos seus dados.

7. Trabalhos relacionados

Vários trabalhos acadêmicos já estudaram problemas relacionados à execução do k-NN conservando a privacidade. No entanto, as soluções foram fornecidas para cenários diferentes, pois eram interativas, envolvendo processamento distribuído entre agentes com poder de computação similar; ou tratavam versões mais simples do problema, focando-se apenas na busca dos k vizinhos mais próximos e ignorando a etapa de classificação. Como resultado, a nossa proposta apresenta melhora qualitativa sobre esses trabalhos, fornecendo funcionalidades adicionais e garantias de privacidade correspondentes.

Os autores de [Zhan et al. 2005] consideraram um cenário conhecido como dados verticalmente particionados, em que cada uma de várias partes detém um conjunto de atributos das mesmas instâncias e querem realizar o k-NN na concatenação de seus conjuntos de dados. Eles propuseram um protocolo interativo no qual cada uma das partes precisa calcular as distâncias entre as instâncias em sua própria partição e um vetor de consulta e então combinar essas distâncias usando um esquema de criptografia homomórfica aditivo, com técnicas de perturbação aleatória, para encontrar os k vizinhos mais próximos. A etapa de classificação é finalmente executada localmente por cada uma das partes.

Em [Xiong et al. 2006, Xiong et al. 2007] os autores assumem que os proprietários de vários dados, cada um com uma base de dados própria, colaboram através da execução de um protocolo distribuído para executar o k-NN sem revelar uns aos outros os seus próprios dados. A classificação de uma nova instância é executada por cada usuário em seu próprio banco de dados e, em seguida, um protocolo distribuído seguro é usado para classificar a instância com base nos k vizinhos mais próximos de cada base de dados. Isso significa que o vetor de consulta é revelado e o processo é interativo, com carga de processamento para cada parte envolvida.

No artigo [Choi et al. 2014], os autores apresentam três métodos para encontrar os k vizinhos mais próximos preservando a privacidade dos dados, mas não abordam o problema de classificação. Além disso, os três métodos são interativos. É interessante notar que, mesmo que encontrar os k vizinhos mais próximos seja o passo principal do classificador k-NN, uma solução apenas para esse etapa implica em o cliente ter que armazenar, pelo menos, uma tabela que relacione os vetores no conjunto de dados e suas respectivas classificações, e implica também a necessidade de classificar os vetores de consulta localmente, depois de receber os k vizinhos mais próximos. Portanto, esse tipo de solução não é apropriada com o modelo de computação em nuvem.

O trabalho [Elmehdwi et al. 2014] considera um cenário diferente: o proprietário cifra os dados e os envia para um primeiro servidor, e envia a chave secreta para um segundo servidor. Assim, qualquer pessoa autorizada é capaz de enviar uma nova instância para o primeiro servidor, que executa um protocolo distribuído com o segundo servidor (este pode decifrar alguns dados nesse processo), e, finalmente, o primeiro servidor devolve os k vizinhos mais próximos. Observe que, novamente, a classificação não é executada, porque os autores estavam apenas interessados em encontrar os vizinhos mais próximos. Além disso, mesmo que o cliente não tenha que processar os dados, este método requer um servidor confiável para armazenar a chave privada, e esse servidor de confiança atua como o cliente no cenário de processamento distribuído. Basear-se em uma terceira parte confiável, naturalmente, apresenta um risco substancial adicional.

Os autores de [Zhu et al. 2013] propõem um cenário em que o proprietário dos dados os cifra e os envia para a nuvem, e então, outros usuários podem fazer requisições ao servidor em nuvem para obter, de forma segura, os vizinhos de vetores consulta arbitrários. A segurança do esquema é garantida graças a um protocolo interativo executado entre os usuários que querem realizar a consulta e o proprietário dos dados: esse protocolo gera uma chave usada para cifrar os vetores consulta e que não é apta a decifrar os vetores cifrados e armazenados na nuvem. Sendo assim, o proprietário dos dados ainda precisa participar da etapa de processamento, mesmo que seja apenas para gerar chaves, logo, esse protocolo pode ser classificado como interativo. Além disso, a classificação não é feita, apenas os vizinhos são encontrados.

Outra abordagem é proposta em [Wong et al. 2009], onde um novo esquema de criptografia chamado de criptografia que preserva o produto escalar é também proposto. Com esse esquema é possível encontrar os k vetores mais próximos sem a necessidade de um processo interativo, pois o servidor pode calcular produtos internos entre os vetores do conjunto de dados através do cálculo do produto interno dos vetores cifrados, determinando assim os vetores mais próximos do vetor de consulta. No entanto, novamente, os autores se preocuparam apenas com a tarefa de encontrar os vizinhos mais próximos, não com o problema de classificação. Além disso, um esquema de criptografia criado *ad hoc* para esta tarefa carece da extensa análise de segurança já realizada sobre sistemas criptográficos mais genéricos e bem estabelecidos.

8. Conclusão

Este trabalho apresenta uma versão não-interativa e privativa do classificador k-NN, e constatou-se por meio de avaliações experimentais que é suficientemente eficiente e precisa para ser viável na prática. O protocolo proposto combina criptografia homomórfica

e criptografia que preserva a ordem e é aplicável para a execução de classificações de instâncias usando dados cifrados armazenados na nuvem. Segundo consta, esta é a primeira proposta do classificador k-NN sobre dados cifrados que pode ser executado de uma forma não-interativa.

Como discutido na seção 3.2, o esquema criptográfico homomórfico empregado utiliza como espaço de textos claros um conjunto de polinômios, mas pode ser facilmente substituído por esquemas mais gerais, bastando apenas codificar cada classe i como um vetor binário com uma única entrada diferente de zero na posição i e adaptar os algoritmos para trabalhar com vetores ao invés de polinômios.

Além disso, se um cliente e um provedor de serviços em nuvem já utilizam qualquer protocolo para encontrar vizinhos mais próximos (por exemplo, usando outras primitivas criptográficas em vez de OPE ou executando algum algoritmo interativo), podem então usar um esquema homomórfico e as técnicas apresentadas aqui para obter uma classe a partir das outras classes.

Como trabalho futuro, possíveis melhorias para o k-NN aqui apresentado podem envolver técnicas de ofuscação de dados e de perturbação para obter propriedades de segurança mais fortes contra ataques de inferência, preservando ao mesmo tempo precisão e eficiência. Pretende-se também estender a versão apresentada aqui para uma versão do k-NN ponderada pelas distâncias.

Referências

Alpaydin, E. (2004). *Introduction to Machine Learning*. The MIT Press.

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185.

Boldyreva, A., Chenette, N., Lee, Y., and O'Neill, A. (2009). Order-Preserving Symmetric Encryption. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: The Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 224–241, Berlin, Heidelberg. Springer-Verlag.

Bos, J. W., Lauter, K., Loftus, J., and Naehrig, M. (2013). *Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme*, pages 45–64. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bost, R., Popa, R. A., Tu, S., and Goldwasser, S. (2014). Machine learning classification over encrypted data. Technical report, Cryptology ePrint Archive, Report 2014/331, 2014. http://eprint. iacr. org.

Choi, S., Ghinita, G., Lim, H.-S., and Bertino, E. (2014). Secure knn query processing in untrusted cloud environments. *Knowledge and Data Engineering, IEEE Transactions on*, 26(11):2818–2831.

Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. (2015). Manual for using homomorphic encryption for bioinformatics.

Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. (2016). Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. Technical Report MSR-TR-2016-3.

Elmehdwi, Y., Samanthula, B. K., and Jiang, W. (2014). Secure k-nearest neighbor query over encrypted data in outsourced environments. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 664–675. IEEE.

Graepel, T., Lauter, K., and Naehrig, M. (2013). Ml confidential: Machine learning on encrypted data. In *Information Security and Cryptology (ICISC)*, pages 1–21. Springer.

Hirt, M. and Sako, K. (2000). Efficient receipt-free voting based on homomorphic encryption. In *Advances in Cryptology – EUROCRYPT*, pages 539–556. Springer.

Jha, S., Kruger, L., and McDaniel, P. (2005). Privacy preserving clustering. In *Computer Security (ESORICS)*, pages 397–417. Springer.

Lepoint, T. and Naehrig, M. (2014). A comparison of the homomorphic encryption schemes FV and YASHE. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8469 LNCS:318–335.

Lindell, Y. and Pinkas, B. (2009). Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):5.

Miller, C. C. (2014). Revelations of N.S.A. spying cost U.S. tech companies. *The New York Times*. http://www.nytimes.com/2014/03/22/business/fallout-from-snowden-hurting-bottom-line-of-tech-companies.html (Acessed February 04, 2016).

Naehrig, M., Lauter, K., and Vaikuntanathan, V. (2011). Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud Computing Security*, pages 113–124. ACM.

Naveed, M., Kamara, S., and Wright, C. V. (2015). Inference Attacks on Property-Preserving Encrypted Databases. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*, pages 644–655.

Rivest, R. L., Adleman, L., and Dertouzos, M. L. (1978). On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180.

Wong, W. K., Cheung, D. W.-l., Kao, B., and Mamoulis, N. (2009). Secure kNN computation on encrypted databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 139–152. ACM.

Xiong, L., Chitti, S., and Liu, L. (2006). K nearest neighbor classification across multiple private databases. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 840–841.

Xiong, L., Chitti, S., and Liu, L. (2007). Mining multiple private databases using a knn classifier. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, pages 435–440, New York, NY, USA. ACM.

Zhan, J., Chang, L., and Matwin, S. (2005). Privacy preserving k-nearest neighbor classification. *International Journal of Network Security*.

Zhu, Y., Xu, R., and Takagi, T. (2013). Secure k-nn computation on encrypted cloud data without sharing key with query users. In *Proceedings of the 2013 International Workshop on Security in Cloud Computing*, Cloud Computing '13, pages 55–60, New York, NY, USA. ACM.

Agregação de Dados na Nuvem com Garantias de Segurança e Privacidade

Leandro Silva[1], **Rodolfo Silva**[1], **Andrey Brito**[1], **Pedro Barbosa**[1]

[1]Departamento de Sistemas e Computação – Universidade Federal de Campina Grande
58.429-140 – Campina Grande – PB – Brasil

leandro@ufcg.edu.br, rodolfomarinho@copin.ufcg.edu.br

andrey@computacao.ufcg.edu.br, pedroyossis@copin.ufcg.edu.br

Abstract. *The use of cloud computing has become common due to advantages such as low cost and sizing of computing resources according to demand. However, there are concerns about security and privacy, because critical data – especially in IoT applications – are stored and processed in the cloud. This paper proposes a software architecture that supports multiple approaches to secure data aggregation. The use of this architecture proved to be viable from experiments with the use of homomorphic encryption techniques and security extensions in hardware (Intel SGX), which, to our best knowledge, had never been used in a cloud environment.*

Resumo. *O uso da computação na nuvem tem se tornado comum por vantagens como baixo custo e contratação de recursos de acordo com a demanda. Todavia, surgem preocupações com segurança e privacidade, pois dados críticos – especialmente em aplicações de IoT – são armazenados e processados na nuvem. Este artigo propõe uma arquitetura de software com suporte a múltiplas abordagens para a agregação segura de dados. O uso dessa arquitetura se mostrou viável em experimentos realizados com a utilização de técnicas de criptografia homomórfica e de extensões de segurança em hardware (Intel SGX), que, segundo nossas pesquisas, ainda não havia sido aplicado em um ambiente de nuvem.*

1. Introdução

Computação na nuvem é um termo que representa serviços hospedados *on-line*. Tais serviços são acessíveis pela Internet, metaforicamente chamada de "nuvem" [Markovic et al. 2013]. Do ponto de vista empresarial, esse modelo de computação é muito atraente porque despesas com software, hardware e infraestrutura física são reduzidos drasticamente, já que a contratação de recursos computacionais é feita em grãos menores e a expansão ou redução destes recursos pode ser feita de forma automatizada de acordo com a demanda. Ainda, a operação da infraestrutura é delegada a um provedor que, por sua escala, tende a ser muito mais eficiente.

Apesar de todas as vantagens citadas, a computação na nuvem traz consigo preocupações com segurança e privacidade. Em uma pesquisa da *Cloud Industry Forum* [CIF 2015] para o Reino Unido, os dois principais inibidores para a adoção de computação na nuvem são as preocupações com segurança dos dados e privacidade, mencionados por 70% e 61% dos consultados, respectivamente. Uma vez que, na maioria dos

casos, os serviços na nuvem são fornecidos de forma compartilhada, ataques adicionais, tanto externos quanto internos podem ocorrer [Pasupuleti et al. 2016], como roubo de senhas de acesso ao serviço de nuvem e falhas de segurança na interface de programação (API) fornecida pelo provedor [Younis et al. 2013].

Em certas aplicações, o cuidado com a confidencialidade dos dados certamente precisará ser maior, que é o caso dos sistemas elétricos e redes inteligentes. Nesse contexto, dados sensíveis, como o gasto de energia de cada consumidor, devem ser tratados com cuidado, já que podem revelar muita informação sobre os consumidores, como seus padrões de comportamento, por exemplo, trechos do dia em que não há indivíduos na residência, horas de chegada e saída ou de repouso.

Este artigo propõe uma arquitetura de software para viabilizar o uso da computação na nuvem em aplicações com requisitos estritos de segurança e privacidade. Tal arquitetura considera como os componentes de uma aplicação de redes elétricas inteligentes (*smart grids*) podem ser integrados de forma que a privacidade e segurança dos dados dos usuários sejam garantidas. A parte sensível do processamento é então isolada e a arquitetura considera diferentes estratégias para agregação de dados sensíveis em ambientes onde não há garantia de plena confidencialidade.

Para ilustrar duas implementações do módulo de agregação segura, duas alternativas foram desenvolvidas, uma baseada em criptografia homomórfica, que é completamente programada em software, mas que impõe altos custos adicionais de processamento, e outra baseada na tecnologia Intel SGX [McKeen et al. 2013], que não impõe custos computacionais elevados, mas requer a disponibilidade de processadores com suporte a esta tecnologia nas máquinas onde forem executados os agregadores[1]. A primeira permite que computações sejam realizadas em cifrotextos sem comprometer a criptografia. Por exemplo, um sistema de busca homomórfico permite que termos sejam localizados sem que o computador tenha conhecimento sobre o que se deseja encontrar ou sobre a base de dados onde se realizou a procura. Já a segunda possibilita que processos possam ser executados em um modo protegido, onde toda a memória e execução são protegidas contra acessos, mesmo que de usuários ou processos com níveis maiores de privilégio, evitando a necessidade de realizar computações em cifrotextos.

Nos nossos experimentos, dois casos de uso foram abordados: o cálculo do consumo de energia em uma região e o cálculo da conta mensal de um consumidor. A estratégia de agregação com Intel SGX – pela primeira vez utilizada em uma plataforma de gerência de computação na nuvem – se mostrou muito mais eficiente do que a criptografia homomórfica. Todavia, como a criptografia homomórfica não possui exigências de hardware específico, essa estratégia pode ser viável para aplicações cujo foco não esteja no volume de dados.

Finalmente, certas limitações foram identificadas durante o desenvolvimento do trabalho, dentre elas: (i) para ser viável, o conjunto de computações para a criptografia homomórfica é limitado, o que dificulta ou inviabiliza o desenvolvimento de módulos de processamento de dados arbitrários; (ii) o uso de Intel SGX impede que certas operações sejam realizadas, como chamadas de sistema (*syscalls*), o que tem implicações no tipo de

[1]Alguns exemplos de processadores com suporte a Intel SGX são os processadores da sexta geração da família Intel Core e alguns processadores Xeon recentes, como os da família E3-1200.

código que executará de forma protegida; e, (iii) Intel SGX também possui limitações de uso de memória, $128\ MB$ na implementação atual, mas com possibilidade de páginação em uma versão lançada recentemente.

O restante do artigo é organizado da seguinte maneira. Os trabalhos relacionados são discutidos na Seção 2. A Seção 3 retrata a arquitetura de software elaborada para garantia de agregação segura de dados. Os experimentos e resultados são apresentados na Seção 4 e são discutidos na Seção 5. Por fim, a Seção 6 ressalta as principais conclusões do artigo, assim como suas limitações e possíveis trabalhos futuros.

2. Trabalhos Relacionados

Em [Reinhold et al. 2014] é descrita uma arquitetura híbrida onde parte dos componentes se encontra em uma nuvem privada e parte em uma nuvem pública. A parte de armazenamento dos dados fica em uma nuvem pública, de forma criptografada. Já a aplicação, com a lógica e processamento dos dados, se encontra em uma nuvem privada, onde há menores preocupações com segurança e privacidade. Somente os clientes têm acesso às chaves privadas, sendo assim, os dados só são tratados na sua forma pura quando o cliente está autenticado. Por fim, sempre que um dado precisa ser armazenado, ele passa por um servidor de criptografia, que fica na nuvem privada, e então é enviado para o servidor de armazenamento na nuvem pública.

[Bohli et al. 2013] traz um estudo com diferentes padrões de arquitetura para distribuição dos recursos em múltiplos provedores de serviços de computação na nuvem: replicação de aplicações, que permite enviar operações em distintas nuvens e comparar se os resultados são os mesmos; partição das camadas de uma aplicação em distintas nuvens, para separar, por exemplo, a lógica da aplicação em uma nuvem e o banco de dados em outra; partição da lógica da aplicação em diferentes nuvens e partição dos dados em nuvens distintas. Os autores concluem que não existe estratégia ótima para todos os casos, já que a implementação dos padrões sugeridos não é trivial e todas possuem falhas de segurança.

Possíveis usos de Intel SGX foram discutidos em [Hoekstra et al. 2013], onde foram apresentados exemplos de aplicações que fazem uso das capacidades de Intel SGX, bem como foi apresentada uma arquitetura de aplicações considerando uma partição da aplicação entre partes que demandam segurança, devendo ser executadas dentro de enclaves, e partes que não necessitam de segurança, que pode ser executado fora de enclaves.

[Barbosa et al. 2016] apresenta um conjunto de ferramentas para a construção de protocolos que ampliam as garantias de ambientes de execução isolados, como é o caso do Intel SGX, através do uso de sua capacidade de realizar atestação remota. Com isso, é possível estabelecer protocolos de troca de chave entre um participante remoto e um ambiente de execução isolado, de forma segura. Esses protocolos são definidos a partir da combinação entre um protocolo de troca de chaves passivamente seguro e o uso de um protocolo de atestação arbitrário.

Sobre computação segura, Rivest, Adleman e Dertouzous [Rivest et al. 1978a] criaram o conceito de criptografia homomórfica. Isto se deve ao fato do RSA, sistema criptográfico desenvolvido por Rivest, Shamir e Adleman [Rivest et al. 1978b], possuir um homomorfismo parcial, chamado de homomorfismo multiplicativo. Ou seja, com

o RSA é possível multiplicar dois valores criptografados e o resultado ainda será a multiplicação criptografada.

Modelos completamente homomórficos se caracterizam por permitir operações de adição e multiplicação em blocos criptografados, de modo que o valor retornado seja uma encriptação do resultado das operações aplicadas sobre os dados originais. Embora o conceito não seja recente, estes modelos foram considerados puramente teóricos até que [Gentry 2009] propuseram um sistema válido, usando reticulados ideais. Entretanto, questões relacionadas à eficiência ainda constituem uma barreira. Atualmente, todos os tipos de esquemas de criptografia completamente homomórfica propostos ainda possuem um longo caminho de evolução antes que sejam utilizados na prática [Naehrig et al. 2011].

3. Solução Proposta

A arquitetura de software proposta possui quatro tipos de componentes: barramentos de mensagens, produtores, agregadores e consumidores. Os barramentos de mensagens são responsáveis pela comunicação entre os produtores, agregadores e consumidores dos dados. Após serem produzidos, esses dados serão publicados no barramento de mensagens e, em algum momento, serão consumido e tratados por agregadores, que podem realizar operações arbitrárias, mas que devem ser capazes de executá-las de forma segura. Posteriormente, os dados agregados serão consumidos por aplicações. Um esquema ilustrativo da arquitetura pode ser visto a seguir na Figura 1.

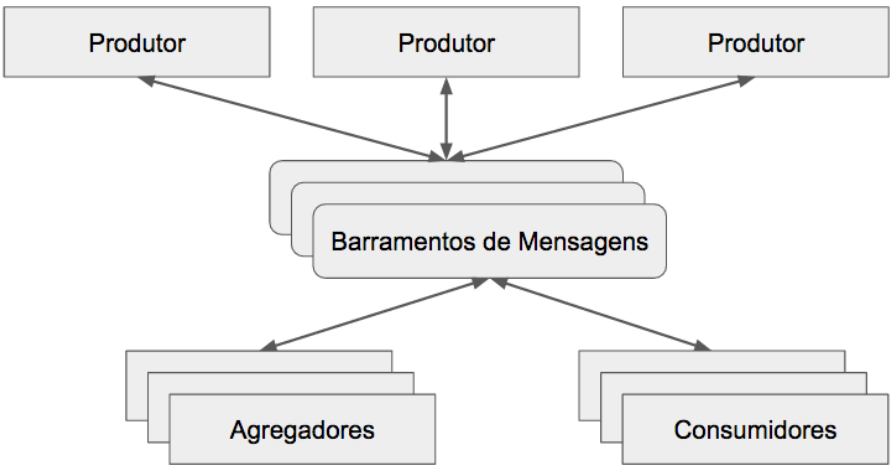


Figura 1. Esquema simplificado da arquitetura de software proposta

É importante salientar que todos os componentes ilustrados seguem interfaces bem definidas, ou seja, é possível utilizar diferentes tipos de produtores, barramentos de mensagens, agregadores ou consumidores desde que eles implementem corretamente a especificação. As seções a seguir detalham os principais componentes dessa arquitetura.

3.1. Barramentos de mensagens

Um barramento de mensagem é responsável pela troca de informações entre produtores, agregadores e consumidores de forma transparente, ou seja, um produtor pode, por exemplo, criar mensagens sem saber maiores detalhes, como localização física ou endereço IP,

sobre os agregadores. Com isso, é possível reduzir o acoplamento e garantir escalabilidade.

Cada componente da arquitetura deve se inscrever em um tópico de mensagens. Todos os agregadores ou consumidores de um tópico receberão as mensagens enviadas pelos produtores. Ademais, é permitido o consumo exclusivo de novas mensagens ou de todas as mensagens retidas de um certo tópico. É valido frisar também que qualquer parte pode enviar mensagens em um tópico a fim de requisitar dados, de realizar passos intermediários em uma ação e outros. Por fim, a quantidade de mensagens armazenadas para um tópico pode ser configurável.

Por ser crítico na comunicação entre todos os envolvidos, esse componente permite a troca segura de mensagens. Cada envolvido pode possuir um certificado emitido por uma unidade certificadora, como forma de autenticação, evitando que intrusos comecem a enviar ou receber mensagens. Além disso, mensagens com conteúdo sensível devem estar criptografadas. É papel do agregador, descrito a seguir, transformar informação sensível e criptografada, como consumo instantâneo de uma residência, em informação agregada e em purotexto, como o consumo em um intervalo maior de tempo ou o consumo de uma região.

3.2. Agregadores

Após o consumo dos dados de um tópico comum, é necessário realizar operações nos mesmos. Tais operações, como, por exemplo, soma, multiplicação ou agrupamento são realizadas pelos agregadores. Dois agregadores são exemplificados na nossa implementação da arquitetura: um agregador **homomórfico** e um agregador baseado em **Intel SGX**. Ambos são projetados para realizar **operações de soma** e serão detalhados nas seções a seguir.

3.2.1. Agregador Homomórfico

Esse agregador faz uso da criptografia homomórfica para realizar operações de forma segura e privada nos dados. A abordagem utilizada por esse componente é baseada no esquema proposto em [Busom et al. 2016], que faz uso do sistema de criptografia de ElGamal [ElGamal 1984], possuidor da propriedade homomórfica multiplicativa mas com possibilidade de adição, como pode ser visto em [Cramer et al. 1997]. Além disso, é possível combinar com um esquema de criptografia de limiar [Saroj et al. 2015], em que há necessidade da colaboração de todas as partes envolvidas para decriptar cifrotextos.

Para que esse tipo de agregação funcione é necessário que cada produtor $p \in [1, n]$ de um tópico comum possua os seguintes itens:

- Um número primo grande q (de pelo menos 2048 bits) e um gerador g de ordem q do grupo multiplicativo G de $\mathbb{Z}_q^*$;
- Uma chave privada x_p;
- Uma chave pública $y_p = g^{x_p}$;
- Um certificado $cert_p$ a ser validado por uma unidade certificadora;

A fim de permitir o envio das informações é obrigatória uma fase de configuração sempre que novos produtores se inscreverem em um tópico de mensagens. O procedimento é descrito a seguir:

1. O agregador envia uma mensagem de requerimento de configuração;
2. Cada produtor envia y_p e $cert_p$ ao tópico;
3. O agregador verifica a validade de cada $cert_p$ e insere uma mensagem com $\{y_1, ..., y_n\}$ e $\{cert_1, ..., cert_n\}$ no tópico;
4. Cada produtor verifica a validade de cada $cert_p$ e calcula uma chave pública global $y = \prod_{p=1}^{n} y_p$.

Os seguintes passos serão realizados a cada instante de tempo que for necessária uma coleta de dados para agregação:

1. O agregador envia uma mensagem de requerimento de dados ou, alternativamente, os produtores podem iniciar uma transmissão periodicamente;
2. Cada produtor p gera um número aleatório $z_p \in \mathbb{Z}_q^*$ e calcula $C_p = E_y(g^{v_p + z_p}) = (c_p, d_p)$, em que v_p representa o valor coletado por p e a função E_y é a função de criptografia de ElGamal;
3. Todos os valores C_p são publicados no tópico do barramento de mensagens associado àquele produtor (por exemplo, a região em que o medidor está instalado);
4. O agregador realiza sua computação: $C = \left(\prod_{p=1}^{n} c_p, \prod_{p=1}^{n} d_p \right)$ e insere C no tópico;
5. Cada produtor calcula $T_p = c^{x_p}.g^{z_p}$ e publica-o no tópico;
6. O agregador pode, então, obter $D = d.\left(\prod_{p=1}^{n} T_p \right)^{-1}$, em que $d = \prod_{p=1}^{n} d_p$;
7. Por fim, é possível obter $V = \sum_{p=1}^{n} v_p$ ao calcular $\log_g D$;
8. O agregador publica o resultado obtido no mesmo ou em outro tópico do barramento de mensagens, de forma que ele fique disponível para possíveis consumidores.

3.2.2. Agregador Intel SGX

O agregador Intel SGX faz uso da tecnologia de mesmo nome para prover segurança e privacidade dos dados sensíveis sendo agregados, através do uso de áreas protegidas de memória (**enclaves**), inacessíveis até mesmo por usuários com mais privilégios. Em nossa implementação, usamos o algoritmo de criptografia simétrica *AES Galois/Counter Mode* (AES-GCM) descrito em [Dworkin 2007], com chave de tamanho de 128 bits, para a troca de mensagens confidenciais entre os produtores e o agregador. Este algoritmo permite verificar a autenticidade dos dados confidenciais recebidos, através da detecção de modificações não intencionais ou modificações intencionais não autorizadas feitas aos dados, bem como torna a comunicação imune a ataques do tipo *side-channel* baseados em *software*.

Ao receber dados confidenciais encriptados, o agregador pode decriptar a mensagem usando o mesmo algoritmo AES-GCM, e após isso realizar a agregação sobre os dados confidenciais decriptados. A segurança e privacidade dos dados decriptados é alcançada através das garantias providas pelos enclaves SGX [McKeen et al. 2013].

As seguintes condições precisam ser satisfeitas para o correto funcionamento deste agregador:

- Cada produtor $p \in [1, n]$ de um tópico possui uma chave simétrica k_p;
- O agregador conhece de antemão cada uma das k_p chaves;

Para que o agregador conheça de antemão cada uma das k_p chaves, é necessária uma troca de chaves através de um protocolo seguro, como o provido pelo próprio Intel SGX, descrito em [Anati et al. 2013], onde os produtores possam atestar que estão se comunicando com o agregador correto. A atestação também será usada para validar o agregador, por exemplo, para assegurar que este não vazará informações (por exemplo, permitindo agregações com poucas medições criptografadas).

Para agregar uma coleta de dados feita em um instante de tempo, os seguintes passos serão realizados:

1. O agregador envia uma mensagem de requerimento de dados ou, alternativamente, os produtores podem iniciar uma transmissão periodicamente;
2. Cada produtor p cria um valor aleatório (***nonce***) n_p, e calcula $C_p, M_p = G_e(v_p, n_p, k_p)$, onde C_p é o valor coletado por p após a encriptação AES-GCM, M_p é o código de autenticação de mensagem de v_p, onde v_p é o valor medido por p, e a função G_e é a função de criptografia AES-GCM no modo de encriptação.
3. Cada produtor publica C_p, M_p, e n_p ao tópico;
4. O agregador obtém $V = \sum_{p=1}^{n} v_p$ ao calcular $\sum_{p=1}^{n} G_d(C_p, n_p, k_p, M_p)$, onde a função G_d é a função AES-GCM no modo decriptação, ao mesmo tempo em que verifica a integridade de cada uma das mensagens recebidas no tópico;
5. O agregador publica o resultado obtido no mesmo ou em outro tópico do barramento de mensagens, de forma que ele fique disponível para possíveis consumidores.

4. Avaliação da arquitetura proposta

Com a finalidade de avaliar a solução proposta, dois casos de uso foram escolhidos para a implementação de provas de conceito para futuros experimentos. Ambos fazem parte do contexto de redes elétricas inteligentes e exigem cautela com relação à confidencialidade dos dados. Os detalhes da implementação das duas aplicações serão discutidos na seção 4.3.

4.1. Cálculo de consumo de energia em regiões

O crescimento das necessidades de recursos de energia elétrica motivou tanto o governo quanto a indústria a buscar formas alternativas de prover energia e, principalmente, de aperfeiçoar o gerenciamento da rede elétrica. Por outro lado, aumentar a eficiência e balancear a malha energética não é uma tarefa trivial. Para isso, uma opção é utilizar medidores inteligentes (*smart meters*) que podem, periodicamente, medir e reportar o consumo energético [Erkin and Tsudik 2012].

Como dito anteriormente, a medição periódica do consumo energético causa preocupações com a privacidade dos consumidores, já que é possível inferir informações pessoais a partir do que é coletado, como tipos de equipamentos na residência, assim

como a presença e o número de moradores [Anderson and Fuloria 2010]. Em casos mais extremos, é possível identificar o canal de televisão sendo assistido [Greveler et al. 2012].

Diante do exposto, uma alternativa para fornecer informações cruciais para o balanceamento energético e, ao mesmo tempo, manter a privacidade dos consumidores é agregar os dados de consumo em grupos de residências, ou regiões.

4.2. Cálculo da conta mensal dos consumidores

O segundo caso de uso consiste em calcular, de forma segura e privada, a conta mensal de cada consumidor a partir de um conjunto de dados de consumo em intervalos curtos. Esta abordagem permite que a concessionária emita as faturas no intervalo de cobrança e que ela ou o consumidor possam usufruir de certos benefícios da medição detalhada (por exemplo, calculo do consumo instantâneo da região para concessionária ou visualização local do consumo instantâneo pelo consumidor) sem que haja risco de que os dados detalhados sejam usados pela concessionária ou parceiros para inferir hábitos do consumidor.

4.3. Detalhes técnicos das provas de conceito

Com o intuito de executar experimentos, as provas de conceito foram implementadas seguindo a arquitetura explicada na Seção 3. A Figura 2 ilustra a esquemática das aplicações desenvolvidas.

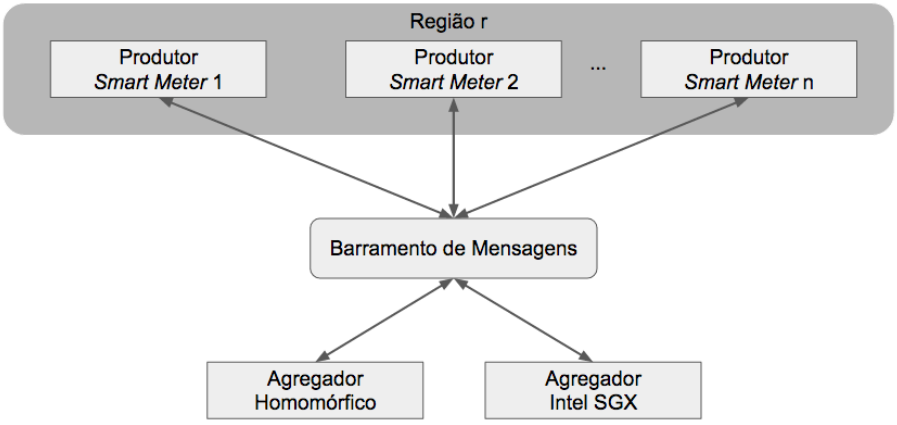


Figura 2. Esquema da aplicação para cálculo de consumo de energia por região

Dentre os componentes acima, os medidores inteligentes (produtores) e os agregadores foram desenvolvidos durante a pesquisa realizada. Para a comunicação entre produtores e agregadores/consumidores, a solução Apache Kafka[2] foi escolhida por prover todos os serviços exigidos para o barramento de mensagens, inclusive a troca segura de mensagens, permitindo a autenticação dos envolvidos e o uso de canais seguros de comunicação.

No experimento realizado, cada medidor inteligente é simulado por uma *thread* em Java que gera valores de consumo e publica no barramento de mensagens. Todos os medidores de uma região r estão inscritos no mesmo tópico do barramento de mensagens. O agregador da região r é um processo em execução que também está inscrito no tópico

[2]http://kafka.apache.org

em questão e é responsável por agrupar as medições para cada instante de tempo t e, quando os dados de consumo do conjunto de medidores da região são coletados para um instante t, é possível calcular a soma desses valores.

Para o cálculo da conta mensal com o agregador homomórfico, também desenvolvido em Java, cada medidor faz uso de k pares de chaves públicas e privadas, um par para cada medição, com um reaproveitamento de forma circular, de acordo com o módulo do identificador da medição. Por fim, a cada fim de ciclo, o processo de agregação é executado para aquele intervalo de tempo.

No caso do agregador Intel SGX, apenas uma chave simétrica fixa, previamente compartilhada com o agregador, é usada por cada cada medidor, além de um *nonce* gerado aleatoriamente à cada medição. Para efeito de simplificação, a troca de chaves de forma segura entre medidores e agregador foi abstraída, e pode ser considerada um problema resolvido na fase de atestação, como mostrado em [Barbosa et al. 2016].

Finalmente, as duas abordagens são robustas contra situações onde o agregador publica resultados agregados mas que permitem a inferência de valores instantâneos. Na abordagem homomórfica, o produtor participa do processo de agregação e pode identificar situações onde as agregações usam poucos dados ou possuem intervalos com sobreposição. Na abordagem Intel SGX, o produtor validou o código do atestador inicialmente e este não pode ser atualizado sem uma nova atestação.

4.4. Experimentos e Resultados

Após a implementação das provas de conceito, as mesmas foram submetidas a experimentos com o objetivo de mensurar e comparar os tempos de resposta para realizar a agregação com criptografia homomórfica e com Intel SGX, bem como sem o uso dessas duas abordagens (sem garantias de segurança e privacidade)

Os testes foram realizados em um ambiente de nuvem privada utilizando OpenStack para orquestração da aplicação na nuvem com contêineres Docker (usando o driver *nova-docker* do OpenStack). Os contêineres executavam Ubuntu Linux 14.04 e os drivers do Intel SGX[3]. De acordo com nossas pesquisas, este é o primeiro trabalho a aplicar uma estratégia com Intel SGX em ambientes de computação na nuvem. Na Figura 3 é possível visualizar como estão dispostos os componentes para o experimento.

Dois conjuntos de testes foram efetuados. O primeiro foi realizado para comparar as agregações utilizando Intel SGX e criptografia homomórfica. Nele, para cada caso de uso, 10 repetições foram realizadas por agregador para regiões com 10, 50, 100 e 200 medidores, resultando em 160 execuções. O segundo teve como objetivo comparar agregações com Intel SGX a agregações totalmente sem criptografia, a fim de avaliar o custo do uso dessa nova tecnologia. Neste segundo conjunto, a mesma quantidade de repetições foi feita para regiões com 200, 400, 800 e 1000 medidores.

As médias dos tempos de resposta podem ser lidos nas Tabelas 1 e 2 e visualizados nas Figuras 4 e 5. Para o consumo por região, o tempo exibido é relativo ao cálculo do consumo da região inteira para um instante de tempo t. Já para a conta mensal, esse é o tempo levado para o cálculo do consumo do mês inteiro com medições a cada 15 minutos.

[3]https://01.org/intel-softwareguard-eXtensions (desde 24 de junho de 2016).

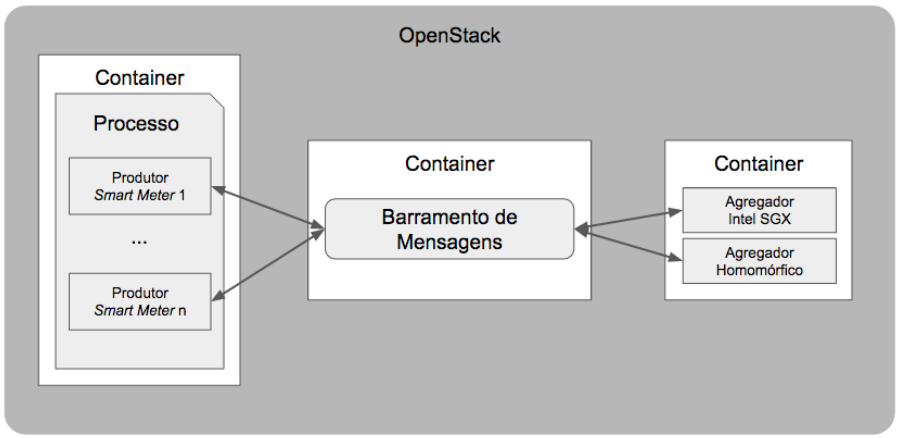


Figura 3. Topologia dos componentes para os experimentos

Caso de Uso	Agregador	$n = 10$	$n = 50$	$n = 100$	$n = 200$
Consumo por região	Homomórfico	362 ms	1812 ms	3755 ms	7596 ms
Consumo por região	Intel SGX	116 ms	117 ms	119 ms	121 ms
Conta mensal	Homomórfico	669 s	3167 s	6220 s	12305 s
Conta mensal	Intel SGX	70 s	74 s	78 s	84 s

Tabela 1. Médias dos tempos de resposta obtidos para comparação entre Intel SGX e criptografia homomórfica

Para os tempos acima, foram considerados os tempos totais, incluindo as comunicações entre os componentes (na rede local), assim como os tempos gastos para criptografia dos dados pelos produtores e descriptografia pelos agregadores Intel SGX.

5. Discussão

Como pode ser observado nos gráficos e na tabela da seção anterior, a abordagem de agregação utilizando Intel SGX possui tempos de resposta bem menores que a agregação com criptografia homomórfica. Não obstante, cada forma de agregação fornece vantagens e desvantagens que serão discutidas a seguir.

A criptografia homomórfica, embora mais lenta, possui vantagens que a torna viável em certas aplicações. Não há necessidade de hardware específico para sua execução, tornando-a facilmente implantável em vários ambientes, além disso, pode ser implementada em diversas linguagens, já que sua base é puramente matemática. Dependendo do tamanho da massa de dados a ser computada, o custo em termos de tempo pode ser irrisório diante dos benefícios de privacidade e segurança que essa abordagem traz consigo, já que todas as computações são realizadas em cifrotextos, fazendo com que o conteúdo que é processado seja desconhecido ao provedor de serviços e a quaisquer atacantes em potencial.

Outra vantagem visível na abordagem utilizada para a agregação homomórfica é que o consumidor dos dados não consegue inferir as medições de um consumidor específico, porque seria necessário que todos os outros consumidores fossem corruptos. Isso é característico da criptografia de limiar.

Caso de Uso	Agregador	$n = 200$	$n = 400$	$n = 800$	$n = 1000$
Consumo por região	Sem criptografia	116 ms	119 ms	119 ms	121 ms
Consumo por região	Intel SGX	122 ms	125 ms	134 ms	139 ms
Conta mensal	Sem criptografia	73 s	74 s	74 s	78 s
Conta mensal	Intel SGX	84 s	96 s	115 s	126 s

Tabela 2. Médias dos tempos de resposta obtidos para comparação entre agregação com Intel SGX e nenhuma criptografia

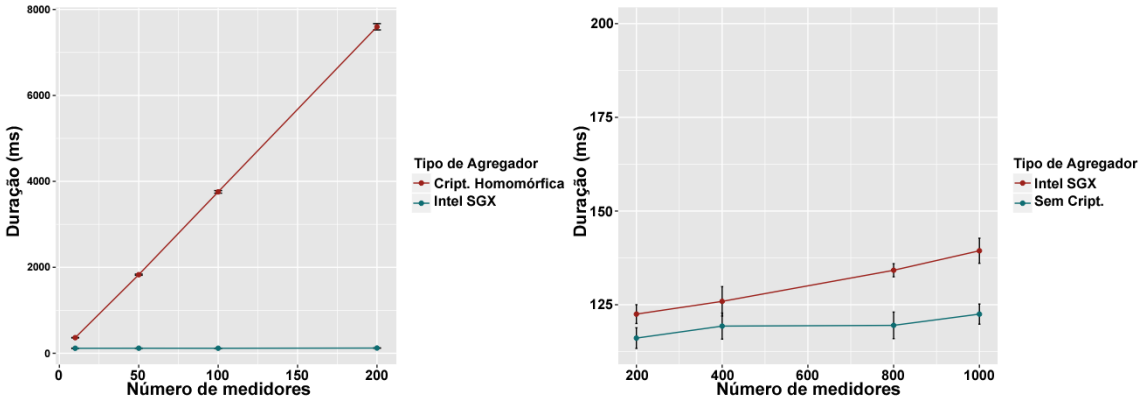


Figura 4. Gráficos do experimento realizado para o cálculo do consumo de uma região

Por outro lado, a técnica homomórfica possui uma limitação relacionada às operações que podem ser realizadas em cifrotextos, já que a maioria dos algoritmos de criptografia são parcialmente homomórficos, e, abordagens que permitem computações arbitrárias em cifrotextos, como pode ser visto em [Gentry 2009], ainda possuem tempos de processamento muito altos.

Outro fator negativo que foi observado na agregação homomórfica é que há necessidade da troca de mensagens adicionais para que o consumo regional seja computado. Isso não é trivialmente implantável, já que o envio da mensagem da concessionária para os medidores obriga que um medidor possua um endereço acessível ou que ele utilize uma estratégia de *polling* para verificar periodicamente se há novas mensagens, como as da segunda fase do algoritmo.

O Intel SGX, por sua vez, possui uma alta performance e baixo custo adicional, se comparado ao uso de criptografia homomórfica, por não precisar realizar computações sempre sobre cifrotextos, ao mesmo tempo que provê garantias de segurança e privacidade de dados de usuários, sem que haja um grande esforço para implementar aplicações que usem a tecnologia, já que o Intel SGX possui bibliotecas que dão suporte à criação, atestação e uso de enclaves. Dessa forma, é interessante o seu uso quando na presença de *hardware* com o devido suporte, e houver necessidade velocidade de execução e segurança e privacidade dos dados envolvidos.

Apesar disso, ainda há algumas questões em aberto quanto à completa integração entre a solução Intel SGX e a filosofia do ambiente de computação na nuvem. Enquanto as soluções de computação na nuvem baseiam-se no desacoplamente entre o hardware físico e as aplicações, por exemplo, através de virtualização do hardware, Intel SGX vincula a

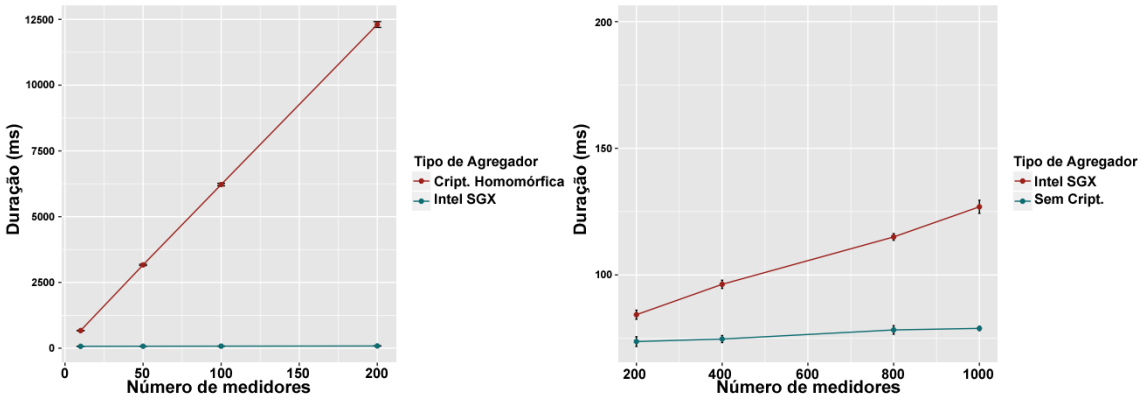


Figura 5. Gráficos do experimento realizado para o cálculo da conta mensal

segurança da aplicação diretamente ao processo de atestação, que, por enquanto, exige a participação da Intel e seu serviço de atestação.

Um outro ponto negativo quanto ao uso de Intel SGX é a limitação quanto à API suportada atualmente, que exige o desenvolvimento de aplicações em C/C++, e a impossibilidade, por motivos de segurança, de executar chamadas de sistema em código executado dentro de enclaves, o que pode dificultar a portabilidade de aplicações para o uso da tecnologia. Por fim, um outro ponto que deve ser levado em conta em qualquer aplicação que faça uso de Intel SGX é o do atual limite de apenas $128\ MB$ de memória que pode usado para a criação e execução de enclaves por *hardware* hospedeiro. Paginação pode ser usada para criptografar a memória antes de exportá-la para fora do processador, mas isso impõe cargas adicionais.

6. Conclusões

Requisitos como segurança e privacidade não devem ser ignorados quando aplicações com dados sigilosos fazem uso da computação na nuvem. Nesse artigo, descrevemos uma arquitetura de software para endereçar tais requisitos. Essa arquitetura permite o uso de diferentes estratégias para a agregação de dados privados. Tais estratégias incluem o uso de criptografia homomórfica ou de tecnologias como Intel SGX.

Para a avaliação, identificamos dois casos de uso em que houvesse preocupações com os requisitos citados. De posse dos casos de uso, foram desenvolvidas provas de conceito que ajudaram a identificar as vantagens e desvantagens de cada forma de agregação. Para criptografia homomórfica, a principal vantagem identificada foi a implantação viável em quaisquer ambientes, embora seja menos eficiente. Já o Intel SGX, que foi pela primeira vez utilizado em um orquestrador de computação na nuvem, possui tempos de resposta bem menores e permite realizar diversas formas de computação sobre os dados, mas exige uma infraestrutura específica do provedor de serviços.

Para trabalhos futuros, almeja-se utilizar a criptografia de ElGamal com outros grupos, como, por exemplo, curvas elípticas, assim como o uso de *garbled circuits* [Kolesnikov and Schneider 2008] e também computação segura multi-parte [Cramer et al. 2000]. Usando Intel SGX, deseja-se propor soluções mais completas, que envolvam a implementação de atestação remota [Barbosa et al. 2016] de forma amigável em um ambiente de nuvem.

7. Agradecimentos

Este trabalho foi parcialmente financiado pelo projeto EU-BRA SecureCloud (MCTI/RNP 3a Chamada Coordenada) e pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Referências

Anati, I., Gueron, S., Johnson, S., and Scarlata, V. (2013). Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, volume 13.

Anderson, R. and Fuloria, S. (2010). On the security economics of electricity metering. In *WEIS*. Citeseer.

Barbosa, M., Portela, B., Scerri, G., and Warinschi, B. (2016). Foundations of hardware-based attested computation and application to sgx. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 245–260. IEEE.

Bohli, J.-M., Gruschka, N., Jensen, M., Iacono, L. L., and Marnau, N. (2013). Security and privacy-enhancing multicloud architectures. *IEEE Transactions on dependable and secure computing*, 10(4):212–224.

Busom, N., Petrlic, R., Sebé, F., Sorge, C., and Valls, M. (2016). Efficient smart metering based on homomorphic encryption. *Computer Communications*, 82:95–101.

CIF (2015). Uk cloud adoption snapshot & trends for 2016: The business case for cloud.

Cramer, R., Damgård, I., and Maurer, U. (2000). General secure multi-party computation from any linear secret-sharing scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 316–334. Springer.

Cramer, R., Gennaro, R., and Schoenmakers, B. (1997). A secure and optimally efficient multi-authority election scheme. *European transactions on Telecommunications*, 8(5):481–490.

Dworkin, M. J. (2007). Sp 800-38d. recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. Technical report, Gaithersburg, MD, United States.

ElGamal, T. (1984). A public key cryptosystem and a signature scheme based on discrete logarithms. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 10–18. Springer.

Erkin, Z. and Tsudik, G. (2012). Private computation of spatial and temporal power consumption with smart meters. *Proceedings of the 10th international conference on Applied Cryptography and Network Security*, pages 561–577.

Gentry, C. (2009). *A fully homomorphic encryption scheme*. PhD thesis, Stanford University.

Greveler, U., Glösekötterz, P., Justusy, B., and Loehr, D. (2012). Multimedia content identification through smart meter power usage profiles. In *Proceedings of the International Conference on Information and Knowledge Engineering (IKE)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Hoekstra, M., Lal, R., Pappachan, P., Phegade, V., and Del Cuvillo, J. (2013). Using innovative instructions to create trustworthy software solutions. In *HASP@ ISCA*, page 11.

Kolesnikov, V. and Schneider, T. (2008). Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer.

Markovic, D. S., Zivkovic, D., Branovic, I., Popovic, R., and Cvetkovic, D. (2013). Smart power grid and cloud computing. *Renewable and Sustainable Energy Reviews*, 24:566–577.

McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C. V., Shafi, H., Shanbhogue, V., and Savagaonkar, U. R. (2013). Innovative instructions and software model for isolated execution. In *HASP@ ISCA*, page 10.

Naehrig, M., Lauter, K., and Vaikuntanathan, V. (2011). Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124. ACM.

Pasupuleti, S. K., Ramalingam, S., and Buyya, R. (2016). An efficient and secure privacy-preserving approach for outsourced data of resource constrained mobile devices in cloud computing. *Journal of Network and Computer Applications*, 64:12–22.

Reinhold, P., Benn, W., Krause, B., Goetz, F., and Labudde, D. (2014). Hybrid cloud architecture for software-as-a-service provider to achieve higher privacy and decrease security concerns about cloud computing. In *Conf. Cloud Computing, GRIDs, and Virtualization (IEEE, 2014)*, pages 94–99. Citeseer.

Rivest, R. L., Adleman, L., and Dertouzos, M. L. (1978a). On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180.

Rivest, R. L., Shamir, A., and Adleman, L. (1978b). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Saroj, S. K., Chauhan, S. K., Sharma, A. K., and Vats, S. (2015). Threshold cryptography based data security in cloud computing. In *Computational Intelligence & Communication Technology (CICT), 2015 IEEE International Conference on*, pages 202–207. IEEE.

Younis, Y. A., Merabti, M., and Kifayat, K. (2013). Secure Cloud Computing for Critical Infrastructure : A Survey.

Armazenamento Seguro de Credenciais e Atributos de Usuários em Federação de *Clouds*

Luciano Barreto[1], Leomar Scheunemann[1] , Joni da Silva Fraga[1], Frank Siqueira[2]

[1]Departamento de Automação e Sistemas – (DAS)
[2]Departamento de Informática e Estatística – (INE)
Universidade Federal de Santa Catarina

{luciano.barreto,leomar.scheunemann}@posgrad.ufsc.br,{joni.fraga,frank.siqueira@ufsc.br}

Abstract. *The use of cloud computing and cloud federations has been the focus of studies. Many of these infrastructures delegate their authentication to Identity Providers. Once these services are available through the Internet, concerns about the confidentiality of user credentials and attributes are high. The main focus of this work is the security of the credentials and user attributes in authentication infrastructure, exploring secrets sharing techniques and using clouds federation as a base for the storage of this information.*

Resumo. *O uso de provedores de cloud e formações de federações tem sido foco de estudos há alguns anos. Muitas destas infraestruturas delegam as autenticações de seus usuários a Provedores de Identidades. Por se tratarem de serviços disponíveis através da Internet, a preocupação com o sigilo das credenciais e atributos de usuários é constante. O foco principal deste trabalho é a segurança das credenciais e atributos de usuários em infraestruturas de autenticação, explorando técnicas de compartilhamento de segredos e utilizando federações de clouds como base para o armazenamento destas informações.*

1. Introdução

O agrupamento de provedores de serviços de *cloud* em associações tem sido apresentado na literatura como solução para o compartilhamento de recursos ociosos. Vários termos têm sido usados para caracterizar estas associações de provedores de *clouds*: federação de *clouds*, *Inter-cloud* [Grozev and Buyya 2012], *Multi-cloud* [Grozev and Buyya 2012] e *Cross-Cloud* [Celesti et al. 2010]. Embora as diferenças de nomenclatura e mesmo que muitas das propostas para a formação destas redes de confiança entre *clouds* tenham diferentes focos, o objetivo final é sempre o uso de serviços de diversos provedores de *clouds* para atender a demanda de clientes e os *SLAs* (*Service-Level Agreement*) acordados. Uma tendência em sistemas distribuídos de larga escala como *clouds,* federações de *clouds*, grades computacionais e redes colaborativas em geral é o uso de infraestruturas que realizem o gerenciamento de identidades [Jøsang et al. 2005]. Estas infraestruturas normalmente integram políticas e tecnologias, permitindo às diferentes organizações que participam destes grandes sistemas terem aplicações que ultrapassem seus domínios locais (domínios administrativos ou de políticas), perfeitamente credenciadas e autorizadas pelas diferentes políticas locais.

Vários modelos de gerenciamento de identidades são identificados na literatura (dentre os quais destacamos o centralizado, identidades federadas e *user-centric* [Jøsang

et al. 2005]) e, em muitos destes modelos, a autenticação não é mais realizada nos provedores de serviços (*SPs*), mas sim em autoridades independentes e confiáveis de autenticação, chamadas de Provedores de Identidades. Estes Provedores de Identidades (autoridades de autenticação) na verdade passaram a ser os pontos centrais na aplicação de políticas de segurança em sistemas distribuídos. Mas, também por serem serviços que ficam disponíveis via Internet, estes *IdPs* estão sujeitos a ataques que podem resultar em intrusões, o que seria catastrófico para a segurança das informações e recursos.

O objetivo deste artigo é apresentar uma abordagem que faz uso de provedores de *cloud* para armazenar as informações de usuário (credenciais e atributos) e vem como uma proposta de solução para a autenticação em federações de provedores de *cloud*, mas não restrita a somente estes ambientes. O uso de *clouds* no armazenamento destas informações de usuário exige um conjunto de mecanismos para fornecer segurança e disponibilidade destas informações, mesmo considerando a falta de controle absoluto sobre estes recursos e serviços delegados. Mas a vantagem desta abordagem está na flexibilidade do acesso às informações permitindo que usuários possam fazer os seus *logins* a partir de um *IdP* de uma federação de *clouds* e ter acessos a recursos em diferentes partes desta federação. O restante deste texto está organizado da seguinte forma: na seção 2 apresentamos o modelo de sistema, na seção 3 a arquitetura proposta. A seção 4 foca nos protocolos e os resultados da abordagem são apresentados na seção 5. Por fim, na seção 6 é apresentada a literatura relacionada e na seção 7 as conclusões do artigo.

2. Modelo de Sistema

2.1 Caracterização de Federação de *Clouds*

O agrupamento de provedores de *clouds* através de redes de confiança formadas com o objetivo de atender um vasto número de usuários coloca também grandes desafios na autenticação e autorização destes usuários junto aos provedores nestas federações. A Figura 1 ilustra uma destas associações que visam o compartilhamento de recursos. Nesta figura, explicitamos basicamente o aspecto dos controles de autenticação, centrando sempre estas funções em provedores de identidades (*IdPs*).

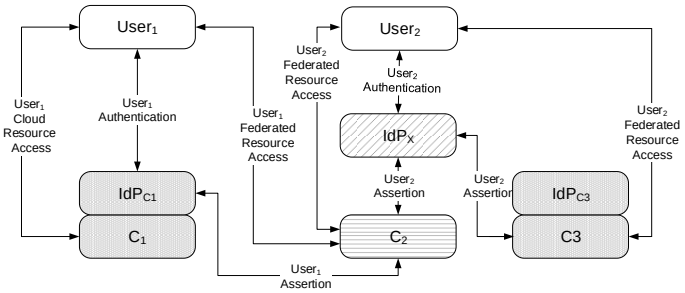


Figura 1 – Autenticação e Autorização em Federação de Clouds

Os provedores de *clouds* (*Clouds* C_1, C_2 e C_3) da Figura 1 apresentam formas diferentes em seus controles de autenticação. Os provedores C_1 e C_3 possuem *IdPs* internos ($IdPc_1$ e $IdPc_3$) para executar suas funções de autenticação e a *cloud* C_2, ao contrário, delega suas autenticações de usuários a um *IdP* externo (IdP_X) com o qual mantém relações de confiança. Considerando o arranjo citado, o usuário $User_1$ é mostrado efetuando a sua autenticação junto ao *IdP* de C_1 ($IdPc_1$). Este usuário pode só ter conta

na *cloud* $Cloud_{C1}$ e, diante disto, uma vez autenticado tem acesso a recursos do provedor C_1. Como uma federação de *clouds* é construída sobre relações de confiança, este mesmo usuário ($User_1$) tem também acesso em recursos de C_2, uma vez que esta última confia nas autenticações de C_1. Ou seja, os dois provedores de *cloud* (C_1 e C_2) mantêm relações de confiança. Nesta mesma figura, $User_2$ se autentica em IdP_X (um *IdP* externo que C_2 confia) e com isto acessa recursos de duas *clouds* da federação (C_2 e C_3).

2.2 Armazenamento de Credenciais e Atributos de Usuário

Nossa proposta está baseada na existência de diversos *IdPs*. Porém, diferentemente das abordagens convencionais, onde os atributos de usuários são armazenados localmente, em nossa proposta estes atributos não são armazenados diretamente nos *IdPs*, ou seja, as *clouds* não mantêm em seus *IdPs* as informações de seus próprios clientes. A ideia é dispor estas informações, de forma segura em uma base de dados construída sobre recursos distribuídos pela federação.

O armazenamento seguro em *clouds* tem como solução usual a encriptação dos dados, o que garante a confidencialidade dos mesmos. As chaves criptográficas usadas nestas encriptações são protegidas normalmente com o uso de técnicas de compartilhamento de segredos [Schoenmakers 1999; Shamir 1979]. As informações criptografadas podem ser replicadas e armazenadas em recursos de várias *clouds* de modo a garantir a disponibilidade das mesmas em ambientes onde podem ocorrer ações maliciosas ou mesmo falhas. Porém, existem soluções com um menor custo computacional que fazem uso de técnicas de dispersão de informação (*Information Dispersal Algorithms*: *IDA* [Rabin 1989]), baseadas em códigos de correção de erro (*erasure codes)*. Estas transformações *IDA* particionam os dados de uma informação em um conjunto de n partes (ou blocos) de tal maneira que t partes (com $t<n$) são suficientes para que se recupere a informação original. Com $t-1$ ou menos partes os dados, como um todo, não podem ser recuperados. No entanto, o uso destas técnicas de recuperação de erros não garante a confidencialidade das informações; é necessário a encriptação das mesmas.

Em um esquema de criptografia de limiar, chamado de compartilhamento de segredo, a partir de uma transformação, um segredo S dá origem a n *shadows* (partes). Com isto, cada uma destas partes do segredo não permite qualquer inferência sobre o segredo S. Esta informação só pode ser reconstruída a partir da obtenção de um conjunto mínimo de t partes ($0 < t \leq n$). Desta forma, com um conjunto de k partes do segredo, com $k < t$, não se consegue recuperar informação alguma de S. Os primeiros esquemas de compartilhamento de segredo propostos na literatura [Shamir 1979] definiam compartilhamentos de segredo baseados em interpolação polinomial e geometria plana (intersecção de retas). Porém, alterações destas partes poderiam acabar por comprometer a recuperação de S. Para lidar com cenários onde as partes estão sob a ação de entidades maliciosas que agem ativamente contra os protocolos de restauração dos segredos, foram criados os mecanismos de compartilhamento de segredo verificável [Schoenmakers 1999], em que cada fragmento de um segredo fornecido a um combinador pode ter sua validade verificada.

Vários sistemas, que por armazenarem arquivos em *clouds* baseados em transformações *IDA* e seus códigos de correção de erro, precisam que seus arquivos de informações sejam criptografados para garantir a confidencialidade dos mesmos. As

chaves usadas na encriptação das informações também precisam ser protegidas e neste caso são aplicados esquemas de compartilhamento de segredo. No nosso esquema, as informações (credenciais e atributos de usuários) a serem armazenadas tem um tamanho pequeno (poucos *bytes)* e não se torna proibitivo a aplicação de esquemas de compartilhamento de segredo diretamente nestas informações.

Na nossa abordagem, portanto, aplicamos somente um esquema de compartilhamento de segredo sobre as informações de usuário que decompõe as mesmas em *n* partes. Estas partes são então distribuídas entre *n* diferentes entidades (uma por cada uma das *n* entre as *m clouds* do sistema). As informações são recuperadas a partir de no mínimo *t* partes que estão armazenadas em *n* das *m clouds* do sistema, onde $0 < t < n < m$. Nas experimentações, usamos dois esquemas de compartilhamento de segredo: um não verificável [Shamir 1979] e um verificável [Schoenmakers 1999].

2.3 Premissas Assumidas

O fato de colocarmos os nossos dados pessoais em recursos gerenciados por terceiros envolve uma preocupação com a confidencialidade e a integridade destes dados. Alguns autores assumem que provedores de *cloud* mantêm um comportamento honesto, mas curioso (*honest-but-curious*). Este tipo de comportamento limita as ações de entidades de uma *cloud* a apenas leitura de informações em dados armazenados nos recursos da *cloud*. Em nossa abordagem, assumimos premissas menos restritivas. Cada provedor de *cloud* pode ser confiável, mas entidades maliciosas podem agir internamente devido à grande disponibilidade dos recursos de uma *cloud* a acessos externos ou mesmo internos. Estas entidades podem formar conluios para quebrar segredos. Os segredos dos usuários são mantidos, em nossa abordagem, quando o número de *clouds* sofrendo intrusões e falhas no sistema não ultrapassa o limiar *f* que definimos como sendo de valor *t-1* ($f = t - 1$). Nesta situação nossos protocolos funcionam corretamente e os segredos das informações não são quebrados e podem ser seguramente transformados.

3. Arquitetura Proposta

A arquitetura para a autenticação em uma federação de *clouds* é ilustrada na Figura 2. Os provedores de identidades (*IdP*), para implementarem os serviços de autenticação, usam as especificações *OpenID Connect* e, como não armazenam os atributos dos usuários, ficam reduzidos a poucas funcionalidades. Os protocolos *OpenID* trabalham sobre uma *cache.* A recuperação das informações de usuário diretamente das diversas *clouds* apresenta restrições de desempenho devido a questões de rede e ao próprio custo dos algoritmos envolvidos. A *cache* é então justificada pela manutenção dos atributos e credenciais de usuários mais recentes e frequentes, evitando sempre que possível a execução dos protocolos de combinação e a recuperação das informações de usuários.

Os atributos e credenciais de usuários que são inseridos nesta *cache* são recuperados da base de dados distribuída na federação de *clouds*. Para tanto, nos *IdPs*, são necessárias as funcionalidades da camada *Secret Sharing* (Figura 2), cujos algoritmos executam a função de combinar as partes dos segredos espalhados pelas *clouds* e, a partir desta combinação, tornam disponível os atributos de usuários na *cache*. Para que esta camada de compartilhamento de segredos (*Secret Sharing*) possa obter as partes de um segredo referente a um usuário na federação de *clouds*, é necessário o uso de *client stubs*

dos diversos bancos de dados usados no armazenamento dos mesmos na federação de *clouds*. Na Figura 2, as *stubs* são apresentadas como parte da camada *Database Client*.

A camada *Account Management* (Figura 2) permite que registros com credenciais e atributos de usuários sejam criados ou alterados em base de dados distribuída na federação. O *Account Management* faz uso mais amplo da camada *Secret Sharing*. Além da combinação de partes de um segredo, algoritmos de geração de partes de um segredo são fornecidos nesta camada. Por sua vez, a camada *Database Client* fornece os acessos (*read*/*write*) aos serviços de base de dados das diversas *clouds* da federação no armazenamento e na recuperação das informações de usuários durante a criação de suas contas. A Figura 2 mostra ainda os provedores de *cloud* da federação. Nas demanda de recursos, as verificações de autorização são baseadas em asserções de autenticação recebidas de *IdPs* da federação. Com isto, as *clouds* devem manter as listas dos *IdPs* com os quais possuem relações de confiança e seus respectivos certificados. Baseadas nestas listas, estas *clouds* podem validar as asserções de autenticação emitidas pelos *IdPs* da federação. Em relação ao sistema de autenticação, estes provedores de *cloud* tornam disponíveis os serviços de base de dados que armazenam as partes de segredos referentes às credenciais e atributos de usuários.

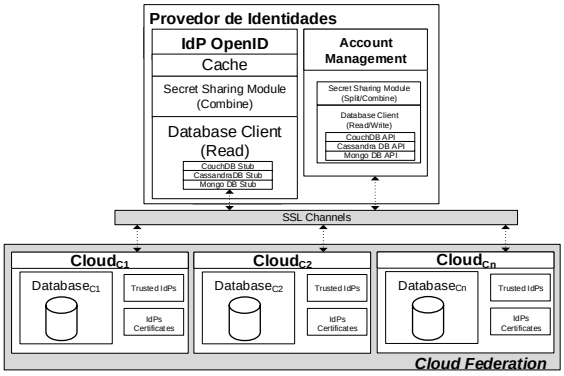


Figura 2 – Organização do Sistema de Autenticação

4. Protocolos da Abordagem

Assumimos duas possibilidades na criação de contas: na primeira, a interface *Account Management* é acionada por pessoal credenciado da *cloud* e é necessária a presença física do usuário; na alternativa, o próprio usuário faz o seu registro através da Internet.

Na criação de conta com a exigência de presença física, o cadastro de usuários é efetuado segundo os conceitos de segurança definidos em [Burr et al. 2006], onde a validade da identidade assegurada é de nível *LoA 4* (*Level of Assurance 4*). Para este nível, o usuário deve comparecer junto ao agente credenciado pela *cloud*, com os documentos necessários para garantir a veracidade de suas informações em seu registro. O agente então, com base nestes documentos, introduz os dados do usuário no sistema. Na criação de contas por acesso remoto à interface da camada *Account Management*, o usuário deve fornecer um certificado de chave pública assinado por autoridade certificadora oficial (*AC* de uma *PKI* oficial), ou ainda, por uma *AC* reconhecida pelos provedores de *cloud*. Nesta alternativa, o usuário fornece ou altera seus atributos e credenciais, devidamente assinados por sua chave privada (*LoA 3*). Uma descrição do protocolo de criação de conta de usuário (segundo o nível *LoA 3*) é ilustrada na Figura 3.

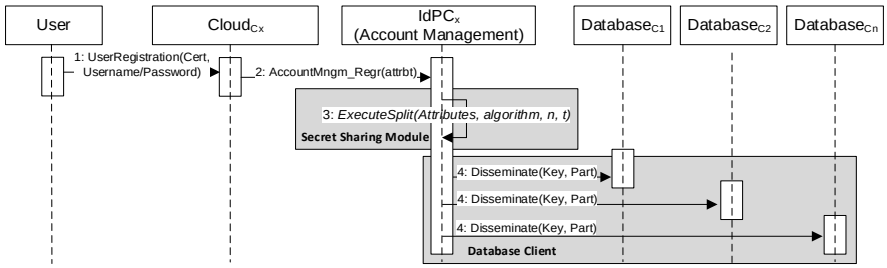


Figura 3 – Registro de Usuários

Em um primeiro passo, o usuário preenche o formulário de criação de conta através de um *IdP* da federação, introduzindo suas informações para o seu registro no sistema (passo 1, na Figura 3). Os dados passam por uma validação local (verificação da correção do certificado do usuário e também das assinaturas sobre os atributos passados pelo usuário). Uma vez validadas estas informações de usuário, as mesmas são passadas à camada *Secret Sharing* para que sejam geradas as partes do segredo e disseminadas entre as *clouds*, na base de dados distribuída que vai alimentar os *IdPs* do sistema. Este processo envolve algumas definições como: a escolha do esquema de compartilhamento de segredo, o número de partes do segredo e o limiar (número mínimo de partes) para recuperação dos segredos, dentre outros (passo 2). Após a geração das partes do segredo a partir das informações do usuário (passo 3), a distribuição destas se dá através das *stubs* apropriadas da camada *Database Client* (passos 4 na Figura 3).

Outro protocolo de nossa abordagem é descrito pelo processo de autenticação dos usuários do sistema, apresentado na Figura 4. Este procedimento é iniciado quando o usuário acessa um provedor de *cloud* para requisitar recursos (passo 1, Figura 4) que, por sua vez, verifica se este usuário mantém alguma sessão de autenticação já válida. Caso não tenha, a *cloud* estabelece comunicação com o *IdP* (passo 2). Se este *IdP* possui asserção de autenticação do usuário, no caso do mesmo já estar autenticado e acessando recursos em outra *cloud* (passo 3), a asserção correspondente é então enviada ao provedor de *cloud* requisitante (passo 4). Caso esta asserção de autenticação também não exista no *IdP* contatado, o usuário é direcionado para este provedor de identidades para que sua autenticação se concretize. O processo de autenticação é iniciado com o *IdP* solicitando ao usuário suas credenciais de autenticação (passo 5). Após a inserção destas credenciais (passo 6), o *IdP* faz consulta para verificar se as informações de *login* do usuário estão em sua *cache* (passo 7).

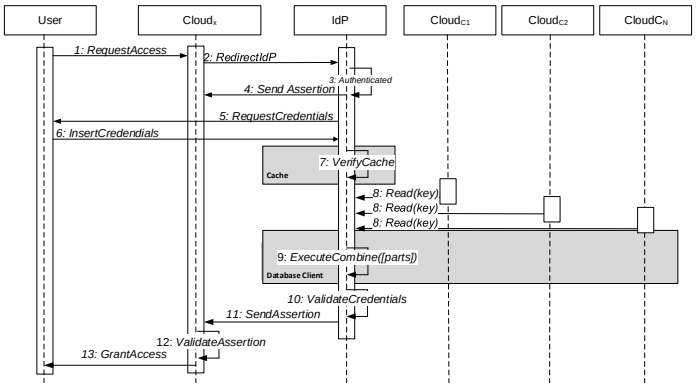


Figura 4 – Autenticação de Usuário

Se estas informações estão presentes, o *IdP* confronta com as credenciais fornecidas, validando o usuário com a geração de uma asserção (passo 10). Caso não estejam disponíveis na *cache*, o *IdP* deve buscar as informações do usuário na base de dados distribuída entre *clouds* da federação (passo 8). A busca de informações do usuário na federação faz uso da camada *Database Client* para a recuperação das partes de segredo que foram distribuídas entre *clouds* da federação. Com *t* partes do mesmo segredo, a camada *Secret Sharing* faz a combinação e recupera as credenciais e atributos do usuário correspondente, tornado disponíveis as mesmas na cache do *IdP* (passo 9).

De posse destas informações, o *IdP* faz as validações necessárias sobre as credenciais informadas pelo usuário (Passo 10) e em caso afirmativo envia para o provedor de *cloud* uma cópia da asserção gerada no processo de autenticação (passo 10). A Figura 4 mostra também a recepção da asserção de autenticação pelo provedor de *cloud* que efetua a validação da mesma com as verificações dos *nonces*, data de validade da asserção e assinaturas (passo 12). É importante ressaltar que o *IdP* em questão deve estar na lista de *IdPs* confiáveis do provedor de *cloud* (possui o certificado de chave pública do *IdP*). Com a asserção válida, o provedor de *Cloud* inicia o tratamento da requisição do usuário e finaliza o processo de autenticação (Passo 13).

5. Protótipo e Resultados

O sistema desenvolvido permite a autenticação de usuários e a geração de asserções de autenticações, que podem ser utilizadas por provedores de *cloud* de toda a federação, como forma de atestar a autenticidade do usuário. A Figura 5 descreve uma visão das ferramentas utilizadas na implementação do protótipo do sistema de autenticação. Na camada *Secret Sharing* que provê as funcionalidades de compartilhamento de segredos, foram implementados os esquemas de *Shamir* e *PVSS*. Os testes foram automatizados através da ferramenta *Apache JMeter*, de onde foram retirados os resultados apresentados nos gráficos das próximas seções.

O serviço provê somente a autenticação de usuários, pois o registro de usuário, como citado no texto, é executado por extensão (a interface *Account Management*), em pilha de protocolos própria a partir dos provedores de identidades. Os atributos e credenciais dos usuários registrados no sistema são armazenados em diferentes provedores de *cloud* que oferecem serviços de base de dados. Atualmente utilizamos serviços de bases de dados gratuitas, sem garantias de seus serviços (não garantem a qualidade do canal de comunicação, a disponibilidade das informações, etc.). Porém, para testar a viabilidade da proposta estes serviços de bases de dados se mostraram suficientes.

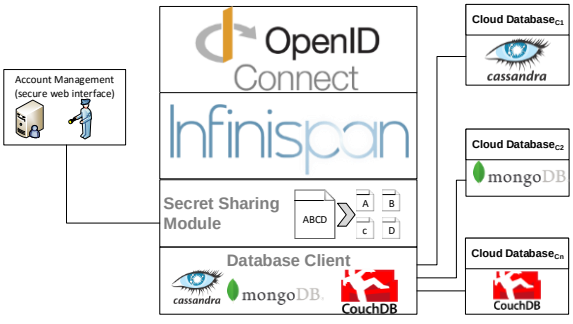


Figura 5 – Tecnologias utilizadas

5.1 Testes e Resultados

Os testes realizados com o protótipo do sistema de autenticação proposto envolveram as operações de registro de usuários, autenticação de usuários e geração das asserções de autenticação correspondentes. Para a realização destes testes, 1000 registros de usuários contendo as credenciais (usuário e senha) e atributos (nome, sobrenome, endereço, data de nascimento, profissão, etc.) foram gerados e armazenados utilizando os dois algoritmos de compartilhamento de segredos. Diferentes parâmetros *n* e *t* foram utilizados (representados nos gráficos como *Algorithm Parameters*).

5.1.1 Testes sobre o Tempo de Registro de Usuários

Neste teste foi medido o tempo necessário para a geração das partes de segredo pelos algoritmos de compartilhamento e também no armazenamento dos dados resultantes nas diversas *clouds* da federação, representando o registro de usuários através da interface *Account Management*. A Figura 6 apresenta os tempos de registro das implementações dos esquemas de *Shamir* e do *PVSS*, respectivamente.

Diante dos resultados apresentados na Figura 6 podemos observar que o tempo de processamento na geração das partes de segredo (operação de *Split* na camada *Secret Sharing*) para o algoritmo *PVSS* é menor se comparado com o algoritmo *Shamir*. Em ambos os casos, o tempo de processamento está relacionado com o método empregado para se encontrar o valor do módulo a ser usado nas operações (aritmética modular) dos esquemas de segredo. No *PVSS* este módulo é definido pelo tamanho da chave criptográfica usada (512 *bits* em nossa simulação), enquanto no método de *Shamir* o mesmo é assumido como o tamanho dos segredos considerados (em nossa simulação os segredos são de aproximadamente 256 *bytes*).

O tempo de envio dos dados para as bases de dados é ligeiramente menor quando utilizadas as implementações do esquema de *Shamir*. Isto se deve ao fato de que a quantidade de dados gerados por este algoritmo é menor que a quantidade de dados gerados no esquema *PVSS*. São enviadas com o *PVSS* mais informações para o armazenamento, como as provas para verificação da correção das partes do segredo.

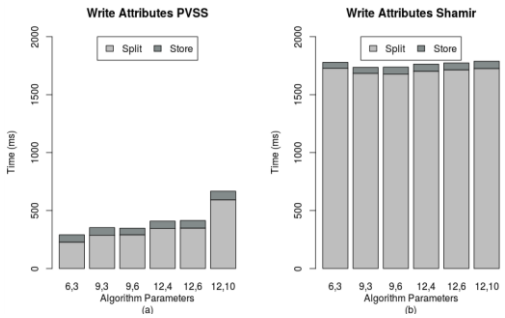


Figura 6 – Tempos de Registro

5.1.2 Testes sobre o Tempo de Consulta (Tempo de Autenticação)

Os tempos de consulta (referente ao tempo correspondente a execução de *querying* por dados pelo *IdP* e a disponibilidade dos mesmos em *cache*) corresponde praticamente ao tempo para a autenticação de um usuário no sistema (desconsiderando o tempo de verificação em *cache*). Na obtenção destes tempos, consideramos primeiro uma pequena

carga de utilização (10 usuários simultâneos) e, posteriormente, uma carga de utilização bem mais significativa (100 usuários simultâneos). A Figura 7 apresenta os tempos de consulta envolvendo também as diferentes configurações de protocolos de compartilhamento de segredo (o *PVSS* e o *Shamir*).

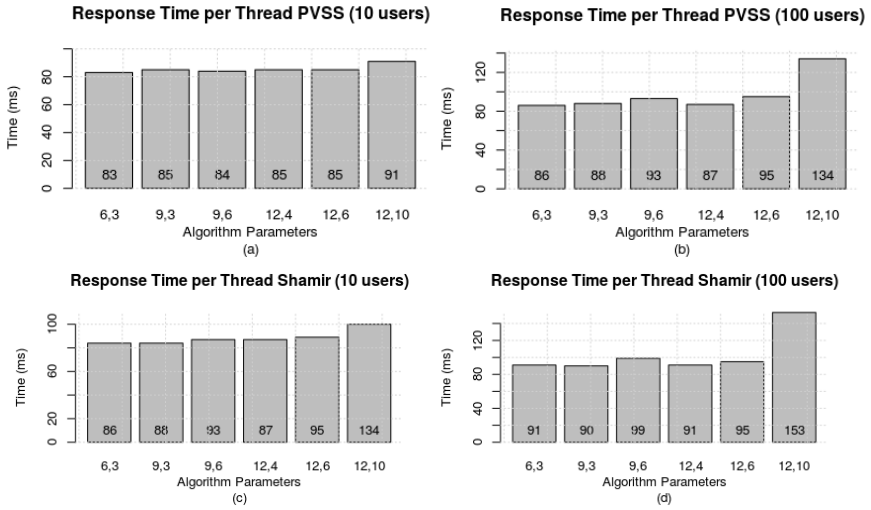


Figura 7 – Tempo de Autenticação

É possível perceber que este tempo de *querying* tem pequenas variações quando consideramos dez (10) usuários simultâneos (Figura 7.a e Figura 7.c). O mesmo não acontece nos testes de cem (100) usuários simultâneos (Figura 7.b e Figura 7.d). As leituras nas diferentes bases de dados das *clouds* acontecem de forma paralela, de modo que o tempo de *querying* será sempre determinado pela *cloud* que responde mais lentamente. No caso de muitos usuários simultâneos, o acesso às informações começa a apresentar baixo desempenho pelos problemas já comentados sobre a contenção nos canais. Apesar de não presentes nos gráficos, o tempo de execução dos protocolos de *Secret Sharing* (por operação) tem seus valores em uma razão menor que 1 *ms* nos casos com 10 usuários e aproximadamente 4,6 *ms* nos casos com 100 usuários.

5.1.3 Testes de Vazão de Autenticações

Estes testes tiveram como objetivo verificar a vazão (operações de autenticação por segundo) do protótipo desenvolvido. A Figura 8 apresenta os resultados obtidos. É possível perceber que os valores de vazão começam a diminuir significativamente a partir de aproximadamente 70 usuários concorrentes no processo de autenticação no sistema. Após a leitura dos resultados percebemos que o aumento do número de usuários simultâneos acaba impactando fortemente no desempenho do sistema.

Porém, em uma análise preliminar foi possível constatar que o problema está relacionado à implementação das estruturas de dados compartilhadas entre as *threads*. Uma implementação mais criteriosa das estruturas de dados compartilhadas e o uso de bases de dados com um maior *QoS* poderia apresentar melhora nestes valores, assim como amenizar o efeito de gargalo no caso de múltiplos usuários.

5.1.4 Testes com o uso de *cache*

Em nossos testes anteriores, sempre levamos em conta que os atributos de usuários não estavam em *cache*. Por outro lado, a Figura 9 apresenta a vazão de autenticações com o

uso de *cache,* porém com somente dados de um pequeno número de usuários. Nestes testes, foram considerados 10% dos registros de usuários presentes na *cache* para consulta. A escolha do valor de 10% de registros em *cache* se justifica diante do número potencial de usuários de uma federação de *clouds*. As autenticações que fazem uso destes 10% de registros em *cache* não executam o algoritmo de compartilhamento de segredos (camada *Secret Sharing*) e tampouco a busca das partes na federação. Com isto, foi possível perceber o acréscimo da vazão de autenticações em relação aos testes onde as informações não estavam em *cache.* Quando o esquema de *Shamir* é usado nesta camada *Secret Sharing*, este aumento é de aproximadamente 13%. Com o PVSS o acréscimo na vazão foi de aproximadamente 15%.

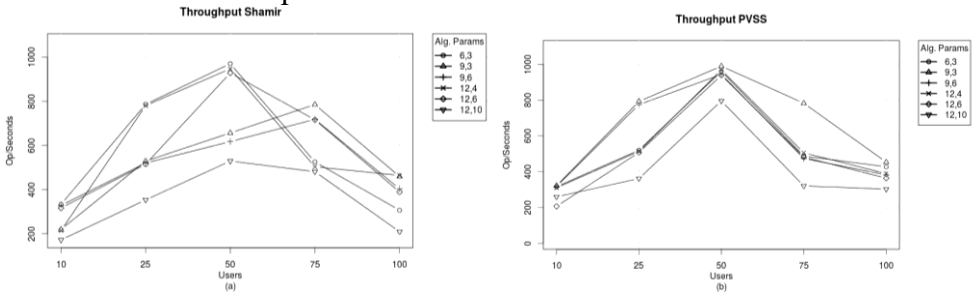


Figura 8 – Vazão de autenticações (operações por segundo)

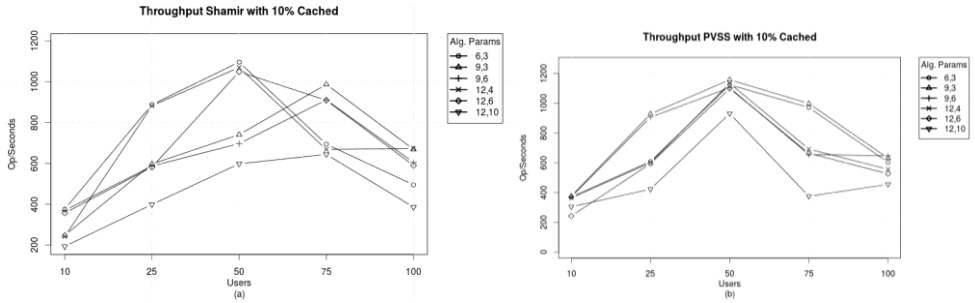


Figura 9 – Vazão com o uso de *cache* (operações por segundo)

5.1.5 Testes dos algoritmos com a presença de intrusões

Nesta seção buscamos avaliar as implementações nos casos onde intrusos podem alterar as partes do segredo comprometendo a montagem dos atributos. Para este fim, partes dos segredos foram alteradas, simulando uma intrusão e o comprometimento de partes dos segredos nas bases de dados distribuídas nas clouds da federação. Nestes testes, foi considerado o número máximo de intrusões (n - t intrusões), ou seja, todos os registros de segredos distribuídos na federação de clouds terão n - t de suas partes corrompidas.

5.1.6 Vazão dos algoritmos com intrusões

Nesta seção, são apresentados os resultados de vazão de autenticações, considerando os algoritmos de *Shamir* e *PVSS* na camada *Secret Sharing* e a presença de intrusões nas *clouds* da federação. O algoritmo PVSS, por se tratar de um esquema de compartilhamento verificável, permite que as partes de um segredo quando recebidas sejam verificadas. As verificações das partes usam as provas geradas no momento do processo de particionamento do segredo [Schoenmakers 1999].

De forma distinta ao protocolo *PVSS*, a abordagem de *Shamir* para o compartilhamento de segredo não apresenta nenhum recurso que permita a verificação da integridade das partes. Basta uma parte corrompida sendo usada para que o algoritmo retorne um valor inadequado. A alternativa para recuperar o segredo é fazer o uso de combinações de *t* partes dentre as *n* disponíveis (C_n^t), até que se consiga *t* partes corretas. Para fins de análise, buscamos então definir dois limites para estes valores: o primeiro foi chamado de *Best Case* que corresponde ao uso de *t* partes corretas na primeira iteração do algoritmo de *Shamir*, com o mesmo retornando de imediato o segredo; o segundo limite identificamos como *Worst Case*, onde somente na última interação do algoritmo ocorre a combinação com *t* partes corretas para conseguir o segredo desejado. A Figura 10 mostra resultados para estas vazões quando o algoritmo utilizado é o *PVSS*. Podemos perceber a partir destes resultados que os valores de vazão (em presença de intrusões) sofrem uma diminuição de aproximadamente 50% quando comparados com os casos sem intrusões. Os testes de vazão realizados utilizando o esquema de *Shamir* na camada *Secret Sharing* são mostrados na Figura 11. Neste caso, a vazão de autenticações apresenta valores de aproximadamente 55% do valor do mesmo teste sem intrusões em *Best Case* e valores de vazão extremamente baixos (menores que uma operação por segundo) em *Worst Case*.

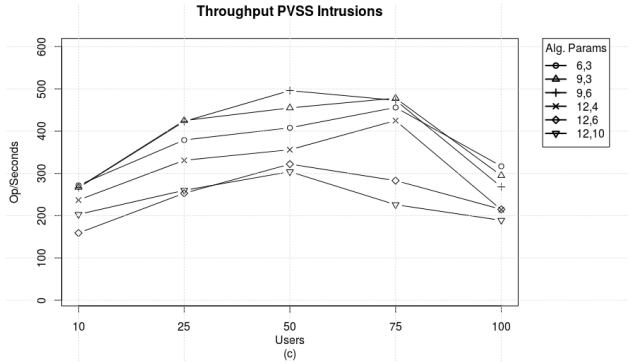


Figura 10 – Vazão com intrusões utilizando PVSS (operações por segundo)

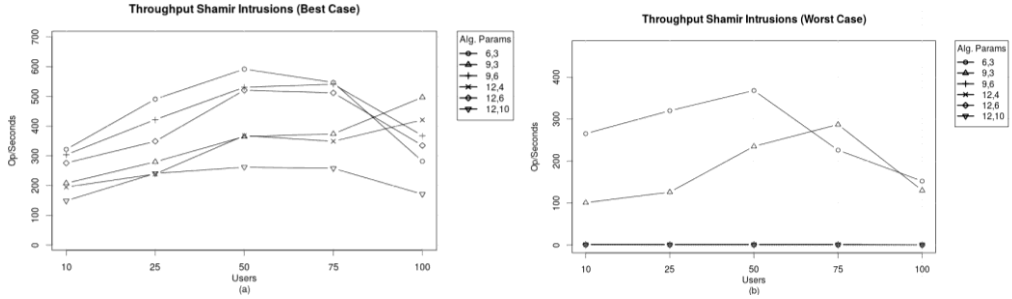


Figura 11 – Vazão com intrusões utilizando Shamir (operações por segundo)

6. Literatura Relacionada

Vários trabalhos na literatura tratam de atributos de usuários e autenticação em *clouds*. Mas, poucos destes coincidem nos seus objetivos com as preocupações que orientaram nossos trabalhos. Na sequência discutimos alguns destes trabalhos.

O projeto *SPICE* [Chow et al. 2012] descreve experiências com o objetivo de proteger a identidade de usuários que utilizam serviços em *clouds*. Os autores propõem a criação para cada usuário de um grupo de assinaturas [Boneh et al. 2004] que são

usadas para a autenticação do usuário nos diferentes provedores de *cloud*, onde cada assinatura gerada provê os atributos necessários para um determinado provedor. Outro *framework* semelhante, utilizando criptografia de grupo, é descrito em [Yan et al. 2009].

BlindIdM [Nuñez and Agudo 2014] discute os problemas de segurança relacionados ao envio de atributos de usuários para provedores de *cloud*. O objetivo principal da proposta é a proteção dos atributos de usuários, uma vez que os autores assumem que as *clouds* são ambientes com segurança limitada e que não podem controlar manipulação indiscriminada dos dados de seus usuários. Os autores então propõe a existência de um "*Blind Identity Provider*", que funciona como uma camada intermediária entre os provedores de identidades das *clouds* e os donos dos atributos.

Em Bertino [Bertino et al. 2009] a segurança dos atributos de usuários é garantida através do uso de duas técnicas: a primeira, onde o usuário e o consumidor da identidade (*cloud provider*) entram em acordo de forma a definir *metadados* para atributos de usuários. Estes *metadados* acordados entre as partes definem como as informações devem ser enviadas pelo usuário e como as mesmas devem ser interpretadas pelo provedor de *cloud*. Estes *metadados* são protegidos por assinaturas.

Estes trabalhos citados acima têm como objetivo facilitar aos usuários na manipulação de seus atributos e restringir a revelação dos mesmos nos acessos a provedores de *cloud*. A nossa abordagem, embora proteja os atributos de usuários, possui um foco diferente. No nosso caso, as *clouds* estão associadas em federação para fornecer serviços e recursos aos seus usuários. A autenticação do usuário, na nossa proposta, é feita no seu *IdP* de registro (na sua *cloud*) e é aceita pelas demais *clouds* da federação. O *BlindIdM* e a proposta apresentada em Bertino, também como a nossa abordagem, tentam evitar a disponibilidade de atributos do usuário nas *clouds*. Mas a ação dos mecanismos dos trabalhos citados está limitada às interações dos usuários no processo de autenticação nas *clouds*. Nós, em nossa abordagem, estamos propondo o armazenamento seguro destas informações em *clouds* da federação. E a autenticação em nosso sistema envolve provedores de identidades que tratam somente da autenticação de usuários em *clouds* desta federação.

Em [Angin et al. 2010], os autores apresentam uma proposta onde os atributos de usuário sempre estão de posse do usuário, nunca armazenados em um provedor de identidades. Isto garante melhorias no controle de que dados serão liberados para as *clouds* pois o usuário é quem define quais atributos liberar. Além disso, para manter atributos de forma segura, a abordagem utiliza assinatura do tipo *Zero-knowledge proof*, de forma que o usuário seja a própria entidade a certificar os atributos enviados**.** Uma abordagem semelhante foi apresentada [Böger et al. 2011] onde os atributos de usuário são mantidos em cartão (*smartcards*). A autenticação é feita via certificados e os atributos de usuário são enviados do *smartcard* para o provedor de serviço, via o *IdP* do sistema. São políticas *user-centric* que definem quais atributos liberar.

A literatura apresenta inúmeras experiências [Bessani et al. 2011; Sujana et al. 2013] na construção de memória segura em *clouds* ou em federação de *clouds*. Estes trabalhos fazem uso de esquemas de compartilhamento de segredo e de algoritmos de dispersão de informação para a memorização segura em *clouds*. Nestas abordagens, a confidencialidade das informações é conseguida com a encriptação das mesmas, usando uma chave simétrica gerada aleatoriamente. Em seguida, o arquivo cifrado é subdividi-

do a uma transformação *IDA*. Por sua vez, a chave criptográfica usada na proteção da informação passa por um processo de compartilhamento de segredo, gerando também partes relacionadas com a chave. Esta distribuição deve evitar que tanto as partes do arquivo como as partes da chave venham a ser enviadas aos mesmos provedores de modo a comprometer os seus segredos. A nossa abordagem é mais econômica no uso destas técnicas. Como os atributos e credenciais de usuários correspondem a registros de poucos bytes, na nossa abordagem podemos aplicar diretamente o compartilhamento de segredo sobre estas informações, evitando com isto o uso de técnicas de correção de erro e de criptografia para a confidencialidade destas informações.

7. Conclusão

Este artigo apresenta uma abordagem que faz uso de provedores de *cloud* para armazenar as informações de usuário (credenciais e atributos). Neste sentido, descrevemos um modelo de autenticação para uso em federações de provedores de *cloud*. Uma arquitetura foi desenvolvida para implementar um protótipo de nossas proposições que definem o modelo. Como ambientes de *clouds* apresentam segurança limitada, aplicamos técnicas de compartilhamento de segredo nas informações de usuário, permitindo a distribuição de partes destes segredos nas várias *clouds*. Testes com ou sem intrusões e usando diferentes esquemas de compartilhamento de segredo foram realizados para mostrar a eficiência de nossos mecanismos.

O modelo apresentado provê certa flexibilidade do acesso às informações permitindo que usuários possam fazer os seus *logins* a partir de um *IdP* de uma federação de *clouds* e ter acessos a recursos em diferentes partes desta federação. A abordagem de gerenciamento de identidades segue o modelo centralizado devido ao uso de protocolos do *OpenID*. Mas pela flexibilidade da disponibilidade das informações nas *clouds*, temos as mesmas características do gerenciamento de identidades federadas, porém sem a necessidade de relações de confiança entre *IdPs*.

Vale a pena ressaltar que, ainda que o processo de criação de registros apresente um desempenho menor do que os tempos de autenticação (até 10 vezes maiores), por serem menos frequentes, estas criações ou alterações de contas não chegam a influenciar no desempenho do sistema de autenticação.

O trabalho apresentado neste texto é o complemento de outra experiência descrita em [Barreto et al. 2013] onde os provedores de identidades são desenvolvidos com o uso extensivo da tecnologia de virtualização para isolar as informações de usuários. Intrusos não têm acesso a *cache* com registros de usuários que é mantida em compartimento isolado e distante da rede. O modelo de autenticação apresentado não se restringe a ambientes de federação de *clouds*. Qualquer *IdP* pode ser concebido usando a esta e fazer uso de alguns provedores de *cloud* que não formem associações para o compartilhamento de recursos como as citadas federações.

8. Agradecimentos

Agradecemos a CAPES pelo apoio financeiro com bolsa de doutorado e mestrado e bolsa CNPQ Processo [233648/2014-3].

Referências

Angin, P., Bhargava, B., Ranchal, R., et al. (oct 2010). An Entity-Centric Approach for Privacy and Identity Management in Cloud Computing. *2010 29th IEEE Symposium on Reliable Distributed Systems*, p. 177–183.

Barreto, L., Siqueira, F., Fraga, J. and Feitosa, E. (2013). An Intrusion Tolerant Identity Management Infrastructure for Cloud Computing Services. In *2013 IEEE International Conference On Web Services*.

Bertino, E., Lafayette, W., Paci, F. and Ferrini, R. (2009). Privacy-preserving Digital Identity Management for Cloud Computing. *Identity*, v. 32, p. 1–7.

Bessani, A., Correia, M., Quaresma, B. and Sousa, P. (2011). DEPSKY: Dependable and Secure Storage in a Cloud-of-Clouds. *European Systems Conference.*

Böger, D., Barreto, L., Fraga, J., et al. (2011). User-Centric Identity Management Based on Secure Elements. n. 590047.

Boneh, D., Boyen, X. and Shacham, H. (2004). Short Group Signatures. *Advances in Cryptology - CRYPTO 2004*, v. 3152, p. 227–242.

Burr, W. E., Dodson, D. F. and Polk, W. T. (2006). Electronic authentication guideline. *NIST Special Publication*, v. 800:63.

Celesti, A., Tusa, F., Villari, M. and Puliafito, A. (2010). How to Enhance Cloud Architectures to Enable Cross-Federation. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*.

Chow, S., He, Y., Hui, L. and Yiu, S. (2012). Spice–simple privacy-preserving identity-management for cloud environment. *Applied Cryptography and Network, p.*526–543.

Grozev, N. and Buyya, R. (2012). Inter-Cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, p. 1–22.

Jøsang, A., Fabre, J., Hay, B., Dalziel, J. and Pope, S. (2005). Trust requirements in identity management. *CRPIT '44: Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, p. 99–108.

Nuñez, D. and Agudo, I. (2014). BlindIdM: A privacy-preserving approach for identity management as a service. *International Journal of Information Security*, p. 199–215.

Rabin, M. O. (1989). Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, v. 36, n. 2, p. 335–348.

Schoenmakers, B. (1999). A Simple Publicly Verifiable Secret Sharing Scheme and its Application to Electronic Voting. *Advances in Cryptology (CRYPTO99)*, p. 148–164.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, p. 612–613.

Sujana, B., Tejaswini, P., Srinivasulu, G. and Karimulla, S. (2013). Secure Framework for Data Storage from Single to Multi clouds in Cloud Networking. v. 2, n. 2.

Yan, L., Rong, C. and Zhao, G. (2009). Strengthen cloud computing security with federal identity management using hierarchical identity-based cryptography. p. 167–177.

Determinando o Risco de Fingerprinting em Páginas Web

Adriana R. Saraiva[1], **Adria M. de Oliveira** [1], **Eduardo L. Feitosa**[1]

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
CEP 69.077-000 – Manaus – AM – Brasil

{adriars,adriah.menezes}@gmail.com,efeitosa@icomp.ufam.ed.br

Abstract. *The fingerprinting techniques applied to Web pages to identify and track users on the Internet has become common. Although there are solutions and countermeasures against such techniques, they often come up against the need to compare codes or running in the background. In this context, this paper proposes a methodology to detect artifacts (scripts) fingerprinting in Web pages and inform the users how dangerous the page is for your privacy. The results show that although simple, the method is efficient to find fingerprinting codes and categorizes them by severity levels of user privacy.*

Resumo. *As técnicas de fingerprinting aplicadas em páginas Web para identificar e rastrear usuários na Internet têm se tornado cada vez mais comum. Embora existam soluções e contramedidas contra tais técnicas, elas muitas vezes esbarram na necessidade de comparação de códigos ou execução em segundo plano. Neste contexto, este artigo propõe uma metodologia para detectar artefatos (scripts) de fingerprinting em páginas Web e informar aos usuários o quão perigosa a página é para sua privacidade. Os resultados mostram que embora simples, a metodologia é eficaz ao encontrar códigos fingerprinting nos websites e categorizá-los em níveis de severidade quanto aos riscos à privacidade dos usuários.*

1. Introdução e Motivação

Com a ineficácia frente as plataformas móveis e a crescente preocupação com privacidade, os cookies tornaram-se obsoletos e passaram a ser limitados na Internet. Entretanto, a indústria de anúncios e publicidade online, a fim de manter a continuidade de seus negócios, investiu em novas formas de rastreio e identificação por meio do próprio conjunto de hardware/software do usuário. Esse conceito, conhecido como *Website Fingerprinting*, *Browser Fingerprinting* ou *Device Fingerprinting*, parte do pressuposto que cada usuário opera o seu próprio hardware. Assim, a identificação de um dispositivo é o mesmo que identificar a pessoa por trás dele. Segundo Nikiforakis [Nikiforakis et al. 2014], *Website Fingerprinting* faz uso de um conjunto de atributos do sistema extraídos do dispositivo do usuário, que gera uma combinação de valores únicos capazes de identificar um usuário/dispositivo.

O fato relevante e motivador para esta pesquisa é que as técnicas de *Website Fingerprinting* estão se tornando mais comuns e que os usuários sabem pouco ou quase nada sobre o assunto. Mesmo quando estão cientes de que estão sendo monitorados (como uma medida de proteção contra a fraude, por exemplo), os usuários simplesmente confiam que as informações coletadas não serão utilizadas para outros fins. Indiferente a quão fácil é

obter informações sobre o navegador e o dispositivo de um usuário, a fim de gerar uma identificação única, o fato é que existem diferentes tipos de artefatos para realizar um *fingerprinting* na Web, onde se emprega uma variedade de tecnologias.

Neste cenário, uma questão pouco explorada é o quão perigoso é um artefato desse. Assim, o objetivo deste artigo é mensurar a severidade de um artefato *fingerprinting* presente em uma página Web. Como forma de alcançar esse objetivo foram analisadas as principais formas e tecnologias para realizar um *Website Fingerprinting*. A principal contribuição desse artigo é a definição de uma simples metodologia para estimar a severidade de artefatos (scripts) de *fingerprinting* na Web, em relação a privacidade dos usuários.

O restante do artigo é organizado como segue: na Seção 2 é apresentada uma breve descrição sobre *Website Fingerprinting*, com seu funcionamento e tecnologias atualmente utilizadas. Na Seção 3 são descritas as estratégias de identificação de usuários com o uso de *fingerprinting* e também de detecção de *fingerprinting*. Nesta Seção, os trabalhos são discutidos de forma a evidenciar as escolhas/técnicas empregadas. Na Seção 4 é apresentada a metodologia proposta para estimar a severidade de artefatos (scripts) de *fingerprinting* na Web, com os detalhes de implementação. Na Seção 5 são apresentados os experimentos e os resultados obtidos com a classificação dos websites quanto ao nível de severidade da *fingerprinting*, com uma breve discussão. Por fim, a Seção 6 expõe as observações finais sobre o trabalho.

2. *Website Fingerprinting*

De acordo com a RFC 6973 [Cooper et al. 2013], o termo *fingerprint* é definido formalmente como "um conjunto de elementos de informação que caracteriza um dispositivo ou uma instância de uma aplicação" e *fingerprinting* como "o processo pelo qual um observador ou atacante identifica, de maneira única e com alta probabilidade, um dispositivo ou uma instância de um aplicativo com base em um conjunto de múltiplas informações".

Este trabalho partilha da visão que *fingerprinting* é parte de um conjunto amplo de tecnologias e técnicas usadas para identificar (ou reidentificar) um usuário ou um dispositivo, através de um conjunto de configurações, atributos (tamanho da tela do dispositivo, versões de software instalado, entre muitos outros) e outras características observáveis durante a comunicação.

De acordo com o W3C (*World Wide Web Consortium*)[1] [W3C 2014], as técnicas de *website fingerprinting* podem ser classificadas em: (1) **Passiva**, baseada nas características observáveis na comunicação HTTP, tais como: cabeçalhos, endereço IP e outras informações do nível de rede; (2) **Ativa**, que executam códigos (JavaScript mais comumente) no lado do cliente para observar as características como dados do hardware e do navegador; e (3) **Cookie-like**, onde um código ou aplicativo é instalado no lado do cliente.

Embora o uso de *website fingerprinting* esteja relacionado à identificação de usuários como medida de segurança e mecanismo anti-fraude, os impactos na privacidade dos usuários causados por essa identificação são ameaças reais de segurança. Três problemas na privacidade são apontados em [W3C 2014]: (1) **identificação do usuário**, visto que um *fingerprinting* tem o potencial de obter dados do usuário sem autorização

[1] http://www.w3c.org

prévia que o deixa exposto e vulnerável; (2) **correlacionar as atividades da navegação**, visto que os usuários podem se surpreender ao perceber que terceiros podem correlacionar suas várias visitas ao mesmo ou diferentes sites com a finalidade de elaborar um perfil; (3) **inferências sobre o usuário**, uma vez que as informações coletadas podem revelar dados sobre os quais se pode tirar conclusões sobre o usuário. O W3C [W3C 2014] afirma que tais inferências permitem oferecer e mostrar ao usuário produtos em determinada faixa de preço, o que pode ser uma forma discriminatória de publicidade.

2.1. Tecnologias

Embora as tecnologias Web visem dinamizar o uso dos conteúdos em páginas na Internet, esta seção descreve, resumidamente, como elas podem ser usadas para *website fingerprinting*. Maiores detalhes e exemplos podem ser encontrados em [Saraiva et al. 2014]

HTML5 Canvas

Canvas é um novo elemento da HTML5 que, via JavaScript, fornece acesso a um conjunto completo de funções de desenho, para que gráficos sejam gerados dinamicamente no lado do cliente. No que diz respeito a *website fingerprinting*, o uso de Canvas já foi demonstrado nos trabalhos de Mowery e Shacham [Mowery and Shacham 2012] e Kirk [Kirk 2014]. Embora pareça inofensivo, o elemento Canvas possui dois (2) métodos - $toDataURL()$ e $getImageData()$ - capazes de extrair informações da tela do navegador (dados pessoais, por exemplo).

JavaScript

JavaScript é uma linguagem de programação interpretada, implementada como parte dos navegadores para que scripts pudessem ser executados do lado do cliente e interagissem com o usuário sem a necessidade deste script passar pelo servidor [Crockford 2008]. Em linhas gerais, JavaScript é usada como mecanismo de execução de um *fingerprinting* e não como a real causa do rastreio ou acesso a informações. Contudo, os trabalhos de Mowery et al. [Mowery et al. 2011] e Mulazzani et al. [Mulazzani et al. 2013] demonstraram como utilizar JavaScript para obter informações sobre o navegador e os usuários.

Adobe Flash

Adobe Flash é utilizado para entrega de um rico conteúdo na Web, no intuito de atrair e envolver os usuários. Através dele é possível exibir animações e interfaces de aplicativos, que são implantadas imediatamente em todos os navegadores e plataformas.

WebGL

WebGL (*Web Graphics Library*) é uma API multiplataforma, que traz a linguagem OpenGL ES 2.0 para a Web de forma a suportar desenhos 3D dentro do HTML [Group 2014]. Em outras palavras, oferece suporte para renderização de gráficos 2D e 3D. Por ser suportada por todos os navegadores, WebGL pode ser empregada para obter identificadores do usuário, como demonstrado no site Browserleaks [BrowserLeaks.com 2015]. Além disso, os estudos de Forshaw [Forshaw 2011] descobriram e provaram a existência de uma série de problemas de segurança com a especificação e implementação do WebGL, que permitiam desde identificar o navegador até realizar ataques de negação de serviço.

Biblioteca Modernizr

Modernizr é uma biblioteca capaz de detectar automaticamente a disponibilidade de tecnologias presentes no navegador do usuário. Ela permite, por exemplo, detectar o suporte a geolocalização, ao elemento $canvas$, aos plugins WebGL e flash, bem como o armazenamento local e armazenamento de sessão. Unger et. al [Unger et al. 2013] afirmam que o a biblioteca modernizr é uma poderosa tecnologia para obtenção de recursos disponíveis no navegador.

CSS

CSS (*Cascading Style Sheets*) é uma linguagem de estilo utilizada para descrever a aparência e a formatação de um documento em uma linguagem de marcação [Bos et al. 2011]. Foi projetada para permitir a separação do conteúdo do documento de apresentação de documentos, através de propriedades de estilo que incluem elementos de layout, cores, fontes, entre outros. No que tange *fingerprinting*, CSS tem potencial para obter vários tipos de informação diferente dos tradicionais dados de fontes.

Silverlight

Silverlight é um framework criado pela Microsoft para o desenvolvimento e execução de aplicações ricas para a Internet, com recursos e propostas similares ao Adobe Flash. Em termos técnicos, o Microsoft Silverlight é uma aplicação *cross-browser*, *cross-platform* do framework .Net. Embora o Silverlight possua várias características importantes e atraentes para os usuários, este plugin também está vulnerável as técnicas de *fingerprinting*. Assim como o Adobe Flash, Silverlight é capaz de obter informações do sistema, tais como: versão do sistema operacional, número de processadores, fuso horário, fontes instaladas, sistema, região, idioma do sistema operacional, entre outros.

3. Trabalhos Relacionados

Embora não exista uma classificação formal, a literatura sobre *website fingerprinting* pode ser dividida em: (i) identificação do usuário e/ou dispositivo e (ii) detecção de *website fingerprinting*.

Na primeira área, o trabalho de Peter Eckersley [Eckersley 2010] é considerado seminal por investigar o grau em que os navegadores modernos estão sujeitos a *website fingerprinting*. Para tanto, desenvolveu um site de teste **Panopticlick**[2] com um algoritmo capaz avaliar características do navegador e gerar um identificador único. Dos 470.161 acessos voluntários, 83,6% tiveram identificações únicas e apenas 5,3% de navegadores/usuários mantiveram o anonimato.

Boda et. al [Boda et al. 2012] seguiram a mesma ideia e desenvolveram o site Portal PET Internacional[3]. Como resultado, os autores perceberam que a correlação de listas de fontes do navegador e *UserAgents*[4] proporcionam uma base sólida para a identificação única dos sistemas operacionais Windows e Mac. Olejnik et. al. [Olejnik et al. 2013] comprovou, com cerca de 368.284 usuários, que é possível detectar, via CSS, os históricos de navegação dos usuários como ferramenta *fingerprinting*. Os resultados mostram que

[2]https://panopticlick.eff.org

[3]http://pet-portal.eu/fingerprint/

[4]De acordo com a RFC 2616 [Fielding et al. 1999], *user-agent* é um campo enviado quando o navegador faz a solicitação de uma página. Basicamente, é uma sequência de caracteres composta por um conjunto de elementos, chamados *tokens*, listados em ordem de importância de acordo com o padrão.

para a maioria dos usuários (69%), o histórico de navegação é único e que com apenas quatro (4) sites é possível identificar unicamente 97% dos usuários.

No âmbito de detecção de *website fingerprinting*, Acar et al. [Acar et al. 2013] desenvolveram um framework, denominado FPDetective, para encontrar códigos de *fingerprinting*. Para validar o trabalho, os autores empregaram 1 milhão de sites do alexa.com[5] e descobriram 404 sites que utilizavam JavaScript para *fingerprinting*. Em relação ao Flash foram encontrados 145 sites. O baixo valor nos resultados se deve ao fato dos autores usarem apenas 27 scripts conhecidos como base de comparação.

Khademi [Khademi 2014] apresentou uma solução, denominada FPGuard, para detectar e proteger os usuários contra o *fingerprinting*. Dada uma página Web em execução no navegador do usuário, o FPGuard monitora e registra sua atividade a partir de seu carregamento. Em seguida, extrai métricas relativas a cada método *fingerprinting* e constrói uma assinatura para a página Web. A avaliação foi realizada usando o top 10K do alexa.com, onde o FPGuard conseguiu identificar 9264 sites como *website fingerprinting*.

3.1. Discussão

A Tabela 1 apresenta, de forma resumida, uma comparação entre os trabalhos tanto de identificação quanto de mitigação de *website fingerprinting*. Vale ressaltar que todos valores indicados na tabela foram retirados dos trabalhos originais.

Tabela 1. Comparação das soluções de identificação e mitigação de *Website fingerprinting*

Artigo	Classificação	Tecnologias Empregadas	Propriedades do Navegador	Protótipo	Fonte de Dados
Panopticlick [Eckersley 2010]	Identificação	JavaScript e Flash	Nav, Scr, TZ, SF, HTTP	Sim	*
FPDetective [Acar et al. 2013]	Identificação	JavaScript, Flash, HTML5 e CSS3	Nav, Scr, Win, SF	Sim	alexa.com
Cross-Browser Fingerprinting [Boda et al. 2012]	Identificação	JavaScript	Nav, Scr, TZ, SF, HTTP	Sim	Panopticlick
Web Browsing History [Olejnik et al. 2013]	Identificação	CSS	Hy	Não	Não
Privaricator [Nikiforakis et al. 2015]	Híbrido	JavaScript e HTML5 Canvas	Nav e Scr	Não	alexa.com, bluecava, coinbase, petportal e fingerprintjs
FPGuard [Khademi 2014]	Híbrido	JavaScript, Flash e HTML5 Canvas	Nav, Scr, SF, TZ, HTTP	Não	alexa.com
Determinando o Risco de Fingerprinting em Páginas Web	Alerta e Classificação de Risco	JavaScript	Win, Doc, Nav, Scr, Sil, Mod, WGL, Can	Não	Diversas

Propriedades do Navegador: Nav-objeto Navigator; Scr-objeto Screen; Win-objeto Window; TZ-timezone;
Doc-objeto Document;
SF-sistema de fontes; HTTP-cabeçalho HTTP; HY-histórico;
Sil-Microsoft Silverligth; Mod-Biblioteca Modernizr; WGL-WebGL; Can-Canvas
*Não aplicável ou não informado

Independente do tipo de trabalho (classificação), os trabalhos diferem em termos do uso da tecnologia. Por exemplo, percebe-se o maciço uso de JavaScript como mecanismo de geração do *fingerprinting* enquanto o uso de Flash é bem pequeno. Em contrapartida, as soluções que empregam JavaScript e Flash acabam tendo uma maior inserção

[5]http://alexa.com

no navegador, conseguindo obter uma grande quantidade de informações, como visto no critério propriedade do navegador.

Outra observação interessante reside no fato de que existem poucas soluções acadêmicas focadas em combater *Website fingerprinting*. Com a exceção dos plugins e as duas soluções de detecção ([Nikiforakis et al. 2015] e [Khademi 2014]), todas as demais soluções visam apenas provar a existência do fato. Em outras palavras, é possível imaginar situações mais robustas e que não tenham como foco a proteção individual dos usuários. Tal fato, reforça a motivação para este trabalho. Por fim, também percebe-se na Tabela 1 que o foco da maioria dos trabalhos existentes na literatura é investigar artefatos JavaScript.

Este trabalho difere dos demais no âmbito da investigação de uma quantidade maior de termos (propriedades do Navegador e aplicações) presentes nos artefatos *fingerprinting*. Tais artefatos são classificados pelo nível de risco, o que possibilita alertar e mostrar ao usuário o perigo ao qual ele está exposto. Além disso, embora o foco da metodologia proposta aqui não ter uma implementação formal (um protótipo), ela pode ser agregada à uma aplicação, um plugin ou um módulo de um Proxy Web. Com isso, é possível verificar e classificar *fingerprinting* em páginas Web em tempo real.

4. Metodologia Proposta

Esta seção apresenta a visão geral da metodologia proposta para estimar a severidade de artefatos (scripts) de *fingerprinting* na Web a partir do conjunto de propriedades e métodos dos objetos HTML DOM e seu nível de risco à privacidade dos usuários.

4.1. Visão Geral

A Figura 1 apresenta a metodologia proposta, executada em quatro etapas: (1) Coleta de Páginas (sementes) para avaliação; (2) Extração de Scripts (códigos em JavaScript) de uma determinada página e geração de um único arquivo de script; (3) Extração de Termos (objetos, propriedades e métodos) relacionados aos artefatos comumente empregados em *fingerprinting*; e (4) Classificação da Severidade.

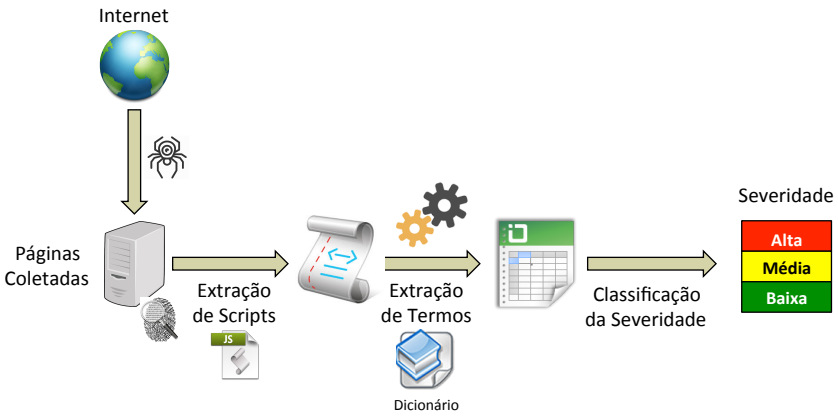


Figura 1. Visão geral da metodologia proposta

A intenção é manter o usuário informado da existência de artefatos *fingerprinting* na página acessada e qual seu nível de severidade. Este alerta permitirá ao usuário ficar

ciente da possibilidade de invasão da sua privacidade, como a captura de dados e sua posterior utilização em atividades maliciosas.

As etapas de **Coleta das Páginas** e **Extração de Script** são simples e correspondem a coleta das páginas Web e a extração dos scripts (JavaScripts) contidos nelas. Para executar essas funções, foi implementado um *crawler*, em linguagem PHP (versão 5.5.15), onde para cada site visitado, são coletados todos os scripts em JavaScript e armazenados em um único arquivo por site.

A etapa de **Extração de Termos** objetiva identificar os termos (objetos, propriedades e métodos) presentes no arquivo de script gerado na etapa anterior. Para esta etapa foi desenvolvido um script de extração na linguagem Python (versão 3.4) que realiza a leitura do arquivo único e extrai as propriedades e métodos de acordo com um dicionário previamente estabelecido. O dicionário de termos foi elaborado com base na literatura acadêmica de implementação de *fingerprinting* quanto contramedidas. Em outras palavras, trabalhos, scripts e exemplos de códigos foram avaliados e os objetos, propriedades e métodos utilizados para realizar *fingerprinting* foram selecionados. No total, o dicionário possui 58 termos.

Em relação ao processo propriamente dito, a extração dos termos é feita através do uso de expressões regulares (REGEX), um excelente recurso para casamento de padrões. O uso de REGEX permite que o script de extração seja capaz de buscar objetos, propriedades, atributos e métodos, possibilitando a descoberta de termos em até três níveis. A Tabela 3 apresenta o uso de expressões regulares na extração de termos.

Tabela 2. Expressões Regulares para Extração de Termos

Expressão Regular	Níveis
$("N2", r'([nw] * n.[nw]*)')$	Essa expressão regular busca por objetos que utilizam em sua sintaxe de chamada até dois níveis de acesso a propriedade. Exemplo: $navigator.userAgent$
$("N3", r'([nw] * n.[nw] * n.[nw]*)')$	Essa expressão regular busca por objetos que utilizam em sua sintaxe de chamada até três níveis de acesso a propriedade. Exemplo: $window.navigator.plugin$
$("N4", r'([nw] * n.[nw] * n.[nw] * n.[nw]*)')$	Essa expressão regular busca por objetos que utilizam em sua sintaxe de chamada até quatro níveis de acesso a propriedade. Exemplo: $window.navigator.plugin.name$

A quarta e última etapa é a **Classificação da Severidade**, cuja finalidade é definir o nível de risco (severidade) a qual o usuário está exposto ao acessar uma página Web. Essa etapa é baseada na metodologia para estimação da severidade de risco da OWASP[6]. Contudo, a severidade neste trabalho é calculada de forma mais simples, sem a necessidade de estimar o impacto de um evento e a probabilidade de uma vulnerabilidade ser explorada, como é necessário na metodologia da OWASP.

A ideia para classificar um artefato *fingerprinting* foi estimar o grau de envolvimento de elementos (objetos, propriedades, atributos e métodos) com ataques e intrusões na Web. Para isso, foi realizada uma detalhada pesquisa na literatura acadêmica e na área de segurança da informação (site, fóruns de discussão, blogs de desenvolvedores, comunidades hackers, entre outros), onde cada objeto, propriedade, atributo e método listados no dicionário de termos foi co-relacionado à ataques, vulnerabilidades e intrusões. Ou seja, como cada um desses elementos pode ser e é usado para explorar vulnerabilidades,

[6]Open Web Application Security Project (https://www.owasp.org) é uma entidade sem fins lucrativos e de reconhecimento internacional que contribui para a melhoria da segurança de softwares aplicativos.

violar privacidade, obter dados e realizar ataques.

Desta forma, inspirando-se na metodologia da OWASP, foram definidas três classes de severidade: Baixa, Média e Alta. Na classificação **Baixa** encontram-se os elementos capazes de fornecer informações usadas apenas para adaptações de conteúdo (tamanho da tela e a versão do navegador, por exemplo). Os objetos DOM HTML $Window$, $Navigator$ e $Screen$, bem como quase todas as suas propriedades, atributos e métodos se enquadram nessa classificação. Vale ressaltar que, até o momento, nenhum trabalho acadêmico provou ou interligou qualquer um desses elementos com ataques ou invasões de privacidade. Por isso, são considerados de baixo impacto quanto ameaça à privacidade. Contudo, no momento em que são combinados com outros elementos, podem compor uma poderosa arma de *fingerprinting* capaz de identificar unicamente o usuário e/ou dispositivo, conforme já demonstrado nos trabalhos de Eckersley [Eckersley 2010], Acar [Acar et al. 2013], entre outros.

Na classificação **Média** encontram-se os artefatos que listam os plugins instalados, os *mime-types* suportados e que verificam a presença dos plugins $SilverLight$ e $WebGL$. Khademi [Khademi 2014], Eckersley [Eckersley 2010] e Nikiforakis et al. [Nikiforakis et al. 2015] descreveram esses objetos como propensos a um médio grau de ameaça. Além disso, Unger et al. [Unger et al. 2013] afirmam que a biblioteca $Modernizr$ é uma poderosa tecnologia para obtenção de recursos disponíveis no navegador e por isso ela também é inserida nessa classificação. Em linhas gerais, a verificação dos plugins e mimeTypes existentes no navegador pode fornecer informações relevantes a um atacante, como, por exemplo, dados dos plugins bancários ou alguma informação de um software específico. Khademi[Khademi 2014], Eckersley [Eckersley 2010], [Nikiforakis et al. 2015] descreveram esses objetos como propensos a um alto e/ou médio grau de ameaça.

A classificação **Alta** é composta de quatro elementos: $canvas.toDataUrl()$, $canvas.getImageData()$, $window.history()$ e $MediaDevices.getUserMedia()$. O método $canvas.getImageData()$ retorna um objeto *ImageData* que copia os dados de pixel para o retângulo especificado em área Canvas. Na verdade, é preciso esclarecer que um objeto *ImageData* não é uma figura. Ele é parte da área Canvas e manipula a informação de cada pixel dentro daquele retângulo. O método $canvas.toDataURL()$ permite obter o conteúdo da tela do navegador. O dado retornado é uma $string$ que representa uma URL codificada que contém os dados gráficos.

O objeto $history$ contém a lista das URLs visitadas pelo usuário dentro de uma janela do navegador. É parte do objeto $window$, possui a propriedade *length*, que retorna o número de URLs no histórico, e três (3) métodos: (1) *back()*, que carrega a URL anterior no histórico; (2) *forward()*, que carrega a próxima URL no histórico; e (3) *go()*, que carrega uma URL específica do histórico. É importante notar que embora não seja um padrão definido, todos os navegadores suportam este objeto. Olejnik et al. [Olejnik et al. 2013] mostraram que o histórico no navegador pode também ser usado como meio para realizar um *fingerprinting* sem a necessidade de ajuda do lado do cliente. Para tanto, eles analisaram uma base de dados gerada quando uma falha do CSS em relação ao histórico ainda estava presente em navegadores.

Por fim, o método $MediaDevices.getUserMedia()$ solicita ao usuário permissão

para usar um dispositivo de vídeo e/ou uma entrada de áudio, como uma câmera, compartilhar a tela e/ou um microfone. Tian et al. [Tian et al. 2014] apresentam múltiplos ataques usando $MediaDevices.getUserMedia()$ que podem comprometer a confidencialidade e integridade dos usuários. Eles mostram como um atacante pode roubar dados de um usuário da tela para modificar suas contas de acesso a site populares.

Tabela 3. Níveis de Severidade

Níveis	Objetos
1	window.innerHeight; window.outerWidth; window.outerHeight; window.devicePixelRatio; document.domain; navigator.userAgent; navigator.appCodeName; navigator.appName; navigator.appVersion; navigator.appMinorVersion; navigator.buildID; navigator.browserLanguage; navigator.cpuClass; navigator.doNotTrack; navigator.language; navigator.onLine; navigator.oscpu; navigator.platform; navigator.userLanguage; navigator.product; navigator.productSub; navigator.securityPolicy; navigator.systemLanguage; navigator.vendor; navigator.vendorSub; navigator.geolocation; navigator.savePreferences; screen.availHeight; screen.availWidth; screen.availLeft; screen.availTop; screen.height; screen.width; screen.colorDepth; screen.pixelDepth; screen.bufferDepth; screen.deviceXDPI; screen.deviceYDPI; screen.logicalXDPI; screen.logicalYDPI; screen.systemXDPI; screen.systemYDPI; screen.updateInterval
2	document.referrer; document.cookie; Modernizr; Silverlight; WebGL; navigator.cookieEnabled; navigator.javaEnabled; mimeTypes; plugins; window.localStorage
3	navigator.getUserMedia; canvas.toDataURL; canvas.getImageData; window.history

5. Experimentos e Resultados

Esta Seção descreve os passos executados para classificação dos websites quanto ao nível de severidade da *fingerprinting* em baixa, média ou alta. A primeira seção relata o protocolo experimental necessário (ambiente, base de dados, entre outros), enquanto a segunda descreve o processo de aplicação da metodologia proposta.

5.1. Ambiente

Os experimentos realizados foram executados em duas máquinas. A primeira é uma estação de trabalho Intel Corei7, 3,4 Ghz, com 8 GB de memória RAM, 500 GB de disco, com sistema operacional Linux, distribuição Ubuntu 14.04. A segunda um notebook Corei5, com 8GB de memória RAM, 1Tera de HD com sistema operacional Windows 8.

5.2. Avaliações

Antes de iniciar a explicação sobre as avaliações, é necessário especificar as bases de dados (páginas Web com *fingerprinting*) utilizadas neste trabalho. Foram utilizadas três (3) base de dados.

A primeira, denominada **base de controle**, é formada por 250 Websites classificados como *fingerprinting* nos artigos de Nickforakis [Nikiforakis et al. 2015], Khademi [Khademi 2014] e Acar et al. [Acar et al. 2013]. Esta base foi empregada na análise e ajuste de uso dos termos que compõem o dicionário. A segunda base, denominada de **base canvas**, é formada por 2.127 sites listados no trabalho de Englehardt et al. [Englehardt and Narayanan 2016] que utilizam API Canvas para realizar *fingerprinting*.

A terceira e última base, denominada **base DMOZ**, é composta por 1.585 sites extraídos aleatoriamente do diretório DMOZ[7], uma base que correspondem a sites considerados benignos. Esta base foi utilizada para verificação da quantidade de termos em páginas não ligadas à *fingerprinting*.

[7]http://www.dmoz.org/

5.2.1. Base de Controle

A Figura 2 ilustra o resultado da metodologia proposta na base de controle. Em destaque, na cor vermelha, estão apresentados os sites com classificação **Alta**, perfazendo um total de 87%. Os sites $coinbase.com$ e $tumblr.com$ possuem destaque nessa base por possuírem em seus códigos referências as quatro (4) propriedades que compõe o nível 3. Na cor verde estão representados os 13% dos sites classificados como **Média**.

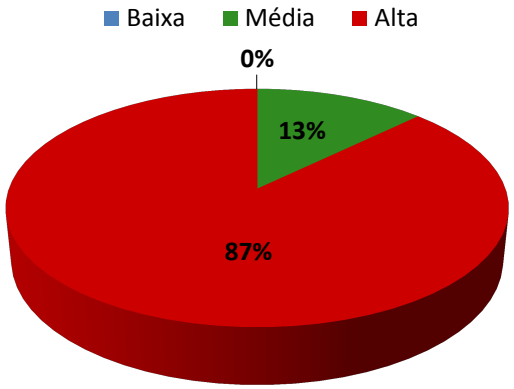


Figura 2. Classificação dos 250 Websites que compõem a Base de Controle

Percebe-se que por se tratar uma base que contêm apenas páginas empregadas para *fingerprinting* não existem artefatos de baixa severidade. Isso não significa que os elementos, objetos e métodos que compõe o Nível 1 não estejam presentes (na verdade, existem mais de 27000 métodos do nível 1 nas 250 páginas da base controle), mas sim que os elementos, objetos e métodos dos outros Níveis 2 e 3 estão presentes e representam um maior risco a privacidade do usuários.

Como forma de melhorar ainda mais o entendimento, traduzindo em números, nas 250 páginas existem 4 referências ao método $navigator.getUserMedia()$, 97 ao método $canvas.getImageData()$, 60 ao método $canvas.toDataURL()$ e 835 ao método $window.history()$.

5.2.2. Base Canvas

A Figura 3 ilustra o resultado da metodologia na base Canvas. Nesta base, observa-se a ocorrência dos 3 níveis de severidade, com maior prevalência para a classificação **Alta** (51%), visto que a base é composta por sites da base de *fingerprinting canvas*.

Diferente da base de controle, na base Canvas existem artefatos de todos os níveis. Nesta base, as páginas classificadas no nível 1 ou não possuem nenhuma referência aos elementos, objetos e métodos deste nível ou embora hajam referências, não existem referências dos outros níveis. Isso apenas comprova que as páginas que compõem esta base possuem artefatos que, de fato, apenas fazem uso correto dos métodos Canvas.

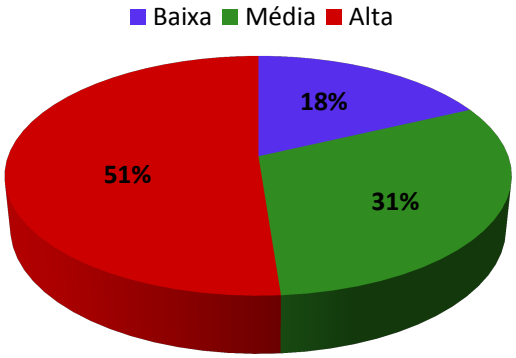


Figura 3. Classificação dos 2.127 Websites que compõem a Base Canvas

5.2.3. Base DMOZ

A Figura 4 ilustra o resultado da metodologia na base DMOZ. Ressalta-se os 33% de sites que possuem alguma das propriedades do nível 3 (no total foram detectados 522 sites para este nível).

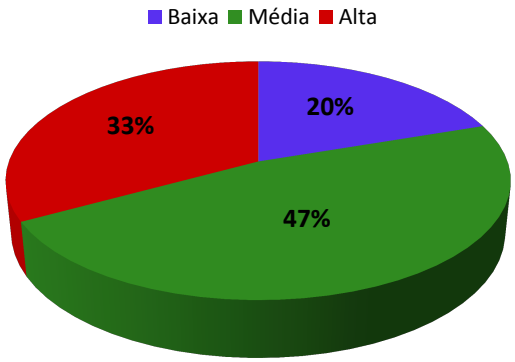


Figura 4. Classificação dos 1.585 Websites que compõem a Base DMOZ

Um aspecto interessante nos resultados desta base é o fato de sites, aleatoriamente escolhidos, de um repositório usado e reconhecido como contendo páginas benignas, apresentar artefatos de todos os níveis. Percebe-se ainda claramente que alguns destes sites tem um grande potencial para invasão de privacidade. Como forma de melhor ilustrar os resultados na base DMOZ, a Tabela 4 enumera as ocorrências de alguns dos termos que compõe o dicionário nesta base.

Os termos do **nível 1 - Baixa**, mencionados na Seção 4, são comumente usados para fornecer informações sobre o dispositivo do usuário. Por exemplo, para o site $pranapoweryoga.com$ houveram ocorrências para as propriedades $innerWidth$, $userAgent$, $screen.height$ e $screen.width$, indicando claramente a obtenção dessas informações para ajustes de tela e conteúdo.

Na coluna do **nível 2 - Média**, ganha destaque o site $bbc.com$ apresentando todos os termos dos níveis 1 e 2, mostrando que além de obter as informações de ajustes de tela, houve a verificação dos plugins e mimeTypes existentes no navegador.

Por fim, sobre as ocorrências do **nível 3 - Alta**, observa-se a prevalência dos termos de canvas como diferencial, uma vez que são usados para obtenção de informações

Tabela 4. Ocorrência de Termos encontrados nos sites da Base DMOZ

Websites	Nível 1 - Baixa					Nível 2 - Média					Nível 3 - Alta			
	innerWidth	innerHeight	userAgent	screen.height	screen.width	referrer	cookie	mimeTypes	plugins	localStorage	getUserMedia	getImageData	toDataURL	history
robertkleingallery.com	0	0	0	2	2	0	0	0	0	0	0	0	0	0
pranapoweryoga.com	1	0	3	2	2	0	0	0	0	0	0	0	0	0
joegreenephoto.com	1	5	3	1	0	0	0	0	0	0	0	0	0	0
captel.com	0	0	0	1	1	0	0	0	0	0	0	0	0	0
dynamicpost.co.uk	5	5	1	0	0	0	0	0	0	0	0	0	0	0
olddogpaws.com	0	0	2	2	3	1	36	0	3	0	0	0	0	0
ilight-tech.com	9	4	8	1	1	1	2	4	8	0	0	0	0	0
bbc.com	10	7	11	32	32	52	118	6	9	7	0	0	0	0
gourmetsleuth.com	6	6	22	6	6	12	6	16	0	0	0	0	0	0
nordicacademy.com.au	4	4	1	0	0	0	3	3	5	0	0	0	0	0
nikonusa.com	29	30	37	10	12	11	49	14	58	8	0	0	0	19
flairstrips.com	14	14	17	0	3	0	12	13	3	0	0	6	1	2
robotshop.com	8	8	13	3	3	7	13	4	6	4	0	3	1	1
fibergarden.com	13	17	9	2	2	5	20	4	5	4	0	3	1	4
newscooters4less.com	5	3	0	0	0	0	0	4	0	0	0	1	1	9

relevantes e que podem identificar o usuário unicamente na Web (conforme descrito em detalhes na Seção 4).

6. Dificuldades encontradas

Com o intuito de detectar *fingerprinting* em páginas Web, inicialmente foi pensada a utilização de grafo de dependência, grafos direcionados que representam relações de dependência entre elementos pertencentes a uma mesma estrutura. A ideia era criar um base de grafos dos scripts de *fingerprinting* que seriam comparados com os script extraídos de páginas Web em tempo real. Contudo, durante a pesquisa percebeu-se que não seria possível analisar as propriedades *fingerprinting* por meio do grafo de dependência, pois a principal finalidade do trabalho é detectar o uso de uma propriedade específica e não da estrutura como a mesma está disposta no código.

Outra ideia foi fazer uso de recuperação de informação, mais especificamente do Modelo Vetorial[8]. A ideia é que cada documento (página Web) fosse representado como um vetor de termos e cada termo possuísse um valor associado que indicasse o grau de importância (peso) deste termo em determinado documento. O ranqueamento dos scripts de *fingerprinting* foi analisado com o classificador k-NN. Contudo, nos resultados percebeu-se que todos os sites possuem algumas das propriedades de *fingerprinting*. Assim, sites rotulados como não *fingerprinting* foram ranqueados como *fingerprinting*.

Foi neste ponto da pesquisa que percebeu-se que praticamente todos os sites possuem alguma propriedade de *fingerprinting*. Assim, surgiu a ideia de classificar os sites pelo nível de severidade de acordo com o nível de risco que as propriedades detectadas nos scripts podem causar ao usuário.

[8]É um modelo onde documentos e consultas são vistos como característica num espaço vetorial n-dimensional, sendo a distância vetorial usada como medida de similaridade

7. Conclusões

As técnicas de *fingerprinting* aplicadas a páginas Web estão se tornando cada vez mais presentes e comuns, mesmo assim os usuários não são informados sobre isso. Embora as formas, os mecanismos e suas consequências sejam conhecidos, questões como a privacidade dos usuários e as formas de evitar a coleta de informações ainda são motivo de discussão. A existência de uma indústria de propaganda especializada, evoluída e financeiramente feliz, aliada ao surgimento de novos serviços e tecnologias, assim como, a constante evolução tecnológica e ao crescente número de usuários (em atividades que envolvem dados pessoais e valores) impõe novos desafios na atividade de detectar e mitigar o *fingerprinting*.

Este trabalho propôs uma metodologia para detectar artefatos (scripts) de *fingerprinting* em páginas Web e informar aos usuários o quão perigosa a página é para sua privacidade. Para tanto, o conjunto de propriedades e métodos dos objetos HTML DOM foram avaliados e o nível de risco à privacidade dos usuários foi estimado. Como forma de prová-la, várias bases de páginas Web contendo ou não *fingerprinting*, foram testadas e o resultado mostra que os artefatos para este fim são comuns e presentes em grande parte das páginas na Internet.

Referências

Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., and Preneel, B. (2013). Fpdetective: Dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 1129–1140, New York, NY, USA. ACM.

Boda, K., Földes, A. M., Gulyás, G. G., and Imre, S. (2012). User tracking on the web via cross-browser fingerprinting. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7161 LNCS:31–46.

Bos, B., Celik, T., Hickson, I., and Lie, H. W. (2011). Cascading style sheets level 2 revision 1 (css 2.1) specification. W3c recommendation, W3C. `http://www.w3.org/TR/CCS2`.

BrowserLeaks.com (2015). Web browser security ? browserleaks.com. `https://www.browserleaks.com`.

Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and Smith, R. (2013). Privacy Considerations for Internet Protocols. RFC 6973 (Informational). `http://www.ietf.org/rfc/rfc6973.txt`.

Crockford, D. (2008). *JavaScript - the good parts: unearthing the excellence in JavaScript*. O'Reilly.

Eckersley, P. (2010). How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 1–18, Berlin, Heidelberg. Springer-Verlag.

Englehardt, S. and Narayanan, A. (2016). Online tracking: A 1-million-site measurement and analysis.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard). http://www.ietf.org/rfc/rfc2616.txt.

Forshaw, J. (2011). WebGL - A New Dimension for Browser Exploitation. http://www.contextis.com/resources/blog/webgl-new-dimension-browser-exploitation/.

Group, K. (2014). Webgl 2 specification. Khronos draft, Khronos Group. https://www.khronos.org/registry/webgl/specs/latest/2.0/.

Khademi, A. F. (2014). Browser Fingerprinting : Analysis , Detection , and Prevention at Runtime. (October).

Kirk, J. (2014). Canvas fingerprinting online tracking is sneaky but easy to halt. http://www.pcworld.com/article/2458280/canvas-fingerprinting-tracking-is-sneaky-but-easy-to-halt.html.

Mowery, K., Bogenreif, D., Yilek, S., and Shacham, H. (2011). Fingerprinting information in JavaScript implementations. In *Proceedings of Web 2.0 Security and Privacy 2011 (W2SP)*, San Franciso.

Mowery, K. and Shacham, H. (2012). Pixel perfect: Fingerprinting canvas in HTML5. In Fredrikson, M., editor, *Proceedings of W2SP 2012*. IEEE Computer Society.

Mulazzani, M., Huber, M., Leithner, M., and Schrittwieser, S. (2013). Fast and reliable browser identification with javascript engine fingerprinting. In *Web 2.0 Workshop on Security and Privacy*, (W2SP).

Nikiforakis, N., Acar, G., and Saelinger, D. (2014). Browse at your own risk. *Spectrum, IEEE*, 51(8):30–35.

Nikiforakis, N., Joosen, W., and Livshits, B. (2015). PriVaricator: Deceiving Fingerprinters with Little White Lies.

Olejnik, L., Castelluccia, C., and Janc, A. (2013). On the uniqueness of Web browsing history patterns. *Annals of Telecommunications - Annales Des Télécommunications*, 69(1-2):63–74.

Saraiva, A. R., Elleres, P. A., Carneiro, G. B., and Feitosa, E. (2014). Device fingerprinting: Conceitos e técnicas, exemplos e contramedidas. In *Livro de Minicursos do XIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg14)*, Belo Horizonte, MG, Brasil. SBC.

Tian, Y., Liu, Y. C., Bhosale, A., Huang, L. S., Tague, P., and Jackson, C. (2014). All your screens are belong to us: Attacks exploiting the html5 screen sharing api. In *2014 IEEE Symposium on Security and Privacy*, pages 34–48.

Unger, T., Mulazzani, M., Fruhwirt, D., Huber, M., Schrittwieser, S., and Weippl, E. (2013). Shpf: Enhancing http(s) session security with browser fingerprinting. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pages 255–261.

W3C (2014). Fingerprinting guidance for web specification authors. http://w3c.github.io/fingerprinting-guidance/.

`VoiDbg`: Projeto e Implementação de um *Debugger* Transparente para Inspeção de Aplicações Protegidas

Marcus Botacin[1], **Paulo Lício de Geus**[1], **André Grégio**[2]

[1]Instituto de Computação (IC)
Universidade Estadual de Campinas (Unicamp)
Campinas – SP – Brasil

[2]Departamento de Informática (DInf)
Unifersidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

Abstract. *Debuggers are important tools for software development, as they support code inspection and, consequently, its validation. In systems security, debuggers can be used to assist malware analysis and reverse engineering, allowing researchers to investigate several execution paths. However, both legitimate software (for intellectual property protection) and malware (for detection avoidance) are often equipped with anti-debug techniques. Therefore, we need to accomplish transparency to overcome these techniques. To do so, we introduce* `VoiDbg`*, a novel debugger able to analyze protected software in a transparent way through hardware monitoring support. We present* `VoiDbg`*'s design and implementation, as well as tests to validate it.*

Resumo. *Debuggers são ferramentas importantes no desenvolvimento de software, pois auxiliam na inspeção de código e, com isso, sua validação. Em segurança de sistemas, debuggers podem ser usados em análise de malware e engenharia reversa, permitindo a investigação de vários caminhos de execução de aplicações. Entretanto, programas legítimos (para proteção de propriedade intelectual) e maliciosos (para evitar detecção) podem ser equipados com técnicas de anti-debug. Logo, sua inspeção deve ser feita transparentemente. Para tanto, introduz-se o* `VoiDbg`*, um debugger inovador capaz de analisar programas protegidos de modo transparente via suporte de monitoração em hardware. Além do projeto e implementação, apresenta-se testes de validação do* `VoiDbg`*.*

1. Introdução

Debuggers são ferramentas fundamentais para o processo de desenvolvimento e análise de *software*. Por meio de seu uso, pode-se identificar falhas de implementação no programa analisado, tais como *memory leaks*, *pointer dereference*, *use-after-free*, entre outras. Seu uso é também importante para as etapas de validação, na qual se pode verificar o correto funcionamento de partes do sistema, incluindo trechos críticos como tratamento de exceções e erros. Os *Debuggers* têm ainda um papel importante para segurança computacional, uma vez que podem ser aplicados a procedimentos de engenharia reversa de *software* em geral e análise de *malware*. No primeiro caso, pode-se inspecionar um dado programa em busca de "artefatos", como códigos indesejados, intrusivos e *backdoors*. No segundo, pode-se obter traços de execução do programa monitorado de forma a se compreender a abrangência de uma potencial infecção pelos caminhos possíveis de serem seguidos. Em ambos os casos, o uso do *debugger* auxilia a tomada das contra-medidas mais apropriadas.

No entanto, o *software* moderno (e em especial o *malware*) conta com muitas técnicas de anti-análise, seja para efeitos de proteção de direitos autorais ou para evitar a detecção e alcançar seu objetivo malicioso. Tais técnicas, quando aplicadas contra um *debugger* tradicional, impedem ou dificultam a análise do programa. Assim, surgiram propostas na literatura de métodos e técnicas para monitoração transparente. Dessa forma, o *software* sob análise não deve perceber que a inspeção de seu funcionamento está ocorrendo, o que é uma extensão do princípio de Heisenberg da minimalidade da intrusão [Rosenberg 1996]. Dentre as diferentes formas de monitoração transparente, destacam-se aquelas que fazem uso de suporte de *hardware* para a extração de dados, dado que esta ocorre em um nível mais privilegiado do que o de execução do *software* sob análise. Neste artigo, introduz-se o `VoiDbg`, uma nova forma de implementar um *debugger* transparente contando com o auxílio do *hardware*. O protótipo proposto se utiliza de medidores de desempenho do processador para possibilitar a inspeção de aplicações executáveis no sistema operacional Windows 8.

O restante deste artigo está organizado como segue: a Seção 2 apresenta os requisitos do procedimento de *debugging* e uma breve revisão da implementação desses mecanismos, bem como os trabalhos relacionados; na Seção 3, introduz-se o funcionamento do mecanismo de *hardware* utilizado para controlar o processo de *debugging* transparente e discute-se as decisões de projeto e as características do `VoiDbg`; na Seção 4, discute-se as especifidades de implementação do `VoiDbg`; a Seção 5 mostra os testes e resultados para validação de transparência e funcionalidade do protótipo; o artigo é concluído na Seção 6.

2. *Debuggers*: visão geral e trabalhos relacionados

Esta seção apresenta os requisitos mínimos para o funcionamento de um *debugger* e analisa como as implementações atuais suprem os mesmos. Além disso, são mostrados trabalhos relacionados à criação de *debuggers* transparentes e não-convencionais.

2.1. Visão geral

De uma forma ampla, um *debugger* deve satisfazer os seguintes requisitos:

1. **Execução em pequenos passos (granularidade)**: O *debugger* deve permitir a execução/inspeção de código de forma granular, em passos tão pequenos quanto se queira—de *single-step* a chamadas de funções—a fim de compatibilizar a região de interesse ou alvo da depuração com as informações providas. Maior granularidade tem a vantagem de se poder inspecionar estados menores, identificando com maior precisão a causa da mudança de estados. Este requisito faz com que abordagens de coleta de dados baseadas em amostragem (*probing*) e simples suspensão de processos não sejam adequadas ao processo de *debugging*.
2. **Identificação de estados (ponto de parada)**: A execução deve garantir que o ponto de parada seja conhecido, isto é, que se tenha previsibilidade sobre a execução de código. O requisito de contexto é apresentado em [Rosenberg 1996]. Este, em conjunto com o requisito de granularidade, assegura que a região de interesse seja efetivamente inspecionada. Devido a este requisito, abordagens de amostragem e simples suspensão de processos não são adequadas ao processo de *debugging*, já que não asseguram a parada da execução na região de interesse.
3. **Inspeção de estados**: Dado um estado, deve-se ser capaz de obter informações de contexto sobre o mesmo. Isso possibilita a construção de um mapa da execução de código e transição de estados, que é a base da identificação de características

do *software* sob análise. Tal procedimento pode ser realizado de diferentes maneiras, seja por instrospecção de máquina virtual, do sistema operacional ou mesmo via simples APIs de sistema, uma vez que o processo encontra-se com execução suspensa.

2.1.1. Implementações tradicionais

A seguir, implementações tradicionais de *debuggers* são revisitadas para que se possa contrastar as soluções vigentes com a solução aqui proposta. No âmbito deste artigo, as abordagens de *debug* são classificadas em: suporte do sistema operacional; injeção de código e emulação; e suporte de *hardware*.

Suporte do sistema operacional. Muitos sistemas operacionais fornecem facilidades e recursos para *debug*. Os sistemas GNU/Linux, por exemplo, possuem a interface `ptrace`[1] para controle de processos. Essa interface faz uso da arquitetura de processos do sistema (*fork*[2] + *copy on write*) para controlar os processos-filho , permitindo a execução passo-a-passo, a obtenção de dados de funções e *syscalls* e a instrumentação de código através de *callbacks* para eventos pré-determinados. Muitos utilitários do sistema são construídos sobre o `ptrace`, como o *debugger* GDB [GDB 2016] e o *tracer* `strace`[3]. No entanto, o `ptrace` não é transparente, podendo ser detectado com o uso de seu próprio *framework* (Listagem 1).

Listagem 1. Detecção de uso do ptrace.

```
1 if (ptrace(PTRACE_TRACEME, 0, NULL, 0) == -1)
2 printf("debugged!\n");
```

O sistema operacional Windows também possui facilidades para *debug*. Embora não possa fazer uso direto da arquitetura de processos—uma chamada de `CreateProcess` [Microsoft 2016a] não compartilha contexto com o processo filho—o sistema fornece uma série de APIs [Microsoft 2016c] para controlar outros processos. Um exemplo de uso destas APIs é o WinDbg [Microsoft 2016d], que também não é transparente, pois pode ser identificado pelo uso da API `IsDebuggerPresent` [Microsoft 2016i].

Injeção de código e emulação. Ferramentas como a Frida [Frida 2015] fazem uso da injeção de código para instrumentar *software*. A maior limitação desta abordagem é sua não-transparência devido à possibilidade de se detectar a alteração no fluxo através de um *hash* da própria memória do *software* sob análise. Ferramentas como o DynamoRIO [Bruening et al. 2012] e Valgrind [Nethercote and Seward 2003], por sua vez, fazem uso de emulação para a construção de um ambiente controlado no qual se pode inspecionar em detalhes aspectos como contexto e memória. Esta abordagem pode não ser transparente sob condições específicas.

Suporte de *hardware*. Além de suporte do sistema operacional e da instrumentação de código e emulação, pode-se contar com recursos do processador para a construção de um *debugger*. Processadores modernos [Intel 2011] contam com recursos como registradores de *debug*, *hardware breakpoints* e execução *single-step* via *trap flag*. *Debuggers* como Ollydbg [OllyDbg 2013] fazem uso deste recurso para inspecionar aplicações. Em-

[1] `http://man7.org/linux/man-pages/man2/ptrace.2.html`
[2] `http://linux.die.net/man/2/fork` [3] `http://linux.die.net/man/1/strace`

bora mais eficiente, esta abordagem é limitada em transparência, uma vez que a *trap flag* altera *bytes* de instruções, dispara tratamento de exceções e define *bits* em registradores de controle.

2.2. Trabalhos relacionados

Devido a questões de transparência e do surgimento de novas linguagens e paradigmas de programação, esforços têm sido direcionados ao desenvolvimento de novos *debuggers*, cujos principais são mencionados a seguir.

Inspeção de código. Embora *debuggers* sejam soluções bem estabelecidas, seu nível de abstração ainda é concentrado nos estados do processador e valores de memória, tendo pouco suporte a construções de alto nível. Moldable [Chiş et al. 2015] tenta suprir as diferenças de abstração entre o baixo nível das informações e as construções de linguagens orientadas a objetos propondo um *framework* para lidar com informações de contexto. Outro problema recorrente em *debuggers* é lidar com sistemas multiprocessados/multi-*thread*. As soluções que atuam em paralelo sofrem de problemas de sincronização e de reconstrução de contexto, cuja abordagem data de 1996 [Hood 1996] e, desde então, tenta-se mitigá-los [Mäkelä et al. 2013, Schulz and Mueller 2000]. Várias das soluções propostas fazem uso de virtualização para monitorar o sistema via introspecção. PDB [Ho et al. 2004] usa o Xen para *debugging* em paralelo de *grids* computacionais, enquanto que [Ho and Hand 2005] propõem uma solução baseada em virtualização para lidar com construções de reuso de código. Um desafio recente é o uso de GPUs, pois seu modo de programação requer diferentes formas de *debug*. Para lidar com isto, [Sharif and Lee 2008] propõem um *framework* para interceptação de chamadas e reconstrução do fluxo até a exibição do pixel.

Análise de *malware*. As abordagens de novos *debuggers* para análise de *malware* focam, sobretudo, no aspecto da transparência. Dada a natureza evasiva dos programas maliciosos mais sofisticados, os *debuggers* para tal finalidade fazem uso de recursos de *hardware* para a inspeção. HyperDbg [Fattori et al. 2010] usa os recursos de virtualização do processador para implementar instrospecção de máquina virtual, *breakpoints* e inspeção de registradores e memória, que é capaz de realizar execução *single-step*. MALT [Zhang et al. 2015], por sua vez, usa o modo SMM do processador por meio da reescrita de BIOS para ter controle total do sistema. A maior limitação das abordagens citadas refere-se à sobrecarga de se ter que instrumentar todo um sistema e à dificuldade de implementação, dado que se requer a reescrita do BIOS ou de uma máquina virtual. A abordagem deste artigo busca ser uma versão aprimorada destas abordagens, isto é, com menor sobrecarga.

Abordagens não-tradicionais. Outros aspectos do processo de *debugging*, como a tarefa de *debug* em si, podem ser vistos em [Araki et al. 1991]. Para fins de proteção de código, técnicas de anti-debug têm emergido, como em [Yi et al. 2009], que faz uso de máquinas virtuais para a construção de um código equipado com técnicas de anti-análise. O experimento mostra a incapacidade do WinDBG inspecionar este código. As técnicas de anti-análise estão bem estabelecidas e disseminadas, com literatura específica para "contra-atacar" *debuggers* [Kaspersky 2007].

3. `VoiDbg`: Projeto

Para a construção de um *debugger* transparente, busca-se atender aos três requisitos básicos definidos na Seção 2.1. Contudo, evitamos utilizar as tecnologias apresentadas anteriormente, dadas as limitações expostas. A proposta deste artigo baseia-se em um recurso

de *hardware*, a monitoração de *branches*, diferente dos apresentados anteriormente—*trap flag, debug register, virtual machine instruction set, SMM mode*.

Processadores modernos de diferentes fabricantes possuem variados medidores de desempenho, os quais possibilitam obter informações acerca do uso de memória, desempenho de cache, predição de *branch* e outros recursos. `VoiDbg` usa o mecanismo BTS (*Branch Trace Store*) dos processadores Intel em sua implementação. Tal mecanismo é um recurso do processador que armazena endereços origem e destino de desvios de fluxo de execução de código (instruções `JMP`, `CALL` e `RET`) em uma página de memória do sistema. O BTS permite ainda definir um *threshold* de armazenamento para o qual uma interrupção é gerada no processador, tornando-o um mecanismo adequado para realizar o controle de fluxo de execução de código no `VoiDbg`.

3.1. Decisões de projeto

Para atender o requisito de **execução em pequenos passos**, o mecanismo BTS foi aplicado de modo que a execução ocorra apenas em blocos entre duas instruções de desvio de fluxo. A implementação desse requisito deu-se pela definição de um *threshold* para o mecanismo BTS de uma única instrução de desvio. Esta abordagem tem a mesma ordem de grandeza de inspeção que o avanço em *single-step*.

Para atender o requisito de **identificação de estado**, o mecanismo BTS com interrupções também é considerado. A ocorrência destas garante que a execução esteja suspensa e, portanto, que o endereço fornecido pelo mecanismo seja o do bloco de execução corrente. Ademais, pode-se também reconstruir o contexto atual através da diferença entre o endereço provido na iteração anterior frente a atual.

Por fim, para a **inspeção do estado**, usam-se as APIs do próprio sistema operacional. A consistência dos dados é garantida pela suspensão da execução na interrupção.

A transparência de execução no sistema, entendida como a ausência de efeitos colaterais de execução, é garantida pelo fato da análise ser realizada em ambiente *bare-metal*, dada a necessidade de uso dos recursos específicos do processador, não acessíveis através de emulação por máquinas virtuais.

3.2. Modelo de ameaça, arquitetura da solução e fluxo de análise

A proposta deste artigo considera o *software* sob análise em execução no nível de espaço do usuário. Abordagens de *debug* em *kernel* envolvem um nível de instrumentação incompatível com a proposta de um *lightweight debugger*. Assume-se também que o objeto da análise atue em *single-core*, dado que a reconstrução do fluxo é realizada a partir dos dados do BTS em uma única CPU. A expansão para um cenário *multi-core* é deixada como trabalho futuro.

Além do programa monitorado, o analisador (*debugger*) também é executado em espaço de usuário, de modo a poder fazer uso das APIs fornecidas pelas bibliotecas de sistema. A coleta dos dados, contudo, só pode ser realizada em *kernel*, pois exige acesso a registradores especiais (MSRs). O tratamento da interrupção gerada pelo mecanismo BTS também só pode ser realizada em *kernel*. Deste modo, a coleta e o tratamento foram implementados através de um *driver* de *kernel*.

Para a comunicação de eventos e dados entre o *kernel* e o espaço de usuário são usadas rotinas de I/O. Porém, como a rotina de interrupção deve ser tratada instantaneamente para permitir a execução granular, o I/O deve ser implementada de forma assín-

crona, gerando aviso ao cliente de *debug* no momento da ocorrência do evento. Tem-se então um fluxo em que o programa em execução realiza suas computações (Figura 1).

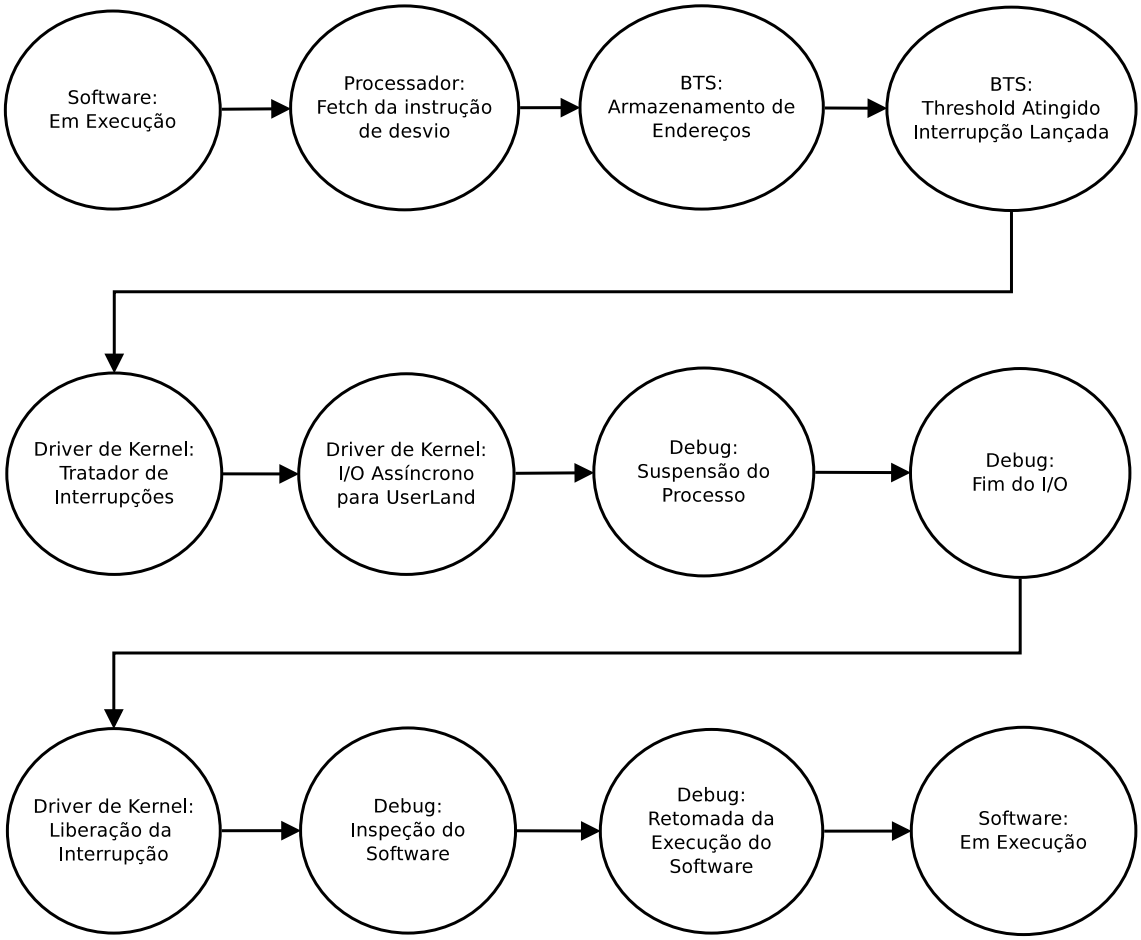


Figura 1. Fluxo de execução de um programa monitorado.

Em determinado momento da execução, o processador realiza o *fetch* de uma instrução de desvio relacionada às computações de um dado programa, cujos endereços de origem e destino são armazenados pelo mecanismo BTS. Uma vez que o *threshold* é de uma única instrução, um sinal de interrupção é disparado. O processo encontra-se em estado ativo, porém com sua execução suspensa devido à interrupção no processador. O sinal de interrupção é redirecionado pelo *kernel* para a rotina de tratamento do *driver*, a qual inicia uma chamada de I/O assíncrona para o cliente. O cliente recebe esta chamada e suspende a execução do processo em definitivo por meio de API do sistema. A chamada de I/O é finalizada, de modo que o *driver* possa terminar o tratamento de interrupção. Mesmo assim, o processo não é retomado pois seu estado mudou para "suspenso". Devido a esse estado, o cliente de *debug* pode realizar a inspeção do estado atual através das APIs do sistema. Após a obtenção de todos os dados desejados, resume-se a execução do processo, através de APIs do sistema, de modo a torná-lo ativo. O processo é retomado quando da geração da próxima interrupção.

3.3. Recursos de inspeção

O uso combinado do mecanismo BTS com as APIs de sistema permite às seguintes operações serem realizadas pelo `VoiDbg`:

- Criação de um novo processo para ser inspecionado;
- Inspeção de um processo corrente;
- Suspensão de um processo;
- Retomada de um processo;
- Identificação do estado atual (funções/bibliotecas);
- Avanço ao próximo bloco;
- Leitura de valores de memória;
- Leitura das bibliotecas carregadas;
- Inspeção de registradores de contexto;
- Integração com outras soluções de *debuggers*.

4. `VoiDbg`: implementação

A seguir, descreve-se a implementação do `VoiDbg` em seus diversos níveis.

Captura de dados com BTS. O processo de configuração e a extração dos dados do mecanismo BTS são realizados diretamente em *kernel* por meio de um *driver*. A configuração é feita através de registradores especiais (*Model Specific Register* - MSR) e, para o armazenamento BTS, é fornecida uma página do sistema operacional.

Tratamento de interrupções. Utilizou-se no protótipo desenvolvido o tratamento de interrupção via *Non Maskable Interrupt* (NMI), cujo tratamento é obrigatório no instante em que o evento ocorre. Para isso, redireções na tabela *Advanced Programmable Interrupt Controler* (APIC) são necessárias. A configuração deve ser realizada em cada *core* caso o sistema seja multi-processado. O tratamento de uma NMI é regulado por um *watchdog* para que o processamento abusivo não cause congelamento do sistema. Desse modo, a única operação realizada dentro da NMI é o disparo da rotina de I/O assíncrono. Demais obtenções de dados são realizadas através de IOCTLs [Microsoft 2016e] após a suspensão do processo.

Disparo e suspensão de processos. O disparo de um novo processo a ser inspecionado é realizado diretamente pela chamada à função `CreateProcess` com o parâmetro CREATE_SUSPENDED. Já a suspensão de processos em execução, seja para anexação do *debugger* ou para sua inspeção após a ocorrência de uma interrupção, é feita através de APIs do sistema. Três métodos são conhecidos para a suspensão de processos: (i) a enumeração de todas as *threads* de um processo e a suspensão destas com a chamada à `SuspendThread` [Microsoft 2016n]; (ii) a chamada a uma função não documentada da biblioteca NTDLL (`NtSuspendProcess`); (iii) a chamada à `DebugActiveProcess` [Microsoft 2016b]. Dado que esta última pode ser identificada através da chamada à `IsDebuggerPresent` e que a primeira pode gerar um *deadlock* devido à não-sincronização da suspensão, a implementação do `VoiDbg` usa `NtSuspendProcess`.

Identificação do processo-alvo. O mecanismo BTS não fornece nenhum filtro no nível dos processos, fazendo com que todas as mudanças de fluxo de todos os processos em execução em uma dada CPU sejam armazenadas. Como o interesse deste trabalho reside nos dados de apenas um processo, faz-se necessário identificar quando o dado armazenado se refere a este. Para isso, o PID de interesse é carregado pelo cliente no *driver* via IOCTL e é comparado com o PID obtido da execução corrente através de `PsGetCurrentProcessId` [Microsoft 2016l].

Step-into vs. Step-over. Parte considerável da execução de um programa não se refere ao código presente em seu próprio binário, mas sim em bibliotecas dinamicamente

ligadas. Estas executam em espaço de usuário, consequentemente ativando o mecanismo de captura BTS. Como as bibliotecas são mapeadas e relocadas dentro do espaço de endereçamento do processo, sua execução é entendida como execução do código monitorado. Assim, ao efetuar o *debugging* de um programa, inspeciona-se tanto o código do *software* quanto das bibliotecas utilizadas por ele em um modo chamado *step-into*. Para os casos em que se deseja inspecionar apenas o código do programa sem suas bibliotecas (chamado de modo *step-over*), deve-se adicionar filtros ao modo nativo de *debugging*.

I/O assíncrono. A comunicação da ocorrência do evento deve ser rápida, logo a escolha pela implementação de I/O assíncrono. No `VoiDbg`, usa-se a técnica conhecida como *Inverted call*, na qual o programa cliente solicita um dado (evento) que, se não estiver disponível no servidor (*driver*), faz com que a chamada de I/O seja colocada em uma fila. O programa cliente passa então a escutar em uma porta de I/O. O *driver*, por sua vez, quando obtém o dado requisitado (interrupção), retira o I/O pendente da fila e o dispara.

Identificação de estado (introspecção). O mecanismo BTS fornece a posição da memória de instruções na qual a execução do programa se encontra em um dado momento. Contudo, este resultado traz pouca informação semântica. Para aumentar o nível de informação provida, aplica-se um processo chamado de instrospecção, através do qual pode-se identificar a biblioteca ou função onde o programa está. Para a introspecção, inicialmente obtém-se o endereço de todas as bibliotecas carregadas no sistema por meio de enumeração [Microsoft 2016f] e *dump* de endereços [Microsoft 2016g] (necessário devido ao processo de aleatorização (ASLR)). Após identificar a biblioteca com endereço compatível com o atual, busca-se a função desta chamada pelo programa monitorado.

Acesso à memória. A inspeção de memória é um recurso que permite reconstruir o estado atual da execução. A obtenção de dados da região de memória do processo monitorado é realizada diretamente através da API `ReadProcessMemory` [Microsoft 2016m]. A criação do processo pelo `VoiDbg` assegura a obtenção de permissões para o procedimento citado.

Bibliotecas carregadas. A verificação de bibliotecas carregadas é uma etapa importante do processo de inspeção de *software*, pois pode indicar a função de um módulo inspecionado deste (pela ligação com uma DLL de criptografia, por exemplo) ou mesmo sua integridade (*hooks* de modo usuário são feitos por meio da injeção de uma biblioteca). A inspeção de módulos carregados é possível com o uso da API `GetModuleHandle`.

Valores de Contexto. Valores de contexto são importantes para a reconstrução/identificação do estado atual do processo. Tais valores incluem os dados nos registradores gerais, o apontador de pilha e as *flags* marcadas. A obtenção desses dados envolve a enumeração de todas as *threads* do processo monitorado e o uso da API `GetThreadContext` [Microsoft 2016h]. A consistência do estado é garantida, dada a suspensão da execução pelo *debugger*.

Integração com outras soluções. Embora o `VoiDbg` forneça um cliente remoto para a inspeção dos dados, a integração com soluções já consolidadas é importante para tornar possível o uso de ferramentas legadas e *scripts* feitos utilizando-as. Para tanto, foi implementada a integração de `VoiDbg` com o cliente do GDB, permitindo seu uso e de suas ferramentas. Pode-se, inclusive, efetuar a inspeção de um cliente Windows a partir de um sistema Linux. A implementação que serviu como referência para o *stub* do GDB compatível com o `VoiDbg` é a disponível em [mseaborn 2014], com a devida migração

da plataforma Linux para Windows. A integração com GDB possibilita a inspeção de registradores, de memória e o avanço de *branches* através do comando `next`. O servidor que implementa o *stub* é responsável por converter o resultado provido pelas APIs do Windows para as estruturas de dados esperadas pelo GDB.

5. Testes e resultados

A seguir, são exibidos os resultados obtidos com os testes do `VoiDbg`, nos quais validou-se o funcionamento de todas as funcionalidades planejadas e avaliou-se sua transparência, desempenho e limitações. A Listagem 2 mostra a suspensão de um processo já em execução, enquanto que a Listagem 3 mostra a criação de um novo processo em estado suspenso. Já a Listagem 4 mostra o avanço da execução de *branch* em *branch*.

Listagem 2. Suspensão de processo.

```
4488 Suspended > _
```

Listagem 3. Criação de processo.

```
Sending Sample ToyProgram.exe
READ 7680 bytes
> Created suspended process PID 1268
```

Listagem 4. *Branch-by-branch steps*.

```
> n
> next branch
<0x1ba68fe9> BRANCH TO 0x33f2f267
> n
> next branch
<0x33f2f267> BRANCH TO 0x33f2e4b7
```

A Listagem 5 mostra a instrospecção do endereço destino. Neste, o endereço é identificado como sendo referente à biblioteca `MSVCR110D.dll`. A Listagem 6 mostra o *dump* de memória de 10 bytes a partir de um endereço interior ao processo sendo analisado.

Listagem 5. Introspecção.

```
> next branch
<0x1ba68fe9> BRANCH TO 0x33f2f267
> I
> Introspect To
> MSVCR110D.dll
```

Listagem 6. Valores de memória.

```
> dump memory
7f7d6d71005
10
> \xe9\x16\x00\x00\x00\xe9\x51\x00\x00\x00\xcc\xcc\xcc\xcc\xcc\
    xcc
```

A Listagem 7 mostra as bibliotecas carregadas pelo executável. Neste caso, apenas módulos do sistema são identificados. A Listagem 8 exibe a inspeção dos registradores de contexto.

Listagem 7. Bibliotecas carregadas.

```
> Process 4488 suspended
> l
> list loaded libs
> C:\Windows\SYSTEM32\ntdll.dll
> C:\Windows\system32\KERNEL32.DLL
> C:\Windows\system32\KERNELBASE.dll
> C:\Windowz\SYSTEM32\MSVCR110D.dll
```

Listagem 8. Registradores de contexto.

```
> C
> Context
> R10 2
> R11 12851598
> R12 12851e00
> R13 0
> R14 0
> R15 12851e00
> RAX 1
> RBX 47e2f4e4
> RCX 0
> RDX 127f1bc
> RSP 47e2f3e8
> EFLAG 246
```

Para validar a integração do `VoiDbg` com o GDB, a porção de código ilustrada na Listagem 9 foi executada em uma sessão de *debugging* remoto. A Figura 2 mostra a exibição desses valores na sessão remota usando o GDB.

Listagem 9. Porção de código ASM.

```
int main()
        __asm{mov eax,1}
        __asm{mov ecx,2}
        __asm{mov edx,3}
        __asm{mov ebx,4}
```

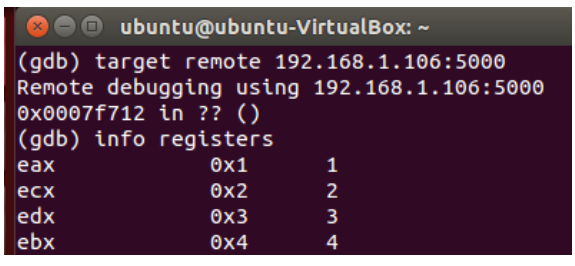


Figura 2. Integração com GDB.

5.1. Detecção do `VoiDbg`

Esta seção divide-se entre testes experimentais diretos e uma breve discussão sobre a transparência do protótipo implementado. Do ponto de vista teórico, o `VoiDbg` não é

passível de injeções de código ou uso de APIs que possibilitem sua identificação. Para validar experimentalmente este ponto, foi criado um trecho de código que usa a API `IsDebuggerPresent`, exibido na Listagem 10. Este código foi submetido para execução no `VoiDbg` e o resultado é exibido na Figura 3, na qual pode-se observar que a anexação do *debugger* não foi detectada.

Listagem 10. Identificação de *debug*.

```
1        if(IsDebuggerPresent())
2                printf("debugged\n");
3        else
4                printf("NO DBG\n");
```

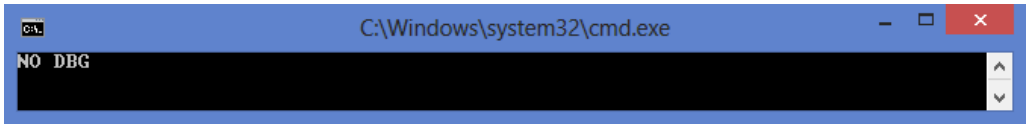


Figura 3. Detecção de *debugger* via API do SO.

Esta forma de detecção de *debuggers* é muito utilizada para a proteção de conteúdo proprietário, como em *game networks* [Woo and Kim 2012]. Um exemplo desta pode ser visto no executável da `Uplay`[4] que, quando submetido à execução em um *debugger*, produz o erro exibido na Figura 4. O uso de `VoiDbg` para análise do executável não apresentou o mesmo erro, permitindo o *debugging*, como exibido na Figura 5.

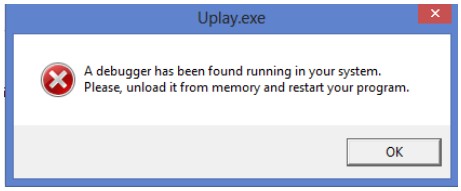


Figura 4. Detecção de *debugger* por uma aplicação legítima.

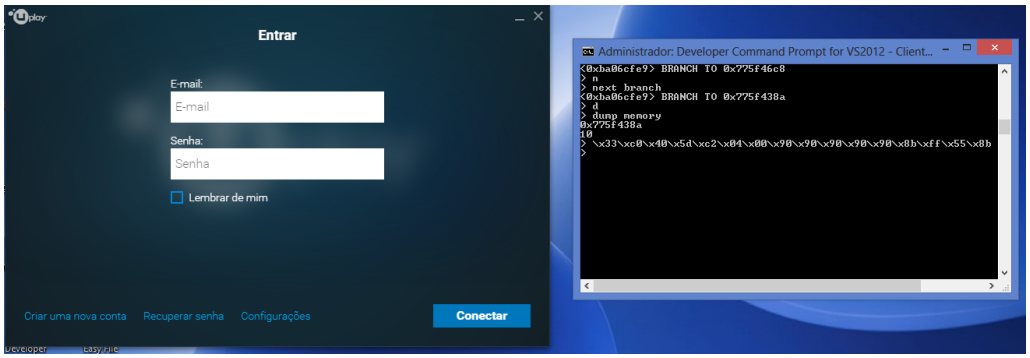


Figura 5. Uso do `VoiDbg` (dir.) com a aplicação legítima protegida (esq.).

Uma forma adicional de detecção poderia se dar através do registrador MSR, que ativa o mecanismo de monitoração de *branch*. Este registrador já é alvo de inspeção por *malware*, pois também é responsável por ativar o suporte de *debug* em *hardware*. Entretanto, os *bits* do mecanismo BTS e de *debug* são diferentes, permitindo a avaliação de que,

[4] `www.uplay.com`

ao contrário dos *bits* de *debugger*, a ativação do mecanismo BTS por si só não pode ser entendida pelo *malware* como uma forma de monitoramento—diversas soluções utilizam-se deles, como o Intel VTune [Intel 2015], o Linux Perf [Linux 2015], e até monitores de desempenho da plataforma .Net [Microsoft 2016k] e do Windows [Microsoft 2016j].

Outra informação de *hardware* que poderia possibilitar a detecção de um processo de *debug* é a variação de tempo na execução. Dado que APIs de tempo do sistema são geralmente não-confiáveis, exemplares de *malware* podem fazer uso da medida de tempo do registrador de *timestamp counter* (TSC) para verificar a velocidade e continuidade de execução. Embora não se tenha abordado este tipo de contra-medida neste trabalho, esta já foi aplicada por outras soluções, como o HyperDbg e MALT, podendo assim ser abordada em um trabalho futuro.

5.2. Desenvolvimento e Portabilidade

O desenvolvimento da solução em sistema Windows exige apenas a escrita de um *driver* de *kernel*, aplicações em modo usuário a partir de bibliotecas pré-existentes e a utilização de um processador compatível com o monitor de *branch* utilizado, consistindo, de fato, em uma alternativa de menor custo de desenvolvimento em relação ao estado-da-arte (HVM e SMM).

Embora construído sobre o ambiente Windows, o sistema é portável para outras plataformas, já que o recurso de *hardware* é independente do sistema utilizado. Nestas plataformas, bibliotecas compatíveis devem ser utilizadas para os processos de introspecção e análise.

6. Conclusão

Neste artigo, apresentou-se o `VoiDbg`, um *debugger* leve e transparente com base no mecanismo BTS de monitoração de *branch* e em APIs do sistema. A solução proposta permite inspecionar o estado das aplicações em Windows 8 e possui integração com o cliente GDB para monitoração remota. O `VoiDbg` pode ser considerado leve em comparação com abordagens no estado-da-arte, pois não exige a reescrita do BIOS ou a construção de uma máquina virtual. O diferencial do trabalho consiste em ser uma solução cuja transparência é alcançada com o auxílio de recursos de *hardware*, permitindo até mesmo a inspeção de *software* com proteção *anti-debug*. Com isso, abre-se também a possibilidade de análise de *malware* equipado com mecanismos de anti-forense. Maiores informações sobre a solução proposta podem ser encontradas na páginas *web* da solução[5].

Agradecimentos

Os autores agradecem o apoio recebido do Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq via Projeto MCTI/CNPq/Universal-A 14/2014 (Processo 444487/2014-0) e da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES, em especial via Projeto FORTE - Forense Digital Tempestiva e Eficiente (Processo: 23038.007604/2014-69 - Edital 24/2014 - Programa Ciências Forenses).

Referências

Araki, K., Furukawa, Z., and Cheng, J. (1991). A general framework for debugging. *IEEE Software*, 8(3):14–20.

[5] `https://sites.google.com/site/branchmonitoringproject/`

Bruening, D., Zhao, Q., and Amarasinghe, S. (2012). Transparent dynamic instrumentation. In *8th ACM SIGPLAN/SIGOPS Conf. Virtual Execution Environments*, VEE '12, pages 133–144.

Chiş, A., Denker, M., Gîrba, T., and Nierstrasz, O. (2015). Practical Domain-specific Debuggers Using the Moldable Debugger Framework. *Comput. Lang. Syst. Struct.*, 44(PA):89–113.

Fattori, A., Paleari, R., Martignoni, L., and Monga, M. (2010). Dynamic and Transparent Analysis of Commodity Production Systems. In *Proc. IEEE/ACM Intl. Conf. on Automated Software Engineering*, ASE '10, pages 417–426, New York, NY, USA. ACM.

Frida (2015). Inject javascript to explore native apps. `http://www.frida.re/`.

GDB (2016). GDB: The GNU project debugger. `www.gnu.org/software/gdb`.

Ho, A. and Hand, S. (2005). On the Design of a Pervasive Debugger. In *Proc. Sixth Intl. Symp. on Automated Analysis-driven Debugging*, AADEBUG'05, pages 117–122, NY, USA. ACM.

Ho, A., Hand, S., and Harris, T. (2004). Pdb: pervasive debugging with xen. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 260–265.

Hood, R. (1996). The p2d2 project: Building a portable distributed debugger. In *Proc. SIGMETRICS Symp. on Parallel and Distributed Tools*, SPDT '96, pages 127–136, NY, USA. ACM.

Intel (2011). Intel 64 and ia-32 architectures software developer's manual. `http://www.intel.com/Assets/en_US/PDF/manual/253668.pdf`.

Intel (2015). Intel Vtune. `software.intel.com/en-us/intel-vtune-amplifier-xe`.

Kaspersky, K. (2007). *Hacker Disassembling Uncovered (Uncovered Series)*. A-List Publishing.

Linux (2015). Linux perf. `perf.wiki.kernel.org/index.php/Main_Page`.

Mäkelä, J.-M., Leppänen, V., and Forsell, M. (2013). Towards a parallel debugging framework for the massively multi-threaded, step-synchronous replica architecture. In *Proc. 14th Intl. Conf. Computer Systems and Technologies*, CompSysTech '13, pages 153–160, NY, USA. ACM.

Microsoft (2016a). Createprocess function. `https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425%28v=vs.85%29.aspx`.

Microsoft (2016b). Debugactiveprocess function. `https://msdn.microsoft.com/pt-br/library/windows/desktop/ms679295%28v=vs.85%29.aspx`.

Microsoft (2016c). Debugging functions. `https://msdn.microsoft.com/en-us/library/windows/desktop/ms679303%28v=vs.85%29.aspx`.

Microsoft (2016d). Debugging tools for windows. `https://msdn.microsoft.com/en-us/library/ff551063.aspx`.

Microsoft (2016e). Device input and output control (ioctl). `https://msdn.microsoft.com/pt-br/library/windows/desktop/aa363219%28v=vs.85%29.aspx`.

Microsoft (2016f). Enumprocessmodules function. `https://msdn.microsoft.com/en-us/library/windows/desktop/ms682631%28v=vs.85%29.aspx`.

Microsoft (2016g). Getmodulehandle function. `https://msdn.microsoft.com/pt-br/library/windows/desktop/ms683199%28v=vs.85%29.aspx`.

Microsoft (2016h). Getthreadcontext function. `https://msdn.microsoft.com/pt-br/library/windows/desktop/ms679362\%28v=vs.85\%29.aspx`.

Microsoft (2016i). Isdebuggerpresent. https://msdn.microsoft.com/pt-br/library/windows/desktop/ms680345%28v=vs.85%29.aspx.

Microsoft (2016j). Performance counters. https://msdn.microsoft.com/pt-br/library/windows/desktop/aa373083\%28v=vs.85\%29.aspx.

Microsoft (2016k). Performancecounter class. msdn.microsoft.com/en-us/library/system.diagnostics.performancecounter\%28v=vs.110\%29.aspx.

Microsoft (2016l). Psgetcurrentprocessid routine. https://msdn.microsoft.com/en-us/library/windows/hardware/ff559935%28v=vs.85%29.aspx.

Microsoft (2016m). Readprocessmemory function. https://msdn.microsoft.com/pt-br/library/windows/desktop/ms680553%28v=vs.85%29.aspx.

Microsoft (2016n). Suspendthread function. https://msdn.microsoft.com/pt-br/library/windows/desktop/ms686345%28v=vs.85%29.aspx.

mseaborn (2014). gdb-debug-stub. github.com/mseaborn/gdb-debug-stub.

Nethercote, N. and Seward, J. (2003). Valgrind: A program supervision framework. *Electronic Notes in Theoretical Computer Science*, 89(2):44 – 66.

OllyDbg (2013). Ollydbg. www.ollydbg.de.

Rosenberg, J. B. (1996). *How Debuggers Work: Algorithms, Data Structures, and Architecture*. John Wiley & Sons, Inc., New York, NY, USA.

Schulz, D. and Mueller, F. (2000). A thread-aware debugger with an open interface. In *Proc. 2000 ACM SIGSOFT Intl. Symp. Software Testing and Analysis*, ISSTA '00, pages 201–211.

Sharif, A. and Lee, H.-H. S. (2008). Total recall: A debugging framework for gpus. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, GH '08, pages 13–20, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Woo, J. and Kim, H. K. (2012). Survey and research direction on online game security. In *Proceedings of the Workshop at SIGGRAPH Asia*, WASA '12, pages 19–25.

Yi, T., Zong, A., Yu, M., Gao, S., Lin, Q., Yu, P., Ren, Z., and Qi, Z. (2009). Anti-debugging framework based on hardware virtualization technology. In *Research Challenges in Computer Science, 2009. ICRCCS '09. International Conference on*, pages 218–220.

Zhang, F., Leach, K., Stavrou, A., Wang, H., and Sun, K. (2015). Using hardware features for increased debugging transparency. In *IEEE Symp. Security and Privacy (SP)*, pages 55–69.

Identificação de Códigos Maliciosos Metamórficos pela Medição do Nível de Similaridade de Grafos de Dependência

Gilbert B. Martins[1], Paulo dos Santos[2], Vitor Danrley[2], Eduardo Souto[2], Rosiane de Freitas[2]

[1]Campus Distrito Industrial – Instituto Federal do Amazonas (IFAM)
Manaus – AM – Brasil

[2]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
Manaus – AM – Brasil

{gilbert.breves, paulohsantosol}@gmail.com,
{vdms, esouto, rosiane}@icomp.ufam.edu.br

Abstract. *In order to correctly identify metamorphic malicious code, different approaches try to model structural characteristics that remain valid even after the application code obfuscation techniques. One approach is based on comparing dependency graphs generated from suspicious code with a baseline graphs generated from previously identified malicious code. However, as the graph comparison process is a NP-Hard problem, the development of methodologies comparison that makes possible this process of identification is required. This article presents the results from a methodology that uses the concepts of differentiation of vertices and adapted topological sort, to propose a metric measuring maximum subgraph isomorphism.*

Resumo. Com o objetivo de identificar corretamente códigos maliciosos metamórficos, diferentes abordagens procuram modelar características estruturais que se mantêm válidas, mesmo após a aplicação de técnicas de ofuscação de código. Uma destas abordagens se baseia na comparação de grafos de dependência, extraídos de códigos suspeitos com uma base de grafos referência gerada a partir de códigos maliciosos previamente identificados. Entretanto, como o processo de comparação de grafos é um problema NP-Difícil, se faz necessário o desenvolvimento de metodologias de comparação que tornem viável este processo de identificação. Este artigo apresenta os resultados obtidos a partir de uma metodologia que usa os conceitos de diferenciação de vértices e ordenação topológica adaptada, para propor uma métrica de medição de máximo subgrafo isomorfismo.

1. Introdução

O aumento das atividades maliciosas e a inclusão de recursos de ofuscação de código que mascaram a identidade dos programas maliciosos (*malware*) tornam contínua a necessidade de desenvolvimento de técnicas alternativas de identificação, capazes de estabelecer corretamente a identidade de um código suspeito [Barossa Community Co-operative Store 2014; Borello and Mé 2008].

A princípio, a simples extração de trechos de código de *malware* previamente identificados, também conhecidos como *assinaturas*[Jacob et al. 2009] [Newsome et al. 2005], era suficiente para identificar corretamente outras cópias de um mesmo programa malicioso. Qualquer código suspeito era comparado com uma base de assinaturas de *malware* e, caso uma dessas assinaturas fosse encontrada em uma parte do código sob análise, a identidade do programa era estabelecida e a ameaça tratada.

Entretanto, como este processo de identificação exige um casamento perfeito entre uma parte do código suspeito e uma das assinaturas, os desenvolvedores de códigos maliciosos começaram a procurar formas de alterar a codificação original de suas criações, de maneira a que a identificação por assinaturas pudesse ser inviabilizada.

A princípio, as técnicas empregadas para este fim eram baseadas no uso de criptografia do código original, que era posteriormente reconstruído apenas no momento do ataque ao sistema alvo. A estrutura básica deste tipo de código furtivo envolve dois componentes distintos: um motor criptográfico e um corpo criptografado do *malware* propriamente dito [Rad et al. 2012] [O'Kane et al. 2011][You and Yim 2010].

O motor criptográfico tem duas responsabilidades principais: *i)* a restauração do corpo criptografado ao seu estado original, para que o *malware* possa agir de acordo com os objetivos para o qual foi construído; e *ii)* a criação de um novo corpo criptografado, baseado numa chave aleatória, no momento em que ocorre o processo de propagação. É este processo de criptografia com chave aleatória que possibilita criação de uma nova instância deste código malicioso e que a torna incompatível com qualquer assinatura gerada a partir do corpo criptografado de outra instância em particular.

Os primeiros *malware* que utilizaram essa técnica empregavam um único motor criptográfico, isso permitiu que as assinaturas geradas a partir deste componente fossem suficientes para identificar estes códigos maliciosos. Para fugir desta limitação, sucessivas alterações modificações na técnica original foram produzidas, com o passar do tempo. Iniciando com o uso de um conjunto de motores criptográficos distintos, escolhidos aleatoriamente no momento que o processo de propagação (Oligomorfismo), e culminando a geração de múltiplos motores criptográficos, com o emprego de técnicas de ofuscação de código para este componente do código malicioso (Polimorfismo) [Rad et al. 2012] [O'Kane et al. 2011][You and Yim 2010].

Apesar da técnica de polimorfismo ter o potencial para criar uma quantidade infinita de instâncias de *malware*, ela ainda compartilha o mesmo problema de suas predecessoras. A partir do momento que o corpo principal do *malware* é reconstruído na memória, este pode novamente ser facilmente identificado por uma assinatura. Assim, para explorar esta característica fundamental, as ferramentas de identificação de códigos maliciosos passaram a carregar os programas suspeitos em um ambiente emulado (*sandbox*), onde o código reconstruído é carregado e analisado, sem que qualquer tipo de contaminação venha a ocorrer [You and Yim 2010].

Percebendo que este conjunto de técnicas de furtividade tinha atingido o seu limite e reconhecendo o potencial das técnicas de ofuscação de código que, até então, tinham sido aplicadas apenas no motor criptográfico do código malicioso, os desenvolvedores de códigos maliciosos passaram a trabalhar com na próxima geração de *malware* furtivo: os códigos metamórficos.

Versões metamórficas de um mesmo *malware* possuem um corpo de código construído de forma que as instruções se apresentem de forma distinta do código original no qual se baseiam. Entretanto, diferente do corpo criptografado das gerações anteriores, este novo código é plenamente operacional, mantendo todas as funcionalidades originais intactas e sem a necessidade de reconstrução do código original. Como cada uma destas versões metamórficas também pode gerar novas versões, qualquer assinatura gerada de acordo com a metodologia tradicional de detecção não será capaz de identificar as novas versões metamórficas geradas.

O efeito de metamorfismo é o obtido pela ação em conjunto de técnicas muito simples. Operações como: *i)* a inserção de instruções e variáveis irrelevantes, também conhecida como inserção de código lixo; *ii)* a alteração no nome de variáveis ou troca mútua de variáveis entre instruções diferentes; *iii)* a substituição de sequências de instruções por outras que produzam o mesmo resultado; e *iv)* a alteração na ordem de execução das instruções [Bruschi et al. 2007], são capazes modificar o código o suficiente para que sua assinatura original seja invalidada.

Diferentes abordagens foram propostas para lidar com este problema [Cozzolino et al. 2012; Griffin et al. 2009; Hu et al. 2009; Jacob et al. 2008, 2009; Kim and Moon 2010]. Algumas dessas abordagens se baseiam em modelagens que procuram invalidar as modificações introduzidas pelo metamorfismo [ ] [ ] [ ] [ ], enquanto outras se baseiam na construção de grafos de dependência[You and Yim 2010] para o mapeamento das relações de interdependência entre componentes de código. Exemplos de elementos que podem ser mapeados por estes grafos incluem: chamadas de funções [Hu et al. 2009], uso de variáveis por instruções [Kim and Moon 2010] e chamadas bibliotecas de APIs (*Aplication Program Interfaces*) [Elhadi et al. 2014].

Como todas as abordagens baseadas em grafos de dependência lidam com o problema NP-Difícil da identificação do nível de similaridade entre grafos[Foggia et al. 2007][Eppstein 1999][Johnson 2005], é fundamental propor uma metodologia de comparação eficiente e uma métrica de medição do nível de similaridade presente nos grafos empregados no processo de identificação.

Este trabalho propõe uma metodologia para identificação de códigos metamórficos baseada na geração de grafos de dependência de referência, gerados a partir da identificação das partes mais relevantes de grafos de dependência extraídos com base na análise estática de códigos executáveis. O processo de identificação emprega uma metodologia de medição que mensura o máximo isomorfismo de subgrafo presente entre as estruturas comparadas, em conjunto com um algoritmo genético modificado que leva em consideração características topológicas dos grafos gerados a partir dos programas executáveis. Avaliações realizadas, tanto com coleções grafos sintéticos como com grafos gerados partir de versões metamórficas de *malware* reais, demonstram melhorias no processo de identificação de códigos maliciosos metamórficos.

O restante deste artigo está organizado da seguinte forma. A seção 2 fornece alguns trabalhos relacionados ao problema de detecção de *malware* metamórfico baseados em grafos de dependência. A seção 3 apresenta uma metodologia para identificação de *malware* metamórfico baseada em grafos de dependência. A seção 4 detalha o processo de diferenciação de tipos de vértices que dá suporte a todos os

aprimoramentos propostos neste artigo. A seção 5 detalha o processo de comparação de grafos de dependência proposto neste artigo. A seção 6 fornece resultados experimentais. A seção 7 apresenta as conclusões e trabalhos futuros.

2. Conceitos Básicos e Trabalhos Relacionados

Grafos de dependência podem ser construídos de forma a modelar as inter-relações entre diversos componentes. Códigos executáveis podem ser analisados de forma a modelar as relações de dependência existentes entre todas as chamadas da função [Hu et al. 2009]. Cada vértice v_i está associado a uma função e uma aresta $v_a v_b$ é criada sempre que no corpo da função v_a existir uma chamada para a função v_b. A partir deste ponto, o processo de identificação compara o grafo gerado com uma base de referência, permitindo a identificação do *malware*. A maior limitação deste processo está associada à dificuldade de mapear corretamente as funções criadas diretamente no código do programa.

Para detecção de códigos maliciosos presentes em *scripts* [Kim and Moon 2010], o código suspeito é analisado para a extração de um grafo de dependência baseado nas variáveis que cada instrução manipula. O grafo gerado passa ainda por um processo de redução que elimina elementos gerados a partir da ação do metamorfismo do código. O processo de detecção procura de encontrar o máximo isomorfismo de subgrafo entre o grafo reduzido e um conjunto de grafos de referência extraídos de malware previamente identificados. Entretanto, a definição de níveis de similaridade usados para definir uma identificação positiva não é muito clara.

Outra proposta [Elhadi et al. 2014], trata o problema de identificação por meio do uso de grafos que modelam chamadas bibliotecas de APIs. Este modelo se concentra em representar as relações de dependência que existem entre os diversos módulos (procedimentos e funções) que são utilizados pelo programa modelado, sejam chamadas a bibliotecas externas ou mesmo chamadas a funções do sistema operacional. Esta proposta procura invalidar totalmente o metamorfismo de código, se concentrando em construir o seu modelo de identificação baseado na forma como os diversos módulos do programa interagem entre si e entre as funções externas utilizadas. Assim, quaisquer programas passam a ser comparados através da medição do nível de similaridade entre seus grafos de chamada. As maiores limitações deste método estão associadas ao nível de precisão do processo de extração do grafo de chamadas e ao custo computacional na execução do processo de comparação dos grafos.

De forma geral, todas essas abordagens não fazem qualquer tipo de distinção entre a natureza de cada vértice, durante a execução deste processo de comparação de grafos. Por exemplo, o algoritmo de [Kim and Moon 2010] propõe o uso de um algoritmo genético que consome parte de seu tempo comparando vértices de processamento com vértices de decisão, desperdiçando tempo de processamento desnecessariamente. A proposta apresentada neste artigo pretende eliminar este desperdício, pelo uso ativo da classificação de vértices para que elementos de natureza distinta não sejam comparados durante o processo de identificação, sem que para isto seja necessário alterar o modelo de codificação utilizado para armazenar os grafos na base de referência.

3. Metodologia de Identificação de Códigos Metamórficos Executáveis

Em uma parte anterior da pesquisa que originou este trabalho [Martins et al. 2014] propôs um processo de identificação de códigos executáveis metamórficos, baseada na utilização de grafos de dependência. Este processo pode ser descrito resumidamente como sendo composto por quatro etapas principais:

i. Reconstrução de código *assembly*, onde o programa executável passa por um processo de engenharia reversa para a obtenção do seu código equivalente em linguagem *assembly*. Esta etapa pode ser executada com o auxílio de programas como OllyDbg [1] e IDA Pro [2].

ii. Geração do grafo de dependência, onde o programa gerado na etapa anterior é analisado e então usado como base para a geração do grafo de dependência.

iii. Redução do grafo, usado para reduzir o grafo de dependência obtido na etapa anterior. Partes do código onde o controle de fluxo nunca irá passar são removidas. De acordo com nossa proposta, um tratamento adicional de redução deve ser executado para os grafos que constituírem a base de referência.

iv. Comparação do grafo reduzido com a base de referência, onde o grafo reduzido é comparado com um grafo correspondente a um *malware* previamente analisado.

É importante destacar que o processo de geração dos grafos que constituem a base de referência também segue as três primeiras etapas do processo de identificação, com a diferença de que a etapa de redução do grafo passa por um processo aprimorado de redução, desenvolvido para identificar os elementos estruturais mais relevantes do grafo, com base na diferenciação dos tipos de vértices presentes e suas inter-relações.

Este trabalho apresenta um aprimoramento dessa abordagem, introduzindo o conceito de diferenciação de vértices e de ordenação topológica, no processo de comparação de grafos. Além disso, serão apresentados resultados comparativos com ferramentas comerciais de detecção de *malware*. Para um melhor entendimento destas contribuições, o conceito de diferenciação de vértices é explicado com detalhes na Seção 4, enquanto a Seção 5 demonstra como os conceitos de ordenação topológica foram introduzidos no processo de comparação entre grafos de dependência.

4. Diferenciação dos Vértices em Grafos de Dependência

Este trabalho propõe uma diferenciação dos vértices de um grafo de dependência com base nas análises estruturais dos vértices pertencentes a grafos de dependência extraídos de programas executáveis e da correlação destes vértices com as instruções presentes nos códigos *assembly*, usados para gerar estes grafos.

[1] http://www.ollydbg.de.

[2] http://www.hex-rays.com/products/ida/index.shtml.

Portanto, a partir dessa análise, estes vértices podem ser classificados em três categorias distintas, conforme: a) vértices de partida; b) vértices de processamento; e c) vértices de decisão. Estes vértices são ilustrados na Figura 1.

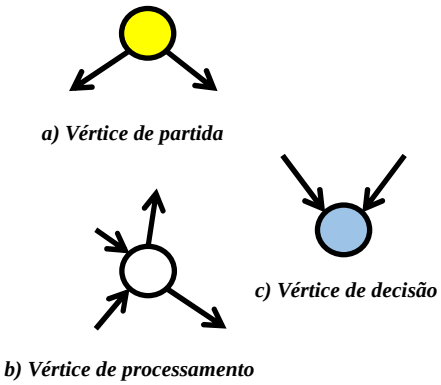


Figura 1 - Tipos de vértice encontrados em grafos de dependência.

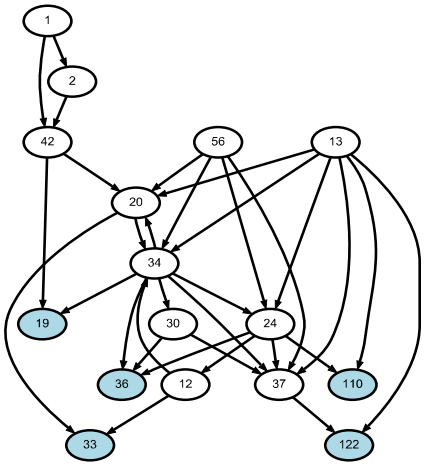


Figura 2 - Grafo de dependência reduzido, com os vértices de decisão em destaque.

Os **Vértices de Partida** dizem respeito ao início da cadeia de dependências associadas à manipulação de uma variável ou registrador. Como estas instruções correspondem no código à primeira manipulação na cadeia de dependências, os vértices derivados não possuem arestas incidentes, assim como uma raiz em uma árvore de busca. Entretanto, diferente de árvores de busca tradicionais, os grafos de dependência possuem mais de um nó com este tipo de características. Este tipo de nó também serve como a indicação de pontos onde o programa original poderia iniciar o seu processamento. Na Figura 2, os nós 1, 13 e 56 são exemplos deste tipo de vértice.

Os **Vértices de Processamento** estão associados a qualquer instrução que manipulam e alteram o conteúdo dos registradores e variáveis presentes no código. A característica estrutural que identifica este tipo de vértice é a presença tanto de arestas incidentes como arestas partindo destes nós. A maioria dos vértices presentes nos grafos de dependência, gerados neste trabalho, fazem parte desta categoria. Na Figura 2, alguns exemplos deste tipo de vértice incluem os nós 2, 20, 34 e 42.

Finalmente, os **Vértices de Decisão** são aqueles gerados a partir das instruções `CMP`, utilizadas para avaliar e comparar o conteúdo de registradores e variáveis. Estas instruções podem ser encontradas antes de qualquer instrução de desvio condicional é o seu resultado que realmente determina se o desvio será ou não executado. Exemplos deste tipo de vértice incluem os nós 19, 33, 36, 110 e 122 na Figura 2.

Como programas escritos em *assembly* não possuem instruções de controle de alto nível como `if-then-else` e `do-while`, é através de instruções `CMP` que a maior parte da lógica de funcionamento do programa é construída. Como estas instruções não alteram o conteúdo dos registradores e variáveis avaliadas, os vértices gerados a partir delas têm a característica distinta de não possuírem arestas de saída, como as folhas em uma árvore binária de pesquisa.

O processo usado para identificar os vértices de decisão mais relevantes para o programa modelado pode ser resumido em 4 etapas: 1) cálculo da menor distância relativa entre cada vértice de decisão; 2) construção de um grafo virtual derivado, constituído apenas dos vértices de decisão, com arestas representando a distância relativa entre cada vértice de decisão; 3) cálculo da clique máxima [Bomze et al. 1999] presente neste grafo virtual derivado; e 4) redução final do grafo de dependência, com a eliminação de qualquer vértice e aresta que não estejam associados aos vértices de decisão presentes na clique máxima do grafo virtual derivado. Detalhes de cada uma dessas etapas podem ser encontrados em um trabalho anterior dos autores [Martins et al. 2014].

Assim, o grafo gerado por este processo de serve como referência para identificação de versões metamórficas do mesmo código que o originou, através de um processo de comparação entre grafos.

5. Comparação dos Grafos de Dependência

Os algoritmos de comparação de grafos visam encontrar uma solução através da construção iterativa de associações estabelecidas entre os vértices de dois grafos, G_1 e G_2, que satisfaça um conjunto de restrições do problema em análise. Neste trabalho, o algoritmo de comparação de grafos tem como objetivo gerar uma solução viável para o problema de isomorfismo de grafos.

A diferenciação entre os tipos de vértices, presentes nos grafos de dependência, também é empregada para aprimorar o processo de comparação e identificação do nível de similaridade entre estes grafos. Nesta visão, o conjunto de vértices V em cada grafo tratado é constituído por dois grupos distintos: a) V_d , o subconjunto de V formado apenas por vértices de decisão e b) V_p , o subconjunto de V formado apenas pelos vértices de processamento. Esta diferenciação contribui para a precisão dos resultados porque o processo de comparação usa esta informação para evitar que vértices de natureza distinta sejam comparados entre si. O conjunto total de propriedades que suportam esta metodologia é apresentado na Tabela 1.

Tabela 1 - Equações para realizar o cálculo da similaridade entre grafos.

Descrição	**Equação**
Definições iniciais	$G_1=(V_1, E_1)$, $G_2=(V_2, E_2)$ e $\|V_1\| < \|V_2\|$
Detalhamento dos vértices	$V=V_d, \cup V_p$
Subgrafo G'	$G'=(V', E') \mid G' \subset G, V' \subset V, E' \subset E$
Subgrafos comparados	$G_1' \subset G_1 , G_2' \subset G_2 \mid \|V_{d1}'\| = \|V_{d2}'\|, \|V_{p1}'\| = \|V_{p2}'\|$
Função de busca de uma aresta e em um conjunto de arestas E	$I(e,E) = \begin{cases} 1, & se \quad e \in E \\ 0, & caso \quad contrário \end{cases}$
Cálculo da similaridade entre G_1 e G_2	$similaridade(G_1', G_2') = \frac{\sum_{e \in E_1'} I(e, E_2') + \sum_{e \in E_2'} I(e, E_1')}{\lvert E_1' \rvert + \lvert E_2' \rvert}$

É importante destacar que, neste processo de comparação, nem todos os vértices (sejam de decisão ou processamento) contribuem para pontuação de similaridade, já que

a quantidade de vértices de decisão e de processamento comparados deve ser igual, conforme as propriedades apresentadas na Tabela 1. Por exemplo, caso G_1 tenha 10 vértices de decisão, 25 de processamento e G_2 tenha 9 vértices de decisão e 50 vértices de processamento, os subgrafos comparados (G_1' e G_2') terão apenas 9 vértices de decisão e 25 vértices de processamento.

A função de cálculo da similaridade entre os grafos gera uma pontuação de similaridade. Esta pontuação é usada para determinar se existe ou não contaminação por *malware* no programa analisado.

O processo base de comparação das amostras utiliza um algoritmo genético, que deixa o grafo G_1 inalterado e gera diferentes arranjos de vértices do grafo $G2$ que, a partir daí, são comparadas com o grafo G_1. Cada um desses arranjos representa um membro de uma população de 100 elementos. Esta população inicial é renovada à medida que o processo de comparação avança, onde k novos filhos (definidos pela taxa de substituição) substituem os k arranjos com menor pontuação existentes na população. Este processo se repete até que seja atingido um limite máximo de gerações definidos pelos parâmetros de controle do algoritmo genético.

5.1. Introduzindo a Diferenciação de Vértices na Comparação de Grafos de Dependência.

Uma das metas deste trabalho é introduzir o processo de diferenciação de vértices no momento da comparação entre grafos, sem que para isto seja necessário alterar o modelo de codificação utilizado para armazenar os grafos na base de referência.

Isto é possível mesmo que as informações de classificação não sejam salvas na estrutura de armazenamento dos grafos de dependência, uma vez que o custo de obtenção desta informação é absorvido pelo processo de leitura destas estruturas no momento da recuperação das informações necessárias para criar as arestas pertencentes ao grafo de dependência original. Afinal, todo e qualquer vértice posicionado como origem de uma aresta será automaticamente reconhecido como vértice de processamento. Todo e qualquer vértice que não apresentar esta característica, será então classificado como vértice de decisão. Este processo classificará os vértices de partida como vértices de processamento, mas isto é esperado, pois vértices de partida podem se transformar em vértices de processamento, e vice-versa, em função da alteração na ordem que as instruções são inseridas no código, como ação do metamorfismo.

Assim, para tirar proveito da classificação dos vértices, este trabalho propõe e avalia a utilização de um processo de *comparação segmentada com restrição de escopo*.

5.2 Segmentação por Tipo de Vértice.

Antes de iniciar o processo de comparação, é necessário que os vértices pertencentes aos grafos que serão comparados sejam separados em dois subconjuntos distintos.

No primeiro conjunto, ficam apenas os nós de decisão e no segundo conjunto, os nós de processamento e nós de partida. Os nós de partida não receberão tratamento diferenciado, pois sua quantidade é muito menor e, dada a possibilidade de reposicionamento de instruções, uma mesma instrução pode tanto gerar um nó de

partida como um nó de processamento. A Figura 3 ilustra a diferença entre a disposição de cromossomos sem o uso da segmentação (Figura 3.a) e com o uso da segmentação (Figura 3.b), para o grafo representado na Figura 2.

a) Disposição de vértices no cromossomo, sem o processo de segmentação

1	2	12	13	19	20	24	30	33	34	36	37	42	56	110	122

b) Disposição de vértices no cromossomo, com o processo de segmentação

19	33	36	110	122	1	2	12	13	20	24	30	34	37	42	56

Figura 3 – Comparação entre a disposição de vértices em cromossomos sem segmentação e com o uso de segmentação.

5.3 Comparação Segmentada com Restrição de Escopo.

Na comparação segmentada com restrição de escopo (ou comparação por escopo), empregamos as mesmas estratégias genéticas utilizadas na abordagem de Kim e Moon [Kim and Moon 2010]. O diferencial aqui é que são executados dois processos de comparação em paralelo, onde os conjuntos gerados na etapa anterior são comparados de acordo com o seu tipo. Assim, apenas vértices que possuam natureza similar serão comparados entre si, otimizando o processo de comparação como um todo.

Entretanto, os vértices de processamento comparados são apenas aqueles que podem ser atingidos a partir dos vértices de decisão selecionados para o primeiro grupo de comparações. Estes resultados são obtidos através da manipulação da população inicial que é repassada ao algoritmo genético para comparação. No momento da construção da população inicial, os vértices de processamento serão dispostos de acordo com a ligação que possuírem com os vértices de decisão, de forma a que fiquem organizados de acordo com a distância que se encontraram destes vértices.

Este processo de construção da população inicial é dividido em 4 etapas:

1) Os vértices de decisão são inseridos nas primeiras posições do cromossomo. No primeiro membro da população, estes vértices são posicionados em ordem crescente, baseado em seu rótulo, nos demais elementos estes vértices de decisão são dispostos aleatoriamente;

2) A lista de vértices de processamento ligados diretamente a cada um dos vértices de decisão é recuperada. Estes vértices são os primeiros vértices de processamento a serem inseridos no cromossomo. A ordem na qual estas listas são recuperadas obedece à mesma disposição dos vértices de decisão contidos no cromossomo que está sendo construído. Entretanto, mesmo que um vértice pertença a mais de uma lista, ele será inserido apenas na sua primeira ocorrência. Este processo continua até que todos os vértices de processamento ligados diretamente a vértices de decisão sejam incluídos;

3) Nesta etapa, os primeiros vértices de processamento inseridos têm suas listas de vértices adjacentes consultadas. Assim, para cada um dos vértices de processamento já inseridos é recuperada a lista de vértices atingidos a partir dele. Estes vértices são então inseridos no cromossomo, desde que já não tenham sido inseridos anteriormente. Este processo se repete até que todos os vértices de

processamento presentes no cromossomo sejam avaliados, tenham eles sido inseridos na etapa anterior ou na etapa atual;

4) A última etapa se encarrega de inserir todos os vértices que não tenham sido inseridos nas etapas anteriores. Este pequeno conjunto de vértices diz respeito a dois tipos de vértices: a) vértices de partida, que não podem ser atingidos pelo processo de construção do cromossomo descrito nas etapas anteriores, pois estes vértices só possuem arestas originadas a partir deles, não existindo qualquer caminho no grafo que leve até eles; e b) vértices de processamento que só podem ser atingidos a partir dos vértices de partida.

Todo este processo organiza o os vértices de uma maneira que é relacionada à organização topológica do grafo original o que proporciona melhores resultados no processo de comparação entre os cromossomos. Além disso, como a maioria dos vértices de processamento está próxima dos vértices de decisão do primeiro grupo, isto diminui o número de elementos a serem trabalhados e otimiza o tempo de processamento, diminuindo o número de iterações necessárias para gerar uma pontuação próxima da máxima pontuação de similaridade entre os grafos comparados.

Outra possibilidade proporcionada por este método é a delimitação de uma distância máxima que os vértices de processamento precisam ter em relação aos vértices de decisão escolhidos para comparação, limitando ainda mais o número de elementos que serão manejados durante o processo de levantamento do nível de similaridade entre grafos. A Figura 4 ilustra a diferença de disposição entre o processo de segmentação (Figura 4.a) e o processo de restrição de escopo (Figura 4.b), para o grafo da Figura 2.

a) Disposição de vértices no cromossomo, com o processo de segmentação

19	33	36	110	122	1	2	12	13	20	24	30	34	37	42	56

b) Disposição de vértices no cromossomo, com o processo restrição de escopo

19	33	36	110	122	34	42	12	20	24	30	13	37	1	2	56

Figura 4 – Comparação entre a disposição de vértices em cromossomos com segmentação e com restrição de escopo.

6. Resultados

Inicialmente, para avaliar o desempenho da técnica de comparação por tipo e comparação por restrição de escopo, foi empregada uma base de dados sintética com o os valores de MCS (*Maximum Common Subgraph* ou maior subgrafo comum) conhecidos. Esta base é formada por um conjunto de pares de grafos com uma quantidade mínima de vértices e arestas iguais previamente estabelecidas[Foggia et al. 2007]. O objetivo é avaliar o impacto que a introdução comparação com restrição de escopo é capaz de produzir na execução do algoritmo genético de comparação.

A partir dos resultados obtidos, é possível perceber que os resultados da comparação sem o uso da restrição de escopo (Figura 5.a) possuem um comportamento similar e estável na maioria dos experimentos. Entretanto, as pontuações obtidas foram, na maioria das vezes, bem superiores aos resultados esperados. Este efeito foi principalmente observado nas faixas onde o MCS possuía o menor valor. Apenas as

faixas de valor de MCS superiores a 70% atingiram pontuações mais condizentes com as características apresentadas pelos pares de grafos analisados.

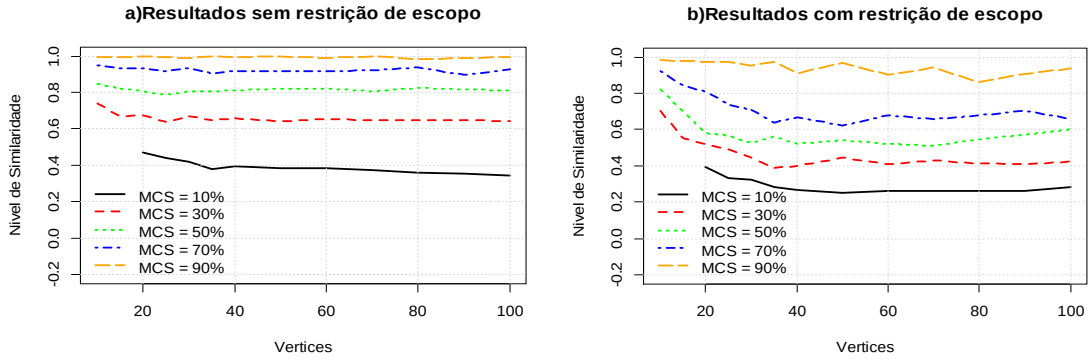


Figura 5 - Gráfico de dispersão relacionando as médias das diferenças com os valores do MCS e da quantidade de vértices.

Assim, apesar das faixas de MCS estarem todas separadas por um fator linear (cada faixa foi gerada em saltos de 20%), as faixas de pontuação obtidas tiveram um comportamento não-linear, quando relacionadas umas às outras

Nos resultados obtidos com a utilização da comparação com restrição de escopo (Figura 5.b), os níveis de pontuação obtidos pelas amostras com menor quantidade de vértices, foram similares àqueles obtidos na primeira avaliação. Entretanto, à medida que a quantidade de vértices aumentou, o nível de pontuação gerado foi se comportando cada vez mais de acordo com aquele compatível com as características das amostras analisadas. Assim, para as amostras com uma quantidade de vértices maior, as faixas de pontuação geradas tiveram um comportamento mais linear quando comparadas umas com as outras.

Quando os testes com as amostras metamórficas foram realizados, as pontuações obtidas em cada grupo geraram resultados de pontuação bem distintos. Ao serem comparadas com o modelo de referência gerado a partir da base sintética, em três dos cinco grupos as pontuações obtidas foram muito abaixo de 20% de similaridade. Este mesmo comportamento tinha sido previamente observado em uma etapa anterior deste projeto de pesquisa [Martins et al. 2014]. Apesar disso, cada grupo comparado apresentou níveis de pontuação muito similares quando analisados isoladamente.

Assim, para se obter uma visão mais clara do nível de pontuação que seria adequado para cada amostra, foi executado um experimento onde o grafo de referência foi comparado com sua versão anterior ao processo de redução final, garantindo que o grafo de referência seria um subgrafo do grafo comparado. Cada uma das amostras foi submetida a este processo em ciclos de 100 repetições. Os resultados deste experimento são apresentados na Figura 6.

Estes resultados demonstraram que cada grupo de amostras possuía um nível de pontuação diferente, relacionado diretamente ao grafo e ao *malware* usado como referência de comparação. Assim, as pontuações obtidas neste experimento definiram as bases de pontuação que cada grupo de amostras metamórficas deveria atingir para ser considerada como positiva a identificação de um programa analisado. Além disso, foi

também definido que todas as amostras metamórficas seriam submetidas a ciclos de dez comparações, com a pontuação de cada amostra sendo definida pela média desses resultados.

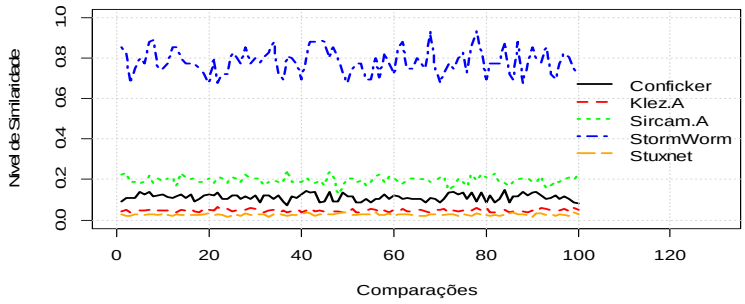


Figura 6 – Resultados da comparação dos grafos de referência com suas versões não reduzidas.

Finalmente, para que a diferença na quantidade de vértices também fosse levada em consideração no momento da geração da pontuação final, foi aplicado um coeficiente de ajuste da pontuação, gerado a partir da quantidade de vértices de cada grafo sendo comparado, onde o menor valor é dividido pelo maior, gerando um redutor de pontuação. Este redutor é aplicado à pontuação original através da multiplicação destes dois valores, obtendo-se assim o valor final da pontuação de similaridade de cada teste. Como limite mínimo de detecção, ou o limiar de identificação, foi definida uma pontuação 10% inferior ao nível de pontuação de similaridade de referência. A taxa de identificação neste experimento foi de 98% das amostras. Os resultados obtidos em todos os testes de identificação são apresentados de forma resumida na Tabela 2.

Além destes testes, também foram avaliadas as ocorrências de identificações de falsos positivos. Os resultados demonstram que a pontuação obtida foi muito abaixo do nível mínimo estipulado, não ocorrendo nenhuma identificação errônea. Um resumo de todos os testes de falso positivos realizados é apresentado na Tabela 3.

Testes de falso positivo foram realizados usando como referência o grafo do *malware* Sircan.A comparado com grafos de dependência extraídos de 28 programas idôneos e livres de qualquer contaminação. Novamente, cada amostra foi comparada diversas vezes com o grafo de referência e tomada como pontuação final a média dos resultados. A taxa de falsos positivos gerada foi de 7%. Infelizmente, a eficiência do algoritmo de comparação foi afetada pela variação entre a quantidade de vértices das amostras sem contaminação e a quantidade de vértices do grafo do *malware*.

Finalmente, algumas das amostras metamórficas foram submetidas à ferramenta Vírus Total[3], que avalia cada arquivo a mais de cinquenta ferramentas de antivírus comerciais. Apesar das ferramentas comerciais implementarem diferentes estratégias para tratar o problema do *malware* metamórfico, elas ainda estão longe de serem soluções efetivas. Em média 65,3% dos testes obtiveram uma identificação positiva da presença de código malicioso. Entretanto, apenas 18,2% das amostras metamórficas foram identificadas corretamente, quanto ao tipo de *malware* a partir do qual se originaram, indicando que muito esforço ainda é necessário para que uma proteção

[3] https://www.virustotal.com/pt/

dessas ferramentas seja completa. A Tabela 4 apresenta alguns dos resultados obtidos onde é indicada a quantidade de testes e de identificações positivas obtidas.

Tabela 2 – Resumo dos resultados de testes de identificação de versões metamórficas de *malware*.

***Malware* Base**	**Vértices no Grafo de Referência**	**Média de Vértices nas Amostras Metamórficas**	**Taxa de Identificações com Sucesso**
Conficker	42	60,51	98%
Klez.A	120	99,97	100%
Sircam.A	28	33	99%
StormWorm	23	24	100%
Stuxnet	184	332,93	97%

Tabela 3 – Resumo dos resultados de testes de falsos positivos.

***Malware* Base**	**Amostras**	**Referência**	**Média de Vértices nas Amostras**	**Taxa de identificações Errôneas**
Conficker	Klez.A	42	99,97	0%
Conficker	Stuxnet	42	332,93	0%
Sircam.A	Klez.A	28	99,97	0%
Sircam.A	Stuxnet	28	332,93	0%
StormWorm	Conficker	23	60,51	0%
StormWorm	Klez.A	23	99,97	0%
StormWorm	Sircam.A	23	33	0%
StormWorm	Stuxnet	23	332,93	0%

Tabela 4 – Resumo dos testes com a ferramenta Vírus Total.

Malware									
Conficker		**Klez.A**		**Sircam.A**		**StormWorm**		**Stuxnet**	
Identificações	**Testes**	**Identificações**	**Testes**	**Identificações**	**Testes**	**Identificações**	**Testes**	**Identificações**	**Testes**
37	56	33	55	37	56	38	55	40	56
36	55	31	55	37	56	40	55	41	56
34	55	28	53	35	56	38	56	38	55
37	55	27	55	36	55	40	56	37	55

7. Conclusões

A diferenciação entre os tipos de vértices presentes nos grafos de dependência tratados nesta pesquisa ofereceu contribuições tanto para o processo de redução aprimorado dos grafos de referência, como para o processo de comparação e identificação do nível de similaridade entre estes grafos.

A introdução do processo de diferenciação de vértices no processo de redução permitiu estabelecer uma base clara para os níveis de pontuação necessários para estabelecer uma identificação positiva. Além disso, quando a diferenciação de vértices foi combinada com características de ordenação topológica, o processo de comparação também foi aprimorado, com a geração de resultados mais fiéis àqueles observados nos testes com uma base de grafos sintéticos. Finalmente, quando comparados com as ferramentas comerciais o processo de identificação obteve resultados bem superiores, atestando o potencial da metodologia proposta.

Como trabalhos futuros, o emprego de metodologias alternativas para a comparação dos grafos de dependência merece ser investigada. Além disso, o algoritmo de comparação atual pode ser modificado para que seu desempenho na comparação entre grafos com diferentes quantidades de vértices seja mais próximo àquele apresentado pela comparação entre grafos com quantidades de vértices iguais.

Agradecimentos

Este trabalho foi parcialmente financiado pela Fundação de Amparo à Pesquisa do Estado do Amazonas – FAPEAM (processo 062.03178/2012), e pela CAPES.

Referências

Barossa Community Co-operative Store (2014). Pandalabs Annual Report 2014. p. 1–28.

Bomze, I. M., Budinich, M., Pardalos, P. M. and Pelillo, M. (1999). The maximum clique problem. *Handbook of combinatorial optimization*. Springer US. p. 1–74.

Borello, J.-M. and Mé, L. (21 feb 2008). Code obfuscation techniques for metamorphic viruses. *Journal in Computer Virology*, v. 4, n. 3, p. 211–220.

Bruschi, D., Martignoni, L. and Monga, M. (mar 2007). Code Normalization for Self-Mutating Malware. *IEEE Security and Privacy Magazine*, v. 5, n. 2, p. 46–54.

Cozzolino, M. F., Martins, G. B., Souto, E. and Deus, F. E. G. (2012). Detecção de variações de malware metamórfico por meio de normalização de código e identificação de subfluxos. In *Anais do XII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*.

Elhadi, A. A. E., Maarof, M. A., Barry, B. I. A. and Hamza, H. (2014). Enhancing the detection of metamorphic malware using call graphs. *Computers & Security*, v. 46, p. 62–78.

Eppstein, D. (9 nov 1999). Subgraph Isomorphism in Planar Graphs and Related Problems. v. 3, n. 3, p. 27.

Foggia, P., Vento, M. and Elettrica, I. (2007). Challenging Complexity of Maximum Common Subgraph Detection Algorithms : A Performance Analysis of Three Algorithms on a Wide Database of Graphs Donatello Conte. v. 11, n. 1, p. 99–143.

Griffin, K., Schneider, S., Hu, X. and Chiueh, T. C. (2009). Automatic generation of string signatures for malware detection. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 5758 LNCS, p. 101–120.

Hu, X., Chiueh, T. and Shin, K. G. (2009). Large-scale malware indexing using function-call graphs. *Proceedings of the 16th ACM conference on Computer and communications security - CCS '09*, p. 611.

Jacob, G., Debar, H. and Filiol, E. (21 feb 2008). Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology*, v. 4, n. 3, p. 251–266.

Jacob, G., Debar, H. and Filiol, E. (2 feb 2009). Malware detection using attribute-automata to parse abstract behavioral descriptions. *arXiv preprint arXiv:0902.0322*, n. 1.

Johnson, D. S. (1 jul 2005). The NP-completeness column. *ACM Transactions on Algorithms*, v. 1, n. 1, p. 160–176.

Kim, K. and Moon, B.-R. (2010). Malware detection based on dependency graph using hybrid genetic algorithm. *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*, p. 1211.

Martins, G., Souto, E., Freitas, R. De and Feitosa, E. (2014). Estruturas Virtuais e Diferenciação de Vértices em Grafos de Dependência para Detecção de Malware Metamórfico. *lbd.dcc.ufmg.br*, p. 237–250.

Newsome, J., Karp, B. and Song, D. (2005). Polygraph: Automatically Generating Signatures for Polymorphic Worms. *2005 IEEE Symposium on Security and Privacy (S&P'05)*, p. 226–241.

O'Kane, P., Sezer, S. and McLaughlin, K. (2011). Obfuscation: The Hidden Malware. *IEEE Security & Privacy*, v. 9, n. 5, p. 41–47.

Rad, B. B., Masrom, M. and Ibrahim, S. (2012). Camouflage in Malware : from Encryption to Metamorphism. v. 12, n. 8, p. 74–83.

You, I. and Yim, K. (2010). Malware Obfuscation Techniques : A Brief Survey. *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, p. 297–300.

Detecção de Vazamentos de Dados em Aplicativos Javascript em Dispositivos Móveis

Thiago de Souza Rocha[1], Eduardo Souto[1], Diego Azulay[1], Brandell Cássio[1], Alex Monteiro[1], Pedro Minatel[2], Breno Silva[2], Felipe Boeira[2]
[1]Instituto de Computação Universidade Federal do Amazonas (UFAM) Manaus, AM – Brasil
[2] Instituto de Pesquisa da Samsung, SP – Brasil

{thiago.rocha,esouto,dsa,bccf,alex.monteiro}@icomp.ufam.edu.br

{p.minatel,breno.p,f.boeira}@samsung.com

Abstract. *The development of applications with HTML5 and JavaScript that can be executed in multiple mobile devices like smartphones, tablets and others provided the appearance of multi-platform operating systems and increased the use of these languages. However most of these devices store sensitive information about the users and have become potential targets of attacks. One of the biggest concerns from companies that develop applications to these devices is the leakage or exposure of sensitive data. In this work we are addressing this problem by modifying the Tizen Web Runtime to add dynamic taint tracking, with that we can track sensitive information that is being leaked, even if the information is obfuscated, and warn the users. From our knowledge this is the first prototype that adds this kind of technique to Tizen and is the first prototype that tracks web applications in mobile devices. The results show that Dynamic Taint Tracking is a promising approach that can be improved and used to detect data leakage in mobile devices.*

Resumo. O desenvolvimento, utilizando as linguagens JavaScript e HTML5, de aplicativos que podem ser executados em diversos dispositivos móveis como celulares, tablets, entre outros proporcionou o surgimento de sistemas operacionais multiplataforma e aumentou a utilização dessas linguagens. Entretanto a maioria desses dispositivos armazenam informações confidenciais, também chamadas de informações sensíveis, sobre os usuários e tem se tornado um alvo em potencial de ataques. Uma das grandes preocupações das empresas que desenvolvem aplicações para esses dispositivos é justamente o vazamento dessas informações sensíveis. Este trabalho enfrenta esse problema através de uma modificação na Web Runtime do Tizen para adicionar uma análise dinâmica de informações sensíveis, com isso é possível acompanhar para onde a informação está indo e informar os usuários, mesmo que a informação esteja codificada. Pelo nosso conhecimento esse é o primeiro protótipo que adiciona esse tipo de técnica no Tizen e também é o primeiro protótipo que realiza análise dinâmica de informação em aplicações web em dispositivos móveis. Os resultados mostraram que a

abordagem é promissora e que pode ser usada para detectar vazamento de dados em dispositivos móveis.

1. Introdução

A contínua evolução das tecnologias web tem possibilitado o fácil acesso e interação com uma ampla variedade de dispositivos em várias plataformas, incluindo dispositivos móveis (e.g. celulares, notebooks e sensores de saúde), eletrodomésticos (e.g. smart TVs e Smart Geladeira) e redes de sensores veiculares.

Devido a essa variedade de dispositivos, aplicações móveis têm sido projetadas para serem executadas em várias plataformas e tal fato proporcionou o surgimento de novos sistemas operacionais multiplataformas como o Tizen [Tizen 2015], webOS [WebOS 2015] e Firefox OS [Firefox 2015] e aumentou a utilização de linguagens como HTML5 [HTML 2016] e JavaScript [ECMA-262 2016].

Graças ao HTML5 e aos navegadores web embarcados nos novos sistemas operacionais, as aplicações web desenvolvidas em JavaScript estão disponíveis em qualquer lugar [Bae et al. 2014]. A natureza multiplataforma da linguagem JavaScript possibilita que uma mesma aplicação possa ser executada em diferentes dispositivos. Alguns aplicativos Tizen (desenvolvidos em Javascript), por exemplo, executam em notebooks, smartphones, smart TVs e smartwatchs.

Por outro lado, esses dispositivos, na maioria das vezes, lidam com informações sensíveis do usuário. Portanto, garantir a segurança das aplicações que são executadas nesses dispositivos é uma tarefa prioritária e indispensável. Existem diversas abordagens que são utilizadas para analisar as aplicações e garantir a segurança, como testes manuais, testes automatizados usando alguma biblioteca como Jasmine[Jasmine 2016], análise estática[Kashyap et al. 2014] ou dinâmica de código[Seth et al. 2011], entre outras.

Este artigo descreve uma extensão do sistema operacional Tizen, denominada de TTizen (acrônimo de *Taint Tizen*), que realiza a análise dinâmica do fluxo de informações (do inglês, *dynamic taint analysis*) sensíveis em aplicações JavaScript. A meta primária é detectar quando uma informação sigilosa deixa o dispositivo por meio de aplicações de terceiros, caracterizando vazamento de dados. A abordagem proposta automaticamente marca um dado (*taint*) de uma fonte de informação considerada sensível e transitivamente rastreia esse dado à medida que ele se propaga por meio de variáveis e arquivos do programa. O TTizen registra os dados marcados e uma notificação é gerada para o usuário sempre que um dado marcado é transmitido pela rede.

Para alcançar esse objetivo, o TTizen apresenta uma modificação do motor (*engine*) JavaScript para realizar a análise *taint* dinâmica. Mais especificamente foram realizadas modificações no interpretador bytecode do WebKit [WebKit 2016], motor JavaScript usado no Safari e em outros navegadores livres.

A partir do nosso conhecimento, esse é o primeiro trabalho que aplica análise dinâmica de informação em JavaScript dentro de um ambiente móvel. De acordo com nosso levantamento, todas as pesquisas existentes nessa área realizam seus testes somente em navegadores web como mostrado na Seção 3.

O restante desse trabalho está organizado da seguinte forma. Seção 2 apresenta alguns conceitos relacionados à plataforma Tizen e análise de informação. Seção 3 detalha alguns trabalhos relacionados. Seção 4 apresenta o TTizen. Seção 5 mostra os experimentos realizados. Finalmente, a Seção 6 conclui o artigo e expõe alguns trabalhos futuros.

2. Conceitos Básicos

2.1. Plataforma Tizen

O Tizen é um sistema operacional móvel livre promovido principalmente pela Samsung e Intel para desenvolver uma distribuição Linux para vários tipos de dispositivos como automóveis, celulares, TVs inteligentes, relógios inteligentes, câmeras e geladeiras [Rick 2016]. Para acomodar essa grande variedade de dispositivos, o Tizen disponibiliza um conjunto de APIs (*Application Programming Interfaces*) e bibliotecas que possibilitam o desenvolvimento e execução de aplicações Web (escritas em JavaScript) e aplicações nativas (escritas em linguagens C e C++). Uma vantagem das aplicações desenvolvidas em linguagem JavaScript é a portabilidade entre diferentes plataformas.

A Figura 1 mostra a plataforma do Tizen composta pelos frameworks para aplicações web e nativas e algumas APIs (denominada de Core) no topo de um Kernel Linux. As aplicações web e nativa possuem seus próprios méritos como a fácil portabilidade e alto desempenho, respectivamente. Como citado anteriormente, este trabalho foca na detecção de vazamento de dados em aplicações web.

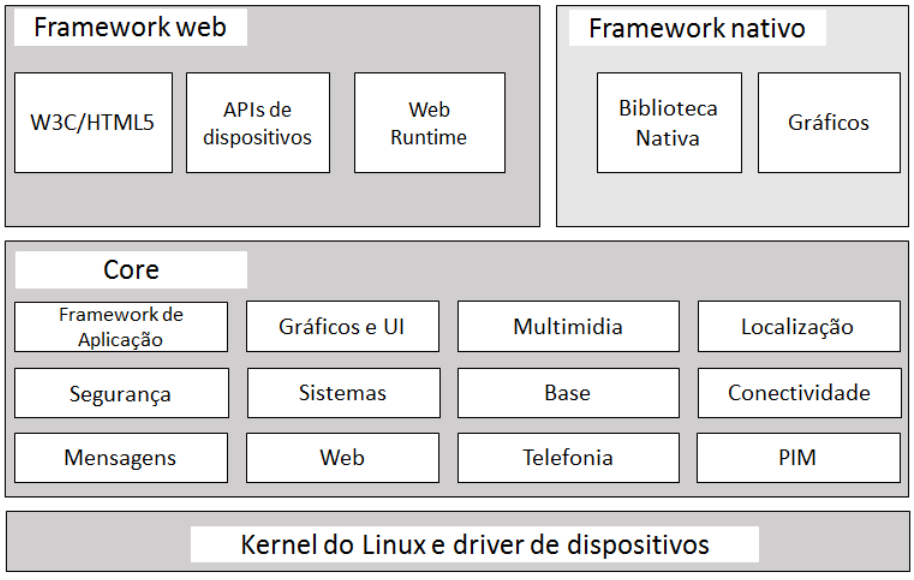


Figura 1. Arquitetura da plataforma Tizen.

2.2. Análise dinâmica do fluxo de Informação

Análise de informação é o processo de acompanhar como os dados marcados se propagam enquanto um programa é executado [Tonin 2016]. Os trabalhos de análise de

informação podem ser descritos por quatro elementos: marcação, fonte, propagação e saída, explicados abaixo:

- Marcação (*taint*): ato de adicionar uma marcação em cada objeto que represente informação sensível de um usuário.
- Fonte (*taint source*): local onde uma aplicação pode obter informações sensíveis sobre um usuário. Por exemplo, a API de localização do Tizen fornece uma interface que permite que uma aplicação obtenha informações sobre GPS.
- Propagação (*taint tracking*): processo de acompanhar (rastrear) um dado marcado enquanto o programa é executado.
- Saída (*taint sink*): interfaces de saída da aplicação, usadas para enviar informações sensíveis para fora da aplicação, por exemplo, uma interface HTTP.

O fluxo da informação que está sendo acompanhado pode ocorrer de duas formas: explícita e implícita [Seth et al. 2011]. No fluxo explícito a informação é passada diretamente entre uma variável marcada que usa o valor e a que recebe. O fluxo implícito é quando o valor de uma variável marcada afeta o valor de outra variável de forma indireta, por exemplo, através de uma estrutura condicional. Inicialmente esse trabalho foca apenas no fluxo explícito.

A análise de informação possui duas categorias, estática e dinâmica. A análise estática realiza a análise do código sem executar o programa, geralmente através da criação de um grafo de fluxo de programa (CFG), porém como JavaScript é uma linguagem dinâmica (permite execução de código através de funções como o *eval*) a análise estática do código se torna difícil [Bichhawat et al. 2014].

Por esse motivo foi escolhida a análise dinâmica que é o processo de executar o rastreio da informação enquanto o programa é executado. O Algoritmo 1 mostra um código fonte que exemplifica a análise dinâmica. Quando um valor de uma fonte de informação sensível (representada no código pelo método *source()*) é atribuído a variável *c*, essa variável recebe uma marcação (*taint*) sinalizando que seu conteúdo é sensível. Posteriormente o valor da variável c é passado para outra variável (*a* no caso do exemplo) e juntamente com o valor, a marcação de c também é passada para a, caracterizando uma propagação. Essa marca (*taint*) é rastreada dinamicamente até que seu conteúdo chegue a uma interface de saída, que no exemplo é o método *send()*.

Algoritmo 1

```
1:  c = source();
2:  a = b+c;
3:  send(a);
4: ...
```

3. Trabalhos Relacionados

Esta seção mostra trabalhos recentes relacionados à análise de informação dinâmica. Através das nossas pesquisas foi notado que todos os trabalhos em dispositivos móveis como [Enck et al. 2014], [Zheng e David 2014], [McClurg et al. 2016], [Zhao e Osorio 2012], [Yang e Yang 2012] e [Egele et al. 2011] realizam a análise das informações apenas nas aplicações nativas. Como o foco desse trabalho é realizar a análise em aplicações web a nossa pesquisa foi direcionada a trabalhos que realizam o rastreio em aplicações JavaScript. Entretanto, todos esses trabalhos fazem testes apenas em navegadores web.

[Tran et al. 2012] criaram uma técnica chamada principal-based tracking onde eles realizam a identificação de bibliotecas que possuem comportamento suspeito. A Uniform Resource Locator (URL) do código JavaScript é definida como principal. Para saber a qual principal cada parte de código JavaScript pertence, é adicionada uma marcação especifica em cada parte de código JavaScript que vai ser compilada em tempo de execução. Por fim, seu sistema monitora o comportamento dos scripts com diferentes principals e identifica os comportamentos suspeitos que acessam informações sensíveis.

Para verificar se algum script suspeito enviou informações sensíveis para servidores de terceiros, os autores desenvolveram um analisador de informação dinâmico em nível de variável apenas nos scripts suspeitos. Uma ferramenta chamada *LeakTracker* foi desenvolvida para testar essa técnica e um *benchmark* chamado SunSpider [SunSpider 2016] foi utilizado no Mozzila Firefox 3.6.13. Cada experimento foi repetido 10 vezes e apresentou um overhead em tempo de execução em torno de 2.2 vezes.

[Jang et al. 2015] propõem um *framework* que insere e propaga marcações enquanto uma aplicação é executada para reforçar a confidencialidade e políticas de integridade. Para realizar esse trabalho, os autores utilizam o código fonte da aplicação e adicionam as suas estruturas de dados e marcações. Os testes foram realizados no navegador Chrome usando benchmarks disponibilizados pela empresa Alexa [Alexa 2016]. Para checar o desempenho foram utilizados 10 benchmarks web onde foram coletados dados de execução do JavaScript e do carregamento das páginas obtendo um overhead em tempo de execução em torno de 2.4 vezes.

[Sayed et al. 2014] apresentam um modelo de controle de fluxo de informação que acompanha dinamicamente todas as informações e previne que uma informação sensível seja enviada para servidores de terceiros. Os autores mostram formalmente a sua abordagem e as modificações que devem ser feitas na semântica Javascript para que a abordagem funcione. Entretanto, nenhuma informação é disponibilizada sobre como a abordagem poderia ser implementada.

[Bichhawat et al. 2014] realizaram uma instrumentação no motor JavaScript para acompanhar as informações dinamicamente. Especificamente, o trabalho propõe a instrumentação do *bytecode* da WebKit e uma modificação do compilador para que ficasse compatível com a instrumentação. A abordagem proposta foi avaliada usando o benchmark SunSpider [SunSpider 2016] no navegador safari e os resultados mostraram um overhead em tempo de execução entre 1.1 e 0.5 vezes.

[Hedin e Sabelfeld 2012] propuseram um controle de fluxo de informação dinâmico para a linguagem que modela as principais características do JavaScript porém foram ignoradas algumas instruções como o *break* e o *continue* e precisam testar a sua abordagem em um navegador real.

A partir dos trabalhos relacionados é possível verificar que toda pesquisa que aplica análise de informação dinâmica em aplicações JavaScript realiza seus testes em um navegador web e possui um overhead considerável. Nossa proposta visa realizar a análise de aplicações JavaScript em dispositivos móveis pois é uma lacuna que existe nesse tipo de pesquisa e também para deixar as aplicações JavaScript nos dispositivos móveis mais seguras.

4. TTizen

4.1. Visão Geral

A abordagem proposta foi concebida para permitir que os usuários possam monitorar como as aplicações embarcadas em dispositivos móveis lidam com os seus dados privados em tempo real. Especificamente, os aplicativos espalhados nos diferentes dispositivos Tizen (e.g. smart TVs, smart geladeiras, smartwatchs, e smartphones) são desenvolvidos em Javascript, que por ser uma linguagem dinâmica torna difícil uma análise estática do código [Bichhawat et al. 2014].

Para tratar esse problema, nós incorporamos ao sistema operacional Tizen um sistema de marcação de dados em tempo real que envolve os seguintes passos básicos:

- Identificar as fontes de informações sensíveis no Tizen (*taint source*) como lista de contatos, e-mail, informações de localização (e.g. dados do GPS), fotos, etc.
- Realizar a marcação das fontes (*taint marking*) de informações sensíveis.
- Definir uma política para rastreamento dos dados marcados (*taint analysis*).
- Identificar as saídas utilizadas pelas aplicações (*taint sink*), usualmente uma interface de rede.

A Figura 2 mostra um diagrama da Web Runtime (WRT) do framework Tizen versão 2.2, onde o sistema de marcação de dados foi implementado. A WRT é responsável por todo ciclo de vida de uma aplicação Tizen desenvolvida em JavaScript, desde a sua instalação até a sua execução dentro da WebKit. Além disso, a WRT fornece um conjunto de APIs que facilitam o desenvolvimento de aplicações provendo informações sobre os dispositivos móveis.

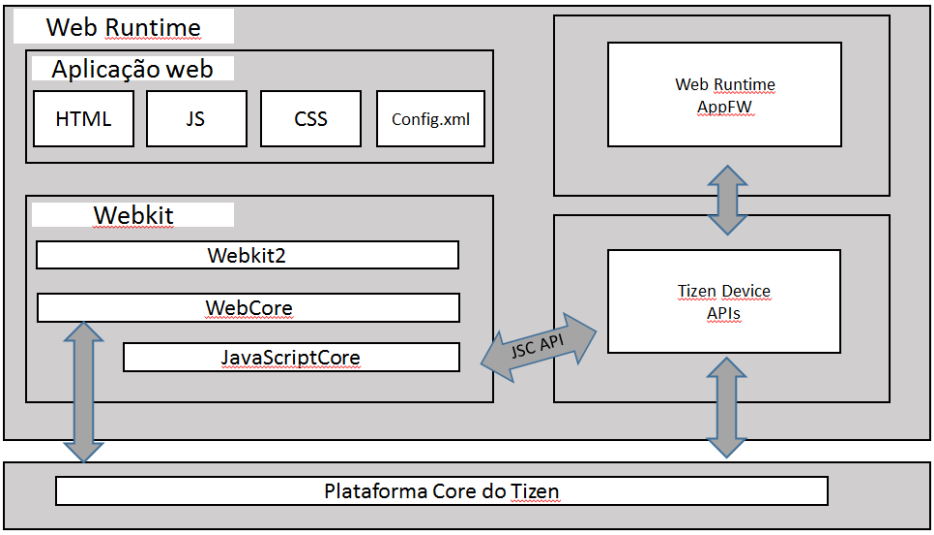


Figura 2. Web Runtime do Tizen.

Primeiramente, foi realizado um estudo em nível de instrução para identificar e marcar as fontes de informações sensíveis. Analisando as APIs de dispositivos móveis do Tizen, nós identificamos as seguintes fontes: código do telefone, lista de contato, número de telefone, informações de rede, endereço de correio eletrônico, fotos e registros de chamadas.

Uma vez que os dados considerados sensíveis foram rotulados (*tainted*), modificações foram realizadas nos componentes da Webkit, especificamente no *engine* do JavaScript (JavaScriptCore), para acompanhar o dado marcado a medida que se propaga durante a execução da aplicação.

A instrumentação do componente de saída foi realizada no componente WebCore da Webkit que é responsável por verificar se uma informação sensível está sendo enviada e por escrever as informações necessárias em um arquivo de log. Atualmente, apenas o envio de informações através de uma conexão HTTP está sendo tratado como saída.

Por fim, para informar o usuário que alguma informação sensível está sendo enviada foi criada uma aplicação Tizen nativa que monitora um arquivo de log. Toda vez que esse arquivo de log é modificado a aplicação lê essas alterações e informa ao usuário qual informação esta sendo vazada e o horário do vazamento.

4.2. Detalhes de Implementação

Esta Seção fornece detalhes das modificações realizadas no componente Web Runtime do Tizen versão 2.2 para realizar a análise de informação dinâmica. Para facilitar a identificação do fluxo, esta seção será dividida em 3 partes : Identificação de fontes, Propagação e Identificação das saídas.

4.2.1 Identificação das fontes

As APIs Tizen se encontram dentro do diretório framework/wrt-plugins-tizen, tendo como diretório raiz todo o projeto do tizen. Entre elas se encontra uma que possui acesso a informações do sistema (*Systeminfo*) e contém alguns métodos para acessar propriedades tanto do sistema quanto do dispositivo, tais como número do IMEI, estado da rede, nível de bateria, etc. Para ter acesso a essas propriedades é preciso fazer uma chamada para o método *getPropertyValue()*.

Para utilizar este método, é necessário seguir a seguinte sintaxe: tizen.systeminfo.getPropertyValue (<Informação>, <Função de Sucesso>, <Função de Falha>), onde:

- <Informação> é a propriedade que se quer do sistema, como o modelo do dispositivo.
- <Função de sucesso> em caso de nenhuma falha na obtenção da informação sensível, essa função é chamada.
- <Função de falha> é chamada caso algo aconteça de errado no na execução do método, sendo opcional.

A implementação do source consiste em marcar o retorno das chamadas que estão na API *Systeminfo,* com a adição de uma *tag* (neste caso, foi utilizada a tag "\1") no final da string que representa a informação sensível.

4.2.2 Propagação

A propagação é feita dentro da JavaScriptCore, núcleo de interpretação e execução do código JavaScript que fica na WebKit. Os arquivos que foram modificados estão na pasta /framework/web/webkit-efl/Source/JavaScriptCore/llint. Nesse diretório há implementações das operações do bytecode gerado pela JavaScriptCore.

Atualmente está sendo tratado apenas o fluxo explícito que pode ser divido em duas categorias: As operações de atribuição, como no caso de operações que passam um valor diretamente, e as que retornam algum valor como no caso de concatenação para strings. Estas operações são geradas pelo bytecode da JavaSriptCore e são compostas por várias operações em nivel de assembly. As categorias serão detalhadas a seguir.

4.2.2.1 Operações de Atribuição

Estas operações atribuem a um espaço de memória, um determinado valor. No código assembly gerado pela JavaScriptCore existem três definições de operações que são responsáveis pela atribuição: mov, put_global_var e put_by_id que serão explicadas a seguir.

4.2.2.1.1 Operação mov

Essa operação é implementada em uma linguagem própria da JavaScriptCore sendo definida como: mov <Reg>, <Valor>. Em que <Reg> é um registrador definido pela linguagem de bytecode gerado pela JavaScriptCore, e <Valor> é o valor que será atribuido para o mesmo. Esta operação é realizada quando se atribui um valor a uma variável dentro de uma função (definida ou anônima), como mostra o Algoritmo 2.

Algoritmo 2

```
1:  function def() {
2:     var a  = 5;
3:     .........
4:  }
```

A operação "var a = 5;", transcrita para bytecode ficaria como "mov r1, @k1", em que @k1 representa uma constante dentro do contexto da função. Como no contexto dessa função existe apenas uma constante, então @k1 representa 5.

4.2.2.1.2 Operação put_global_var

Essa operação também é implementada em uma linguagem própria da JavaScriptCore e atribui valores para uma variável quando esta se encontra num escopo global, fora de uma função. Ela é definida da forma put_global_var <Reg>, <Valor>. Seguindo o exemplo do Algoritmo 3 a operação da linha 1 "var a = 5;" seria transcrita para "put_global_var r1, @k1".:

Algoritmo 3

```
1:  var a = 5;
2:  function def() {
3:     .........
4:  }
```

4.2.2.1.3 Operação put_by_id

A operação put_by_id é implementada na linguagem C e é utilizada quando não é definida uma variável com a palavra reservada "var", como no caso de "a = 5". Essa operação é mais lenta pelo fato de que a JavaScriptCore necessita mapear explicitamente essa variável.

4.2.2.2 Operações de Retorno

As operações de retorno são as operações aritiméticas no caso de inteiros e para strings são operações como concatenação, tamanho da string, entre outras. O algoritmo 4 mostra um exemplo de adição.

Algoritmo 4

```
1:  var a = 5;
2:  var b =6;
3:  var c = a + b;
```

A operação da linha 3 "var c = a + b", é transformada em bytecode para "add r1, r2,r3" em que r2 e r3 representam respectivamente o valor de a e de b, e r1 representa o valor onde deve ser guardado o resultado da soma, que será reservado para a variável c.

Para verificar a propagação foi feita uma checagem em todas essas operações para avaliar se algum dos registradores envolvidos está marcado. Caso esteja marcado, a marcação é adicionada no resultado da operação. Um pseudocódigo é apresentado no Algoritmo 5.

Algoritmo 5

```
1:  propagacao (Objeto v_1, v_2, ...., v_n)
2:  Inicio
3:     resultado = operacao (v_1, v_2, ...., v_n);
4:        Se estaMarcado( v_1, v_2, ...., v_n) então
5:           resultado.adicionaMarca()
6:        Fim se
7:     retorna resultado;
8:  Fim
```

4.2.3 Identificação de saídas

Para verificar a saída foi modificada a classe WebSocket que se encontra no WebCore, nela é verificado quando uma conexão HTTP é criada. Foi adicionado um teste no método *send*() dessa classe para verificar se a informação que está saindo encontra-se marcada (*tainted*). Caso isso aconteça, um vazamento de dados é detectado e o método de log é chamado para registrar a informação sensível e data e hora (*timestamp*) que esse evento ocorreu.

5. Experimentos e Resultados

Essa seção apresenta os resultados obtidos através da avaliação de desempenho da abordagem. Para realizar os experimentos foram utilizadas dois dispositivos ODROID U3 [ODROID 2016] da empresa Hardkernel. Estes dispositivos oferecem suporte a distribuições Linux como Xubuntu 13.10 e Android 4.X, além de oferecer suporte as distribuições Tizen 2.2 e 2.3. O hardware é composto por um processador Arm Quad-Core 1.7GHz,om 2GB de RAM, com suporte a cartão de memória de 64 GB, memória eMMC e saídas de audio e de comunicação serial. Tais configurações se assemelham a um Smartphone SAMSUNG Galaxy S3.

Para validar e comparar as alterações foram realizados experimentos quantitativos e qualitativos. Nos experimentos quantitativos foram medidos a porcentagem de uso de CPU e de memória dos processos relacionados à execução de uma aplicação de teste. No Tizen cada aplicativo web possui dois processos. O primeiro

é responsável pelo ciclo de vida da aplicação e o segundo é responsável por gerenciar e renderizar o conteúdo da aplicação, denominado de *Web Process* [Jaygarl et al. 2016].

No experimento qualitativo foi verificado se a abordagem conseguia identificar um vazamento de dados primeiramente com a informação sendo passada diretamente e posteriormente com a informação sendo passada de forma codificada. Devido à utilização de um dispositivo limitado para embarcar o Tizen (ODROID U3), que não possui propriedades como IMEI e SIM Card, foi utilizada como informação sensível o tipo de rede em que o aparelho está conectado (3G, 4G, WiFi ou NONE). No caso do ODROID U3 o Tizen sempre retorna NONE. Por ser um sistema operacional novo, o Tizen não possui nenhum benchmark público disponível. Portanto, para fazer a análise quantitativa, foi desenvolvida e utilizada uma aplicação Tizen que obtém informações sensíveis do usuário. Como foi utilizada uma plataforma que não possui todas as características de um dispositivo móvel não foi possível testar todas as fontes de informações sensíveis.

A aplicação desenvolvida obtém informações sensíveis do sistema, realiza a propagação e envia os dados para um servidor externo a cada dois segundos. O tempo de cada experimento foi de 10 minutos, esse processo foi repetido seis vezes, totalizando uma hora de experimento.

Durante esse tempo, foram obtidas informações sobre a porcentagem de uso de CPU e memória. Para auxiliar na obtenção das informações foi desenvolvido um *shell* script utilizando a ferramenta PS [PS 2016] do sistema operacional Linux que é utilizada para mostrar os processos que estão sendo executados no sistema operacional Linux.

A Figura 3 mostra a porcentagem de utilização de memória e CPU do aplicativo de teste que foi desenvolvido. Esse aplicativo recebeu o nome de APP e a informação sensível que é transmitida através de uma conexão HTTP são informações sobre a rede. O aplicativo foi executado em dois ODROIDs, um com o TTizen e outro com o Tizen original sem modificações.

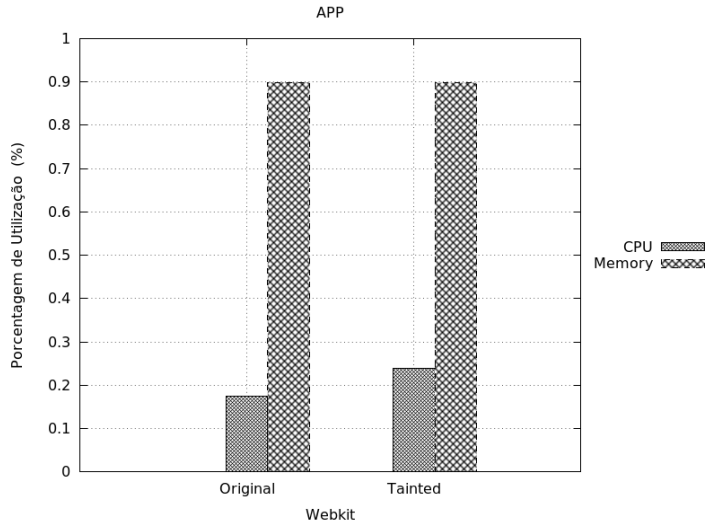


Figura 3. Porcentagem de utilização de memória e CPU do APP em um ODROID com TTizen e outro ODROID com Tizen original.

Os resultados mostram que a porcentagem de utilização de memória da aplicação usando um ODROID com o TTizen e outro ODROID com o Tizen original foi a mesma de 0,9% enquanto que a porcentagem de utilização de CPU foi de aproximadamente 0.19% no ODROID com a versão original e de 0.22% no ODROID com TTizen, obtendo um pequena diferença de 0.03%. A Figura 4 mostra a porcentagem de utilização de memória e CPU do *Web Process* da APP.

Os resultados mostram que a porcentagem de utilização de memória do *Web Process* usando um ODROID com o TTizen foi de aproximadamente 1,7% e a porcentagem de utilização de CPU no ODROID com a versão original foi de aproximadamente 0,9%, obtendo uma diferença de 0,8% enquanto que a porcentagem de utilização de CPU foi de aproximadamente 0.3% no ODROID com a versão original e de 1.5% no ODROID com TTizen, obtendo uma diferença de 1.2%.

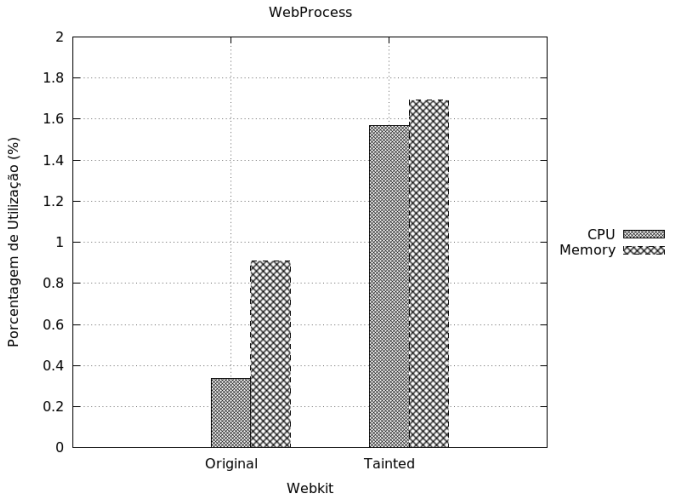


Figura 4. Porcentagem de utilização de memória e CPU do Web Process em um ODROID com TTizen e outro ODROID com Tizen original.

Para o experimento qualitativo foi desenvolvida uma aplicação, denominada *MyObserver*, para notificar o usuário caso ocorra vazamento de dados. Essa aplicação foi desenvolvida em C++ e funciona como um serviço que e em segundo plano para monitorar um arquivo de log.

Quando um vazamento de dados ocorre a informação e horário do vazamento são adicionadas a um arquivo de log, posteriormente esse arquivo é lido e o usuário notificado. A Figura 5 mostra o resultado do experimento qualitativo com a APP sendo executada e com a MyObserver exibindo uma notificação no topo da tela informando ao usuário que uma informação sensível estava sendo vazada.

No terceiro experimento foi utilizada a mesma aplicação que envia as informações sensíveis, porém a informação enviada foi ofuscada através da aplicação de uma codificação *Base64* [Base64 2016]. Como a análise dinâmica da informação realiza o rastreio da informação durante toda a execução da aplicação mesmo que seja aplicada alguma codificação a informação continuará marcada e acompanhada, portanto o experimento obteve os mesmos resultados do experimento anterior com a aplicação MyObserver enviando uma notificação para o usuário.

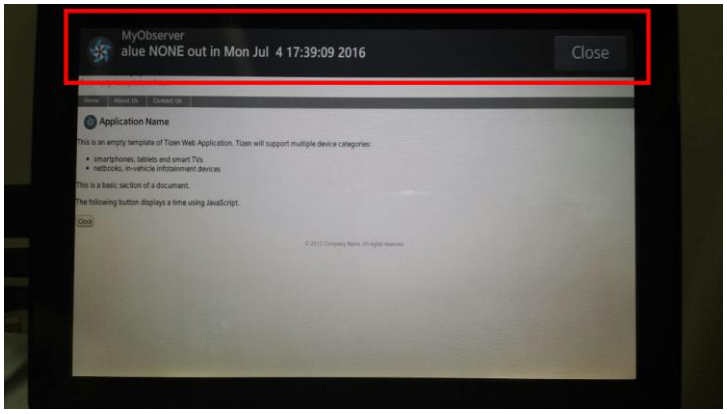


Figura 5. Usuário sendo informado sobre um vazamento de dados no topo da tela do ODROID U3.

6. Conclusões e Trabalhos Futuros

Este artigo apresentou uma versão modificada do sistema operacional Tizen que aplica análise de informação dinâmica em aplicações web. Para realizar a análise dinâmica das informações foi necessário um estudo sobre o sistema operacional Tizen e de como funciona a sua WebKit. Pelas nossas pesquisas esse é o primeiro protótipo que aplica esse tipo de técnica ao sistema operacional Tizen e também é o primeiro protótipo que realiza rastreio de aplicações web em um ambiente móvel.

Os resultados mostraram que no processo de execução do aplicativo no ODROID com TTizen e com o Tizen original a porcentagem de utilização de memória foi a mesma (0,9%) enquanto que a porcentagem de utilização de CPU obteve uma pequena diferença de 0.3%.

Já no processo responsável pela renderização do aplicativo (*Web Process*) a diferença da porcentagem de utilização de memória no ODROID com TTizen e no ODROID com Tizen original foi de 0,8% enquanto que a diferença da porcentagem de utilização de CPU foi de 1.2%. Por fim, no experimento qualitativo o TTizen foi efetivo e informou ao usuário que uma informação sensível estava sendo enviada para um servidor externo.

Para trabalhos futuros serão realizadas as modificações necessárias para cobrir tratamento de exceções, variáveis do tipo real e, principalmente, a propagação implícita, pois diferentes técnicas tem que ser empregadas para abordar esse tipo de propagação.

Outro aspecto pesquisado é que será estudada uma forma de descobrir se o vazamento de dados é com a permissão do usuário, pois existem vezes em que o usuário realmente precisa enviar a informação sensível para servidores de terceiros.

Agradecimentos. Partes dos resultados apresentados nesse artigo foram obtidas através de pesquisas em um projeto chamado "Um Sistema para Detectar e Prevenir Vazamento de Dados em um Ambiente Android", patrocinado pela Samsung Eletrônica da Amazônia Ltda. Dentro dos termos da lei federal brasileira No. 8.387/91. (SUFRAMA).

Referências

Abhishek Bichhawat, Vineet Rajani, Deepak Garg e Christian Hammer “Information Flow in WebKits JavaScript Bytecode” Disponível em: http://www.mpi-sws.org/~dg/papers/post2014.pdf, Janeiro 2016.

Alexa “Alexa – Top Sites by Category: Computers/Performance and Capacity/Benchmark” Disponível em: http://www.alexa.com/topsites/category/Computers/Performance_and_Capacity/

Benchmarking, Janeiro 2016.

Base64. “Base64 encoding and decoding – Web APIs | MDN”. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/WindowBase64/Base64_encoding_and_decoding. Janeiro 2016.

Bassam Sayed, Issa Traoré and Amany Abdelhalim (2014) “Detection and Mitigation of Malicious JavaScript Using Information Flow Control”. Proceedings of the Twelfth Annual International Conference on Privacy, Security and Trust, páginas 264-273.

Daniel Hedin and Andrei Sabelfeld (2012) “Information-flow security for a core of JavaScript”. Proceedings of 25th IEEE Computer Security Foundations Symposium, páginas 3–18.

Dongseok Jang, Ranjit Jhala, Sorin Lerner (2015). Disponível em: https://pdfs.semanticscholar.org/60b9/11615642efeabcb3485c093d7cade0fa77b8.pdf , Dezembro 2015.

ECMA-262 “Standard ECMA-262” Disponível em: http://www.ecma-international.org/publications/standards/Ecma-262.htm, Janeiro 2016.

Firefox OS “Firefox OS – Just what you need – Great smartphone features, apps and more - Mozzila” Disponível em: https://www.mozilla.org/en-US/firefox/os/, Janeiro 2016.

HTML5 “HTML5” Disponível em: https://www.w3.org/TR/html5/, Janeiro 2016.

JASMINE “Jasmine: Behavior-Driven Javascript” Disponível em: http://jasmine.github.io/, Janeiro 2016.

Jaygarl Hojun, Luo Cheng, Kim YooSoo, Choi Eunyoung, Bradwick Kevin, Lansdell Jon “Professional Tizen Application Development” Disponível em: https://goo.gl/uWIpvn, Janeiro 2016.

Jedidiah McClurg, Jonathan Friedman and William Ng “Android Privacy Leak Detection via Dynamic Taint Analysis”. Disponível em: http://www.jrmcclurg.com/papers/ internet_security_final_report.pdf Fevereiro 2016.

Manuel Egele, Christopher Kruegel, Engin Kirda and Giovanni Vigna (2011) “PiOS: Detecting Privacy Leaks in iOS Applications”. Proceedings of the International Secure Systems Lab.

Minh Tran, Xinshu Dong, Zhenkai Liang and Xuxian Jiang (2012) “Tracking the trackers: fast and scalable dynamic analysis of web content for privacy violations”. Proceedings of the 10th International Conference on Applied Cryptography and Network Security, páginas 418-435.

ODROID. “ODROID | Hardkernel”. Disponível em: http://www.hardkernel.com/main/main.php. Janeiro 2016.

PS. “PS man page”. Disponível em: http://www.petefreitag.com/tools/man-pages/ps.html. Janeiro 2016.

Rick Lehrbaum “Slides from Intels Tizen talk at IDF2013 Beijing” Disponível em: http://hackerboards.com/intel-tizen-talk-slides-idf2013/, Fevereiro 2016.

Seth Just, Alan Cleary, Brandon Shirley e Christian Hammer (2011) “Information flow analysis for javascript”. Proceedings of the 1st ACM SIGPLAN international workshop on Programming language and system technologies for internet clients, páginas 9-18.

SungGyeong Bae, Hyunghun Cho, Inho Lim e Sukyoung Ryu (2014) “SAFEWAPI: web API misuse detector for web applications”. Proceedings of the 22nd ACM SIGSOFT international symposium on Foundations of Software Engineering, páginas 507-517.

SunSpider “SunSpider 1.0.2 JavaScript Benchmark” Disponível em: https://webkit.org/perf/sunspider/sunspider.html, Janeiro 2016.

Tizen. “Tizen | An open source, standards-based software platform for multiple devices categories”. Disponível em: https://www.tizen.org. Janeiro 2016.

Tonin, G., “Tendências em computação móvel” Disponível em: http://grenoble.ime.usp.br/~gold/cursos/2012/movel/mono-1st/2305-1_Graziela.pdf, Janeiro 2016.

Vineeth Kashyap, Kyle Dewey, Ethan A. Kuefner, John Wagner, Kevin Gibbons, John Sarracino,Ben Wiedermann e Ben Hardekopf (2014) “JSAI: a static analysis platform for JavaScript”. Proceedings of the 22nd ACM SIGSOFT international symposium on Foundations of Software Engineering, páginas 121-132.

Wei Zheng, Lie David (2014) “LazyTainter: Memory-Efficient Taint Tracking in Manager Runtimes”, ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, páginas 27-38.

webOS “Open webOS” Disponível em: http://www.openwebosproject.org/, Janeiro 2016.

WebKit “WebKit” Disponível em: https://webkit.org/, Janeiro 2016.

William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel and Anmol N. Sheth (2014) “TaintDroid: An Information Flow Tracking System for Real-Time Privacy Monitoring on Smartphones”, ACM transaction on Computer Systems (TOCS), páginas 99-106.

Zhibo Zhao and Fernando C. Colon Osorio (2012) “TrustDroid™": Preventing the use of SmartPhones for information leaking in corporate networks through the use of static analysis taint tracking”, International Conference on Malicious and Unwanted Software, páginas 135-143.

Zhemin Yang and Min Yang (2012) “Leakminer: Detect Information Leakage on Android with static taint analysis”, World Congress on Software Engineering (WCSE), páginas 101-104.

Detecção de ataques por ROP em tempo real assistida por *hardware*

Marcus Botacin[1], **Paulo Lício de Geus**[1], **André Grégio**[2]

[1]Instituto de Computação (IC)
Universidade Estadual de Campinas (Unicamp)
Campinas – SP – Brasil

[2]Departamento de Informática (DInf)
Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

Abstract. *Code injection used to be one of the main attack vectors to subvert systems functioning. The adoption of non-executable pages supported by hardware (NX/XD) and data execution prevention (DEP) eliminated this problem in practice. However, attackers found another way of leveraging control flow deviation by chaining blocks of code (gadgets) through* `ret` *instructions. This is known as Return-Oriented Programming (ROP) and is currently the main injection vector. Recently, techniques based on Control Flow Integrity (CFI) and code length arose to counter such attacks with reasonable effectiveness. We hereby propose a Windows-based, hardware-assisted system, that overcomes limitations of state-of-the-art, hardware-assisted approaches to detect ROP attacks in realtime.*

Resumo. *A injeção de código foi um dos principais ataques contra sistemas computacionais. A adoção das páginas não-executáveis com apoio de hardware (NX/XD) e prevenção de execução de dados (DEP) eliminou o problema na prática. Entretanto, os atacantes passaram a desviar o fluxo de controle legítimo, encadeando blocos de código (gadgets) via instruções de retorno. Tal técnica, a Programação Orientada à Retorno (ROP), tornou-se o principal vetor de injeção. Recentemente, abordagens baseadas em integridade do fluxo de controle (CFI) e comprimento de gadgets surgiram para tratar tais ataques. Propomos aqui um sistema assistido por hardware, construído sobre sistemas Windows, que supera limitações de abordagens da literatura na detecção de ataques por ROP em tempo real.*

1. Introdução

A injeção de código figura como um dos maiores vetores de ataque contra sistemas computacionais, seja em sua vertente *Web* (p.ex., *SQL Injection* e *Cross Site Scripting*) ou de exploração da memória, como *buffer overflow/stack overflow*. Devido à popularidade dos ataques de injeção, vários mecanismos foram desenvolvidos de modo a se tentar bloqueá-los, tais como a inserção de canários na pilha, a desativação da execução de código em páginas de dados, pilha e heap (DEP, NX, XD) e a aleatorização de endereços-base (ASLR) de bibliotecas e módulos (que impede ataques com *offset* estático). Embora esses mecanismos tenham sido efetivos no bloqueio de execução de códigos externos, o abuso de sistemas ainda persiste, uma vez que os atacantes encontraram uma nova forma de subverter o fluxo de execução original através do reuso de código legítimo. Esta técnica,

conhecida como *Return-Oriented Programming* (ROP), consiste no encadeamento de trechos de código e permite todo tipo de computação.

Abordagens para se lidar com ROP se dividem entre: aplicáveis ao código-fonte através da instrumentação do compilador; *patching* de binários em tempo de execução; detecção em tempo real com auxílio de *hardware*. A primeira abordagem é de aplicação restrita, pois o código-fonte nem sempre está disponível; a segunda abordagem é extremamente dependente de arquitetura e pode sofrer de limitações para ser aplicada em *software* protegido e/ou com geração de código em tempo de execução; a terceira abordagem destaca-se por sua flexibilidade, embora sua implementação possa causar *overhead*. Neste trabalho, propõe-se implementações de técnicas para detecção de ataques por ROP em tempo real, no ambiente Windows[1], com auxílio de *hardware*, buscando minimizar o *overhead* de análise e desenvolvimento em comparação com as ferramentas estado-da-arte. As contribuições incluem, além da detecção dos ataques por ROP suportada por instruções do processador e da minimização de *overhead*, a introdução de uma técnica de monitoração do sistema como um todo, que não depende de informações *a priori* de aplicativos-alvo, tampouco de injetar código nos processos monitorados.

Este artigo é dividido da seguinte maneira: introduz-se na Seção 2 os conceitos básicos de ROP e os detalhes de funcionamento dos monitores de desempenho utilizados, bem como as abordagens existentes para a detecção de ataques por ROP e suas limitações; na Seção 3, discute-se o sistema proposto e os detalhes das técnicas utilizadas em sua implementação; na Seção 4, apresenta-se os testes realizados para validar a eficácia do sistema proposto, resultados obtidos e discussão, incluindo as limitações e outros ataques similares em potencial. Conclui-se o artigo na Seção 5.

2. Aspectos Técnicos e Trabalhos Relacionados

Nesta seção, são apresentados conceitos sobre Programação Orientada a Retorno (ROP) e monitoramento de desvios (*branches*), tecnologia usada na implementação da proposta deste artigo. São também apresentadas as abordagens já propostas na literatura para lidar com ataques ROP, classificadas de acordo com seu cenário de atuação (de acordo com [Bania 2010]). De modo geral, tais abordagens se propõem a reforçar alguma política de controle de fluxo (CFI) e se tornaram populares a ponto de serem implementadas em aplicativos, como Chromium/Google Chrome[2] e EMET [Microsoft 2013].

2.1. Conceitos

ROP. Ataques por injeção de código são uma das principais formas de desviar o fluxo normal de execução de aplicações para o propósito dos atacantes. As variações de *heap* e *stack overflow* mobilizaram a indústria em busca de soluções para barrar este tipo de ataque. As soluções propostas mitigaram o problema à medida em que se buscou detectar a injeção de código, através do uso de canários, impedir ataques baseados em *offset* fixos através de ASLR e impedir que os códigos injetados fossem executados (DEP, NX, XD). Contudo, os atacantes desenvolveram uma nova forma de desviar o fluxo de execução que não inclui executar diretamente código malicioso injetado, o que recoloca a atenção da comunidade sobre este tipo de ataque. Esses ataques baseiam-se em encadear uma sequência de instruções do código original de modo a se executar uma ação maliciosa. A execução não é impedida por nenhum mecanismo de proteção, uma vez que se

[1] Este sistema foi escolhido devido sua relevante fatia de mercado, afetando a maioria dos usuários. [2] `http://www.chromium.org/developers/testing/control-flow-integrity`

trata da execução de código legítimo, embora fora da especificação original. Tais ataques de reuso de código são conhecidos como ROP, dado que as sequências de instruções são usualmente terminadas em instruções de retorno (`ret`), as quais desviam o fluxo de execução para um outro ponto. As referidas sequências, denominadas *gadgets*, são geralmente formadas por poucas instruções que, quando combinadas, permitem a realização de uma computação completa (execução de uma ação). A Figura 1 exemplifica um ataque por ROP onde se subtrai dois valores da memória (`eax,edx`), resultando em `eax`.

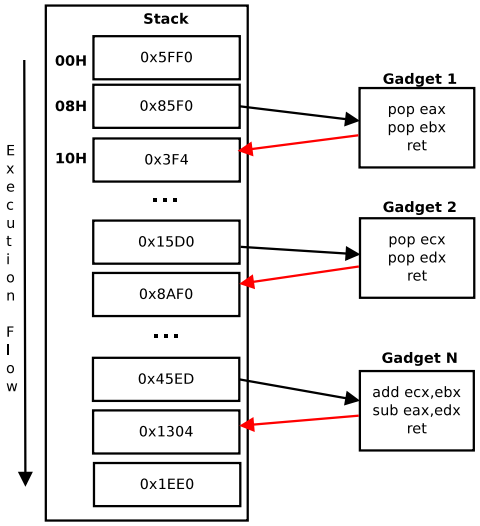


Figura 1. Ataque por ROP - subtração dos registradores EAX e EDX.

Monitoramento de *branch*. Processadores modernos de diferentes fabricantes e arquiteturas contam com monitores em *hardware* para obter informações de desempenho e funcionamento. Dentre eles, destaca-se o monitoramento de *branch*, capaz de armazenar informações como origem e destino de instruções de desvio de fluxo (p.ex., `JMP`, `CALL`, `RET`). Neste artigo, usa-se a tecnologia de monitoramento presente nos processadores Intel, por questões de disponibilidade. O monitor de *branch* é um recurso do processador que pode armazenar os últimos desvios executados em registradores específicos (MSR), em número limitado, ou em uma página do sistema operacional, limitado apenas por um *threshold* definido pelo desenvolvedor que, ao ser atingido, pode gerar uma interrupção. O monitoramento, por se tratar de recurso de *hardware*, introduz *overhead* teoricamente nulo. Além disso, conta com filtros que permitem selecionar o modo de captura (*kernel/userland*) e os tipo de ações (`CALL`, `RET`, `JMP`). O acesso aos dados é permitido apenas no nível do *kernel* e é em geral implementado por um *driver*. O uso dos recursos de monitoramento é independente de plataforma e, portanto, portável – requerindo apenas bibliotecas compatíveis para a interpretação dos dados, como nos procedimentos de *disassembly* e *dump* de memória.

2.2. Trabalhos Relacionados

A seguir, são mostradas algumas abordagens para lidar com ataques por ROP.

Abordagens em tempo de compilação/Reescrita de código. Para mitigar a exploração causada em um programa por um ataque por ROP, as abordagens em tempo de compilação visam, em geral, impedir a construção de determinados tipos de *gadgets*, que estes estejam acessíveis ou que o compilador crie certas estruturas de controle mais vulneráveis. [Bletsch et al. 2011a] apresentam *control flow locking* (CFL), uma técnica

alternativa de proteção que cria um marcador para as estruturas de transição. Seu funcionamento é semelhante ao de um *mutex*, que deve ser ativado sempre que o código precisar ser executado. A idéia é que códigos oriundos de ataques por ROP se insiram em um ponto sem passar por este marcador, causando a detecção da execução do código ilegítimo. Sua implementação é por reescrita do código *assembly* gerado. *Gfree* [Onarlioglu et al. 2010], por sua vez, trata do problema das instruções desalinhadas, as quais podem ser convertidas em *gadgets* para ataques por ROP. O sistema implementa a substituição dessas instruções usando um pré-processador para o *GNU Assembler*. As abordagens citadas são de aplicação restrita, pois dependem do código-fonte e deixam sistemas legados desprotegidos, por não serem compatíveis com programas já compilados.

Instrumentação. Para contornar algumas limitações e ser aplicável a *software* pré-existente, muitas abordagens se baseiam na instrumentação do binário para a detecção de ataques por ROP. [Davi et al. 2009] propõe uma arquitetura de verificação dinâmica (DynIMA) que modifica o conteúdo do programa em tempo de carregamento para incluir instruções de *tainting* para monitoração. *DROP* [Chen et al. 2009] baseia-se em interceptar a execução de instruções `ret` e verificar o suposto *gadget*. Sua implementação é alcançada por instrumentação no *Valgrind*[3]. *Ropguard* [Fratric 2012] não visa detectar os *gadgets* em si, mas a transição destes para APIs que possam vir a ser usadas para propósitos maliciosos, como a criação de um processo no sistema. A monitoração é realizada através da injeção de DLL. *ROPMEMU* [Graziano et al. 2016] implementa um emulador capaz de reconstruir os fluxos de execução e analisar ataques ROP a partir de *dumps* de memória física. Esta solução é altamente dependente e limitada pela implementação dos mecanismos de emulação e aquisição de memória. Tais abordagens, embora aplicáveis a qualquer binário, sofrem tanto com as já estudadas limitações de seus mecanismos de monitoração quanto com o fato do *overhead* ser proibitivo em alguns cenários de uso.

Reescrita de binário. A tentativa de superar as limitações dos mecanismos de instrumentação resultou na técnica de reescrita de binários já existentes, a qual pode ser aplicada uma vez durante seu primeiro uso de forma a gerar um binário instrumentado. ILR [Hiser et al. 2012] expande a ideia de aleatorização de endereços de dados (ASLR) para instruções, de modo que ataques com *offsets* codificados estaticamente não funcionem. Estratégia semelhante é proposta em [Pappas et al. 2012] e [Wartell et al. 2012]. Já [Zhang et al. 2013] propõem uma abordagem que consiste em uma política na qual desvios indiretos são proibidos, exceto para uma série de posições especificadas por uma *white-list*. Contudo, as técnicas baseadas em reescrita de binários ainda apresentam limitações quando aplicadas a binários que geram códigos dinamicamente, como os que executam em interpretadores ou máquinas virtuais.

Monitoramento com suporte de *hardware*. Uma abordagem de funcionamento mais amplo é o uso de recursos de *hardware* para a construção de um mecanismo de monitoramento, pois não dependem do código-fonte ou da reescrita do binário e atuam em tempo real. *Hypercrop* [Jiang et al. 2011] é uma máquina virtual com suporte de *hardware* (HVM) que pode ser utilizada para monitorar o sistema de forma ampla e identificar um ataque ROP. No entanto, dado seu grande número de interrupções, o *overhead* de monitoramento se torna proibitivo. Abordagens com baixo *overhead*, de maneira geral, têm se baseado no uso de contadores de desempenho, em especial o monitor de *branch* mencionado anteriormente. [Yuan et al. 2011] e [Kompalli 2014] se utilizam dos dados

[3] `http://valgrind.org/`

de desempenho providos pelo `perf`[4] do Linux e *VTune Amplifier*[5] da Intel, respectivamente, para a identificação de ataques. Embora exibam exemplos bem sucedidos, os mecanismos usados não são totalmente apropriados para o tipo de monitoramento pretendido. Isto causou o surgimento de ferramentas de propósito específico, como *ROPecker* e *KBouncer*. *ROPecker* [Cheng et al. 2014] é um módulo para o *kernel* do Linux capaz de detectar ataques por ROP durante sua execução. Sua lógica consiste de uma janela deslizante que verifica o comprimento da cadeia de *gadgets* e atua em duas etapas: a criação de um banco de dados de *gadgets* por meio do *disassembly* da aplicação e a verificação dos acessos usando o recurso de *last branch record* (LBR) dos processadores. *KBouncer* [Pappas et al. 2013] detecta ataques por ROP usando LBR para identificar se uma chamada de função veio de um *gadget* ou não. Para tanto, as chamadas de função dos aplicativos monitorados são interceptadas por meio da biblioteca *Detours*[6]. Os aplicativos a serem monitorados podem ser escolhidos através de uma interface semelhante à do Microsoft EMET[7]. As políticas de detecção se baseiam no comprimento da cadeia e na existência de instruções `RET` sem a respectiva `CALL`. Embora as abordagens no estado-da-arte tenham avançado os métodos de detecção de ataques por ROP, estas ainda apresentam sérias limitações, sobretudo na aplicação do mecanismo LBR (provê informação limitada) e pela injeção de código no binário monitorado ser uma escolha de projeto. A proposta do presente artigo é a de tratar adequadamente tais limitações.

3. Arquitetura e Implementação

Para alcançar os objetivos estabelecidos para este trabalho, propõe-se um sistema de monitoramento baseado em arquitetura cliente-servidor, onde o servidor é um *driver* de *kernel* responsável por obter as informações do sistema e o cliente é um programa responsável por interpretar as informações obtidas.

O *driver* captura as informações diretamente do *hardware*, utilizando o monitor de *branch*. O monitor utilizado é o *Branch Trace Store* (BTS), em vez do LBR usado em outras abordagens da literatura, pois permite definir o *threshold* de armazenamento e a geração de uma interrupção. Assim, definiu-se este para uma instrução de desvio, que interrompe o sistema sempre que esta ocorre, além de permitir a identificação do processo que a causou. Os dados capturados são enfileirados em ordem.

O cliente se comunica com o *driver* via IRP, retirando os dados da fila criada pelo lado servidor. O processo de introspecção é realizado para cada tupla (endereços de origem e destino da instrução de desvio) de modo a identificar a biblioteca ou processo base de um endereço-alvo. Isto é feito por meio de chamada à `GetModuleHandle`—para lidar com ASLR—e pelas ferramentas *Dumpbin*[8] e/ou *DLL Export Viewer*[9]—para conhecer a função chamada por meio de seu *offset*.

É importante ressaltar que a abordagem proposta não requer injeção de código em nenhum processo. Além disso, dado que a captura de dados é feita de modo *system-wide*, todos os processos podem ser monitorados simultaneamente. Por fim, a adoção de BTS no lugar de LBR elimina o risco da sobrescrita de valores. Mais detalhes sobre o sistema proposto, em sua versão inicial e estendida, são descritos a seguir.

[4] `https://perf.wiki.kernel.org`
[5] `https://software.intel.com/en-us/intel-vtune-amplifier-xe`
[6] `http://research.microsoft.com/en-us/projects/detours/`
[7] `https://support.microsoft.com/en-us/kb/2458544`
[8] `https://support.microsoft.com/en-us/kb/177429`
[9] `http://www.nirsoft.net/utils/dll_export_viewer.html`

3.1. Sistema Inicial

Conforme mencionado, o cliente é responsável por obter os dados capturados pelo servidor e processá-los de acordo com regras estabelecidas. O cliente pode filtrar os dados por processo ou atuar de maneira abrangente, monitorando todas as aplicações em execução. Neste caso, embora se tenha uma área de proteção maior, o desempenho do sistema piora devido ao maior processamento requerido. Para minimizar este problema, força-se a execução do cliente em um núcleo do processador diferente do núcleo sob monitoração, não gerando penalidades de desempenho para a aplicação monitorada. O cliente pode optar por implementar diferentes políticas, como as discutidas a seguir. Quando o cliente identifica uma violação de política, emite um alerta através de seu *daemon* (Figura 2).

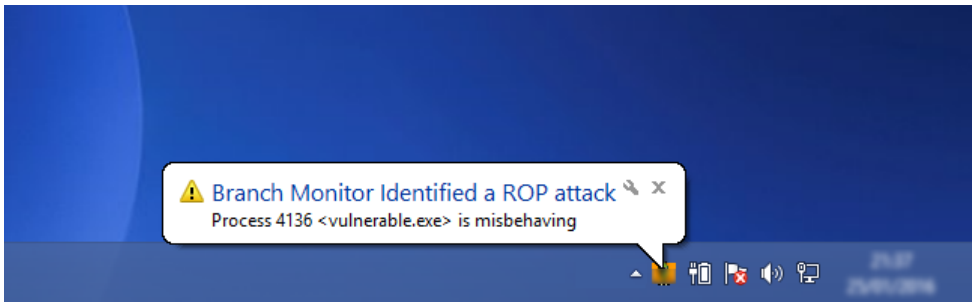


Figura 2. Notificação de ataque identificado emitida pelo cliente de monitoração.

3.1.1. A Política do "CALL−RET"

Esta política pode ser implementada para processos sob monitoração por meio da decodificação do primeiro *byte* da instrução apontado pelo endereço capturado pelo monitor de *branch*, identificando seu tipo. Para efeitos de comparação e facilidade de implementação, considerou-se para esta proposta a mesma versão da política usada por *KBouncer* [Pappas et al. 2013]. Essa versão é considerada menos restritiva, pois desconsidera casos degenerados, por exemplo, o uso de instruções `JMP` e `POP+RET` para fazer o papel de chamadas do tipo `CALL`. A Tabela 1 reproduz os *opcodes* relativos às instruções do tipo `CALL`[10], enquanto que a Tabela 2 reproduz os da instrução `RET`[11].

Tabela 1. *Opcodes* de instruções do tipo `CALL`.

Opcode	E8 cw	E8 cd	FF /2	9A cd	9A cp	FF /3
Mnemonic	CALL rel16	CALL rel32	CALL r/m16 ou r/m32	CALL ptr16:16	CALL ptr16:32	CALL m16:16 ou m16:32

Tabela 2. *Opcodes* de instruções do tipo `RET`.

Opcode	C3	CB	C2 iw	CA iw
Mnemonic	RET	RET	RET imm16	RET imm16

A Listagem 1 ilustra correspondências de instruções `RET` com `CALL` anteriores, o que configura um fluxo de execução íntegro. Em caso de desvio de fluxo por ataque por ROP, seriam vistas instruções `RET` sem as respectivas `CALL`. Os endereços (`FROM` e `TO`) são obtidos diretamente através do monitor BTS. O *framework* proposto é responsável por obter o PID, através da interrupção, os *bytes* `INSTR`, através da leitura da memória correspondente aos endereços, e interpretá-los (mnemônicos `CALL` e `RET`), através das Tabelas 1 e 2.

[10] `http://x86.renejeschke.de/html/file_module_x86_id_26.html` [11] `https://pdos.csail.mit.edu/6.828/2004/readings/i386/RET.htm`

Listagem 1. Correspondências "CALL-RET" indicando fluxo de execução íntegro.

```
1 PID 3140 FROM   6b8e7f17 INSTR e8 - CALL
2 PID 3140 TO     6b9d90c1 INSTR c3 - RET
3 PID 4196 FROM   77b2ce8e INSTR e8 - CALL
4 PID 4196 TO     77aa591e INSTR c2 - RET
5 PID 2532 FROM   3de50b6c INSTR e8 - CALL
6 PID 2532 TO     40714979 INSTR c2 - RET
```

3.1.2. A Política do Comprimento do *gadget*

Apesar de efetiva em muitos casos, a política de `CALL-RET` não é suficiente para cobrir todos os casos de ataques por ROP. Logo, abordagens adicionais fazem-se necessárias. Implementou-se então no sistema a política de comprimento de *gadget*, também disponível no Kbouncer. Esta política consiste em identificar uma sequência de *gadgets* curtos, característica dos *payloads* de ataque por ROP, dado que sequências longas tendem a incorrer em muitas mudanças de estado, dificultando seu sequenciamento para propósitos maliciosos. Na abordagem proposta, foi implementada uma janela de 20 instruções (mesmo tamanho usado por KBouncer).

A Listagem 2 ilustra o tamanho dos blocos de código de programas legítimos, obtidos manualmente a partir da memória (lida com `ReadProcessMemory`[12] e interpretados pela *LibOpcodes*[13] e/ou *Capstone*[14]). A Listagem 3 reproduz o tamanho dos *gadgets* do *exploit* ROP tal qual utilizado pelo Kbouncer. Nota-se que, para programas legítimos, o tamanho dos blocos[15] é significativamente superior ao *threshold* estabelecido, diferentemente dos *gadgets* ROP.

Listagem 2. Comprimento dos blocos em programas legítimos (em número de instruções).

```
1 PID 3820 FROM 5f0dea04 TO 5f0deb30 INSTR 15
2 PID 3820 FROM 5f0deb4a TO 5f0deb53 INSTR 21
3 PID 3820 FROM 5f0dea0e TO 5f0dea17 INSTR 19
```

Listagem 3. Tamanho dos *gadgets* encontrados em programa atacado por ROP (em número de instruções).

```
1 ADDR 7C3411C0 INSTR 2
2 ADDR 7C3415A2 INSTR 1
3 ADDR 7C34252C INSTR 2
4 ADDR 7C346C08 INSTR 1
5 ADDR 7C378C84 INSTR 3
```

3.1.3. Discussão Preliminar

Embora implemente as mesmas políticas presentes no *KBouncer*, o sistema proposto neste artigo possui diferenças que o tornam sensivelmente mais amplo no âmbito de proteção das aplicações. Por exemplo, enquanto o *KBouncer* destina-se à proteção de aplicações de forma independente, o sistema proposto neste é capaz de monitorar mais de um processo

[12] `https://msdn.microsoft.com/en-us/library/windows/desktop/ms680553(v=vs.85).aspx` [13] `https://github.com/Groundworkstech/pybfd` [14] `http://www.capstone-engine.org` [15] em número de instruções

simultaneamente (modo *system-wide*). Essa característica faz com que não seja necessário injetar código no processo monitorado, ao contrário do *KBouncer*, que requer a injeção de uma DLL no processo monitorado para a redireção por *Detours*.

Além disso, o sistema proposto não depende da chamada de APIs para a avaliação de registros de *branch*, a qual ocorre quando há interrupção pelo mecanismo BTS. *KBouncer*, por sua vez, exige a redireção do fluxo para uma chamada de API previamente interceptada (via *hooking*). Outra diferença relevante do sistema proposto é a capacidade de armazenar registros de *branch* ilimitados devido ao mecanismo de BTS. Já o *KBouncer* é limitado aos registradores do mecanismo LBR (atualmente 16 registradores na arquitetura Haswell). [Carlini and Wagner 2014] apontam esta limitação do *KBouncer* como uma possível forma de se evitar a detecção de ataques por ROP.

Apesar de geralmente efetivas, as políticas disponíveis no *KBouncer* e no sistema aqui proposto não são capazes de lidar com todos os casos de ataques ROP, ou estão suscetíveis a técnicas de evasão. [Göktaş et al. 2014] discute que, apesar de elevar o nível de dificuldade para a efetividade dos ataques, abordagens baseadas no comprimento dos *gadgets* são de difícil calibração—um comprimento muito grande pode permitir que ataques sejam bem sucedidos e comprimentos muito pequenos podem impedir aplicações legítimas de funcionar. O autor discute em detalhes tanto o conceito de comprimento do *gadget* como das cadeias que estes formam. [Carlini and Wagner 2014] mostram como é possível construir longas sequências de *gadgets*, tornando assim os *payloads* de ROP mais parecidos com blocos de código de aplicações legítimas. Portanto, requer-se esforços complementares para se lidar de forma consistente com ataques do tipo ROP.

3.2. Sistema Estendido

Como alternativa para lidar com ataques por ROP, destaca-se o surgimento de heurísticas com base em determinar a frequência global de desvios de fluxo encontrados em aplicações em operação normal e sob ataque. Dentre as abordagens disponíveis, considerou-se a apresentada por [Ferreira et al. 2014], que instrumenta aplicações por meio do *framework* PIN [Luk et al. 2005] para determinar e verificar o limiar da referida frequência. O sistema estendido conta com implementação auxiliada por *hardware* e baseia-se no sistema de monitoração de *branch* já apresentado para detectar ataques por ROP. A comparação entre a abordagem de Ferreira et al. (com PIN) e o sistema estendido deste artigo explicitam as vantagens do último: não-dependência de injeção de código (feita pelo PIN); uso em múltiplas aplicações sem a necesssidade de instrumentação individual; *overhead* de captura teoricamente nulo, devido ao mecanismo de monitoração de *branch* em *hardware*. Na prática, todo *overhead* adicionado deve-se ao processamento da política. Para não impactar no desempenho da aplicação monitorada, pode-se executar o monitor em um núcleo de CPU diferente do da aplicação, como feito em [Quinn 2012].

3.2.1. Implementação

Na implementação do sistema estendido, levou-se em conta as considerações apresentadas por [Ferreira et al. 2014] como base de funcionamento do algoritmo de janela. O funcionamento do sistema aqui proposto é análogo à implementação com PIN, ou seja, baseia-se na interpretação dos blocos de dados resultantes da interrupção gerada pelo mecanismo de BTS. Os dados fornecidos pelo BTS são os endereços de origem e de destino do desvio. Assim, considerando duas interrupções consecutivas, tem-se o ponto de entrada e o de saída em um bloco de código, o qual pode ser entendido como os blocos

Listagem 4. Identificação de um bloco de código através da diferença entre dois endereços consecutivos. Neste caso, o bloco tem início em `0x48ff5ab8` e término em `0x48ff5ac0`.

```
1 PID: 4876 FROM: 48ff5ab0 TO: 48ff5ab8
2 PID: 4876 FROM: 48ff5ac0 TO: 48ff5ad0
```

básicos (*Basic Blocks* - BBLs) do PIN. A leitura do conteúdo de memória no bloco permite a obtenção das instruções intermediárias, pois sabe-se que a extremidade do bloco é uma instrução de desvio. Tem-se então um desvio por bloco e, de acordo com o tamanho da janela a ser aplicada ao bloco (ou a múltiplos blocos), pode-se contar a frequência de desvios como no algoritmo proposto originalmente.

A Listagem 4 ilustra a identificação de blocos a partir de *branches* consecutivos. Na linha 1, inicia-se a execução do bloco residente no endereço `0x48ff5ab8`. Na linha 2, na interrupção seguinte, tem-se que o fluxo sai do bloco pela instrução presente no endereço `0x48ff5ac0`. Desta forma, sabemos que foram executados 8 *bytes* (`c0-b8`) de instruções. Deve-se realizar o *dump* de memória da instrução correspondente a esses *bytes* para identificar as instruções executadas.

A Figura 3 ilustra quatro blocos de código distintos, apresentando, portanto, quatro instruções de desvio. Para janelas de 16 instruções, por exemplo, tem-se que a frequência de desvios pode variar de um a três em 16.

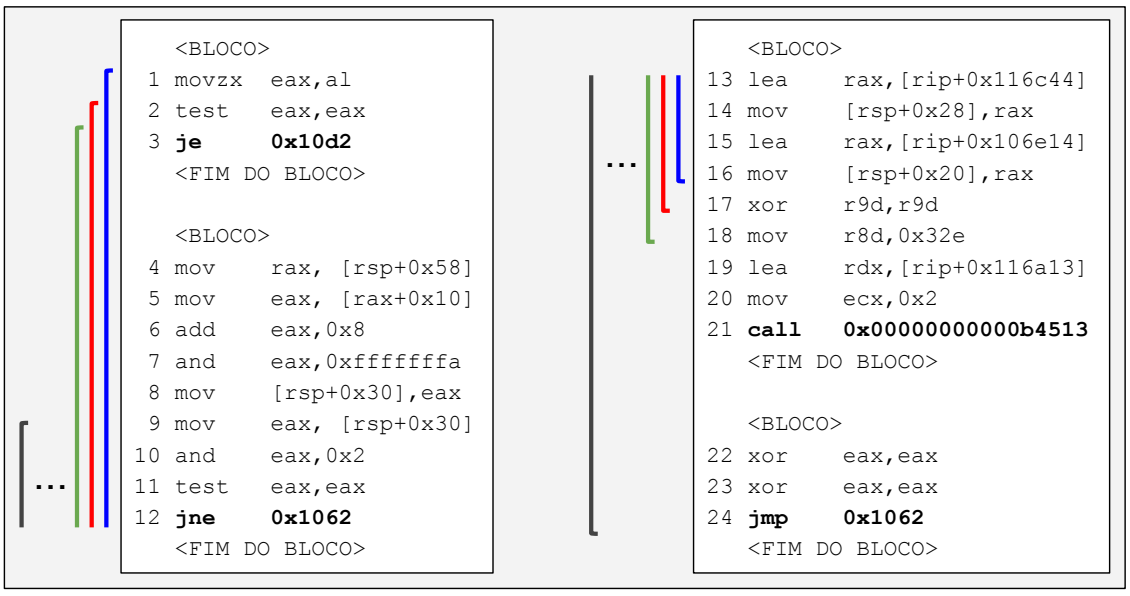


Figura 3. Blocos com instruções de desvios nas extremidades (em negrito) e janela deslizante de 16 instruções.

4. Testes e Resultados

Para validar a aplicação da solução proposta para detecção de ataques por ROP com casos reais, usou-se o *exploit* de Nguyen [Son 2011] em ambiente de testes contendo a vulnerabilidade descrita pelo CVE 2011-0065 [CVE 2011]. O referido *exploit* faz uso de ROP para se evadir das técnicas de proteção do Windows, permitindo a exploração de uma falha de *use-after-free* no navegador Firefox 3.5. A escolha deste *exploit* deu-se pela

disponibilidade de uma amostra cuja execução funcionasse em sistemas operacionais modernos (Windows 7 e 8), ainda que de 32 bits (necessário para o sucesso do ataque de *heap spray*). Cabe enfatizar que a predominância de amostras de ataque por ROP tem por alvo outros sistemas operacionais (Windows XP e Linux), impedindo a comparação direta do sistema proposto com outras abordagens existentes.

Nos testes, amparados pelas discussões e estudos de caso da literatura, foram aplicadas as configurações ótimas definidas pelos respectivos autores. A execução do *exploit* disponível resultou em *crash* do navegador e na ativação do sistema proposto causada pelas métricas de comprimento de *gadget* e densidade de desvios. Os resultados são detalhados a seguir.

4.1. Comprimento de *gadget*

A Tabela 3 mostra um trecho da janela de instruções de desvio capturada pelo mecanismo de monitoração de *branch* durante a execução do *exploit*. Observa-se a execução de pequenos *gadgets* de dois bytes cada (entradas `0x7c346c0a-0x7c346c0b` e `0x7c37a140-0x7c37a141`.

Tabela 3. Janela de instruções de desvio contendo 2 *gadgets* de 2 *bytes* e 2 instruções.

FROM	TO
—-	0x7c346c0a
0x7c346c0b	0x7c37a140
0x7c37a141	—-

A fim de prover uma análise qualitativa do fato observado, fez-se o *disassembly* estático da biblioteca *MSVCR71.dll 7.10.3052.4 - 32bits* com a ferramenta *objdump*, resultando na Listagem 5.

Listagem 5. Código legítimo (alinhado) contendo uma sequência de bytes que pode ser abusada em um ataque *(gadget).*

```
1 7c346c08:     f2 0f 58 c3             addsd   %xmm3,%xmm0
2 7c346c0c:     66 0f 13 44 24 04       movlpd %xmm0,0x4(%esp)
```

Considerando o salto para o endereço `0x7c346c0a`, desalinhado, obtém-se o *gadget* real executado (*bytes* \x58\xc3), como mostrado na Listagem 6.

Listagem 6. Código desalinhado contendo o *gadget* realmente executado.

```
1 0x1000 (size=1)   pop     rax
2 0x1001 (size=1)   ret
```

4.2. Densidade de desvios

Para a verificação da densidade de desvios, usou-se uma janela de 32 instruções, uma vez que esta é a que provê maior poder de identificação [Ferreira et al. 2014]. Neste teste, cujos resultados são apresentados na Figura 4, observa-se que aplicações limitadas por *I/O*, tais como "adobe", "soffice" e "mplayer", apresentam baixa densidade de desvios. Uma aplicação limitada pela *CPU*, como a "superpi", apresenta densidade de desvios superior, ainda que dentro do *threshold* estabelecido (10 instruções de desvio). Aplicações como o navegador Internet Explorer ("iexplo") apresentam limitações tanto de *I/O*, ao se conectar ao servidor remoto, quanto de *CPU*, ao renderizar a página, e por isso apresentam taxa

de desvios mediana. O *exploit* apresentou a maior taxa de desvios dentre os calculados (superior ao limiar), conforme esperado.

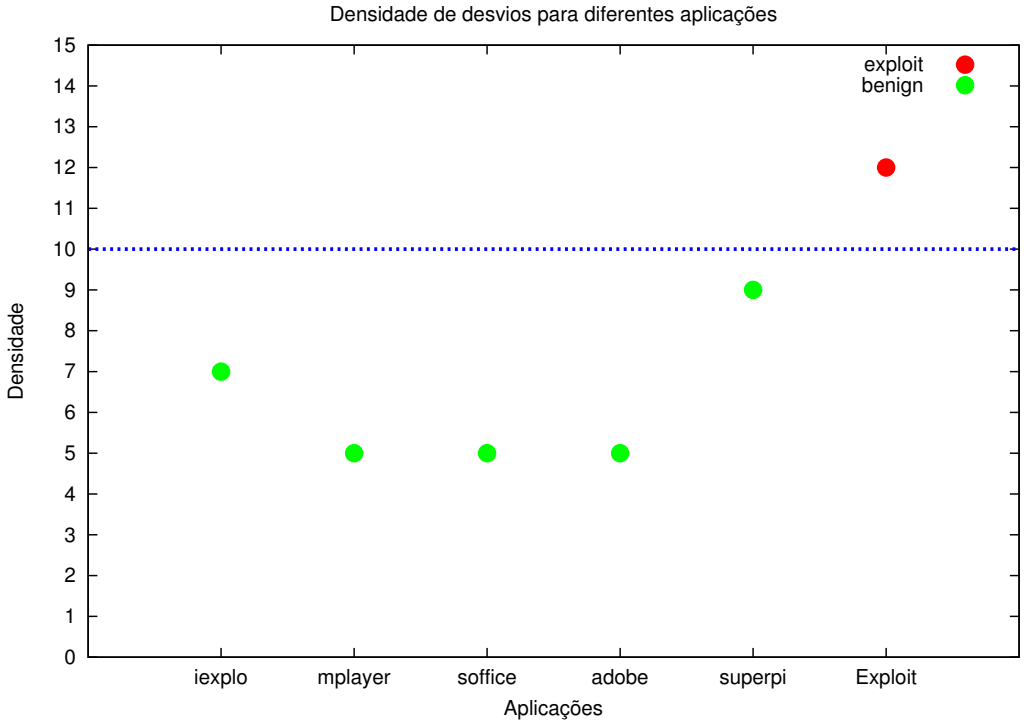


Figura 4. Densidade de desvios em aplicações "benignas" e no *exploit* de Nguyen.

4.3. Discussão

Devido à natureza dinâmica dos ataques ROP, as dificuldades em sua mitigação são inúmeras. O fato do código executado ser "legítimo" impacta na detecção do ataque e na criação de heurísticas, pois pode impedir que aplicações não maliciosas funcionem corretamente. A monitoração do fluxo de execução é dificultada por haver várias formas de se realizá-la, com saltos diretos e indiretos. Bloquear tais desvios, contudo, é impraticável—muitas formas de código polimórfico, compilação *Just-In-Time* (JIT), entre outros, se baseiam nesse tipo de construção.

Abordagens baseadas em tempo de compilação não são suficientes para o bloqueio deste tipo de ameaça, uma vez que novas formas de se construir os *gadgets* são frequentemente encontradas [Schuster et al. 2014]. Ademais, novas formas de ataques são propostas periodicamente, como os ataques de programação orientada a saltos (JOP) [Bletsch et al. 2011b] e a laços (LOP) [Lan et al. 2015]. Portanto, uma solução para os ataques por ROP deve ser encarada com abordagem multi-fatorial, como as empregadas para ataques por *buffer overflow*. Nestas, a solução passa por abordagens em tempo de compilação (canários), em nível de sistema operacional (ASLR) e em nível de *hardware* (DEP/NX e, mais recentemente, MPX [Intel 2013]).

Dado o cenário apresentado dos ataques por ROP, o sistema proposto neste artigo é capaz de complementar as demais abordagens existentes. Além disso, o monitoramento de *branch* pode ser feito mais facilmente com base no cumprimento de hipóteses acerca das garantias providas por outros fatores de proteção (externos ou do sistema operacional).

Considerando as limitações na detecção de ataques por ROP, a monitoração pelo mecanismo de *branch* impõe vários desafios, alguns deles não completamente solucionados. Um exemplo é o tratamento de transição para o espaço de *kernel*, onde há perda de controle do endereçamento de retorno devido ao mecanismo de BTS ter sido desabilitado no *kernel*. A perda do endereço de retorno, entretanto, não afeta o detector de ataques por ROP, pois apenas os blocos extremos (de transição) são afetados e estes são significativamente menores do que todas as janelas utilizadas. Processo semelhante ocorre quando se usa comunicação entre processos (IPC) de forma direta, mas este não é o cenário de aplicação de *exploits* baseados em ROP. É possível implementar a habilitação do mecanismo para detecção de ataques no nível do *kernel*, porém isso requer esforço adicional na interpretação dos blocos, já que os dados precisam ser filtrados e ameaças neste nível estão fora do escopo deste trabalho. Seu tratamento é deixado como trabalho futuro.

Finalmente, existem outros ataques que envolvem a computação baseada em *gadgets*. Construções do tipo *weird computing* [Vanegue 2014, Bangert et al. 2013, Shapiro et al. 2013] podem vir a ser a próxima forma de exploração, dado que possuem características similares aos ataques por ROP (como o uso de elementos legítimos).

4.4. Desempenho

O *overhead* adicionado pela solução é dependente da aplicação e do número de instâncias monitoradas. Em um caso de monitoração onde todo o *disassembly* é realizado em tempo de execução, este pode atingir valores de até 43%. Tais valores, contudo, são inferiores às soluções estado-da-arte, que atingem valores de até 100%. Maiores informações sobre o desempenho da solução proposta podem ser encontradas na página *web* da solução[16].

5. Conclusão

Neste artigo, introduziu-se uma nova solução de monitoramento em tempo real de ataques por ROP, a qual não requer injeção de código em processos ou recompilação/instrumentação de binários. A solução proposta foi utilizada para implementar algoritmos de detecção por integridade de fluxo de controle e heurística de detecção por comprimento de *gadgets*. Implementou-se também uma abordagem heurística para o cálculo da frequência de instruções de desvios, que mostrou-se suficiente para diferenciar execuções de blocos de códigos benignos de um *exploit* baseado em ROP. Fazer uso de suporte de *hardware*, como no caso do sistema proposto, é um passo importante para o aprimoramento das técnicas de detecção deste tipo de ataque.

Agradecimentos

Os autores agradecem o apoio recebido do Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq via Projeto MCTI/CNPq/Universal-A 14/2014 (Processo 444487/2014-0) e da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES, em especial via Projeto FORTE - Forense Digital Tempestiva e Eficiente (Processo: 23038.007604/2014-69 - Edital 24/2014 - Programa Ciências Forenses).

Referências

Bangert, J., Bratus, S., Shapiro, R., and Smith, S. W. (2013). The page-fault weird machine: Lessons in instruction-less computation. In *Proc. of the 7th USENIX Conf. on Offensive Technologies*, WOOT'13, pages 13–13.

[16] `https://sites.google.com/site/branchmonitoringproject/`

Bania, P. (2010). Security mitigations for return-oriented programming attacks. https://www.kryptoslogic.com/download/ROP_Whitepaper.pdf. Acesso em junho/2016.

Bletsch, T., Jiang, X., and Freeh, V. (2011a). Mitigating code-reuse attacks with control-flow locking. In *Proc. of the 27th Annual Computer Security Applications Conf.*, ACSAC '11, pages 353–362, New York, NY, USA. ACM.

Bletsch, T., Jiang, X., Freeh, V. W., and Liang, Z. (2011b). Jump-oriented programming: A new class of code-reuse attack. In *Proc. of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 30–40.

Carlini, N. and Wagner, D. (2014). Rop is still dangerous: Breaking modern defenses. In *Proc. of the 23rd USENIX Conf. on Security Symposium*, SEC'14, pages 385–399.

Chen, P., Xiao, H., Shen, X., Yin, X., Mao, B., and Xie, L. (2009). Drop: Detecting return-oriented programming malicious code. In *Proc. of the 5th International Conf. on Information Systems Security*, ICISS '09, pages 163–177, Berlin, Heidelberg. Springer-Verlag.

Cheng, Y., Zhou, Z., Miao, Y., Ding, X., DENG, H., et al. (2014). Ropecker: A generic and practical approach for defending against rop attack. *Network and Distributed System Security Symposium*.

CVE (2011). Cve-2011-0065. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-0065. Acessado em junho/2016.

Davi, L., Sadeghi, A.-R., and Winandy, M. (2009). Dynamic integrity measurement and attestation: Towards defense against return-oriented programming attacks. In *Proc. of the 2009 ACM Workshop on Scalable Trusted Computing*, STC '09, pages 49–54, New York, NY, USA. ACM.

Ferreira, M. T., Filho, A. S., and Feitosa, E. (2014). Controlando a frequência de desvios indiretos para bloquear ataques rop. *Anais do SBSEG 2014*.

Fratric, I. (2012). Runtime prevention of return-oriented programming attacks. https://github.com/ivanfratric/ropguard/blob/master/doc/ropguard.pdf. Acessado em junho/2016.

Göktaş, E., Athanasopoulos, E., Polychronakis, M., Bos, H., and Portokalidis, G. (2014). Size does matter: Why using gadget-chain length to prevent code-reuse attacks is hard. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 417–432, San Diego, CA. USENIX Association.

Graziano, M., Balzarotti, D., and Zidouemba, A. (2016). ROPMEMU: A framework for the analysis of complex code-reuse attacks. In *ASIACCS 2016, 11th ACM Asia Conference on Computer and Communications Security, May 30-June 3, 2016, Xi'ian, China*.

Hiser, J., Nguyen-Tuong, A., Co, M., Hall, M., and Davidson, J. W. (2012). Ilr: Where'd my gadgets go? In *Proc. of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 571–585, Washington, DC, USA. IEEE Computer Society.

Intel (2013). Introduction to intel® memory protection extensions. https://software.intel.com/en-us/articles/introduction-to-intel-memory-protection-extensions. Acessado em junho/2016.

Jiang, J., Jia, X., Feng, D., Zhang, S., and Liu, P. (2011). Hypercrop: A hypervisor-based countermeasure for return oriented programming. In *Proc. of the 13th International Conf. on Information and Communications Security*, ICICS'11, pages 360–373, Berlin, Heidelberg. Springer-Verlag.

Kompalli, S. (2014). Using existing hardware services for malware detection. In *Security and Privacy Workshops (SPW), 2014 IEEE*, pages 204–208.

Lan, B., Li, Y., Sun, H., Su, C., Liu, Y., and Zeng, Q. (2015). Loop-oriented programming: A new code reuse attack to bypass modern defenses. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 1, pages 190–197.

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. (2005). Pin: Building customized program analysis tools with dynamic instrumentation. In

Proc. of the 2005 ACM SIGPLAN Conf. on Programming Language Design and Implementation, PLDI '05, pages 190–200, New York, NY, USA. ACM.

Microsoft (2013). Introducing Enhanced Mitigation Experience Toolkit (EMET) 4.1. `http://blogs.technet.com/b/srd/archive/2013/11/12/introducing-enhanced-mitigation-experience-toolkit-emet-4-1.aspx`. Acessado em junho/2016.

Onarlioglu, K., Bilge, L., Lanzi, A., Balzarotti, D., and Kirda, E. (2010). G-free: Defeating return-oriented programming through gadget-less binaries. In *Proc. of the 26th Annual Computer Security Applications Conf.*, ACSAC '10, pages 49–58, New York, NY, USA. ACM.

Pappas, V., Polychronakis, M., and Keromytis, A. D. (2012). Smashing the gadgets: Hindering return-oriented programming using in-place code randomization. In *Proc. of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 601–615.

Pappas, V., Polychronakis, M., and Keromytis, A. D. (2013). Transparent rop exploit mitigation using indirect branch tracing. In *Proc. of the 22Nd USENIX Conf. on Security*, SEC'13, pages 447–462, Berkeley, CA, USA. USENIX Association.

Quinn, R. (2012). *Detection of malware via side channel information*. PhD thesis, Binghamton University.

Schuster, F., Tendyck, T., Pewny, J., Maaß, A., Steegmanns, M., Contag, M., and Holz, T. (2014). Evaluating the effectiveness of current anti-rop defenses. In *Research in Attacks, Intrusions and Defenses: 17th International Symposium*, RAID 2014, pages 88–108.

Shapiro, R., Bratus, S., and Smith, S. W. (2013). "weird machines"in elf: A spotlight on the underappreciated metadata. In *Proc. of the 7th USENIX Conf. on Offensive Technologies*, WOOT'13, pages 11–11.

Son, N. H. (2011). Rop chain for windows 8. `http://security.bkav.com/home/-/blogs/rop-chain-for-windows-8/normal`. Acessado em junho/2016.

Vanegue, J. (2014). The weird machines in proof-carrying code. In *Security and Privacy Workshops (SPW), 2014 IEEE*, pages 209–213.

Wartell, R., Mohan, V., Hamlen, K. W., and Lin, Z. (2012). Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code. In *Proc. of the 2012 ACM Conf. on Computer and Communications Security*, CCS '12, pages 157–168.

Yuan, L., Xing, W., Chen, H., and Zang, B. (2011). Security breaches as pmu deviation: Detecting and identifying security attacks using performance counters. In *Proc. of the Second Asia-Pacific Workshop on Systems*, APSys '11, pages 6:1–6:5, New York, NY, USA. ACM.

Zhang, C., Wei, T., Chen, Z., Duan, L., Szekeres, L., McCamant, S., Song, D., and Zou, W. (2013). Practical control flow integrity and randomization for binary executables. In *Proc. of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 559–573, Washington, DC, USA. IEEE Computer Society.

An Architecture for Self-adaptive Distributed Firewall

Edmilson P. da Costa Júnior[1], Silas T. Medeiros[2], Carlos Eduardo da Silva[1], Marcos Madruga[2]

[1]Digital Metropolis Institute
Federal University of Rio Grande do Norte (UFRN)
Natal – RN – Brazil

[2]Departament of Informatics and Applied Mathematics
Federal University of Rio Grande do Norte (UFRN)
Natal – RN – Brazil

edmilsonjnior@info.ufrn.br,silastiagoo@gmail.com

kaduardo@imd.ufrn.br,marcos@dimap.ufrn.br

Abstract. *The notion of secure perimeter given by border firewalls ignores the possibility of attacks originating from inside the network. Although distributed firewalls allow the protection of individual hosts, the provided services might still be susceptible to attacks, as firewalls usually do not analyze application protocols. In this way, software vulnerabilities may be exploited until the problem has been fixed. From vulnerability discovery to the application of patches there is an exposure window that should be reduced. In this context, this paper presents an architecture for a distributed firewall system, in which a Vulnerability Assessment System is integrated for providing a self-adaptive mechanism capable of detecting vulnerabilities and executing actions to reduce exposure, contributing to mitigate the risk of vulnerability exploitation.*

1. Introduction

Several institutions nowadays, such as universities all over the world, deal with complex network infrastructure, involving an increasingly number of equipments (e.g., switches, routers) and servers, usually providing different services. Considering the diversity of activities and different research topics conducted throughout an university, it is common to find situations where several services, and servers, need to be provided for different groups of people, and more often than not, maintained by these different groups. This leads to an inconsistency in management and security procedures, where servers poorly configured and/or with outdated services, become potential targets for known vulnerabilities.

In this context, the traditional approach for network security, in which firewalls are deployed on the border of the network is no longer effective. Also known as centralized firewalls, these components are placed between the internal network of an institution and the Internet, with the objective of filtering the network traffic that goes in and out of the institution, limiting the services that would be exposed to the Internet, and defining the notion of a security perimeter. Clearly, this centralized model is not able to deal with attacks originated from inside the security perimeter [Ioannidis et al. 2000].

Today's technology movements, such as Bring Your Own Device (BYOD) and the availability of 3G/4G connections, mean that a malicious user has already penetrated the

border defenses. This is exacerbated when we consider university environments, which is usually open to the public in general, and contains a number of servers maintained by researchers, with outdated and potentially vulnerable services. Once one of those vulnerable servers is compromised, the attacker is free to search the network for other vulnerable servers, even though those servers are not exposed to the Internet. The bottom line is that, once an attacker compromises one of those vulnerable servers, he/she is inside the network, and the use of border firewall will contribute nothing in deterring his/her actions.

A solution for such scenario is the use of distributed firewall [Bellovin 1999]. A distributed firewall solution increments security by including firewalls in different points of the network and servers, besides the traditional border firewall. In this way, it is possible to control what services running on those servers are exposed on the network, and only for specific client hosts. However, the application of distributed firewall also brings some challenges, such as the management of these firewalls and their rules, and the response time in case of an incident. Traditional solutions for intrusion detection usually notify an administrator, which then assess and decides how to respond for rectifying and/or containing the situation [Meng et al. 2015]. However, this approach is usually not fast enough for avoiding service unavailability, information theft, or the infection of new systems/servers, mainly because attackers usually conduct their activities during strategic times, such as the middle of the night or weekends.

In this context, the contribution of this paper is an architecture for network security based on self-adaptive concepts. The motivation for using self-adaptation is the proven effectiveness and efficiency of self-adaptation in dealing with uncertainty in a wide range of applications, including those related to security [Bailey et al. 2014, Pasquale et al. 2012, Yuan et al. 2014]. The Self-Adaptive Distributed Firewall (SADF) architecture is based on the cooperation of different components usually found in a network infrastructure, such Vulnerability Assessment Systems and Configuration Management Systems. More specifically, we demonstrate how this architecture can be applied for managing a distributed firewall, in which possible threats can be detected (i.e., servers with vulnerabilities) and appropriate decisions be made for mitigating their impacts.

The remain of this paper is organized as follows: Section 2 contextualizes our work defining its scope and presenting some background on self-protection. Section 3 presents a conceptual view of the SADF architecture. Section 4 describes a prototype that has been implemented to demonstrate our approach feasibility. Section 5 discuss some related work. Section 6 concludes the paper.

2. Contextualization

As previously mentioned, due to the limitations of a centralized model, the border firewall does not protect equipment against internal attacks. This motivated the definition of a distributed firewall model [Bellovin 1999]. In a distributed firewall, security policies are defined in a centralized fashion using an specific language, and then distributed, by secure means, to be applied into different enforcement points. These enforcers can either be located on different segregation points inside the network, such as routers and switches, or on each host of the network [Ioannidis et al. 2000]. Figure 1 presents a general view of a network infrastructure where we can identify a Rules Management Server,

which is responsible for dealing with and distributing the firewall rules into the different enforcement points, in this case, firewalls in different servers of the network.

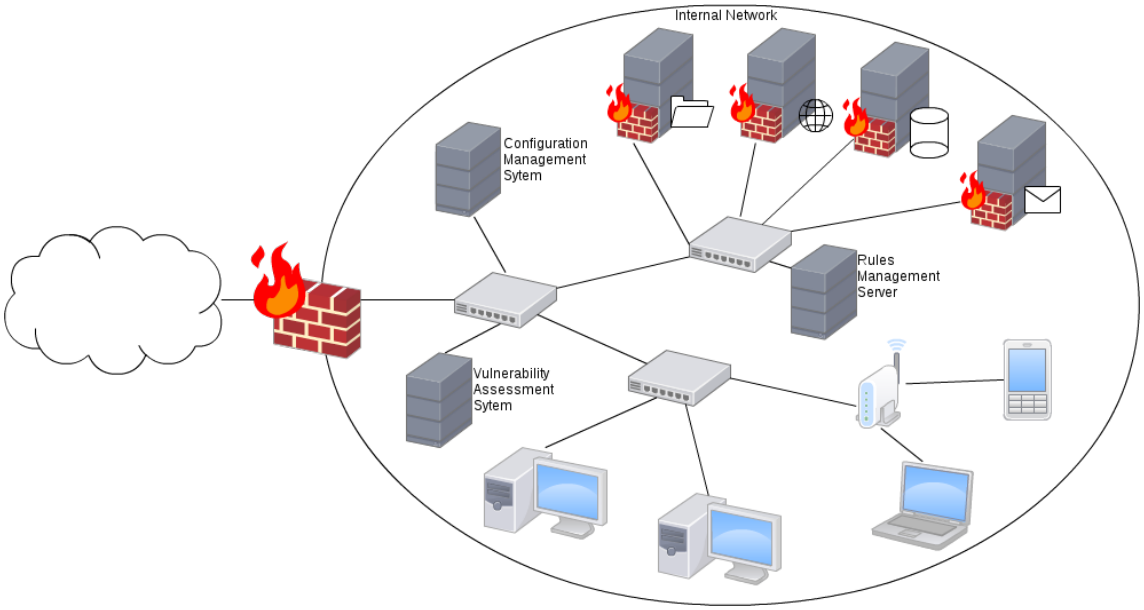


Figure 1: General view of the distributed firewall scope considered in this paper.

Operating and maintaining this infrastructure required a continuous effort as, even though there are different tools for facilitating management and maintenance actions, it is common to find out that most of these operations are still manually conducted. For example, as shown in Figure 1, the majority of institutions use, apart from firewalls, some sort of tool for configuration management, resource and service monitoring, and vulnerability assessment systems, which scans the network pointing out vulnerable services. These are important tools for maintaining the network infrastructure, but there is a lack of integration among them, requiring human intervention for conducting some tasks, and wasting valuable time between the moment an incident is detected and an administrator performs some corrective action to mitigate its impact.

A self-adaptive software system is able to modify its own structure and/or behaviour during run-time in order to deal with changes in its requirements, the environment in which it is deployed, or the system itself [Cheng et al. 2009]. Among the different properties of a self-adaptive system, self-protection has been identified as a key concept for building autonomous self-managed systems. While systems' architectures are becoming more dynamic and adaptive, the majority of the protection mechanisms have kept simple, with security policies usually manually defined, in a slow and costly way.

One way for achieving self-adaptation is through the Monitor-Analyse-Plan-Execute-Knowledge (MAPE-K) feedback control loop over a target system [Kephart and Chess 2003]. In this way, a self-protection mechanism allows the protected system to monitor and analyze its resources in order to detect possible problems, being able to react accordingly to deal with the detected problem. This reaction depends on the type of incident and the type of system being protected, and can range from emergency system shutdown, deactivation of damaged module and replacement for a new instance, user and/or connection blocking, etc [Yuan et al. 2014].

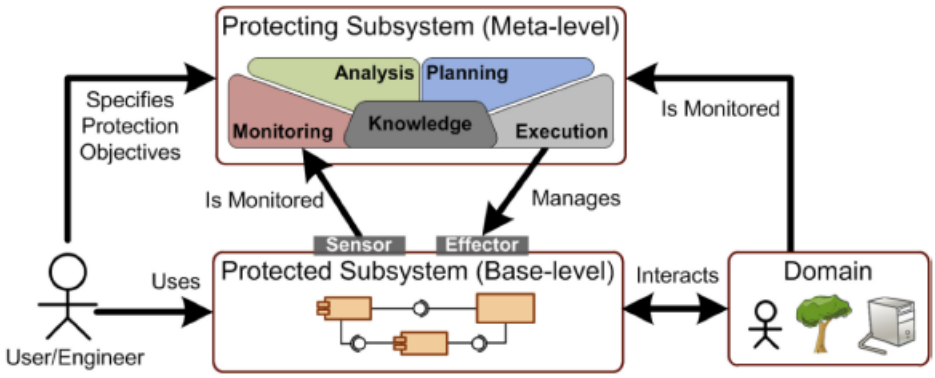


Figure 2: Self-Protection Reference Architecture [Yuan et al. 2014].

Figure 2 presents a reference architecture for a system that implements self-protection. At a meta-level we have a *protecting sub-system*, responsible for implementing the MAPE-K feedback control loop that protects the *protected sub-system* at the base level. The *protected sub-system* contains the system functionality associated with the main application logic, and may incorporate different security mechanisms, such as access control and cryptography. The meta-level subsystem is responsible for detecting security related incidents and for the decision making associated with the use of the security mechanisms by the base-level [Yuan et al. 2014]. The base-level sub-system runs over, and interacts with, a domain, which can also be monitored for helping in the decision making of the MAPE-K at the meta-level.

In this way, a Self-Adaptive Distribute Firewall (SADF) solution can be employed as a preventive mechanism for dealing with new vulnerabilities. For example, whenever a particular server contains a vulnerability with a score greater than a pre-defined value, the firewall could be configured to only allow access to server from clients in the same network.

3. Architecture for Self-Adaptive Distributed Firewall

Our solution for a Self-Adaptive Distributed Firewall (SADF) is built on top of the MAPE-K reference model as the means for logically structuring the different tasks involved in the management of the security aspects for a network infrastructure, and for integrating the different tools usually involved in those tasks, allowing for their automation. Figure 3 presents our architecture.

Each phase of the MAPE-K feedback control loop is implemented by an engine, which encapsulates the concrete components that allow for each engine functionality. To perform self-adaptation, the Monitor, Analyze, Plan and Execute engine components use different models that provide an abstraction of relevant aspects of the managed system, its environment, and the self-adaptation goals [Iglesia and Weyns 2015]. These models are maintained by a knowledge base (not represented in the Figure).

The *Monitoring engine* is responsible for collecting information about the different servers of the network infrastructure. This collection happens through *Sensor* interfaces in each server. This data is represented by a *Server description* model, which is a format that can be manipulated and reasoned upon by the components of SADF. A service

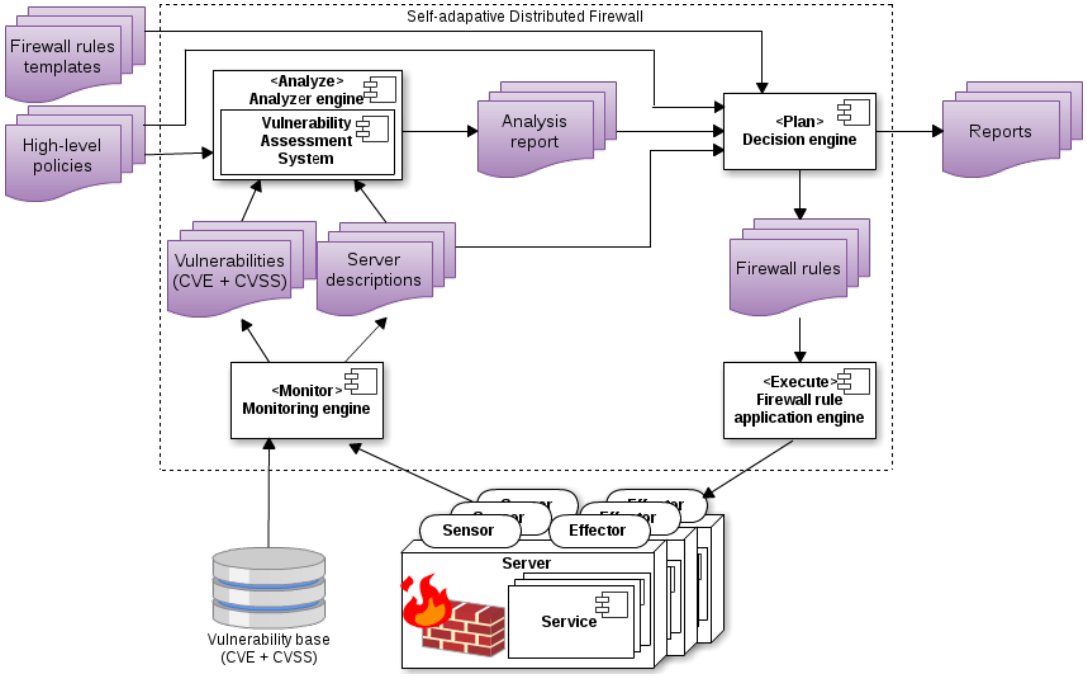


Figure 3: Conceptual architecture of proposed solution.

description model contains, among other information, details about operating system, IP Address, services' names, versions and network port. Furthermore, it captures the firewall rules currently in effect on the server. The *Monitoring engine* is also responsible for obtaining *Vulnerabilities* from an external *Vulnerability base*. Vulnerabilities are represented through CVE[1], which defines a dictionary and standard representation format for vulnerabilities descriptions. These descriptions are published through the CVE List and maintained by different vulnerabilities databases (e.g., the NVD[2]). Vulnerabilities have an associated severity score calculated based on the CVSS[3], which defines metrics and formulas for deriving a vulnerability score, and a standard format representation.

The *Analyzer engine* relies on a Vulnerability Assessment System (VAS) to search for known vulnerabilities on the services currently running on the network. A VAS works by scanning the network and conducting different tests in order to find vulnerabilities in systems and servers, producing a vulnerability report for each server. Based on the server descriptions, the VAS can be employed with higher priority to scan known services running on each server, usually when there are changes in the server descriptions or new vulnerabilities have been published. In the meantime, full vulnerability analysis of servers can still be performed. A *High-level policy* captures the requirements of the administrator, and together with the VAS report and the server description, is used for detection of policy violations. For example, servers with a vulnerability score greater than a particular threshold should only be accessible from machines in the same network, but the current firewall rules allow access from anywhere. All these data is used by the *Analyzer engine* component for producing *Analysis report*, which indicates for example, servers with known vulnerabilities.

[1]Common Vulnerabilities and Exposures - `https://cve.mitre.org`
[2]The National Vulnerability Database maintained by NIST - `https://nvd.nist.gov/`
[3]Common Vulnerability Scoring System - `https://www.first.org/cvss`

The *Decision engine* component is responsible for the plan phase of the MAPE-K loop. This component is responsible for making decisions on how to respond to the encountered situation based on the analysis report, the server descriptions, the high-level policies and a set of *Firewall rules templates*. These templates provide a sort of parameterized firewall rules for different services, which can then be employed by the Decision engine for defining specific firewall rules to be applied. The creation of firewall rules must employ mechanisms for avoiding conflicts between rules. Besides creating firewall rules to be applied onto the servers of the network, the Decision engine also produces a report intended for a human administrator.

At the execute phase we have a *Firewall rule application engine* component, which is responsible for effecting the new firewall rules on the servers. This component must take in consideration mechanisms for guaranteeing secure communication with each server, and configuration management techniques.

4. Instantiating the SADF

The proposed SADF architecture has been instantiated into a prototype implementation using a combination of existing open source and in-house developed components. This instantiation has been used to build a case study in order to demonstrate the feasibility of our approach. In order to demonstrate the prototype, in this paper we consider the server to be protected a Web server running the Apache HTTPD software and the ssh daemon.

In this section we present details about the different representation models employed in our instantiation, followed by a description of the developed prototype. We conclude this section with a brief discussion on our approach.

4.1. Representation Models

One aspect that must be considered for a self-protection solution is the representation of the protected environment, such as servers, services, and firewall rules.

For representing servers and their deployed services we chose the representation language defined by the Puppet[4] configuration management tool. The Puppet language allows the description of servers, services, and configurations using a parameterized approach and well defined semantics. Puppet configures systems in two main stages compiling and applying a catalog. A catalog is a document that describes the desired system state. It lists all of the resources that need to be managed, as well as any dependencies between those resources. The core of the Puppet language is declaring resources. Groups of resources can be organized into classes, which are larger units of configuration. While a resource may describe a single file or package, a class may describe everything needed to configure an entire service or application. The Figure 4a show a simple example of representation of a server and some services. One class was created with name *foo*. This class has an attribute *host* to describe a hostname and IP address. Furthermore, httpd and sshd were describe using the attribute *service*. The *ensure* attribute is used to indicate that those services should be in a running state.

In a similar way, it is necessary to represent firewall rules in a format that can be reasoned upon. For this purpose, we decided to employ the FLIP language

[4]`https://puppet.com/`


```
class foo {
  host { 'foo':
    host_aliases => foo.domain,
    ip           => 200.177.2.46,
  }
  service {'httpd':
    ensure     => running,
  }
  service {'sshd':
    ensure     => running,
  }
}
```

(a) Puppet class

```
domain serverHTTP = [192.168.100.35],
       serverVAS = [10.5.54.10],
service http = tcp.[port=80],
        ssh = tcp.[port=22, port=2222],
policy_group serverHTTP_policy{
  incoming:
    ssh {allow *}
    http {allow *}
}
policy_group serverHTTP_with_vulnerability{
  incoming:
    ssh {allow *}
    http {deny * except serverVAS}
}
apply serverHTTP_policy on serverHTTP;
```

(b) FLIP rule

Figure 4: Example of descriptions: (a) Class description using the Puppet language. (b) Firewall rules using the FLIP language.

[Zhang et al. 2007][Al-Shaer 2014]. In FLIP, firewall rules are defined using a high-level language that can be automatically translated into device specific format. FLIP provides as well defined language with formal semantics, together with proven sound and complete algorithms for conflict resolution and translation into device specific firewall rules. Its formalism was one of the main reasons for choosing FLIP. The Figure 4b presents an example of a rule in FLIP. The first block defines the domains that can be networks or hosts. We define a HTTP server and a server that would be our VAS. The second block of FLIP defines services, a service may have one or more ports. In this example http and ssh was specified. Then defines the group policy which specifies the behavior that will be taken in relation to services in a given scenario. Two groups were created, one that allows access to services and another that blocks the http. Finally, is necessary to make a connection between the group and the protected domain.

4.2. Prototype Implementation

In the sequence we describe our prototype implementation, whose concrete architecture is presented in Figure 5.

As previously mentioned, we employ the Puppet configuration management language for describing servers' configurations. Puppet provides tools for applying configurations, and for obtaining the current status of a host. A Puppet agent component runs on each host, and reports to (and receive commands from) the Puppet master component, which stores servers description into the Puppet catalog. In this way, Puppet agents fulfill the roles of sensor and effector of servers, while the Puppet master is responsible for the monitor and execute phases of the MAPE-K. It is important to mention that the definition of puppet description files for servers is out of the scope of this paper. We assume that this is achieved by a third party (e.g., a member of the network administration team), while our focus is on defining and applying firewall rules for the established configuration, or for dealing (i.e., mitigating the impact) with known vulnerabilities that the server may be susceptible to. In this way, the puppet agent and master components are used to obtain the current service(s) configuration on each server.

Once we know the services running on each server of the network, we can then

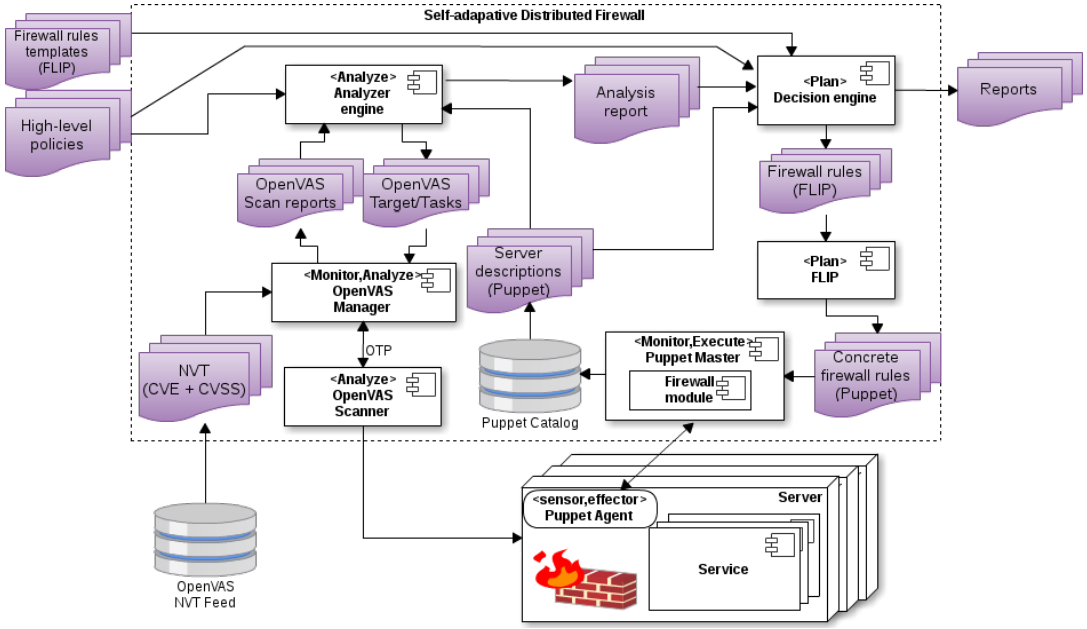


Figure 5: Architectural view of prototype implementation.

employ vulnerability assessment as triggers for adaptation. In order to achieve this, we employ the OpenVAS[5] Vulnerability Assessment System (VAS). OpenVAS provides an open-source solution for the identification, quantification and prioritization of vulnerabilities, and we explore this for our *Analyzer engine* component. The *OpenVAS Scanner* is responsible for conducting Network Vulnerability Tests (NVTs) on the hosts of the network. An NVT can be defined as a test script, together with CVE description, and CVSS score. NVTs can be obtained from the OpenVAS NVT Feed by the *OpenVAS Manager* (acting in both monitor and analyze roles), and are developed based on the CVE and CVSS score obtained from the NIST Network Vulnerability Database (NVD). The *OpenVAS Manager* is responsible for managing the OpenVAS Scanner, providing its input and processing its output. The interaction between the Manager and the Scanner happens via the OpenVAS Transfer Protocol (OTP), which provides the means for controlling scan execution. The Manager provides the OpenVAS Management Protocol (OMP), a XML-based stateless API that can be used to interact and control the OpenVAS Manager. The Analyzer engine employs the Servers' description for driving the OpenVAS scans. The interaction with OpenVAS is achieved by the OMP protocol, which receives as input an XML description of the target host and the scanning tasks (i.e., the vulnerability tests) to conduct on the host. The target can be defined as one, or a group of hosts, with an optional list of ports. A task corresponds to scanner configurations, and defines how the environment will be checked.

We detail the interaction between the *Analyzer engine* component and the OpenVAS in Figure 6, using an UML sequence diagram. Once the servers' descriptions have been obtained, the *Analyzer engine* create the inputs for OpenVAS Manager (calls 2 and 3 in the sequence diagram) and starts a scan, receiving the scan ID as response. The scan ID is then used to obtain the scan results and, in case a vulnerability has been found, the

[5]`http://www.openvas.org/`

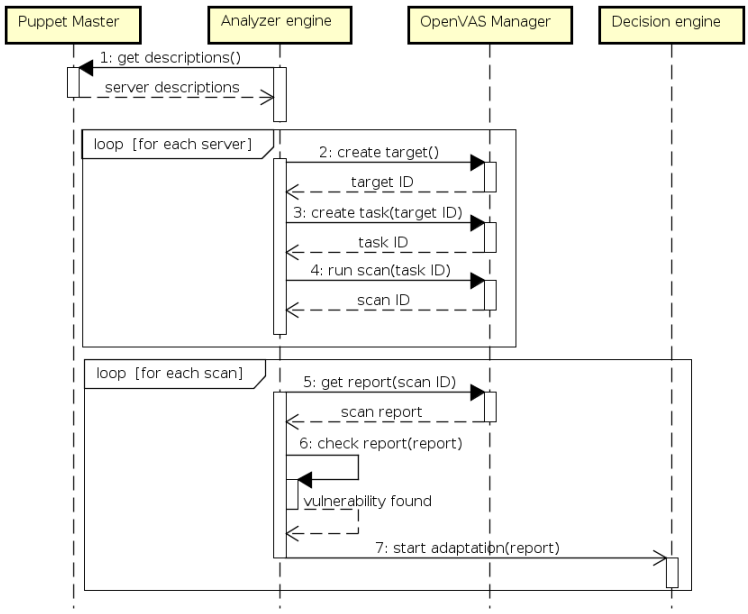


Figure 6: UML sequence diagram showing the behaviour of analysis.

Decision engine component is activated to deal with it.

Following our case study, the OpenVAS found a vulnerability with CVSS score 7.8 in HTTP service that allows remote attackers to cause a denial of service (memory and CPU consumption). An extract of the XML report is presented in Figure 7.

```
...
<port>80/tcp
  <host>10.3.128.20</host>
  <severity>7.8</severity>
  <threat>High</threat>
</port>
<nvt oid="1.3.6.1.4.1.25623.1.0.901203">
  <name>
    Apache httpd Web Server Range Header Denial of Service Vulnerability
  </name>
  <family>Denial of Service</family>
  <cvss_base>7.8</cvss_base>
  <cve>CVE-2011-3192</cve>
  <bid>49303</bid>
  ...
```

Figure 7: Extract of OpenVAS report to a vulnerable HTTP server.

Once the *Decision engine* has been activated, it must decide on how to respond to the found vulnerability. The behavior of this component is presented in Figure 8. Each report is considered together with the actual server description, including the firewall rules currently in place, against the high-level policy for deciding what to do regarding the firewall of the affected host. For example, a CVSS score above 7 might require that the service is no longer accessible. If this is the case, the *Decision engine* define the adequate firewall rules using the FLIP language, and sends it to the FLIP tool, which is responsible for checking the rule is conflict free, translating it into the puppet format, and applying them in the puppet master. Regardless of the decision, the *Decision engine* sends a notification report (represented by the *notify administrator* call). This notification can then be

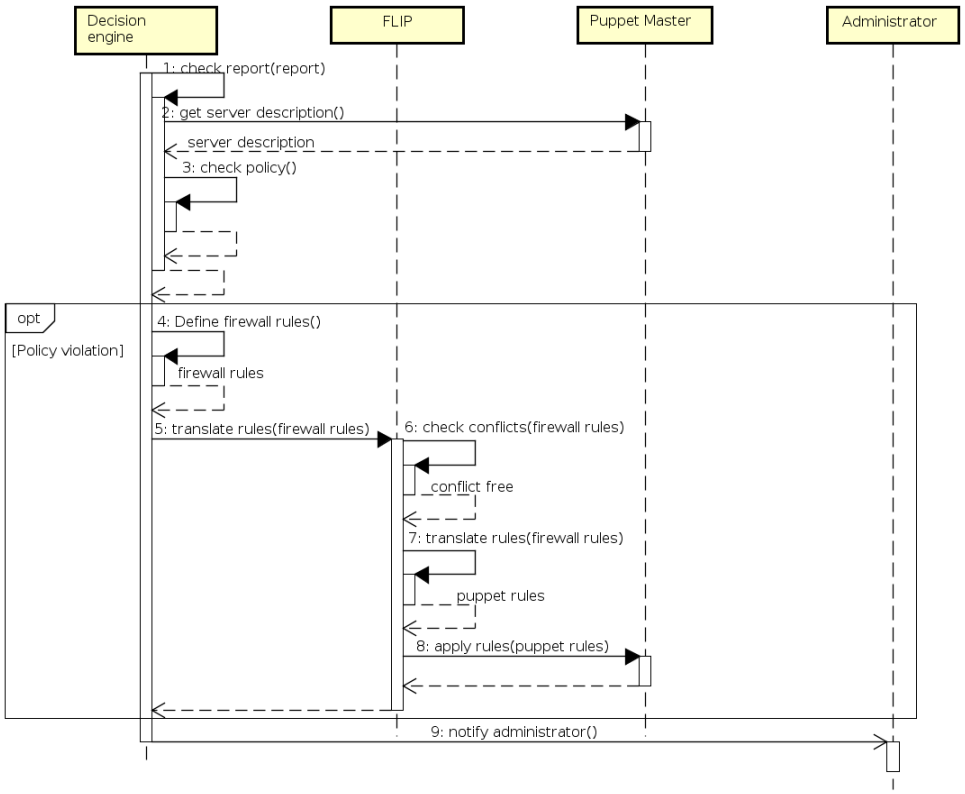


Figure 8: Sequence diagram showing behaviour of decision engine.

used by a visualization tool, for example, a Web page listing all active vulnerabilities and the respective firewall rules applied to those servers.

The Puppet master, together with its firewall module[6], manage and configure firewall rules from within the Puppet DSL. This module offers support for iptables and ip6tables, which is the most common firewall in GNU/Linux distributions, being capable of acting on a running firewall. All rules employ a numbering system in the resource's title that is used for ordering. The Figure 9a presents an example of a puppet rule for allowing http and ssh services, while Figure 9b presents the rule created by our prototype for closing access to the http service.

Once the firewall rules are in place, new alerts of the same vulnerability will be notified to administrator, but will not trigger changes in the firewall of the affected host. On the other hand, once the vulnerability has been fixed, our tool will detect it, and adjust the firewall rules accordingly.

4.3. Discussion

The main objective of this prototype was to demonstrate the feasibility of our approach in integrating a VAS and a distributed firewall in an autonomic way. For this reason, the decision making performed by our prototype has been implemented through simple if-then-else statements using the fields of the different descriptions as parameters. We have also employed very rudimentary high-level policies, simply defining range thresholds for each response (e.g., access granted to all, access granted to clients in the same network,

[6]`https://forge.puppet.com/puppetlabs/firewall`


```
class my_fw::pre {
  Firewall {
    require => undef,
  }
  firewall { '000 Allow inbound HTTP':
    dport    => [80,443],
    proto    => tcp,
    action   => accept,
  }
  firewall { '001 Allow inbound SSH':
    dport    => 22,
    proto    => tcp,
    action   => accept,
  }
}
```

(a) Allowing http and ssh

```
class my_fw::pre {
  Firewall {
    require => undef,
  }
  firewall { '000 Deny inbound HTTP':
    dport    => [80,443],
    proto    => tcp,
    action   => deny,
  }
  firewall { '001 Allow inbound SSH':
    dport    => 22,
    proto    => tcp,
    action   => accept,
  }
}
```

(b) Denying http and allowing ssh

Figure 9: Examples of firewall rules in the Puppet language.

access denied to all). The experiments conducted, although initial, have showed that we are indeed able to activate firewalls running on each server for preventing access to vulnerable services.

We believe the administrator involvement is crucial to those types of solutions, and the administrator notification aims to be a starting point for a more intelligent human interaction. For example, in our prototype we have applied the new firewall rules into the affected server without any kind of confirmation, but we also envision a scenario where the rules as presented to an administrator that then authorizes their application, or the existence of policies separating scenarios that should be automatically acted upon from those where some sort of human confirmation is needed. Our solution can also be used to identify new servers on the network that are not managed by the configuration management tool in place (e.g., Puppet). Those would be part of the notification reports sent to administrators.

The Puppet configuration management tool presents a pull based model, in which agents query the master at pre-determined intervals in search of new commands (usually, 30 minutes). This might be considered an issue when responding on-the-fly to detected situations, which is not the case in this paper. We are focused on preventing the exploit of known vulnerabilities before they happen, opposed to traditional IDPS systems, which focus on responding in real-time to incidents. In this way, we consider that 30 minutes is a reasonable time for applying firewall rules blocking vulnerable services, although this time could be reduced, and there are alternatives for employing a push-model to puppet. Regarding the secure communication between the SADF and protected servers, we employ the certificate based security provided by the Puppet tool, which takes care of authentication and secure transit between the Puppet master and its agents.

5. Related Work

The discussion on centralized and distributed firewalls is well established in the literature [Bellovin 1999] [Stallings 2010]. One of its first implementation proposal has been presented by Ioannidis [Ioannidis et al. 2000], in which kernel extensions have been developed for the OpenBSD distribution, together with a policy definition language (denominated KeyNote) and use of IPsec for secure traffic amongst the hosts of the network.

More recently [Lai et al. 2009] introduces a distributed firewall system for Linux platform that works upon Iptables/Netfilter for IPv6 networks with IPsec support. In order to improve performance for handle the additional costs of encryption of packages with IPsec, a distributed firewall architecture was proposed.

Autonomic computing and self-protection has been gaining traction as the means for dealing with new security challenges and systems, in which static and rigid security practices are not enough to deal with security threats that need to be detected and mitigated at runtime [Yuan et al. 2014]. In this context, Yuan et al. [Yuan et al. 2014] have done an extensive systematic survey of the state of the art on self-protecting software, identifying trends, patterns, and gaps. Some works focus on adapting authorization policies, such as the Self-Adaptive Authorisation Framework (SAAF) [Bailey et al. 2014] that focus on adapting access control policies on the PERMIS system [Chadwick et al. 2008], and SecuriTAS [Pasquale et al. 2012], a tool that enables dynamic decisions in awarding physical access, based on a perceived state of the system and its environment.

Several researchers focus on Intrusion Detection Systems. For example, Uribe and Cheung [Uribe and Cheung 2004] have looked into the integration between IDSs and firewalls, proposing an approach for optimizing IDS configuration by only analysing traffic that is not considered by the firewalls' rules. Zhang and Shen [Zhang and Shen 2009] employ a statistical learning based approach in order to reduce false-positives on IDSs. These works are concentrated on improving the IDS.

Few works consider the analysis of vulnerabilities for making a decision at the network level (i.e., firewall rules). Debar et al. [Debar et al. 2007] present formalisms for the definition of security policies that can be dynamically modified in response to detected threats. The formalism presented in their paper is at an abstract level, and may consider vulnerability analysis in the threat detection process. Compared to our work, their approach can be considered as complementary, providing a series of formalism that could improve the robustness of our approach regarding the definition of high-level policies.

One aspect that must be considered for a self-protection solution is the representation of the protected environment, such as servers, services, and firewall rules. We present here some of the options found in the literature during our searches. The Network Markup Language (NML) [van der Ham et al. 2013] is a generic model defined by the OpenGrid Forum (OGF)[7] as a standard for modelling networks, such as switches and links, which is out of the scope considered in this paper. Regarding the representation of firewall rules, apart from the FLIP language (already presented), there is also the AFPL2 [Pozo et al. 2009] (Abstract Firewall Policy Language 2), a domain-specific language that provides an XML Schema for the definition of firewall rules independent of firewall product. Although its support of NAT rules, AFPL2 has not evolved as FLIP, and does not provide conflict resolution of firewall rules. The Distributed Management Task Force (DMTF) proposed the Common Information Model (CIM), an specification aimed at allowing the interoperation of management information. The CIM also provides an extensible XML model and, although it has been employed by different vendors, its extension for Network Policy Management [DMTF 2016] is still considered work-in-progress, and may be subject to changes.

[7] https://www.ogf.org/

6. Conclusions & Future Work

This paper presented an approach for Self-Adaptive Distributed Firewall (SADF). We presented an architecture built on top of the MAPE-K reference model as the means for logically structuring the different tasks involved in the management of the security for the network. The *Analyze* and *Decision* engine are at the heart of the operation of our architecture for mitigating the risks of known vulnerabilities. Together, they detect vulnerabilities and protect services before they are exploited. We made a prototype implementation to demonstrate their feasibility using a combination of existing open source and in-house developed components.

Some simple experiments were conducted with encouraging preliminary results, where we are able to dynamically modify the firewalls on protected servers. The VAS in Analyze engine detect vulnerabilities caused by software bugs or misconfiguration. However, since we are at a prototype phase, our approach presents some limitations. For instance, currently we only deal with each individual host in an independent way, and more interesting possibilities arise due to the nature of distributed firewalls, in which firewall could be employed on other points of the network, such as routers and switches.

The decision making and the definition of high-level policies have been simplified, and could be improved by using other specialized components, such as rule-based systems. Some improvements on Decision engine may add capabilities to deal with a lot of information and possibilities in order to enhance the security of network. For example, a host firewall might redirect all incoming connection to an application level proxy when a particular vulnerability has been detected, adding an extra layer of authentication while the vulnerability has not been fixed. Another future work involves the integration of our solution with traditional IDPSs, allowing it to react to attack exploiting zero-day vulnerabilities.

References

Al-Shaer, E. (2014). *Automated Firewall Analytics: Design, Configuration and Optimization*, chapter Specification and Refinement of a Conflict-Free Distributed Firewall Configuration Language, pages 49 – 74. Springer International Publishing.

Bailey, C., Chadwick, D. W., and de Lemos, R. (2014). Self-adaptive federated authorization infrastructures. *Journal of Computer and System Sciences*, 80(5):935 – 952.

Bellovin, S. M. (1999). Distributed firewalls. *;login:*, pages 39–47.

Chadwick, D. W. et al. (2008). PERMIS: A Modular Authorization Infrastructure. *Concurr. Comput. : Pract. Exper.*, 20(11):1341–1357.

Cheng, B. H. et al. (2009). Software Engineering for Self-Adaptive Systems: A Research Roadmap. In Cheng, B. H., de Lemos, R., Giese, H., Inverardi, P., and Magee, J., editors, *Software Engineering for Self-Adaptive Systems*, pages 1–26. Springer-Verlag, Berlin, Heidelberg.

Debar, H., Thomas, Y., Cuppens, F., and Cuppens-Boulahia, N. (2007). Enabling automated threat response through the use of a dynamic security policy. *Journal in Computer Virology*, 3(3):195 – 210.

DMTF (2016). Network policy management profile. `https://www.dmtf.org/sites/default/files/standards/documents/DSP1048_1.0.0c_0.pdf`. [Online; accessed 19-June-2016].

Iglesia, D. G. D. L. and Weyns, D. (2015). Mape-k formal templates to rigorously design behaviors for self-adaptive systems. *ACM Trans. Auton. Adapt. Syst.*, 10(3):15:1–15:31.

Ioannidis, S., Keromytis, A. D., Bellovin, S. M., and Smith, J. M. (2000). Implementing a distributed firewall. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, CCS '00, pages 190–199, New York, NY, USA. ACM.

Kephart, J. O. and Chess, D. M. (2003). The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50.

Lai, Y., Jiang, G., Li, J., and Yang, Z. (2009). Design and implementation of distributed firewall system for ipv6. In *Communication Software and Networks, 2009. ICCSN '09. International Conference on*, pages 428–432.

Meng, G., Liu, Y., Zhang, J., Pokluda, A., and Boutaba, R. (2015). Collaborative security: A survey and taxonomy. *ACM Comput. Surv.*, 48(1):1:1–1:42.

Pasquale, L. et al. (2012). SecuriTAS: A Tool for Engineering Adaptive Security. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 19:1–19:4, New York, NY, USA. ACM.

Pozo, S., Varela-Vaca, A. J., and Gasca, R. M. (2009). Afpl2, an abstract language for firewall acls with nat support. In *Second International Conference on Dependability (DEPEND 2009)*, pages 52–59.

Stallings, W. (2010). *Network Security Essentials: Applications and Standards*. Prentice Hall, 4th edition.

Uribe, T. E. and Cheung, S. (2004). Automatic analysis of firewall and network intrusion detection system configurations. In *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering (FMSE 2004)*, pages 66 – 74.

van der Ham, J., Dijkstra, F., Łapacz, R., and Zurawski, J. (2013). Network markup language base schema version 1. Grid Final Draft (GFD), Proposed Recommendation (R-P) GFD-R-P.206, Open Grid Forum.

Yuan, E., Esfahani, N., and Malek, S. (2014). A systematic survey of self-protecting software systems. *ACM Trans. Auton. Adapt. Syst.*, 8(4):17:1–17:41.

Zhang, B., Al-Shaer, E., Jagadeesan, R., Riely, J., and Pitcher, C. (2007). Specifications of a high-level conflict-free firewall policy language for multi-domain networks. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, SACMAT '07, pages 185–194, New York, NY, USA. ACM.

Zhang, Z. and Shen, H. (2009). M-aid: An adaptive middleware built upon anomaly detectors for intrusion detection and rational response. *ACM Trans. Auton. Adapt. Syst.*, 4(4):24:1 – 24:35.

Sobre a Influência dos Nós Egoístas no Descarregamento de Tráfego com Redes Oportunistas

Claiton Luiz Soares[1,2], **Igor Monteiro Moraes**[1]

[1]Laboratório MídiaCom, PGC-TCC
Instituto de Computação - Universidade Federal Fluminense
Niterói, Rio de Janeiro, Brasil

[2]Instituto Federal do Triângulo Mineiro -IFTM
Paracatu, Minas Gerais, Brasil

{csoares,igor}@ic.uff.br

Abstract. *The cellular infrastructure is overloaded because of the increasing number of mobile devices and applications. A solution to tackle this problem is to offload the network traffic by using an alternative network as a secondary communication channel. Several studies in the literature propose to employ opportunistic networks as this secondary channel. To be effective, however, opportunistic communication relies that nodes cooperate each other in order to forward messages and to confirm to the infrastructure the successful reception of messages sent by other nodes. In this paper, we evaluate the impact of selfish nodes on the performance of the traffic offloading with opportunistic networks. Selfish are those nodes that do not forward messages to other nodes or do not send acknowledgments to the infrastructure or combine both behaviors. Results show the offloading efficiency decreases from 70% to up to 18% if selfish nodes are in the network.*

Resumo. *A infraestrutura celular está sobrecarregada com o crescente número de dispositivos e aplicações móveis. Uma solução para este problema é descarregar parte do tráfego da rede utilizando outra rede como canal de comunicação secundário. Muitos trabalhos na literatura propõem o uso de redes oportunistas como este canal secundário. Para se ter sucesso na comunicação oportunista, entretanto, os nós precisam cooperar uns com os outros para encaminhar mensagens e confirmar para a infraestrutura o recebimento das mensagens encaminhadas por outros nós. Neste artigo, avalia-se a influência dos nós egoístas no descarregamento de tráfego com redes oportunistas. Nós egoístas são aqueles que não encaminham mensagens para outros nós ou não enviam reconhecimentos positivos para infraestrutura ou que combinam ambos os comportamentos. Resultados mostram que a eficiência do descarregamento é reduzida de 70% para até 18% com a presença de nós egoístas na rede.*

1. Introdução

A crescente proliferação de dispositivos móveis juntamente com o rápido desenvolvimento de tecnologias de comunicação móvel está provocando um aumento nas solicitações de conteúdo da rede celular. Consequentemente, a crescente demanda de tráfego faz com que a infraestrutura de rede celular se aproxime do seu limite. Embora

as novas tecnologias de comunicação ofereçam mais largura de banda para os usuários, prevê-se que até mesmo redes celulares 4G não vão ser capazes de lidar com a enorme demanda de tráfego de dados dos usuários nos próximos anos [Li et al. 2014].

Uma possível solução para este problema é utilizar uma técnica conhecida como descarregamento de tráfego (*traffic offloading*) [Rebecchi et al. 2015, Han et al. 2010]. O objetivo dessa técnica não é substituir a rede móvel de dados, que seria o canal principal de comunicação, mas empregar um canal secundário de comunicação, no caso uma rede oportunista, para reduzir o volume de dados transportado na rede celular. Neste contexto, muitos usuários estão interessados em um mesmo conteúdo, premissa que é válida para muitos conteúdos disponíveis na Internet. Se esses conteúdos são recuperados usando somente o canal primário, várias cópias desse mesmo conteúdo, uma para cada usuário interessado, são encaminhadas pela infraestrutura de rede. Isso gera desperdício de recursos e pode reduzir a qualidade de experiência do usuário. Com as redes oportunistas, os nós não precisam recuperar exclusivamente um conteúdo da sua fonte. Ele pode ser recuperado também de outro nó da rede que anteriormente tenha solicitado esse mesmo conteúdo. A comunicação entre os nós é feita de forma *ad hoc* através de outras interfaces sem-fio cada vez mais comuns nos dispositivos móveis, como Wifi e *Bluetooth*. A ideia é se beneficiar da mobilidade dos nós utilizando uma rede secundária em troca de uma maior tolerância ao atraso que pode ser experimentada pelos usuários. Para incentivar a participação nas redes oportunistas, as operadoras podem oferecer descontos de preços para os assinantes de telefonia móvel em troca dos seus recursos de bateria e armazenamento para descarregar o tráfego [Zhuo et al. 2011].

A ideia das redes oportunistas é permitir que dois nós consigam se comunicar mesmo que nunca exista um caminho fim-a-fim entre eles. Para tanto, essas redes empregam, em geral, a arquitetura DTN (*Delay Tolerant Networking*) [Oliveira et al. 2007]. As DTNs encaminham as mensagens segundo o paradigma armazena-carrega-e-encaminha (*store-carry-and-forward*). Assim, se não existe um caminho fim-a-fim entre uma fonte e um destino num determinado momento, os nós podem armazenar e transportar os dados até encontrar futuramente outros nós. Quando o nó entra no raio de alcance de outro nó e tem uma oportunidade de encaminhamento, chamada de contato, todos os nós encontrados podem ser candidatos a receberem cópias das mensagens carregadas. Portanto, as redes oportunistas dependem da mobilidade e cooperação dos nós para proverem pelo menos caminhos esporádicos para que a informação possa ser devidamente entregue ao destino [Chaintreau et al. 2006].

Uma proposta que utiliza as redes oportunistas como um canal secundário para auxiliar a infraestrutura no descarregamento de tráfego é o *framework Push-and-Track* (PnT) [Whitbeck et al. 2011] . Com o PnT, a infraestrutura envia cópias da mensagem de interesse para um subconjunto de nós através do canal primário, ou seja, a rede celular 3G ou 4G. Em seguida, os nós que receberam uma dessas cópias começam a encaminhar a mensagem para outros nós através do canal secundário (Wi-Fi ou Bluetooth) à medida que entram em contato com esses nós. Ao receber a mensagem de forma oportunista, os nós enviam para a infraestrutura uma mensagem de reconhecimento positivo (ACK) e também passam a encaminhar a mensagem para outros nós. Essa realimentação permite que a infraestrutura mantenha o controle dos nós que receberam a mensagem e avalie a oportunidade de reenviar novas cópias da mensagem. Se a infraestrutura percebe que o en-

caminhamento da mensagem através da comunicação oportunista não irá atingir todos os interessados dentro de um intervalo de tempo predefinido, ela envia cópias da mensagem usando o canal primário para todos os interessados que ainda não receberam a mensagem. Esse mecanismo faz com que o *Push-and-Track* garanta a entrega das mensagens mesmo usando uma rede oportunista para descarregamento de tráfego.

O *Push-and-Track*, porém, assume que todos os nós têm interesse em encaminhar a mensagem para outros nós e confirmam para a infraestrutura as mensagens recebidas de outros nós. Na prática, tal fato pode não ser verdade em função de falhas de funcionamento dos nós ou comportamentos egoístas para prejudicar o desempenho da rede ou simplesmente para economizar recursos. Esse comportamento egoísta prejudicará a disseminação da mensagem entre os nós e pode provocar reenvios desnecessários da mensagem, uma vez que é através das mensagens de reconhecimento que a infraestrutura tem o controle de quantos e quais nós receberam a mensagem. Neste artigo, propõe-se avaliar a influência dos nós egoístas, que não encaminham mensagens para outros nós ou não enviam reconhecimentos positivos para infraestrutura ou que combinam ambos os comportamentos, no desempenho do PnT. Avaliam-se diferentes cenários por simulação, variando-se o número de nós egoístas e a função de disseminação das mensagens. Considera-se um modelo real de mobilidade e mede-se a carga de tráfego enviada pela rede oportunista e pela infraestrutura. Nota-se que o aumento dos nós egoístas reduz a eficiência do descarregamento de tráfego com o PnT de 70% para 54% quando 20% dos nós tem o comportamento egoísta combinado. Se 60% dos nós são egoístas, a eficiência é reduzida para 18%.

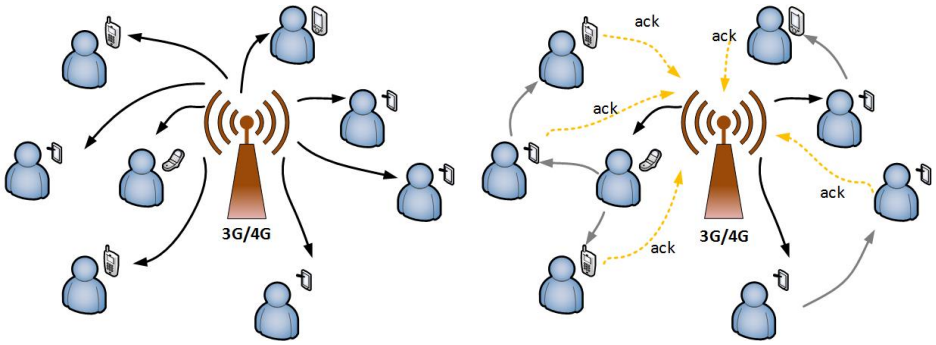
O restante do artigo está organizado da seguinte forma. A Seção 2 aborda a questão do descarregamento de tráfego com redes oportunistas e detalha o funcionamento do *Push-and-Track*, que é o mecanismo usado na avaliação deste artigo. A Seção 3 define os modelos de ataques empregados na avaliação neste artigo. A Seção 4 define os parâmetros de configuração e os cenários de avaliação utilizados nas simulações. A Seção 5 analisa e discute os resultados. A Seção 6 apresenta os trabalhos relacionados à presença de nós egoístas em redes oportunistas e ao descarregamento de tráfego. Por fim, a Seção 7 discute as conclusões e as direções futuras deste trabalho.

2. Descarregamento de Tráfego

A premissa de funcionamento dos mecanismos de descarregamento de tráfego é que vários usuários estão interessados no mesmo conteúdo. Uma situação em que essa premissa é válida é quando torcedores querem receber o mapa de assentos na chegada ao estádio ou na saída quando querem saber a localização das estações de metrô, pontos de táxi e ônibus para voltarem para suas casas. A ideia, portanto, é que parte deste tráfego composto por cópias do mesmo conteúdo seja entregue através de um canal secundário, diminuindo assim a carga de tráfego do canal primário da infraestrutura. Nesse contexto, diversos trabalhos de pesquisa estão sendo realizados usando redes oportunistas como canal secundário para descarregar parte do tráfego das redes 4G [Rebecchi et al. 2015].

Uma das propostas é o *framework Push-and-Track* (PnT) [Whitbeck et al. 2011, Whitbeck et al. 2012], cuja principal característica é garantir a entrega de todas as mensagens mesmo usando uma rede oportunista para aliviar a carga da infraestrutura de rede celular. A Figura 1 compara os modos de operação nos quais somente a infraestrutura en-

trega as mensagens e com o *Push-and-Track* funcionando. Na Figura 1(a), a infraestrutura envia uma cópia da mensagem através da rede celular para cada um dos nós interessados no mesmo conteúdo. Logo, a carga da rede está relacionada com o número de nós interessados nesse conteúdo. Na Figura, 1(b), com o PnT, a infraestrutura envia apenas três cópias inicialmente para alguns nós da rede e espera a disseminação da mensagem através da comunicação oportunista. Cada nó ao receber a mensagem através do canal secundário envia um reconhecimento positivo (ACK) para a infraestrutura confirmando o recebimento correto da mensagem. Assim, alivia-se a carga da infraestrutura, pois algumas cópias da mensagem podem ser entregues através da comunicação oportunista.



(a) Infraestrutura pura. (b) *Push-and-Track.*

Figura 1. Modos de Operação: infraestrutura pura e PnT.

O PnT possui um claro compromisso de desempenho. Quanto mais cópias iniciais enviadas, menor a redução da carga da infraestrutura, porém, menor o tempo de entrega das mensagens. Por outro lado, quanto menos cópias iniciais, menor a carga da infraestrutura e, provavelmente, maior o tempo de entrega das mensagens. Os pontos-chave do PnT, portanto, são (*i*) decidir quantas cópias iniciais devem ser injetadas, (ii) para quem devem ser enviadas essas cópias e (iii) se novas cópias devem ser injetadas ao longo do tempo. Para isso, o PnT implementa um sistema de controle com base nos ACKs recebidos pela infraestrutura de rede. Para decidir se novas cópias devem ser injetadas, o sistema de controle do PnT utiliza informações da função objetivo e da taxa de infecção. Define-se que a função objetivo é uma métrica predefinida que deve ser atendida no processo de divulgação da mensagem através da comunicação oportunista. A taxa de infecção, por sua vez, é dada pelo número de nós que receberam a mensagem, calculado através do número de ACKs recebidos pela infraestrutura. Em intervalos de tempos curtos, compara-se a função objetivo com a taxa de infecção. Caso a taxa de infecção seja menor do que a função objetivo, a infraestrutura envia mais cópias da mensagem pelo canal primário para outros nós que ainda não receberam o conteúdo para acelerar a disseminação. Para tomar a decisão para quem enviar cópias, pode-se utilizar estratégias aleatórias, baseadas no tempo de permanência, localização e conectividade [Whitbeck et al. 2011, Whitbeck et al. 2012]. Whitbeck *et al.* concluem que escolher aleatoriamente nós para envio de novas cópias pela infraestrutura é uma estratégia simples e eficiente.

Para garantir que todos os nós receberão uma cópia da mensagem desejada dentro de um intervalo máximo de tempo, o PnT define uma zona de pânico. No instante inicial da zona de pânico, a infraestrutura envia através do canal primário cópias da mensagem

para todos os nós interessados que ainda não confirmaram o recebimento da mensagem. A fatia tempo da zona de pânico é calculada pela diferença entre o tempo necessário para a infraestrutura enviar uma cópia da mensagem para todos os nós interessados através do canal primário e o tempo máximo predefinido para entregar a mensagem a todos os nós. Portanto, entrar na zona de pânico tem um efeito negativo, pois a eficiência do descarregamento diminui, principalmente se a infraestrutura tiver que enviar um número muito grande de cópias neste período. Isso resulta em um aumento significativo da carga de tráfego por mensagem.

Whitbeck *et al.* definem diferentes funções objetivo e analisam o desempenho da rede utilizando essas funções. Definem, por exemplo, funções com taxas de envio de mensagens mais lentas (*Slow Start*) e funções com taxas mais rápidas (*Fast Start*). Os resultados apresentados mostraram uma alta eficiência do descarregamento com o PnT. Porém, foram obtidos em um cenário perfeito, no qual todos os nós cooperam com a comunicação oportunista e enviam mensagens de reconhecimento positivo para a infraestrutura. Na prática, os nós podem apresentar comportamento egoísta não enviando mensagens de reconhecimento e não encaminhando mensagens para outros nós. Neste trabalho, é analisada a influência destes comportamentos no desempenho do descarregamento de tráfego com o PnT.

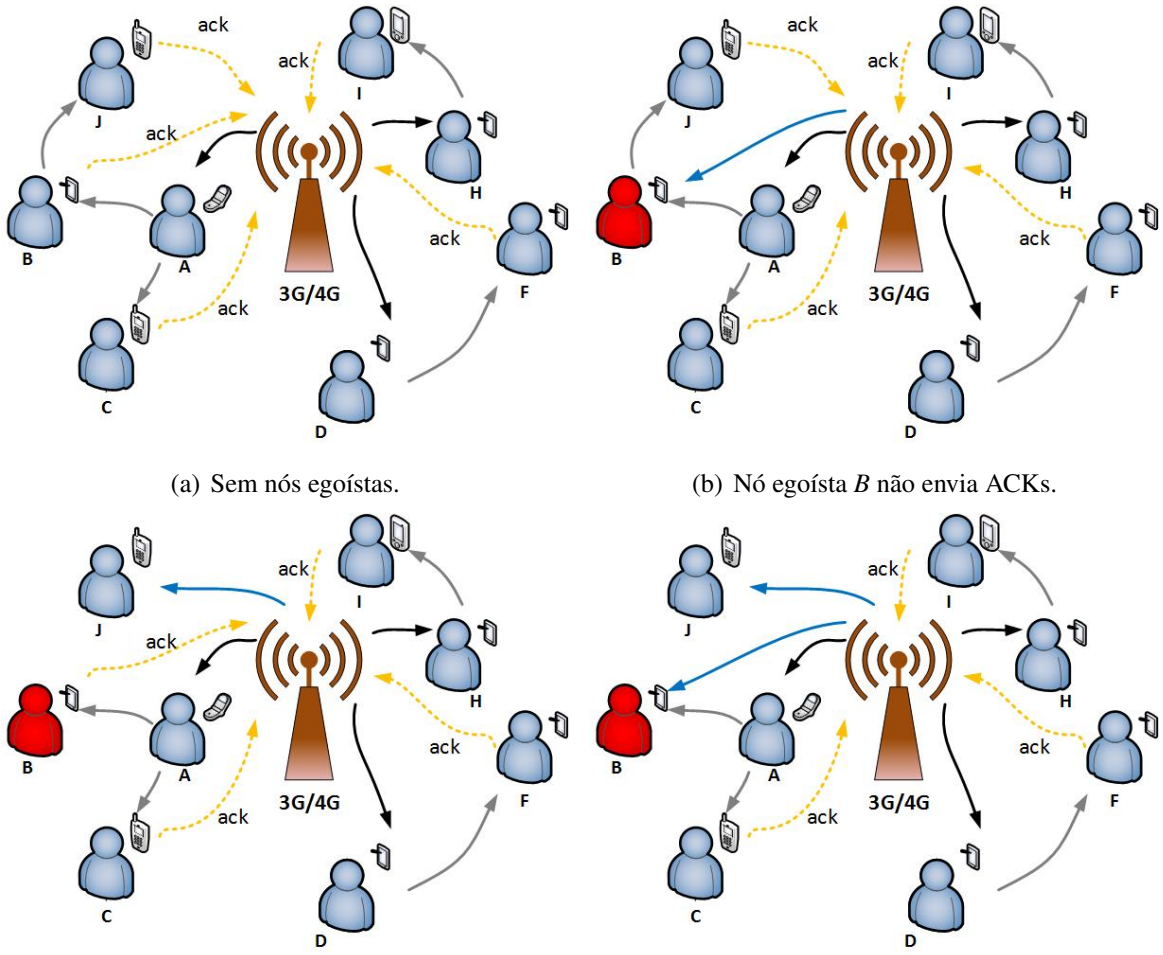
3. Modelos de Ataque

Neste trabalho, são considerados três possíveis comportamentos de nós egoístas, chamados de ataques, por simplificação:

- Ataque de não confirmação: nós egoístas recebem as mensagens através da rede oportunista, mas não enviam ACKs para a infraestrutura;
- Ataque de não encaminhamento: nós egoístas recebem as mensagens da infraestrutura ou da rede oportunista, mas não encaminham as mensagens para outros nós;
- Ataque combinado: nós egoístas recebem as mensagens da infraestrutura ou da rede oportunista, mas não encaminham as mensagens para outros nós e não enviam ACKs para a infraestrutura.

No ataque de não confirmação, como os nós não encaminham ACKs, a infraestrutura não terá controle dos nós que receberam as mensagens através da comunicação oportunista. Na Figura 2(b), o nó *B* apresenta comportamento egoísta. *B* recebe a mensagem através do contato oportunista com o nó *A*, porém não envia um ACK para a infraestrutura. Este comportamento prejudica o sistema de controle da infraestrutura no cálculo da taxa de infecção, ou seja, a infraestrutura não terá a informação correta de quantos e quais nós receberam a mensagem. Logo, a infraestrutura poderá reinjetar mais cópias desnecessárias para atingir a função objetivo ou reenviar a mensagem duplicada na zona de pânico por não ter a informação que o nó recebeu a mensagem. Por um desses motivos, o nó *B* recebe novamente a mesma mensagem da infraestrutura, ilustrado pela seta de cor azul. Neste ataque a comunicação oportunista não é prejudicada, porque os nós egoístas apenas não enviam mensagens de reconhecimento positivo. Por exemplo, na Figura 2(b), o nó *B* encaminha a mensagem para o nó *J*, cooperando com a comunicação oportunista.

Com o ataque de não encaminhamento, o objetivo é reduzir a eficiência da comunicação oportunista. Este comportamento na prática pode não ser necessariamente



(a) Sem nós egoístas.

(b) Nó egoísta *B* não envia ACKs.

(c) Nó egoísta *B* não encaminha mensagens.

(d) Nó egoísta *B* não encaminha mensagens e não envia ACKs.

Figura 2. Comportamento dos nós egoístas.

visto como um ataque, mas acontece pelo fato dos usuários estarem mais preocupados com a conservação da vida útil da bateria dos seus dispositivos do que cooperarem com a comunicação oportunista, por exemplo. A Figura 2(c) ilustra este ataque. O nó *B* recebe a mensagem do nó *A*, em seguida, *B* tem uma oportunidade de contato com o nó *J*, porém não encaminha a mensagem. Como *J* não tem outra oportunidade de contato com outro nó, a infraestrutura envia a mensagem diretamente para ele. Este envio pode acontecer no momento que a infraestrutura reinjetar novas cópias da mensagem para atender a função objetivo ou no momento em que entra na zona de pânico para garantir que todos os nós recebam a mensagem. Neste ataque o cálculo da taxa de infecção não é afetado porque ao receber mensagens na oportunidade de contato, os nós enviam ACKs para a infraestrutura.

O ataque com a combinação dos comportamentos de não confirmação e não encaminhamento aumenta a probabilidade de reenvio de cópias desnecessárias e, assim, diminui a eficiência da utilização das redes oportunistas para aliviar a carga de tráfego da infraestrutura. Na Figura 2(d), o nó *B* não encaminha mensagem para o nó *J* e não envia ACKs para a infraestrutura. Neste exemplo, a carga de tráfego aumentará porque a infraestrutura terá que enviar duas novas mensagens: uma para o nó *J* e uma mensagem

duplicada para o nó *B*. Percebe-se os prejuízos deste comportamento quando se comparam as Figuras 2(a) e 2(d). No cenário sem nós egoístas, a infraestrutura envia somente 3 mensagens e no cenário com o ataque combinado envia 5 mensagens, um aumento de 40% da carga da infraestrutura nesse exemplo simples.

4. Cenário de Avaliação

O impacto dos nós egoístas na rede oportunista é avaliado por simulação. Para tanto, utiliza-se um simulador desenvolvido especificamente para avaliar o *Push-and-Track* e que foi cedido por seus autores [Whitbeck et al. 2011]. Esse simulador é uma adaptação do ONE (*Opportunistic Network Environment*) [Keränen et al. 2009]. Para analisar o comportamento dos nós egoístas, foi necessário modificar algumas classes do simulador para que nós da rede apresentassem um dos comportamentos egoístas descritos na Seção 3. Assim, nós podem não enviar ACKs ao receberem a mensagem através da comunicação oportunista ou/e não encaminhar mensagens para outros nós em uma oportunidade do contato. Uma vez que um nó é definido como egoísta no início de uma rodada de simulação ele mantém esse comportamento durante toda a rodada. Os nós egoístas são escolhidos aleatoriamente a cada rodada de simulação.

No cenário avaliado, existe apenas uma célula e dentro dessa célula os nós se movem com base em um registro real de mobilidade, o Rollernet [Benbadis 2009]. O conjunto de dados desse registro foi coletado em um período de cerca de 3 horas em um experimento no qual foram distribuídos 62 iMotes para participantes de um circuito de patinação, que acontece regularmente em Paris. No registro de dados foram considerados os 62 iMotes e mais 1050 dispositivos externos que tiveram contato com os iMotes.

Em todos os cenários de simulação, define-se que taxa de transferência *downlink* da infraestrutura é 100 KB/s e *uplink* é 10 KB/s. Esses são valores típicos de uma rede celular definidos por Whitbeck *et al.* que afirmam que pesquisas na Europa e nos EUA mostram que a taxa média de *downlink* 3G é normalmente abaixo de 128 KB/s e *uplink* de 10 KB/s [Whitbeck et al. 2011]. Durante um contato, a taxa de transferência entre os nós é de 1 MB/s. O tamanho da mensagem de dados é de 1 MB e o dos ACKs é de 256 B. Novas mensagens de dados são geradas a cada 70 s e são do interesse de todos os nós. O algoritmo de roteamento utilizado em todas as simulações é o epidêmico [Ip et al. 2008]. Considera-se o tempo de vida das mensagens (TTL) igual a 60 s. A infraestrutura entra em zona de pânico 10 s antes de expirar o tempo de vida da mensagem, tempo este necessário para a infraestrutura enviar a mensagem diretamente para os usuários que não receberam com taxas de 100 KB/s. Os destinatários das cópias iniciais de uma mensagem são escolhidos aleatoriamente, assim como os nós da fase de reinjeção. Whitbeck *et al.* concluíram que a estratégia aleatória apresenta bons resultados e, portanto, não necessita-se da complexidade adicional de um canal de controle mais sofisticado. Nesse caso não é preciso armazenar informações sobre o tempo de permanência, localização e/ou informação de proximidade dos nós. É interessante destacar que o simulador restringe a comunicação com apenas um nó por vez no momento de um contato.

5. Resultados

O objetivo dos experimentos é mostrar que a presença de nós com diferentes comportamentos egoístas na rede reduz a eficiência do descarregamento de tráfego, indepen-

dentemente da função objetivo empregada pelo PnT. Cada comportamento egoísta descrito na Seção 3 é analisado para três funções objetivos: *Fast Start*, *Slow Start* e adaptativa. A função objetivo *Slow Start* é dada pela seguinte expressão $\frac{x}{2}$, se $t < TTL/2$; $\frac{3x}{2} - \frac{1}{2}$, se $t \geq TTL/2$. A função *Fast Start* é dada por $\frac{3x}{2}$, se $t < TTL/2$; $\frac{x}{2} + \frac{1}{2}$, se $t \geq TTL/2$. Os valores de x são encontrados através da divisão do tempo de vida atual pelo o tempo de vencimento da mensagem [Whitbeck et al. 2012]. A função adaptativa proposta por Rebecchi *et al.* evita períodos com taxas de infecção constante e superior ao valor determinado pela função objetivo. Diferente das duas primeiras funções, na função adaptativa mantém-se o histórico da taxa de infecção e compara-se esse valor. Caso ele se mantenha constante, a infraestrutura envia novas cópias para a rede [Rebecchi et al. 2014]. Para cada configuração são executadas 10 rodadas de simulação. Em cada rodada, sorteia-se um novo conjunto de nós egoístas.

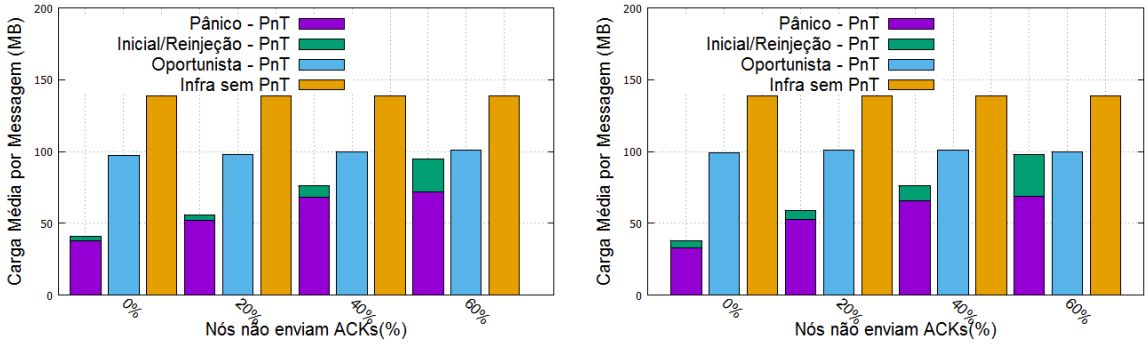
5.1. Cenário 1: Nós egoístas não enviam reconhecimentos positivos (ACKs)

Neste primeiro cenário, avalia-se o ataque de não-confirmação, ou seja, nós egoístas recebem e encaminham mensagens para outros nós de forma oportunista, mas não enviam ACKs para a infraestrutura. Assim, a infraestrutura não terá controle dos nós que receberam as mensagens através do contato com outros nós, fazendo com que a estimativa da taxa de infecção seja prejudicada. Portanto, o objetivo deste ataque é reduzir a eficiência do sistema de controle do PnT.

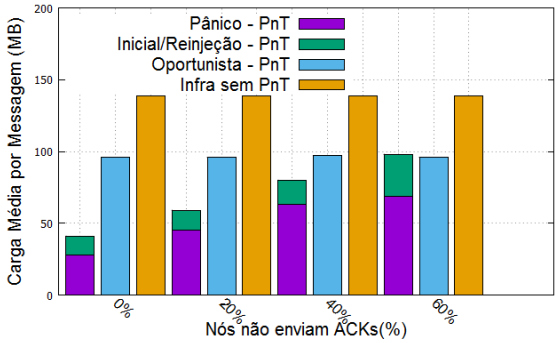
Cada grupo de barras da Figura 3 representa a carga de tráfego média necessária para entregar uma mensagem para todos os nós considerando percentuais específicos de nós egoístas. A barra de coloração laranja representa a carga média sem utilização de qualquer mecanismo de descarregamento. A barra de cor azul mostra a carga de tráfego média entregue pela comunicação oportunista entre os nós utilizando o PnT. A barra de coloração verde e roxa representa a carga média total por mensagem entregue pela infraestrutura com PnT. A porção verde da barra representa a carga de tráfego gerado pelo envio das mensagens na fase inicial e nas fases de reinjeção para atender a função objetivo. A porção roxa indica a carga gerada pelo envio de mensagens durante a zona de pânico.

Observa-se nos resultados apresentados na Figura 3 que a carga média por mensagem da comunicação oportunista mantém-se em aproximadamente 100 MB com o aumento dos nós egoístas em todos os cenários, mesmo utilizando funções objetivo diferentes. A carga de tráfego da comunicação oportunista não é afetada com o aumento do número de nós que não envia ACKs, uma vez que nesse cenário os nós egoístas continuam a encaminhar mensagens para outros nós quando há um contato.

Conforme aumenta o percentual de nós egoístas na rede cresce a carga de tráfego por mensagem da infraestrutura com o PnT. Diminui-se, assim, a eficiência do descarregamento. Por exemplo, observa-se na Figura 3(a) que a carga de tráfego é de 42 MB quando não existem nós egoístas. Com 20% de nós que não enviam ACKs, a carga cresce para 57 MB e chega 77 MB com 40% de nós egoístas, um aumento de aproximadamente 90% da carga. Mesmo utilizando funções objetivos diferentes (Figuras 3(b) e 3(c)), o desempenho do descarregamento de tráfego sofreu praticamente o mesmo impacto com o aumento do percentual dos nós egoístas. No entanto, é importante ressaltar que mesmo com 60% dos nós da rede não enviando ACKs a infraestrutura com o PnT ainda enviou



(a) Função Objetivo *Slow Start.* (b) Função Objetivo *Fast Start.*



(c) Função Objetivo Adaptativa.

Figura 3. Carga de tráfego para o ataque de não confirmação.

aproximadamente 50 MB menos do que a infraestrutura sem PnT. Ou seja, mesmo na presença de nós egoístas o PnT ainda descarrega o tráfego da infraestrutura.

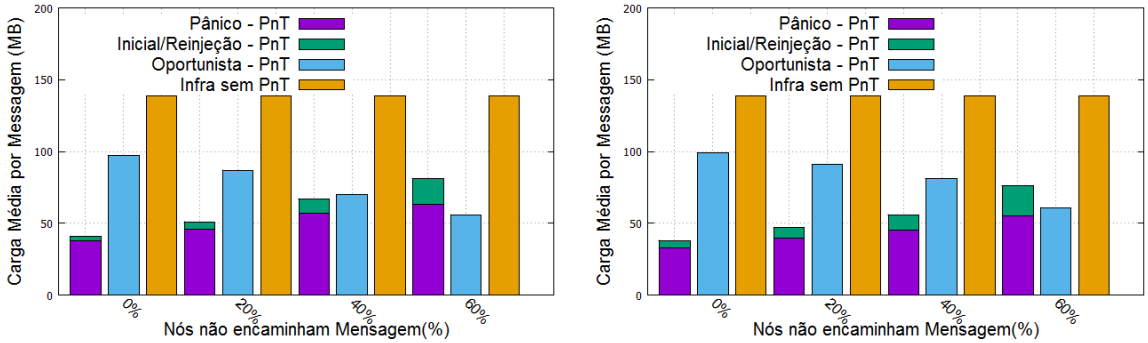
O ataque de não confirmação aumenta a carga da infraestrutura com o PnT por dois motivos. O sistema de controle é prejudicado porque os nós não enviam ACKs ao receberem a mensagem através da comunicação oportunista. Assim a infraestrutura calcula a taxa de infecção com um valor inferior ao valor real do número de nós que receberam a mensagem. Este valor inferior pode ser menor que a função objetivo e provocar envio de mais cópias na fase de reinjeção para atender a função objetivo. Quando a infraestrutura com o PnT entra na zona de pânico, ela envia cópias duplicadas para nós que receberam a mensagem e não confirmaram, devido à infraestrutura não ter o controle de quais nós receberam a mensagem.

5.2. Cenário 2: Nós egoístas não encaminham mensagens

No segundo cenário avalia-se o ataque de não encaminhamento, ou seja, nós egoístas recebem mensagens pelo canal primário e pelo secundário não encaminham para outros nós, prejudicando assim a comunicação oportunista entre os nós.

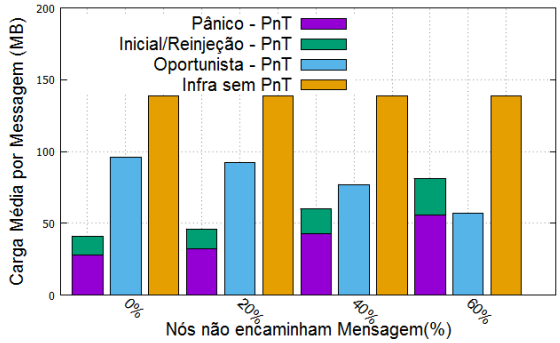
A carga de tráfego gerada na comunicação oportunista no ataque de não encaminhamento difere da carga gerada no ataque de não confirmação. Observa-se na Figura 4 que para todas as funções objetivos analisadas a carga de tráfego média por mensagem na comunicação oportunista diminui com o aumento do percentual de nós que não encaminham as mensagens na oportunidade de contato. Com 60% de nós que não encaminham mensagens a carga da comunicação oportunista diminui aproximadamente 44 MB (*Slow*

Start), 39 MB (*Slow Start*) e 43 MB (adaptativa) comparados com os resultados apresentados na Figura 3. No Cenário 1, mesmo atacando a rede ao não enviar ACKs, os nós egoístas continuam a encaminhar as mensagens na oportunidade de contato. Para todos os experimentos no Cenário 1, a carga de tráfego é de aproximadamente 100 MB.



(a) Função objetivo *Slow Start*.

(b) Função objetivo *Fast Start*.



(c) Função objetivo adaptativa.

Figura 4. Carga de tráfego para o ataque de não encaminhamento.

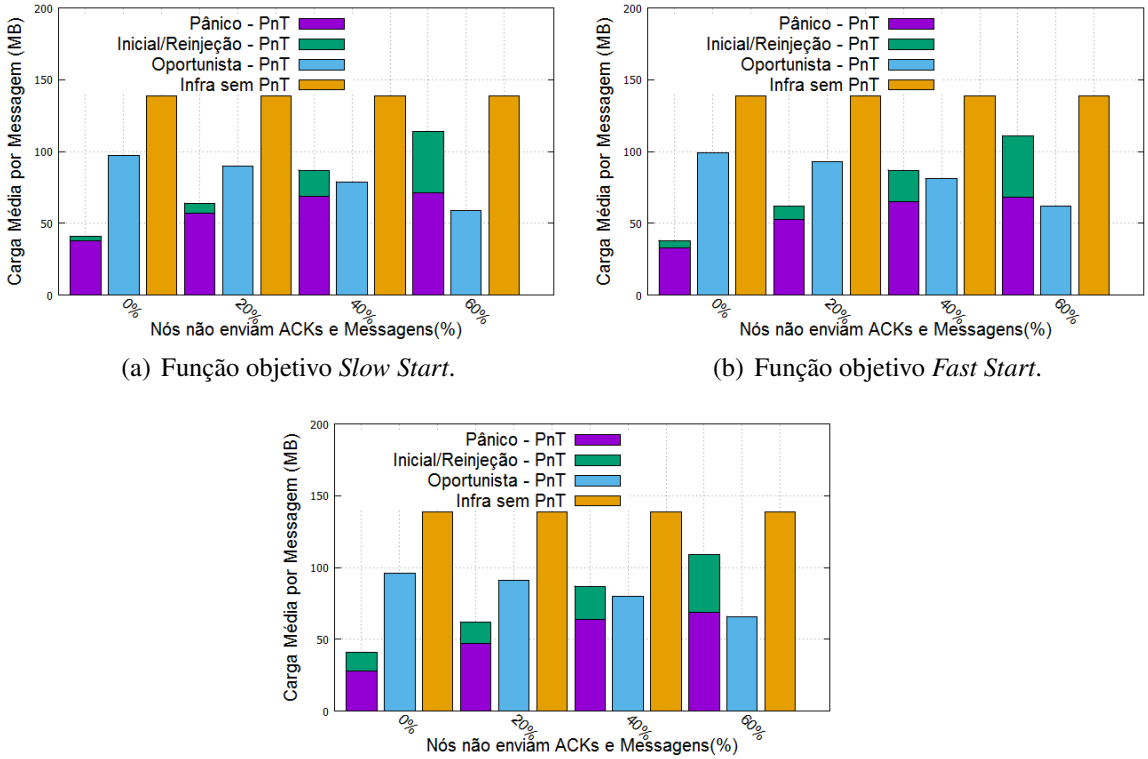
Com o aumento do número de nós egoístas a carga da infraestrutura com o PnT cresce para todas as funções objetivo analisadas. Assim, diminui-se a eficiência do descarregamento. Com 20% de nós que não encaminham mensagens, a carga aumenta para 51 MB, 47 MB e 47 MB, com 40% aumenta para 68 MB, 58 MB e 61 MB para as funções *Slow Start*, *Fast Start* e adaptativa, respectivamente. Justifica-se esse aumento na carga da infraestrutura com o PnT pelo aumento da probabilidade de contato de nós legítimos com nós que não encaminham mensagens. Observa-se que a carga da infraestrutura com PnT com a função objetivo *Slow Start* (Figura 4(a)) é maior do que com as outras funções objetivo. Os valores da função objetivo *Slow Start* na primeira metade de vida da mensagem são calculados pela expressão $\frac{x}{2}$, que são menos agressivos que os valores da função objetivo *Fast Start* (Figura 4(b)). Em outras palavras, a função objetivo *Slow Start* envia um número menor de cópias da mensagem para a rede. E neste cenário, um nó tem maior probabilidade de não receber a mensagem. A função adaptativa (Figura 4(c)) apresentou resultados semelhantes aos da função objetivo *Fast Start*.

Comparando os resultados apresentados nos Cenários 1 e 2, observa-se que o ataque de não encaminhamento é menos prejudicial para o desempenho do descarregamento de tráfego. Com aumento do percentual dos nós que não enviam ACKs, a infraestrutura com o *Push-and-Track* não tem o controle de quais e quantos nós receberam a mensagem,

provocando reenvio de cópias desnecessárias na fase de reinjeção e na zona de pânico.

5.3. Cenário 3: Nós egoístas não encaminham mensagens e não enviam reconhecimentos positivos

Neste cenário é analisada a influência do ataque combinado, ou seja, quando nós egoístas não encaminham mensagens e não enviam ACKs para a infraestrutura. Nota-se na Figura 5 que com o aumento do percentual de nós egoístas na rede, a carga de tráfego da infraestrutura com o *Push-and-Track* aumenta consideravelmente, reduzindo a eficiência do descarregamento de tráfego. Observa-se na Figura 5(b) que com 60% de nós egoístas, a carga gerada pela infraestrutura com o PnT é de 112 MB. No Cenário 1, utilizando a função objetivo *Fast Start* e com 60% de nós que não enviam ACKs, a carga da infraestrutura com o PnT é de 98 MB. Com o mesmo percentual de nós que não encaminham mensagens (Cenário 2) a carga é de 77 MB.



(a) Função objetivo *Slow Start*.

(b) Função objetivo *Fast Start*.

(c) Função objetivo adaptativa.

Figura 5. Carga de tráfego para o ataque combinado.

O aumento da carga da infraestrutura com o *Push-and-Track* ocorre pela junção dos problemas apresentados nos cenários anteriores. O não encaminhamento das mensagens de forma oportunista faz com que a infraestrutura tenha que enviar mais cópias para atender a função objetivo. Por sua vez, o não envio de reconhecimentos positivos, prejudica o cálculo da taxa de infecção, o que causará maior reenvio de cópias da mensagem desnecessariamente pela infraestrutura. Outra consideração, é que a soma da carga de tráfego da comunicação oportunista mais a carga de tráfego da infraestrutura com o PnT difere da carga de tráfego da infraestrutura sem PnT, também em virtude da estimativa errada da taxa de infecção pelo não envio de ACKs, como ocorre no Cenário 1.

Tabela 1. Eficiência do descarregamento de tráfego para os três ataques analisados.

		Funções objetivo		
		Slow Start	*Fast Start*	Adaptativa
Cenário 1	0%	70%	72%	70%
	20%	59%	57%	57%
	40%	45%	45%	42%
	60%	32%	29%	29%
Cenário 2	0%	70%	72%	70%
	20%	63%	65%	66%
	40%	51%	58%	56%
	60%	40%	44%	42%
Cenário 3	0%	70%	72%	70%
	20%	54%	55%	51%
	40%	36%	37%	37%
	60%	18%	20%	21%

A Tabela 1 mostra a eficiência do descarregamento de tráfego para os três ataques analisados. A eficiência do descarregamento de tráfego é calculada pela expressão $(1 - \frac{L_{PnT}}{L_{infra}}) * 100$, onde L_{PnT} é carga de tráfego média por mensagem da infraestrutura com o PnT e L_{infra} é a carga da infraestrutura sem o PnT. Em todos os cenários, mesmo com um aumento significativo de nós egoístas, há descarregamento de tráfego da infraestrutura com o PnT, porém menos significativo. O resultado menos significativo é com o ataque combinado com 60% de nós egoístas com a função objetivo *Slow Start*.

6. Trabalhos Relacionados

Naves e Moraes avaliaram o ataque de falsificação de reconhecimentos positivos (ACKs) em redes oportunistas [Naves and Moraes 2014]. Nesse trabalho, nós maliciosos forjam reconhecimentos para que os nós legítimos descartem mensagens ainda não entregues, reduzindo o desempenho da rede. Analisa-se também uma contramedida proposta na literatura. Portanto, diferencia-se da proposta deste artigo nos seguintes aspectos: não avaliam o desempenho da rede oportunista aplicada como um canal secundário para auxiliar a infraestrutura e não analisam o caso em nós não cooperam com o encaminhamento das mensagens.

Theja e Ramesh avaliaram o impacto de nós não cooperativos, ou seja, nós que não encaminham mensagens para outros nós, no tempo de vida das mensagens [Theja and Ramesh 2012]. Tal avaliação se diferencia deste trabalho, por não avaliar a carga de tráfego, e a utilização das redes oportunistas no descarregamento de tráfego e por não considerar que os nós não enviam ACKs diretamente para a infraestrutura ao receberem cópias das mensagens através da comunicação oportunista.

Rebecchi *et al.* propõem modificações na proposta de descarregamento de tráfego com o *Push-and-Track* e denominam essa nova versão como *Droid (Derivative Reinjection to Offload Data)* [Rebecchi et al. 2014]. Basicamente, eles definem uma nova função objetivo, que evita períodos de planalto, ou seja, momentos em que a taxa de

infecção mantém-se constante ou superior à função objetivo. Portanto, também não analisam o comportamento dos nós egoístas no desempenho da proposta, assim como no trabalho original do PnT.

Liu *et al.* também introduzem um mecanismo de descarregamento de tráfego através das redes oportunistas [Liu et al. 2013]. Os autores definem uma política para selecionar o número de réplicas iniciais e uma política para distribuir as réplicas iniciais limitadas sob a influência de comportamentos egoístas. Nesta proposta, entretanto, os autores não analisaram o impacto dos nós egoísta na eficiência do descarregamento de tráfego. O enfoque era o aumento do tempo médio da entrega das mensagens. Além disso, a proposta não garante que 100% das mensagens serão entregues, assim como faz o *Push-and-Track*.

7. Conclusões e Trabalhos Futuros

Neste artigo, avaliou-se a eficiência do descarregamento de tráfego com o *framework Push-and-Track* (PnT) na presença de nós egoístas. Esses nós não encaminham mensagens para outros nós, não enviam reconhecimentos positivos para infraestrutura ou combinam esses dois comportamentos. O desempenho do PnT foi avaliado para três funções objetivo diferentes: *Slow Start, Fast Start* e adaptativa.

Os resultados mostraram que quando 40% de nós não enviam ACKs para a infraestrutura, houve um aumento na carga da infraestrutura com o PnT de até 90% comparado com o cenário perfeito, no qual todos os nós enviam ACKs. Esse comportamento é semelhante independentemente da função objetivo usada. Observou-se também que o ataque de não encaminhamento é menos prejudicial para o desempenho do descarregamento de tráfego do que o ataque de não confirmação. Isso porque com aumento do percentual dos nós que não enviam ACKs, a infraestrutura com o PnT não tem o *feedback* de quais e quantos nós receberam a mensagem, provocando reenvio de cópias desnecessárias na fase de reinjeção e na zona de pânico. Quando combinados os ataques, a eficiência do descarregamento de tráfego com o PnT foi reduzida de 70% para 18%.

Como trabalhos futuros, pretende-se propor uma contramedida para identificar os nós egoístas na rede, visando melhorar a eficiência do descarregamento de tráfego através da comunicação oportunista com o PnT. Também pretende-se estudar e analisar outros ataques a redes oportunistas no descarregamento de tráfego e propor novas contramedidas.

Agradecimentos

Os autores agradecem a J. Whitbeck e a F. Rebecchi pelo simulador disponibilizado. Em especial a CAPES, pelo financiamento do DINTER IFTM/UFF.

Referências

Benbadis, F. (2009). CRAWDAD data set upmc/rollernet (v. 2009-02-02). Downloaded from http://crawdad.org/upmc/rollernet/.

Chaintreau, A., Hui, P., Crowcroft, J., Diot, C., Gass, R., and Scott, J. (2006). Impact of human mobility on the design of opportunistic forwarding algorithms. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–13.

Han, B., Hui, P., Kumar, V. A., Marathe, M. V., Pei, G., and Srinivasan, A. (2010). Cellular traffic offloading through opportunistic communications: A case study. In *Proceedings of the 5th ACM Workshop on Challenged Networks*, CHANTS '10, pages 31–38, New York, NY, USA. ACM.

Ip, Y.-K., Lau, W.-C., and Yue, O.-C. (2008). Performance modeling of epidemic routing with heterogeneous node types. In *Communications, 2008. ICC '08. IEEE International Conference on*, pages 219–224.

Keränen, A., Ott, J., and Kärkkäinen, T. (2009). The ONE Simulator for DTN Protocol Evaluation. In *SIMUTools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, New York, NY, USA. ICST.

Li, Y., Qian, M., Jin, D., Hui, P., Wang, Z., and Chen, S. (2014). Multiple mobile data offloading through disruption tolerant networks. *Mobile Computing, IEEE Transactions on*, 13(7):1579–1596.

Liu, Z., Wu, Y., and Deng, S. (2013). Optimal mobile data offloading policy through delay tolerant networks with selfish nodes. *International Journal of e-Education, e-Business, e-Management and e-Learning*, 3(5):386–390.

Naves, J. F. and Moraes, I. M. (2014). Uma avaliação do ataque de falsificação de reconhecimentos positivos em redes tolerantes a atrasos e desconexões. *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos SBRC2014*, pages 105–118.

Oliveira, C. T., Moreira, M. D. D., Rubinstein, M. G., Costa, L. H. M. K., and Duarte, O. C. M. B. (2007). Redes tolerantes a atrasos e desconexões. In *Minicursos Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos SBRC2007*, Belém, PA, Brasil.

Rebecchi, F., Dias de Amorim, M., and Conan, V. (2014). Droid: Adapting to individual mobility pays off in mobile data offloading. In *Networking Conference, 2014 IFIP*, pages 1–9.

Rebecchi, F., Dias de Amorim, M., Conan, V., Passarella, A., Bruno, R., and Conti, M. (2015). Data offloading techniques in cellular networks: A survey. *Communications Surveys Tutorials, IEEE*, 17(2):580–603.

Theja, R. and Ramesh, Y. (2012). The impact of message lifetime on delay-tolerant networks with non-cooperative nodes. In *Internet (AH-ICI), 2012 Third Asian Himalayas International Conference on*, pages 1–4.

Whitbeck, J., Amorim, M., Lopez, Y., Leguay, J., and Conan, V. (2011). Relieving the wireless infrastructure: When opportunistic networks meet guaranteed delays. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, pages 1–10.

Whitbeck, J., Amorim, M., Lopez, Y., Leguay, J., and Conan, V. (2012). Push-and-track: Saving infrastructure bandwidth through opportunistic forwarding. *Pervasive and Mobile Computing*, 8(5):682–697.

Zhuo, X., Gao, W., Cao, G., and Dai, Y. (2011). Win-coupon: An incentive framework for 3g traffic offloading. In *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, pages 206–215.

Tramonto: Uma estratégia de recomendações para Testes de Penetração

Daniel Dalalana Bertoglio[1], **Avelino Francisco Zorzo**[1]

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brazil

daniel.bertoglio@acad.pucrs.br, avelino.zorzo@pucrs.br

Abstract. *In the past years, a wide variety of contributions for security testing in organizations, networks and systems have been presented. Furthermore, there is a variety of environments that may be the target for those tests, such as networks, systems, databases, web applications, Internet of Things (IoT) or Cloud Computing. Hence, to have a standard for security tests, using methodologies and frameworks, may increase the efficiency and help the tester. Therefore, this work presents Tramonto, a recommendations strategy for Penetration Testing that is based on the main existing security testing methodologies.*

Resumo. *As pesquisas envolvendo técnicas para testes de segurança em organizações, redes e sistemas, têm apresentado grande variedade nas suas contribuições. A diversidade de cenários-alvo nos quais esses testes são executados pode envolver, por exemplo, Redes, Sistemas, Bancos de Dados, Aplicações Web, Internet das Coisas (IoT) ou Computação em Nuvem. A partir disso, a padronização destes testes, através de metodologias e frameworks, pode aumentar a eficácia e ajudar o tester na execução das atividades. Este trabalho apresenta a Tramonto, uma estratégia de recomendações para Testes de Penetração criada a partir das principais metodologias de teste de segurança existentes.*

1. Introdução

Os riscos relacionados com a segurança de informações em empresas, organizações e entidades que trabalham com dados sensíveis, sejam eles públicos ou não, têm obtido certa expressividade nas preocupações de tais instituições [8]. Riscos que não são controlados potencializam ataques a segurança, que por sua vez implicam em perdas consideráveis. O uso de técnicas para testes de segurança, então, é uma alternativa relevante combater e reduzir esses riscos [15].

Dentre esses testes de segurança, uma das técnicas conhecidas é o Teste de Penetração (Pentest). Pentest é a tentativa controlada de penetrar um sistema ou rede a fim de detectar vulnerabilidades, empregando as mesmas técnicas que são utilizadas em um ataque propriamente dito. Essa alternativa permite que sejam tomadas medidas adequadas para eliminar as vulnerabilidades antes que possam ser exploradas por terceiros não autorizados [14].

O processo de um Pentest se divide, normalmente, em atividades como: coleta de informações sobre o sistema alvo; escaneamento do sistema alvo e descoberta dos serviços; identificação de sistemas e aplicações; descoberta de vulnerabilidades e

exploração de vulnerabilidades [4]. Para uma empresa, este processo é aplicado visando objetivos variados, como o natural aumento da segurança dos sistemas, identificação de vulnerabilidades, teste da equipe de segurança da empresa alvo e até mesmo o aumento da segurança organizacional e de pessoas. Ainda nesse sentido, a aplicação do Pentest pode ser classificada em critérios como [14]: base de informações, que trata o nível de conhecimento sobre o alvo; agressividade, representando o nível de profundidade do teste; escopo do teste e, por fim, abordagem e técnica utilizadas.

Testes de penetração, enquanto técnica para avaliação de segurança, são capazes de fornecer um nível de verificação de segurança adequado e com muitos detalhes em respeito as fraquezas do alvo. Ainda assim, pode-se ressaltar que as atividades de um Pentest também possuem, em sua essência, preocupações legais da execução e aplicação do teste, principalmente em virtude das inúmeras formas de aplicação do mesmo e dos cenários-alvo.

Com as variações nas formas e processos de um Pentest, a padronização deste pode ser considerada uma tarefa complexa. As principais metodologias de teste de segurança contém aspectos que não contemplam amplamente essas variações, além do fato de que um teste de penetração também apresenta diferenças em razão do *tester* que o executa. Assim, é toda a experiência e *know-how* do *tester* precisa ser levada em consideração para pontuar essas diferenças.

Dessa forma, a criação de uma estratégia de recomendações de teste é uma maneira relevante para auxiliar as atividades no processo de um teste de penetração. A partir das metodologias de teste de segurança que são consolidadas na comunidade científica, é possível criar fluxos, tarefas e indicações que permitam ao *tester* aplicar as verificações de segurança independente do cenário e da classificação do tipo de teste. Portanto, este trabalho apresenta a Tramonto, uma estratégia de recomendações direcionada a testes de penetração e suas principais concepções. A Seção 2 descreve as principais metodologias de teste de segurança existentes e que são base da Tramonto. Já a Seção 3 apresenta a conceituação e formalização das características da estratégia de recomendações criada. Além disso, é apresentado ainda um comparativo da Tramonto em relação às demais metodologias, envolvendo critérios relevantes no âmbito de modelos de teste. A Seção 4 detém a discussão e as principais contribuições da Tramonto. Por fim, a Seção 5 apresenta as considerações finais deste trabalho.

2. Metodologias de Teste

Através da realização de um mapeamento sistemático [1], foi possível identificar as seguintes metodologias, frameworks e modelos de teste de segurança: OSSTMM (*Open Source Security Testing Methodology Manual*) [6][9][12][2], ISSAF (*Information Systems Security Assessment Framework*) [12], PTES (*Penetration Testing Execution Standard*) [9], NIST (*National Institute of Standards and Technology) Guidelines* [12] e OWASP *Testing Guide* [6][9]. As próximas subseções descrevem, resumidamente, cada uma destas metodologias.

2.1. OSSTMM

OSSTMM (*Open Source Security Testing Methodology Manual*) [5] é a metodologia que detém um padrão internacional para testes de segurança, mantida pela ISECOM (*Institute for Security and Open Methodologies*). Suas definições são estabelecidas a partir

do escopo, que representa todo o ambiente de segurança operacional possível para qualquer interação com qualquer ativo. Este escopo é composto por três classes: COMSEC (*Communications Security Channel*), PHYSSEC (*Physical Security Channel*) e SPECSEC (*Spectrum Security Channel*). Essas classes, por sua vez, são divididas em cinco canais antes de serem usados pelo *tester*: Humano, Físico, Sem Fio, Telecomunicações e Redes de Dados.

Com base nas especificações apresentadas nos canais, a metodologia OSSTMM tornou-se uma das mais completas e robustas dentre os modelos de teste de segurança. Um importante diferencial que ela apresenta é a inclusão de fatores humanos como parte dos testes. Por outro lado, pode-se afirmar que a metodologia desconsidera itens como uma avaliação cíclica de vulnerabilidades encontradas e diagramas representando os fluxos do teste. Esses dois itens impactam, respectivamente, em uma diminuição de descobertas alternativas de vulnerabilidades e em um maior trabalho de interpretação dos módulos presentes na metodologia.

2.2. ISSAF

A metodologia ISSAF (*Information Systems Security Assessment Framework*) [3] é um *framework* capaz de modelar os requisitos de controle internos para a segurança da informação, e tem por objetivo avaliar a segurança de redes, sistemas e aplicações. Sua concepção é estruturada em três grandes áreas de execução: planejamento e preparação, avaliação e relatório, limpeza e destruição de artefatos. A fase de Planejamento e Preparação trata os passos necessários para definir o ambiente de teste, ferramentas de teste, contratos e aspectos legais, definição da equipe de trabalho, prazos, requisitos e estrutura dos relatórios finais. Já a fase de Avaliação representa o centro da metodologia, onde o teste de segurança é realmente executado. Esta fase possui nove (9) atividades principais, que seguem o fluxo básico de um ataque (reconhecimento, invasão e pós-invasão).

A metodologia ISSAF possui ampla documentação sobre a sua estrutura, e apresenta como uma das principais vantagens a criação de uma conexão entre as tarefas do teste e as ferramentas utilizadas. Da mesma forma, a ordem na qual a metodologia descreve o teste é otimizada para ajudar o *tester* em um trabalho com fluxo mais claro, evitando erros comumente associados com estratégias de ataques selecionadas aleatoriamente. Como limitações, pode-se ressaltar a falta de melhores orientações na elaboração de relatórios e também o fato de que a ISSAF desconsidera hipóteses que podem melhorar o procedimento do teste.

2.3. PTES

A metodologia PTES (*Penetration Testing Execution Standard*) [11] detalha instruções de como executar as atividades que são requeridas para testar precisamente o estado da segurança em um ambiente. A intenção da metodologia é não estabelecer padrões muito rígidos para um teste de penetração. A comunidade de analistas e profissionais de segurança, responsável por sua criação, trata a ideia de que as diretrizes para o processo de avaliação da segurança de um ambiente devem ser de fácil compreensão para as organizações. Por essa razão, as orientações técnicas ajudam a definir procedimentos a serem seguidos durante um Pentest, fazendo com que a metodologia forneça um estrutura base para iniciar e conduzir um teste de segurança. A estrutura da metodologia

é composta por sete fases: *Pre-engagement interactions*, *Intelligence gathering*, *Threat modeling*, *Vulnerability analysis*, *Exploitation*, *Post-exploitation* e *Reporting*.

Em resumo, PTES é projetado para fornecer às empresas uma linguagem mais comum para a realização de um Pentest. Isso é possível através da utilização de padrões e orientações de como o teste precisa ser realizado. Ao longo de sua descrição, a PTES apresenta esses padrões que, aliado ao fato de que a metodologia considera o conhecimento do *tester* como aspecto primordial, representa a principal vantagem da PTES em relação as demais.

Além disso, a construção da metodologia por parte da comunidade de experts na área de segurança tem atribuição direta com maiores preocupações ligadas aos critérios técnicos de um teste de segurança [7]. Em contraponto, isso impacta na falta de detalhes mais específicos nos aspectos de negócio.

2.4. NIST Guidelines

A metodologia proposta pela NIST (*National Institute of Standards and Technology*) [13] foi inicialmente introduzida como GNST (*Guideline on Network Security Testing*), reproduzida na publicação especial 800-42, e a sua última versão continuada é apresentada na publicação especial 800-15 como *Technical Guide to Information Security Testing and Assessment*. Basicamente, sua estrutura segue quatro etapas principais: **planejamento**, onde o sistema é analisado para encontrar os alvos de teste mais interessantes; **descoberta**, onde o *tester* procura as vulnerabilidades no sistema; **ataque**, onde o *tester* verifica se as vulnerabilidades encontradas podem ser exploradas; e **relatório**, onde cada resultado proveniente das ações realizadas na etapa anterior é reportado.

A elaboração deste metodologia é considerada como a primeira que introduz um processo mais detalhado para a escrita de relatórios, assim como o fato de lidar com hipóteses induzidas. De acordo com as melhores práticas, a metodologia sugere escrever um relatório passo-a-passo, onde o *tester* relata suas descobertas depois da fase de planejamento e depois de cada ataque (realizado com sucesso ou não), descrevendo as vulnerabilidades que puderam ou não ser exploradas. Além disso, outro aspecto importante da metodologia é a maneira como são construídos os vetores de vulnerabilidades [13].

2.5. OWASP Testing Guide

A metodologia proposta no OWASP (*Open Web Application Security Project*) *Testing Guide* [10] é um advento consolidado de todos os estudos que a comunidade OWASP realiza. Sua concepção é guiada pela ideia norteadora de tornar softwares seguros uma realidade, e por essa razão percebe-se que suas diretrizes são direcionadas a testes de segurança em softwares e aplicações web. Na grande maioria das organizações voltadas a desenvolvimento de software, as preocupações com segurança não fazem parte do processo de desenvolvimento padrão e, por muitas vezes, também não detém importância para as mesmas. A metodologia, então, idealiza o uso de testes de segurança como forma de conscientização e estrutura-se com base em outros projetos providos pela própria OWASP como o *Code Review Guide* e *Development Guide*.

A metodologia é dividida em três grandes blocos: a etapa introdutória que trata os pré-requisitos para testar as aplicações web e também o escopo do teste, a etapa inter-

mediária que apresenta o *OWASP Testing Framework* e suas tarefas e técnicas relacionadas as diversas fases do ciclo de vida de desenvolvimento de software, e a etapa conclusiva que descreve como as vulnerabilidades são testadas através da inspeção de código e dos testes de penetração.

3. Tramonto

As metodologias de teste de segurança são diferenciadas em diversas questões, como, por exemplo, nas definições sobre planejamento do teste, no tratamento de atividades a serem executadas, nos possíveis cenários-alvo dos testes, e até mesmo no rigor o qual a metodologia lida com a execução do teste.

Aliado com essas diferenças, mensurar e comparar as metodologias de teste de segurança é uma tarefa que requer cuidado. Os principais critérios para essa comparação podem ser considerados: flexibilidade, que diz respeito a forma como a metodologia permite a atuação do *tester* em suas escolhas em contraponto a rigidez do padrão proposto, e eficácia, que está relacionado com aspectos de como metodologia planeja e adequa as suas indicações de modo a atender o objetivo do teste. Baseado nesses critérios, outras características podem ser avaliadas para a comparação entre as metodologias. Algumas dessas características são apresentadas na Seção 3.2 e também dispostas na Tabela 1.

A partir da análise das principais metodologias de teste de segurança, foi criada uma estratégia de recomendações chamada Tramonto. A Tramonto tem como objetivo principal solucionar os problemas relacionados a falta de metodologias voltadas para Pentest, auxiliando o *tester* no processo de intrusão. Contudo, a Tramonto também tem por objetivo adequar atividades, planos, processos e fases de forma a oferecer recursos para um Pentest padronizado, flexível e eficaz.

No que diz respeito aos possíveis cenários-alvo de um Pentest, a Tramonto oferece maiores possibilidades de tratamento, considerando que as metodologias que a constroem ampliam naturalmente essa atuação nos cenários. Tais cenários-alvo foram identificados conforme o mapeamento sistemático disposto em [1], e a Figura 1 apresenta a relação de atuação das metodologias existentes e da própria Tramonto com estes cenários.

3.1. Formalização da Estrutura da Tramonto

Inicialmente, para que fosse possível idealizar soluções e definições prévias da estratégia de recomendações Tramonto, foi necessário estabelecer a estrutura básica que determina as etapas pelos quais as atividades do Pentest estão contidas, conforme apresentado na Figura 2.

Dessa forma, a estrutura da Tramonto é dividida em cinco etapas, que são as seguintes:

- Adequação: etapa responsável por gerenciar todas as escolhas iniciais do *tester* em respeito às informações de escopo, abordagem, dados do alvo, tipo do teste a ser realizado, entre outras. A partir das escolhas e determinações efetuadas, a Tramonto conduzirá o *tester* para os fluxos de trabalho seguintes, fornecendo todas as escolhas possíveis que são categorizadas de acordo com o andamento do teste. Neste ponto, toda experiência e *know-how* do *tester* é levada em conta já que a estratégia irá recomendar as melhores alternativas, não engessando as escolhas caso o profissional venha a decidir por outro caminho de execução.

		METODOLOGIAS					Tramonto
		OSSTMM	ISSAF	PTES	NIST Guidelines	OWASP Testing Guide	
CENÁRIOS - ALVO	Aplicações Web e Web Services	✖	✖	✖	✖	✓	✓
	Serviços e Protocolos de Rede	✓	✓	✓	✓	✖	✓
	Aplicações e Software	✓	✓	✓	✓	✖	✓
	Físico	✓	✖	◉	✖	✖	✓
	Cloud	✖	✖	◉	✖	✖	◉

Legenda: ✓ - Trata devidamente o cenário; ◉ - Recomenda aspectos de teste no cenário; ✖ - Não trata o cenário.

Figura 1. Cenários-alvo para cada metodologia e Tramonto.

Figura 2. Estrutura da Tramonto.

- Verificação: mediante as alternativas delimitadas na etapa anterior, a etapa de verificação consistirá em efetuar os *checklists* de necessidades, dados, atividades orientadas que foram ou não efetuadas e demais itens relevantes. O intuito dessa etapa é minimizar o número de falhas para a continuidade do teste, de forma com que a Tramonto contribua para que o teste seja o mais detalhista possível, sem retrabalhos.
- Preparação: envolve a escolha das estratégias de teste de penetração a serem efetuadas, bem como a indicação de kits de ferramentas de acordo com os processos anteriores. O processo de preparação idealiza fornecer ao *tester* a análise e detalhamento do planejamento e forma de execução do teste. Nesta etapa a Tramonto permite que as soluções de aplicação do teste de penetração estejam suficientemente elaboradas e com as devidas prescrições mediante os dados provenientes das etapas anteriores.
- Execução: trata o núcleo principal de execução do teste de penetração. Nesta etapa, a Tramonto fornece todo o aparato em relação aos vetores de ataque, que são os possíveis caminhos utilizados pelo *tester* para realizar as intrusões. Os vetores de ataque refletem o planejamento do tipo de ataque que é efetuado, podendo esse ser baseado em computadores (por meios tecnológicos) ou baseado em

pessoas (caracterizados pelo contato direto). Além disso, são listados também os possíveis resultados a serem obtidos de acordo com as ações e demais informações relacionadas. É essencial ressaltar que, nesta etapa, os fluxos oferecidos como alternativas de execução do teste devem estar filtrados de acordo com as informações e verificações das etapas anteriores. Essa funcionalidade permite que o *tester* otimize o tempo do teste mediante suas escolhas anteriores, além de induzir atividades mais precisas sobre o escopo delimitado.

- Finalização: a última etapa proposta na estrutura da Tramonto contempla as ações relacionadas com a elaboração dos relatórios a serem fornecidos ao cliente. Ao mesmo tempo, como característica adicional, a construção de um relatório destinado ao próprio *tester* é um dos itens que a estratégia de recomendações produz ao término do teste, permitindo a criação de um padrão que, posteriormente, pode vir a fornecer as possibilidades de comparação entre os resultados obtidos. Ainda nesse processo são tratadas as atividades de cobertura de rastros, limpeza de registros e controle de estado dos sistemas e aplicações alvo.

É possível ainda descrever um sexto processo que descreve uma avaliação, oferecendo alternativas baseadas no resultado obtido e no resultado desejado inicialmente. Nesse sentido, é estabelecida uma nova funcionalidade específica da Tramonto chamada de Plano Alternativo (PA), responsável por fornecer atividades, fluxos, ferramentas, estratégias e todo o tipo de características iminentes que, dependendo das escolhas e ações do *tester*, trazem novas possibilidades para o fluxo processual do teste de penetração. Dessa forma, cada uma das etapas estabelecidas pode conter planos alternativos que são identificados pelo nome do processo, seguido de um identificador (por exemplo, *PAP1* refere-se a um plano alternativo na etapa de preparação e detém um *ID* 1). Assim, compreende-se que devido a vasta quantidade de informações diferentes contidas nas metodologias analisadas, podem ser criados diversos sub-planos de acordo com as informações iniciais o teste.

3.2. Comparação da Tramonto com as principais metodologias

A Tramonto, conforme descrito anteriormente, é uma estratégia de recomendações destinada exclusivamente para testes de penetração. Nesse sentido, é preciso considerar que a Tramonto não trata-se de uma metodologia, e também que a comparação em relação às suas metodologias base é realizada para compreender quais itens podem ou não ser atendidos pela estratégia.

O fato de ser uma estratégia direcionada para testes de penetração já representa um quesito passível de comparação. Percebe-se que, dentre as metodologias citadas na Seção 2, somente a PTES é específica para testes de penetração, enquanto as demais destinam-se para testes de segurança diversos.

Para estabelecer uma comparação entre as metodologias e a Tramonto, foram consideradas as seguintes características: Abrangência, Flexibilidade, Modelagem, Adaptação, Planejamento e Relatório. Essas características foram definidas a partir dos critérios citados no início desta Seção 3: flexibilidade e eficácia do teste. Características como Abrangência, Modelagem, Adaptação e Planejamento, representam aspectos que contribuem diretamente para ambos os critérios principais. Tais características são explicadas nas subseções 3.2.1, 3.2.3, 3.2.4 e 3.2.5, respectivamente. Além disso, a Flexibilidade (subseção 3.2.2) também é considerada uma característica para comparação,

Tabela 1. Comparativo de características entre as metodologias e a Tramonto.

	OSSTMM	ISSAF	PTES	NIST Guidelines	OWASP Guide	**Tramonto**
Última Atualização	2010	2006	2012	2008	2014	**2016**
Específica para Pentest	Não	Não	Sim	Não	Não	**Sim**
Abrangência	A	AP	A	AP	NA	**A**
Flexibilidade	AP	NA	AP	A	NA	**A**
Modelagem	A	AP	AP	NA	A	**A**
Adaptação	A	AP	NA	AP	AP	**AP**
Planejamento	AP	A	A	AP	A	**A**
Relatório	A	AP	AP	A	AP	**A**

Legenda - A: Atende; AP: Atende Parcialmente; NA: Não Atende.

juntamente com o item Relatório (subseção 3.2.6), que trata a documentação do teste e outros detalhes.

A Tabela 1 apresenta um comparativo dessas características em relação as metodologias. Os valores para essa comparação são categorizados em:

- **Atende** (A): Fornece definições detalhadas para lidar com a característica de maneira apropriada.
- **Atende Parcialmente** (AP): Questões sobre a característica são mencionadas, mas sem o rigor e detalhe necessários.
- **Não Atende** (NA): A metodologia não menciona nada relacionado à característica, ou não contempla as definições da mesma.

Com base na comparação, as subseções a seguir descrevem cada característica e como a Tramonto busca adequar-se para atender os critérios.

3.2.1. Abrangência

Inicialmente, uma das atividades importantes para um teste de segurança é a definição do escopo. Escopo refere-se às preocupações do alcance do teste em relação aos possíveis cenários-alvo. As metodologias OSSTMM, ISSAF, PTES e NIST são facilmente integradas e podem ser adequadas para aplicações e sistemas operacionais, banco de dados, avaliações de segurança física e aplicações web. Contudo, o modelo *OWASP Testing Guide* tem seu foco precisamente definido: serviços e aplicações web. Dessa forma, a abrangência dessa metodologia pode representar uma limitação.

Na Tramonto a abrangência é tratada na fase de Adequação. A principal contribuição da estratégia em relação a este critério se dá através do direcionamento do fluxo das atividades de cada etapa de acordo com o cenário-alvo. Assim, quanto melhor delimitado for o andamento durante a execução do teste, maior será a facilidade no tratamento de possíveis problemas referentes a características do cenário-alvo.

3.2.2. Flexibilidade

A possibilidade de integrar meios, itens e direções adicionais ao teste de segurança a partir dos resultados obtidos em cada etapa ou fase da metodologia, é uma característica importante no contexto atual de verificações de segurança. Nesse sentido, mesmo que uma definição estática dos planos e passos a serem seguidos seja um requisito primordial, a flexibilidade de incluir novos itens torna uma metodologia mais interessante. Para essa característica, o modelo fornecido pela NIST apresenta um aspecto interessante ao permitir que o *tester* tenha maior dinamicidade ao longo do teste, podendo considerar e reavaliar seus artefatos obtidos a cada atividade. Em contraponto, algumas metodologias, como OSSTMM ISSAF e *OWASP Testing Guide*, ao mesmo tempo que são extremamente robustas, limitam tal flexibilidade por tratarem os cenários de execução especificamente.

De forma cíclica e continuada, a Tramonto trata o critério flexibilidade com o intuito de garantir que cada atividade ou passo dentro das etapas seja validada. Nesse sentido, a Tramonto estabelece duas opções possíveis para essa validação: um sistema de notas, para mensurar a capacidade de cada atividade frente ao teste, e uma análise das recomendações, considerando tabelas de ”memória”. O sistema de notas pode balizar o andamento do teste e redirecionar o mesmo de acordo com a satisfação ou não da atividade. Isso é feito de acordo com a nota definida, mecanismo esse que é estabelecido pelo próprio *tester*. Já a tabela de ”memória”consiste em armazenar os conteúdos e experiências relevantes obtidos durante cada atividade para permitir alterações no teste.

3.2.3. Modelagem

Ao definir detalhadamente os aspectos e conceitos norteadores para o processo de teste, a metodologia pode até limitar a flexibilidade, mas incrementa a qualidade da modelagem. Esses conceitos-chave facilitam o *tester* na sua atividade de modelar todo o fluxo de ações do teste, além de modelar o sistema e ambiente alvo. Isso ratifica um ponto crucial em testes de segurança, que é a eliminação de possíveis ambiguidades em relação a cada passo subsequente a ser realizado. Para essa característica, os modelos OSSTMM, *OWASP Testing Guide* e PTES atendem adequadamente, principalmente pela forma com que abordam a etapa de planejamento do seu processo de teste.

Na etapa de Adequação, a Tramonto trata o critério modelagem, onde é inicialmente abordado. Nela são determinadas os meios de abordagem aos conceitos pré-estabelecidos, fazendo com que o *tester* consiga traçar os objetivos do teste e suas subetapas mediante definições *a priori*.

3.2.4. Adaptação

É importante possuir conceitos bem definidos com o intuito de evitar possíveis ambiguidades e, portanto, impactar na adaptação. Além disso, a possibilidade de adaptar a metodologia e suas ações para diferentes ambientes fornece um fluxo do teste mais completo. Entre as possíveis adaptações possíveis estão, por exemplo, a escolha dos tipos de teste, o plano de teste ou a definição de escopo. A partir das metodologias estudadas, por um lado, a OSSTMM é aquela que pode melhor satisfazer esta característica, uma vez

que tem um processo com atividades bem definidas. Por outro lado, a metodologia PTES apresenta algumas limitações por não detalhar concisamente essas adaptações como uma alternativa.

A adaptação para a Tramonto tem uma relação semelhante ao critério abrangência. Contudo, cabe considerar que a Tramonto objetiva fornecer conjuntos pré-definidos de tipos de teste, oferecendo variações em atividades específicas. Em contraponto a isso, também permite que o *tester* consiga alterar alguns desses conjuntos, tornando a execução do teste dinâmica e permissiva.

3.2.5. Planejamento

Todo o conjunto de aspectos definidos em um teste de segurança deve ser devidamente planejado antes de iniciar a execução do teste. Assim, o planejamento é a característica que representa o suporte fornecido ao *tester* para a fase de definição, execução das atividades, pré-requisitos para continuação e andamento do teste, escolha das ferramentas a serem utilizadas e também o retorno esperado para cada atividade dentro do teste. PTES é uma metodologia que fornece esse tipo de característica. Ela descreve, cuidadosamente, todo o planejamento que deve ser definido, além de estabelecer o conjunto de ferramentas, e como operar as mesmas, que serão utilizadas em cada atividade do Pentest. OSSTMM e NIST, uma vez que tentam fornecer maior flexibilidade, não são focadas em prover um planejamento muito detalhado.

Nesse ponto a Tramonto é estritamente vinculada as demais etapas. O planejamento do teste tem inúmeras variações de acordo com a flexibilidade, adaptação e abrangência, fazendo com que cada escolha seja notavelmente acolhida às decisões do *tester*.

3.2.6. Relatório

Por fim, é possível considerar a documentação como parte das características principais da constituição de um Pentest. Todas as metodologias estudadas fornecem como a documentação precisa ser produzida, contendo sugestões e indicações para o preenchimento do relatório final de teste.

Para tal, a Tramonto estabelece uma frequente e assídua documentação de todo o fluxo do teste e com registro dos detalhes considerados mais importantes. Uma contribuição nesse ponto é a possibilidade de constituir um resumo mediante a descrição das atividades e resultados mais relevantes, que pode ser entregue juntamente ao relatório completo.

4. Discussão e Principais Contribuições

O conflito entre padronizar um teste por completo e torná-lo o mais flexível possível a partir das escolhas do *tester* é um problema no qual a Tramonto procura solucionar de uma forma balanceada. Através da criação de uma estratégia de recomendações busca-se indicar soluções para a realização de um teste de penetração que auxilie o *tester* durante toda a execução.

Uma das principais contribuições da Tramonto é permitir alternativas e sugerir, durante as etapas, atividades devidamente planejadas. Essas sugestões, por sua vez, não inibem a atuação do *tester*, já que a Tramonto considera que o seu *know-how* pode implicar diretamente no sucesso do teste. A etapa de Adequação da Tramonto é um exemplo dessa descrição.

Considerando um exemplo da aplicação de um teste de penetração para avaliar a segurança dos servidores de uma instituição de ensino, as discussões da etapa de Adequação da Tramonto ficam em torno de:

1. Informações sobre o alvo: as indicações iniciais do teste são guiadas, essencialmente, pelas informações obtidas em reuniões e/ou repassadas pelo cliente. Por vezes, o cliente pode ser induzido a avaliar o estado de segurança de outros pontos nos quais ele pode não ter considerado. Dessa forma, o *tester* pode sugerir meios que possam contribuir para que o teste seja mais eficaz e, aliado a isso, acabem testando esses pontos que possam interferir na preocupação inicial do cliente. No exemplo de aplicação, o cliente (instituição) passa apenas o desejo de testar a segurança de seus servidores, e fornece os dados sobre os mesmos (nomes, *range* endereços IP e localização física, entre outros). Além disso, o cliente repassa as informações de controles, diretivas e políticas de segurança que estão relacionadas aos servidores. Essas informações são requisitadas pela Tramonto e são recomendações provenientes, principalmente, das metodologias NIST e PTES.
2. Informações gerais do teste: neste ponto da etapa de Adequação são estabelecidas as características gerais do teste, como datas, intervalos de tempo para execução do teste, avaliações e métodos utilizados. Determinar datas e horários para a aplicação do teste é um quesito que a Tramonto recomenda não só com o intuito de assegurar um melhor funcionamento do mesmo, mas também para relacionar com o tipo de Pentest que será efetuado. Caso seja optado por avaliar também a equipe que responde pelos incidentes de segurança, os horários podem ser diferentes dos determinados inicialmente pelo cliente. Ainda nesse sentido, os intervalos para execução do teste também são determinados e discutidos com o cliente para que os cuidados com possíveis interrupções no negócio sejam devidamente tomados. A Tramonto apresenta alguns *checklists* na etapa de Verificação, mas esses são criados a partir dessas informações da etapa de Adequação. Outro aspecto contido nas informações gerais do teste são as escolhas para a avaliação ou não dos controles, políticas e demais aspectos gerenciais anteriormente obtidos com o cliente. Para o caso da avaliação dos servidores, o intervalo de tempo fica aberto ao *tester*, ou seja, sem restrição por parte da instituição. Não são avaliadas as políticas e diretrizes, apenas é considerado o plano de contingência para eventuais problemas de funcionamento dos sistemas armazenados nestes servidores. A Tramonto efetua as recomendações desse item a partir de tópicos tratados pelas metodologias OSSTMM e NIST.
3. Determinações contratuais e implicações legais: antes do início do teste, é requisito obrigatório da Tramonto a atribuição de aspectos contratuais. É necessário delimitar em contrato qual o teste será efetuado, os limites que a penetração não pode ultrapassar ou alcançar (e caso ocorra ou o cliente solicite posteriormente, saber quais as implicações contratuais), o acordo de confidencialidade (em relação à informações sensíveis passíveis de serem descobertas ao longo do teste, por exem-

plo) e por fim, os custos envolvidos em toda a operação. Além disso, existem os cuidados com as implicações legais, envolvendo a violação de controles ou até mesmo a descoberta de registros que possam comprovar atividades maliciosas por parte de colaboradores da empresa cliente. Os prazos de entrega do relatório final e de soluções para mitigação de ataques e correção de vulnerabilidades é outro ponto tratado neste item. No exemplo da aplicação do teste nos servidores da instituição de ensino, não é autorizada a divulgação das informações a respeito desses itens.

4. Estratégias e técnicas do teste: a etapa de Adequação encerra-se listando, a partir das informações coletadas anteriormente, as possibilidades de estratégias e técnicas que o *tester* pode querer utilizar na execução do teste. Essas possibilidades são criadas pela Tramonto com o intuito de facilitar o teste, fazendo com que o *tester* possa indicar suas preferências a partir de soluções pré-determinadas. Contudo, essa pré-determinação é construída com base em duas vertentes: padrões indicados pelas metodologias (como OSSTMM, OWASP e PTES) e testes de penetração efetuados anteriormente. O segundo ponto é um contribuição relevante da Tramonto, já que ao término de cada teste efetuado, é possível atribuir uma avaliação que pode indicar que as escolhas deste teste realizado mostraram-se eficientes para atingir o objetivo proposto. Para o exemplo em questão, as estratégias sugeridas pela Tramonto são duas: teste de segurança a partir da rede interna e teste de segurança física.

Dessa forma, é perceptível que o *tester* possui uma flexilidade relevante utilizando a Tramonto. Assim, conclui-se que o fato do teste não ser completamente padronizado e de que há a interferência do *tester* indica que a Tramonto resolve o problema anteriormente descrito através de uma solução equilibrada que mostra-se interessante para o âmbito de testes de segurança.

A proposta de uma estratégia de recomendações é uma solução nova na área de Testes de Penetração, o que ratifica a relação de nível diferente entre a Tramonto e a demais metodologias de teste de segurança (OSSTMM, por exemplo). Neste ponto, uma limitação da Tramonto é não representar uma metodologia, pois considera-se que criar uma metodologia requer o know-how de muitos especialistas da área e principalmente de um número elevado de avaliações de aplicação da mesma em testes executados. Por outro lado, a Tramonto contempla, além de tarefas, passos, fluxos e conceitos provenientes de cada uma das metodologias, características próprias (como a solução de planos alternativos, por exemplo). Novas funcionalidades aliadas às demais características das metodologias base podem incrementar a qualidade do teste executado, e este ponto representa outra contribuição relevante da Tramonto.

Da mesma forma, a Tramonto apresenta também como contribuições:

- Análise das metodologias base: Cada etapa da Tramonto foi determinada e construída a partir de uma análise detalhada das principais metodologias utilizadas para testes de segurança. Assim, a criação da Tramonto é um processo que contempla uma análise de conteúdo das metodologias, realizada com o viés destinado a Testes de Penetração. A partir disso foi possível identificar características as quais não se encaixam adequadamente em padrões de teste de segurança atuais, como a identificação de vetores de ataque indicada pela ISSAF, por exemplo. Em

contraponto, o fornecimento de *checklists* e outros aspectos que são relacionados a outros tipos de teste de segurança contribuíram para funcionalidades além das triviais de um Pentest.

- Fornecer aos executores dos testes uma alternativa de padronização mais intuitiva e principalmente flexível, considerando que a Tramonto busca determinar um processo que se aproxime da melhor solução na relação inversamente proporcional entre modelos com passos e fluxos fechados e as escolhas pessoais dos próprios executores do teste, como afirmado anteriormente.
- A utilização de uma aplicação que auxilie o *tester* a seguir as recomendações da Tramonto e que, paralelamente, registra as atividades e resultados de cada etapa da estratégia. Neste ponto existe a proposta de uma solução de elaboração de relatório de um teste a partir das indicações feitas na aplicação. Esse fator também permite que a aplicação possa fornecer, a partir da extração de conhecimento dos dados de testes já executados, padrões de solução de testes de penetração realizados com a Tramonto.

5. Considerações Finais

O uso de testes de penetração como método para avaliação da segurança de sistemas e demais cenários-alvo mostra-se uma solução interessante para organizações e empresas. Isso se deve ao fato da forma como o teste é executado, simulando intrusões e comportamentos de atacantes maliciosos, fornecendo assim uma percepção próxima da realidade dos ataques reais que, por sua vez, vêm tomando uma proporção exponencial em quantidade e efetividade. Para o tratamento desses ataques e o uso dos testes de segurança é essencial a atuação da comunidade de profissionais de segurança e das pesquisas relacionadas a esse contexto, pois a partir desse movimento, surgem padrões e soluções que auxiliam a área. As metodologias analisadas neste trabalho possuem ampla consolidação em estudo e uso, porém com objetivos e características variadas. Isso motivou a construção da estratégia de recomendações Tramonto, contribuindo para uma solução destinada a testes de penetração com base nas principais metodologias de teste de segurança. No cenário de Pentest, a Tramonto apresenta-se como uma alternativa única no que diz respeito ao desenvolvimento de recomendações, avaliando cuidadosamente o processo de Pentest e propondo funcionalidades para aumentar a eficácia e auxiliar o *tester* em suas atividades. Nesse sentido, a Tramonto é norteada pelo objetivo de atender os critérios de eficácia e flexibilidade do teste, sabendo que atualmente grande parte dos *testers* utiliza metodologia própria ou alguma que não é destinada exclusivamente para Pentest, implicando em não conseguir mensurar devidamente esses critérios. Assim, acredita-se que a Tramonto pode futuramente ser aprimorada para incluir os diferentes cenários de aplicação de testes de penetração, aprimorando o nível de detalhe e idealizando a criação de padrões de teste que possam oferecer soluções e avaliações mais rápidas e eficazes.

Referências

[1] D. D. Bertoglio and A. F. Zorzo. Um mapeamento sistemático sobre testes de penetração. Technical Report TR 084, Faculdade de Informática, Programa de Pós-Graduação em Ciência da Computação, Pontifíca Universidade Católica do Rio Grande do Sul (PUCRS), Porto Alegre, Rio Grande do Sul, 2015.

[2] T. Dimkov, A. van Cleeff, W. Pieters, and P. Hartel. Two methodologies for physical penetration testing using social engineering. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 399–408, New York, NY, USA, 2010. ACM.

[3] O. I. S. S. Group. *Information Systems Security Assessment Framework*. Open Information Systems Security Group, 2006.

[4] K. M. Henry. *Penetration Testing: Protecting Networks and Systems*. IT Governance Publishing, 2012.

[5] P. Hertzog. *OSSTMM - Open Source Security Testing Methodology Manual*. Institute for Security and Open Methodologies (ISECOM), 2010.

[6] F. Holik, J. Horalek, O. Marik, S. Neradova, and S. Zitta. Effective penetration testing with metasploit framework and methodologies. In *Computational Intelligence and Informatics (CINTI), 2014 IEEE 15th International Symposium on*, pages 237–242, Nov 2014.

[7] D. Kennedy, J. O'Gorman, D. Kearns, and M. Aharoni. *Metasploit: The Penetration Tester's Guide*. No Starch Press, San Francisco, CA, USA, 1st edition, 2011.

[8] K. Lam, D. LeBlanc, and B. i. Smith. *Assessing network security*. Redmond, Wash. Microsoft Press, 2004.

[9] B. Liu, L. Shi, Z. Cai, and M. Li. Software vulnerability discovery techniques: A survey. In *Proceedings of the 2012 Fourth International Conference on Multimedia Information Networking and Security*, MINES '12, pages 152–156, Washington, DC, USA, 2012. IEEE Computer Society.

[10] M. Meucci, E. Keary, and D. Cuthbert. *OWASP Testing Guide v.3*. OWASP Foundation, 2008.

[11] C. Nickerson, D. Kennedy, E. Smith, A. Rabie, S. Friedli, J. Searle, B. Knight, C. Gates, and J. McCray. *Penetration Testing Execution Standard*. PTES, 2014.

[12] M. Prandini and M. Ramilli. Towards a practical and effective security testing methodology. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 320–325, June 2010.

[13] K. Stouffer, J. Falco, and K. Scarfone. *NIST SP 800-115: Technical Guide to Information Security Testing and Assessment*. National Institute of Standards and Technology, 2008.

[14] A. Whitaker and D. Newman. *Penetration Testing and Cisco Network Defense*. Cisco Press, 2005.

[15] J. J. Zhao, S. Y. Zhao, and S. Y. Zhao. Opportunities and threats: A security assessment of state e-government websites. *Government Information Quarterly*, 27(1):49 – 56, 2010.

Detecção de DDoS Através da Análise da Quantificação da Recorrência Baseada na Extração de Características Dinâmicas e Clusterização Adaptativa

Marcelo Antonio Righi[1], Raul Ceretta Nunes[1]

[1]Programa de Pós-Graduação em Informática – CT – UFSM
Av. Roraima, 1000, B. Camobi – Santa Maria (RS) – Brasil

marcelo.righi@mail.ufsm.br, ceretta@inf.ufsm.br

Abstract. *The high number of Distributed Denial of Service (DDoS) attacks have demanded innovative solutions to guarantee reliability and availability of internet services. In this sense, different methods have been used to analyze network traffic for denial of service attacks, such as neural networks, decision trees, principal component analysis and others. However, few of them explore dynamic features to classify network traffic. This article proposes a new method, called DDoSbyAQR,that uses the recurrence quantification analysis based on the extraction of dynamic characteristics and an adaptive clustering algorithm (A-kmeans) to perform better classification of the attack network traffic. The experiments were done using the CAIDA and UCLA databases and have demonstrated ability to increase the accuracy (98.41%) of DDoS detection.*

1. Introdução

Tradicionalmente, sistemas de detecção de intrusão (*Intrusion Detection System* – IDS) procuram por comportamentos maliciosos utilizando técnicas baseadas em assinaturas ou anomalias. A detecção por assinatura compara o tráfego de rede com uma base de dados de ataques previamente conhecidos (assinaturas), enquanto a detecção por anomalias compara os dados coletados com registros de atividades consideradas normais no sistema [Tsai et al. 2009].

A detecção de intrusão baseada em anomalias vem sendo muito explorada [Silva et al. 2012][Raut e Singh 2014] devido aos inúmeros e persistentes Ataques Distribuídos de Negação de Serviço (DDoS), os quais utilizam até milhares de computadores (pessoais ou servidores) para atacar uma determinada máquina, distribuindo a ação de ataque entre elas.

Para detecção de DDoS são utilizadas variadas técnicas [Raut e Singh 2014]. Entretanto, muitas ainda possuem limitações e a sua eficácia pode ser comprometida devido ao excesso de falsos alertas, como observado em [Raut and Singh 2014]. A existência de comportamentos dinâmicos não lineares e não estacionários na série de tráfego pode ser um dos fatores [Palmieri e Fiori 2010] para maior efetividade, dado que o tráfego de rede contém propriedades como auto-similaridade [Willinger et al. 1998], dependência de longo alcance [Grossglauser e Bolot 1999] e recorrência [Marwan e Webber 2015].

A Análise da Quantificação da Recorrência (AQR) [Webber e Marwan 2015], utilizada inicialmente com limitações em [Palmieri e Fiori 2010] [Kumar 2012] [Jeyanthi et al. 2014] e em outros domínios [Vieira et al. 2012], pode também proporcionar

soluções para segurança em redes de computadores de maneira mais eficaz e em tempo real [Righi e Nunes 2015], pois permite analisar o comportamento do tráfego não linear que se repete ao longo de um determinado intervalo de tempo e emitir alertas para reagir a um ataque DDoS.

Na AQR é possível extrair diversas características dinâmicas do comportamento específico para cada sistema, que são chamadas de Medidas de Quantificação da Recorrência (MQR), tais como Taxa da Recorrência (REC), Determinismo (DET), Entropia (ENTR), Tendência (TREND), Laminaridade (LAM), dentre outras. Tais medidas norteiam a AQR, o que resulta numa análise focada nas características dinâmicas extraídas ao invés de uma análise focada nas variabilidades momentâneas do tráfego. Esta propriedade foi preliminarmente explorada [Righi e Nunes 2015], tendo demonstrado melhor resultado quando comparado com medidas extraídas diretamente da análise de séries temporais por métodos estatísticos tradicionais.

Este trabalho propõe o DDoSbyAQR, um novo método para detecção de DDoS que utiliza a AQR baseada na extração de características dinâmicas e a clusterização adaptativa [Bhatia 2004] para classificar o tráfego de rede em tempo real. Os resultados dos testes realizados com as bases CAIDA e UCLA demonstraram incremento na acurácia para a detecção de ataques DDoS em redes de computadores. As principais contribuições deste trabalho são: (1) demonstrar que a partir de um conjunto de atributos que caracterizam ataques DDoS pode-se extrair características dinâmicas de recorrência; (2) demonstrar que a AQR pode ser utilizada para diminuir a proporção de falsos positivos e incrementar a de verdadeiros positivos, contribuindo para aumentar a acurácia dos IDS na detecção de DDoS; e (3) demonstrar que um clusterizador adaptativo (A-Kmeans), que calcula automaticamente o número de clusters, pode ser um bom aliado da AQR para aumentar a eficácia na classificação de ataque DDoS.

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados e a Seção 3 apresenta uma revisão teórica sobre a AQR; a Seção 4 apresenta detalhes da implementação do método proposto (DDoSbyAQR**)** e a Seção 5 apresenta os testes, os resultados e a comparação com outros métodos de detecção; finalmente, a Seção 6 apresenta a conclusão do artigo e indica possíveis trabalhos futuros.

2. Trabalhos Relacionados

Esta seção faz uma revisão de algumas técnicas existentes para detecção de DDoS, com a finalidade de apresentar o esforço dos pesquisadores em utilizar os melhores atributos e algoritmos para caracterizar esse tipo de ataque e, também, demonstrar que alguns métodos exploram características dinâmicas, porém com muitas limitações ainda.

A AQR foi utilizada em [Kumar 2012] [Jeyanthi et al. 2014], no entanto, a utilização é genérica, pois aplicam diretamente a extração de características dinâmicas nos traços de rede, gerando gráficos através do site http://www.recurrence-plot.tk/glance.php e visualizando-os de maneira empírica na ferramenta Weka para concluir que mudanças excessivas percebidas visualmente nos valores dessas características sinalizam um ataque ou anomalia, sem a preocupação em verificar os resultados, sua eficácia, falsos positivos e além disso, sem levar em conta o tempo de execução dessas ações e a geração de alertas em tempo real e sem definir atributos estatísticos que caracterizem um ataque DDoS.

Em [Oo et al. 2013], o método proposto procura caracterizar ataques DDoS com base em sete atributos extraídos diretamente do tráfego. Os atributos correspondem a parâmetros específicos do tráfego em situações de ataque e de normalidade e são utilizados diretamente pelo algoritmo classificador K-NN [Nguyen e Choi 2010], que faz a clusterização pela regra do vizinho mais próximo, na qual a escolha do número de clusters é fixa e determinada pelo pesquisador. No entanto, de acordo com a nossa

abordagem, a utilização de um classificador para operar diretamente a série temporal formada em cada atributo, bem como a escolha fixa do número de clusters, pode ser um limitador significativo para obtenção de uma boa eficiência do método de detecção.

Limwiwatkul e Rungsawang (2006) propuseram um método para detectar um ataque DDoS que analisa os cabeçalhos dos pacotes TCP/IP e procura criar um perfil de ataque concentrado principalmente em ICMP Flood, TCP Flood e UDP Flood. Este trabalho reforça a ideia de extração de atributos baseados no comportamento do tráfego de rede para caracterizar um ataque DDoS.

Em [Wu et al. 2011] os autores propõem a detecção de ataques DDoS utilizando um classificador baseado em árvore da decisão (algoritmo C4.5). No trabalho foram utilizados dezesseis atributos para descrever o padrão de tráfego em situação de normalidade. Porém a taxa de falsos positivos cresce quando o tráfego aumenta, conforme demonstrado nos gráficos do artigo, denotando uma menor eficácia do método em uma situação de aumento normal do fluxo da rede. Além disso, a escolha dos atributos não considerou características importantes para DDoS, dado que os atributos escolhidos não contemplam a variância do tamanho dos pacotes e a variância do tempo dos pacotes recebidos, que tendem a zero durante um ataque DDoS [Oo et al. 2013].

De maneira similar, em [Zhong e Yue 2010] os autores apresentam um método de detecção de ataques DDoS que captura o tráfego de rede e analisa o status da conexão do protocolo TCP/IP. Porém, os autores utilizam os algoritmos Apriori, FCM e K-Means, demonstrando que a combinação de múltiplos classificadores pode melhorar a acurácia da detecção, tal como também observado em [Silva et al. 2012].

Em [Grossglauser e Bolot 1999][Palmieri e Fiori 2010] é sugerido que o tráfego de rede se expõe a propriedades onipresentes de auto-similaridade e dependência de longa duração, ou seja, de correlações em uma ampla gama de escalas de tempo. Tais características, de acordo com [Marwan e Webber 2015], podem permitir a aplicação da Análise da Quantificação da Recorrência (AQR) como uma possível técnica para detecção de anomalias. É possível criar gráficos de recorrência e analisá-los através de características dinâmicas [Marwan e Webber 2015] que podem ser extraídas através de testes exaustivos que melhor caracterizam cada tipo de sistema avaliado. Em outras palavras, isto permite representar matematicamente as correlações dos pontos de recorrência e definir o comportamento da série temporal não estacionária.

Dos estudos, observou-se que a AQR, apesar de ser explorada em alguns trabalhos [Kumar 2012] [Jeyanthi et al. 2014] para caracterizar o tráfego de rede, ainda possui limitações e um dos poucos trabalhos que empregam características dinâmicas de uma maneira mais aprofundada é o de [Yuan et al. 2014], onde o autor combina um algoritmo de clusterização com a Transformada Wavelet (TW) e a AQR. Porém a quantidade de falsos positivos no trabalho de Yuan et al (2014) ultrapassa 8% quando utiliza a base de dados DARPA 1999, os dados estatísticos não caracterizam ataques DDoS e o clusterizador é o K-Means que possui a limitação de usar um número fixo de clusters, o que diminui sua eficácia. Diferentemente, neste trabalho exploramos em profundidade o uso da AQR baseada na extração de características dinâmicas, evitando observações apenas visuais dos Gráficos da Recorrência e análises estatísticas tradicionais de séries temporais. Nossa análise baseia-se nos limiares (*Thresholds*) das medidas de quantificação da recorrência (MQRs) para o tráfego considerado normal e na clusterização adaptativa, a qual calcula automaticamente o número de clusters necessários para formar o agrupamento para comparação com os limitadores definidos na fase de testes.

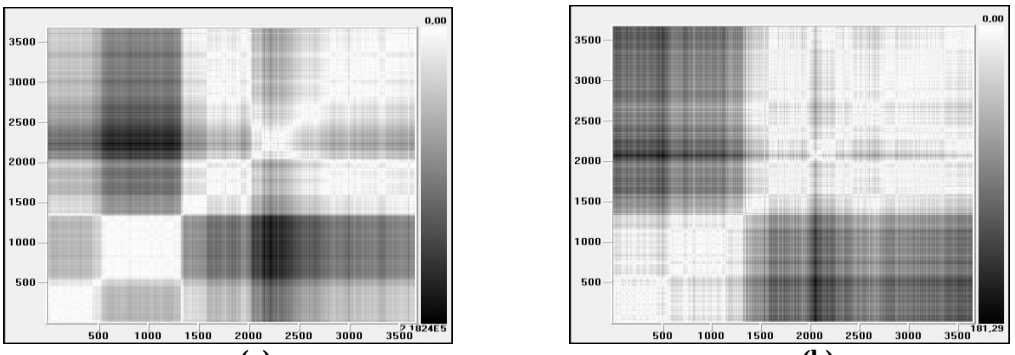
Complementarmente aos trabalhos descritos, foi observado que diversos autores [Rahmani et al. 2009][Bhaya e Manaa 2014][Suresh e Anitha 2011] utilizam as Bases de

Dados CAIDA 2008 e CAIDA 2007 para caracterizar tráfego normal e tráfego de ataque, indicando uma boa aceitação dessa base em pesquisas científicas na área de redes e detecção de DDoS. Tal observação foi fundamental para adoção destas bases nos experimentos deste trabalho.

3. Análise da quantificação da recorrência (AQR)

A quantificação da recorrência corresponde à quantificação do que os gráficos da recorrência mostram. Os Gráficos da Recorrência (vide exemplo na Figura 1) foram propostos em [Eckmann et al. 1987] como uma técnica de análise não linear de sistemas dinâmicos e proporcionam uma visualização do comportamento da trajetória do espaço de fases multidimensional [Webber e Marwan 2015].

Na prática, os Gráficos de Recorrência são matrizes bidimensionais quadradas que representam a evolução dos estados do sistema dinâmico e que são preenchidas por pontos pretos e brancos (vide Figura 1). Os pontos pretos indicam que há recorrência, ou seja, os estados do sistema dinâmico referentes a esses pontos orbitam em regiões próximas uns dos outros na trajetória do espaço de fases. Tais regiões são chamadas de Raio da Recorrência. Um ponto preto marcado na coordenada (i, j) do gráfico representa a recorrência do estado do sistema $e(i)$ no instante j [Eckmann et al, 1987][Webber e Marwan 2015]. Em outras palavras, considerando os gráficos da recorrência da Figura 1, gerados na fase de testes deste trabalho, cada estado do desvio padrão (Figura 1(a)) e média do tamanho dos pacotes (Figura 1(b)) em um determinado instante (i) é comparado com todos os outros estados em cada instante correspondente ($j, j+1, \ldots, n$). No caso de recorrência, um ponto preto será marcado a partir de cada resultado de cada comparação, caso contrário será marcado um ponto branco. No instante ($i+1$) seu estado será novamente comparado com todos os outros estados ($j, j+1, \ldots, n$) e assim sucessivamente até o término da série temporal para cada atributo utilizado. O resultado deste processo é uma matriz quadrada de pontos pretos e brancos que indicam a recorrência do atributo de interesse.



(a) **(b)**

Figura 1. Gráficos da recorrência para atributos do tráfego normal: (a) série temporal do Desvio Padrão do tamanho dos pacotes e seu Gráfico da Recorrência; (b) série temporal da Média do tamanho dos pacotes e seu Gráfico da Recorrência.

A expressão matemática que define a formação do gráfico da recorrência (estados definidos pela Equação 1) é computada com base em uma série temporal $X = \{ e_i, \ldots, e_n\}$ com seus estados e_i, onde $i = 1, 2, 3, \ldots, n$ [Eckmann et al. 1987][Marwan e Webber 2015]. Na equação, m é a dimensão de imersão (representa quantos atrasos são utilizados em relação à série temporal inicial), que neste trabalho é 1, n é o número total de

amostras que contém a série temporal, τ é o tempo de atraso (tempo de espera entre um estado e o outro) e *N* é o número de estados, sendo $N = n - (m-1)\tau$.

$$E = [e_j, e_{j+\tau}, ..., e_N],\ j = 1,2,3,...,N \quad (1)$$

Depois de calcular os estados de tráfego a partir de atributos pré-definidos (para o DDoSbyAQR os 7 atributos que expressam ataques de DDoS, conforme Seção 4), para cada um deles utiliza-se a análise dos fenômenos de recorrência que é realizado segundo a Equação da Recorrência (2).

$$R_{ij} = \theta(\varepsilon - \| e_i - e_j \|) \quad (2)$$

Onde, R_{ij} corresponde a um elemento da matriz de recorrência, ε ao limiar adotado, e_i e e_j aos estados do sistema no espaço de fase *m*-dimensional ora em análise, *N* ao número de estados considerados e θ a função de decisão definida pela Equação (3). De acordo com a Equação (3), se a distância entre os estados e_i e e_j é menor do que o limiar ε, então o valor de $\theta(\varepsilon)$ é 1 e existe um ponto preto na posição (*i*, *j*) do Gráfico da Recorrência; caso contrário, o valor de $\theta(\varepsilon)$ é 0 e existe um ponto branco em (*i*, *j*).

$$\theta(f(\varepsilon)) = \begin{cases} 0 & (\varepsilon - \|e_i - e_j\|) \leq 0 \\ 1 & (\varepsilon - \|e_i - e_j\|) > 0 \end{cases} \quad (3)$$

Cabe salientar que um parâmetro importante na AQR é o raio de vizinhança. Este raio é quem define os pontos recorrentes no gráfico de recorrência, logo é um parâmetro chave na AQR. Como este limiar depende de cada tipo de sistema que está sendo analisado e dos seus objetivos [Marwan e Webber 2015], a literatura não apresenta um método específico para estabelecer o raio da vizinhança ideal que defina os pontos recorrentes, tendo este que ser ajustado de acordo com o tipo de aplicação, no nosso caso detecção de DDoS.

Para o raio da vizinhança foi utilizada a taxa de 10%, com base na observação de valores de outros trabalhos como o de [Yuan et a. 2014]. Porém, para chegar a este valor foi utilizada, também, a técnica em que usa-se, inicialmente, o máximo do valor do raio da vizinhança e tem-se uma recorrência máxima no Gráfico da quantificação da Recorrência (GQR) e gradativamente ela é decrescida ao ponto de termos uma mudança abrupta na observação visual do GQR chegando-se a uma taxa relativamente próxima da ideal. O impacto do raio da vizinha é extremamente relevante, pois se tivermos uma taxa alta haverá muita recorrência no GQR e não se poderá visualizar suas mudanças de comportamento, por outro lado, se a taxa for muito baixa não haverá pontos de recorrência no GQR e nenhuma MQR poderá ser extraída.

A partir do Gráfico da Recorrência, a Análise de Quantificação de Recorrência (AQR) permite realizar e potencializar avaliações visuais da recorrência. Entretanto, a análise visual dos gráficos de recorrência é subjetiva e pode levar a diferentes interpretações. Por isso, com o objetivo de trazer mais precisão às análises, as estruturas presentes nos gráficos de recorrência foram quantificadas através de Medidas de Quantificação de Recorrência (MQR).

De acordo com Marwan (2003), as principais MQRs são:

a) Razão de Recorrência (RR) - mede a densidade dos pontos de recorrência no gráfico de recorrência;

b) Determinismo (DET) - razão entre o número de pontos de recorrência que formam as estruturas diagonais e todos os pontos de recorrência. Está relacionada com a previsibilidade do sistema;

c) Comprimento médio das linhas diagonais (L) – indica o tempo médio que dois segmentos de uma trajetória se mantêm próximos um do outro, podendo ser interpretado como o tempo médio de predição;

d) Comprimento máximo das linhas diagonais (Lmax) – indica o tempo máximo que dois segmentos de uma trajetória se mantêm próximos um do outro. Mais utilizado na análise de quantificação de recorrência do que o comprimento médio das linhas diagonais;

e) Entropia de Shannon (ENTR) - representa a distribuição de frequências dos comprimentos das linhas diagonais e reflete a complexidade da estrutura determinística presente no sistema;

f) Tendência (TREND) - é um coeficiente de regressão linear sobre a densidade dos pontos de recorrência das diagonais paralelas a diagonal principal (linha de identidade). Essa medida fornece informações a respeito da não-estacionariedade do processo;

g) Laminaridade (LAM) - razão entre os pontos de recorrência que formam as estruturas verticais e todo o conjunto de todos os pontos de recorrência presentes no gráfico;

h) Comprimento médio das estruturas verticais (TT) – também conhecida como tempo de permanência em um estado (*Trapping Time*). Essa medida contém informação acerca da quantidade e do comprimento das estruturas verticais no gráfico de recorrência. Ela mede o tempo médio que o sistema permanece em um estado específico.

As Linhas diagonais mostram a evolução dos estados similares em tempos diferentes e as Linhas Verticais indicam os estados que não mudam ou mudam lentamente ao longo do tempo [Marwan e Kurths 2005].

Note que através da AQR e suas MQRs é possível avaliar computacionalmente o comportamento recorrente de sistemas dinâmicos não estacionários independentemente da dimensionalidade dos mesmos, mesmo sob variabilidade do tráfego na rede. Esta propriedade da AQR elimina a necessidade de considerar a estacionariedade da série temporal, tal como necessitam os métodos estatísticos tradicionais, permitindo seu uso inclusive na análise de séries curtas e não-estacionárias. Em síntese, uma das principais vantagens oferecidas pela AQR, em comparação a outras técnicas de análise dinâmica não-linear, é habilitar a análise de pontos de recorrência no espaço de fase bidimensional de um sistema não-estacionário, o que pode evitar desvios (bias) na análise sujeita a sobrecargas eventuais nos parâmetros amostrais do sistema.

4. Detalhamento do Método *DDoSbyAQR*

Nesta seção é apresentado o método de detecção de anomalias *DDoSbyAQR.* A Figura 2 apresenta a arquitetura modular da solução de detecção, onde o módulo de detecção de ataques DDoS que abriga o método proposto é destacado. As seções 4.1, 4.2 e 4.3 detalham cada módulo e a seção 4.4 apresenta o algoritmo que implementa o método *DDoSbyAQR.*

De maneira geral o *DDoSbyAQR* pode ser visto como um método que combina a Análise da Quantificação da Recorrência (AQR) e a Clusterização Adaptativa (A-Kmeans). No contexto da arquitetura da solução proposta (Figura 2), cabe salientar que a extração de atributos corresponde à fase de seleção de atributos de rede que forneçam informações relevantes ao problema de interesse (detecção de DDoS).

Diferentemente, numa das fases da AQR são extraídas características dinâmicas do tráfego de rede e que visam habilitar a análise da recorrência através de MQRs. A clusterização adaptativa flexibiliza o cálculo do número de clusters utilizados para classificar o tráfego, feito automaticamente no A-Kmeans. O cálculo automático potencializa a minimização dos erros de acurácia do classificador. Por exemplo, no Kmeans o número de clusters é determinado pelo pesquisador.

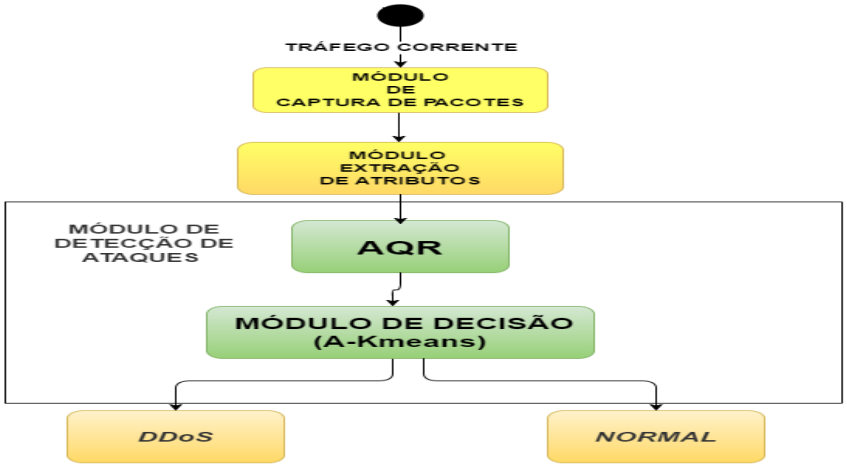


Figura 2. Arquitetura da solução de detecção com o método *DDoSbyAQR*.

4.1. Módulo de Captura de Pacotes

Este módulo seleciona o tráfego de entrada de rede, em intervalos de tempo de 60 segundos, que serão utilizados pela AQR e serve como estação de captura de pacotes, recebendo, periodicamente, dados coletados por um sensor posicionado estrategicamente na rede que utiliza a ferramenta TCPDump para coleta dos traços.

Após coletados, os dados são enviados ao Módulo de Extração de Atributos.

4.2. Módulo de Extração de Atributos

Para detecção de ataques DDoS, a aplicação da AQR exige a utilização de atributos que caracterizem as anomalias de interesse em uma série temporal. A identificação destes atributos foge ao escopo deste trabalho, tendo sido adotados os atributos identificados em [Oo et al. 2013], os quais verificaram em seus experimentos que os atributos que se adequam melhor ao tipo de ataque DDoS são sete, conforme ilustra a Tabela 1.

O Módulo de Extração de Atributos consiste na extração, do tráfego de rede com os listados na Tabela 1, os quais são repassados ao Módulo de Detecção, sendo a ferramenta TCPSTAT utilizada para extração dos atributos estatísticos do tráfego de uma série temporal de 60 segundos, ou seja, são formadas sete séries temporais, uma para cada atributo extraído com o TCPSTAT e aplicada a AQR para extração das características dinâmicas (3 para cada série temporal – Entropia, Determinismo e Razão da Recorrência).

Tabela 1. Atributos utilizados pela AQR. Adaptado de [Oo et al. 2013]

ATRIBUTOS	DESCRIÇÃO
NUM_PAC	Número de pacotes recebidos na rede
NUM_BYTES	Número de bytes recebidos na rede
M_PAC	Média do tamanho dos pacotes recebidos
VAR_TEM_PAC	Variância do tempo dos pacotes (Timestamp) recebidos
VAR_TAM_PAC	Variância do tamanho dos pacotes recebidos
TAX_PAC	Taxa total de pacotes
TAX_BYTES	Taxa total de bytes

4.3. Módulo de Detecção de Ataques

O Módulo de Detecção de Ataques é o módulo central da solução proposta (vide Figura 2), sendo composto pelo Módulo de Análise da Quantificação da Recorrência (seção 4.3.1) e pelo Módulo de Decisão centrado em um classificador adaptativo (seção 4.3.2).

4.3.1. Módulo AQR

No Módulo AQR cada atributo passado pelo Módulo de Extração de Atributos é representado na forma de uma série temporal (60 segundos), modelada por amostras realizadas em períodos equidistantes. A cada série temporal, correspondente a um dos 7 atributos que expressam ataques de DDoS (vide Tabela 1), é aplicada a Análise da Quantificação da Recorrência, conforme definida na seção 3.

Após a formação do gráfico da recorrência, para cada atributo representado como uma série temporal são extraídas três características dinâmicas: a Razão de Recorrência (RR), a Entropia (ENTR) e o Determinismo (DET). Estas características correspondem a Medidas de Quantificação de Recorrência e foram utilizadas em [Vieira et al. 2012] para detecção de anomalias no sinal de voz, sendo reutilizadas no *DDoSbyAQR* para detecção de DDoS por análise de anomalias. Naturalmente, outras medidas, tais como as citadas na seção 3, também podem ser aplicadas nesta etapa do método *DDoSbyAQR.*

Para extrair as características dinâmicas a serem repassadas ao Módulo de Decisão, os cálculos de quantificação (computo da RR, DET e ENTR) aplicados ao Gráfico da Recorrência são feitos como segue.

1) **Razão de Recorrência (RR) -** mede a densidade dos pontos de recorrência no Gráfico da Recorrência.

$$RR = \frac{1}{N^2} \sum_{i,j=1}^{N} R_{i,j} \tag{4}$$

2) **Determinismo (DET) -** razão entre o número de pontos de recorrência que formam as estruturas diagonais e todos os pontos de recorrência. Está relacionado com a previsibilidade do sistema.

$$DET = \frac{\sum_{l=l\min}^{N} lP(l)}{\sum_{i,j=1}^{N} R_{i,j}} \tag{5}$$

Onde $P(l)$ é o número de pontos de recorrência para cada diagonal formada.

3) **Entropia de Shannon (ENTR) -** representa a distribuição de frequências dos comprimentos das linhas diagonais e reflete a complexidade da estrutura determinística presente no sistema. Está relacionada com a incerteza do sistema.

$$ENT = \sum_{l=l\min}^{N} p(l) \log_2 p(l) \tag{6}$$

$$p(l) \frac{P(l)}{\sum_{l=l\min}^{N} P(l)} \tag{7}$$

Salienta-se que as três características dinâmicas utilizadas neste módulo são descritores dinâmicos de comportamento que mantêm a estacionariedade em seus resultados durante o tráfego livre de ataques, mesmo em momentos de alta variabilidade dos estados dos atributos (sete) utilizados para cada uma das séries temporais. Esta propriedade dos descritores pode evitar alarmes falsos na presença de *outliers* não relacionados a ataques de negação de serviço, ou seja, analisar as MQRs geradas é mais eficaz na detecção de DDoS do que analisar a série temporal de origem para cada atributo.

As estruturas diagonais e verticais citadas são parâmetros que se formam no Gráfico da Quantificação da Recorrência, mas que podem ocorrer em maior ou menor quantidade,

ou seja, são linhas verticais ou diagonais que são moldadas pela quantidade de pontos pretos que as formam, sendo que em um determinado sistema pode haver formação maior de estruturas diagonais do que verticais e em outro os valores sejam totalmente inversos. Portanto, cada medida é uma maneira de visualizar matematicamente o GQR e aproveitar cada mínimo detalhe para verificar se há recorrência ou não em uma determinada série temporal de diferentes sistemas.

O Módulo AQR, através das 21 características dinâmicas (3 para cada um dos 7 atributos), forma um conjunto que procura expressar através das propriedades de recorrência o comportamento dos sete atributos analisados. Este conjunto é então encaminhado ao Módulo de Decisão para ser clusterizado e classificado.

4.3.2. Módulo de Decisão

Dado o conjunto de características dinâmicas recebido do Módulo AQR, o Módulo de Decisão tem a função de organizar os dados, particionando-os em grupos por similaridade (*clusters*) e classificando-os como ataque DDoS (anômalos) ou não (não anômalos).

Para contornar a dificuldade de definir o número ideal de clusters, o método *DDoSbyAQR* aplica o algoritmo A-Kmeans [Bhatia 2004] que processa um conjunto com 21 características dinâmicas, três para cada atributo (sete), sendo em cada atributo utilizados os valores da Entropia, o Determinismo e a Razão ou Taxa de Recorrência. Após isso o A-Kmeans calcula automaticamente o número de clusters (valor de "k" é automático) e compara cada um deles com os limiares pré-estabelecidos na fase de treinamento com as bases de traços normais. Na sequência, se a maioria dos clusters for considerada anômala, então o tráfego será nominado como ataque DDoS.

A decisão do módulo está então centrada no cálculo dos centroides (pontos centrais de cada cluster) do conjunto de características dinâmicas recebidas do Módulo AQR e na análise se a maioria dos agrupamentos formados forem classificados como anômalos, situação em que o tráfego será classificado como Ataque DDoS.

4.4. Algoritmo do DDoSbyAQR

A seguir o detalhamento das etapas do algoritmo que implementa o método DDoSbyAQR.

Entrada: séries temporais de tráfego (sete atributos).
Saída: indicação de ataque DDoS ou tráfego normal
Etapa 1: para cada série de tráfego *X* (uma para cada um dos sete atributos), utilizar a AQR para calcular as características dinâmicas (Taxa de Recorrência, Entropia e Determinismo).

$$F_1 = f(X_{NUM\_PAC}) \quad F_2 = f(X_{NUM\_BYTES}) \quad F_3 = f(X_{M\_PAC}) \quad F_4 = f(X_{VAR\_TEM\_PAC})$$

$$F_5 = f(X_{VAR\_TAM\_PAC}) \quad F_6 = f(X_{TAX\_PAC}) \quad F_7 = f(X_{TAX\_BYTES})$$

$$F_n = \{RR_n, ENT_n, DET_n\}, \quad n = 1,2,3,4,5,6,7$$

Etapa 2: combinar as 21 características dinâmicas resultantes da Etapa 1 para descrever os padrões dinâmicos de comportamento do tráfego sintetizados em *F*.

$$F = \{[RR_n, ENT_n, DET_n]\}$$

Etapa 3: usar o algoritmo A-Kmeans para agrupar as características dinâmicas em *F* dentro de diferentes clusters e classificar o comportamento do tráfego como Ataque DDoS ou Normal.

Note que de maneira geral de cada atributo o algoritmo extrai três MQRs (RR, DET e ENTR) através da AQR e permite que o A-Kmeans opere um conjunto de 21 (vinte e uma) características dinâmicas do intervalo de tempo de análise definido pela série temporal.

5. Experimentos e Resultados

5.1. Organização dos Experimentos

Nos experimentos do DDoSbyAQR foram usadas três bases de dados: CAIDA 2007, CAIDA 2008 e UCLA CSD. A base de dados CAIDA 2007 possui cerca de uma hora de ataques DDoS (ICMP *Flood* e TCP *Flood*) divididos em arquivos "pcap" sanitizados de 5 minutos cada. A base de dados CAIDA 2008 possui cerca de 16 horas de tráfego sem ataque divididos em arquivos "pcap" sanitizados de uma hora cada um, colhidos durante dezesseis dias na rede CAIDA de Chicago e San Jose nos EUA. A base de dados UCLA CSD possui traços de uma hora de ataques DDoS (UDP Flood) e traços de tráfego sem ataque coletados em dez dias diferentes. Dos dados, foram extraídos sete atributos, conforme Tabela 1, resultando em uma série temporal *X* para cada atributo de interesse.

Os experimentos foram organizados em duas fases, uma de treinamento e outra de testes. Em ambas fases foram utilizadas séries temporais correspondentes a uma amostragem de 60 segundos. Os dados usados nos experimentos foram organizados tal como ilustra a Tabela 2.

Na fase de treinamento o experimento teve como objetivo calibrar os limiares do método DDoSbyAQR, identificando o comportamento de cada característica dinâmica em traços com e sem ataques. Para tal, o experimento utilizou um conjunto de dados com 62 minutos de traços da Base de Dados CAIDA 2008 e 152 minutos da base UCLA CSD sem traços de ataques, para caracterizar o tráfego normal, e um conjunto de dados apenas com traços contendo ataques DDoS, com 66 minutos de amostragem da base CAIDA 2007 e 56 minutos da base UCLA CSD, para caracterizar o tráfego anômalo.

Na fase de testes o experimento teve como objetivo avaliar a acurácia do método, ou seja, a proporção de predições corretas. Para tal, foram intercaladas linhas de tráfego contendo ataque DDoS com linhas de tráfego sem ataque, sendo que a inserção de linhas de ataque foi feita de forma aleatória em pontos variados no tráfego normal, porém a quantidade de linhas de ataque foi sempre a mesma (300 linhas ou 300 segundos), ou seja, o sistema deverá detectar em cada série de ataque no máximo cinco alertas de ataque DDoS. Neste experimento foram utilizados 128 minutos de dados intercalando dados da base CAIDA 2007 (com ataques) e da base CAIDA 2008 (sem ataques) e 210 minutos de dados intercalando dados da base UCLA CSD com e sem ataques.

Tabela 2. Bases de Dados Utilizadas

BASE DE DADOS	*Tipo de Tráfego*	*Qtde Linhas (1 seg)*	*Tempo (min)*
CAIDA 2008	**Normal**	**3661**	**62**
UCLA CSD Normal	**Normal**	**9155**	**152**
CAIDA 2007	**Ataque DDoS**	**3955**	**66**
UCLA CSD DDoS	**Ataque DDoS**	**3354**	**56**
CAIDA 2007/2008	**Normal/DDoS**	**7616**	**128**
UCLA CSD Normal/DDoS	**Normal/DDoS**	**12509**	**210**

5.2. Testes e Resultados

A Figuras 3 ilustra o resultado da fase de treinamento para uma das MQRs (RR) dos sete atributos de tráfego utilizados, a média do tamanho dos pacotes (M_PAC). A análise das características dinâmicas dos demais atributos seguem a mesma metodologia e sua demonstração foi suprimida para eliminar redundância. Para o primeiro conjunto de dados (Série Normal), a Razão da Recorrência do atributo M_PAC demonstrou ser estacionária, com a taxa em torno de 25%. Já no segundo conjunto de dados, contendo apenas traços de ataques, o comportamento estacionário se manteve, mas os níveis de RR se elevaram para praticamente o dobro do observado em séries sem ataque. Para fins de detecção isto indica a viabilidade da adoção de limiares que permitam realizar a distinção entre tráfego normal e com ataques DDoS usando as MQRs.

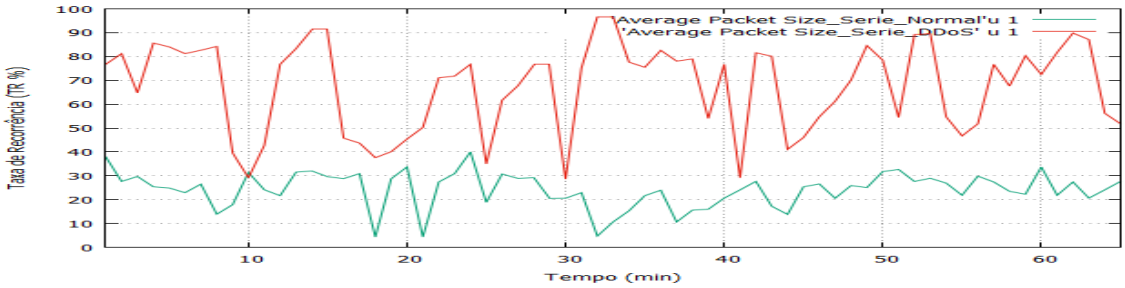


Figura 3. Taxa de Recorrência para Média do Tamanho dos pacotes (M_PAC)

As Tabelas 3 e 4 apresentam os resultados da fase de testes. O experimento avaliou a proporção de verdadeiros positivos (VP), de falsos positivos (FP) e a acurácia resultante (AC), sendo a acurácia definida como segue.

$$AC = \frac{VP}{(VP + FP)} \tag{8}$$

Para fins de comparação foram realizados testes com os algoritmos K-Means, AQR + K-Means, A-Kmeans e AQR + A-Kmeans, este último correspondendo ao método *DDoSbyAQR*, permitindo avaliar o impacto tanto da inclusão da AQR quanto da clusterização adaptativa, sendo que o valor escolhido para o parâmetro *k* (número de clusters) do K-Means e AQR + K-Means foi 4 (quatro), comparando-se com o A-Kmeans e AQR + A-Kmeans, nos quais o parâmetro *k* é calculado automaticamente, com o objetivo de ser mais eficaz. Os testes também consideraram dois conjuntos de dados, um intercalando dados das bases de dados CAIDA 2007 e CAIDA 2008 (Tabela 3) e outro intercalando dados das bases de dados UCLA CSD Normal e UCLA CSD com DDoS.

Os resultados obtidos com os dados intercalando traços com e sem ataques (Tabelas 3 e 4) indicam, para ambos conjuntos de dados, uma melhora dos classificadores quando aplicados em conjunto com a AQR. A proporção de verdadeiros positivos quando a AQR é associada ao K-Means melhorou mais de 13% (13,88% para dados da CAIDA e 18,15% para dados da UCLA CSD) e mais de 19% quando associada ao A-Kmeans (20,03% para dados da CAIDA e 19,69% para dados da UCLA CSD). A redução da proporção de falsos positivos também foi significativa (vide Tabelas 3 e 4). Como resultado, em ambos os conjuntos de dados, houve um incremento na acurácia dos classificadores, chegando a uma melhora de 10,54% para o A-Kmeans na base CAIDA. A associação do AQR+A-Kmeans também se mostrou mais eficaz do que a AQR+KMeans, demonstrando a eficácia da clusterização adaptativa no método *DDoSbyAQR*. Com dados da CAIDA a acurácia melhorou 12,42% e com dados da

UCLA CSD melhorou 8,62%, demonstrando incremento de acurácia na detecção de DDoS quando aplicado o método *DDoSbyAQR*.

Tabela 3. Resultados para o conjunto de dados CAIDA 2007/2008

ALGORITMO	AC (%)	VP (%)	FP (%)
K-Means	**70,96**	**69,23**	**28,33**
AQR+K-Means	**85,99**	**83,08**	**13,54**
A-Kmeans	**85,96**	**75,35**	**12,31**
AQR+A-Kmeans	**98,41**	**95,38**	**1,54**

Tabela 4. Resultados para o conjunto de dados UCLA CSD Normal/DDoS

ALGORITMO	AC (%)	VP (%)	FP (%)
K-Means	**84,34**	**60,63**	**11,25**
AQR+K-Means	**88,26**	**78,78**	**10,48**
A-Kmeans	**94,23**	**74,24**	**4,54**
AQR+A-Kmeans	**96,88**	**93,93**	**3,03**

6. Considerações finais

A eficácia de métodos de detecção de DDoS baseados em análise de anomalias tem sido um desafio para projetistas de algoritmos de detecção. O uso de Análise da Quantificação da Recorrência, mesmo utilizada com sucesso em outras áreas, é explorada com limitações no contexto de detecção de anomalias no tráfego de rede. Este trabalho explorou sua aplicação na detecção de DDoS, avaliando-a conjuntamente com a extração de características dinâmicas e o clusterizador adaptativo A-Kmeans.

O trabalho demonstrou que a partir de atributos que caracterizam DDoS (sete coletados a partir do tráfego de rede) é possível extrair características dinâmicas da recorrência, e que a análise destas permite incrementar a acurácia da detecção de DDoS. Salienta-se que a análise da recorrência ao permitir uma avaliação do tráfego num outro domínio, o de comportamento dinâmico da recorrência, possibilita sobrepor a influência negativa de variabilidades nos atributos do tráfego que poderiam levar a detecções errôneas.

Os experimentos demonstraram que a utilização da AQR incrementa a acurácia na identificação de ataques DDoS. Quando avaliado o benefício de classificar características dinâmicas de recorrência ao invés de classificar atributos de tráfego (avaliação dos classificadores com e sem a AQR), observa-se incrementos de até 10,54% na acurácia da detecção. Este resultado está associado a significativo incremento nos verdadeiros positivos e decremento nos falsos positivos. Porém, é consequência do método que, em tráfego aparentemente sem variações bruscas, permite observar mudanças nos padrões comportamentais da recorrência que auxiliam os classificadores a gerar agrupamentos que indicam anomalias e, em tráfego com variações bruscas não oriundas de ataques, permite observar regularidade no comportamento da recorrência.

A utilização do algoritmo A-Kmeans, um clusterizador adaptativo que calcula automaticamente o número de clusters, também demonstrou se adequar bem à detecção de DDoS baseada na AQR, tendo melhorado, no pior caso, 8,62% a acurácia da detecção quando comparado com um clusterizador não adaptativo (K-Means). A dificuldade encontrada pelo K-Means reflete a dificuldade de calibrar um clusterizador não adaptativo, o que pôde ser observado pela variabilidade de acurácia quando explorados dois conjuntos de dados com características distintas.

O método *DDoSbyAQR*, embora eficaz para a detecção de DDoS, também pode ser explorado em outros contextos de análise comportamental de rede, principalmente

pela sua característica de permitir a análise no domínio da recorrência, minimizando a influência negativa de variabilidades que causam desvios na análise de estatísticas tradicionais do tráfego.

7. Referências

Bhatia, S. K. (2004). Adaptive K-Means Clustering. *Proc. of the Int. Florida Artificial Intelligence Research Society Conference*, Miami, AAAI, pp. 695-699.

Bhaya, W., Manaa, M. E. (2014). A Proactive DDoS Attack Detection Approach Using Data Mining Cluster Analysis. Journal of Next Generation Information Technology (JNIT) Volume 5, nº 4.

Eckmann, J. P., Kamphorst, S. O., Ruelle, D. (1987). Recurrence plots of dynamical systems. Europhys. Lett., 56(5), p. 973–977.

Grossglauser, M., Bolot, J. C. (1999). On the relevance of long-range dependence in network traffic. IEEE/M Transactions on Networking, 7(5): p. 629-640.

Jeyanthi, N.; Thandeeswaran, R.; Vinithra, J. (2014). RQA based approach to detect and prevent DDoS attacks in VoIP networks, Cybernetics and Information Technologies. v. 14, p. 11-24.

Kumar, C. A.; Bhargavi, K.; Garima, J. (2012). A Note on Implementing Recurrence Quantification Analysis for Network Anomaly Detection. Defence Science Journal, [S.l.], v. 62, n. 2, p. 112-116.

Limwiwatkul, L., Rungsawang, A. (2006). Distributed denial of service detection using TCP/IP header and traffic measurement analysis. Proceedings of the IEEE International Symposium Communications and Information Technology, Sapporo, Japan, 26-29 October, p. 605–610. IEEE CS.

Marwan, N. (2003). Encounters With Neighbours - Current Developments of Concepts Based on Recurrence Plots and Their Applications. Ph.D. thesis, University of Potsdam.

Marwan, N., Kurths, J. (2005). Line structures in recurrence plots. Physics Letters A, 336(4–5), p. 349–357.

Marwan, N., Webber, C.L., Jr. (2015). Mathematical and computational foundations of recurrence quantifications. In: Recurrence Quantification Analysis: Theory and Best Practices. Springer Series: Understanding Complex Systems. Springer International Publishing, Cham, Switzerland, p. 1-41.

Nguyen, H. and Choi, Y. (2010). Proactive Detection of DDoS Attacks Utilizing k-NN Classifier in an Anti-DDos Framework. International Journal of Electrical and Electronics Engineering, Vol. 4, nº 4.

Oo, T. T., Phyu, T. (2013). A Statistical Approach to Classify and Identify DDoS Attacks using UCLA Dataset. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), Volume 2, Issue 5.

Palmieri, F., Fiore, U. (2010). Network anomaly detection through nonlinear analysis, Computers & Security, 29(7), p. 737–755.

Rahmani H., Sahli, N., Kammoun, F. (2009). Joint Entropy Analysis Model for DDoS Attack Detection. In International Conference on Information Assurance and Security, p. 267-271.

Raut, A.S., Singh, K. R. (2014). Anomaly Based Intrusion Detection-A Review. Int. J. on Network Security, Vol. 5.

Righi, M. A., Nunes, R. C. (2015). Detecção de DDoS Através da Análise da Recorrência Baseada na Extração de Características Dinâmicas. Anais do XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg 2015, p. 314-317.

Silva, J. L. C., Maia, J. E. B., Fonseca, N. L. S. (2012). Identificação de Ataques em Redes de Computadores usando Comitê de Classificadores. Anais do XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, p. 263-276.

Suresh, M., Anitha, R. (2011). Evaluating Machine Learning Algorithms for Detecting DDoS Attacks. In 4th international Conference on Advances in Network Security and Applications (CNSA), p. 441-452.

The CAIDA "DDoS Attack 2007" Dataset - < Acesso em 15 maio 2015 11:12h > https://data.caida.org/datasets/security/ddos-20070804/

The CAIDA UCSD Anonymized Internet Traces 2008 - < Acesso em 05 maio 2015 11:12h > https://data.caida.org/datasets/passive-2008/

Tsai, C. F., Hsu, Y. F., Lin, C. Y. e Lin, W. Y. (2009). Intrusion detection by machine learning: A review. Expert Systems with Applications, v. 36, n. 10, p. 11994–12000.

UCLA CSD packet traces. http://www.lasr.cs.ucla.edu/ddos/traces/public/usc

Vieira, V. J. D., Costa, S. C., Costa, W. C. A. (2012). Análise de Quantificação de Recorrência e Análise Discriminante Aplicadas à Classificação de Sinais de Vozes Saudáveis e Sinais de Vozes Patológicas. In: Anais do VII CONNEPI©2012; ISBN 978-85-62830-10-5, Palmas-TO, Brasil.

Webber, C. L., Marwan, N. (2015). Recurrence Quantification Analysis: Theory and Best Practices. Springer series: Understanding Complex Systems. Springer International Publishing, Cham Switzerland.

Willinger, W., Paxson, V., Taqqu, M. S. (1998). Self-similarity and heavy tail: structural modeling of network traffic, A Pratical Guide to Heavy Tails: Statistical Techniques and Applications. ISBN:0-8176-3951-9, p. 27-53, BirkhRäuser, Boston, USA.

Wu, Y. C., Tseng, H. R., Yang, W., and Jan, R. H. (2011). DDoS detection and traceback with decision tree and grey relational analysis. International Journal of Ad Hoc and Ubiquitous Computing, 7, p. 121–136.

Yuan, J., Yuan, R., Chen, X. (2014). Network Anomaly Detection based on Multi-scale Dynamic Characteristics of Traffic. INT J COMPUT COMMUN, ISSN 1841-9836, 9(1), p. 101-112.

Zhong, R. and Yue, G. (2010). DDoS detection system based on data mining. Proceedings of the 2nd International Symposium on Networking and Network Security, Jinggangshan, China, 2-4 April, p. 062–065. Academy Publisher.

Go With the FLOW: Fine-Grained Control-Flow Integrity for the Kernel

João Moreira[1], **Sandro Rigo**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Av. Albert Einstein, 1251 - Cidade Universitária, Campinas - SP, 13083-852

joao.moreira@lsc.ic.unicamp.br, srigo@ic.unicamp.br

Abstract. *This paper describes FLOW: a fine-grained control-flow integrity (CFI) implementation that focuses on protecting the Linux kernel. By combining source-code and binary analysis, FLOW maps valid execution paths into a fine-grained control-flow graph, which is later used to instrument the kernel with label-based CFI checks that prevent control-flow hijacking attacks. FLOW induces an average overhead of 17% on system call latency and 5% on I/O throughput, while its impact on real-world applications is* $\approx$ *1%.*

1. Introduction

Control-flow attacks target the modification of regular instruction sequences of a computer system, leading to unexpected behaviors. These attacks usually take advantage of algorithmically misplaced memory operations to overwrite control data, such as function pointers or return addresses. By modifying these data, attackers redirect control-flow to force a malicious program logic, built to obtain privileges on the target system.

Different protection mechanisms have been proposed since Stack Smashing [One 1996], the first big control-flow attack, was disclosed. These defenses, one after another, were proven inefficient as more attack methodologies were discovered in the wild or exposed by security researchers. Amongst the proposed solutions, Control-Flow Integrity (CFI) [Abadi et al. 2005] stands out as a practical and simple alternative, compatible with most existent software and hardware, capable of preventing powerful attacks to which other solutions are inefficient, such as return-oriented programming (ROP) attacks that can perform Turing-complete computation by arbitrarily chaining branch terminated instruction sequences that exist in the attacked code [Checkoway et al. 2010].

This paper presents FLOW, a practical fine-grained CFI solution focused on the Linux kernel that is capable of preventing control-flow hijacking attacks. To the best of our knowledge, this is the first fully fine-grained CFI mechanism supporting the Linux kernel. FLOW overheads are small, presenting averages of 17% on micro-benchmarks focused on stressing kernel code and 0.9% on user-land benchmarks running over the protected kernel. The contributions of this paper are: (i) we describe the implementation of FLOW, a tag based fine-grained CFI mechanism for the Linux kernel; (ii) we describe a methodology for building a specialized Control-Flow Graph used to build CFI policies; (iii) we present a study on tail call optimizations and how this feature influences on CFI mechanisms and (iv) we show, through a study of protected and unprotected code, how to solve incompatibilities due to these kinds of binary interactions.

The paper is organized as follows: Section 2 reviews related work; Section 3 brings background on CFI mechanisms that are crucial to understanding FLOW; Section

4 describes how FLOW was implemented and the challenges involved; Section 5 evaluates FLOW regarding overheads, protection efficiency and code coverage; Section 6 brings a final discussion on protections against ROP; Section 7 draws our conclusions.

2. Related Work

Control-Flow Integrity (CFI) [Abadi et al. 2005] is a technique originally proposed to ensure the correctness of control-transfer instructions on software. The general idea behind CFI consists of instrumenting the program with instructions (guards) that will validate the targets of indirect control-flow transfers right before its execution takes place. By doing that, these mechanisms are capable of identifying and deflecting sophisticated attacks such as ROP and variants [Shacham 2007, Checkoway et al. 2010, Bletsch et al. 2011]. Different ways of validating a target exist, but the most generic one consists of dereferencing the pointer used in the indirect control transfer and verifying if it is actually pointing towards pre-defined values (tags). In this sense, enhancing software with CFI consists of correctly calculating the set of valid targets for every indirect control-transfer instruction, instrumenting these instructions with guards and the destinations with the respective tags.

CFI is said to be coarse-grained [Zhang et al. 2013, Mingwei and Sekar 2013] if its branch target validation is loose, allowing returns to any site after a *call* instruction and indirect calls to the first instruction of any function. These mechanisms were proven vulnerable as valid malicious flows could be traced inside the protected applications even though the CFI restrictions were applied [Göktas et al. 2014, Carlini and Wagner 2014, Göktaş et al. 2014, Davi et al. 2014]. Motivated by that, fine-grained CFI mechanisms were developed [Tice et al. 2014, Mashtizadeh et al. 2015], tightening the control-flow target validation through only allowing returns to sites right after a *call* to the function which is returning and indirect calls to the first instruction of functions that follow a predefined rule, such as having the same prototype as the code pointer used in the call.

Despite all the previously mentioned efforts, the problem remains open as, for different reasons, these systems were not largely adopted and new control-flow hijacking attacks are constantly discovered. Also, few implementations focus specific domains, such as the operating system (OS) kernel. In this direction, while HyperSafe [Wang and Jiang 2010] implements CFI for indirect calls on hypervisors, kCoFI [Criswell et al. 2014] protects the OS Kernel in FreeBSD through the use of coarse-grained policies. The first fine-grained CFI mechanism for OSes was proposed by Ge et al. [Ge et al. 2016], but with restrictions on language features and execution implications that led this system to only support FreeBSD and MINIX.

3. Background

The first challenge in implementing an efficient fine-grained CFI scheme for a program consists in building a CFI-specific graph from where the sets of valid targets for a control-transfer will be extracted. For CFI-specific we mean a specialized callgraph that brings information about valid targets for indirect control-transfers between functions from different modules. We call this graph an *indirect control-flow graph (ICFG)*, to distinguish it from the control-flow graph in the traditional compiler theory, which is limited to mapping direct transitions between basic blocks.

Identifying valid targets for indirect transfers is not a trivial task, what raises the

bar of building an ICFG. Besides that, programming language features and compiler optimizations, such as symbol aliasing, weak functions, inlining, tail call and link-time optimizations, imply on different ICFG transformations that happen in different parts of the compilation pipeline. While some of these constraints can be tackled with more efficiency in the early compilation stages, others are better understood by analyzing the final binary, requiring a combination of analysis strategies and algorithms to achieve the tightest possible ICFG. The goal is to obtain an ICFG as consistent as possible to the control-transfers present in the binary, i.e., containing the smallest, but whole, number of valid paths to avoid unnecessary permissiveness.

In the ICFG, edges represent indirect control-transfers usually seen in code that is written in C. These transfers come from regular return instructions, indirect calls through function pointers, and indirect jumps resulted from tail call optimizations on indirect calls. Nodes in the ICFG can represent a function or a cluster of functions. This way, an edge can represent a return between two clearly defined functions, or an indirect call from a function to a cluster which represents all functions that can be invoked through that pointer. From now on, we will refer to the control-transfers due to return instructions as *back-edges* and to the other control-transfers as *forward-edges*.

While dynamic libraries are not a concern in a CFI mechanism for the Kernel, dynamically loadable modules require support. It is also important to notice that the majority of the kernel code is written in C, with a significant part written in Assembly, and that frequent interactions between code written in both languages exist. This is especially important because the underlying infrastructure used to implement FLOW has limitations on Assembly instrumentation. Details on these matters are discussed in Section 4.3.1.

4. FLOW

FLOW is a fine-grained CFI mechanism for the Linux kernel. Its implementation was mostly written as compilation passes inside the LLVM compiler [Lattner and Adve 2004], making use of its capabilities and Intermediate Representation (IR) to instrument kernel code during compilation time. FLOW supports dynamically loadable kernel modules, as long as the whole system is compiled and instrumented altogether. Due to differences between the Kernel source-code and LLVM syntax standards, the kernel code was patched with llvm-linux project [Linux Foundation] patches. Despite its focus on the Linux kernel, the principles behind FLOW are general enough to be applied on any other OS.

4.1. Design

As most of the existing CFI mechanisms, FLOW relies on two primitives, which are *tags* and *guards*. Tags mark special spots in the binary. Guards ensure that a certain pointer targets a specific spot, marked by a pre-defined tag, prior to its use. Theses primitives consist of sequences of assembly instructions exemplified in Figure 1.

Figure 1(a) shows the standard tag used for marking code throughout the whole kernel binary. The tag, which is a unique value highlighted in the figure, is encoded as the operand being copied by the `movl` instruction. The destination address of this instruction is a global variable reserved by the compiler for this purpose. As the value at this address is never read, the whole scheme benefits from write-back cache strategies.

Figure 1(b) shows a forward-edge guard example. In this snippet, the highlighted instruction `callq *%rcx` was protected by the instructions above it. In line 1, the value pointed to by `rcx` is verified through the `cmpl` instruction. This instruction will dereference the value pointed to by `rcx` plus an offset (`0x7`) and compare it to the expected tag. The offset exists because, as explained before, the tag is encoded as an operand of `movl`, whose first bytes represent the instruction opcode. The following instruction, `je`, will branch the code to line 6 if the previous comparison matches. If the comparison does not match, `je` does not branch and the code continues to line 3, that will push the attempted target address into the stack for analysis by the violation handler function, which is called in the next line. Once invoked, the violation handler reports a control-flow mismatch and returns to the next instruction, which will restore the stack and execute the original call. In some cases, an extra memory dereference may be needed before the guard if the function pointer is in memory, and not loaded into a register as in the example.

Figure 1(c) is a back-edge guard example where the highlighted `ret` instruction is protected. As a `ret` implicitly pops its target address, the return address is not yet loaded into any register, requiring it to be loaded by dereferencing of `rsp`, as seen on line 1. The following steps are analogous to the forward-edge guard: the tag is verified and if it does not match, the violation handler is invoked before executing the `ret` instruction.

The violation handler is a function written in C which is compiled and linked along with other kernel files. It can be easily adjusted to any system's needs, being suitable to report incidents, raise system crash or even reestablish valid flows whenever possible.

The careful reader may argue that this scheme is not efficient in terms of branch prediction or code size. On a legit program flow, it forces a branch to skip the violation handler invocation every time an indirect call happens. Also, by using a single basic block to invoke the violation handler, instead of multiple similar basic blocks for every indirect call, it is possible to save space. Although this is true, using just one basic block, positioned anywhere else in the code that is not between the comparison and the indirect call, would limit the capabilities of the violation handler for two reasons: (i) there would be no reliable way of pushing the target of the instruction that triggered the violation into the stack because indirect calls may use different registers and memory locations to hold its pointers and (ii) as the violation handler is called right before the control-transfer, the address pushed into the stack when it is called can be used as a reference to identify the instruction responsible for the violation, reducing the complexity on reporting the violation. Experiments with both approaches on the SPEC 2006 [Henning 2006] benchmark showed that the adopted strategy only degrades performance by an average of 0.9%, which is an acceptable overhead given its benefits.

The original CFI scheme [Abadi et al. 2005] also considered the use of a shadow stack to validate return addresses before using them. This enhancement would harden return address protection by ensuring that only one amongst many different call sites to the same function is considered a valid return target. Although slightly more loose, FLOW does not make use of shadow stacks for two reasons: (i) Shadow Stacks are too costly in terms of efficiency and incur significant overheads that are prohibitive, especially in the context of kernel code; (ii) A memory corruption bug in the kernel context allows not only overwriting code pointers but also the corruption of values inside the shadow stack - it is hard to protect the shadow stack against privileged exploits in kernel-land.

(a) tag

```
1 movl $0x3a66ac66,0xffffffff8585e000
```

(b) forward-edge guard

```
1 cmpl  $0x3a66ac66,0x7(%rcx)
2 je    <6>
3 push  %rcx
4 callq <violation_handler>
5 pop   %rcx
6 callq *%rcx
```

(c) back-edge guard

```
1 mov   (%rsp),%rdx
2 cmpl  $0x3a66ac66,0x7(%rdx)
3 je    7
4 push  %rdx
5 callq <violation_handler>
6 pop   %rdx
7 retq
```

Figure 1. Example Tags and Guards

4.2. FLOW Pipeline

FLOW takes advantage of information available during compilation time, such as pointer types and function attributes, to build an ICFG of valid control transfers. Part of the toolset consists of binary analysis tools, ICFG construction enhancement, and optimizations. The tools are organized as a compilation pipeline composed of three different stages:

- **Stage 1: Exploratory Compilation:** In this stage the whole kernel is compiled and information regarding control transfers is extracted. Besides compiling the code, this stage also builds a data structure to describe kernel functions, indirect transfers, and groups of functions which are suitable as destinations for the same indirect transfers for having similar prototypes. This stage also instruments every function with a unique identifier and generates a map of all aliases and aliasees present in the source code, both used for solving ICFG mismatches and source code ambiguities due to weak symbols, aliasing, and inlining.
- **Stage 2: ICFG Closure:** This stage consists of a series of analysis made over the data structure and the binary generated in the previous stage. It is composed of four different tools: (i) **Dump Shrinker**, which removes information that does not concern control-flow from the first-stage generated kernel dump, making it easier and faster to parse; (ii) **CFI merge**, that merges information generated by multiple compilation threads into a single file; (iii) **Assembly Node Mapper**, that creates ICFG nodes for the Assembly functions; and (iv) **Direct Edge Mapper**, that builds ICFG information regarding direct edges between functions, task that cannot be performed during compilation time due to linking restrictions.
- **Stage 3: Final Compilation:** Instruments the kernel with CFI guards and tags.

After these stages, the generated kernel image is ready to be installed and run.

4.3. ICFG Construction and CFI Binding

FLOW incrementally builds a data structure that holds information about the ICFG. For every module compiled, the functions are parsed and stashed as an ICFG node. Indirect calls through function pointers are stashed as an ICFG edge. Function pointer prototypes are used to create an ICFG cluster, which is a special kind of node that represents many functions, for every respective prototype. Figure 2(a) shows an example source code which is written in a given *example.c* file. If this code is processed throughout *Exploratory Compilation* and *ICFG Closure* (FLOW's pipeline stages 1 and 2), it will generate a data structure with the contents shown in Figure 2(b).

The indirect call in function bar (line 8 in Figure 2(a)) is represented by the indirect call edge (*icall*) in the ICFG Edges table (Figure 2(b)). The *From* field contains the id of the node representing the function `i32 bar(i32)` (`7d63f629`) and the *To* field contains the `i32 (i32)` cluster id (`6a8597ea`), because every function with this respective prototype is represented by this cluster, thus is considered valid. This relationship can be seen in Figure 2(c), where the gray area represents the cluster which holds both nodes `i32 foo(i32)` and `i32 bar(i32)`. In this figure, the indirect call is represented by the dashed edge, which points to the respective cluster of allowed targets.

Figures 2(b) and 2(c) also represent the direct call from `i32 bar(i32)` to `void dummy(i32)`. On FLOW's pipeline, this edge is only discovered on Stage 2. Such information is not extracted during compilation since it is easier to solve function name ambiguities through binary analysis once all the modules were already compiled and linked.

(a) source-code

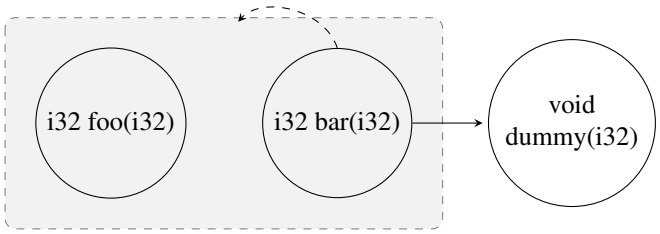
```
1  int foo(int x){
2    return x*x;
3  }
4  int bar(int y){
5    int(*f)(int);
6    f = &foo;
7    dummy(30);
8    return 7 * f(y);
9  }
10 void dummy(int z){
11   while(1){ };
12 }
```

(b) resulting ICFG data structure

ICFG Nodes				
Name	**Prototype**	**Module**	**ID**	**ret TAG**
foo	i32 (i32)	example.c	290f2fd5	1dc2aaf0
bar	i32 (i32)	example.c	7d63f629	6e28b9d1
dummy	void (i32)	example.c	6ba8458b	164e44a8

ICFG Clusters			
Prototype	**ID**	**call TAG**	**ret TAG**
i32 (i32)	6a8597ea	69e1b040	46068a5c

ICFG Edges			
Type	**ID**	**From**	**To**
icall	7dcdc019	7d63f629	6a8597ea
dcall	7728cc01	290f2fd5	6ba8458b



(c) resulting ICFG

Figure 2. ICFG Construction

On what concerns matching guards and tags, the algorithm for building guards works as follows. (i) Forward-edges: the algorithm checks the pointer type and acquires the respective cluster from the ICFG data structure, depicted in Figure 2(b). Then, the `cmp` instruction from the guard is generated with the *call TAG* for this cluster, as exemplified on line 1 of Figure 1(b). (ii) Back-edges: the algorithm first searches a cluster with a prototype that matches the function being instrumented. If such cluster exists, the `cmp` instruction is generated with the *ret TAG* for this cluster. If the cluster does not exist, this means that this function will never be called through a pointer and it is safe to use an exclusive *ret TAG* for it, so the `cmp` instruction is generated with the node's *ret TAG*. Tags and guards for back-edges are shown in Figure 1(c).

Similarly to what is done for placing the guards, the algorithm that places the tags after each `call` first checks if the transfer is direct or indirect. If it is an indirect transfer, the algorithm uses the *ret TAG* of the cluster. If it is a direct call, the algorithm checks if there is a cluster that matches the prototype of the called function. If such cluster exists, then its *ret TAG* is used. If no cluster matches the directly called function, then the *ret TAG* placed is the one respective to the node. Only functions that can be called indirectly must have a *call TAG*. Again, the algorithm processing the function verifies if it has a correspondent cluster and, if it does, places the *call TAG* on the function's prologue.

Both *ret* and *call TAGS* are mutually unique to avoid unexpected edges.

4.3.1. Assembly Nodes

A meaningful part of FLOW's code was implemented as optimization passes in the LLVM compiler infrastructure. LLVM translates source-code into an IR to which it applies optimizations and then translates into machine code. Assembly code is not translated into IR by LLVM, being directly translated into machine code. Because of that, the passes responsible for source-code analysis and CFI instrumentation are unable to process Assembly functions, leaving part of the code apart from FLOW.

The first problem generated by Assembly functions concerns the ICFG construction. By the end of FLOW's *Exploratory Compilation*, the node list only includes C functions and not Assembly functions. To fix that, FLOW runs a tool called **Assembly Node Mapper**, that parses the binary generated by this stage and maps which functions were not instrumented with unique identifiers, meaning that these are Assembly functions. These functions are added to the node list with a special Assembly flag.

After mapping all the Assembly nodes, it is possible to run the tool **Direct Edge Mapper** that identifies all edges. If this tool was executed previously, some edges would have dangling targets as some nodes would not have been identified yet.

The second problem regarding Assembly code arises from the interaction between Assembly and C code. While there is no problem in directly calling Assembly functions from C code, it is not straightforward to call C code from Assembly functions. This happens because return instructions originated from C code are instrumented with back-edge guards and will check for a tag which is not present in the Assembly code since it was not instrumented. Similarly, whenever C code indirectly calls Assembly functions, the forward-edge guard will search for a tag which is again not present, raising another violation. Such false-positives were fixed through a white-list embedded into the violation handler function. The white-list was built upon information extracted from the binary which allowed the identification of interaction sites between Assembly and C code.

The last problem due to this constraint is the difficulty of protecting edges that originate from Assembly code. As FLOW is unable to add guards to returns and indirect calls in such functions, these edges are left unprotected. Protection coverage and exploitation feasibility due to this limitation are discussed in detail in Section 5.3.

4.3.2. Tail Call Optimizations

Compilers apply Tail Call Optimizations (TCO) by replacing a `call` instruction positioned at the end of a function by a `jmp` instruction. Whenever applied, this optimization will make the callee reuse the stack of the caller function and, because of that, whenever the callee returns, it will return directly to the function underneath the caller.

This optimization poses a special challenge to CFI mechanisms because the callee's return guard will check for the callee's tag in the return address, which is actually instrumented with the caller's tag, as depicted in Figure 3. In this example, when the optimization is not applied (Figure 3(a)) the tags and the guards match. When the optimization is applied (Figure 3(b)), the function *anubis_crypt* will check for its tag before returning, but the instruction right after the address to which it will return contains the tag for the function *anubis_encrypt*.

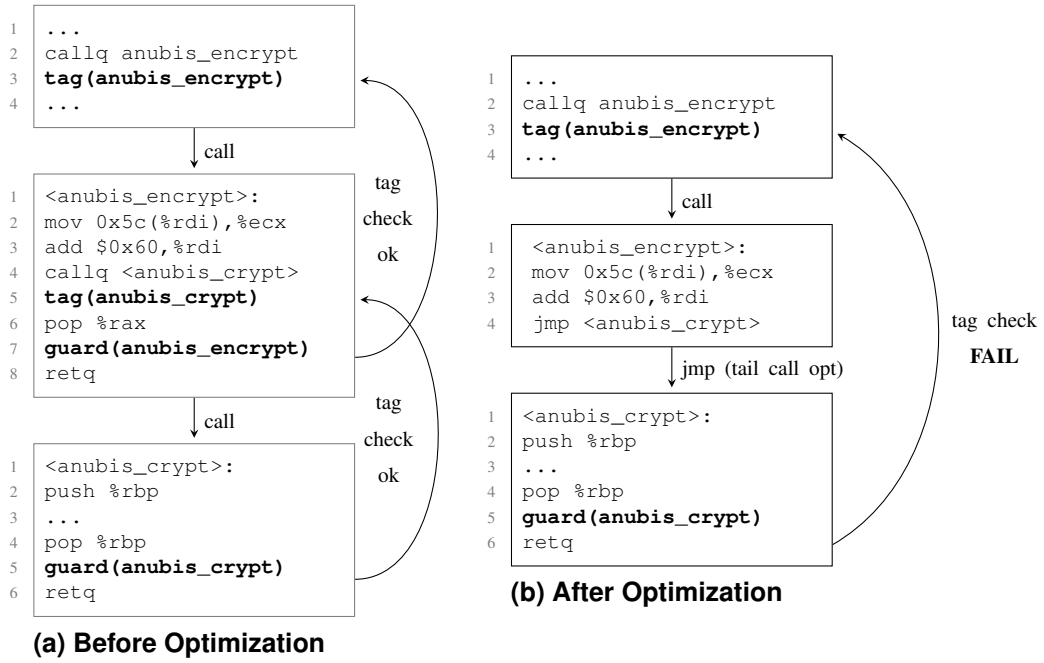


Figure 3. Assembly Before/After Tail Call Optimization

Tackling this issue by validation inside the violation handler is not worthy. A violation handler invocation requires extra `push`/`pop` instructions, plus at least one `call` and one `ret`. Even if the total cost of the computation executed inside the violation handler is ignored, these extra instructions already overweight TCO benefits.

A second solution would be collapsing the ICFG Nodes that represent the callee and the caller, so they would have similar tags. Although this solution would work, it would imply in a less-fine-grained ICFG and would only be justifiable if the benefits from applying TCO to the Kernel were significantly meaningful.

To evaluate TCO impact on kernel performance, the vanilla version of the Linux kernel was compiled with and without TCO. We used the kernel specific micro-benchmark LMBench [McVoy and Staelin 1996] in this experiment. An average performance overhead of 5% was generated by disabling TCO. We considered this as a low

impact, especially considering a kernel-specific benchmark, what implies in even smaller overheads for user-land software. For this reason, we incorporated this overhead to the overhead generated by FLOW in our evaluation and proceeded its development without TCO support. Alternatives for incorporating TCO stands as future work.

5. Evaluation

We evaluate FLOW from two different perspectives: performance and security. First, we assess the performance overhead implied by using FLOW with two well-known benchmarks, LMBench [McVoy and Staelin 1996] and SPEC2006 [Henning 2006]. Second, we evaluate FLOW coverage and its capabilities as a security enhancer through verifying its efficiency on defeating control-flow hijacking attacks.

The system used for measurement on all tests was an Intel(R) Core(TM) i7-6700 8 core CPU @ 3.40GHz, with 32GB RAM memory, running Debian Linux with GNU kernel v3.19.0 on VMware Workstation 12 Player. Every program unit inside each benchmark was executed 10 times, and the results are averages of the observed numbers. Also, it is important to highlight that the protection was only applied to the system's kernel code, thus, all benchmarks were compiled with GCC v4.9.2 without CFI instrumentation.

To clearly separate FLOW's overhead from the overhead generated by disabling TCO, we also tested an unprotected kernel without TCO, allowing the performance comparison of three different subjects: (i) a regular compilation of the Linux vanilla kernel in minimal configuration (Vreg), (ii) the same kernel compiled without TCO (VnoTCO) and (iii) the same kernel instrumented with FLOW and also without TCO (Vflow).

5.1. LMBench

To assess FLOW effects on kernel performance, we used the LMBench micro-benchmark [McVoy and Staelin 1996] . LMBench stresses the OS functionalities, allowing the observation of overheads on tasks such as performing system calls or handling page-faults.

While testing LMBench, we focused on the assessment of latencies and communication bandwidth overheads. Specifically, we evaluated the latencies of OS capabilities for executing: null syscalls; I/O critical syscalls (which are read/write, fstat and select); open/close syscalls; the installation of a signal handler; process creation followed by exit, execve and /bin/sh; context switching between processes; select syscall on 100 file descriptors; page fault handling and inter-process communication with socket and pipe. The bandwidth was measured while communicating through pipe, unix sockets (AF_UNIX) and TCP sockets. The results of these tests can be seen in Figure 4, which shows latency and bandwidth overheads while comparing VnoTCO and Vflow to our baseline, Vreg.

We measured an average latency overhead of 17.75% while comparing Vflow against Vreg. When comparing VnoTCO against Vreg, the overhead was 6.33%, making it clear that there is room for improving a future version of FLOW that can benefit from being TCO-compatible. Regarding Vflow, the smallest overheads appeared in *null syscall*, which did not show discernible overheads, and in *fork+exec*, which had a 9%overhead. On the other hand, a *select* operation on 100 file descriptors (*select 100 fd's*) presented the biggest overhead, which is 42%. The high overhead for *select* is due to a bigger than average number of CFI checks performed by the invoked syscall.

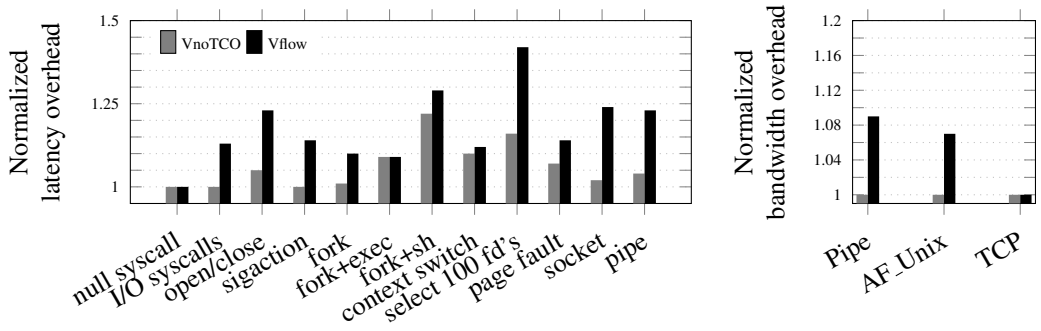


Figure 4. FLOW performance on LMBench

SPEC INT 2006 Vflow x Vreg overheads					
perlbench	0.5%	**bzip2**	0.9%	**gcc**	1.1%
mcf	0.4%	**gobmk**	1.2%	**hmmer**	1.5%
sjeng	1.4%	**libquantum**	1.3%	**h264ref**	0.8%
omnetpp	0.8%	**astar**	0.3%	**xalancbmk**	0.6%
				average	0.9

Figure 5. SPEC INT 2006 overheads

The bandwidth overheads describe how less efficient the communication was, meaning that less data was transported in the same period of time. Vflow presented an average bandwidth overhead of 5.33%, while VnoTCO did not present any discernible overhead at all. The biggest overhead (9%) was verified on *pipe*, while *AF_Unix* presented an overhead of 7% and *TCP* did not present any significative overhead.

As shown, FLOW performs well on the execution of LMBench, presenting low overhead on latencies for the execution of system calls and not harming significantly the communication bandwidth. When a comparison is possible, FLOW performs better or similarly to other existing CFI mechanisms for OSes: kCoFI [Criswell et al. 2014] presented overheads for LMBench applications that ranged from 100% to 250%.

5.2. SPEC 2006

Although LMBench shows meaningful numbers, computer systems have most of their useful work done by user-land software. We used the SPEC 2006[Henning 2006] benchmark to assess FLOW in this scenario. SPEC is composed of user-land CPU-intensive applications, therefore, the amount of execution time spent on the kernel code is reduced. All programs in the SPEC-INT 2006 suite were executed using the reference input.

When comparing the Vreg against VnoTCO, the average overhead was of 0.3%, with a maximum of 0.7% for *omnetpp*. Applications *perlbench*, *gcc*, and *mcf* presented overheads below 0.1%, what was considered indiscernible. The full comparison between Vreg and Vflow is presented in Figure 5. As shown, Vflow induced an average overhead of 0.9%, with a maximum of 1.5% for *hmmer*, confirming the low overhead expectancy.

5.3. Security and Coverage

FLOW improves previously existent coarse-grained CFI mechanisms. For being a fine-grained, FLOW has a smaller set of valid execution paths, imposing harder exploitation

barriers. If the same tested kernel was instrumented with both approaches, the function pointers with the prototype `void ()` would have 1130 valid target functions when instrumented with FLOW, while on a coarse-grained system this permissiveness would be $\approx$ 2525% higher, allowing it to target the beginning of any function in the kernel, a total of 28537 valid targets. On the compiled system, the function prototype `void ()` is the most common one, thus 1130 functions is the biggest allowed target functions set on the system, what makes this proportion even larger when comparing other prototypes. The average number of valid targets for a function pointer on FLOW is $\approx$10.6.

As explained before, Assembly is not translated into IR by LLVM and, thus, is not instrumented by FLOW. Statistics on how much of the kernel was left unprotected were grabbed through binary analysis. The kernel binary has 28537 functions from which 351 are Assembly functions, representing $\approx$ 1.2% of the code. Throughout the binary, we had 28865 return instructions from which 85 were not protected, representing $\approx$ 0.3% of the total `ret` instructions. This proportion is even smaller for indirect calls as 8 unprotected indirect calls out of 6053 were found, representing less than 0.2% of the total.

To evaluate the effectiveness of FLOW against real-world control-flow hijacking attacks, we used the ROP exploit for CVE-2013-2094 [Kemerlis et al. 2014], which targets Linux v3.8 (x86-64). We first verified that the exploit was successful on the respective kernel, and then tested it on the same kernel protected with FLOW. As expected, the code-reuse attack failed; the ROP payload used by the exploit relied on pre-computed gadget addresses, none of which corresponded to a valid control-flow transfer under FLOW.

These results show that a very significant part of the code is covered by our mechanism, imposing significant difficulties to attackers intended to compromise a system with such protection. Overcoming FLOW requires an attacker to exploit these specific targets, what is unlikely to be possible as attacks require very specific contexts to be deployable and these are now heavily limited by such a small window of unprotected instructions.

5.4. Code-size

Disabling TCO and instrumenting the binary added an overhead of 15.9% on code-size between Vflow to Vreg. No discernible overheads were seen between VnoTCO to Vreg.

6. Discussion

Recently, it was shown that fine-grained CFI may be vulnerable to attacks if very specific circumstances are met. Control-Flow Bending (CFB) [Carlini et al. 2015] is a control-flow attack built on top of a data-only attack. This attack uses a technique called *printf-oriented programming*, which is strongly based on an old attack called *format strings* [Scut and Teso 2001] to perform Turing-complete computation. CFB also relies on the creation of execution loops, which is performed by the corruption of return pointers towards valid targets that will later take the execution towards the exploited function. Control Jujutsu [Evans et al. 2015] demonstrates how indirect calls are still vulnerable even under fine-grained CFI. This work describes a hypothetical fine-grained CFI whose valid indirect-call targets were gathered through the DSA [Lattner et al. 2007] algorithm. Later, the paper shows a specific pointer corruption that enables system exploitation.

In spite of these attacks towards CFI, we argue that our scheme remains relevant as (i) the requirements for both attacks are very specific and tight, hardly met in real-world

scenarios and unlikely to be found in the kernel context; (ii) if extended with a shadow stack mechanism, FLOW would be able to deflect CFB, so it stands as an important first step in this direction; and (iii) for using a different algorithm to build its ICFG, FLOW does not allow the edges used in the Control Jujutsu attack, proving that using extensible and modular techniques to build an ICFG is a good strategy to tackle the problem.

Branch frequency-based mechanisms were also proposed as a solution against ROP [Pappas et al. 2013, Cheng et al. 2014]. These defenses monitor a ratio of instructions executed in between branches. As ROP gadgets are small sequences of instructions followed by a control-transfer and launching ROP attacks consists in chaining the execution of these gadgets, it is possible to monitor the branching ratio, define acceptable thresholds and identify the anomalous behaviors. Even thought optimizations have been proposed to these schemes [Rubens et al. 2015], the problem remains open as they (i) present prohibitive overheads, much higher than compiler-based solutions, (ii) are vulnerable to gadget chaining obfuscation attacks [Göktas et al. 2014, Carlini and Wagner 2014, Davi et al. 2014, Göktaş et al. 2014] and (iii) do not add extra defenses and are also vulnerable to the two attacks previously described against CFI.

We also would like to add that FLOW differs from the fine-grained CFI implementation proposed by Ge [Ge et al. 2016]. This work covers FreeBSD and MINIX and relies on techniques such as the conversion of indirect calls into direct calls. Its ICFG is built through taint analyses that impose restrictions on the use of language features such as operations on function pointers. FLOW also brings fine-grained CFI for the kernel but differs from Ge's work by (i) supporting the Linux kernel, which is a larger and more diverse code base; (ii) not relying on code modifications, such as pointer conversion, which can harm system design, and (iii) using a different clustering method, not dependent of analyses that restricts the use of language features and that was designed to be easily extended by methods such as static control-flow analysis [Lattner et al. 2007].

7. Conclusions

In this paper, we presented FLOW, a CFI solution for the Linux kernel. To develop a CFI mechanism capable of protecting an OS, specific challenges were faced, such as defining a control-flow graph that describes valid flows without much relaxation and is compatible to circumstances imposed by the domain, such as interactions between C and Assembly code, symbol aliasing, weak linking, inlining and other compiler optimizations.

FLOW proved to be an efficient solution against control-flow attacks and presented low overheads, showing an average of 17% on kernel micro-benchmarks. The paper also shows that this overhead is significantly amortized for user-land applications, reaching an average of 0.9%, when measured in the SPEC 2006 benchmark.

References

Abadi, M., Budiu, M., Erlingsson, U., and Ligatti, J. (2005). Control-flow integrity. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, New York, NY, USA. ACM.

Bletsch, T., Jiang, X., Freeh, V. W., and Liang, Z. (2011). Jump-oriented programming: A new class of code-reuse attack. In *Proceedings of the 6th ACM Symposium on*

Information, Computer and Communications Security, ASIACCS '11, pages 30–40, New York, NY, USA. ACM.

Carlini, N., Barresi, A., Payer, M., Wagner, D., and Gross, T. R. (2015). Control-flow bending: On the effectiveness of control-flow integrity. In *24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C. USENIX Association.

Carlini, N. and Wagner, D. (2014). Rop is still dangerous: Breaking modern defenses. In *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA. USENIX Association.

Checkoway, S., Davi, L., Dmitrienko, A., Sadeghi, A.-R., Shacham, H., and Winandy, M. (2010). Return-oriented programming without returns. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, New York, NY, USA. ACM.

Cheng, Y., Zhou, Z., Yu, M., Ding, X., and Deng, R. H. (2014). Ropecker: A generic and practical approach for defending against rop attacks. In *NDSS*. The Internet Society.

Criswell, J., Dautenhahn, N., and Adve, V. (2014). Kcofi: Complete control-flow integrity for commodity operating system kernels. In *2014 IEEE Symposium on Security and Privacy*.

Davi, L., Sadeghi, A.-R., Lehmann, D., and Monrose, F. (2014). Stitching the gadgets: On the ineffectiveness of coarse-grained control-flow integrity protection. In *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA. USENIX Association.

Evans, I., Long, F., Otgonbaatar, U., Shrobe, H., Rinard, M., Okhravi, H., and Sidiroglou-Douskos, S. (2015). Control jujutsu: On the weaknesses of fine-grained control flow integrity. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, New York, NY, USA. ACM.

Ge, X., Talele, N., Payer, M., and Jaeger, T. (2016). Fine-grained control-flow integrity for kernel software. In *IEEE European Symposium on Security and Privacy 2016*, Euro S&P, Washington, USA. IEEE Computer Society.

Göktas, E., Athanasopoulos, E., Bos, H., and Portokalidis, G. (2014). Out of control: Overcoming control-flow integrity. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, Washington, DC, USA. IEEE Computer Society.

Göktaş, E., Athanasopoulos, E., Polychronakis, M., Bos, H., and Portokalidis, G. (2014). Size does matter: Why using gadget-chain length to prevent code-reuse attacks is hard. In *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA. USENIX Association.

Henning, J. L. (2006). SPECCPU 2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4).

Kemerlis, V. P., Polychronakis, M., and Keromytis, A. D. (2014). ret2dir: Rethinking Kernel Isolation. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA. USENIX Association.

Lattner, C. and Adve, V. (2004). LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code*

Generation and Optimization: Feedback-directed and Runtime Optimization, CGO '04, Washington, USA. IEEE Computer Society.

Lattner, C., Lenharth, A., and Adve, V. (2007). Making Context-Sensitive Points-to Analysis with Heap Cloning Practical For The Real World. In *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'07)*, San Diego, California.

Linux Foundation. LLVMLinux. `http://llvm.linuxfoundation.org/`. Accessed 2016-05-22.

Mashtizadeh, A. J., Bittau, A., Boneh, D., and Mazières, D. (2015). Ccfi: Cryptographically enforced control flow integrity. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, New York, NY, USA. ACM.

McVoy, L. and Staelin, C. (1996). Lmbench: Portable tools for performance analysis. In *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference*, ATEC '96, Berkeley, CA, USA. USENIX Association.

Mingwei, Z. and Sekar, R. (2013). Control flow integrity for cots binaries. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, Washington, D.C. USENIX.

One, A. (1996). Smashing the stack for fun and profit. *Phrack*, 7(49).

Pappas, V., Polychronakis, M., and Keromytis, A. D. (2013). Transparent rop exploit mitigation using indirect branch tracing. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, Washington, D.C. USENIX.

Rubens, E., Tymburibá, M., and Pereira, F. (2015). Inferência estática da frequência máxima de instruções de retorno para detecção de ataques rop. In *Simpsósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, SBSEG XV.

Scut and Teso, T. (2001). Exploiting format string vulnerabilities. `http://julianor.tripod.com/bc/formatstring-1.2.pdf`. Accessed 2016-05-22.

Shacham, H. (2007). The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, New York, NY, USA. ACM.

Tice, C., Roeder, T., Collingbourne, P., Checkoway, S., Erlingsson, Ú., Lozano, L., and Pike, G. (2014). Enforcing forward-edge control-flow integrity in gcc & llvm. In *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA. USENIX Association.

Wang, Z. and Jiang, X. (2010). Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *2010 IEEE Symposium on Security and Privacy*.

Zhang, C., Wei, T., Chen, Z., Duan, L., Szekeres, L., McCamant, S., Song, D., and Zou, W. (2013). Practical control flow integrity and randomization for binary executables. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, Washington, DC, USA. IEEE Computer Society.

Agrupamento de malware por comportamento de execução usando lógica fuzzy

Lindeberg Leite[1], **Daniel G. Silva**[2], **André Grégio**[3]

[1] Departamento de Engenharia Elétrica – Universidade de Brasília
Brasília, DF / Polícia Federal

[2]Departamento de Engenharia Elétrica – Universidade de Brasília
Brasília, DF.

[3]Departamento de Informática – Universidade Federal do Paraná
Curitiba, PR.

lindeberg.lpl@dpf.gov.br, danielgs@ene.unb.br, gregio@inf.ufpr.br

Abstract. *The threat of malware variants continuously increases. Several approaches have been applied to malware clustering for a better understanding on how to characterize families. Among them, behavioral analysis is one that can use supervised or unsupervised learning methods. This type of analysis is mainly based on conventional (crisp) logic, in which a particular sample must belong only to one malware family. In this work, we propose a behavioral clustering approach using fuzzy logic, which assigns a relevance degree to each sample and consequently enables it to be part of more than one family. This approach enables to check other behaviors of the samples, not visualized in conventional logic. We compare the chosen fuzzy logic algorithm — Fuzzy C-Means (FCM) — with K-Means so as to analyze their similarities and show the advantages of FCM for malware behavioral analysis.*

Resumo. *A ameaça de variantes de malware aumenta continuamente. Várias abordagens para agrupamento de malware já foram aplicadas para entender melhor como caracterizar suas famílias. Destas, a análise comportamental pode usar tanto métodos de aprendizado supervisionado como não-supervisionado. Neste caso, a análise é comumente baseada em lógica convencional, onde um dado exemplar deve pertencer a apenas uma família. Neste trabalho, propõe-se uma abordagem de agrupamento comportamental por lógica fuzzy, que atribui um grau de relevância à cada exemplar e permite que este pertença a mais de uma família. Essa abordagem possibilita verificar outros comportamentos das amostras, não visualizados na lógica convencional. Compara-se o algoritmo escolhido — Fuzzy C-Means (FCM) — com o algoritmo K-Means para analisar similaridades e mostrar as vantagens do FCM na análise comportamental de malware.*

1. Introdução

A quantidade de *malware* vem aumentando significativamente, ano após ano. De acordo com a McAfee [Intel Security 2014], mais de 307 novas ameaças surgem a cada minuto,

ou seja, mais de cinco por segundo. No terceiro trimestre de 2014, já eram contabilizados mais de 40 milhões de novos tipos de *malware*. Esses exemplares, embora considerados "novos" ou "únicos", nada mais são do que mutações de *malwares* já existentes. Pequenas variações no código, como mudanças em uma lista de endereços IP para varredura ou comunicação, em *strings* ou o uso de mecanismos de ofuscação (*packers*) fazem com que o exemplar seja considerado novo em relação aos demais. Os exemplares decorrentes de mutação de outros são conhecidos como *variantes*. As variantes, responsáveis pela elevação na quantidade de *malware* em atuação, dificultam a criação de assinaturas de detecção e facilitam a tarefa dos atacantes. Por outro lado, estimula-se a busca por técnicas capazes de detectar, classificar e/ou agrupar tais exemplares de maneira eficiente e eficaz.

Os fabricantes de antivírus (AV) criam assinaturas estáticas usando procedimentos majoritariamente manuais, pois é preciso que um analista humano descompile ou desmonte o código a fim de encontrar um padrão que, ao mesmo tempo em que detecte um *malware*, não gere falsos-positivos que prejudiquem a experiência do usuário. Devido a isso, grande parte dos exemplares de *malware*, incluindo os mais complexos, permanece não detectada por um tempo além do adequado. Por exemplo, estima-se um típico intervalo de 54 dias entre a disseminação de um dado *malware* e sua detecção por algum AV; 15% das amostras passam desapercebidas por 180 dias [Damballa 2009]. Por isso, torna-se essencial desenvolver métodos automatizados para detectar e agrupar exemplares de *malware* desconhecidos, ou seja, que ainda não pertencem aos bancos de dados de assinaturas dos AV [Salehi et al. 2012].

A dissecação de um exemplar de *malware* para melhor compreensão de seu modo de atuação (disseminação, infecção, ocultação de rastros, comunicação com o atacante, entre outras ações) envolve formas distintas de análise: (*i*) a análise estática atua diretamente no binário sem a necessidade de execução, ou seja, obtém-se informações do cabeçalho, verifica-se pela presença de *packers*, descompila-se, quando possível, para se avaliar suas instruções e se tentar identificar blocos suspeitos, busca-se por *strings* que correspondam a endereços IP, URLs, e-mail, etc.; (*ii*) a análise dinâmica consiste na observação da execução do *malware* em um ambiente controlado (*sandbox*) a fim de se extrair seu comportamento, isto é, quais são as atividades realizadas no nível do sistema operacional e da rede que podem ser utilizadas para definir como a infecção ocorre; (*iii*) a combinação de ambos os tipos, a fim de se valer das vantagens providas tanto pela análise estática quanto pela dinâmica.

Além da análise estática, dinâmica ou mista para detecção, há uma preocupação crescente com o agrupamento de exemplares na família correta. Com isso, entende-se melhor as características de cada grupo e aumenta-se a chance de detecção. Ademais, descobrir o rótulo adequado de um determinado artefato é fundamental para conhecer antecipadamente sua intenção maliciosa, bem como seu método de infecção e propagação. Em uma abordagem investigativa, saber o rótulo correto direciona o esforço do analista para as atividades maliciosas da amostra, inclusive para antever contramedidas. Nesse sentido, técnicas de aprendizado de máquina podem ser aplicadas para se realizar o processo de agrupamento (em inglês, *clustering*) de forma automatizada. Tradicionalmente, tais tentativas envolvem a aplicação de algoritmos fundamentados na lógica tradicional ou *"crisp"*, a qual se associa à teoria clássica de conjuntos: sob esta ótica, um exemplar de *malware* pertence exclusivamente a uma família, dentre todas as possíveis. No entanto,

com a sofisticação dos processos de criação de *malware* usando kits de faça-você-mesmo, os exemplares produzidos passaram a empregar intensivamente o reuso de código. Dessa forma, o *malware* atual não é mais limitado a exibir o comportamento específico de uma classe pré-determinada, podendo carregar ao mesmo tempo características de várias classes. Logo, faz-se necessário um método capaz de retratar, de forma mais fidedigna, a dinâmica comportamental de um *malware*. Neste artigo, propõe-se um método para agrupamento de exemplares de *malware* pelo seu comportamento de execução, o qual tem sua base na lógica fuzzy. Realizou-se um estudo comparativo entre um método que emprega a lógica *crisp* — *K-Means* — e outro que adota a lógica fuzzy — *Fuzzy C-Means* (*FCM*) [Bezdek et al. 1984]. Este trabalho, portanto, fornece as seguintes contribuições:

- Aplicação de um novo método para agrupar exemplares de *malware* de forma mais flexível que os classificadores tradicionalmente utilizados;
- Um processo de rotulação eficaz e realista, pois considera os diversos comportamentos exibidos por um *malware* moderno. Nos casos em que há *clusters* sem rótulos, o *FCM* apresentou vantagens na atribuição de um nome ao grupo devido à sua matriz de pertinência;
- Avaliação comparativa entre os algoritmos de agrupamento *K-Means* (lógica tradicional) e *FCM* (lógica *fuzzy*), permitindo a análise dos resultados para verificação de semelhanças entre eles, bem como a análise do tempo de execução gasto por algoritmos de diferentes paradigmas aplicados ao mesmo *dataset*. Constatou-se um alto grau de equivalência entre os *clusters* produzidos por ambos e mensurou-se o tempo de execução dos experimentos realizados com eles.

O restante deste artigo está organizado da seguinte forma: a Seção 2 discute os trabalhos relacionados; a Seção 3 apresenta as etapas da análise comportamental de *malware* e a extração de características para posterior agrupamento; a Seção 4 introduz o processo de aprendizado de máquina e os algoritmos usados; a Seção 5 mostra o resultado dos experimentos realizados e, por fim, a Seção 6 expõe as considerações finais e os trabalhos futuros.

2. Trabalhos Relacionados

Nesta seção, são apresentados alguns trabalhos correlatos que envolvem a aplicação de algoritmos de aprendizado de máquina na classificação ou agrupamento de exemplares de *malware*.

[Salehi et al. 2012] criaram um método para detecção de *malware* baseado nas chamadas de função de API e seus argumentos. Primeiramente, efetuou-se o processo de coleta de exemplares, os quais foram divididos em dois conjuntos (*malware* e não *malware*), nos quais 385 deles eram programas nativos encontrados no sistema operacional Windows ou outras ferramentas consideradas benignas e 826 eram programas considerados maliciosos. Esses arquivos foram obtidos de [Sami et al. 2010]. Os binários executaram em um sistema operacional Windows virtualizado no VMware e o comportamento foi monitorado pelo `WINAPIOverride32`. Foram monitoradas 126 funções de API das seis DLLs (*Dynamic-Link Library*) mais importantes do Windows: `advapi32.dll`, `kernel32.dll`, `ntdll.dll`, `user32.dll`, `wininet.dll` e `ws2_32.dll`. Além da monitoração das funções de API, foram observados também seus argumentos. O resultado dessa monitoração foi convertido para o formato ARFF e,

no processo de classificação, a melhor acurácia (97%) foi obtida com o algoritmo *Functional Trees* (FT) do WEKA.

[Mangialardo 2015] elaborou uma técnica integrada de análises estática e dinâmica na identificação de *malware* utilizando aprendizado de máquina. O processo iniciou-se com a coleta de exemplares, então divididos em dois conjuntos: um composto por 2.659 executáveis benignos obtidos do *SourceForge*[1], *OldApps*[2] e outras fontes; outro por 131.073 exemplares de *malware* obtidos do *VirusShare*[3]. No processo de escolha das características, levou-se em consideração tanto *strings* extraídas da análise estática como funções de API capturadas da análise dinâmica. Para formar o dicionário de *strings* da análise estática, usaram-se várias ferramentas ou informações presentes no cabeçalho dos binários, tais como *Yara*[4], *PEiD*[5] e dados obtidos pelo *PEScanner*[6] (DateSusp, EPSusp, SuspEntropy, NumberOfSections, IATSusp, SuspiciousString, FileSize, CRCSusp). Esse dicionário posssui os atributos mais utilizados por *malware*. No caso da análise dinâmica, montou-se um dicionário de termos semelhante ao visto em [Andrade et al. 2013]. Para o processo de aprendizado, foi utilizado o framework FAMA [Duarte et al. 2012]. Os algoritmos de aprendizado de máquina escolhidos para a tarefa de classificação foram o C5.0 (árvore de decisão) e a técnica *Random Forest*, cuja acurácia foi de 95,75%.

[Provataki and Katos 2013] apresentaram um *framework* para análise forense de *malware* baseado em Cuckoo Sandbox, usado para avaliar e elaborar relatórios sobre o comportamento dos exemplares executados. O trabalho abordou as limitações da análise dinâmica e ampliou a funcionalidade da ferramenta Cuckoo Sandbox a fim de automatizar o processo de correlacionamento e investigação de várias execuções de um binário suspeito sobre plataformas distintas de sistemas operacionais. A abordagem apresentada permite que o analista identifique mudanças comportamentais quando o artefato é executado e possibilita responder questões relacionadas às atividades do programa malicioso que auxiliem uma investigação mais aprofundada.

[Firdausi et al. 2010] demonstraram uma prova de conceito de um método para detecção de *malware*. Inicialmente, o comportamento dos artefatos foi obtido via análise dinâmica no sistema Anubis [Iseclab 2015], e os relatórios gerados foram pré-processados de forma a se criar vetores para classificação. Procedeu-se uma comparação de desempenho de cinco diferentes técnicas — *K-Nearest Neighbors* (KNN), *Naive Bayes*, Árvore de Decisão (J.48), *Support Vector Machine* (SVM) e Rede Neural do tipo Perceptron de Múltiplas Camadas (MLP). Foi produzido um pequeno conjunto de amostras maliciosas e dados de executáveis benignos. Os resultados obtidos mostraram que, em geral, o melhor desempenho é conseguido pela árvore de decisão J.48 com uma taxa de falsos-positivos de 2,4% e acurácia de 96,8%.

[Pirscoveanu 2015] investigou as inconsistências associadas à geração de rótulos por mecanismos antivírus no processo de aprendizado não-supervisionado, com objetivo de realizar análise comportamental de *malwares*. Uma versão personalizada do Cuckoo

[1] `https://sourceforge.net`
[2] `http://www.oldapps.com`
[3] `https://virusshare.com`
[4] `http://virustotal.github.io/yara/`
[5] `https://www.aldeid.com/wiki/PEiD`
[6] `https://code.google.com/archive/p/malwarecookbook/`

Sandbox foi utilizada para coletar ações de cerca de 270.000 exemplares maliciosos e, posteriormente, criar um dicionário de termos consistindo de chamadas de API que foram executadas com sucesso e aquelas que falharam, com seus respectivos códigos de retorno. Além disso, avaliou-se os resultados de detecção fornecidos por fabricantes de antivírus. Com isso, criou-se uma solução que estabelece como rótulo o nome de família que recebeu a maioria do votos dos AVs. Devido às inconsistências entre os rótulo de AVs, a distância de Levensthein foi usada a fim de medir a similaridades entre os rótulos.

[Huang et al. 2013] explora a possibilidade de utilizar técnicas *fuzzy* para identificar semelhanças entre famílias de softwares maliciosos. No modelo proposto, características maliciosas de um programa são inicialmente captadas pelo reconhecimento de padrão *fuzzy* e, em seguida, uma ontologia *fuzzy* para análise comportamental de *malwares* é apresentada. A abordagem é composta por uma inferência *fuzzy* e um mecanismo semântico de tomada de decisão para descrever as mudanças do valor de chave de registo, conexão de rede e alteração de arquivos, permitindo construir a ontologia e regras de comportamento. O modelo é capaz de detectar programas maliciosos desconhecidos e variantes de *malwares*.

Para fins de avaliação dos resultados deste trabalho frente ao paradigma da lógica convencional, também se realizou uma análise comparativa entre o algoritmo de agrupamento *K-Means*, bastante empregado na literatura, e o algoritmo *FCM*.

3. Análise Comportamental

Nesta seção, discute-se brevemente as fases necessárias para a geração do vetor de características a ser usado na etapa de agrupamento com os algoritmos de aprendizado de máquina, bem como o ambiente de execução utilizado. Basicamente, todo o processo para se completar a análise comportamental é composto por três fases:

1. Coleta de exemplares de *malware*;
2. Captura das chamadas de funções das APIs;
3. Geração do dicionário de termos.

3.1. Ambiente de execução

As amostras foram processadas no sistema de análise dinâmica *Cuckoo Sandbox* executando no sistema operacional *Linux Ubuntu*. No *Cuckoo*, configuraram-se 3 máquinas virtuais com *Windows XP* atualizados com *Service Pack* 3. Além disso, para se aproximar de um ambiente real de usuário, *softwares* comumente usados como *Acrobat Reader, Skype, Flash, Office*, entre outros foram instalados nas máquinas virtuais. Esses programas foram baixados e instalados por meio do *Ninite* [Swieskowski and Kuzins 2016]

3.2. Coleta de *malware*

Nesta fase, foram obtidas amostras de *malware* do *site VirusShare*. Neste site, é possível a busca de exemplares para *download* por ano de recebimento. Assim, foram separados, de forma aleatória, exemplares de 2012, 2013, 2014, 2015 e 2016, totalizando a quantidade de 21.455 amostras consideradas maliciosas.

3.3. Captura das funções de API

Os 21.455 exemplares de *malware* coletados foram submetidos para execução no sistema de análise dinâmica *Cuckoo Sandbox*. Durante cada execução, esse analisador monitorou as funções de API chamadas pelo artefato sob análise e gerou um relatório em formato JSON para cada amostra. Os relatórios contêm informações do *malware* obtidas de maneira estática e dinâmica pelo Cuckoo.

3.4. Geração do dicionário de termos

Os relatórios gerados anteriormente passaram por um processo de filtragem (implementado por meio de um *script* em Python), usado para identificar as funções de API mais relevantes. Assim, para cada arquivo JSON gerado, levantou-se quais chamadas a funções de API são consideradas maliciosas. Para isso, as funções de API presentes nos relatórios gerados foram comparadas com 153 funções de API comumente utilizadas por *malware* de acordo com o trabalho de [Pirscoveanu 2015]. Apenas as funções acessadas pelo exemplar que pertencem a essa lista de 153 chamadas de funções foram utilizadas para a geração do dicionário de termos. Assim, após a filtragem e comparação de todos os 21.455 relatórios produzidos a partir da execução dos exemplares do conjunto de dados, obteve-se um dicionário composto por 144 termos (ou chamadas de funções).

O *script* de filtragem, após criar o dicionário de termos, gera um arquivo em formato CSV (*comma-separated values*). Este arquivo contém a quantidade de vezes em que uma dada função de API presente no dicionário de termos é chamada por cada um dos exemplares.

4. Aprendizado de Máquina

Nesta seção, apresenta-se a aplicação dos algoritmos de aprendizado de máquina nos dados obtidos na etapa de análise comportamental. O arquivo CSV gerado com os vetores de características extraídos a partir de todos os exemplares executados serviu de entrada tanto para o algoritmo *K-Means* quanto para o *FCM*. Ambos foram implementados em linguagem "R". Porém, devido à grande dimensão do vetor de características (144 termos) construído na etapa anterior, antes da aplicação dos algoritmos de agrupamento é necessário um procedimento para redução de dimensionalidade, cuja descrição vem a seguir.

4.1. Análise de Componentes Principais

Observe que o *dataset* inicial, gerado na etapa descrita na Seção 3, é composto por uma matriz de 144 colunas (dimensões) por 21.455 linhas (observações). Para viabilizar o tempo de execução dos algoritmos de agrupamento e reduzir o risco de baixo desempenho devido à "maldição da dimensionalidade" [Duda et al. 2001], é necessário proceder com algum pré-processamento que diminua a dimensão do problema.

Dessa maneira, aplicou-se a reconhecida técnica de Análise de Componentes Principais (PCA, do inglês *Principal Component Analysis*), que consiste em identificar uma projeção linear da matriz de dados $\mathbf{S}$ original

$$\mathbf{X} = \mathbf{AS} \tag{1}$$

tal que a nova matriz $\mathbf{X}$ possua um número de características menor que $\mathbf{S}$, simultaneamente atendendo à restrição de tais características serem descorrelacionadas entre si e de mínimo erro quadrático médio com respeito aos dados originais [Duda et al. 2001].

Consequentemente, o uso das características geradas, via PCA, reduz o número de colunas presentes na nova matriz de dados e, devido à restrição de descorrelação, também reduz o grau de redundância. Nesse sentido, é possível observar na Tabela 1 o acúmulo da variação dos dados (variância ou nível de energia) à medida que se considera mais componentes principais.

Tabela 1. Acúmulo da variação dos dados

PCA	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
Acumulada	49,6%	60,2%	68,4%	73,2%	77,0%	79,8%	82,0%	84,0%
PCA	PC9	PC10	PC11	PC12	PC13	PC14	PC15	PC16
Acumulada	85,8%	87,1%	88,2%	89,2%	90,1%	90,9%	91,7%	92,4%
PCA	PC17	PC18	PC19	PC20				
Acumulada	93,2%	93,8%	94,4%	94,8%				

Dado que as 20 primeiras componentes principais explicam 94,8% da variância dos dados, elas foram selecionadas como características do vetor de entrada tanto para o algoritmo *K-Means* quanto para o *FCM*. Vale destacar que essas 20 primeiras componentes principais são combinações lineares das 144 originais.

4.2. Métodos não-supervisionados

A ideia de algoritmos não-supervisionados é fornecer classificação de informações de acordo com os próprios dados, baseado em análises e comparações entre os seus valores numéricos. Dessa maneira, esses métodos classificam os dados automaticamente, sem a necessidade de supervisão [Borges 2010].

Entre esses algoritmos, o *K-Means* [MacQueen 1967] é amplamente utilizado. Tornou-se popular devido à sua simplicidade e tem sido, com sucesso, aplicado durante os últimos anos, sobretudo para agrupamento de *malwares*. O objetivo desse método é particionar as amostras em K grupos, minimizando a distância *intra-clusters* e maximizando a distância *inter-clusters* [Borges 2010]. A distância euclidiana é um critério de similaridade amplamente utilizado para as amostras no espaço euclidiano.

O *Fuzzy C-Means* proposto por [Dunn 1974] e estendido por [Bezdek 1981] é um método de agrupamento que permite estabelecer, para um certo dado, um determinado grau de relação com cada um dos agrupamentos obtidos. Ele possui funcionamento e estrutura semelhantes ao *K-Means*, mas possui uma abordagem *soft*, ao permitir que um dado não esteja associado exatamente com um único *cluster*. O grau de relação (também chamado de grau de pertinência) entre uma amostra e um agrupamento é um valor que está no intervalo [0, 1]. Uma pertinência próxima a 1, significa que o exemplar e o agrupamento em questão são similares. Caso contrário, se esse valor se aproxima de zero, indica dissimilaridade entre a amostra e o agrupamento analisado [Borges 2010].

4.3. Número de *clusters*

Uma das dificuldades em se aplicar os principais algoritmos de agrupamento, como o *K-Means* e o *FCM*, é determinar antecipadamente o número de *clusters* que deve ser gerado

a partir das amostras de entrada. Em alguns casos, como o do presente trabalho, não se sabe exatamente o número correto de *clusters*, pois quer-se agrupar automaticamente os exemplares de acordo com o comportamento de execução apresentado (e não por classes geradas por rótulos de AV). Desse modo, para se chegar a um valor de parâmetro apropriado, foram usados dois métodos de validação: o método *Elbow* e o *Silhouette*.

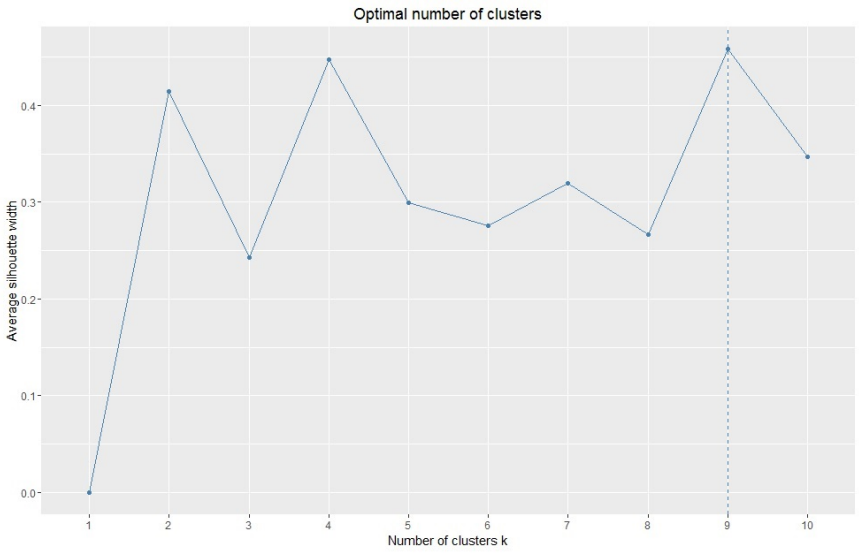


Figura 1. Aplicação do método Silhouette nas amostras de entrada

O método *Elbow* baseia-se na análise de variância em função do número de *clusters*. Tal como muitos outros métodos, ele exige o uso de um algoritmo de agrupamento, como *K-Means*, para executar o agrupamento e retornar a quantidade de variância produzida por cada *cluster* [Thorndike 1953]. O método Silhouette (Figura 1) refere-se a um método de interpretação e validação de consistência dentro dos *clusters*. A técnica fornece uma representação gráfica sucinta de como cada objeto se encontra dentro de seu *cluster*. O valor *silhouette* mede o quanto um objeto é similar ao seu próprio grupo (coesão) em comparação com os outros grupos (separação) [Rousseeuw 1987].

Com o método *Elbow*, não foi possível observar claramente o número de *clusters* necessários/adequados. Entretanto, a aplicação do método Silhouette resultou na sugestão de definição de 9 *clusters*, conforme Figura 1. Logo, primeiramente executou-se o algoritmo *K-Means* com $k = 9$, obtendo-se a distribuição das amostras conforme a Tabela 2.

Tabela 2. Distribuição das amostras entre os 9 *clusters*, após aplicação do *K-Means*.

Cluster	1	2	3	4	5	6	7	8	9
# Exemplares	5579	3526	740	1305	3656	409	1284	4765	191

Uma vez que o intuito é comparar o resultado dos algoritmos, o *FCM* também foi aplicado com a definição de 9 *clusters* como parâmetro de entrada. No caso do *FCM*, essa amostra não pertence completamente a um determinado cluster, os graus de pertinência indicam participação dela em cada cluster. Dessa forma, verificou-se em qual cluster o artefato possui maior pertinência, para fins de comparação com o *K-Means*. A distribuição dos exemplares resultante pode ser observada na Tabela 3.

Tabela 3. Distribuição das amostras entre os 9 *clusters*, após aplicação do *FCM*.

Cluster	1	2	3	4	5	6	7	8	9
# Exemplares	1945	2935	1830	399	1197	774	2277	5622	4476

5. Resultados dos Experimentos

Os resultados obtidos por meio da aplicação dos dois algoritmos de aprendizado de máquina foram analisados sob três aspectos: semelhança entre *clusters* produzidos, processo de rotulação e tempo de execução. A seguir, cada um dos aspectos supracitados é discutido.

5.1. Semelhança entre *clusters*

Para a análise deste aspecto, foi desenvolvido um *script* em *Python* com a finalidade de comparar os *clusters* gerados como saída de cada algoritmo aplicado. Conforme mostrado na Tabela 4, pode-se observar o grau de coincidência dos exemplares para cada par de *clusters*, medido em porcentagem. Cinco *clusters* das duas técnicas apresentaram um elevado grau de semelhança, acima de 90%: por exemplo, o *cluster* K3 e C6 mostraram semelhança de 98%. O *cluster* 2 do *K-Means* (K2) mostrou a maior diferença, ao alocar 48% dos seus dados no *cluster* 1 do *FCM* (C1) e 45% no *cluster* 2 do *FCM* (C2). Com base nesses resultados, pode-se concluir que os dois métodos produziram agrupamentos semelhantes, conforme Tabela 4.

Tabela 4. Semelhança entre *clusters* produzidos por *K-Means* (K) e *FCM* (C).

Clusters	K1/C7	K1/C8	K2/C1	K2/C2	K3/C6	K4/C5
Semelhança	40%	60%	48%	45%	98%	91%
Clusters	K5/C2	K5/C8	K6/C4	K7/C3	K8/C9	K9/C3
Semelhança	37%	63%	95%	92%	93%	55%
Clusters	K9/C1					
Semelhança	40%					

Os métodos *K-Means* e *FCM* possuem os processos de inicialização, iteração e término bastante semelhantes. A diferença reside no emprego da função de pertinência por parte do método *FCM*. De fato, pode-se interpretar o *K-Means* como um caso especial do *FCM* para uma função de pertinência *crisp*, i.e. que retorna 1 se o ponto de dados é mais próximo do centroide ou 0, caso contrário. No caso do *FCM*, sabe-se que a função de pertinência pode gerar valores entre 0 e 1. Ao analisar os *clusters* que apresentaram diferenças na Tabela 4, pode-se levantar a hipótese de que categorias de artefatos de comportamento mais complexo, à luz da função de pertinência *fuzzy*, levam a uma associação "diversificada" de *clusters*, o que não se alinha com os resultados equivalentes do *K-Means*.

5.2. Processo de rotulação

No momento de execução das amostras, o *Cuckoo* consulta a plataforma *online* do *VirusTotal*[7] de modo a verificar se o *malware* em questão já foi previamente enviado para

[7] `https://www.virustotal.com`

detecção pelos antivírus disponíveis. Em caso positivo, o *VirusTotal* retorna um conjunto de rótulos, que corresponde ao resultado da varredura por vários AVs. Esse conjunto é armazenado como uma estrutura de dados no arquivo *JSON* referente ao relatório de cada exemplar. Como o cenário usual consiste em se obter rótulos distintos e/ou inconsistentes para um mesmo exemplar, elegeu-se o antivírus Avast para ser responsável pela atribuição de uma classe aos exemplares do conjunto deste experimento. A escolha do Avast deu-se pelo fato deste apresentar taxa de detecção elevada, de acordo com o VB100 [VirusBulletin 2016].

Tabela 5. Exemplares de *malware* rotulados com Avast encontrados em *clusters* resultantes da aplicação do *K-Means* no *dataset*.

Cluster	K1	K2	K3	K4	K5	K6	K7	K8	K9
Trojan	2118	847	83	287	1025	80	401	2170	27
Worm	132	373	32	79	135	44	153	26	1
Spyware	243	182	15	45	98	9	20	130	0
Rootkit	72	13	7	5	27	4	10	7	0
PuP	258	627	437	384	432	99	53	341	70
Genérico	1707	833	93	299	836	49	436	680	55
Sem rótulo	1049	651	73	206	1103	124	211	1411	38

Observa-se que o Avast, ao rotular as 21.455 amostras, distribuiu-as em cinco tipos de *malware* com escopo de comportamento bem definido—*Trojan*, *Worm*, *Spyware*, *Rootkit* e *Potentially Unwanted Program (PUP)*. O rótulo ”Genérico”corresponde a nomes genéricos atribuídos pelo Avast (em geral detectados por procedimentos heurísticos), enquanto que ”Sem rótulo”refere-se aos artefatos que não foram identificados por este AV.

Ao analisar a Tabela 5, nota-se que 33% das amostram foram rotuladas como *Trojan* e 23% forma enquadradas como ”Genérico”, perfazendo um total de 55% de todas as amostras. Além disso, esses dois rótulos se espalharam por todos os *clusters*. Na verdade, os rótulos *Trojan* e ”Genérico”são usados por muitos fornecedores de AV como rótulos guarda-chuva [Mohaisen and Alrawi 2013], ou seja, cada um desses rótulos abrange mais de uma família de *malware*.

Nesta seção, apresenta-se uma solução para rotulação baseada em detecção por AV. No entanto, diferentemente do trabalho de [Pirscoveanu 2015], utiliza-se a tabela de pertinência do *FCM* para auxiliar no processo: inicialmente, realiza-se a rotulação do agrupamento resultante do *K-Means* e, após isso, rotula-se com o *FCM*.

O rótulo atribuído a cada *cluster* advém da classe mais representativa encontrada dentro dele, de acordo com os resultados obtidos pelo Avast. Assim, o *cluster* K8 recebeu o rótulo *Trojan*, uma vez que possui 2170 exemplares classificados como tal.

Seguindo essa lógica, tem-se esta atribuição de rótulos: K8 - *Trojan*; K2 - *PuP*; K1 - *Spyware*; K5 - *Rootkit*; e k7 - *Worm*

Nota-se que o tipo *Trojan* está presente em todos os agrupamentos e que cada rótulo só pode ser atribuído a um único *cluster*. Desse modo, por exemplo, ao atribuir o rótulo *Trojan* ao K8, este não pode ser concedido também ao K2, mesmo que neste *cluster* a quantidade desse tipo de artefato seja maioria. Nesse caso, opta-se pelo segundo

mais representativo, no caso *PuP*. Ademais, como exitem apenas cinco rótulos, quatro agrupamentos ficam sem rótulos (K3, K4, K6 e K9)

Tabela 6. *Malwares* rotulados com Avast encontrados em *clusters* resultantes da aplicação do *FCM* no conjunto de exemplares.

Cluster	C1	C2	C3	C4	C5	C6	C7	C8	C9
Trojan	393	928	463	78	263	89	874	1806	2144
Worm	303	88	174	45	70	37	65	171	22
Spyware	101	110	72	8	40	19	127	164	101
Rootkit	8	21	11	3	5	7	27	58	5
PuP	381	425	220	96	351	443	131	400	254
Genérico	360	755	584	47	281	101	571	1669	620
Sem rótulo	399	608	306	122	187	78	482	1354	1330

Para o método *FCM*, repete-se o mesmo processo, chegando aos seguintes rótulos para cada agrupamento, conforme Tabela 6: C9 = *Trojan*; C1 = *Worm*; C8 = *Spyware*; C6 = *PuP*; e C7 = *Rootkit*.

Os agrupamentos C2, C3, C4 e C5 estão sem rótulo. No entanto, neste caso, diferentemente do *K-Means*, pode-se tirar vantagem da tabela de pertinência. Para ilustrar essa ideia, foram selecionadas 10 amostras de C2 com suas respectivas pertinências, conforme a Tabela 7.

Tabela 7. Tabela de pertinência de C2 com 10 exemplares, e seus dois *clusters* (C1 e C8) mais pertinentes/aderentes a ele.

Amostras/Cluster	C1 (Worm)	C2	C8 (Spyware)
Exemplar 1	30%	32%	7,5%
Exemplar 2	10%	7,5%	7,6%
Exemplar 3	35%	43%	4,0%
Exemplar 4	20%	67%	4,1%
Exemplar 5	18%	28%	16%
Exemplar 6	1,8%	53%	42%
Exemplar 7	6,7%	86%	2,4%
Exemplar 8	14%	27%	26%
Exemplar 9	22%	44%	11%
Exemplar 10	5,3%	66%	22%

Na Tabela 7, C1 e C8 foram selecionados na medida em que apresentaram o maior grau de pertinência/aderência ao C2. Por exemplo, analisando o Exemplar 5, nota-se que sua pertinência ao C2 é 28%, seguido de C1 (18%) e C8 (16%). Desse modo, sugere-se o rótulo "Worm-Spyware"para o *cluster* C2. Repetindo esse procedimento para os três agrupamentos restantes sem rótulos, têm-se: C3 = *Worm-Trojan*; C4 = *PuP-Worm-Trojan*; e C5 = *PuP-Worm*

Por fim, vale destacar que a rotulação, de um modo geral, fornece um subsídio importante ao analista de *malwares*, ao especificar como rótulo um nome descritivo das características dos artefatos que ali se encontram. Em particular, o *FCM* permite verificar

graus de relacionamento entre os *clusters*, ajudando a observar outros comportamentos da amostra. Por exemplo, ao rotular C2 com *"Worm-Spyware"*, o *FCM* informa ao analista que atividades maliciosas de *Worms* e *Spywares* devem ser monitoradas. Isso permite que a análise se concentre nas atividades normalmente executadas por essas espécies, tornando o exame mais direcionado ao real comportamento da amostra. Ademais, na atualidade, os artefatos normalmente executam atividades de mais de uma família, o que torna a modelagem por meio do *FCM* mais fidedigna.

5.3. Tempo de Execução e Escalabilidade

Nos experimentos realizados, o algoritmo *K-Means* foi executado da seguinte forma: `kmeans(características, 9, nstart = 25)`, onde o parâmetro "características" representa os vetores obtidos do conjunto de amostras, o valor "9" corresponde ao número de *clusters* definidos para geração da saída e "nstart = 25" é o número de vezes diferentes que os *clusters* de partida são aleatoriamente gerados, a fim de que se possa escolher a melhor configuração para inicialização. Com relação ao *FCM*, este foi executado da seguinte maneira: `cmeans(características, 9, iter.max = 100, dist = "euclidean", m = 1.5)`, onde os dois primeiros parâmetros são idênticos aos do *K-Means*, "iter.max = 100" é o número máximo de iterações para agrupamento, "dist = euclidean" é a métrica para calcular a distância entre os exemplares (no caso, distância Euclidiana), e "m = 1.5" se refere ao índice de *fuzzificação*. A Tabela 8 mostra a medida dos tempos para a aplicação de cada algoritmo.

Tabela 8. Tempo de execução medido para cada algoritmo.

Algoritmo	Tempo (s)
K-Means	21.1
FCM	19.6
Diferença	2.5

Cabe ressaltar que, apesar do *K-Means* possuir complexidade inferior ao *FCM*, especificamente neste experimento, o parâmetro "nstart = 25", que possibilita uma aplicação mais robusta do *K-Means*, elevou consideravelmente seu tempo de execução.

Por fim, no caso de novas amostras a serem analisados, faz-se necessário reexecutar o método para agregar esses novos artefatos aos *clusters*. No entanto, todo o processo foi automatizado por meio de *scripts* em *Python* e R[8]

6. Conclusões e Trabalhos Futuros

Este artigo propõe a aplicação de um método baseado em lógica *fuzzy* para alcançar uma forma adequada de agrupar exemplares de *malware* modernos. Pôde-se observar a vantagem do uso do algoritmo *Fuzzy C-Means (FCM)*, cujas tabelas de pertinência permitem sugerir nomes aos grupo de exemplares de *clusters* e fornecem mais informações do que simples rótulos atribuídos por antivírus. Além disso, outro benefício de aplicação de lógica *fuzzy* em relação ao método de lógica *crisp* reside no fato de que os programas maliciosos não se limitam apenas a um comportamento específico de uma dada família,

[8]Disponíveis em: `https://drive.google.com/drive/folders/0B09rUNJplNMLTGVrb0VvZ0tvaDA?usp=sharing`.

ou seja, podem pertencer a várias delas ao mesmo tempo. Portanto, a lógica *fuzzy*, apresentada nos resultados deste trabalho, modela, de forma mais fidedigna, o real comportamento malicioso exibido durante uma infecção.

Como trabalhos futuros, pretende-se aprimorar o processo de rotulação de cada amostra e, consequentemente dos *clusters* produzidos. Nos experimentos, percebeu-se que o principal objetivo de um fornecedor de AV é detectar códigos maliciosos, sendo o processo de rotulação uma prioridade secundária. Logo, AVs não possuem um processo de rotulação confiável, tornando necessário o desenvolvimento de um novo método. De acordo os testes realizados neste trabalho, parece viável produzir rótulos baseados apenas nas características internas de cada amosta no *cluster*. Para isso, pode-se eleger as amostras mais representativas de cada *cluster* e estas devem ser analisadas a fim de se descobrir seu comportamento e posterior rotulação do *cluster*. Nessa abordagem, o rótulo de cada *cluster* não seria mais baseado em AVs. Com relação à análise de novas amostras, pretende-se utilizar um método supervisionado, uma vez que se pode comparar melhor agrupamentos conhecidos a priori. Planeja-se também executar uma quantidade maior de exemplares para se verificar a formação de outros *clusters*, os quais podem apresentar novos comportamentos. Almeja-se também melhorar o processo de definição da quantidade de *clusters* e distribuição das amostras nos agrupamento, de modo a se ter mais dinamicidade na geração de modelos de classificação.

Referências

Andrade, C. A. B., de Mello, C. G., and Duarte, J. C. (2013). Malware automatic analysis. In *2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 681 – 686. IEEE.

Bezdek, J. C. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers Norwell, MA, USA, Utah State University, Logan, Utah, USA, 1 edition.

Bezdek, J. C., Ehrlich, R., and Full, W. (1984). FCM: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2–3):191–203.

Borges, V. R. P. (2010). Comparação entre as Técnicas de Agrupamento K-Means e Fuzzy C-Means para Segmentação de Imagens Coloridas. *XII Encontro Anual de Computação (EnAComp)*.

Damballa (2009). 3% to 5% of enterprise assets are compromised by bot-driven targeted attack malware. `http://www.prnewswire.com/news-releases/3-to-5-of-enterprise-assets-are-compromised-by-bot-driven-targeted-attack-malware-61634867.html`. Acessado em junho de 2016.

Duarte, J. C., de Almeida Oliveira, F., dos Santos, J. C., and de Oliveira Neto, G. A. (2012). Framework de Aprendizado de Máquina (FAMA). `https://code.google.com/archive/p/fama/`. Acessado em junho de 2016.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. Wiley-Interscience, New York, USA, 2 edition.

Dunn, J. C. (1974). A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics*, 3(3):32–57.

Firdausi, I., lim, C., Erwin, A., and Nugroho, A. S. (2010). Analysis of machine learning techniques used in behavior-based malware detection. In *Proceedings of the 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*, ACT'10, pages Pages 201 – 203.

Huang, H.-D., Acampora, G., Loia, V., Lee, C.-S., Hagras, H., Wang, M.-H., Kao, H.-Y., and Chang, J.-G. (2013). Fuzzy markup language for malware behavioral analysis. In *On the Power of Fuzzy Markup Language*, pages 113–132. Springer.

Intel Security (2014). Relatório do mcafee labs sobre ameaças. `http://www.mcafee.com/br/resources/reports/rp-quarterly-threat-q3-2014.pdf`. Acessado em junho de 2016.

Iseclab (2015). Anubis - Analyzing Unknown Binaries. `http://analysis.iseclab.org/`. Acessado em junho de 2015.

MacQueen, J. (1967). Some Methods for Classification and Analysis of Multivariate Observations. *5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281 – 297.

Mangialardo, R. J. (2015). Integrando as análises estática e dinâmica na identificação de malwares utilizando aprendizado de máquina. Master's thesis, Mestrado em Sistemas e Computação, Instituto Militar de Engenharia, Rio de Janeiro.

Mohaisen, A. and Alrawi, O. (2013). Unveiling zeus: automated classification of malware samples. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 829–832. ACM.

Pirscoveanu, R.-S. (2015). Clustering Analysis of Malware Behavior. Master's thesis, Institute of Electronic Systems Department of Communication Technology at Aalborg University, Denmark.

Provataki, A. and Katos, V. (2013). Differential malware forensics. *Digital Investigation: The International Journal of Digital Forensics & Incident Response*, 10(4):Pages 311–322.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65.

Salehi, Z., Ghiasi, M., and Sami, A. (2012). A Miner for Malware Detection Based on API Function Calls and Their Arguments. In Salehi, Z., editor, *Artificial Intelligence and Signal Processing (AISP)*, pages 563 – 568. IEEE.

Sami, A., Yadegari, B., Rahimi, H., Hamzeh, A., Hashemi, S., and Hamzeh, A. (2010). Malware detection based on mining api calls. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC'10, pages 1020 – 1025. ACM New York, NY, USA.

Swieskowski, P. and Kuzins, S. (2016). Ninite. `https://ninite.com/`. Acessado em fevereiro de 2016.

Thorndike, R. L. (1953). Who Belongs in the Family? *Pyschometrika*, 18(4):267–276.

VirusBulletin (2016). Vb100. `https://www.virusbulletin.com/testing/vb100/`. Acessado em fevereiro de 2016.

Análise Transparente de *Malware* com Suporte por *Hardware*

Marcus Botacin[1], **Paulo Lício de Geus**[1], **André Grégio**[2]

[1]Instituto de Computação (IC)
Universidade Estadual de Campinas (Unicamp)
Campinas – SP – Brasil

[2]Departamento de Informática (DInf)
Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

Abstract. *Dynamic analysis is one of the main techniques used for malware profiling, identification of features and development of countermeasures. Therefore, malware authors continuously seek for ways of preventing their code from running inside analysis environments to thwart detection. Besides, operating system improvements make their instrumenting for malware monitoring more difficult. Hence, hardware-assisted analysis approaches have been developed to overcome these issues. In this paper, we propose a low-overhead malware dynamic analysis system based on branch monitoring supported by hardware (Intel processor monitors), in order to accomplish the transparency required to prevent malware from identifying (and evading) monitoring.*

Resumo. *A análise dinâmica é uma das principais técnicas utilizadas para caracterização de malware, identificação de suas funcionalidades e desenvolvimento de contra-medidas. Portanto, desenvolvedores de malware buscam continuamente por formas de impedir a execução de seus códigos nesses ambientes, dificultando a detecção. Além disso, avanços nos sistemas operacionais criaram obstáculos para sua instrumentação. Para superar tais problemas, surgiram abordagens de análise assistidas por hardware. Neste artigo, propõe-se um sistema de análise dinâmica de malware baseado em monitoramento de branches com suporte de hardware (monitores dos processadores Intel), alcançando a transparência necessária para evitar a identificação do ambiente e apresentando baixo overhead.*

1. Introdução

Os danos causados por programas maliciosos abrangem desde o comprometimento de um sistema operacional até prejuízos pessoais e financeiros aos usuários. Estatísticas do CERT.br [CERT.br 2015] indicam que mais de 50% dos incidentes reportados, como *scans* e fraudes, podem ter sido causados por *malware*. Perdas devido a fraudes eletrônicas alcançam 1,8 bilhão de Reais [FEBRABAN 2015]. A análise de *malware* é uma etapa importante para a identificação do potencial de danos e extensão destes, bem como para o desenvolvimento de vacinas, assinaturas de detecção e procedimentos de perícia forense computacional. As técnicas de análise são classificadas como estáticas ou dinâmicas [Sikorski and Honig 2012].

Na análise estática, o arquivo suspeito é inspecionado sem sua execução, através apenas da observação de suas características estruturais (*headers* e *strings*), estatísticas

(entropia) e *disassembly*. Entretanto, limitações intrínsecas [Moser et al. 2007] trazem a necessidade adicional da análise dinâmica na qual o exemplar é efetivamente executado em um ambiente controlado (*sandbox*), exibindo seu comportamento de infecção. Contudo, criadores de *malware* têm buscado formas de identificar a execução do exemplar em um ambiente de análise [Chen et al. 2008] de modo a bloquear a execução do *payload* malicioso e evadir a detecção do *malware*. Dentre as técnicas utilizadas para tal fim, destacam-se as baseadas na detecção de emulação e de injeção de código.

Diante deste novo cenário, muitas técnicas foram propostas para mascarar os efeitos colaterais da monitoração, usados pelos atacantes para a identificação do ambiente de análise [Balzarotti et al. 2010, Vasudevan and Yerraballi 2005]. Porém, qualquer abordagem via *software* não está protegida o suficiente da subversão por parte do atacante. Assim sendo, abordagens com suporte de *hardware* têm sido propostas. As mais bem sucedidas delas baseiam-se na criação de uma máquina virtual de análise que usa instruções especiais do processador (*Hardware Virtual Machine* - HVM) e na reescrita do *Basic Input Output System - BIOS* para analisar a execução a partir de um modo privilegiado (*System Management Mode* - SMM). Embora eficazes, dado sua *Trusted Code Base* (TCB) reduzida, as abordagens citadas apresentam alto custo de desenvolvimento e desempenho, com *overhead* acima de 100% dependendo da frequência de interrupção.

Recentemente, uma abordagem assistida por *hardware* menos custosa (*lightweight*) surgiu através do uso de monitores de desempenho, sobretudo os monitores de desvios (*branches*), os quais possuem *overhead* próximo de zero. O uso atual desses monitores tem sido a detecção de ataques por *Return Oriented Programming* (ROP). Neste trabalho, introduz-se uma nova aplicação para estes monitores e importante contribuição para a área de segurança: a geração de traços de execução de *malware* de forma transparente (na visão do programa monitorado). Até onde se sabe, este é o primeiro trabalho a fazer uso deste monitores para tal propósito. Ao longo do artigo, detalha-se as vantagens e limitações da abordagem proposta.

Este artigo é dividido da seguinte maneira: na Seção 2, introduz-se os conceitos e os detalhes de funcionamento dos monitores utilizados, as abordagens existentes para monitoramento de *malware* e as limitações destas; na Seção 3, apresenta-se o sistema proposto, incluindo detalhes de sua implementação; na Seção 4, aplica-se a solução implementada na reconstrução do *Call Graph* e do *Control Flow Graph* de um código compilado, bem como a exemplares reais de *malware*; na Seção 5, discute-se os desafios, o desempenho e os limites do sistema proposto; por fim, a Seção 6 apresenta as considerações finais sobre os resultados obtidos e possíveis evoluções para o sistema.

2. Aspectos Técnicos e Trabalhos Relacionados

Descreve-se, nesta seção, os detalhes técnicos relativos às plataformas de monitoração usando *hardware*. Embora presentes em diversas arquiteturas, limita-se a descrição à tecnologia utilizada pela Intel, escolhida para a implementação do protótipo deste trabalho por questões de disponibilidade. Apresenta-se também um panorama das abordagens para monitoração de *malware*, agrupadas por modo de atuação, de modo que se possa avaliar as limitações de cada uma delas a fim de justificar a escolha dos monitores de desempenho.

2.1. Monitoração assistida por *hardware*

A plataforma de monitoração da Intel pode ser dividida em dois componentes, o *Precise Event Based Sampling* (PEBS) e o monitoramento de *branches*. O PEBS consiste em uma série de registradores configuráveis para contar diferentes eventos, tais como instruções executadas, número de ciclos transcorridos, *misses* de memória cache, entre outros. Os dados gerados pelo PEBS podem ser acessados via registradores específicos (*Model Specific Register* - MSR) ou registro em memória. Neste último, uma página do sistema operacional pode ser fornecida para o armazenamento dos dados, gerando uma interrupção quando o *buffer* estiver cheio. O monitoramento de *branches* atua de forma similar, mas com escopo nas informações dos endereços de origem e destino das instruções de desvio, como mostrado na Figura 1. O mecanismo fornece ainda, para cada desvio, a informação se o salto foi ou não previsto corretamente (*branch-prediction*).

FROM	TO
fff80145f42118	fff8801ae01f36
fff80145e5a510	fff80145ecf2ae
fff80145f420dc	ff80146079523
fff80145ecf2b3	ffff8801ae0157

Figura 1. Exemplo de *branch stack* obtida com o mecanismo de monitoramento do processador.

Os dados obtidos podem ser armazenados de duas formas: a primeira, denominada *Last Branch Record* (LBR), consiste em pares de registradores (de 8 a 32, dependendo da família de processadores) do processador que formam uma fila circular, sendo acessível através dos registradores MSR; a segunda, denominada *Branch Trace Store* (BTS), consiste em um mecanismo análogo ao PEBS de fornecimento de páginas do sistema operacional para o armazenamento dos dados, podendo gerar uma interrupção quando o *buffer* estiver cheio. Embora denominado monitor de *branch*, os dados capturados não se restringem às instruções de salto do processador (`JMP`, `JNE`), mas a qualquer desvio do fluxo, incluindo instruções `CALL` e `RET`. A captura de dados de cada um dos tipos de instruções de desvio pode ser filtrada pela inserção de *flags* em um registrador de controle. Pode-se ainda filtrar ações nos níveis de *kernel* e *userland* e decidir se a captura será ou não interrompida quando uma interrupção for gerada.

2.2. Trabalhos Relacionados

Os trabalhos relacionados discutidos a seguir são divididos de acordo com a técnica utilizada para monitoração do *malware* em execução.

Análise em *userland*. A monitoração no nível do usuário é uma das formas mais simples de implementação para interceptação de programas em execução. A inserção de

hooks via injeção de *DLL* permite interceptar qualquer ação de determinado processo, sendo usada pelos sistemas de análise de *malware* Cuckoo Sandbox [Guarnieri 2013] e CWSandbox [Willems et al. 2007]. Porém, além de requerer que a injeção da DLL de monitoração seja realizada em cada processo sob análise, a técnica é facilmente detectável, pois altera a imagem do processo em memória.

Tradução dinâmica. Algumas ferramentas atuam em nível mais granular do que o das chamadas de sistema, podendo lidar diretamente com as instruções do binário, tais como Valgrind [Nethercote and Seward 2003], PIN [Luk et al. 2005] e DynamoRIO [Bruening et al. 2012]. Este tipo de abordagem traz mais possibilidades para detecção, criação de heurísticas e demais formas de monitoração, mas pode ser evitada pelo *malware* dados seus efeitos colaterais—comportar-se como um *debugger* ou alterar as instruções do programa. Outra geração de instrumentadores dinâmicos surgiu mais recentemente, como VAMPIRE [Vasudevan and Yerraballi 2005] e SPIKE [Vasudevan and Yerraballi 2006], capazes de alterar as instruções de evasão em tempo real. Porém, esse tipo de solução depende de regras pré-estabelecidas que podem ser subvertidas por novas técnicas de evasão implementadas por desenvolvedores de *malware*.

***Hardware-assisted Virtual Machines* (HVM).** Dada as limitações intrínsecas das abordagens por software, a evolução natural é obter suporte via *hardware*, cuja subversão é mais difícil. Um exemplo são as máquinas virtuais em *hardware* (HVM), nas quais as instruções de virtualização do processador são aplicadas na construção de um sistema de análise. Ether [Dinaburg et al. 2008], MAVMM [Nguyen et al. 2009], CXPInspector [Carsten Willems 2012] e SPIDER [Deng et al. 2013] são exemplos de sistemas baseados em HVM. Tais sistemas possuem características singulares: implementação minimalista, sem *drivers* e componentes adicionais, reduzindo assim a base de código confiável (*Trusted Code Base* - TCB) requerida e tornando-se menos "exploráveis"; ausência de efeitos colaterais, pois as instruções não são emuladas ou traduzidas, mas sim executadas no *hardware* real; granularidade da análise, uma vez que qualquer subsistema pode ser monitorado diretamente. Por outro lado, sistemas baseados em HVM possuem desvantagens associadas: são muito complexos de implementar, devido à necessidade de escrita de um *hypervisor*; limitados no uso prático, por exemplo, devido à restrição a execução em *single-core* e/ou a exigência de *hypervisor* específico; e são dependentes da versão e do sistema analisado, pois a instrospecção é baseada nos *offsets* e estruturas de dados opacas de cada *kernel*.

***System Management Mode* (SMM).** Outra abordagem com auxílio de *hardware* é a instrumentação das rotinas do modo SMM dos processadores. Trata-se de um modo protegido, com memória isolada e voltado para o gerenciamento do sistema, o qual possui funções para controle de energia e temperatura. O isolamento provido por este modo garante o não comprometimento trivial do sistema de análise. A análise realizada neste modo é transparente—as instruções são executadas diretamente no hardware, sem qualquer tradução ou emulação—e as ferramentas baseadas em SMM visam tanto *debugging* (MALT [Zhang et al. 2015] e SPECTRE [Zhang et al. 2013]) quanto forense do sistema operacional (SMMDumper [Reina et al. 2012]). Assim como as abordagens de HVM, a escrita de uma ferramente SMM é complexa, exigindo técnicas de introspecção e a reescrita do BIOS, a qual nem sempre é possível devido a travas que visam garantir sua

integridade. O modo SMM também exige a escrita de *drivers* de dispositivos, como os de rede, para que se possa enviar os dados da máquina monitorada para o meio externo.

***Performance Counters*.** As restrições inerentes às soluções supracitadas incentivaram o surgimento de outras ferramentas—menos custosas em processamento e implementação—para o monitoramento assistido por *hardware*. A que mais se destaca e está presente na maioria das arquiteturas atuais é representada pelos monitores de desempenho, que permite coletar dados de *caches*, instruções e *branches*. Seu uso é bastante difundido em ferramentas de *profiling* como o `perf`[1] e o *Intel V-Tune* [Intel 2015]. Porém, as aplicações em segurança computacional são recentes: [Kompalli 2014] mostra como os monitores de *branch* e de uso de memória podem indicar comportamento anômalo em um sistema; *Kbouncer* [Pappas et al. 2013] e *Ropecker* [Cheng et al. 2014] são usadas na detecção de ataques por ROP.

Uma das maiores limitações das abordagens existentes é a forma de coleta de dados, por vezes feita de maneira *system-wide*, sem monitoramento individual de processos e de forma intrusiva (injeção de código). Ressalta-se ainda que nenhuma delas monitora a interação do *malware* com o sistema, mas sim seu fluxo de execução. Portanto, superar os desafios atuais é a maior contribuição do presente artigo.

3. Projeto do Sistema Proposto

Nesta seção, mostra-se as escolhas de projeto e os detalhes de implementação do sistema proposto para análise transparente assistida por *hardware* de programas maliciosos.

3.1. Modelo de Ameaças e Decisões Tomadas

Assume-se que os exemplares sob análise possuem as seguintes características: atuam no nível do usuário, isto é, não carregam *drivers* no sistema operacional alvo; podem ser equipados com técnicas de anti-análise; e interagem com o sistema operacional alvo por meio de suas chamadas de APIs. Este modelo de ameaça embasa uma série de decisões de projeto, apresentadas abaixo.

- **Análise em *userland***: a coleta de informações dos mecanismos de monitoramento só pode ser feita em modo *kernel*, seja pela necessidade de se lidar com páginas do sistema, seja pelo acesso a registradores especiais (MSR). De modo geral, um *driver* é responsável por essa tarefa. A garantia de que exemplares analisados estarão em *userland* elimina a possibilidade de subversão do mecanismo de análise, já que este estará em um nível mais privilegiado [Rossow et al. 2012].
- **Análise transparente**: as ferramentas com suporte de *hardware* exigem ambiente real (*bare-metal*) para a sua execução, fazendo que muitas técnicas de anti-análise falhem, sobretudo as baseadas em detecção de máquina virtual. Contudo, outras técnicas, como a verificação da presença de *debuggers*, ainda poderiam ser efetivas em evitar a análise. Os mecanismos PEBS e LBR/BTS são habilitados via registradores de controle que também controlam o uso de registradores de *debug*, e a má configuração destes poderia permitir a identificação de seu uso. Para evitar este cenário, não foram utilizadas *flags* que habilitassem o uso de registradores de *debug*, como no caso de *single-step on branch*.

[1] https://perf.wiki.kernel.org/index.php/Main_Page

- **Uso de APIs de sistema**: considerando que o mecanismo de *branch* monitora endereços-alvo, se estes forem conhecidos de antemão, pode-se relacionar o alvo do *branch* com a biblioteca carregada no referido endereço. Se os exemplares usarem o mecanismo de chamadas de sistema para sua execução, garante-se a reconstrução de todo o fluxo de chamadas destes.
- **Sistema operacional moderno**: a partir do Vista, sistemas operacionais da família Windows apresentam restrições às modificações que podem ser feitas no *kernel* para fins de monitoração. Técnicas alternativas são, em geral, baseadas em filtros limitados à monitoração de eventos providos por uma interface do SO. O sistema proposto monitora qualquer atividade no sistema, independentemente dos meios providos por este para tal fim. Para tanto, combinou-se o mecanismo de monitoração de *branch* com uma técnica de introspecção.

3.2. Implementação

Nesta seção, os detalhes específicos de implementação do sistema proposto são apresentados, considerando-se aspectos de funcionamento e interação com o sistema operacional alvo. O protótipo para testes foi desenvolvido em Windows 8 de 64 bits em execução sob processador Intel Core i5 com plataforma *Haswell*.

3.2.1. Captura de dados

Optou-se então por utilizar o modo BTS, que permite a gravação em páginas do sistema e a geração de uma interrupção para alertar quando o *buffer* está cheio. Além de mais de 16 endereços/registros (limite do modo LBR na geração atual de processadores, incorrendo em captura insuficiente e ineficiente de dados), pode-se capturar os dados sem o risco de perder qualquer instrução, o que ocorreria se a implementação fosse feita por *polling*. A interrupção gerada garante que os processos do sistema operacional encontram-se suspensos no momento da captura, permitindo a inspeção destes sem efeitos colaterais.

Um problema existente nas demais soluções de monitoramento é a não identificação da origem dos saltos. Para resolvê-lo, configurou-se o *threshold* de interrupção para apenas uma posição, ou seja, interrompe-se o fluxo a cada instrução de desvio para se obter o último processo em execução (gerador do desvio e, portanto, origem). O funcionamento do mecanismo de interrupção consiste no processador chamar a *Interrupt Service Routine* registrada na *Interrupt Descriptor Table* (IDT) na posição dada pelo vetor definido na *Advanced Programmable Interrupt Controller - Local Vector Table* (APIC-LVT).

Tal interrupção deveria, a princípio, ser entregue em modo *Fixed-Edge Delivery*. Contudo, o Windows já registra uma interrupção para seu próprio uso na faixa de valores do monitor e a alteração dessa rotina é impedida pelo mecanismo de proteção de *kernel*[2]. As alternativas encontradas incluem a desativação deste mecanismo e a utilização de funções não documentadas. Porém, visando minimizar qualquer subversão do sistema operacional alvo e tornar o protótipo mais portável, optou-se por alterar o modo de entrega da interrupção para *Non-Maskable Interrupt* (NMI)—uma interrupção especial para tratar exceções e erros cujo *handler* é nativo do sistema operacional.

[2]`http://technet.microsoft.com/pt-br/library/cc759759(v=ws.10).aspx`

3.2.2. Funcionamento do Sistema

Quando uma instrução de desvio é executada, o processador preenche o *buffer* fornecido com os dados referentes a ela e lança a interrupção. Esse monitoramento é totalmente executado por *hardware* e não depende de qualquer injeção de código no processo monitorado. Na prática, todos os processos são monitorados, já que o processador não é ciente de qual processo gerou aquele *branch*. A identificação do processo é realizada pelo tratador da interrupção através da função `PsGetCurrentProcess`[3];

A interrupção é tratada em *kernel* por meio de um *driver*, o qual é responsável por ler os valores do *buffer* e colocá-los em uma fila na ordem em que as interrupções foram geradas, para posterior análise, atuando como o servidor de uma arquitetura cliente-servidor. Nessa arquitetura, o cliente é uma aplicação independente em modo usuário que recebe os valores lidos pelo *driver*. O cliente também responde por todo o processo de coleta, introspecção e visualização de dados, além de filtrar e exibir informações apenas dos processos desejados, uma vez que as informações recebidas incluem dados de todo o sistema.

Essa abordagem "*system-wide*" deve-se aos exemplares de *malware* atuais serem desesenvolvidos modularmente e obterem funcionalidades adicionais por *download* de componentes, frequentemente dividindo a infecção entre diferentes processos. Embora através da introspecção seja possível interceptar a chamada de API da criação de processos, não é possível obter diretamente o PID do processo filho criado pelo exemplar. Para resolver esse problema, registrou-se uma *callback* de processos[4] para a identificação do processo-filho, permitindo ao cliente adicionar tal PID à lista de processos monitorados.

3.2.3. Introspecção

A tradução dos endereços-alvo dos desvios para informações de mais alto nível é um processo conhecido como introspecção. Através deste processo é possível, por exemplo, reconstruir o grafo das chamadas de função de um exemplar de *malware* (*call graph*). O primeiro passo da introspecção é descobrir o endereço-base das bibliotecas. Em sistemas operacionais modernos, os endereços de carregamento são aleatorizados a cada execução para dificultar ataques de injeção de código. A Tabela 1 mostra o endereço-base de bibliotecas em duas inicializações do sistema.

Tabela 1. Exemplo do efeito do mecanismo de aleatorização de endereços (ASLR) sobre os módulos dinâmicos em duas inicializações consecutivas do SO.

Biblioteca	ntdll.dll	KERNEL32.DLL	KERNELBASE.dll	NETAPI32.dll
Endereço 1	0xBAF80000	0xB9610000	0xB8190000	0xB6030000
Endereço 2	0x987B0000	0x98670000	0x958C0000	0x93890000

A solução encontrada para analisar *malware* adequadamente apesar do mecanismo de ASLR é obter os endereços das bibliotecas a cada reinicialização do sistema, através

[3]https://msdn.microsoft.com/en-us/library/windows/hardware/ff559933%28v=vs.85%29.aspx
[4]https://msdn.microsoft.com/en-us/library/windows/hardware/ff542860%28v=vs.85%29.aspx

da função `GetModuleHandle`[5]. A partir do endereço-base das bibliotecas, é preciso identificar qual das funções foi chamada, pois cada biblioteca pode conter diferentes funções. A Tabela 2 mostra diferentes funções da biblioteca `ntdll.dll`. Com tais endereços, a introspecção é então feita da seguinte forma: dado um endereço, obtém-se o endereço-base de uma biblioteca conhecida mais próxima, subtrai-se este do valor original e considera-se o valor resultante como *offset*. O processo descrito é ilustrado pela Figura 2.

Tabela 2. Exemplos de *offsets* das funções de biblioteca `ntdll.dll`

Função	***Offset***
NtCreateProcess	0x3691
NtCreateProcessEx	0x30B0
NtCreateProfile	0x36A1
NtCreateProfileEx	0x36B1
NtCreateResourceManager	0x36C1
NtCreateSemaphore	0x36D1
NtCreateSymbolicLinkObject	0x36E1
NtCreateThread	0x30C0
NtCreateThreadEx	0x36F1

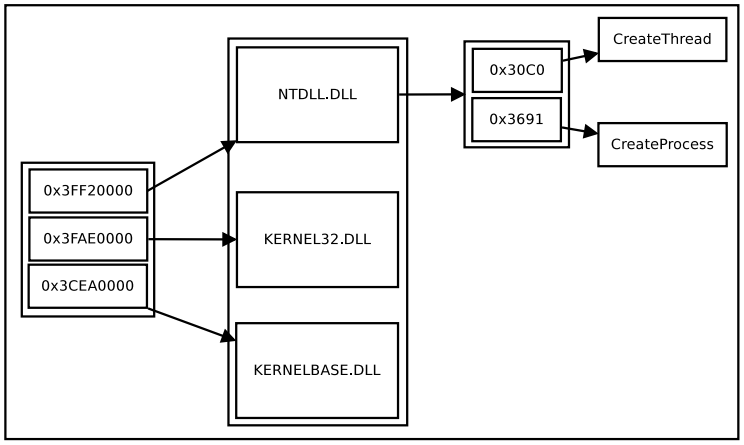


Figura 2. Diagrama de funcionamento do mecanismo de introspecção para associação de nomes de funções a endereços de módulos.

4. Aplicações, Testes e Resultados

Para validar o sistema, foram realizados testes baseados na geração de grafos de chamadas a partir da monitoração de códigos escolhidos para execução. O grafo de chamadas (*callgraph* ou CG) é um grafo onde os vértices representam funções e as arestas a relação entre elas (dependência, temporalidade etc.). Um dos usos de CG para detecção de códigos maliciosos é a identificação de variantes metamórficas com base em busca por isomorfismos [Martins et al. 2014]. No sistema proposto neste artigo, as funções chamadas são obtidas diretamente do mecanismo de monitoramento de *branch*, quando programado

[5]https://msdn.microsoft.com/pt-br/library/windows/desktop/ms683199%28v=vs.85%29.aspx

para capturar chamadas do tipo `CALL`. Como este método de captura atua de maneira ampla (*system-wide*), pode-se reconstruir o grafo incluindo tanto as chamadas diretas de funções (*step-over*) somente, quanto as chamadas internas a estas (*step-into*), ambos discutidos adiante. Destaca-se ainda que a obtenção do CG se dá a partir do comportamento observado durante a execução e não por *disassembly*. Logo, o sistema não se confunde na presença de *dead-code*[6]. Os resultados foram alcançados a partir da execução do código exibido pela Listagem 1 (executado no sistema pelo processo `NewToy.exe`).

Listagem 1. Código de exemplo para a reconstrução do CG.

```
1  scanf("%d",&n);
2  scanf("%s",val);
3  for(i=0;i<n;i++)
4    printf("%s\n",val);
```

4.1. *Step-Into*

Neste modo, considera-se que o analista "mergulha" em cada uma das bibliotecas e funções externas chamadas, podendo observar com alta granularidade aspectos internos da implementação, resultando em um CG ampliado, como mostrado na Figura 3.

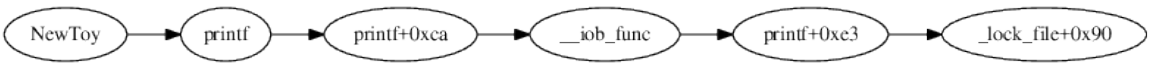


Figura 3. Visualização completa de trecho do CG.

4.2. *Step-Over*

Neste modo, o analista pode verificar as funções chamadas pelo programa sob monitoração, sem visualizar aspectos internos das bibliotecas e funções externas. Como o objetivo é ter uma visão global das chamadas, permite-se a definição de uma lista de funções de interesse, o que corresponderia à escolha das funções a serem interceptadas via *hooks* em sistemas de análise dinâmica tradicionais. O processamento consiste em realizar uma busca (como uma *Breadth First Search* - BFS) nos dados obtidos, onde os vértices correspondentes às funções de interesse, marcados em um caminho a partir da raiz, são ligados diretamente (transitividade). Desta vez, a execução resultou no CG reduzido da Figura 4.

Figura 4. Visualização do CG reduzido.

4.3. Reconstrução do CFG

O grafo de fluxo de controle (*Control Flow Graph* - CFG), representa todos os caminhos que podem ser percorridos por um programa em execução. O CFG é composto por blocos de instruções, denominados blocos básicos, limitados por instruções de desvio de fluxo [Appel and Palsberg 2003]. Esta estrutura tem uma associação natural com

[6]Código parte do executável ou ligado ao mesmo que não é acionado durante uma execução do mesmo.

as informações de desvios coletadas pelo sistema aqui proposto. Dado que as instruções executadas correspondem ao nível mais granular possível de análise, a obtenção do CFG de um programa é de grande valia para a análise de *malware*. As soluções adotadas para permitir a reconstrução do CFG a partir dos dados de *branch* estão detalhadas a seguir.

***Disassembly*.** A reconstrução do CFG requer todas as instruções existentes dentro de um bloco básico. Porém, o mecanismo monitor de *branch* só fornece informações acerca da primeira (alvo do *branch*) e da última (origem do *branch*) instrução de cada bloco. Para superar esta limitação, optou-se pelo *dump*—via `ReadProcessMemory`[7]—de todo o espaço de instruções contido entre dois *branches* consecutivos, obtendo-se os códigos de instruções como os exibidos na Listagem 2. Estes foram "disassemblados" e seu resultado pode ser visto na Listagem 3.

Listagem 2. Exemplo de *buffer* de instruções obtido a partir dos endereços fornecidos pelo mecanismo BTS.

```
1 \xff\x15\x0a\x11\x00\x00\x48\x8d\x0d\x9f\x11\x00\x00
```

Listagem 3. Conversão das instruções do *buffer* para *opcodes*.

```
1 0x1000 (size=6)   call    QWORD PTR [rip+0x110a]
2 0x1006 (size=7)   lea     rcx,[rip+0x119f]
```

***Plot* do CFG.** Para visualização da reconstrução do CFG, considera-se que os vértices do grafo são os endereços coletados pelo monitor de *branch*, enriquecidos com o *disassembly*, representando as instruções geradas, enquanto que as arestas são as transições entre os endereços coletados pelo monitor. A Figura 5 exibe a reconstrução do CFG a partir do código de exemplo, exibido na Listagem 4.

Listagem 4. Código de exemplo para a reconstrução do CFG.

```
1         a=0;
2         scanf("%d",&n);
3         for(i=0;i<n;i++)
4                 if(i%2==0)
5                         a++
6                 else
7                         a—
8                 printf("%d\n",a)
```

4.4. Testes com exemplares reais

Validados os testes de reconstrução do CG e do CFG, o sistema proposto foi usado em exemplares reais de *malware* sabidamente evasivos. A identificação destes se deu através da presença de funções de anti-análise de acordo com avaliação pela ferramenta `PEframe`[8]. A efetividade da evasão foi comprovada através da tentativa de execução destes em soluções de *sandbox* tradicionais. Todos os exemplares foram coretamente analisados pela solução proposta e os detalhes dos resultados obtidos (*traces*) podem ser encontrados na respectiva página do projeto[9].

[7]https://msdn.microsoft.com/pt-br/library/windows/desktop/ms680553%28v=vs.85%29.aspx
[8]https://github.com/guelfoweb/peframe
[9]https://sites.google.com/site/branchmonitoringproject/tools/tracer/evaluation/real-malware-tests

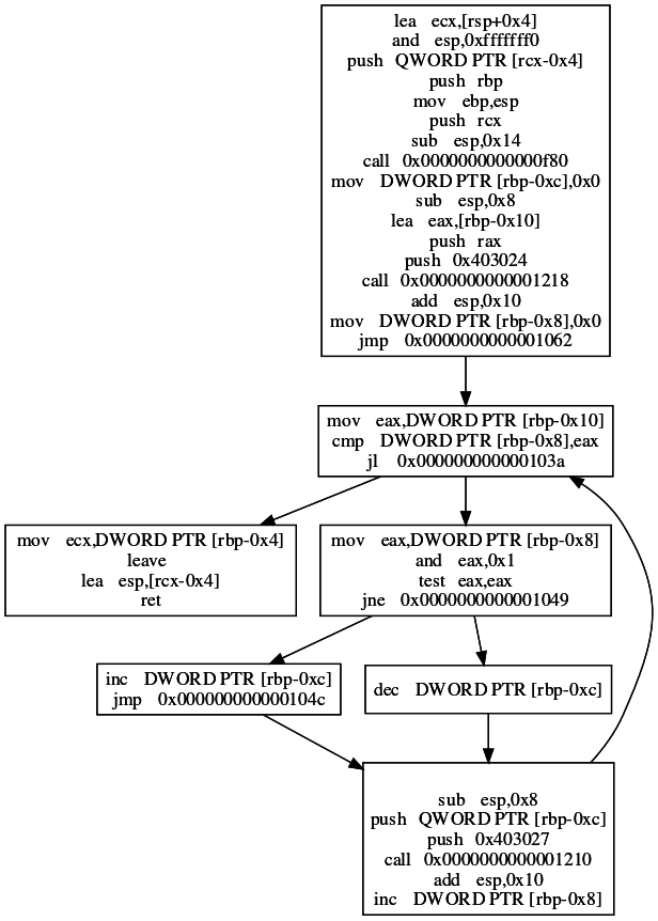


Figura 5. CFG reconstruído a partir da execução do código de exemplo.

5. Discussão

Vantagens e limitações da reconstrução do CFG. Esta abordagem, por sua natureza dinâmica, não exibe todos os caminhos possíveis de serem percorridos durante a execução do programa, o que pode ser considerado uma limitação. Entretanto, é possível assegurar a execução dos ramos exibidos no CFG (e a real instrução executada), pois o *dump* é feito nos dados coletados e não está sujeito ao desalinhamento de instruções. Ressalta-se que esta abordagem é capaz de exibir a execução de código gerado dinamicamente, pois é realizada a partir do *disassembly* do código de desvio executado pelo processador.

***Overhead* do sistema.** A detecção de sistemas de análise por meio de *timing* é uma técnica usada por autores de *malware* para impedir sua monitoração. Uma vantagem da monitoração via monitores de desempenho é o baixo *overhead*, pois trata-se de mecanismo de *hardware*—a captura dos dados por si só apresenta *overhead* teoricamente nulo. No entanto, o *overhead* total é proporcional à aplicação envolvida, sendo considerada quase nula quando se realiza apenas a exibição dos *branches* (captura), mas notável quando se realiza operações em tempo real, como o *dump* da memória. Sempre que possível, deve-se priorizar o processamento *offline* (p.ex., interpretação dos *opcodes* das instruções, que pode ser feito após a coleta dos dados de memória). Do contrário, recomenda-se a execução em um núcleo do processador diferente do monitorado, tal como na abordagem de [Quinn 2012]. Para validar tais afirmações, desenvolveu-se um programa simples

que realiza um milhão de *branches*. Comprovou-se com esse teste a nulidade prática do *overhead* imposto pela ativação tanto do mecanismo BTS quanto LBR. Verificou-se ainda que os mecanismos de interrupção e de *polling* afetam o sistema em 14% e 26%, respectivamente. O processo de introspecção impõe um custo adicional de 26% quando em execução no mesmo núcleo e zero quando em um núcleo diferente. Pôde-se verificar ainda diferenças entre os sistemas operacionais Windows e Linux. Estes resultados podem ser verificados na página do projeto[10], por questões de espaço.

Desafios. Dado o nível de abstração do monitoramento provido pelo mecanismo de monitoramento de *branch*, há desafios a serem superados. Por exemplo, a interpretação dos dados por si só apresenta desafios, pois requer grande compreensão da arquitetura sob análise (entender o funcionamento do sistema é essencial para avaliar quando os *branches* foram tomados e quando não). Além deste fator, algumas situações em especial restringem a análise. Por exemplo, a transferência da execução para o *kernel* causa perda de controle do endereçamento de retorno, uma vez que o mecanismo de BTS está desabilitado no nível deste. Embora a perda do endereço de retorno não afete nenhuma das abordagens de geração do traço de chamadas, pois elas são feitas para bibliotecas de modo usuário, tal ausência pode criar um *gap* para reconstrução do CFG caso o processo diretamente invoque *syscalls* ou faça uso de comunicação inter-processos (IPC), o que não está de acordo com o modelo de ameaça definido. A habilitação do mecanismo para captura em *kernel* pode ser abordada em trabalhos futuros, porém requer esforço adicional de interpretação—os dados precisam ser filtrados, uma vez que a proposta não considera ameaças em nível privilegiado.

Portabilidade: Embora desenvolvida na plataforma Windows, a solução pode ser aplicada nas demais plataformas e versões deste sistema, dado que o monitor em questão é um recurso do processador, e não da plataforma. O tratamento adequado dos dados deste monitor em cada plataforma exige o uso das bibliotecas compatíveis a cada uma delas.

Aplicações Futuras: A solução desenvolvida pode ser empregada para vários propósitos relativos a análise de *malware*, da identificação de novas ameaças a criação de contra-medidas. De imediato, planeja-se o agrupamento dos resultados de análise (*traces*) para o propósito de identificação de variantes evasivas.

6. Conclusão

Neste artigo, apresenta-se uma proposta de sistema para análise dinâmica de *malware* com base no suporte de *hardware* provido pela tecnologia de monitoramento de *branch* da Intel. A solução é capaz de analisar os exemplares de forma transparente, sem qualquer injeção de código e com baixo *overhead*. Os resultados obtidos mostram o funcionamento desta na reconstrução do grafo de chamadas e de fluxo de controle, permitindo análises mais granulares mesmo em sistemas operacionais modernos[11] e com restrições de *patching*. Está em andamento um trabalho baseado no sistema proposto que envolve a análise de exemplares de *malware* evasivos, isto é, equipados com técnicas de anti-forense.

[10]https://sites.google.com/site/branchmonitoringproject/tools/tracer/evaluation/overhead-measures

[11]Entende-se por modernos *kernels* do Windows a partir da versão NT 6.x de 64 bits

Agradecimentos

Os autores agradecem o apoio recebido do Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq via Projeto MCTI/CNPq/Universal-A 14/2014 (Processo 444487/2014-0) e da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES, em especial via Projeto FORTE - Forense Digital Tempestiva e Eficiente (Processo: 23038.007604/2014-69 - Edital 24/2014 - Programa Ciências Forenses).

Referências

Appel, A. W. and Palsberg, J. (2003). *Modern Compiler Implementation in Java*. Cambridge University Press, New York, NY, USA, 2nd edition.

Balzarotti, D., Cova, M., Karlberger, C., Kruegel, C., Kirda, E., and Vigna, G. (2010). Efficient detection of split personalities in malware. In *NDSS 2010, 17th Annual Network and Distributed System Security Symposium, February 28th-March 3rd, 2010, San Diego, USA*, San Diego, UNITED STATES.

Bruening, D., Zhao, Q., and Amarasinghe, S. (2012). Transparent dynamic instrumentation. In *8th ACM SIGPLAN/SIGOPS Conf. Virtual Execution Environments*, VEE '12, pages 133–144.

Carsten Willems, Ralf Hund, T. H. (2012). Cxpinspector: Hypervisor-based, hardware-assisted system monitoring. Technical report, Horst Görtz Institute for IT Security.

CERT.br (2015). Estatísticas do cert.br. `http://www.cert.br/stats/incidentes/`. Acessado em junho/2016.

Chen, X., Andersen, J., Mao, Z. M., Bailey, M., and Nazario, J. (2008). Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 177–186.

Cheng, Y., Zhou, Z., Miao, Y., Ding, X., DENG, H., et al. (2014). Ropecker: A generic and practical approach for defending against rop attack. *Network and Distributed System Security Symposium*.

Deng, Z., Zhang, X., and Xu, D. (2013). Spider: Stealthy binary program instrumentation and debugging via hardware virtualization. In *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '13, pages 289–298, New York, NY, USA. ACM.

Dinaburg, A., Royal, P., Sharif, M., and Lee, W. (2008). Ether: Malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 51–62, New York, NY, USA. ACM.

FEBRABAN (2015). FEBRABAN dá dicas de segurança eletrônica. `http://www.febraban.org.br/Noticias1.asp?id_texto=2758`. Acessado em junho/2016.

Guarnieri, C. (2013). Cuckoo sandbox. `http://www.cuckoosandbox.org/`. Acessado em junho/2016.

Intel (2015). Intel vtune. `https://software.intel.com/en-us/intel-vtune-amplifier-xe`. Acessado em junho/2016.

Kompalli, S. (2014). Using existing hardware services for malware detection. In *Security and Privacy Workshops (SPW), 2014 IEEE*, pages 204–208.

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. (2005). Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, pages 190–200, New York, NY, USA. ACM.

Martins, G. B., Souto, E., de Freitas, R., and Feitosa, E. (2014). Estruturas virtuais e diferenciação de vértices em grafos de dependência para detecção de malware metamórfico. *Anais do SBSEG 2014*.

Moser, A., Kruegel, C., and Kirda, E. (2007). Limits of static analysis for malware detection. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 421–430.

Nethercote, N. and Seward, J. (2003). Valgrind: A program supervision framework. *Electronic Notes in Theoretical Computer Science*, 89(2):44 – 66.

Nguyen, A. M., Schear, N., Jung, H., Godiyal, A., King, S. T., and Nguyen, H. D. (2009). Mavmm: Lightweight and purpose built vmm for malware analysis. In *Proceedings of the 2009 Annual Computer Security Applications Conference*, ACSAC '09, pages 441–450, Washington, DC, USA. IEEE Computer Society.

Pappas, V., Polychronakis, M., and Keromytis, A. D. (2013). Transparent rop exploit mitigation using indirect branch tracing. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 447–462, Berkeley, CA, USA. USENIX Association.

Quinn, R. (2012). *Detection of malware via side channel information*. PhD thesis, Binghamton University.

Reina, A., Fattori, A., Pagani, F., Cavallaro, L., and Bruschi, D. (2012). When hardware meets software: A bulletproof solution to forensic memory acquisition. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 79–88, New York, NY, USA. ACM.

Rossow, C., Dietrich, C. J., Kreibich, C., Grier, C., Paxson, V., Pohlmann, N., Bos, H., and van Steen, M. (2012). Prudent Practices for Designing Malware Experiments: Status Quo and Outlook . In *Proceedings of the 33rd IEEE Symposium on Security and Privacy (S&P)* , San Francisco, CA.

Sikorski, M. and Honig, A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, CA, USA, 1st edition.

Vasudevan, A. and Yerraballi, R. (2005). Stealth breakpoints. In *Proceedings of the 21st Annual Computer Security Applications Conference*, ACSAC '05, pages 381–392, Washington, DC, USA. IEEE Computer Society.

Vasudevan, A. and Yerraballi, R. (2006). Spike: Engineering malware analysis tools using unobtrusive binary-instrumentation. In *Proceedings of the 29th Australasian Computer Science Conference - Volume 48*, ACSC '06, pages 311–320, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Willems, C., Holz, T., and Freiling, F. (2007). Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, 5:32–39.

Zhang, F., Leach, K., Stavrou, A., Wang, H., and Sun, K. (2015). Using hardware features for increased debugging transparency. In *2015 IEEE Symposium on Security and Privacy*, pages 55–69.

Zhang, F., Leach, K., Sun, K., and Stavrou, A. (2013). Spectre: A dependable introspection framework via system management mode. In *Proceedings of the 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, DSN '13, pages 1–12, Washington, DC, USA. IEEE Computer Society.

SBSeg16

XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

Niterói, RJ, novembro de 2016

CTDSeg – IV Concurso de Teses e Dissertações em Segurança

CCA1-secure somewhat homomorphic encryption

Eduardo Morais[1], **Diego F. Aranha**[1], **Ricardo Dahab**[1]

[1]Institute of Computing – University of Campinas (Unicamp)

{dfaranha, rdahab}@ic.unicamp.br, emorais@lasca.ic.unicamp.br

Abstract. *This paper proposes the combination of homomorphic encryption and verifiable computation to avoid key recovery attacks and achieve CCA1-secure constructions of Somewhat Homomorphic Encryption (SHE) schemes described in the literature. We also provide concrete parameters, based on the best-attack analysis, concluding that the approximate greatest common divisor (AGCD) [van Dijk et al. 2010] family of SHE schemes may be the best implementation choice under certain circumstances.*

1. Introduction

Homomorphic encryption has been a topic of great interest for the Cryptology community over the last few years, since Craig Gentry's breakthrough in 2009 [Gentry 2009]. Ciphertexts generated by a homomorphic encryption scheme can be algebraically manipulated, i. e. it is possible to add and multiply ciphertexts preversing the same operations over the corresponding plaintexts. A *Fully Homomorphic Encryption* (FHE) is one that allows us to run arbitrary algorithms over encrypted data. In other words, given an arbitrary algorithm $\mathcal{A}$, it is possible to find an equivalent algorithm $\mathcal{A}'$ such that, if we consider the input parameters of $\mathcal{A}'$, say $(c_1, \ldots, c_t)$, we have that they correspond to the componentwise encryption of the input parameters of $\mathcal{A}$, say $(m_1, \ldots, m_t)$. Furthermore, we have that the output of $\mathcal{A}'$ is an encryption of the output of $\mathcal{A}$ when executed using input arguments as described above. Although Fully Homomorphic Encryption (FHE) is yet to become practical, Somewhat Homomorphic Encryption (SHE) schemes can be used to construct practical applications. An important drawback, however, is the fact that all but one of the SHE schemes described in the literature are susceptible to *key recovery attacks*, a concrete threat in many scenarios. In this work, we investigate how verifiable computation (see Section 1.2) can be combined with homomorphic encryption in order to avoid key recovery attacks. Indeed, we show that it is possible to achieve CCA1-security for the homomorphic evaluation of quadratic multivariate polynomials.

1.1. Main contributions

During the PhD program I have participated in a research project about efficient implementation of elliptic curve protocols for Android architecture, giving rise to two publications [Braga and Morais 2014, Braga et al. 2015]. Since this research area is not related to homomorphic encryption, we are not going to explore it further here.

We have contributed in a book chapter, that corresponds to an introduction to Lattice-based Cryptography [Barreto et al. 2014]. This work derived from a short course in 2013 [Barreto et al. 2013] and gave rise to a technical report [Morais et al. 2016b]. We have also contributed with an introduction to homomorphic encryption [Dahab and Morais 2012], which is the text of a short course that gave rise to another technical report [Morais et al. 2016a].

One of the main results of my thesis is a key-recovery attack for the NTRU-based family of SHE schemes [Dahab et al. 2015]. We then contributed constructively to this line of research, showing how verifiable computation can be used to construct CCA1-secure SHE schemes [Morais et al.], for the case of quadratic multivariate polynomial functions.

1.2. Homomorphic verifiable computation

Although homomorphic encryption is a very flexible cryptographic primitive, when applied to the cloud computing scenario it lacks an important property: the ability to verify if a given homomorphic computation corresponds to what the client desired. A verifiable computation scheme could solve this problem, provided two requirements are met. First, the cloud must not spend much more time to perform the verifiable computation when compared to the non-verifiable solution. Second, the client must be able to verify the result faster than the time it takes to perform the entire computation by himself. There are proposals [Gennaro et al. 2010, Chung et al. 2010] that use homomorphic encryption to construct a verifiable scheme, because it is possible to offer input and output privacy, since both are encrypted. However, the underlying security model does not allow verification queries. Recently, Fiore, Gennaro and Pastro [Fiore et al. 2014] proposed a new construction that does allow verification queries, improving on the security model. They showed how to solve practical problems, such as computing quadratic multivariate polynomials over encrypted data, which can be used to homomorphically compute statistical functions. We remark that this application requires only one level of multiplications, which is an important characteristic to be considered in order to calculate the parameters of the underlying SHE scheme.

Definition 1.1. *A verifiable computation scheme $\mathcal{VC}$ is defined by the algorithms* $(\text{KEYGEN}, \text{PROBGEN}, \text{COMPUTE}, \text{VERIFY})$*, as follows:*

Key generation. *Algorithm* $\text{KEYGEN}(1^\lambda, f)$ *generates secret key* $\mathtt{sk}$ *and evaluation key* $\mathtt{esk}$*.*

Problem generation. *Using secret key* $\mathtt{sk}$*, algorithm* PROBGEN *receives as input ciphertext c_i and computes the corresponding authentication tag σ_i, such that* $\sigma_i = \text{AUTH}_{vk}(c_i, (\cdot, i))$*.*

Verification. *Given the secret* $\mathtt{sk}$*, tag σ and ciphertext c, we have that* $\text{VERIFY}_{sk}(\sigma, c)$ *returns* 1 *if* $c = f([c_i])$ *and* $\sigma = \text{AUTH}_{vk}(c, (\cdot, i))$*. Otherwise, it returns* 0*.*

Evaluation. *Given $\sigma_1, \ldots, \sigma_t$ and the description of function f, algorithm* $\text{COMPUTE}_{esk}([\sigma_i], \Delta, f)$ *returns the authentication tag σ that corresponds to the ciphertext* $c = \text{EVAL}_{edk}([c_i], f)$ *obtained by running the* EVAL *algorithm from the underlying homomorphic encryption scheme. We say that the $\mathcal{VC}$ scheme is* correct *if* $\text{VERIFY}_{sk}(\sigma, c)$ *outputs* 1*.*

1.3. Homomorphic encryption

Informally, homomorphic encryption provides the possibility of having a pair of encryption and decryption functions, ENC, DEC, that allows the computation of a function f on an encrypted text c, such that $\text{DEC}(f(c)) = f(m)$, where $c = \text{ENC}(m)$. That is, allowing

functions to be computed on encrypted texts without the need for decrypting them first. The following two definitions formalize this notion.

Definition 1.2. *Correctness. A scheme* $\mathcal{E}(\text{KEYGEN}, \text{DEC}, \text{ENC}, \text{EVAL})$ *is* correct *if, for a determined circuit* $\mathbf{C}$ *and every key pair* (sk, pk)*, where* sk *is the private key and* pk *is the public key generated by* KEYGEN*, any message tuple* $\overline{m} = \langle m_1, \ldots, m_t \rangle$ *and corresponding ciphertexts* $\overline{c} = \langle c_1, \ldots, c_t \rangle$*, that is,* $c_i = \text{ENC}_{pk}(m_i)$ *for* $1 \leq i \leq t$*, then we have that*

$$\text{DEC}_{sk}(\text{EVAL}_{pk}(\mathbf{C}, \overline{c})) = \mathbf{C}(\overline{m}).$$

Furthermore, algorithms KEYGEN*,* DEC*,* ENC *and* EVAL *must have polynomial complexity.*

Definition 1.3. *Fully Homomorphic Encryption. A scheme* $\mathcal{E}$ *is* correct for a class $\mathbf{S}_\mathbf{C}$ *of circuits, if it is correct for each* $\mathbf{C} \in \mathbf{S}_\mathbf{C}$*. Moreover,* $\mathcal{E}$ *is denominated* fully homomorphic *if it is correct for every algebraic circuit, or, equivalently, if it is correct for every Boolean circuit.*

SHE schemes correspond to homomorphic encryption schemes that are correct for circuits whose multiplicative depth is limited by a certain upper bound, denoted by ℓ.

1.4. Security model

We say that a cryptosystem is secure against *chosen ciphertext attacks* (CCA2) if there is no polynomial time adversary that can win the following game with non-negligible probability.

Setup. The challenger obtains $(\mathtt{sk}, \mathtt{pk}) = \text{KEYGEN}(\lambda)$, where λ is a security parameter, and sends $\mathtt{pk}$ to the adversary $\mathcal{A}$.

Queries. $\mathcal{A}$ sends ciphertexts to the challenger, before or after the challenge, who returns the corresponding plaintexts.

Challenge. The adversary randomly generates two plaintexts $m_0, m_1 \in \mathcal{M}$ and sends to the challenger, who then randomly chooses a bit $b \in \{0, 1\}$ and computes the ciphertext $c = \text{ENC}_{\mathtt{pk}}(m_b)$. The challenger sends c to $\mathcal{A}$.

Answer. $\mathcal{A}$ sends a bit b' to the challenger and wins the game if $b' = b$.

If we allow queries only before the challenge, we say that the cryptosystem is secure against CCA1 adversaries (lunchtime attacks). Queries can be interpreted as accesses to a *decryption oracle*. If, instead, we only allow access to an *encryption oracle*, namely the adversary can choose any message to be encrypted under the same key pair, then we say that the cryptosystem is secure against *chosen plaintext attacks* (CPA).

In homomorphic encryption, it is impossible to achieve CCA2 security, because the adversary can simply add to the encrypted message some encryption of zero, which can be obtained by querying the encryption oracle, and send it back to the decryption oracle. Many FHE schemes have as public value an encryption of the private key bits, which can be sent to the decryption oracle before the challenge, making such schemes insecure

against CCA1 adversaries. Indeed, a *key recovery* attack is stronger than a CCA1 attack; also, Loftus et al [Loftus et al. 2011] showed that Gentry's construction over ideal lattices is vulnerable to key recovery attacks and presented the only somewhat homomorphic encryption scheme that is known to be CCA1-secure.

In 2015, Dahab, Galbraith and Morais showed that the NTRU-based family of SHE schemes is vulnerable to key recovery attacks [Dahab et al. 2015]. Hence, except for Loftus et al's [Loftus et al. 2011] scheme, no other known SHE proposal achieves CCA1 security.

2. The scheme

The secret key encryption scheme that can homomorphically compute the quadratic multivariate polynomial f over encrypted input is defined as follows:

Key generation. Given the description of the function f, let $\mathcal{E}_{\text{CPA}}$ be a CPA-secure secret-key homomorphic encryption scheme and let $\mathcal{VC}$ be a private and adaptively secure verifiable computation scheme as previously defined and containing the inherent message authentication algorithm HomMAC. We compute $(\texttt{dk}) = \mathcal{E}_{\text{CPA}}.\textsc{KeyGen}(1^\lambda, f)$ and $(\texttt{sk}, \texttt{esk}) = \mathcal{VC}.\textsc{KeyGen}(1^\lambda, f)$. The secret key is given by $(\texttt{dk}, \texttt{sk})$ and the evaluation key is given by $(\texttt{edk}, \texttt{esk})$.

Encryption. For $m \in \mathcal{M}$ and multi-label (Δ, i), if the multi-label was not previously used, compute $c = \mathcal{E}_{\text{CPA}}.\textsc{Enc}_{\texttt{dk}}(m)$, compute

$$\sigma = \text{HomMAC}.\textsc{Auth}_{\texttt{vk}}(c, \Delta, i)$$

and output $(c, \sigma, \Delta, i, f_{\text{ID}})$.

Decryption. For $(c, \sigma, \Delta, i, f^\star) \in \mathcal{C}$, when $f^\star$ is the identity function f_{ID}, if $\text{HomMAC}.\textsc{Ver}_{\texttt{vk}}(\sigma, c, \Delta, i)$ rejects then return $\perp$. When $f^\star = f$, if $\mathcal{VC}.\textsc{Verify}_{\texttt{sk}}(\sigma, \Delta, f)$ rejects then return $\perp$, otherwise return

$$m = \mathcal{E}_{\text{CPA}}.\textsc{Dec}_{\texttt{dk}}(c).$$

Evaluation. Given the evaluation key $\texttt{esk}$ and $([c_i], [\sigma_i], f)$, for $1 \leq i \leq t$, return (c, σ), where $(c, \sigma) = \mathcal{VC}.\textsc{Compute}_{\texttt{esk}}([c_i], [\sigma_i], f)$.

3. AGCD-based scheme

In this section we describe the construction of homomorphic encryption over the integers, which was originally proposed by Dijk, Gentry, Halevi and Vaikuntanathan [van Dijk et al. 2010], and was improved many times afterwards [Coron et al. 2014, Coron et al. 2012]. Among these improvements, we focus on batch computation by extending the original idea to apply the Chinese Remainder Theorem [Cheon et al. 2013]. The secret-key somewhat homomorphic cryptosystem is defined as follows:

Definition 1. *Let λ be the security parameter and consider the parameters ρ, η, γ as functions of λ. The algorithm* KeyGen *randomly generates the secret key dk as an odd integer p with bit-length η. To encrypt a message $m \in \mathbb{Z}_Q$, the algorithm* Enc *randomly chooses the integers r with ρ bits and q with γ/η bits and computes the ciphertext:*

$$c = qp + Qr + m.$$

The decryption algorithm computes $m = \text{DEC}_{dk}(c) = [c]_p \pmod{Q}$. *It is easy to see that the encryption is a ring homomorphism.*

Definition 2. *Consider the following distribution*

$$\mathcal{D}_{\gamma,\rho}(p) = \{qp + r \mid \quad q \leftarrow \mathbb{Z} \cap [0, 2^\gamma/p), \\ r \leftarrow \mathbb{Z} \cap (-2^\rho, 2^\rho)\}.$$

Given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$*, finding* p *is a problem called* approximate greatest common divisor (AGCD).

The AGCD problem was studied by Howgrave-Graham in the context of cryptanalysis [Howgrave-Graham 2001]. To obtain a secure cryptosystem, parameters ρ, η, γ must be chosen to resist against attacks described in the literature [Lepoint 2014].

Definition 3. *The symmetric scheme described above can be adapted to allow batch operations [Cheon et al. 2013] as follows. Let* ℓ *be the number of slots in the the plaintext space* $\mathcal{M}$ *and* w *for the bit-length of each slot. We assume that* (ℓ, w) *are fixed and public values. Also, we use secondary security parameters* (ρ, η, γ) *to describe the cryptosystem. Since these parameters are functions of the primary security parameter* λ*, and are intimately connected to the complexity of the best-known attacks to the AGCD problem, we postpone the concrete description in order to get a cleaner definition of the scheme.*

Key Generation. *The* $\text{KEYGEN}(1^\lambda, f)$ *algorithm chooses pairwise coprime integers* Q_j *with* w *bits, for* $1 \leq j \leq \ell$*, and pairwise coprime* p_j *with* η *bits, for* $0 \leq j \leq \ell$*. We have that* Q_j *represents the size of each plaintext slot, while the plaintext space is given by* $\mathcal{M} = \mathbb{Z}_{Q_1} \times \cdots \times \mathbb{Z}_{Q_\ell}$*. Note that* $\mathcal{M}$ *is isomorphic to* $\mathbb{Z}_Q$ *for* $Q = \prod_{j=1}^{\ell} Q_j$*. The algorithm computes* $p = \prod_{j=0}^{\ell} p_j$*. The ciphertext space is given by* $\mathcal{C} = \mathbb{Z}_p$*. The secret key is given by* $dk = [p_j]$ *and the evaluation key is* $edk = p$*.*

Encryption. *Given* $[m_j] \in \mathcal{M}$*, algorithm* $\text{ENC}_{dk}([m_j])$ *chooses a random integer* r_0 *in the interval* $(-p_0/2, p_0/2]$ *and the random integers* $r_1, \ldots, r_\ell$ *with* ρ *bits. The ciphertext is computed by*

$$c = \text{CRT}(r_0, [m_j + r_j Q_j]),$$

where CRT *returns the unique integer modulo* p *that is congruent to* r_0 *modulo* p_0 *and congruent to* $m_j + r_j Q_j$ *modulo* p_j*, for every* j*. Thus the output is equal to* $c = \text{ENC}_{dk}(m)$*.*

Decryption. *Given* $c \in \mathcal{C}$*, the* $\text{DEC}_{dk}(c)$ *algorithm computes*

$$m_j \equiv c \pmod{p_j} \pmod{Q_j},$$

for $1 \leq j \leq \ell$ *and outputs* $[m_j] = \text{DEC}_{dk}(c)$*.*

Evaluation. *Homomorphic operations are carried out by simply adding and multiplying integers modulo* p*.*

The construction makes use of the following parameters:

- γ is the bit-length of ciphertexts. This parameter must be large enough in order to avoid attacks against the AGCD problem, such as the ones derived from Coppersmith's method, as for example Howgrave-Graham and Cohn-Heninger attacks, the simultaneous Diophantine approximation strategy of Lagarias and Nguyen and Stern's orthogonal lattice attack. In summary, we have that these attacks lead to the condition $\gamma = \eta^2 \Omega(\lambda)$, as described in Tancrède Lepoint's PhD thesis [Lepoint 2014];

- η is the bit-length of secret key p_j. It must be large enough to accommodate the noise growth after homomorphic operations. However, it must also be quadratic in the security parameter in order to avoid the elliptic curve method (ECM) factorization attack;
- ρ is the bit-length of the noise r_j. This parameter must be chosen satisfying $\rho = \Omega(\lambda)$, such that the scheme resists attacks against the noise [Chen and Nguyen 2012].

4. Choice of parameters and conclusion

We followed the ideas presented in Tancrède Lepoint PhD thesis [Lepoint 2014] in order to choose parameters to our scheme. Firstly, we compute η to avoid the ECM factorization attack. Then we establish an upper bound for ρ and γ. Thus, we can compute the ciphertext size to resist against the orthogonal lattice attack by decreasing the value of the noise parameter ρ until it is secure against Chen and Nguyen's attack. Afterwards, since we have calculated the values of ρ, η and γ as in Table 1, we can obtain the scheme parameters ℓ and w, with which we can calculate the plaintext size and the overhead of the scheme. We implemented a routine in Sage to calculate these values and the result is shown in Table 2. To compute ℓ we had to decrease the size of the ciphertext from γ to $\gamma = \gamma' - (\ell - 1)\eta$. In [Cheon et al. 2013], the authors show that the $\text{AGCD}_{1,\gamma}$ problem, the usual approximate GCD problem, can be reduced to the $\text{AGCD}_{\ell,\gamma'}$ problem, i.e. the CRT-based construction defined in Section 3. Then we have recomputed the ciphertext size using the relation $\gamma' = 1.5\gamma$ and calculated the number of slots using $\ell = \gamma/2\eta$.

λ	ρ	η	γ (MBits)
80	96	351	1.78
112	94	475	3.27
128	92	603	5.28

Table 1. AGCD parameters

λ	γ' (MBits)	ℓ	w	m (KBits)	overhead
80	2.67	2535	67	170	15.70
112	4.90	3442	131	451	10.87
128	7.92	4378	197	862	9.18

Table 2. Low overhead configuration

If we smaller ciphertexts are desired, then we can use the relations $\gamma' = 1.1\gamma$ and $\ell = \gamma/10\eta$, resulting in a larger overhead, as shown in Table 3.

In this work we proposed the utilization of verifiable computation to mitigate the key recovery attack problem in the context of homomorphic encryption. The proposed scheme however can only evaluate quadratic multivariate polynomials. This is enough to compute some statistical functions over encrypted data under the CCA1 security model. Remarkably, regarding this scenario, our AGCD-based construction offers an interesting choice of parameters.

λ	γ' (MBits)	ℓ	w	m (KBits)	overhead
80	1.95	507	67	34	57.40
112	3.59	688	131	90	39.88
128	5.80	875	197	172	33.72

Table 3. Smaller ciphertext configuration

References

Barreto, P. S. L. M., Biasi, F. P., Dahab, R., López-Hernández, J. C., Morais, E. M., Oliveira, A. D. S., Pereira, G. C. C. F., and Ricardini, J. E. (2013). Introdução a criptografia pós-quântica. In de Castro Andrade, R. M., editor, *Minicursos do XIII Simpósio em SeguranÃ§a da Informação e de Sistemas Computacionais. 193ed.*

Barreto, P. S. L. M., Biasi, F. P., Dahab, R., López-Hernández, J. C., Morais, E. M., Oliveira, A. D. S., Pereira, G. C. C. F., and Ricardini, J. E. (2014). A panorama of post-quantum cryptography. In *Open Problems in Mathematics and Computational Science. 1ed.: Springer International Publishing*, pages 387–439.

Braga, A. M. and Morais, E. M. (2014). Implementation issues in the construction of standard and non-standard cryptography on android devices. In *Securware, The Eighth International Conference on Emerging Security Information, Systems and Technologies*, pages 144–150.

Braga, A. M., Morais, E. M., Schwab, D. C., Vannucci, A. L., and Zanco Neto, R. (2015). Integrated technologies for communication security and secure deletion on android smartphones. *ICQMN*, 8:28.

Chen, Y. and Nguyen, P. (2012). Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers. In Pointcheval, D. and Johansson, T., editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 502–519. Springer.

Cheon, J., Coron, J., Kim, J., Lee, M., Lepoint, T., Tibouchi, M., and Yun, A. (2013). Batch fully homomorphic encryption over the integers. In Johansson, T. and Nguyen, P., editors, *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 315–335. Springer Berlin Heidelberg.

Chung, K., Kalai, Y., and Vadhan, S. (2010). Improved delegation of computation using fully homomorphic encryption. In Rabin, T., editor, *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 483–501. Springer Berlin Heidelberg.

Coron, J., Lepoint, T., and Tibouchi, M. (2014). Scale-invariant fully homomorphic encryption over the integers. In Krawczyk, H., editor, *Public-Key Cryptography - PKC 2014*, volume 8383 of *Lecture Notes in Computer Science*, pages 311–328. Springer Berlin Heidelberg.

Coron, J., Naccache, D., and Tibouchi, M. (2012). Public key compression and modulus switching for fully homomorphic encryption over the integers. In Pointcheval, D. and Johansson, T., editors, *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 446–464. Springer Berlin Heidelberg.

Dahab, R., Galbraith, S., and Morais, E. (2015). Adaptive key recovery attacks on NTRU-based somewhat homomorphic encryption schemes. In Lehmann, A. and Wolf, S., editors, *Information Theoretic Security*, volume 9063 of *Lecture Notes in Computer Science*, pages 283–296. Springer International Publishing.

Dahab, R. and Morais, E. M. (2012). Encriptação homomórfica. In Santos, A. L. (Org.), S. A. O. M. C. O. and Gonçalves, P. A. S., editors, *Minicursos do XII Simpósio em Segurança da Informação e de Sistemas Computacionais. 12ed.*, pages 1–195.

Fiore, D., Gennaro, R., and Pastro, V. (2014). Efficiently verifiable computation on encrypted data. Cryptology ePrint Archive, Report 2014/202. `http://eprint.iacr.org/`.

Gennaro, R., Gentry, C., and Parno, B. (2010). Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Rabin, T., editor, *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer Berlin Heidelberg.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 169–178, New York, NY, USA. ACM.

Howgrave-Graham, N. (2001). Approximate integer common divisors. In *CaLC*, pages 51–66.

Kim, J., Lee, M. S., Y., A., and Cheon, J. (2013). CRT-based fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2013/057. `http://eprint.iacr.org/`.

Lepoint, T. (2014). *Design and Implementation of Lattice-Based Cryptography*. PhD thesis, École Normale Supérieure and University of Luxembourg.

Loftus, J., May, A., Smart, N. P., and Vercauteren, F. (2011). On CCA-secure somewhat homomorphic encryption. In *In Selected Areas in Cryptography*, pages 55–72.

Morais, E. M., Aranha, D. F., and Dahab, R. AGCD-based CCA1-secure somewhat homomorphic encryption using verifiable computation. [Submitted].

Morais, E. M., Aranha, D. F., and Dahab, R. (2016a). Homomorphic encryption. Technical Report 02, Institution of Computing. `http://www.ic.unicamp.br/~reltech/2016/16-02.pdf`.

Morais, E. M., Aranha, D. F., and Dahab, R. (2016b). Lattice-based cryptography. Technical Report 01, Institution of Computing. `http://www.ic.unicamp.br/~reltech/2016/16-01.pdf`.

van Dijk, M., Gentry, C., Halevi, S., and Vaikuntanathan, V. (2010). Fully homomorphic encryption over the integers. In *Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 24–43, Berlin, Heidelberg. Springer-Verlag.

Lyra2: Password Hashing Scheme with improved security against time-memory trade-offs*

Ewerton R. Andrade[1], **Marcos A. Simplicio Junior**[1]

[1] Laboratório de Arquitetura e Redes de Computadores (LARC)
Departamento de Engenharia da Computação e Sistemas Digitais (PCS)
Escola Politécnica da Universidade de São Paulo (Poli-USP) – São Paulo – Brazil.

{eandrade,mjunior}@larc.usp.br

Abstract. *To protect against brute force attacks, modern password-based authentication systems usually employ mechanisms known as Password Hashing Schemes (PHS). Basically, a PHS is a cryptographic algorithm that generates a sequence of pseudorandom bits from a user-defined password, allowing the user to configure the computational costs involved in the process aiming to raise the costs of attackers testing multiple passwords trying to guess the correct one. In this context, the goal of this research effort is to propose a novel and superior PHS alternative. Specifically, the objective is to improve the Lyra algorithm, a PHS built upon cryptographic sponges whose project counted with the authors' participation. The resulting solution, called Lyra2, preserves the efficiency and flexibility of Lyra, and it brings important improvements when compared to its predecessor: (1) it allows a higher security level against attack venues involving time-memory trade-offs; (2) it includes tweaks for increasing the costs involved in the construction of dedicated hardware to attack; (3) it balances resistance against side-channel threats and attacks relying on cheaper (and, hence, slower) storage devices. Besides describing the algorithm's design rationale in detail, the thesis also includes a detailed analysis of its security and performance in different platforms. It is worth mentioning that Lyra2 received a special recognition in the Password Hashing Competition (PHC).*

1. Introduction

User authentication is one of the most vital elements in modern computer security. Even though there are authentication mechanisms based on biometric devices ("what the user is") or physical devices such as smart cards ("what the user has"), the most widespread strategy still is to rely on secret passwords ("what the user knows"). This happens because password-based authentication remains as the most cost effective and efficient method of maintaining a shared secret between a user and a computer system (Chakrabarti and Singbal 2007).

Password-based systems usually employ some cryptographic algorithm that allows the generation of a pseudorandom string of bits from the password itself, known as a Password Hashing Scheme (PHS), or Key Derivation Function (KDF) (NIST 2009). Typically, the output of the PHS is employed in one of two manners (Percival 2009): it can be locally stored in the form of a "token" for future verifications of the password or used

* Full thesis and implementations available at http://lyra2.net/.

as the secret key for encrypting and/or authenticating data. Whichever the case, such solutions employ internally a one-way (e.g., hash) function, so that recovering the password from the PHS's output is computationally infeasible (Percival 2009; Kaliski 2000).

Despite the popularity of password-based authentication, the fact that most users choose quite short passwords leads to a serious issue: they commonly have much less entropy than typically required by cryptographic keys (NIST 2011). Such weak passwords greatly facilitate many kinds of "brute-force" attacks, such as dictionary attacks and exhaustive search (Chakrabarti and Singbal 2007; Herley et al. 2009), allowing attackers to completely bypass the non-invertibility property of the password hashing process. The feasibility of such attacks depends basically on the amount of resources available to the attacker, who can speed up the process by performing many tests in parallel.

2. Motivation

A straightforward approach for addressing this problem is to force users to choose complex passwords. This is unadvised, however, because such passwords would be harder to memorize and, thus, more easily forgotten or stolen due to the users' need of writing them down, defeating the whole purpose of authentication (Chakrabarti and Singbal 2007). For this reason, modern password hashing solutions usually employ mechanisms for increasing the *cost* of brute force attacks. Schemes such as PBKDF (Kaliski 2000) and bcrypt (Provos and Mazières 1999), for example, include a configurable parameter that controls the number of iterations performed, allowing the user to adjust the time required by the password hashing process.

A more recent proposal, scrypt (Percival 2009), allows users to control both processing time and memory usage, raising the cost of password recovery by increasing the silicon space required for running the PHS in custom hardware, or the amount of RAM required in a Graphics Processing Unit (GPU). Since this may raise the RAM costs of password cracking to unbearable levels, attackers may try to trade memory for processing time, discarding (parts of) the memory used and recomputing the discarded information when it becomes necessary (Percival 2009). The exploitation of such time-memory trade-offs (TMTO) leads to the hereby-called *low-memory attacks*. Another approach that might be used by attackers trying to reduce the costs of password cracking is to use low-cost (and, thus, slower) storage devices for keeping all memory used in the legitimate process, using the spare budget to run more tests in parallel and, thus, compensating the lower speed of each test; we call this approach a *slow-memory attack*.

Besides the need for protection against low- and slow-memory attacks, there is also interest in the development of solutions that are safe against side-channel attacks, in especial the so-called *cache-timing attacks*. Basically, a cache-timing attack is possible if the attacker can observe a machine's timing behavior by monitoring its access to cache memory (e.g., the occurrence of cache-misses), building a profile of such occurrences for a legitimate password hashing process (Forler et al. 2013; Bernstein 2005). Then, at least in theory, if the password being tested does not match the observed cache-timing behavior, the test could be aborted earlier, saving resources. Although this class of attack has not been effectively implemented in the context of PHSs, it has been shown to be effective, for example, against certain implementations of the Advanced Encryption Standard (AES) (NIST 2001) and RSA (Rivest et al. 1978).

The considerable interest by the research community in developing new (and better) password hashing alternatives has recently even led to the creation of a cryptographic competition with this specific purpose, the Password Hashing Competition (PHC 2013).

3. Goals and Original Contributions

Aiming to address this need for stronger alternatives, our early studies led to the proposal of Lyra (Almeida et al. 2014), a mode of operation of cryptographic sponges (Bertoni et al. 2007; Bertoni et al. 2011) for password hashing. In this research, we propose an improved version of Lyra, called simply Lyra2. Basically, Lyra2 preserves the flexibility and efficiency of Lyra, including: (1) the ability to configure the desired amount of memory and processing time to be used by the algorithm; (2) the capacity of providing a higher memory usage than what is obtained with scrypt for a similar processing time.

In addition, it brings important security improvements when compared to its predecessor: (1) it allows a higher security level against attack venues involving time-memory trade-offs (TMTO); (2) it includes tweaks to increase the costs involved in the construction of dedicated hardware for attacking the algorithm (e.g., FPGAs or ASICs); (3) it balances resistance against side-channel threats and attacks relying on cheaper (and, hence, slower) storage devices. For example, the processing cost of memory-free attacks against the algorithm grows exponentially with its time-controlling parameter, surpassing scrypt's quadratic growth in the same conditions. Hence, with a suitable choice of parameters, the attack approach of using extra processing for circumventing (part of) the algorithm's memory needs becomes quickly impractical. In addition, for an identical processing time, Lyra2 allows for a higher memory usage than its counterparts, further raising the costs of any possible attack venue.

4. Brief overview of the state-of-the-art

Arguably, the main password hashing solutions currently in use are (PHC 2013): PBKDF2 (Kaliski 2000), bcrypt (Provos and Mazières 1999) and scrypt (Percival 2009). Nevertheless, until recently scrypt was the only solution available in the literature exploring both memory and processing costs. This scenario changed with Lyra (Almeida et al. 2014), which also introduced improvements such as: the decoupling of the memory and processing parameters, giving further flexibility to users; the usage of a single underlying function (a cryptographic sponge) rather than the two employed in scrypt; higher resistance against attacks exploiting time-memory trade-offs; and higher performance, allowing for a higher memory usage with a similar processing time.

Over the last years, some candidates submitted to the Password Hashing Competition (namely, Argon2, Catena, and yescrypt, besides Lyra2 itself, to cite only algorithms deemed as recommended) also allow memory and processing time to be configured, and some of them (especially Catena, Argon2 and Lyra2) try to provide a thorough security analysis. Due to space limitations, we do not analyze each solution in this document, limiting the discussion to the comparative performance and security assessment in Sections 5.1 and 5.2. Nevertheless, we refer the interested reader to (Almeida et al. 2014) for a discussion on scrypt and improvements proposed in Lyra (and inherited by Lyra2), and to the PHC official website (PHC 2013) for details on each submission.

5. Lyra2

As any PHS, Lyra2 takes as input a salt and a password to create a pseudorandom output. Internally, its memory is organized as a two-dimensional array whose cells are iteratively read and written, called simply the *memory matrix*. Hence, discarding a cell for saving memory leads to the need of recomputing it, until the point it was last modified, whenever it is accessed again. The matrix's initialization and visitation uses a stateful combination of the underlying sponge's absorbing, squeezing and duplexing operations, and was designed specifically to avoid password cracking using commom attack platforms such as GPUs and dedicated hardware. Also, users can define the matrix's size and the number of times its cells are revisited after initialization, allowing Lyra2's resource usage to be fine-tuned. Due to space limitations, here we limit the discussion of Lyra2's design to this brief overview, whereas the detailed design, some possible variants, as well as an extended security and performance analysis can be found at Lyra2's reference guide and PhD thesis — both available at `www.lyra2.net` —, and published papers (Andrade et al. 2016; Andrade and Simplicio Jr 2016).

5.1. Security

Table 1 summarizes the Lyra2's security analysis, providing a comparison with other Password Hashing Schemes that are considered part of the state of the art. We note that, among the results appearing in this table, only the data related to Lyra2 were actually analyzed and presented in our thesis, while the rest of the data shown were collected on the reference guides, manuals and articles describing and/or analyzing the other solutions, including also third-party analysis. Hence, we refer the interested reader to the original sources for details: Argon2 (Biryukov et al. 2016), battcrypt (Thomas 2014), Catena(Forler et al. 2013), Pomelo (Wu 2015), yescrypt (Peslyak 2015), scrypt (Percival 2009), and garbage collector attacks (Forler et al. 2014). As can be seen in this

[1]Provides protection when in its "read-write" mode (`YESCRYPT_RW`) (Forler et al. 2014).

Table 1. Security overview of the PHSs considered the state of art. In the TMTO analysis, the time parameter is denoted by T and the memory is denoted by R.

Algorithm	Time-Memory trade-offs (TMTO)	Slow Memory	Side Channel	Hardware and GPUs	Garbage Collector
Argon2i	for $R' = R/6$ $\approx 2^{15.5} \cdot T \cdot R$	—	✓	✓	✓
Argon2d	for $R' = R/6$ $\approx 2^{19.6} \cdot T \cdot R$	✓	✗	✓	✓
battcrypt	—	✓	✗	✓	✓
Catena	$O(1)$ $\Theta(R^{T+1})$	—	✓	✓	✓
Lyra	$O(1)$ $O(R^{T+1})$	✓	✗	✓	✓
Lyra2 [ours]	for $R' = R/2^{n+2}$, where $n \geq 0$ $O(2^{2nT}R^{2+n})$, for $n \gg 1$	✓	!	✓	✓
POMELO	—	✓	!	✓	✓
yescrypt	$O(1)$ $O(R^{T+1})$	✓	✗	✓	✓[1]
scrypt	$O(1)$ $O(R^2)$	✓	✗	!	✗

✓- Has protection; ✗- Has no protection; ! - Partial protection; — - Nothing declared.

table, Lyra2 is the only solution that provides at least partial protection against all known attacks.

5.2. Benchmarks

In our assessment of Lyra2's performance, we used an SSE-enabled implementation of Blake2b's (Aumasson et al. 2013) and BlaMka's (Andrade 2016, Sec. 4.4.1) compression function as the underlying sponge's f function of Lyra2's Algorithm. The actual implementations, as well as test vectors, are available at `www.lyra2.net`. Figure 1 shows the results for Lyra2 using Blake2b's as compression function, with $C = 256$, $\rho = 1$, and different T and R settings. As depicted, Lyra2 is able to execute in: less than 1 s when using up to 400 MB (with $R = 2^{14}$ and $T = 5$) or up to 1 GB of memory (with $R \approx 4.2 \cdot 10^4$ and $T = 1$); or in less than 5 s with 1.6 GB (with $R = 2^{16}$ and $T = 6$). The testbed employed is an Intel Xeon E5-2430 (2.20 GHz, $W = 64$ bits) equipped with 48 GB of DRAM, running Ubuntu 14.04 LTS 64 bits. The source code was compiled using gcc and g++ 4.9.2.

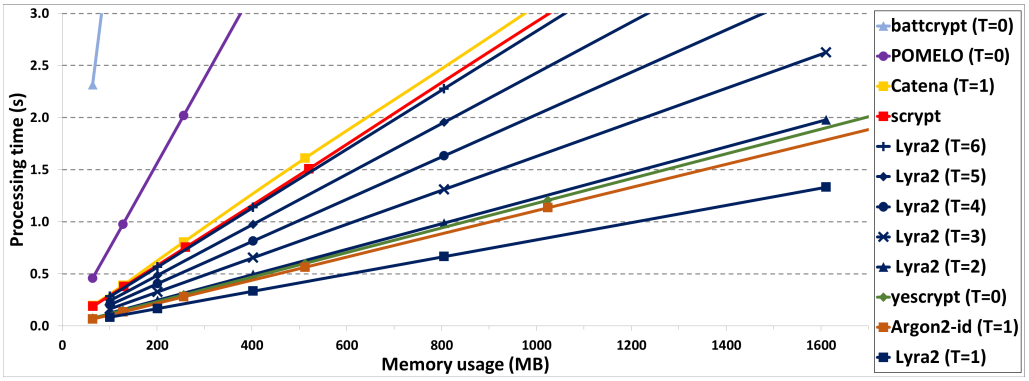


Figure 1. Performance of SSE-enabled Lyra2, for $C = 256$, $\rho = 1$, $p = 1$, and different T and R settings, compared with SSE-enabled scrypt and memory-hard PHC finalists with minimum parameters.

The same Figure 1 also compares Lyra2 with scrypt's SSE-enabled implementation available at `www.tarsnap.com/scrypt.html`, using the parameters suggested by scrypt's author in (Percival 2009) (namely, $b = 8192$ and $p = 1$). We also performed tests aiming to compare the performance of Lyra2 and the other 5 memory-hard PHC finalists: Argon2, battcrypt, Catena, POMELO, and yescrypt; in the absence of a complete set of optimized implementations for gathering such data, a reasonable approach is to consider the minimum parameters suggested by the authors of each scheme. The results, which basically confirm existing analysis done in (Broz 2014) and show that Lyra2 is a very competitive solution in terms of performance, are depicted in Figure 1.

5.3. Expected attack costs

Considering that the cost of DDR3 SO-DIMM memory chips is currently around U$8.6/GB (TrendForce 2015), Table 2 shows the cost added by Lyra2 with $T = 1$ and $T = 5$ when an attacker tries to crack a password in 1 year using the above reference hardware, for different password strengths; we refer the reader to (NIST 2011, Appendix A) for a discussion on how to compute the approximate entropy of passwords, and to (Percival 2009) for a more detailed description on how to obtain such cost estimates. Notice

Table 2. Memory-related cost (in U$) added by the SSE-enable version of Lyra2 with $T = 1$ and $T = 5$, for attackers trying to break passwords in a 1-year period using an Intel Xeon E5-2430 or equivalent processor.

Password	Memory usage (MB) for $T = 1$				Memory usage (MB) for $T = 5$			
entropy (bits)	200	400	800	1,600	200	400	800	1,600
35	315.1	1.3k	5.0k	20.1k	917.8	3.7k	14.7k	59.1k
40	10.1k	40.2k	160.7k	642.9k	29.4k	117.7k	471.9k	1.9M
45	322.7k	1.3M	5.1M	20.6M	939.8k	3.8M	15.1M	60.5M
50	10.3M	41.2M	164.5M	658.3M	30.1M	120.6M	483.2M	1.9B
55	330.4M	1.3B	5.3B	21.1B	962.4M	3.9B	15.5B	62.0B
Where: k (kilo) = $\times 1,000$; M (Million) = $\times 1,000,000$; B (Billion) = $\times 1,000,000,000$.								

that, in a real scenario, attackers would also have to consider costs related to wiring and energy consumption of memory chips, besides the cost of the processing cores.

6. Final Considerations

In this document, we presented the main aspects of Lyra2, a password hashing scheme (PHS) that allows legitimate users to fine tune memory and processing costs according to the desired level of security and resources available in the target user platform. To this end, Lyra2 builds on the properties of sponge functions operating in a stateful mode, creating a strictly sequential process. Indeed, the whole memory matrix of the algorithm can be seen as a huge state, which changes together with the sponge's internal state. In summary, the combination of a strictly sequential design for thwarting attacks with parallel platforms (e.g. GPUs or ASICs), the high costs of exploring time-memory trade-offs, and the ability to raise the memory usage beyond what is attainable with similar-purpose solutions (e.g., scrypt) for a similar security level and processing time make Lyra2 an appealing PHS solution.

6.1. Publications and other results

The following adoptions, publications, presentations, submissions and envisioned papers are a direct or indirect result of the research effort carried out during this thesis:

- *PHC special recognition*: Lyra2 received a special recognition for its elegant sponge-based design, and alternative approach to side-channel resistance combined with resistance to slow-memory attacks (PHC 2015).
- *BlaMka's adoption*: the winner of the PHC, Argon2, adopts BlaMka by default as its permutation function (Biryukov et al. 2016).
- *Lyra2's adoptions*: the Vertcoin electronic currency announced that it is migrating from scrypt to Lyra2 due to the excellent performance against custom mining hardware, as well as to the latter's ability to fine tune memory usage and processing time Lyra2 (a432511 2014; Day 2014). Furthermore, a few months ago one of the major bitcoin mining software on GPU, the Sgminer, added support to Lyra2 in its distribution package (Crypto Mining 2015).
- *Journal Articles*: in (Almeida et al. 2014), we present the Lyra algorithm, describing the preliminary ideas that gave rise to Lyra2; in (Andrade et al. 2016), we describe Lyra2 and provide a brief security and performance analysis of the algorithm.

- *Extended abstract*: in (Andrade and Simplicio Jr 2014b), we present the initial ideas and results of Lyra2, and took the opportunity of the event to discuss these results with other graduate students from the graduate program of Electrical Engineering (Computer Engineering area) at Escola Politecnica; in (Andrade and Simplicio Jr 2014a), we gave an oral presentation at LatinCrypt'14 with some preliminary results; and in (Andrade and Simplicio Jr 2016) we present the current status of our project.
- *Award*: recently, we received a best PhD project award due Lyra2's design and analysis (Andrade and Simplicio Jr 2016; ICISSP 2016).

6.2. Future works

As future work, we plan to provide a more detailed performance and security analysis of Lyra2 with different underlying sponges and in different platforms. Among those sponges, in especial we intend to evaluate multiplication-hardened solutions, including BlaMka, since this kind of solutions would increase the protections against attacks performed with dedicated hardware.

Another interesting subject of research involves adapting Lyra2's design to allow parallel tasks during the password hashing process. Finally, we can also evaluate the consequences to adopt a password independent approach on the Lyra2 structure, aiming make it totally resistant against cache-timing attacks.

Acknowledgements

This work was supported by CNPq, FDTE, FAPESP, and CAPES.

References

a432511 (2014). PoW Algorithm Upgrade: Lyra2 – Vertcoin. https://vertcoin.org/.

Almeida, L., Andrade, E., Barreto, P., and Simplicio, M. (2014). Lyra: Password-based key derivation with tunable memory and processing costs. *JCE*, 4(2):75–89.

Andrade, E. R. (2016). *Lyra2: Password Hashing Scheme with improved security against time-memory trade-offs*. PhD thesis, Escola Politécnica da Universidade de São Paulo (Poli-USP), São Paulo. Available from: http://lyra2.net/.

Andrade, E. R. and Simplicio Jr, M. A. (2014a). Lyra2: a password hashing schemes with tunable memory and processing costs. LATINCRYPT'14. Brazil.

Andrade, E. R. and Simplicio Jr, M. A. (2014b). Lyra2: Um Esquema de Hash de Senhas com custos de memória e processamento ajustáveis. 53–55, October, III WPG-EC. São Paulo, SP, Brazil. http://www2.pcs.usp.br/wpgec/2014/index.htm.

Andrade, E. R. and Simplicio Jr, M. A. (2016). Lyra2: Efficient Password Hashing with high security against Time-Memory Trade-Offs. In *Doctoral Consortium – Proceedings of 2nd ICISSP*, Rome, Italy. INSTICC. http://www.icissp.org/?y=2016.

Andrade, E. R., Simplicio Jr, M. A., Barreto, P. S. L. M., and Santos, P. C. F. d. (2016). Lyra2: efficient password hashing with high security against time-memory trade-offs. *IEEE Transactions on Computers*, PP(99).

Aumasson, J.-P., Neves, S., Wilcox-O'Hearn, Z., and Winnerlein, C. (2013). BLAKE2: simpler, smaller, fast as MD5. https://blake2.net/.

Bernstein, D. J. (2005). Cache-timing attacks on AES. Technical report, University of Illinois. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. (2007). Sponge functions. (ECRYPT Workshop). http://sponge.noekeon.org/SpongeFunctions.pdf.

Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. (2011). Cryptographic sponge functions - version 0.1. http://keccak.noekeon.org/.

Biryukov, A., Dinu, D., and Khovratovich, D. (2016). *Argon2: the memory-hard function for password hashing and other applications*. PHC, v1.3 of Argon2 edition.

Broz, M. (2014). Another PHC candidates "mechanical" tests – pub archives of PHC list.

Chakrabarti, S. and Singbal, M. (2007). Password-based authentication: Preventing dictionary attacks. *Computer*, 40(6):68–74.

Crypto Mining (2015). Updated Windows Binary of sgminer 5.1.1 With Fixed Lyra2Re Support – Crypto Mining Blog. http://cryptomining-blog.com/.

Day, T. (2014). Vertcoin (VTC) plans algorithm change to Lyra2 – Coinbrief. http://coinbrief.net/vertcoin-algorithm-change-lyra2/.

Forler, C., List, E., Lucks, S., and Wenzel, J. (2014). Overview of the Candidates for the Password Hashing Competition - And Their Resistance Against Garbage-Collector Attacks. Cryptology ePrint Archive. http://eprint.iacr.org/2014/881.

Forler, C., Lucks, S., and Wenzel, J. (2013). Catena: A memory-consuming password scrambler. Cryptology ePrint Archive. eprint.iacr.org/2013/525.

Herley, C., Oorschot, P. V., and Patrick, A. (2009). Passwords: If We're So Smart, Why Are We Still Using Them? In *Financial Cryptography and Data Security*, volume 5628 of *LNCS*, pages 230–237, Berlin, Germany. Springer Berlin Heidelberg.

ICISSP (2016). Previous awards. International Conference on Information Systems Security and Privacy – website. http://www.icissp.org/PreviousAwards.aspx.

Kaliski, B. (2000). *PKCS#5: Password-Based Cryptography Specif. v2.0 (RFC 2898)*.

NIST (2001). *Federal Information Processing Standard (FIPS 197) – Advanced Encryption Standard (AES)*. NIST, USA.

NIST (2009). *Special Publication 800-18 – Recommendation for Key Derivation Using Pseudorandom Functions*. NIST, USA.

NIST (2011). *Special Publication 800-63-1 – Electronic Authentication Guideline*. NIST.

Percival, C. (2009). Stronger key derivation via sequential memory-hard functions. In *BSD Conference*.

Peslyak, A. (2015). *yescrypt - a Password Hashing Competition submission*. Password Hashing Competition, Moscow, Russia, v1 edition. https://password-hashing.net/submissions/specs/yescrypt-v0.pdf.

PHC (2013). Password Hashing Competition. https://password-hashing.net/.

PHC (2015). Password Hashing Competition. https://password-hashing.net/\#phc.

Provos, N. and Mazières, D. (1999). A future-adaptable password scheme. In *USENIX'99*.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126.

Thomas, S. (2014). *battcrypt (Blowfish All The Things)*. Password Hashing Competition, Lisle, IL, USA, v0 edition. https://password-hashing.net/submissions/specs/battcrypt-v0.pdf.

TrendForce (2015). DRAM contract price (jan.13 2015). http://www.trendforce.com/price (visited on Jan.13, 2015).

Wu, H. (2015). *POMELO – A Password Hashing Algorithm (Version 3)*. Password Hashing Competition, Nanyang Ave, Singapore, v3 edition. https://password-hashing.net/submissions/specs/POMELO-v3.pdf.

Controle de acesso baseado em criptografia para a distribuição segura de conteúdo multimídia em Redes Centradas em Informação

Elisa Mannes[1]*
Orientador: Carlos Maziero

[1]Programa de Pós-graduação em Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brasil

{elisam,maziero}@inf.ufpr.br

Abstract. *The information-centric network represents a promising approach in which the content is the main entity. In these networks, the content can be stored on in-networks caches, generating concerns regarding unauthorized access to content. This work provides an access control solution in based on proxy re-encryption. In the proposed solution, contents are encrypted with a public key, unique to the content. By accessing the content, the user requests the provider for a re-encryption key, based on the user's public key and content's private key. The performance of the proposed solution was evaluated and the results confirm that the proposal is more secure, more efficient and make better use of caches and reveal a large computational load on the user's device. Thus, we propose an optimization that reduces the time of cryptographic operations on the user's device by 96%, making it an attractive scheme for content access control.*

Resumo. *As redes centradas em informação representam uma abordagem promissora na qual o conteúdo é a entidade principal. Nessas redes, o conteúdo pode ser armazenado em caches na rede, gerando preocupações com o acesso não autorizado aos conteúdos. Este trabalho propõe uma solução de controle de acesso fundamentada no esquema criptográfico de recifragem por proxy. Na solução proposta, os conteúdos são cifrados, com uma chave pública exclusiva do conteúdo. Ao acessar um conteúdo, o usuário solicita ao provedor uma chave de recifragem, calculada com base na chave pública do usuário e na chave privada do conteúdo. O desempenho da solução proposta foi avaliado e os resultados confirmam que a solução proposta é mais segura, mais eficiente e faz o melhor uso dos caches e revelam uma grande carga computacional no dispositivo do usuário. Assim, é proposta uma otimização que diminui o tempo das operações criptográficas em até 96%, tornando o esquema atrativo.*

1. Introdução

A distribuição de conteúdos populares, como os conteúdos multimídia, destaca-se como o serviço mais utilizado atualmente na Internet, revelando as deficiências de uma arquitetura inicialmente planejada para a comunicação ponto a ponto. Neste contexto, o protocolo IP é inerentemente ineficiente, uma vez que exige uma conexão entre máquinas específicas

*A tese está disponível em *http://www.inf.ufpr.br/elisam*

quando o interesse é no conteúdo, independente de sua localização [Ahlgren et al. 2012]. Assim, o protocolo IP impossibilita a adoção de diversos mecanismos atrativos, como o aproveitamento de cópias próximas ao usuário e a otimização do tráfego quando muitos usuários solicitam conteúdos idênticos. Desta forma, o descompasso entre o protocolo IP e as exigências da distribuição de conteúdo representa a motivação para uma arquitetura mais dinâmica, modular e adaptável, adequada para acomodar os novos padrões de uso.

As redes centradas em informação (*Information-Centric Network* - ICN) ganharam atenção considerável ao propor superar as deficiências atuais da Internet na distribuição de conteúdos modificando a principal entidade da camada de rede de máquinas para *conteúdos* [Ahlgren et al. 2012]. Assim, a ICN muda substancialmente a maneira como os dados fluem na Internet, pois a requisição, o roteamento e o encaminhamento são realizados a partir do *nome do conteúdo*. Como consequência, a rede pode armazenar cópias do conteúdo e usá-las para responder às requisições dos usuários, criando *caches* diretamente na rede e potencializando um melhor desempenho na entrega do conteúdo. Além disso, a requisição pelo nome do conteúdo permite que os roteadores realizem a agregação de tráfego, contribuindo para que somente uma cópia trafegue pela rede.

Apesar do *cache* representar benefícios óbvios, ele também impõe novos desafios relacionados ao controle de acesso. Por exemplo, o provedor de conteúdo não tem controle sobre quais *caches* possuem seus conteúdos e nem interage com os usuários que têm suas requisições atendidas pelos *caches*, dificultando a execução de políticas de acesso. Ao considerar a distribuição de conteúdo multimídia protegido, fica evidente a necessidade de assegurar que o provedor possa validar políticas de acesso para conteúdos em *cache*, pois tais serviços geralmente requerem um rigoroso controle das contas de usuários. Em geral, as soluções atuais não proveem o nível de segurança desejado ou demandam mudanças nas arquiteturas de ICN, necessitando uma solução que permita o controle por parte do provedor ao mesmo tempo que não limite o benefício dos *caches*.

Este trabalho apresenta uma solução de controle de acesso para ICN, que engloba três aspectos principais: (i) o conteúdo pode ser armazenado em qualquer dispositivo e recuperado por qualquer usuário, fazendo uso eficiente dos *caches* previstos no paradigma de ICN; (ii) os usuários são autorizados pelo provedor de conteúdo para acessá-lo; (iii) não há adição de novas entidades na rede para a aplicação ou validação de políticas de acesso. A solução proposta é fundamentada na recifragem por *proxy*, que prevê que um conteúdo cifrado com uma chave pública do usuário $u1$ pode ser transformado em um conteúdo cifrado com uma chave pública do usuário $u2$, usando uma *chave de recifragem*. Neste trabalho, *controle de acesso* é a garantia de que o provedor de conteúdo possa impor as *regras de acesso* desejadas, decidindo quem pode acessar o conteúdo que ele disponibiliza. Essa decisão é feita a partir de uma *política de acesso* que define regras, como o pagamento de mensalidades, o tipo de inscrição ou a idade do usuário.

O artigo está organizado como segue: a Seção 2 apresenta a ICN e discute as soluções de controle de acesso existentes. A Seção 3 introduz os conceitos da recifragem por *proxy*. A Seção 4 apresenta a solução proposta, enfatizando a adaptação proposta. A Seção 5 avalia o desempenho da solução proposta em três aspectos: rede, esquema de recifragem por *proxy* e *download* de músicas e vídeos. A Seção 6 detalha e avalia a otimização nos algoritmos de recifragem e decifragem, e a Seção 7 apresenta uma comparação da solução proposta com outras duas soluções. A Seção 8 conclui este trabalho.

2. As redes centradas em informação e o desafio do controle de acesso

As redes centradas em informação trazem diversos novos benefícios como consequência da nomeação de conteúdos diretamente na camada de rede, como a redução de tráfego nos canais de comunicação e a consequente diminuição no atraso de entrega de conteúdo. A Figura 1 ilustra um exemplo simples do funcionamento do paradigma de ICN. Neste exemplo, a rede é composta pelo provedor do conteúdo, quatro roteadores e três usuários. As requisições são enviadas pelo usuário em direção à rede, que as roteiam com base no nome do conteúdo solicitado. Na Figura 1(a) o usuário $u1$ envia uma requisição para o conteúdo $/p1/a$ ao seu roteador de borda. O roteador, por sua vez, roteia a requisição em direção ao provedor. A requisição segue o caminho até o provedor, sendo que a resposta contendo o conteúdo segue o caminho reverso, em direção ao usuário, como mostra a Figura 1(b). Cada roteador no caminho armazena uma cópia em seu *cache*, possibilitando que em requisições futuras para esse mesmo conteúdo a rede possa usufruir de redução de tráfego nos canais e diminuição da latência na rede. Por exemplo, na Figura 1(c) o usuário $u3$ solicita o conteúdo $/p1/a$ para $R3$, que roteia a requisição em direção ao provedor. Contudo, o roteador $R1$ possui o conteúdo em seu *cache* e, portanto, responde à requisição sem necessitar consultar o provedor.

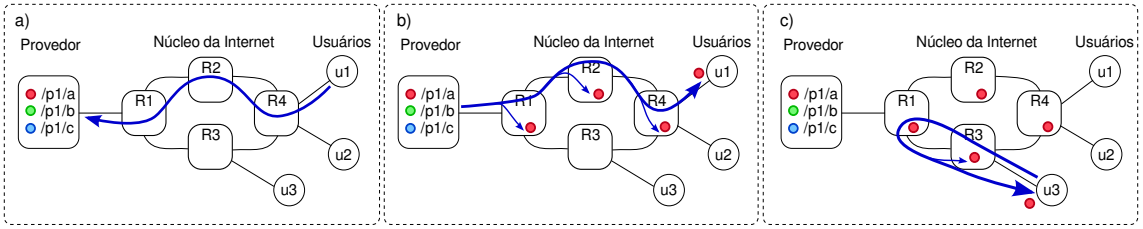


Figura 1. Modelo de distribuição de conteúdo em ICN.

A infraestrutura de armazenamento em *cache* na rede representa um grande desafio para a proteção de conteúdos contra o acesso não autorizado, pois já não é obrigatória a conexão com um servidor específico para a recuperação de um conteúdo. A cifragem do conteúdo é a solução pioneira em ICN, em que somente usuários que possuam uma chave válida possam acessá-lo. Contudo, cifrar um mesmo conteúdo para usuários diferentes não é conveniente, pois o conteúdo cifrado para um usuário não pode ser acessado por outro e, portanto, as cópias armazenadas em *cache* não são aproveitadas. Assim, soluções mais adequadas foram propostas, explorando esquemas criptográficos alternativos e infraestruturas terceirizadas, visando o máximo aproveitamento do *cache*.

Em geral, as soluções criptográficas utilizam uma chave secreta para cifrar os conteúdos e a distribuem cifrada com um esquema de criptografia assimétrica, tais como a criptografia de *broadcast* [Misra et al. 2013] e a criptografia baseada em atributos [Papanis et al. 2013]. Porém, o uso de chaves secretas, apesar de eficiente, tem a desvantagem de ser facilmente repassada para usuários não autorizados, que podem acessar ao conteúdo a partir dos *caches*. Já as soluções infraestruturadas assumem a existência de servidores de autenticação, ou que os roteadores ou *caches* verifiquem políticas de acesso antes de encaminhar um conteúdo. Além de representar passos extras na recuperação do conteúdo, essas soluções geralmente exigem mudanças na arquitetura de ICN, o que pode influenciar negativamente a distribuição de conteúdo de forma eficiente, direta e simples que a ICN propõe. Assim, é necessária uma solução alinhada às características da ICN.

3. Recifragem por *proxy* (*Proxy Re-Encryption* - PRE)

A recifragem por *proxy* [Chow et al. 2010] é um esquema criptográfico utilizado como uma alternativa mais segura e eficiente em cenários em que os usuários desejam delegar direitos de acesso às suas mensagens cifradas, sem a necessidade de entregar a sua própria chave privada. A delegação é realizada através de um *proxy*, mediante a entrega de uma chave de recifragem, rk, que funciona somente para o usuário que recebe a delegação. A Figura 2 ilustra o funcionamento básico da recifragem por *proxy*. O usuário $u1$ cifra uma mensagem m com a sua chave pública, que só pode ser decifrada com a chave privada de $u1$. Para delegar acesso à m' ao usuário $u2$ através da recifragem, o usuário $u1$ envia m' ao *proxy* e uma chave de recifragem $rk_{u1 \rightarrow u2}$, calculada com base na chave pública do usuário $u2$. O *proxy* utiliza $rk_{u1 \rightarrow u2}$ para recifrar o conteúdo, gerando m''. Na recifragem, o *proxy* não tem acesso à mensagem m, nem à chave privada de $u1$. O *proxy* envia m'' para o usuário $u2$, que utiliza a sua chave privada para recuperar a mensagem original m.

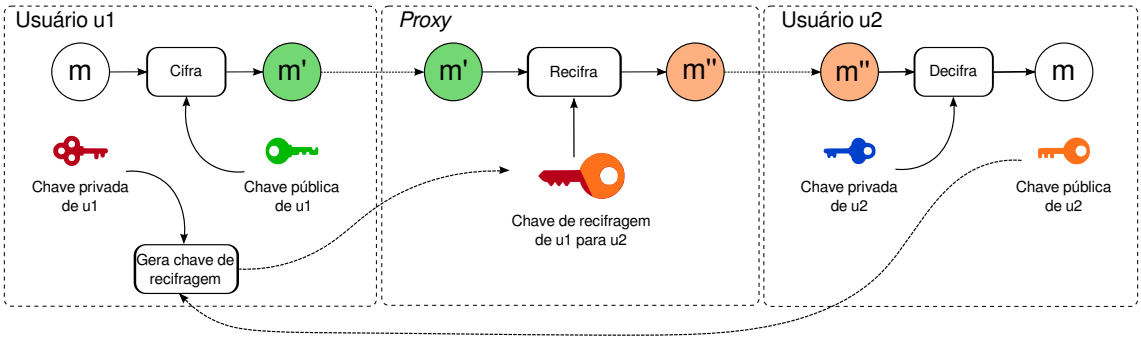


Figura 2. Modelo básico de um esquema tradicional de recifragem por *proxy*.

Os esquemas de PRE têm sido explorados em diversas aplicações, sendo o controle de acesso uma aplicação natural. Devido às suas características, o PRE se mostra uma alternativa promissora aos esquemas de criptografia assimétrica também no controle do acesso aos conteúdos em *cache* em uma ICN, pois permite que vários usuários acessem uma mesma mensagem cifrada através da delegação de chaves de recifragem, permitindo o *cache* na rede sem prejudicar o controle de acesso. Em geral, os esquemas de recifragem por *proxy* devem garantir três asserções básicas: (1) o *proxy* não pode ser capaz de acessar o conteúdo da mensagem que recifra; (2) o usuário $u2$ não pode obter o conteúdo da mensagem sem a intervenção da função de recifragem; e (3) de posse da mensagem cifrada e da chave de recifragem, o *proxy* não pode obter as chaves privadas de $u1$ e $u2$.

Para utilizar um esquema de recifragem por *proxy* como solução de controle de acesso aos conteúdos em ICN, é ideal que ele sustente cinco propriedades principais: *unidirecionalidade*, garantindo que a chave de recifragem $u1 \rightarrow u2$ não pode ser utilizada para delegação de $u2 \rightarrow u1$; *único salto*, garantindo que um conteúdo recifrado não possa ser cifrado novamente; *intransitividade*, garantindo que um *proxy* não seja capaz de criar novas chaves de recifragem para outros usuários; *interatividade*, exigindo que o usuário $u2$ se comunique com o $u1$ antes de gerar a chave de recifragem; e que seja *robusto contra conluio*, para que o *proxy* ou o usuário, de posse do conteúdo recifrado e da chave de recifragem, não consiga descobrir a chave privada de $u1$. Este trabalho utiliza o esquema *Efficient Unidirectional Proxy Re-Encryption* (EU-PRE) [Chow et al. 2010] pois possui as propriedades desejadas para a aplicação em uma solução de controle de acesso.

4. Solução proposta

Diferente das soluções existentes para ICN, a solução proposta visa ser completamente alinhada aos requisitos do paradigma. Desta forma, o EU-PRE é adaptado para que o *proxy* não participe do processo de recifragem, sendo eliminado do modelo. A solução proposta é estruturada em três domínios: *domínio do provedor de conteúdo*, *domínio da rede* e *domínio do usuário* [Mannes et al. 2014]. O domínio do provedor de conteúdo engloba a cifragem do conteúdo e a geração de chaves de recifragem para os usuários, sob demanda. O domínio da rede refere-se ao roteamento e ao encaminhamento de conteúdo seguindo o paradigma de ICN. O domínio do usuário é composto pela aplicação do provedor de conteúdo e pelas operações de recifragem/decifragem. A Figura 3 ilustra o funcionamento geral da solução proposta, considerando esses três domínios.

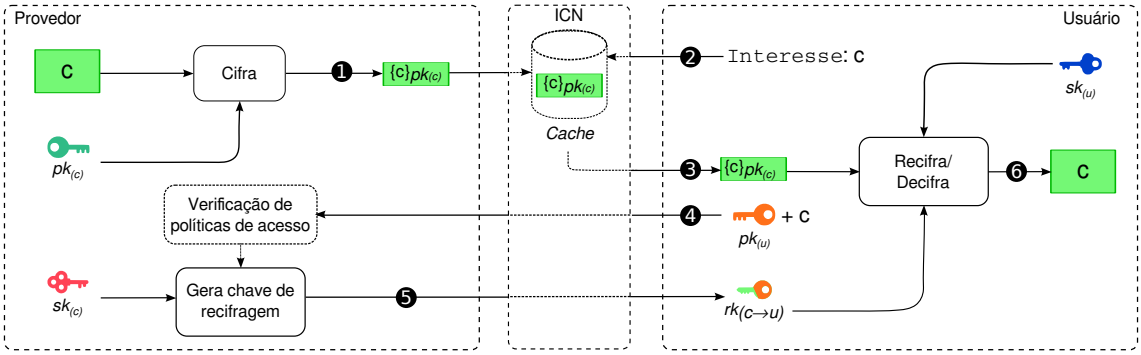


Figura 3. Visão geral da solução de controle de acesso proposta.

A primeira medida do provedor na preparação dos conteúdos para a distribuição aos usuários é a criação de um par de chave pública-privada para cada conteúdo. Desta forma, o provedor pode controlar individualmente o acesso a cada um dos conteúdos, pois é necessário criar chaves de recifragem diferentes para cada conteúdo e para cada usuário que deseja acessá-lo. Assim, os conteúdos são cifrados pelo provedor, neste exemplo representados pelo conteúdo c, com a chave pública exclusiva do conteúdo ($pk_{(c)}$) (seta 1). A chave privada correspondente ($sk_{(c_i)}$) é guardada em segredo pelo provedor de conteúdo. O conteúdo é distribuído conforme as requisições dos usuários, sendo armazenado em *cache* na rede, de acordo com as políticas de *cache*. Neste estágio, o conteúdo pode estar em qualquer lugar na rede; porém, como a chave privada do conteúdo é conhecida somente pelo provedor, nenhum usuário, legítimo ou malicioso, é capaz de decifrá-lo.

O usuário que deseja acessar um conteúdo deve requisitá-lo para a rede (seta 2), que roteia as requisições conforme a arquitetura de ICN. A rede pode satisfazer a requisição com um conteúdo recuperado diretamente do provedor do conteúdo, assim como de um *cache* na rede (seta 3). Antes que a aplicação no usuário possa consumir o conteúdo, ele deve ser decifrado. Para isso, o usuário solicita uma chave de recifragem $rk_{c_i \to u}$ para o provedor (seta 4). Nessa etapa, o provedor tem a oportunidade de aplicar as políticas de acesso, de acordo com o usuário e o conteúdo ao qual ele deseja acessar. Se autorizado, o provedor identifica a chave privada do conteúdo, calcula a chave de recifragem e a envia para o usuário (seta 5). Isso significa que o usuário u é o único habilitado a utilizar $rk_{c_i \to u}$ para decifrar o conteúdo c_i (seta 6), já que esse procedimento requer a chave privada de u. É inútil para uma entidade maliciosa requisitar o conteúdo e interceptar uma chave de recifragem, pois a cifra resultante só poderá ser decifrada pela chave privada de u.

5. Avaliação da solução

A avaliação de desempenho da solução proposta é dividida em três partes: desempenho de uma arquitetura de ICN na distribuição de conteúdos populares, desempenho dos algoritmos do EU-PRE e o desempenho da solução proposta no recebimento de músicas e vídeos [Mannes et al. 2015]. Para aferir o desempenho na distribuição de conteúdos, foi utilizado o simulador ndnSIM com *cache* de 100 *chunks* e 8 e 23 servidores, que mantêm um catálogo de 100 conteúdos de 1GB cada, divididos em *chunks* de 4KB. As requisições seguem uma distribuição Zipf com 1.000, 2.000 e 3.000 requisições. Os resultados apresentados são a média de 35 simulações com intervalo de confiança de 95%.

Uma das principais vantagens na adoção das ICNs é a presença de *cache* na rede, que beneficia diretamente a entrega de conteúdos para os usuários. A Figura 4(a-b) apresenta os resultados obtidos, comparando-os ao cenário sem *cache*. Como esperado, a presença de *cache* na rede representa um ganho substancial na rapidez da entrega de conteúdo. Além disso, a presença de mais servidores na rede implica em menos tempo para a recuperação de um *chunk*. Já o desempenho do *cache* (c) está relacionado à concentração de requisições em menos conteúdos (0 - 5), confirmando a hipótese de que as ICNs têm um impacto mais significativo em conteúdos populares. Assim, fica claro o potencial de uma ICN bem ajustada na melhora da entrega de conteúdo multimídia.

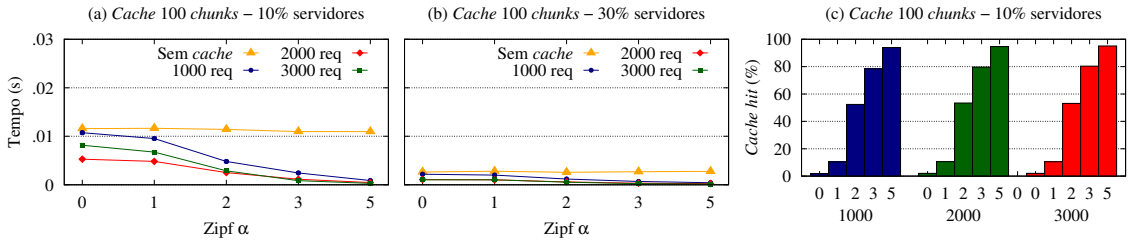


Figura 4. Tempo para recebimento de um *chunk* e quantidade de *cache hits*.

Para a avaliação de desempenho do EU-PRE, foram implementados os quatro algoritmos do esquema em Python: *cifragem*, *geração de chaves de recifragem*, *recifragem* e *decifragem*. A avaliação é realizada com diferentes tamanhos de mensagens e de chaves: 1.024, 2.048 e 3.072 *bits* e em dois ambientes distintos: um servidor AMD Opteron 6136 2,4GHz e 86GB de RAM para a cifragem e geração de chaves de recifragem e um *notebook* Intel Core i5 1,60GHz, com 8GB de RAM para a recifragem e decifragem. Os resultados, apresentados (Figura 5) mostram que o esquema é adequado, cifrando 4KB em 18ms com uma chave de 2.048 *bits*, levando menos de 40ms para gerar uma chave de recifragem e aproximadamente 91ms para recifrar/decifrar 4KB. Um bom desempenho nas operações de recifragem e decifragem é essencial para o dispositivo do usuário recifrar e decifrar os *chunks* a uma velocidade que permita o uso do conteúdo sem pausas.

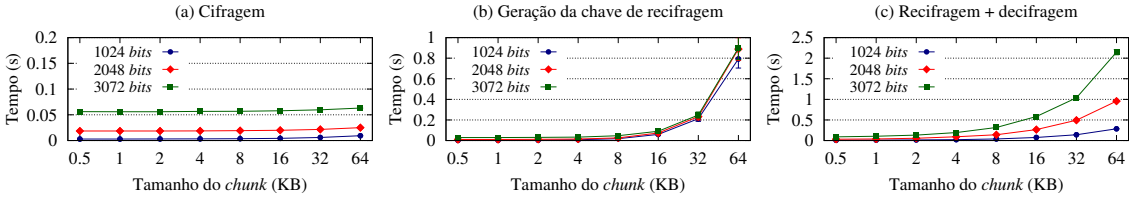


Figura 5. Tempo de execução das operações do esquema EU-PRE

Para avaliar o desempenho da solução proposta, são analisados os tempos totais para cifragem e decifragem de conteúdos inteiros, considerando fluxos de música e de vídeos com diferentes quantidades de operações paralelas (núcleo do processador). Na cifragem, são considerados arquivos de dois minutos de música e duas horas de vídeo, com diferentes qualidades. Para a decifragem, é considerada a quantidade de dados necessários para usufruir de um minuto de música e uma hora de vídeo. A cifragem, Figuras 6(a-b), mostra que o provedor leva menos de 2 segundos para cifrar as músicas e até 30 minutos para cifrar vídeos. Já a decifragem, Figuras 6(c-d), mostra que o tempo decorrido desde a requisição do primeiro *chunk* do conteúdo até a recifragem/decifragem do último *chunk* é menor que um minuto para músicas, porém para arquivos de vídeo, ultrapassa o tempo esperado de execução, revelando que há margens para otimizações.

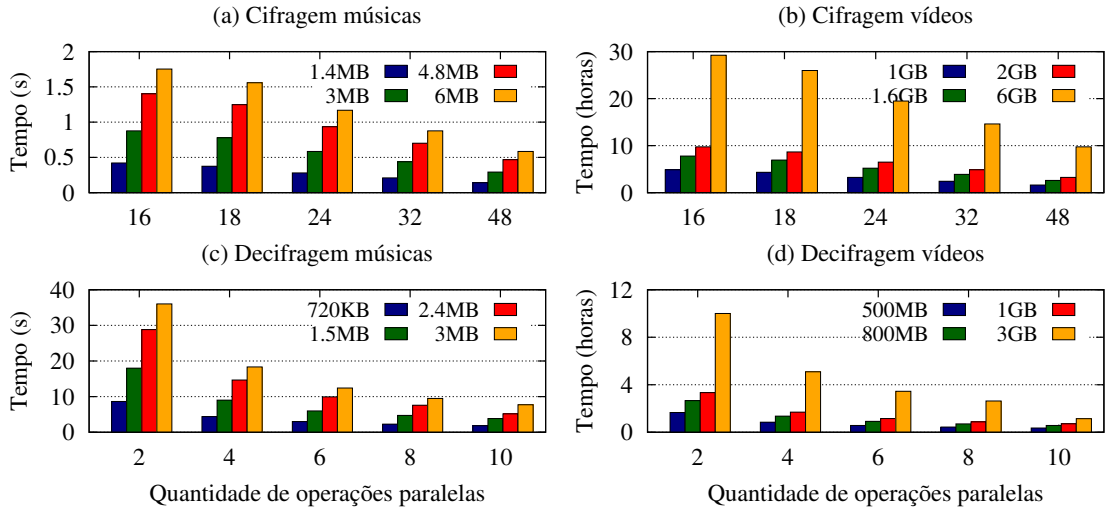


Figura 6. Tempo estimado para cifragem e decifragem de músicas e de vídeos.

6. Otimização da solução

Com a alocação conjunta das funções do *proxy* no usuário, é possível realizar uma sobreposição parcial das operações de recifragem e decifragem, resultando em um menor volume de cálculos e na execução mais rápida dessas funções. A junção dessas operações é justificável, pois são operações dependentes e sempre são executadas em conjunto pelo usuário. As propriedades do esquema EU-PRE garantem que essa otimização não introduz vulnerabilidades no esquema (pois ele é imune à possibilidade de conluio entre o usuário e *proxy*. A validação mostra que a otimização proposta resulta em um ganho expressivo nos tempos de recifragem e decifragem, obtendo uma redução de até 96% (Figura 7(a)). Assim, o esquema otimizado pode também ser utilizado para o recebimento de vídeos de alta definição a partir de seis operações paralelas (Figura 7(b-c)).

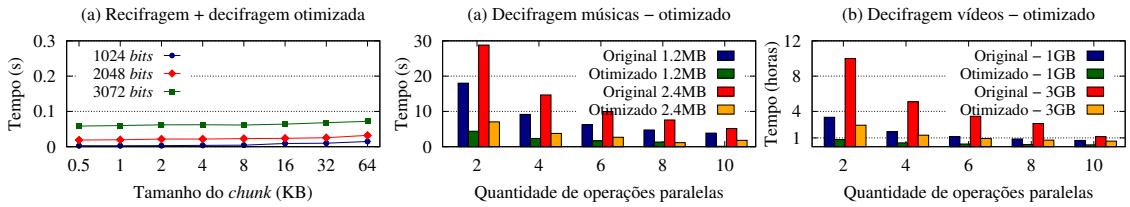


Figura 7. Tempo otimizado para recebimento e decifragem de arquivos.

7. Análise comparativa com outras soluções criptográficas

Foram escolhidas a criptografia de *broadcast* (BE) e a criptografia baseada em atributos (ABE) para a comparação com a solução proposta [Mannes et al. 2016]. A avaliação do BE considera um grupo de 8.192 usuários, o ABE com 30 atributos e EU-PRE com chaves de 2.048 *bits*. O BE é o mais eficiente, sendo que 66% do tempo total é para o recebimento do conteúdo e o restante para as operações criptográficas (Figura 8(a)). Já o ABE supera o tempo necessário para um fluxo de conteúdo sem interrupções (Figura 8(b)). A solução proposta, com sua otimização, tem uma menor sobrecarga nos usuários, comparado aos modelos de ABE, e evita o uso das chaves secretas para o compartilhamento de conteúdos devido à fraca segurança agregada a esse modelo, como o BE (Figura 8(c)).

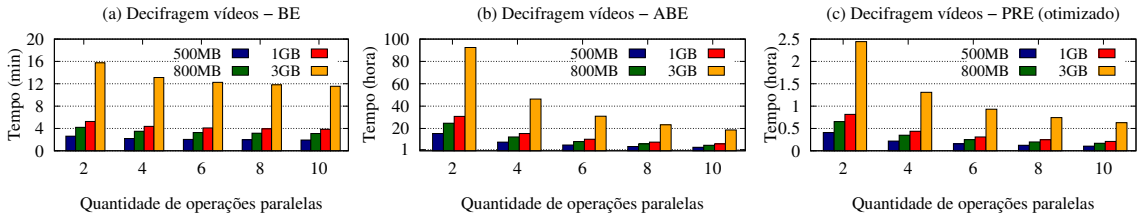


Figura 8. Comparação de *download* e decifragem das soluções BE, ABE e PRE.

8. Conclusão

A solução de controle de acesso proposta para ICN, não interfere no funcionamento da rede, usufruindo de todo o ganho proporcionado pelos *caches*. Comparada a outras duas soluções de controle de acesso criptográficas, a solução proposta tem melhor desempenho, é mais segura e faz o melhor uso dos *caches*. As oportunidades de trabalhos futuros incluem estudar o impacto da solução diante da distribuição de um mesmo conteúdo com codificações diferentes e identificar outras oportunidades de otimização no EU-PRE.

Referências

Ahlgren, B., Dannewitz, C., Imbrenda, C., Kutscher, D., and Ohlman, B. (2012). A Survey of Information-Centric Networking. *IEEE Communications Magazine*, 50(7):26–36.

Chow, S. S. M., Weng, J., Yang, Y., and Deng, R. H. (2010). Efficient Unidirectional Proxy Re-encryption. In *3rd International Conference on Cryptology in Africa*, pages 316–332.

Mannes, E., Maziero, C., Lassance, L. C., and Borges, F. (2014). Controle de Acesso Baseado em Reencriptação por Proxy em Redes Centradas em Informação. In *XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 2–15.

Mannes, E., Maziero, C., Lassance, L. C., and Borges, F. (2015). Optimized Access Control Enforcement over Encrypted Content in Information-centric Networks. In *Symposium on Computers and Communications (ISCC'15)*, pages 924–929.

Mannes, E., Maziero, C., Lassance, L. C., and Borges, F. (2016). Assessing the Impact of Cryptographic Access Control Solutions on Multimedia Delivery in Information-centric Networks. In *15th IEEE/IFIP Network Operations and Management Symposium (NOMS'16)*, pages 1–6.

Misra, S., Tourani, R., and Majd, N. E. (2013). Secure Content Delivery in Information-centric Networks: Design, Implementation, and Analyses. In *ACM SIGCOMM Workshop on ICN*, pages 73–78.

Papanis, J. P., Papapanagiotou, S. I., Mousas, A. S., Lioudakis, G. V., Kaklamani, D. I., and Venieris, I. S. (2013). On the Use of Attribute-Based Encryption for Multimedia Content Protection over Information-Centric Networks. *Transactions on Emerging Telecommunications Technologies*, 25(4):422–435.

Securing Networked Embedded Systems through Distributed Systems Analysis

Fernando A. Teixeira[1], **José Marcos S. Nogueira, Leonardo B. Oliveira**

[1] UFSJ – Ouro Branco, MG, Brazil. UFMG – Belo Horizonte, MG, Brazil

teixeira@ufsj.edu.br,{jmarcos,leob}@dcc.ufmg.br

Abstract. *The growing importance of Internet of Things (IoT) calls for tools able to provide users with secure systems. Traditional approaches to analyze distributed systems are not expressive enough to address this challenge. As a solution, we present a framework to analyze networked embedded systems. Our key insight is to look at a distributed system as a single body, and not as separate programs that exchange messages. We then can crosscheck information inferred from different nodes. To construct this global view of a distributed system, we introduce a novel algorithm that discovers inter-program links efficiently. Such links lets us build an inter-program view, a knowledge that we can thus forward to a traditional static analysis tool. We prove that our algorithm always terminates and that it correctly models the semantics of a distributed system. We have implemented our solution on top of the LLVM compiler, and have used it to secure six ContikiOS applications against buffer overflow attacks. Our solution produces code that is as safe as code secured by more traditional analyses; however, our binaries are on average 18% more energy-efficient.*

1. Introduction

The emergence of the Internet of Things (IoT) has increased importance of Networked Embedded Systems (NES) [Borgia 2014]. In fact, more than ever, the everyday person and "things" are surrounded by NES in a most varied set of devices, which perform a very diverse list of services. However, programming these systems is more challenging, due, not only to their shear volume, but also to this diversity. Above all, software embedded in appliances, cars and sensors scattered throughout cities, running in hardware of different capacities, and subject to very different natural conditions.

IoT faces a plethora of security problems [Heer et al. 2011]. It suffers from the same security issues as traditional Internet-based and/or wireless systems. In addition, it is more prone to other issues such as out-of-bound memory accesses due to a few factors: IoT costs must be kept as low as possible, and to meet this requirement, they are usually endowed with the least amount of resources necessary to accomplish their duties. Accordingly, applications for IoT are commonly developed using lightweight languages such as C, an inherently unsafe language [Chess and West 2007]. Its semantics allows out-of-bound memory accesses. This type of accesses are dangerous because they give room to Buffer Overflow (BOF) attacks. A BOF takes place whenever a system allows data to be accessed out of the bounds of an array.

Much work has already been done to turn C into a safer language (e.g. SAFECode [Dhurjati et al. 1996] and AddressSanitizer (ASan) [Serebryany et al. 2012]). Existing proposals resort to Array-Bound Checks (ABC), which are tests done at runtime to

ensure that a particular array access is safe. These proposals work in a two-pass fashion. They first scan programs' assembly representation to find code snippets containing vulnerabilities; in a second step, they return to the potential vulnerabilities and insert ABCs. While effective in preventing out-out-bound memory accesses from taking place, these proposals impose a significant overhead on compiled programs, and are thus inadequate *as-is* to IoT. As an example, ASan is known to slowdown programs by over 70%, and to increase their memory consumption by over 200%. It is therefore paramount to develop more efficient techniques that can be used to protect IoT.

The goal of this work is to describe a general framework for analysis and optimization of distributed systems, which we can use to implement an efficient solution to counter buffer-overflow attacks in IoT. Our **key insight** is to look at a distributed system as a single entity, rather than as multiple separate message-exchanging programs. Using a novel algorithm, we can infer the communication links between different programs that converse through a network. This knowledge lets us model how data flows across distributed programs; hence, it gives us a holistic view of the entire system. Such a view can be coupled with traditional static analysis tools to improve their precision.

To validate our claims, we have used our framework to protect IoT systems against BOF attacks. We call our solution Securing Internet of Things (SIoT). More specifically, we applied Tainted Flow Analysis (TFA) [Balzarotti et al. 2008] on the model we proposed, and sanitized C programs against out-of-bound memory accesses. TFA tracks potentially malicious data (i.e., data that can be influenced by attackers) flows across the program. Memory indexed by tainted data can then be guarded against invalid access during runtime using ABC. Because the analysis has a holistic view of the entire system, it produces a smaller number of false-positives than if each module of the system were analyzed individually. This extra precision yields a smaller runtime overhead. Notice that this framework is general enough to support a wider range of analysis. The solution for out-of-bound memory accesses presented in this work is just an instance of it.

Our contribution. This work brings forth both theoretical and practical contributions.

On the theoretical side, we propose a way to model distributed systems as single entities. More specifically: **1.** We propose an extension to the standard Control Flow Graph (CFG) [Allen 1970], called Distributed Control Flow Graph (DCFG), expressive enough to model the control flow spanning multiple programs that communicate over a network.; **2.** We propose an algorithm that infers communication links between different programs from a distributed system, and prove that the algorithm (i) never misses possible communication paths between programs; and (ii) always reaches a fixed point, and hence always terminates.

On the practical side, we have implemented our algorithm, and showed that it can protect IoT against BOF, and can do so more efficiently than traditional approaches. More specifically: **1.** We have implemented our algorithm and its companion distributed TFA in the LLVM compiler [Lattner and Adve 2004]. **2.** We have applied this analysis on six applications present in ContikiOS [Dunkels et al. 2004], and the results show that our proposal is 18% more energy-efficient than existing solutions.

Publications. The problem and first idea of solution was published as part of an SBSeg13 tutorial. Then, a first version of the SIoT with preliminary results was published

in SBRC14. An extension of this work was published at Latin American Transactions. In 2015, a complete version of SIoT was published at the IPSN [Teixeira et al. 2015][1]. The list of these publications can be seen below:

- Segurança de Software em Sistemas Embarcados: Ataques & Defesas. *SBSeg'2013*.
- Defending Code from the Internet of Things against Buffer Overflow. *SBRC'2014*.
- Defending Internet of Things against Exploits. *IEEE Latin America Transactions, 2015*.
- SIoT: Securing the Internet of Things through Distributed System Analysis. *IPSN'2015*.

From our work was derived two undergraduate projects and one master degree dissertation. The undergraduate projects have finished in 2014. The master dissertation uses our framework to create a Distributed Range Analysis. It was conclude in 2015 and was published at SBSeg14 and at Latin American Transactions 2015.

2. Language-Based Techniques for Addressing BOF Vulnerabilities

Software code can harbor different types of security vulnerabilities, and those susceptible to BOF attacks are the most exploited. Solutions to address this class of vulnerabilities have long existed, and are largely based on static analysis [Chess and West 2007], dynamic analysis [Serebryany et al. 2012], or a combination of both. In static analysis, analysis is performed without actually running the program; instead either the source code or the object code is inspected, and vulnerabilities flagged. Dynamic analysis, on the other hand, is performed during system executions, and takes advantage of information that is available only at runtime. Armed with runtime information, it is then able to accurately flag problems in actual runs of the system. Due to their complementary nature, it is common to use hybrid analysis, i.e., the combination of static and dynamic techniques. Usually, static analysis is used first, to identify potential vulnerabilities; the vulnerable stretches are then instrumented and monitored at runtime by dynamic analysis.

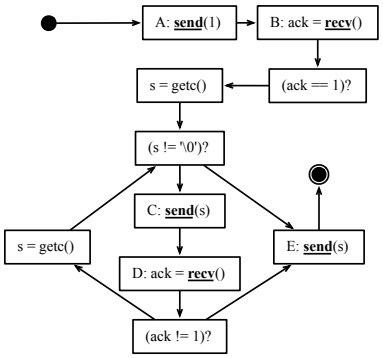
Code Analysis using CFG. The CFG is used to model the control flow of computer programs (Fig. 1). The CFG of a program P is a directed graph defined as follows. For each instruction $i \in P$, we create a vertex v_i; we add an edge from v_i to v_j if it is possible to execute instruction j immediately after instruction i. There are two additional vertices, start and exit, representing the start and the end of control flow.

One class of potential BOF vulnerabilities we might be interested in flagging is variables assignments where the data being assigned are originated externally from user or environment input. If we assume that neither the data sent over the network, nor the executable of the various distributed modules can be tampered with (see Section 3 for a discussion of these assumptions), then we would see the assignments in lines 1 and 5 (Fig. 1b) differently. Even though they both involve data coming from the network (through the `RECV` function), we would deem the one in line 5 as vulnerable, but not the one in line 1. The assignment in line 5 is vulnerable because the data being assigned to `msg` comes from `getc` (line 4, Fig. 1a), which could provide malicious data from attackers (`msg` has been used in buffer access). The first assignment in server program (line 1, Fig. 1b) is not vulnerable because the data being assigned is a hard-coded constant from the client program (line 1, Fig. 1a).

[1]IPSN is a flagship conference on networked embedded systems – http://ipsn.acm.org/

(a)

```
1  send(1);
2  ack = recv()
3  if (ack == 1) {
4      s = getc();
5      while (s != '\0') {
6          send(s)
7          ack = recv();
8          if (ack != 1) {
9              break;
10         } else {
11             s = getc();
12         }
13     }
14     send(s);
15 }
```



(b)

```
1  msg = recv();
2  if (msg == 1) {
3      send(1);
4      do {
5          msg = recv();
6          putc(msg);
7          if (msg != '\0')
8              send(1);
9          else
10             break;
11     } while (1);
12 } else {
13     send(0);
14 }
```

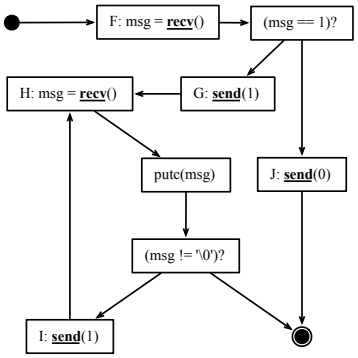


Figure 1. Echo application's programs and their respective CFGs. (a) Echo client. (b) Echo server.

In their standard form, CFGs are unable to model the overall control flow of programs that span multiple distributed processes. Thus, they do not provide support to distinguish the two assignments mentioned above. To be safe, both assignments are usually flagged as vulnerable, yielding one true-positive (line 5, Fig. 1b) and one false-positive (line 1, Fig. 1b). Our proposal addresses this shortcoming (Section 4).

3. System Assumptions & Attack Model

For the purpose of this work, we assume networks of distributed and embedded nodes, organized as IoT systems. Each node can interact with its environment through sensors, actuators, or user interfaces. We assume that both the system running at each node and the communication between different nodes is protected against tampering. Different security mechanisms can be used to implement such protections. For example, Trusted Platform Module (TPM) [Kinney 2006] can be employed to ensure the integrity of nodes systems and cryptographic solutions like [Perrig et al. 2002, Kothmayr et al. 2011] can be used to establish a secure communication channel.

Attackers have control over the input data that the nodes receive from its environment. This includes data captured by the sensors or input from the user interfaces, but excludes data coming from network interfaces (we assume a secure communication channel). Though limited in the type of attacks they can launch, such attackers can potentially cause security problems if the code running on the nodes harbors certain types of vulnerabilities. For example, if the code does not check array bounds, certain inputs may cause BOF. Attackers can then manipulate the environment (to produce spurious sensor readings) or provide spurious user input to launch a BOF attack, leading the nodes to denial of service or malicious behavior. Because these nodes are connected to the Internet,

misbehaving nodes can be used as a proxy to attack other nodes in the network. Note that malicious input injection attacks can be most effectively exploited if the attacker has information of vulnerabilities in the code. This information is readily available in case of open source programs, and can also be obtained from code reverse engineering, and program fuzzing exercises.

4. Communication Links Inference

CFGs model the control flow of individual programs. To analyze an entire distributed system, we need to work with CFGs that transcend program boundaries. We propose the notion of DCFG for this purpose, and describe how they are built below. Each DCFG models the communication between two programs in a system. If the system contains more than two programs, a DCFG is necessary to model the relationship between each pair of them. Let $\{\mathcal{C}_1, \mathcal{C}_2\}$ be a pair of CFGs that constitute a system and D the resulting DCFG. D contains C_1 and C_2 as a subgraph. Inter-program edges connecting C_1 and C_2 are then added to D: for each pair of SEND and RECV vertices (each from the two different CFGs) that may communicate, we add an edge from the former to the latter. That is, for each pair of vertices $s_i \in \mathcal{C}_1$ and $r_j \in \mathcal{C}_2$, if there is an execution sequence in which a message issued by s_i reaches r_j, we add to D an inter-program edge from s_i to r_j. And, for each pair of vertices $s_k \in \mathcal{C}_2$ and $r_t \in \mathcal{C}_1$, if there is an execution sequence in which a message issued by s_k reaches r_t, we add to D an inter-program edge from s_k to r_t.

In principle, we can add inter-program edges linking every send vertex in one of the CFGs to every receive vertex in the other CFG in the system. However, the resulting DCFG would have inter-program edges linking sends and receives that could not be the matching ends of a communication. For instance, in Fig. 1, sends from vertex A are not received by vertex H; every send from A will be received by F before H has a chance to execute. To define a DCFG that better model the workings of a system, we introduce the notions of Send-Graph, Receive-Graph, and levels.

Given a CFG $\mathcal{C}$ of a program, we define its associated Send-Graph S and Receive-Graph R as follows. For each vertex $v \in \mathcal{C}$ labeled with a send operation, we add a vertex v' to S. We also add $start'$ and $exit'$ vertices, which correspond to start and exit in $\mathcal{C}$. Edges in S correspond to paths between sends in the original $\mathcal{C}$. For every pair of vertices $u, v \in \mathcal{C}$, we add an edge $\overrightarrow{u'v'}$ to S if, and only if: (i) there exists a path p from u to v in $\mathcal{C}$, and (ii) p does not contain any other sends. We create R in a similar way, replacing sends by recvs in the procedure described above.

Next, we move on to the concept of *level*. Given a Send-Graph, its level 0 contains the start vertex. Level 1 contains the sends that are reachable, in one step, from the root. More generally, level $n + 1$ contains the immediate successors of vertices in level n. The procedure is complete when the vertices in the just-generated level do not have successors, or the just-generated level is a duplicate of a previously existing one. The concept of level can be similarly defined for Receive-Graphs. We show an example in Fig. 2. Consider echo client program Send-Graph (Fig. 2 left-hand-side). Its level 1 contains the immediate successors of root node, i.e., $\{A\}$. Its level 2 contains the immediate successors of each send node of level 1, i.e., $\{C, E\}$. The successors of C is $\{C, E\}$, and E does not have successors. We find ourselves in a cycle, and the traversal can now stop. The levels for echo server Receive-Graph can be similarly determined.

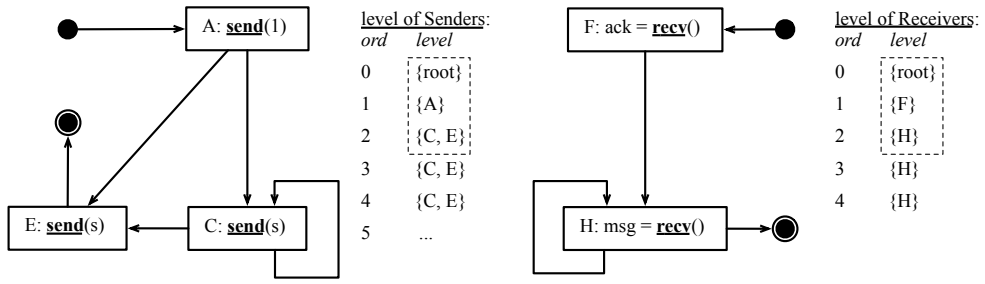


Figure 2. Levels for echo client's SENDs (left-hand-side) and echo server's RECVs (right-hand-side).

Algorithm 1: Elevator

Input: CFGs $\{\mathcal{C}_1, \mathcal{C}_2\}$, Send-Graphs $\{\mathcal{S}_1, \mathcal{S}_2\}$ and Receive-Graphs $\{\mathcal{R}_1, \mathcal{R}_2\}$.
Output: a DCFG D

▷ Set the SEND and RECV levels
foreach $G_i \in \{\mathcal{S}_1, \mathcal{S}_2\} \cup \{\mathcal{R}_1, \mathcal{R}_2\}$ **do**
  $n \leftarrow 0$
  $L_{G_i,n} \leftarrow \{root\}$
  ▷ While the new generated set $L_{G_i,n}$ is unique
  while $L_{G_i,n} \neq L_{G_i,0..n-1}$ **do**
    foreach *vertex* v *in* $L_{G_i,n}$ **do**
      $S_{succs} \leftarrow$ successors of v
      $L_{G_i,n+1} \leftarrow L_{G_i,n+1} \cup S_{succs}$
    $n \leftarrow n + 1$

▷ Link SENDs and RECVs of the same level
$D \leftarrow \mathcal{C}_1 \cup \mathcal{C}_2$
for $k \leftarrow 1$ **to** n **do**
  foreach $v_s \in L_{\mathcal{S}_1,k}$ **and** $v_r \in L_{\mathcal{R}_2,k}$ **do**
    add an edge from v_s to v_r in D
  foreach $v_s \in L_{\mathcal{S}_2,k}$ **and** $v_r \in L_{\mathcal{R}_1,k}$ **do**
    add an edge from v_s to v_r in D

Given the CFGs in Fig. 1, their resulting DCFG can be built by linking the SEND vertices in one CFG with the RECV vertices of the same level in the other. Links are established between SENDs and RECVs that have the same level because they model matching ends of message exchanges. The steps describe above are captured in the *Elevator Algorithm* (Algorithm 1). (See Section 3.4, in the Thesis, for formalization of Elevator algorithm, with a prove of its correctness and termination.)

5. Evaluation

The holistic program view that SIoT gives to a static analysis lets it consider network channels as links between modules instead of input operations. Therefore, all the ABCs that depend exclusively on the network and do not reach user inputs can be eliminated. The end result of this extra precision is more efficient executable code. To validate this claim, we have used SIoT to improve the code that ASan generates. To give the reader some perspective on our results, we compare SIoT to a hypothetical traditional tainted flow analysis, i.e., a technique that treats distributed system separate programs, and not as a single entity. Henceforth, we refer to this technique as Baseline.

We perform the experiments into six pairs of ContikiOS applications (Table 1). For each of these pairs, we compare the number of ABCs that ASan inserts without

Table 1. ABCs inserted by ASan, Baseline, and SIoT.

Applications	Arrays	Memory Accesses	ABCs inserted			% ABCs Reduction	
			ASan	Baseline	SIoT	SIoT vs ASan	SIoT vs Baseline
netdb client/server	6,181	22,819	4,641	172	16	99.66%	90.70%
ping6 / new-ipv6	4,683	16,871	7,453	166	14	99.81%	91.57%
ipv6-rpl-collect udp-sender/sink	4,786	17,301	3,831	168	14	99.63%	91.67%
ipv6-rpl-udp client/server	4,760	17,162	3,787	170	14	99.63%	91.76%
udp-ipv6 client/server	4,701	16,945	3,736	212	14	99.63%	93.40%
coap-client / rest-server	5,195	18,693	4,032	214	14	99.65%	93.46%

Table 2. Energy consumption for the unprotected (Plain) and protected (SIoT and Baseline) versions of applications. CI: Confidence Interval.

Application	Plain		SIoT		Baseline	
	Energy (J)	CI	Energy (J)	CI	Energy (J)	CI
netdb client	1.941	0.025	2.452	0.014	2.486	0.011
ping6	0.109	0.001	0.111	0.001	0.151	0.002
ipv6-rpl-collect udp-sender	2.277	0.043	2.286	0.043	2.996	0.029
ipv6-rpl-udp udp-client	3.062	0.029	3.076	0.016	3.127	0.005
udp-ipv6 client	3.842	0.019	3.860	0.011	3.958	0.029
coap-client / rest-server	4.856	0.020	4.861	0.037	5.034	0.041

any optimization against the number of ABCs that the Baseline and the SIoT-based approaches insert. This table shows that ASan introduces between 3,736 ABCs (*udp-ipv6 client/server*) and 7,453 ABCs (*ping6 / new-ipv6*). The Baseline approach reduces ASan's numbers substantially, between 170 (*ipv6-rpl-udp client/server*) and 214 (*rest-server/coap-client*), because it is eliminating every guard that is not influenced by data coming from an external function. SIoT can further reduce this number one order of magnitude more. In this case, contrary to what is done by the Baseline approach, network functions are no longer marked as dangerous, unless they read data that comes from genuine inputs. We conclude from these experiments that the automatic inference of links between distributed programs improves the precision of static analyses tools in non-trivial ways.

On Energy Saving. Each ABC that we eliminate represents a small saving in terms of energy consumption. To back up this observation with actual data, we performed an experiment with six ContikiOS applications. We tested three versions of each application: (i) without ABCs; (ii) with the ABCs inserted by the Baseline; and (iii) with the ABCs inserted by SIoT. To carry on this experiment, we have installed the applications in *IRIS XM2110* sensors and have measured the amount of energy that they consume. Our results (Table 2) show that SIoT outperforms Baseline for all applications. On average SIoT is 18% more energy efficient than Baseline.

6. Conclusion

This work has presented a framework for analysis and optimization of distributed system, which we have used to protect IoT systems against BOF attacks. Our framework provides typical static analyses with a holistic view of a distributed system. This view improves the precision of such analyses. To validate this claim, we have created SIoT to instantiate a version of TFA that points out which memory accesses need to be guarded against BOF

attacks. Our experiments have demonstrated that our approach is effective and useful to make programs running over a network safer. As future work, we plan to use our framework to enable other kinds of program analyses. In particular, we are interested in using it to secure programs against errors caused by integer overflows. We also want to use our framework to enable compiler optimizations. As an example, if we go back to Figure 1, we see that the conditional test at line 2 of our server is unnecessary. Such an observation requires SIoT's global view of a distributed system.

Thesis Links. The Thesis text is publicly available at http://homepages.dcc.ufmg.br/~teixeira/teixeira-thesis.pdf. The SIoT code is publicly available at http://cuda.dcc.ufmg.br/siot/.

Acknowledgment. We thank Fernando Pereira and Hao Chi Wong for their valuable contributions. This work was partially supported by Intel Corporation, CNPq and FAPEMIG.

References

Allen, F. E. (1970). Control flow analysis. *ACM Sigplan Notices*, 5:1–19.

Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C., and Vigna, G. (2008). Saner: Composing static and dynamic analysis to validate sanitization in web applications. In *Symposium on Security and Privacy (S&P)*. IEEE.

Borgia, E. (2014). The internet of things vision: Key features, applications and open issues. *Computer Communications*, 54:1–31.

Chess, B. and West, J. (2007). *Secure Programming with Static Analysis*. Addison-Wesley Professional, first edition.

Dhurjati, D., Kowshik, S., and Adve, V. (1996). SAFECode: enforcing alias analysis for weakly typed languages. In *Conference on PLDI*. ACM.

Dunkels, A., Gronvall, B., and Voigt, T. (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN*. IEEE.

Heer, T., Garcia-Morchon, O., Hummen, R., Keoh, S. L., Kumar, S. S., and Wehrle, K. (2011). Security challenges in the IP-based Internet of Things. *Springer Wireless Personal Communications*, 61(3):527–542.

Kinney, S. L. (2006). *Trusted platform module basics: using TPM in embedded systems*. Newnes.

Kothmayr, T., Hu, W., Schmitt, C., Bruenig, M., and Carle, G. (2011). Poster: Securing the internet of things with DTLS. In *SenSys*. ACM.

Lattner, C. and Adve, V. S. (2004). LLVM: A compilation framework for lifelong program analysis & transformation. In *CGO*. IEEE.

Perrig, A., Szewczyk, R., Wen, V., Culler, D., and Tygar, J. D. (2002). SPINS: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534.

Serebryany, K., Bruening, D., Potapenko, A., and Vyukov, D. (2012). AddressSanitizer: a fast address sanity checker. In *Annual Technical Conference (ATA)*. USENIX.

Teixeira, F. A., Machado, G. V., Pereira, F. M., Wong, H. C., Nogueira, J., and Oliveira, L. B. (2015). SIoT: Securing the Internet of Things through Distributed System Analysis. In *Proceedings of 14th IPSN*, pages 310–321. ACM.

Computação sobre dados cifrados em *GPGPUs*

Pedro Geraldo M. R. Alves, Diego F. Aranha

Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Cidade Universitária Zeferino Vaz – CEP: 13083-852 – Campinas – SP – Brazil

{pedro.alves,dfaranha}@ic.unicamp.br

Abstract. *Under the dominant cloud computing paradigm, employing encryption for data storage and transport may not be enough for assurance of secrecy, because the data owner has no real control over the processing hardware. This way, security guarantees should also be extended to data processing. Homomorphic encryption schemes are natural candidates for computation over encrypted data in the cloud, since they are able to satisfy all security requirements imposed by that environment. This work investigates strategies to efficiently implement the leveled fully homomorphic scheme* YASHE *on GPGPUs. As result of this research, the* cuYASHE *library was developed and made available to the community. In particular, it presents a speedup between* 6 *and* 35 *times for homomorphic multiplication. This operation is performance-critical for evaluating any function over encrypted data, demonstrating that GPGPUs are an appropriate technology for bootstrapping privacy-preserving cloud computing environments.*

Resumo. *No contexto da computação na nuvem, a aplicação de métodos criptográficos exclusivamente no armazenamento e transporte dos dados não é suficiente para a preservação de sigilo uma vez que os detentores dos dados não tem controle real sobre o hardware de processamento. Dessa forma, os requisitos de segurança precisam também ser estendidos para o processamento dos dados. Esquemas de cifração homomórfica são candidatos naturais para computação sobre dados cifrados na nuvem, visto que são capazes de satisfazer todos os requerimentos de segurança impostos pelo ambiente. Este trabalho investiga estratégias para implementação eficiente do criptossistema homomórfico em nível* YASHE *em GPGPUs. Como fruto das conclusões obtidas, a biblioteca* cuYASHE *foi desenvolvida e disponibilizada à comunidade. Em particular, ela apresenta ganhos de velocidade entre* 6 *e* 35 *vezes para a operação de multiplicação homomórfica. Por ser uma operação com desempenho critico para a avaliação de funções sobre criptogramas, esse resultado demonstra que GPGPUs são uma tecnologia apropriada para computação que preserve privacidade em ambientes de nuvem.*

1. Introdução

A computação em nuvem tem sido responsável por uma profunda mudança nas soluções de processamento distribuído. A possibilidade de terceirizar a instalação, manutenção e a escalabilidade de servidores, somada a preços competitivos, faz com que esses serviços se tornem altamente atraentes, tendo sua aplicação feita em setores diversos, desde o meio científico até o mercado de dispositivos móveis [Hoffa et al. 2008, Dinh et al. 2013].

A adoção da computação em nuvem, entretanto, levanta importantes questões de segurança. A possibilidade do vazamento de informações acompanha o crescimento do

número de entidades que manipulam os dados, e o sigilo é apontado como uma das maiores preocupações [Xiao and Xiao 2013].

Diversos esquemas criptográficos são utilizados como padrão para o armazenamento e transferência de dados. Contudo, no caso de serviços na nuvem existe a possibilidade de se lidar com um adversário que não apenas tenha acesso aos dados como também ao *hardware* que realiza seu processamento. Desse modo, é necessário que se implemente estratégias de proteção também durante o processamento.

A criptografia homomórfica se mostra promissora para satisfazer esse requisito de segurança. Os esquemas dessa classe permitem que o processamento seja feito sobre criptogramas mesmo sem o conhecimento das chaves de cifração ou decifração. Assim, não há razão para que dados sejam revelados no momento do processamento.

Objetivo. O objetivo deste trabalho consistiu em realizar uma implementação do criptossistema homomórfico YASHE [Bos et al. 2013] com ganho de velocidade nas operações homomórficas em relação ao estado da arte. A abordagem definida foi aplicar técnicas de paralelismo por meio de *GPGPUs* sobre a plataforma CUDA.

Contribuições. Este trabalho apresenta a primeira implementação em *GPGPUs* do criptossistema homomórfico em nível YASHE. Técnicas de implementação paralela em CUDA foram aplicadas no desenvolvimento da aritmética necessária e explorou-se propriedades especiais de sua estrutura algébrica com o intuito de reduzir a complexidade computacional de certas operações, como por exemplo redução polinomial e modular. Aplicou-se o Teorema Chinês do Resto (*CRT*) com o intuito de evitar o uso de aritmética multi-precisão e foi realizada análise de como a Transformada Rápida de Fourier, *FFT*, e a *Number-Theoretic Transform*, *NTT*, podem ser aplicadas na redução da complexidade de multiplicações polinomiais. Por fim, a biblioteca CUYASHE foi desenvolvida e disponibilizada à comunidade [Alves and Aranha 2016c]. Ela consolida as conclusões apresentadas neste documento e, em particular, demonstra ganhos de velocidade entre 6 até 35 vezes na multiplicação homomórfica em relação ao estado da arte, uma operação vital para o criptossistema.

Resultados preliminares deste trabalho foram apresentados no X Workshop de Teses, Dissertações e Trabalhos de Iniciação Científica em Andamento do Instituto de Computação da Unicamp (X WTD) [Alves and Aranha 2015a]; e no XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2015) [Alves and Aranha 2015b]. Suas conclusões foram apresentadas na defesa de mestrado do autor [Alves and Aranha 2016a] e no I Encontro de Teoria da Computação (ETC 2016) dentro do XXXVI Congresso da Sociedade Brasileira de Computação (CSBC 2016) [Alves and Aranha 2016b].

2. Fundamentação teórica

A implementação eficiente da aritmética de um criptossistema é fator fundamental para o ganho de velocidade em suas operações. Neste trabalho, foi necessário o estudo de estratégias para a implementação eficiente da aritmética polinomial sobre inteiros multi-precisão em *GPUs*. Essas estratégias envolveram a simplificação da implementação da aritmética por meio do *CRT* e o uso de variantes da Transformada Discreta de Fourier na redução da complexidade computacional de multiplicações polinomiais.

Teorema Chinês do Resto (*CRT*). O *CRT* foi utilizado como ferramenta para simplificar a manipulação dos coeficientes polinomiais. Por meio do teorema, um polinômio com coefi-

cientes inteiros arbitrariamente grandes é decomposto em um conjunto de polinômios com coeficientes inteiros arbitrariamente pequenos chamados polinômios residuais, ou simplesmente resíduos. Essa substituição permite que os coeficientes sejam manipulados utilizando uma aritmética mais simples e suportada nativamente pela arquitetura.

A aplicação desse teorema implica no encarecimento das operações aritméticas, uma vez que precisam ser replicadas entre os resíduos de cada um dos operandos. É esperado que esse problema seja absorvido pela capacidade de processamento paralelo da *GPU*.

Multiplicação polinômial. A quantidade de operações necessárias em um algoritmo simples para o cálculo de uma multiplicação polinomial tem custo $\Theta(N^2)$, para operandos de grau N, o que compromete a escalabilidade dessa operação. No contexto dos criptossistemas baseados no problema *RLWE*, como observado por Lindner e Peikert, a segurança é fortemente relacionada com o grau do anel de polinômios [Lindner and Peikert 2011]. Especificamente para o YASHE, Lepoint e Naehrig utilizam parâmetros com N variando de 2^{11} até 2^{16} para atingir um padrão de segurança equivalente a $\lambda = 80$ *bits* [Lepoint and Naehrig 2014]. Dessa forma, a multiplicação de polinômios de grau alto é uma operação vital para esses esquemas, o que implica que melhorias de velocidade geram impacto considerável na velocidade de suas operações.

Nesse sentido, duas transformadas, variantes da Transformada Discreta de Fourier e capazes de prover o ganho de velocidade almejado se destacam e foram estudas neste trabalho:

Transformada Rápida de Fourier (*FFT*). O algoritmo da Transformada Rápida de Fourier, ou *FFT*, pode ser empregado na redução da complexidade computacional de uma multiplicação polinomial [Cooley and Tukey 1965]. Utiliza aritmética do corpo dos números complexos e provê um domínio em que multiplicações polinomiais são reduzidas a multiplicação elemento-a-elemento. Dessa forma, considerando o custo da aplicação da transformada sobre os operandos, essa operação assume complexidade $\Theta(N \log N)$, o que representa uma melhoria significativa de velocidade.

Number-Theoretic Transform (*NTT*). Enquanto a *FFT* constrói sua aritmética sobre um elemento do corpo complexo, a *NTT* propõe uma abordagem diferente e faz uso de um elemento de um corpo finito. Dessa forma, a aplicação da *NTT* em contextos onde se utilize operações sobre inteiros, como é o caso do YASHE, evita custos relacionados a conversão dos operandos e erros de precisão por conta da aritmética de ponto flutuante da *FFT*, mantendo a redução de complexidade.

3. Implementação

A biblioteca CUYASHE foi desenvolvida durante a execução deste trabalho. Ela foi escrita em C++; utiliza paralelismo sobre CUDA para acelerar a aritmética polinomial; *CRT* para simplificar a manipulação dos operandos na *GPU*; e oferece uma implementação comparativa das transformadas *FFT* e *NTT* na redução da complexidade da multiplicação polinomial. Seu código está disponível ao público sob uma licença GNU GPLv3 [Alves and Aranha 2016c].

A aplicação do *CRT* nos operandos da CUYASHE é feita completamente na *GPU*. Dessa forma, uma vez que os resíduos ainda não foram calculados, o suporte à aritmética de multi-precisão nesse processador torna-se necessário. A biblioteca RELIC [Aranha and Gouvêa 2009] foi utilizada como base e as rotinas requeridas (adição, subtração,

multiplicação, divisão e resto modular) foram adaptadas para CUDA. Enquanto isso, a manipulação de inteiros multi-precisão pela *CPU* (por exemplo, na inicialização e cópia dos dados para a memória global da placa gráfica) foi implementada sobre a biblioteca *NTL* [Shoup 2003].

A implementação da *FFT* foi realizada sobre a biblioteca CUFFT [NVIDIA 2015]. Para isso, precisou-se aplicar operações de conversão dos coeficientes entre o conjunto dos inteiros e o corpo dos complexos. Os erros de precisão oriundos da aritmética de ponto flutuante foram mitigados por meio da redução do tamanho dos coprimos utilizados na decomposição do *CRT*.

Como não se encontrou biblioteca semelhante para a *NTT*, foi necessário que se projetasse e implementasse um algoritmo próprio para essa transformada. A formulação de Stockham para raios 2 e 4 foi utilizada como base [Govindaraju et al. 2008]. Para aplicar a *NTT* de raio R em uma sequência de N elementos, o algoritmo faz uso de $\log_R N$ iterações. Em cada iteração, a transformada é aplicada em R subsequências de tamanho N_s, começando com $N_s = 1$. Ao final, define-se $N_s = RN_s$ e uma nova iteração é iniciada até que $N_s = N$.

Para maximizar os ganhos com o uso das estratégias descritas, a CUYASHE trabalha com resíduos dentro do domínio da transformada. Ou seja, aplica-se o *CRT* seguido da *FFT* ou *NTT* em cada um dos polinômios residuais. Assim, as operações polinomiais de adição e multiplicação têm aplicação coeficiente-a-coeficiente, o que implica em complexidade linear. Além disso, os resíduos são mantidos exclusivamente na memória da *GPU*, dispensando os custos envolvidos com a cópia de dados entre as memórias.

Por fim, desejou-se que a amostragem de distribuições probabilísticas requeridas pelo criptossistema, estreita e Gaussiana discreta, fosse executada exclusivamente na *GPU*, o que elimina o custo de cópia de dados entre as memórias. Para isso, a implementação dessas distribuições foi feita utilizando a biblioteca CURAND [NVIDIA 2015].

4. Resultados

Com o intuito de avaliar a qualidade da implementação e das estratégias utilizadas na CUYASHE, foram tomados quatro trabalhos no estado da arte: Bos *et al.* (BLLN) [Bos et al. 2013]; a implementação de Lepoint e Naehrig (*LN*) [Lepoint and Naehrig 2014]; a biblioteca de Dowlin *et al.* (SEAL) [Dowlin et al. 2015]; e o trabalho de Pöppelmann *et al.* (*PNPM*) [Pöppelmann et al. 2015]. Os três primeiros são baseados em *CPUs*, enquanto o último apresenta uma implementação em *FPGAs*. Foram utilizados os parâmetros de configuração do YASHE propostos por Bos *et al.* como padrão para as comparações de tempo de execução[1]. Lepoint e Naehrig argumentam que esses parâmetros oferecem um nível de segurança de 80 *bits*.

As implementações *LN* e SEAL foram disponibilizadas por seus autores à comunidade. Dessa forma, os tempos de execução das principais operações do YASHE (cifração, decifração, adição e multiplicação homomórfica) apresentados para essas implementações, assim como para a CUYASHE, foram medidos em uma mesma máquina portando uma *CPU Intel Xeon E5-2630 @ 2,6GHz* com uma *GPU NVIDIA GeForce GTX TITAN Black @ 0,98 GHz*. Além disso, utiliza-se a versão 7.5 do kit de desenvolvimento CUDA; a versão 4.8.4 do compilador *g++*; e empregou-se a biblioteca *NTL* 9.1.0 compilada sobre a *GMP*

[1] $R = \mathbb{Z}[X] / \left(x^{4096} + 1\right)$, $q = 2^{127} - 1$, $w = 2^{32}$, $t = 2^{10}$.

6.0.0 [Shoup 2003, Granlund 2012]. Os resultados apresentados são os tempos médios calculados a partir de 100 execuções isoladas de cada operação. A comparação com a BLLN, por outro lado, precisou ser feita exclusivamente por meio dos resultados oferecidos pelos autores, uma vez que seu código não é público. Eles afirmam que os tempos foram medidos sobre uma *CPU Intel Core i7-3520 @ 2,8 GHz* [Bos et al. 2013].

Como comparação adicional, desejou-se avaliar operações intermediárias implementadas na CUYASHE. Para isso, tomou-se como referência a biblioteca CUHE, apresentada no trabalho de Dai e Sunar [Dai and Sunar 2015]. Ela faz uso de paralelismo com a plataforma CUDA e propõe uma série de otimizações na implementação da aritmética polinomial de esquemas baseados em reticulados. Apesar de utilizarem criptossistemas diferentes, a CUHE e a CUYASHE compartilham a estratégia de empregar a *NTT*. Dessa forma, decidiu-se desconsiderar os tempos de execução para operações do criptossistema e apenas manter a comparação para essa função.

4.1. Multiplicação polinomial

Duas estratégias foram testadas para a implementação da multiplicação polinomial: *FFT*, implementada pela CUFFT; e *NTT*, implementada com código próprio. A Tabela 1 apresenta uma comparação do tempo necessário para aplicar uma multiplicação polinomial utilizando cada uma delas. Como pode ser visto, a CUFFT se mostrou de 4 até 7 vezes mais rápida.

Tabela 1. Comparação dos tempos necessários para a multiplicação de dois operandos utilizando a CUFFT e a implementação da *NTT* baseada na formulação de Stockham de raio 4. O *CRT* foi executado com primos de 19 *bits* para a CUFFT e 24 para a *NTT*.

Grau	CUFFT (ms)	*NTT* (ms)
1024	0,3	2,16
2048	0,53	2,06
4096	0,9	4,61
8192	1,71	9,05

Apesar disso, quando comparada com a CUHE, a nossa implementação da *NTT* apresentou ganhos consideráveis, como visto na Tabela 2.

Tabela 2. Comparação dos tempos de execução para *NTT* com raio 4 entre as bibliotecas CUYASHE e CUHE. Os parâmetros de execução são definidos por Dai-Sunar onde os primos usados pelo *CRT* possuem 24 *bits* e são usados respectivamente 15, 25 e 40 polinômios residuais para anéis de grau 8.192, 16.384 e 32.768. Os tempos apresentados para a CUHE foram fornecidos pelos autores em seu trabalho e medidos em uma *GPU NVIDIA GeForce GTX690 @ 1,020GHz*.

Biblioteca	Grau	*NTT* (ms)
CUYASHE	8.192	0,12
CUHE	8.192	0,84
CUYASHE	16.384	0,51
CUHE	16.384	1,78

A *NTT* implementada pela CUHE é uma versão iterativa do algoritmo de Cooley-Tukey [Cooley and Tukey 1965], adaptada para aritmética de corpos finitos. Em relação

a CUHE, a CUYASHE apresenta ganho de 7 vezes em um anel de grau 8.192 e 3,5 vezes em um anel de grau 16.384. Essas resultados sugerem que, apesar de não ter sido capaz de trazer ganho de velocidade em comparação com a CUFFT, a *NTT* implementada neste trabalho tem desempenho compatível ou até mesmo superior ao estado da arte.

A maior velocidade da CUFFT implicou em seu uso como estratégia padrão para multiplicação polinomial na CUYASHE, e dessa forma foi utilizada na obtenção dos resultados apresentados adiante.

4.2. Operações do YASHE

Como pode ser visto na Tabela 3, as operações de cifração, decifração e multiplicação homomórfica da CUYASHE apresentaram tempos 8, 7 e 9 vezes menores do que o trabalho *LN* e 15, 6 e 2, 5 vezes menores do que o BLLN, respectivamente. Sobre a SEAL os ganhos foram ainda mais expressivos, atingindo 19, 17 e 35 vezes, respectivamente. Esses resultados demonstram a influência dos ganhos de velocidade na aritmética polinomial da CUYASHE, principalmente em relação a multiplicações uma vez que as operações do criptossistema são fortemente dependentes dela.

Tabela 3. Comparação entre a CUYASHE e as implementações *LN*, SEAL e BLLN. Os valores para as três primeiras foram medidos na mesma máquina, enquanto a última teve seus tempos fornecidos pelos autores em uma máquina similar. Os tempos para multiplicação homomórfica incluem o custo de relinearização.

Operação	CUYASHE(ms)	*LN*(ms)	SEAL(ms)	BLLN(ms)
Cifração	1,85	15,4	34,93	27
Decifração	2,01	13,71	34,1	5
Adição Homomórfica	0,02	0,59	0,18	0,02
Multiplicação Homomórfica	5,49	31,07	194,94	31

A Tabela 4 compara as operações homomórficas da *PNPM* com a CUYASHE. Apesar da adição homomórfica não ser tão eficiente, a multiplicação se mostra consideravelmente próxima, o que demonstra que a implementação do YASHE em *FPGA* pode ser feita de maneira tão eficiente quanto em *GPGPU*.

Tabela 4. Tempos para o CUYASHE e comparação com os resultados fornecidos pelo trabalho de *PNPM* [Pöppelmann et al. 2015]. Os tempos para multiplicação homomórfica incluem o custo de relinearização.

Operação	CUYASHE(ms)	*PNPM* (ms)
Adição Homomórfica	0,02	0,19
Multiplicação Homomórfica	5,49	6,75

O principal motivo para a utilização de um criptossistema homomórfico é a operação sobre criptogramas. Dessa forma, apesar de ganhos na cifração e decifração serem importantes, operações homomórficas são o alvo principal para otimizações. Assim, os expressivos ganhos de velocidade apresentados sobre o estado da arte têm grande importância na viabilização do criptossistema em uma aplicação real.

A revisão na máquina de estados da CUYASHE em relação a resultados preliminares [Alves and Aranha 2016c, Alves and Aranha 2015b] implicou em uma considerável

melhora nos tempos de execução. A versão atual permite a implementação da aritmética com operandos não apenas decompostos em resíduos pelo *CRT* como também interpolados para o domínio da *FFT* ou *NTT*. Assim, a complexidade computacional dessas operações se tornou majoritariamente linear e evitou-se o custo de aplicação da transformada. Além disso, toda a aritmética passou a ser computada exclusivamente na *GPU*, restando à *CPU* apenas o gerenciamento das operações.

5. Conclusão

Este trabalho investigou estratégias para a implementação do criptossistema YASHE em *GPGPUs* por meio da plataforma CUDA. Com os resultados obtidos, a biblioteca CUYASHE foi desenvolvida, otimizada e disponibilizada à comunidade [Alves and Aranha 2016c].

Aplicou-se o *CRT* para simplificar a manipulação de inteiros multi-precisão na *GPU* e a *FFT* para a redução da complexidade de operações de multiplicação polinomial, das quais o YASHE é altamente dependente. Além disso, realizou-se a implementação comparativa entre as transformadas *FFT* e *NTT*. A primeira foi fornecida pela biblioteca CUFFT, enquanto a *NTT* foi implementada com código próprio baseado na formulação de Stockham. Este trabalho não conseguiu gerar uma implementação da *NTT* que se equiparasse em velocidade com a CUFFT.

A comparação com outros trabalhos da literatura demonstrou ganhos expressivos de velocidade, atingindo uma redução de até 35 vezes no tempo de execução de uma operação de multiplicação homomórfica em relação ao estado da arte, o que sugere que a metodologia proposta foi adequada ao contexto.

Como trabalho futuro, espera-se a avaliação do desempenho da CUYASHE ao ser executada com parâmetros consideravelmente maiores e que evitem o ataque recentemente proposto por Albrecht *et al.* [Albrecht et al. 2016]. Tal ataque tem causado dúvidas na comunidade quanto a viabilidade do uso do YASHE em aplicações reais, como em algoritmos de análise de dados e aprendizagem de máquina que requerem considerável profundidade multiplicativa.

Referências

[Albrecht et al. 2016] Albrecht, M., Ducas, L., and Bai, S. (2016). A subfield lattice attack on overstretched ntru assumptions: Cryptanalysis of some fhe and graded encoding schemes. In *CRYPTO 2016*.

[Alves and Aranha 2015a] Alves, P. and Aranha, D. (2015a). cuYASHE: Computação sobre dados cifrados em GPGPUs. In *X Workshop de Teses, Dissertações e Trabalhos de Iniciação Científica em Andamento do IC-Unicamp*, pages 198–210.

[Alves and Aranha 2015b] Alves, P. and Aranha, D. (2015b). cuYASHE: Computação sobre dados cifrados em GPGPUs. In *XV Simpósio Brasileiro de Segurança da Informação e Sistemas Computacionais (SBSeg 2015)*, pages 55–60.

[Alves and Aranha 2016a] Alves, P. and Aranha, D. (2016a). Computação sobre dados cifrados em GPGPUs. Dissertação de mestrado, Instituto de Computação da Universidade Estadual de Campinas, Campinas, SP, Brasil.

[Alves and Aranha 2016b] Alves, P. and Aranha, D. (2016b). Computação sobre dados cifrados em GPGPUs. In *Anais do XXXVI Congresso da Sociedade Brasileira de Computação - I ETC*, pages 816–819.

[Alves and Aranha 2016c] Alves, P. and Aranha, D. (2016c). cuYASHE. `https://github.com/cuyashe-library/cuyashe`. Último acesso: 13/07/2016.

[Aranha and Gouvêa 2009] Aranha, D. F. and Gouvêa, C. P. L. (2009). RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.

[Bos et al. 2013] Bos, J., Lauter, K., Loftus, J., and Naehrig, M. (2013). Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In Stam, M., editor, *Cryptography and Coding*, volume 8308 of *Lecture Notes in Computer Science*, pages 45–64. Springer Berlin Heidelberg.

[Cooley and Tukey 1965] Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301.

[Dai and Sunar 2015] Dai, W. and Sunar, B. (2015). cuHE: A Homomorphic Encryption Accelerator Library. Cryptology ePrint Archive, Report 2015/818.

[Dinh et al. 2013] Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, 13(18):1587–1611.

[Dowlin et al. 2015] Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., and Wernsing, J. (2015). Manual for Using Homomorphic Encryption for Bioinformatics.

[Govindaraju et al. 2008] Govindaraju, N. K., Lloyd, B., Dotsenko, Y., Smith, B., and Manferdelli, J. (2008). High Performance Discrete Fourier Transforms on Graphics Processors. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Piscataway, NJ, USA. IEEE Press.

[Granlund 2012] Granlund, T. (2012). *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5.0.5 edition. Último acesso: 05/03/2016.

[Hoffa et al. 2008] Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., and Good, J. (2008). On the Use of Cloud Computing for Scientific Workflows. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 640–645.

[Lepoint and Naehrig 2014] Lepoint, T. and Naehrig, M. (2014). A Comparison of the Homomorphic Encryption Schemes FV and YASHE. In *Progress in Cryptology – AFRICACRYPT 2014*, volume 8469, pages 318–335. Springer International Publishing.

[Lindner and Peikert 2011] Lindner, R. and Peikert, C. (2011). *Better Key Sizes (and Attacks) for LWE-Based Encryption*, pages 319–339. Springer Berlin Heidelberg, Berlin, Heidelberg.

[NVIDIA 2015] NVIDIA (2015). CUDA Toolkit Documentation. Último acesso: 23/03/2016.

[Pöppelmann et al. 2015] Pöppelmann, T., Naehrig, M., Putnam, A., and Macias, A. (2015). *Accelerating Homomorphic Evaluation on Reconfigurable Hardware*. Springer Berlin Heidelberg.

[Shoup 2003] Shoup, V. (2003). NTL: A library for doing number theory. `http://www.shoup.net/ntl`. Último acesso: 05/03/2016.

[Xiao and Xiao 2013] Xiao, Z. and Xiao, Y. (2013). Security and Privacy in Cloud Computing. *IEEE Communications Surveys Tutorials*, 15(2):843–859.

Efficient Methods for Lattice-Based Cryptography

João Marcos de Mattos Barguil[1], **Paulo S. L. M. Barreto**[1]

[1]Department of Computer and Digital Systems Engineering
University of São Paulo, São Paulo, Brazil

jbarguil@usp.br, pbarreto@larc.usp.br

Abstract. *Lattices have been applied in many different ways in cryptography. Firstly used for the destruction of cryptosystems, they were later applied in the construction of new schemes, including asymmetric cryptosystems, blind signature schemes and the first methods for fully homomorphic encryption. Nonetheless, performance is still prohibitively slow in many cases. In this work, we expand techniques originally devised for homomorphic encryption, making them more general and applying them to the GGH-YK-M cryptosystem, a lattice-based public-key cryptosystem, and to the LMSV scheme, the only known homomorphic scheme that has not succumbed to IND-CCA1 key recovery attacks to this date. In our tests, we reduce public key bandwidth occupation of GGH-YK-M by an order of complexity, specifically, from* $O(n^2 \lg n)$ *down to* $O(n \lg n)$ *bits, where* n *is a public parameter of the scheme. The new technique also attains faster processing in all operations involved in an asymmetric cryptosystem, that is, key generation, encryption, and decryption. The most significant improvement in performance is in key generation, which becomes more than 3 orders of magnitude faster than previous results, while encryption becomes about 2 orders of magnitude faster. For decryption, our implementation is ten times faster than the literature. We also show that it is possible to improve security of LMSV against the quantum key recovery attacks recently published by British GCHQ. We do so by adopting non-cyclotomic lattices based on nearly-circulant irreducible polynomial rings. In our implementation, performance of encryption remains virtually the same, and decryption becomes slightly worse, a small price to pay for the improved security. Key generation, however, is much slower, due to the fact that it is necessary to use a more generic and expensive method. The existence of highly efficient dedicated methods for key generation of this secure variant of LMSV remains as an open problem.*

1. Background and motivation

There is a rising, medium-to-long term concern with the potential technological viability of quantum computers, because traditional cryptosystems based on the assumed hardness of integer factorization or discrete logarithm computation can be successfully attacked with the help of this new kind of equipment. New schemes based on different computational problems are thus necessary to address this concern, leading to the development of purely classical, but believed to be quantum-resistant constructions known as *post-quantum* cryptosystems.

One of the most popular families of post-quantum cryptosystems is that of schemes based on the theory of lattices. Lattices have permitted advancements in asym-

metric cryptography [Goldreich et al. 1997] and play a crucial role in the development of homomorphic schemes [Gentry 2009b].

One of the first lattice-based encryption schemes, proposed by Goldreich, Goldwasser and Halevi [Goldreich et al. 1997], or GGH for short, was later broken by Nguyen [Nguyen 1999], who proved that the original scheme had structural flaws. For several years GGH was deemed irretrievably broken, until Yoshino and Kunihiro [Yoshino and Kunihiro 2012] described a variant that prevented all known attacks, named GGH-YK. However, this variant does not allow for the construction of proper parameter sets, and therefore has no practical use. More recently, Barros and Schechter [de Barros and Schechter 2014] expanded this construction, proposing a modification called GGH-YK-M that effectively yields a suitable parametrization. The result is very promising, as it brings the simplicity of GGH and GGH-YK back to life. This family of schemes may, once again, be a viable foundation for the construction of other post-quantum systems.

Another relevant cryptosystem based on GGH-style lattices is the pioneering scheme devised by Gentry [Gentry 2009a]. It is the first known instance of *fully homomorphic encryption*, that is, the first scheme that supports arbitrary arithmetic to be performed over encrypted data while still yielding valid results upon decryption, as if the computation had been done over unencrypted data. It produces, though, very large keys, of the order of Terabytes, and demands Gigabytes of ciphertext to encrypt a single bit. Nevertheless, this work represents a major breakthrough, and prompted the development of many other constructions, some even based on different kinds of lattices. Unfortunately, most of these have already succumbed to cryptanalysis [Szydlo 2003, Chenal and Tang 2014], with the exception of the method proposed by Loftus *et al.* [Loftus et al. 2012], known as LMSV and a direct descendant of Gentry's original scheme. That scheme is *proven* to be IND-CCA1, uniquely among all fully or somewhat homomorphic encryption schemes known today.

Nonetheless, the biggest drawbacks of all cryptosystems based on GGH-style lattices, up to now, are their latent high bandwidth occupation and computational cost, which make this family of schemes less competitive in practice with other lattice-based encryption methods. The technique proposed by Smart and Vercauteren [Smart and Vercauteren 2010] and improved by Gentry and Halevi [Gentry and Halevi 2011], aimed at homomorphic encryption, can be modified to tackle these specific issues, being applicable not only in homomorphic schemes, but also in traditional public-key cryptography.

Very recently, the British Government Communications Headquarters (GCHQ) announced a quantum attack on their homemade Soliloquy scheme [Campbell et al. 2014]. This revelation has caused great discussion, as it allows for the recovery of the private key in polynomially-bounded time complexity. The issue is still surrounded by controversy and far from settled, but it is nevertheless a very important topic and must be taken into account when designing improvements to these methods.

2. Goals

The broad goal of this work is to layout efficient methods for lattice-based cryptography. Our focus is on improving the performance of constructions based on ideal lattices, such as the aforementioned GGH-style cryptosystems. Because efficiency in general is still one of the major hindrances to the development of these families of schemes, our goal is to allow not only for better processing times, but also for lower bandwidth occupation.

We target at generalizing existing methods, expanding their range of applications for contexts other than the ones they were originally designed for. In order to investigate the applicability of such methods in both traditional asymmetric cryptography and homomorphic encryption, we study the specific cases of the GGH-YK-M and LMSV constructions.

Also, it is crucial that these improvements do not affect negatively the security of said schemes. Moreover, increasing their security is also devised as a complimentary goal, taking into account the newly developed quantum key recovery attacks.

3. Contributions

In this work, we extend the technique proposed by Gentry and Halevi, making it more comprehensive, and apply it to a traditional public-key encryption scheme and a homomorphic cryptosystem, specifically, the GGH-YK-M and LMSV schemes. We reduce public key bandwidth occupation of GGH-YK-M by an order of complexity, specifically, from $O(n^2 \lg n)$ down to $O(n \lg n)$ bits, where n is a public parameter of the scheme. The new technique also attains faster processing in all operations involved in an asymmetric cryptosystem, that is, key generation, encryption, and decryption. The most significant improvement in the performance of GGH-YK-M is in key generation, which becomes more than 3 orders of magnitude faster than previous results, while encryption becomes about 2 orders of magnitude faster. For decryption, our implementation is ten times faster than the literature.

In LMSV, our main focus is on increasing its security against the quantum attack devised by GCHQ. We suggest a different choice of polynomial ring, namely, an irreducible instance that yields nearly circulant matrices. This particular choice is able to successfully thwart the new attack and does not have an appreciable impact on the performance of encryption and decryption, if compared to usual ring choices. Key generation, however, is greatly affected, as there is no known dedicated technique and it is necessary to resort to more generic algorithms. How to improve efficiency of key generation in this particular case remains as an open research problem, but our results show that it is possible to increase security of the scheme without great loss of performance in encryption and decryption.

3.1. Improvements on efficiency

We propose a new and more efficient method to compute the Hermite normal form (HNF) [Cohen 1993, Section 2.4.2] of matrices. In GGH-style cryptosystems, the HNF is widely used as the public-key, so improving its calculation directly impacts key generation. The obvious way to obtain shorter keys in other lattice-based settings, namely, resorting to certain rings of structured (e.g., circulant or negacyclic) matrices, fails for

GGH because mapping the private key to a public key, that is, computing the HNF, ends up destroying the underlying structure that would enable the size reduction.

Contrary to this intuitive observation, one can still benefit from an underlying structure in the private key to reduce the size of the public key in a nontrivial way. This was first indicated by Smart and Vercauteren [Smart and Vercauteren 2010], but it seems to require computing the HNF of the lattice basis. Gentry and Halevi [Gentry and Halevi 2011, Lemma 1] offer a proof of this property that avoids computing the HNF for the case $p(x) = x^n + 1$ (where n is a power of 2). We show that, in fact, it holds for any ideal matrix, regardless of the choice of $p(x)$, even though some choices may be more efficient (and possibly more secure) than others.

Furthermore, by applying this method for generating keys, both encryption and decryption can be simplified, allowing for an even greater impact on general performance of these cryptosystems.

3.2. Improvements on security

In recent times, new key recovery attacks to cyclotomic rings have been described. There exist classical approaches, but a new quantum attack in particular has provoked heated discussions, as it represents big news related to what had been always deemed as *"post-quantum cryptography"* up until this point. It is known as the Soliloquy [Campbell et al. 2014] quantum-attack, developed by GCHQ. This issue is still somewhat controversial, in particular whether the attack is polynomial-time or not. There is still very limited literature on the subject, and the matter is far from settled. Nonetheless, it is very important to take note of it, as it seems to apply to some of the most commonly used lattices, including the homomorphic schemes of Smart-Vercauteren [Smart and Vercauteren 2010] and LMSV [Loftus et al. 2012].

The new method mentioned in 3.1 can be modified to improve security against Soliloquy, without great impact on performance. We do so by adopting the suggestion by Bernstein [Bernstein 2014], namely, adopting a field of form $\mathcal{Z}[x]/(x^n - x - 1)$, which yields nearly circulant matrices and fairly efficient arithmetic, instead of circulant fields of form $\mathcal{Z}[x]/(x^n - 1)$.

4. Results

We have implemented the newly developed methods, and report the results obtained by applying them to two cryptosystems: GGH-YK-M and LMSV. For the former, our alternative reduces public key bandwidth occupation by an order of complexity, specifically, from $O(n^2 \lg n)$ down to $O(n \lg n)$, where n is a public parameter of the scheme. The new technique also attains faster processing in all operations involved in a public-key cryptosystem, that is, key generation, encryption, and decryption. By far the most pronounced improvement in performance is in key generation, which becomes more than 3 orders of magnitude faster than published results [de Barros and Schechter 2014], while encryption becomes about 2 orders of magnitude faster. For decryption, our implementation is ten times faster than the literature. Key sizes are essentially the same in our proposal for a given dimension n regardless of the choice of $p(x)$. Sample public key sizes are listed on Table 2.

Table 1. Timings for GGH-YK-M (in seconds)

source	(n, σ, h, k)	keygen (s)	encrypt (ms)	decrypt (ms)
previous[1]	$(350, 256, 526, 64)$	368.16	13.3	37.6
Java	$(353, 256, 526, 64)$	0.34	2.7	55.2
C	$(353, 256, 526, 64)$	0.10	0.1	4.8
previous[1]	$(400, 256, 601, 64)$	692.48	15.5	59.8
Java	$(401, 256, 601, 64)$	0.46	2.0	67.9
C	$(401, 256, 601, 64)$	0.12	0.1	5.3
Java	$(509, 256, 769, 80)$	1.04	4.0	166.3
C	$(509, 256, 769, 80)$	0.22	0.2	9.4
Java	$(512, 256, 769, 80)$	3.01	2.4	124.5
C	$(512, 256, 769, 80)$	0.11	0.2	9.8

[1] [de Barros and Schechter 2014].

Table 2. Public key sizes for GGH-YK-M (in bits)

source	(n, σ, h, k)	$\|pk\|$
previous[1]	$(350, 256, 526, 64)$	1157800
ours	$(353, 256, 526, 64)$	6682
previous[1]	$(400, 256, 601, 64)$	1543200
ours	$(401, 256, 601, 64)$	7738
previous[1]†	$(512, 256, 769, 80)$	2621440
ours	$(512, 256, 769, 80)$	10240

[1] [de Barros and Schechter 2014].

† Inferred.

For LMSV [Loftus et al. 2012], we compare the cyclotomic ring $\mathcal{Z}[x]/(x^n - 1)$ and the more secure irreducible ring $\mathcal{Z}[x]/(x^n - x - 1)$. Performance of encryption remains virtually the same, and decryption becomes slightly worse. Key generation, however, is much slower, due to the fact that it is necessary to use a more generic and expensive method. The scheme was implemented in Java on an Intel i5-2450M 2.5 GHz platform under 64-bit Ubuntu 14.10, and the results are shown in Table 3.

Table 3. Timings for LMSV

$P(x)$	n	keygen (s)	encrypt (ms)	decrypt (ms)
$x^n - 1$	353	0.3	1.5	41.1
$x^n - x - 1$	353	15.3	1.8	45.0
$x^n - 1$	401	0.4	2.5	60.1
$x^n - x - 1$	401	20.3	2.4	64.8
$x^n - 1$	509	1.4	7.2	129.9
$x^n - x - 1$	509	42.8	7.3	134.0
$x^n - 1$	2039	100.3	390.0	10.4×10^3
$x^n - 1$	8191	7.3×10^3	18.3×10^3	879×10^3

Despite being a widely used and throughly available platform, Java provides only limited support for cryptographic applications. The biggest hindrance is due to the fact that *BigInteger* objects are immutable. This means that new instances are created for each

and every arithmetic operation. When a large amount of such operations is executed, a significant impact on performance is observed, caused not by the arithmetic calculations themselves, but by the expensive memory management associated to repeated object instantiation. Analyzing the results presented in Table 3, it is clear that for $n > 509$ these limitations distort the obtained results, as timings increase disproportionately fast.

Implementing the algorithm in C or C++, for instance, could help achieve better performance. It would not, however, address the main issue, that is, the complexity of key generation in $\mathcal{Z}[x]/(x^n - x - 1)$. It is then an open question whether the lower *asymptotic* complexity generates an appreciable improvement for the rather small values of n that are used in practice. Hence, the issue of poor performance in $\mathcal{Z}[x]/(x^n - x - 1)$ is not simply a matter of fine-tuning the implementation, but still an algebraic problem.

5. Publications

Part of the work presented in Section 4 and some preliminary results have been published [Barguil et al. 2014] at SBSeg 2014.

While exploring applications of somewhat homomorphic encryption that might benefit from our new techniques, we briefly considered some e-cash protocols, in particular Sarkar's [Sarkar 2013] because of its apparent reliance on ring operations. While that protocol turned out not to be a significant example of a scenario where somewhat homomorphic encryption would be useful *per se*, as a side contribution of our analysis we were able to entirely break it, undermining all of its security goals. These results have also been published [Barguil and Barreto 2015], and are summarized as an Appendix in the full thesis.

Finally, the complete thesis [Barguil 2015] is available for download from the archives of the University of São Paulo: `http://www.teses.usp.br/teses/disponiveis/3/3141/tde-12072016-083255/pt-br.php`.

References

Barguil, J. M. M. (2015). Efficient methods for lattice-based cryptography. Master's thesis, University of São Paulo.

Barguil, J. M. M. and Barreto, P. S. L. M. (2015). Security issues in Sarkar's e-cash protocol. *Information Processing Letters*, 115(11):801–803.

Barguil, J. M. M., Lino, R. Y., and Barreto, P. S. L. M. (2014). Efficient variants of the GGH-YK-M cryptosystem. In *Brazilian Symposium on Computer System and Information Security – SBSeg 2014*, Belo Horizonte, Brazil. SBC.

Bernstein, D. J. (2014). A subfield-logarithm attack against ideal lattices. Blog entry.

Campbell, P., Groves, M., and Shepherd, D. (2014). Soliloquy: A cautionary tale. In *ETSI 2nd Quantum-Safe Crypto Workshop*, Ottawa, Canada. ETSI.

Chenal, M. and Tang, Q. (2014). On key recovery attacks against existing somewhat homomorphic encryption schemes. In *International Conference on Cryptology and Information Security in Latin America – Latincrypt 2014*, Lecture Notes in Computer Science, Florianópolis, Brazil. Springer.

Cohen, H. (1993). *A course in computational algebraic number theory*, volume 138. Springer, Berlin, Germany.

de Barros, C. F. and Schechter, L. M. (2014). GGH may not be dead after all. In *XXXV Congresso Nacional de Matemática Aplicada e Computacional – CNMAC 2014*, Natal, Brazil. Sociedade Brasileira de Matemática Aplicada e Computacional – SBMAC.

Gentry, C. (2009a). *A fully homomorphic encryption scheme*. PhD thesis, Stanford University.

Gentry, C. (2009b). Fully homomorphic encryption using ideal lattices. In *XLI Annual ACM Symposium on Theory of Computing – STOC 2009*, volume 9, pages 169–178, Bethesda, MD, USA. ACM.

Gentry, C. and Halevi, S. (2011). Implementing Gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148, Tallinn, Estonia. Springer.

Goldreich, O., Goldwasser, S., and Halevi, S. (1997). Public-key cryptosystems from lattice reduction problems. In *Advances in Cryptology – CRYPTO '97*, pages 112–131. Springer, Santa Barbara, USA.

Loftus, J., May, A., Smart, N. P., and Vercauteren, F. (2012). On CCA-secure somewhat homomorphic encryption. In *International Conference on Selected Areas in Cryptography – SAC 2011*, volume 7118 of *Lecture Notes in Computer Science*, pages 55–72, Toronto, Canada. Springer.

Nguyen, P. (1999). Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from CRYPTO '97. In *Advances in Cryptology – CRYPTO '99*, pages 288–304, Santa Barbara, USA. Springer.

Sarkar, P. (2013). Multiple-use transferable e-cash. *International Journal of Computer Applications*, 77(6):35–38.

Smart, N. P. and Vercauteren, F. (2010). Fully homomorphic encryption with relatively small key and ciphertext sizes. In *XIII International Conference on Practice and Theory in Public Key Cryptography – PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443, Paris, France. Springer.

Szydlo, M. (2003). Hypercubic lattice reduction and analysis of GGH and NTRU signatures. In *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 433–448. Springer, Warsaw, Poland.

Yoshino, M. and Kunihiro, N. (2012). Improving GGH cryptosystem for large error vector. In *XI International Symposium on Information Theory and its Applications – ISITA 2012*, pages 416–420, Honolulu, HI, USA. IEEE.

Amostragem Gaussiana aplicada à Criptografia Baseada em Reticulados

Jheyne N. Ortiz[1], **Ricardo Dahab**[1], **Diego F. Aranha**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Campinas – SP – Brasil

{jheyne.ortiz,rdahab,dfaranha}@ic.unicamp.br

Abstract. *In Lattice-Based Cryptography, cryptosystems usually require sampling lattice points and integers following a Gaussian distribution. Sampling lattice points can also be used to solve variants of the SVP (Shortest Vector Problem) and CVP (Closest Vector Problem). This work presents constant-time implementations of the Knuth-Yao and discrete Ziggurat methods for Gaussian sampling over integers. The Knuth-Yao implementation was applied to sampling over lattices and to the Ring-LWE-based (Learning with Errors over Rings) encryption scheme. Our experiments targeted an Intel Ivy Bridge processor and all implementations are in C++ supported by Victor Shoup's NTL library.*

Resumo. *Na Criptografia Baseada em Reticulados, vários esquemas requerem a amostragem de vetores de um reticulado e de inteiros seguindo uma distribuição que, convencionalmente, é Gaussiana. A amostragem de vetores de um reticulado pode também ser usada para resolver variantes dos problemas SVP (vetor mais curto) e CVP (vetor mais próximo). Este trabalho apresenta implementações com tempo de execução constante para os métodos Knuth-Yao e Ziggurat Discreto, apropriados à amostragem Gaussiana sobre os inteiros. Uma implementação para o método Knuth-Yao é aplicada à amostragem sobre reticulados e ao esquema de encriptação baseado no problema LWE (Learning with Errors) sobre anéis. Os experimentos foram feitos em processador Intel Ivy Bridge, usando C++ com suporte da biblioteca NTL de Victor Shoup.*

1. Introdução

A Criptografia Baseada em Reticulados é um conjunto de primitivas criptográficas calcado em problemas difíceis em reticulados, capazes de prover encriptação [Lyubashevsky et al. 2010, Stehlé and Steinfeld 2011], assinatura digital [Ducas et al. 2013, Hoffstein et al. 2014], acordo de chaves [Bos et al. 2015, Alkim et al. 2015] e esquemas funcionais [O'Neill 2010, Boneh et al. 2011]. Até o presente, tais primitivas se mostraram resistentes a ataques clássicos e quânticos. Além de resistentes a ataques quânticos, esquemas como o de encriptação baseado no problema Ring-LWE [Lyubashevsky et al. 2012] (*Learning with Errors over Rings*) e o NTRU [Hoffstein et al. 2010] são simples e eficientes, sendo candidatos à substituição de criptossistemas em uso atualmente, ameaçados pelos algoritmos de Peter Shor.

Esquemas sobre reticulados, como a Encriptação Baseada em Identidades [Agrawal et al. 2010], as funções de resumo seguidas de assinatura [Gentry et al. 2008] e Baseada em Atributos [Boneh et al. 2014], requerem a

amostragem de vetores estatisticamente próximos de um reticulado. Além disso, a amostragem sobre os inteiros é um passo nos métodos de amostragem em reticulados [Gentry et al. 2008, Peikert 2010], além de necessária em esquemas de encriptação [Lyubashevsky et al. 2012] e no esquema de assinatura [Ducas et al. 2013] baseados no problema Ring-LWE [Regev 2005]. Usualmente, ambos os tipos de amostragens seguem uma distribuição Gaussiana.

No esquema de encriptação baseado no problema Ring-LWE, a amostragem de ruídos ocorre na fase de encriptação, que comumente é implementada em um dispositivo vulnerável a ataques por canais laterais. Similarmente, como demonstrado por Bruinderink et al., o esquema de assinatura BLISS é suscetível a ataques por canais laterais, bem como o amostrador utilizado na fase de assinatura [Bruinderink et al. 2016]. Assim, a fim de evitar ataques por canais laterais de tempo, uma medida preventiva é a implementação com tempo de execução constante, que descorrelaciona a latência do algoritmo da informação que está sendo processada.

Objetivos. Em concordância com os fatores motivadores acima expostos, nosso objetivo aqui é o estudo e a implementação segura de métodos para amostragem Gaussiana de vetores de reticulados e de inteiros. Em particular, a implementação de algoritmos específicos para o esquema de encriptação baseado no Ring-LWE, para um esquema HIBE (*Hierarchical Identity-Based Encryption*) [Mochetti and Dahab 2014] e para reticulados NTRU [Hoffstein et al. 2010, Lyubashevsky and Prest 2015, Ducas and Prest 2015].

Contribuições. Neste trabalho, implementações com tempo constante dos algoritmos Knuth-Yao [Knuth and Yao 1976] e Ziggurat discreto [Buchmann et al. 2014] são propostas para a tarefa de amostragem de inteiros conforme uma distribuição Gaussiana. Por ser um método mais eficiente e mais propício a implementações resistentes a ataques por canais de tempo, uma implementação do método Knuth-Yao de propósito geral é aplicada ao contexto de amostragem Gaussiana de pontos de um reticulado. Ainda, este trabalho apresenta uma implementação para o Knuth-Yao específica para o esquema de encriptação baseado no problema Ring-LWE [Lyubashevsky et al. 2012], cuja distribuição de erros tem parâmetros fixos. Como resultado desta dissertação, resultados preliminares foram publicados no XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais [Ortiz et al. 2015a], e agraciado com o prêmio de terceiro melhor artigo, e no X Workshop de Teses, Dissertações e Trabalhos de Iniciação Científica do Instituto de Computação da Universidade Estadual de Campinas [Ortiz et al. 2015b].

2. Aspectos da Amostragem Gaussiana Discreta

A distribuição Gaussiana unidimensional sobre os inteiros com centro $c \in \mathbb{R}$ e desvio padrão $\sigma \in \mathbb{R}^+$, denotada por $\mathcal{D}_{\mathbb{Z},\sigma,c}$, é definida na Equação 1.

$$\mathcal{D}_{\mathbb{Z},\sigma,c}(x) = \frac{\rho_{\sigma,c}(x)}{\sum_{y=-\infty}^{\infty} \rho_{\sigma,c}(y)}, \text{ com } \rho_{\sigma,c}(x) := \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-c)^2}{2\sigma^2}\right) \tag{1}$$

Tendo em vista que o intervalo de amostragem definido na Equação 1 é infinito, algoritmos em máquinas finitas são capazes somente de amostrar elementos de distribuições

Gaussianas estatisticamente próximas da ideal. Neste sentido, as amostragens ocorrem no intervalo $[c - t\sigma, c + t\sigma] \cap \mathbb{Z}$, com t o comprimento da cauda da distribuição. Portanto, a cauda da distribuição, ou seja, a área correspondente a $|x| > t\sigma$, é ignorada, relativamente a uma precisão $2^{-\lambda}$. Comumente, o corte de cauda empregado em primitivas criptográficas corresponde a $t = 13{,}2$, de forma que, para todo $\sigma \geq 1$, a massa correspondente à região da cauda é negligenciável, tal que

$$1 - \frac{S_\sigma(13, 2\sigma)}{S_\sigma(\infty)} < 2^{-128},$$

com S_σ a função cumulativa $S_\sigma(b) = \sum_{k=-b+1}^{b-1} \rho_{\sigma,0}(k)$ para $b \geq 1$ [Saarinen 2016].

Em um esquema criptográfico, tanto os parâmetros da distribuição como a distância estatística são determinados por cálculos oriundos da demonstração de segurança do esquema. Habitualmente, a distância estatística carece de um valor inferior a $2^{-\lambda}$, com λ o nível de segurança. Para tal, a implementação de um método de amostragem Gaussiana sobre os inteiros deve considerar precisão mínima de λ bits para as operações com ponto flutuante. Ademais, a precisão em bits define a dimensão das tabelas de consulta, impactando nos custos espacial e temporal dos algoritmos. Assim, além de resistentes a ataques por canais laterais, tais implementações devem ser eficientes.

3. Resultados Experimentais

A Figura 1 é uma representação do escopo desta dissertação. Tendo em vista reticulados NTRU e um esquema HIBE [Mochetti and Dahab 2014], nossas contribuções consistem na implementação de métodos para a amostragem de pontos de um reticulado que, por sua vez, requerem um oráculo para amostragem de inteiros, assim como criptossistemas baseados no problema Ring-LWE. A amostragem em ambos os casos segue uma função Gaussiana. Nesta seção, todas as implementações estão em linguagem C++ e utilizam a biblioteca NTL [Shoup 2016] para geração de valores aleatórios e para operações com vetores e matrizes. Além disso, as implementações tem como alvo um processador Ivy Bridge Intel®Core™i5-3570 @ 3.40 GHz e 8 GB de memória física e de *swap*.

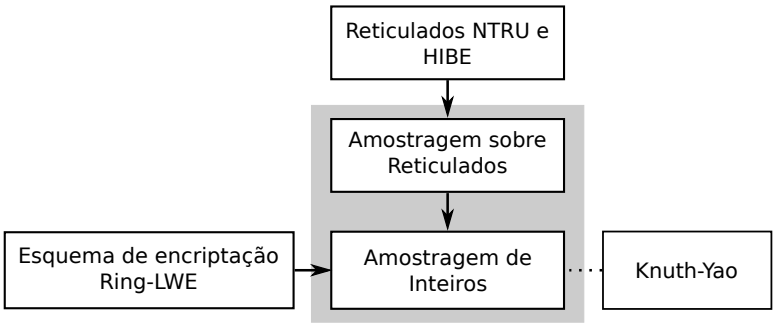


Figura 1. Amostragem Gaussiana para esquemas sobre reticulados.

Para a amostragem sobre os inteiros, com base em experimentos realizados com o algoritmo Knuth-Yao e o método discreto de Ziggurat, o método Knuth-Yao apresenta tempos de execução com uma ordem de magnitude menor do que os obtidos para o Ziggurat. Portanto, implementações com tempo constante para o Knuth-Yao foram propostas

para o esquema de encriptação baseado no problema Ring-LWE, bem como para métodos de amostragem de pontos de um reticulado. Implementações com tempo constante são resistentes a ataques por canais laterais de tempo, mas com tempos de execução superiores aos reportados em implementações com tempo variante. Tal acréscimo deve-se à necessidade de forçar a execução do algoritmo para o pior caso, bem como à inserção de novas instruções para que os caminhos de execução tenham todos o mesmo tamanho.

3.1. Esquema de Encriptação Ring-LWE

Para o esquema Ring-LWE, os parâmetros da distribuição Gaussiana são fixos, diferentemente de alguns métodos para amostragem sobre reticulados que requerem inteiros amostrados conforme uma função Gaussiana distinta a cada iteração. Neste sentido, tabelas de consulta podem ser calculadas e definidas estaticamente para acesso futuro na fase de amostragem.

De forma geral, o método Knuth-Yao consiste em processar uma matriz de probabilidades binária, que representa a parte negativa da distribuição Gaussiana. Assim, a matriz de probabilidades contém uma grande região nula “mais à esquerda"que permite o emprego de técnicas de compactação. Uma dessas técnicas é o armazenamento decimal das linhas da matriz binária. Além disso, a visitação de tais bits pode ser feita linha-a-linha em vez de bit-a-bit, uma vez que os pesos de Hamming das linhas da matriz são conhecidos. Nesta implementação, atribuições e desvios condicionais são substituídos por funções com tempo constante compostas de operadores aritméticos e lógicos.

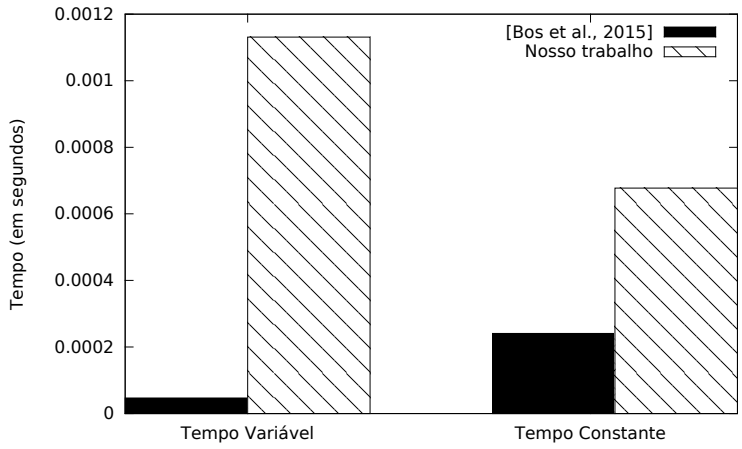


Figura 2. Comparação entre métodos para amostragem Gaussiana de inteiros.

Na literatura, Bos et al. apresentam um trabalho similar, onde uma implementação do método da transformação inversa é aplicada ao esquema assimétrico Ring-LWE [Lyubashevsky et al. 2012]. Na Figura 2, resultados em termos de tempo de execução, em segundos, são apresentados para a implementação do método da transformação inversa [Bos et al. 2015] e para o algoritmo Knuth-Yao. Em suma, nossa implementação com tempo constante é até quatro vezes menos eficiente que a implementação respectiva de Bos et al.. Apesar disso, nossa implementação requer menos bits para armazenamento das tabelas de consulta, uma diferença de 1852 bits quando $\sigma = 8/\sqrt{2\pi}$.

3.2. Amostragem Gaussiana sobre Reticulados

Para amostragem em reticulados, o esquema HIBE proposto por Mochetti e Dahab e reticulados NTRU são abordados. A base curta, que gera o reticulado q-ário do esquema HIBE [Mochetti and Dahab 2014], permite o uso de quatro algoritmos para amostragem Gaussiana: o amostrador Gaussiano usual, como apresentado em [Lyubashevsky and Prest 2015]; o método de Klein [Gentry et al. 2008], que produz vetores com normas menores e, portanto, melhores; o método de Peikert [Peikert 2010] que, em contraste com o custo quadrático do método de Klein, tem custo quase-linear apesar de produzir saídas piores; e o amostrador Gaussiano compacto [Lyubashevsky and Prest 2015], que requer somente uma fração da base de Gram-Schmidt presente na memória.

	Conjuntos de Parâmetros (n, m, q)			
Algoritmo	$(2, 12, 13)$	$(4, 35, 11)$	$(16, 111, 499)$	$(64, 133, 1019)$
AMOSTRADOR-GAUSSIANO	$0,06$	$1,02$	$51,81$	$471,86$
AG-COMPACTO	$0,05$	$0,79$	$53,81$	$775,39$
KLEIN	$0,06$	$1,02$	$44,11$	$786,92$
PEIKERT	$0,07$	$0,73$	$13,43$	$86,27$

Tabela 1. Tempos de execução em segundos para a fase de amostragem Gaussiana sobre reticulados para o esquema HIBE.

A Tabela 1 apresenta resultados dos tempos de execução, em segundos, para a fase de amostragem de tais métodos. Neste caso, os conjuntos de parâmetros são fictícios. Um conjunto de parâmetros real com nível de segurança de 128 bits é $(n, q) = (128, 2083)$ tal que $\mathbf{B} \in \mathbb{Z}^{mn \times mn}$ é uma base curta para o reticulado $\Lambda_q^{\perp}(\mathbf{B})$. Para o esquema HIBE, todos os métodos para amostragem de pontos de um reticulado possuem uma fase de pré-computação de forma a gerar e ortogonalizar ou inverter a base curta que gera o reticulado em questão. A ortogonalização é calculada utilizando o método usual de Gram-Schmidt. A fim de justificar tais escolhas de conjuntos de parâmetros, quando $(n, m, q) = (64, 133, 1019)$, o tempo despendido pelo algoritmo de Peikert na fase de pré-computação e na amostragem é de, aproximadamente, 42 horas e requer cerca de $17{,}33$ GB de espaço em memória.

Por outro lado, para reticulados NTRU, a base $\mathbf{B}$ que gera o reticulado é composta por blocos de bases isométricas, tal que

$$\mathbf{B}_{f,g,F,G} = \left[\begin{array}{c|c} \mathcal{A}(f) & \mathcal{A}(g) \\ \hline \mathcal{A}(F) & \mathcal{A}(G) \end{array}\right],$$

sendo que para cada $p \in \mathbb{Z}_N[x]$, $\mathcal{A}(p)$ denota uma matriz $N \times N$ cuja i-ésima linha é dada pelos coeficientes de $x^{i-1}p(x) \mod (x^N + 1)$. Ainda, q é um inteiro positivo e f, g, F, G são polinômios no anel $\mathbb{Z}_N[x]$ tais que $fG - gF = q \mod (x^N + 1)$.

Neste caso, os métodos OGS-EM-BLOCOS [Lyubashevsky and Prest 2015] e AMOSTRADOR-HÍBRIDO [Ducas and Prest 2015] usufruem dessa estrutura da base curta a fim de computar eficientemente a ortogonalização de Gram-Schmidt e a amostragem de pontos do reticulado, respectivamente.

	Dimensão (N)			
Algoritmo	**128**	**256**	**512**	**1024**
GERAÇÃO-DE-CHAVES	$0,10$	$0,44$	$2,20$	$11,02$
OGS-EM-BLOCOS	$0,11$	$0,45$	$2,37$	$7,43$
PRÉ-COMPUTAÇÃO-KLEIN	$3,60$	$17,15$	$97,46$	$505,39$
PRÉ-COMPUTAÇÃO-PEIKERT	$3,19$	$5,60$	$20,94$	$82,37$
AMOSTRADOR-HÍBRIDO	$15,58$	$72,38$	$373,23$	$2042,71$

Tabela 2. Tempos de execução em segundos para o método híbrido com variação no valor de N.

Na Tabela 2, a primeira amostragem sobre o reticulado NTRU tem a penalidade de todos os algoritmos, a saber GERAÇÃO-DE-CHAVES para geração da base do reticulado, OGS-EM-BLOCOS para sua ortogonalização, PRÉ-COMPUTAÇÃO-KLEIN e PRÉ-COMPUTAÇÃO-PEIKERT como fases preparatórias para a amostragem propriamente dita e, então, o AMOSTRADOR-HÍBRIDO. Nas amostragens seguintes, o único custo associado é o de execução do procedimento AMOSTRADOR-HÍBRIDO. Neste cenário, os conjuntos de parâmetros refletem os valores adotados na literatura e, no pior caso, quando $N = 1024$, a amostragem pode consumir cerca de 34 minutos.

Dimensão (N)	**Knuth-Yao**	**Método Híbrido**	**Proporção** $(\%)$
128	0,0013	1564,60	0,0085
256	0,0021	7261,47	0,0029
512	0,0037	37446,08	0,0010
1024	0,0068	204856,22	0,0003

Tabela 3. Proporção do tempo, em segundos, consumida pelo algoritmo Knuth-Yao no método híbrido.

A Tabela 3 ilustra a proporção do tempo de execução do método híbrido consumida pela tarefa de amostragem Gaussiana sobre os inteiros. Neste experimento, o método híbrido [Ducas and Prest 2015] é executado cem vezes, uma vez que há diferença na latência para amostragem do primeiro vetor e para a amostragem dos seguintes. Além disso, a amostragem de cada vetor sobre o reticulado NTRU requer que um polinômio com dimensão N seja gerado, tal que suas coordenadas são inteiros amostrados pelo algoritmo Knuth-Yao. Então, a proporção se torna ínfima com o crescimento do valor de N, posto que o tempo de execução do método híbrido cresce na proporção aproximada $5/2$ em relação ao crescimento da dimensão N.

Em ambos os casos, para reticulados NTRU e para o esquema HIBE, não há na literatura trabalhos similares que permitam uma comparação justa de desempenho. Apesar de serem passíveis de otimizações algorítmicas, os resultados para as implementações dos métodos de Klein e de Peikert, e para os métodos compacto e híbrido dão uma noção inicial da penalidade associada com a tarefa de amostragem Gaussiana sobre reticulados. O código-fonte de todas as implementações mencionadas pode ser encontrado em repositórios públicos no GitHub no endereço `https://github.com/jnortiz`.

Referências

[Agrawal et al. 2010] Agrawal, S., Boneh, D., and Boyen, X. (2010). Efficient Lattice (H)IBE in the Standard Model. In Gilbert, H., editor, *Advances in Cryptology – EUROCRYPT 2010*, volume

6110 of *LNCS*, pages 553–572. Springer Berlin Heidelberg.

[Alkim et al. 2015] Alkim, E., Ducas, L., Pöppelmann, T., and Schwabe, P. (2015). Post-quantum key exchange - a new hope. Cryptology ePrint Archive, Report 2015/1092. `http://eprint.iacr.org/`.

[Boneh et al. 2014] Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V., and Vinayagamurthy, D. (2014). Fully Key-Homomorphic Encryption, Arithmetic Circuit ABE and Compact Garbled Circuits. In Nguyen, P. and Oswald, E., editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer Berlin Heidelberg.

[Boneh et al. 2011] Boneh, D., Sahai, A., and Waters, B. (2011). Functional Encryption: Definitions and Challenges. In Ishai, Y., editor, *Theory of Cryptography*, volume 6597 of *LNCS*, pages 253–273. Springer Berlin Heidelberg.

[Bos et al. 2015] Bos, J. W., Costello, C., Naehrig, M., and Stebila, D. (2015). Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 553–570.

[Bruinderink et al. 2016] Bruinderink, L. G., Hülsing, A., Lange, T., and Yarom, Y. (2016). Flush, Gauss, and Reload – A Cache Attack on the BLISS Lattice-Based Signature Scheme. Cryptology ePrint Archive, Report 2016/300. `http://eprint.iacr.org/`.

[Buchmann et al. 2014] Buchmann, J., Cabarcas, D., Göpfert, F., Hülsing, A., and Weiden, P. (2014). Discrete Ziggurat: A Time-Memory Trade-Off for Sampling from a Gaussian Distribution over the Integers. In Lange, T., Lauter, K., and Lisoněk, P., editors, *Selected Areas in Cryptography – SAC 2013*, volume 8282 of *LNCS*, pages 402–417. Springer Berlin Heidelberg.

[Ducas et al. 2013] Ducas, L., Durmus, A., Lepoint, T., and Lyubashevsky, V. (2013). *Advances in Cryptology – CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, chapter Lattice Signatures and Bimodal Gaussians, pages 40–56. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Ducas and Prest 2015] Ducas, L. and Prest, T. (2015). A Hybrid Gaussian Sampler for Lattices over Rings. Cryptology ePrint Archive, Report 2015/660. `http://eprint.iacr.org/`.

[Gentry et al. 2008] Gentry, C., Peikert, C., and Vaikuntanathan, V. (2008). Trapdoors for Hard Lattices and New Cryptographic Constructions. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 197–206, New York, NY, USA. ACM.

[Hoffstein et al. 2010] Hoffstein, J., Howgrave-Graham, N., Pipher, J., and Whyte, W. (2010). Practical Lattice-Based Cryptography: NTRUEncrypt and NTRUSign. In Nguyen, P. Q. and Vallée, B., editors, *The LLL Algorithm*, Information Security and Cryptography, pages 349–390. Springer Berlin Heidelberg.

[Hoffstein et al. 2014] Hoffstein, J., Pipher, J., Schanck, J. M., Silverman, J. H., and Whyte, W. (2014). *Applied Cryptography and Network Security: 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings*, chapter Practical Signatures from the Partial Fourier Recovery Problem, pages 476–493. Springer International Publishing, Cham.

[Knuth and Yao 1976] Knuth, D. and Yao, A. (1976). *Algorithms and Complexity: New Directions and Recent Results*, chapter The Complexity of Nonuniform Random Number Generation. Academic Press, New York, j. f. traub edition.

[Lyubashevsky et al. 2010] Lyubashevsky, V., Peikert, C., and Regev, O. (2010). On Ideal Lattices and Learning with Errors over Rings. In Gilbert, H., editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer Berlin Heidelberg.

[Lyubashevsky et al. 2012] Lyubashevsky, V., Peikert, C., and Regev, O. (2012). On Ideal Lattices and Learning with Errors Over Rings. Cryptology ePrint Archive, Report 2012/230. `http://eprint.iacr.org/`.

[Lyubashevsky and Prest 2015] Lyubashevsky, V. and Prest, T. (2015). Quadratic Time, Linear Space Algorithms for Gram-Schmidt Orthogonalization and Gaussian Sampling in Structured Lattices. Cryptology ePrint Archive, Report 2015/257. `http://eprint.iacr.org/`.

[Mochetti and Dahab 2014] Mochetti, K. and Dahab, R. (2014). Ideal Lattice-based (H)IBE Scheme. Technical Report IC-14-18, Institute of Computing, University of Campinas.

[O'Neill 2010] O'Neill, A. (2010). Definitional Issues in Functional Encryption. Cryptology ePrint Archive, Report 2010/556. `http://eprint.iacr.org/`.

[Ortiz et al. 2015a] Ortiz, J. N., Aranha, D. F., and Dahab, R. (2015a). Implementação em Tempo Constante de Amostragem de Gaussianas Discretas. In *XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, SBSeg 2015 (full papers)*, Florianópolis, SC.

[Ortiz et al. 2015b] Ortiz, J. N., Aranha, D. F., and Dahab, R. (2015b). Implementação em Tempo Constante de Amostragem de Gaussianas Discretas. In *X Workshop de Teses, Dissertações e Trabalhos de Iniciação Científica, WTD 2015*, Campinas, SP.

[Peikert 2010] Peikert, C. (2010). An Efficient and Parallel Gaussian Sampler for Lattices. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, pages 80–97, Berlin, Heidelberg. Springer-Verlag.

[Regev 2005] Regev, O. (2005). On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA. ACM.

[Saarinen 2016] Saarinen, M.-J. O. (2016). Arithmetic Coding and Blinding for Lattice Cryptography. Cryptology ePrint Archive, Report 2016/276. `http://eprint.iacr.org/`.

[Shoup 2016] Shoup, V. (2016). Number Theory Library (NTL). Website. `http://www.shoup.net/ntl/` - Último acesso em 13 de julho de 2016.

[Stehlé and Steinfeld 2011] Stehlé, D. and Steinfeld, R. (2011). *Advances in Cryptology – EUROCRYPT 2011, Tallinn, Estonia, May 15-19, 2011. Proceedings*, chapter Making NTRU as Secure as Worst-Case Problems over Ideal Lattices, pages 27–47. Springer Berlin Heidelberg, Berlin, Heidelberg.

An Adaptive Selective Defense for Application Layer DDoS Attacks

Yuri Gil Dantas[1], **Vivek Nigam**[2], **Iguatemi E. Fonseca**[2],

[1] Technische Universität Darmstadt – Darmstadt, Hessen – Germany

[2]Federal University of Paraíba (UFPB) – João Pessoa, PB – Brazil

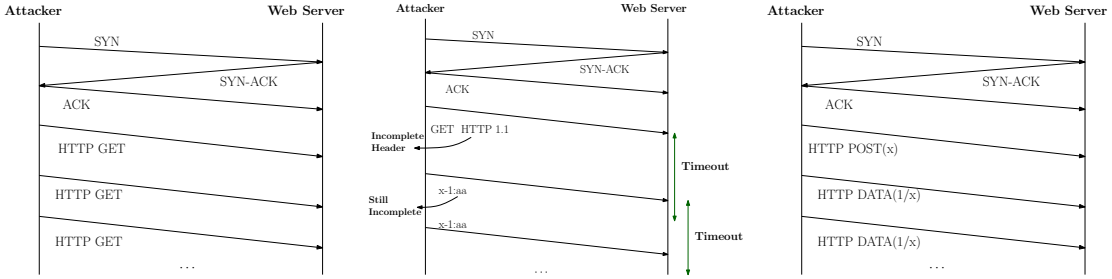
Abstract. *Recent Distributed Denial of Service (DDoS) attacks have been carried out over the application layer where an attacker can target a particular application of the server while leaving the others applications still available, thus generating less traffic and being harder to detected. This work proposes a novel defense called SeVen which uses Selective Strategies to mitigate such attacks. We validate SeVen by: (1) Simulation: The entire defense mechanism was formalized in Maude and simulated using the statistical model checker (PVeStA). 2) Experiments: Analysis of efficiency SeVen, implemented in C++, in a real experiment on the network showing that SeVen is effective in mitigating the HTTP POST and Slowloris attacks yielding high levels of availability.*

1. INTRODUCTION

Denial of Service Attacks (DDoS) have always been a major concern to network administrators. Traditionally, DDoS attacks are carried out on the transport layer by sending a number of packets to a server much greater than the server's processing capabilities making it unavailable to legitimate users. Although still dangerous, such attacks can be identified by using traffic analysis tools and mitigated by existing defenses [Zargar et al. 2013], such as the Adaptive Selective Defense [Khanna et al. 2008, Khanna et al. 2012].

In the recent years, a new generation of sophisticated attacks has emerged that exploit application layer protocols, such as HTTP and SIP protocols. Application Layer DDoS attacks (ADDoS) can target a single application in the server, *e.g.*, a web-server, instead of the whole server machine. The attack does not need to generate a huge amount of traffic, thus rendering existing defenses ineffective [Zargar et al. 2013]. Moreover, since the traffic generated by the attack is similar to legitimate traffic, it is hard to distinguish when a request is part of an attack or it is a legitimate request. These attacks are becoming increasingly preferred by attackers. In 2015, DDoS exploiting the HTTP protocol represented more than 30% of the total DDoS attacks just behind the traditional SYN-ACK DDoS with above 50% of all DDoS attacks (`https://tinyurl.com/q9p7qfz`).

Contributions: We formalize three different ADDoS attacks in the the computational tool Maude [Clavel et al. 2007], namely HTTP GET flooding, HTTP POST and Slowloris attacks (described in Section 2). We propose (Section 3) a novel defense against ADDoS attacks, called SeVen, based on ASV [Khanna et al. 2008]. As ASV was designed for mitigating transport layer DDoS attacks, it assumes that communication is a simple client-server stateless sync-ack interaction. This is, however, not enough for mitigating ADDoS attacks, as the protocols used by these attacks, such as HTTP, have a notion of state. SeVen thus extends ASV by incorporating into the defense a notion of state needed for mitigating ADDoS attacks, such as the HTTP POST and Slowloris attacks; We formalize SeVen in Maude [Clavel et al. 2007] and validate (Section 4) our defense by simulation using the



(a) Sequence of messages in the HTTP GET flooding Attack. (b) Sequence of messages in the Slowloris Attack. (c) Sequence of messages in the HTTP POST attack.

Figure 1. Behavior of three Application Layer DDoS Attacks.

statistical model checker tool PVeStA [AlTurki and Meseguer 2011]. We implement a tool for SeVen in C++. We carry out experiments on the network by realizing attacks on a Apache web-server using two different scenarios, i.e., with and without running SeVen. In each experiment, we measure the Success Rate of clients and their Time of Response (TTS). For all attacks, we observe great improvements on the Success Rate and TTS. Finally we briefly discuss the outcome of this work (Section 5).

2. ATTACK DESCRIPTIONS

HTTP GET Flooding Attack Flooding attacks [Zargar et al. 2013, Shankesi et al. 2009] are very similar to transport layer DDoS attacks in that the attacker and his zombies send a great number of packages. However, instead of consuming the resources of a whole server as in network layer DDoS, the target is an application running on the server, such as a web-server. When the attack is carried out, the application is overwhelmed and is no longer available to honest clients. The attack pattern is as shown in Figure 1(a), where the attacker establishes a connection by completing the SYN-ACK handshake with the target web-server. At this time, the attacker sends a great number of packages in order to overload such a web-server.

Slowloris Attack The Slowloris attack [slowloris 2013] exploits the HTTP GET method, used to retrieve web content. The attack follows by using the sequence of messages shown in Figure 1(b). The attacker completes the SYN-ACK handshake with the target web-server thus establishing a connection with the web-server machine exactly as a legitimate client would do. Then the attacker sends an incomplete header with a GET request. At this moment the web-server allocates one of its workers to handle this incoming request. However, since the request header is not yet complete, the web-server waits, as specified by the HTTP protocol version 1.1 [RFC 1999], until a new piece of the request header is received. If such a piece is not received until a given timeout elapses, the web-server rejects this request. The attacker can easily infer, beforehand, this timeout by measuring the time until a request with an incomplete header is rejected. Thus, once the timeout is almost reached, the attacker sends another piece of the header, which can be anything actually as long as it does not complete the header of the request. The tool implemented by Anonymous sends “x-1:aa”. This causes the web-server to reset the timeout count and wait once again for another piece of the header.

HTTP POST Attack The HTTP POST attack [r-u-dead yet 2013] exploits the POST method from the HTTP protocol. It works on web-pages that have some type of form to be filled. Figure 1(c) depicts the sequence of messages used in the HTTP POST attack.

The attacker establishes a connection by completing the SYN-ACK handshake with the web-server's machine. At this point, the attacker sends a (complete) HTTP POST request informing that it will send a large number, x, of bytes to the web-server corresponding to the contents of some form. The web-server, then, allocates one of its workers to serve this request. However, instead of sending big chunks of this data to the web-server as a legitimate client would do, the attacker sends very small pieces, typically a single byte per subsequent message. Moreover, it sends each piece in large intervals (typically 10s per byte). In the meantime the allocated worker waits until it receives all x bytes and cannot serve any other (legitimate) requests. By sending a number of POST requests greater than the number of workers in the web-server, all workers are busy waiting for attacker requests to be completed and therefore the web-server is not able to serve legitimate clients.

Slowloris attack and HTTP POST attacks do not generate high traffic load, nor high CPU and memory consumption. Thus, it is easy for administrators (or automated tool) monitoring these parameters to believe that their application is not under attack.

3. SEVEN

Our novel defense Selective Verification in Application Layer (SeVen as a reference to the number of the Application Layer in the OSI model) is based on ASV [Khanna et al. 2008]. Instead of analysing the traffic pattern of a given attack, SeVen assumes that whenever under attack the number of requests from the attacker is greater than that of honest users. An application using SeVen does not immediately process incoming messages, but waits for a period of time, t_S, called a *round*. During a round, SeVen accumulates messages received in a buffer. Depending on the size of this buffer, it may reject some requests.

More precisely, SeVen is composed of a natural number (`PMod`) and of two buffers, $\mathcal{P}, \mathcal{R}$, represented by lists:

$$\langle \mathcal{P}, \mathcal{R}, \texttt{PMod} \rangle.$$

The buffer $\mathcal{P}$ contains the requests *partially processed*, and $\mathcal{R}$ the requests *are expected to be received and be processed* by the application. The natural number `PMod` is a counter, also used in ASV, that is used to modify the probability distributions of when a request should be kept as it will become clear below. At the beginning of a round $\texttt{PMod} = 0$. The buffers $\mathcal{P}$ and $\mathcal{R}$ contain requests of the form $\langle \texttt{ip} : \texttt{socket}, \texttt{data} \rangle$, where $\texttt{ip} : \texttt{socket}$ is the requesting agent's identifier represented by its IP and the socket it is using and `data` is the data in the request, for example, the contents of a form. The following invariants hold:

- We assume that two different requests in $\mathcal{P}$ (respectively, in $\mathcal{R}$) have necessarily different identifiers;
- $\langle \texttt{ip} : \texttt{socket}, \texttt{data}_2 \rangle \in \mathcal{R}$ iff $\langle \texttt{ip} : \texttt{socket}, \texttt{data}_1 \rangle \in \mathcal{P}$, where $\texttt{data}_1$ and $\texttt{data}_2$ are pieces of data;
- At the beginning of a round, for all $\langle \texttt{ip} : \texttt{socket}, \texttt{data} \rangle \in \mathcal{R}$, $\texttt{data} = 0$, where 0 denotes the empty data.

The first invariant specifies that requests in a buffer corresponds to different agents, while the second invariant specifies that all agents that are sending data during a *round* are being processed by $\mathcal{P}$. The third invariant specifies that at the beginning of a round, the application has not received any further data to be processed. It also assumes an upper-bound, k, on the number of elements in $\mathcal{P}$ and in $\mathcal{R}$. Intuitively, this bound specifies the number of requests an application can process at any given time. For instance in a web-sever, this bound is the number of sockets that it has to process requests.

During a round, SeVen does not reject any incoming requests until k requests has arrived. However, under the assumption that the application can only handle k requests at a time, if the buffer $\mathcal{R}$ has already k requests and yet another request, r_I, arrives, then SeVen needs to make a choice of which requests to keep in its buffers and which to drop:

- It may decide to not keep the incoming request, r_I, which means that the requests currently in the buffers are not affected;
- or it may decide to keep the incoming request, which means that one request in the buffers should be replaced by the new request.

These choices are governed by probability distributions. At the end of a round, SeVen processes the request that remain in $\mathcal{R}$.

3.1. Defense by Example

Instead of describing the algorithm in its complete detail, for which we refer to [Dantas et al. 2014, Dantas 2015], we illustrate its behavior by means of an example. Assume that the upper-bound $k = 4$. Assume that initially there are three requests to be processed, where for simplicity we use natural numbers for data parameter of requests:

$$\mathcal{P}_0 = [\langle \mathtt{ip}_1 : \mathtt{socket}_1, 10\rangle, \langle \mathtt{ip}_2 : \mathtt{socket}_2, 5\rangle, \langle \mathtt{ip}_3 : \mathtt{socket}_3, 2\rangle]$$
$$\mathcal{R}_0 = [\langle \mathtt{ip}_1 : \mathtt{socket}_1, 0\rangle, \langle \mathtt{ip}_2 : \mathtt{socket}_2, 0\rangle, \langle \mathtt{ip}_3 : \mathtt{socket}_3, 0\rangle]$$

This state specifies that the application has received partial data from three requets, $\mathtt{ip}_1 : \mathtt{socket}_1, \mathtt{ip}_2 : \mathtt{socket}_2, \mathtt{ip}_3 : \mathtt{socket}_3$, and it is waiting for more data to be received. It waits for the time, t_S, of a round to collect new incoming requests. Say that it receives the request: $\langle \mathtt{ip}_1 : \mathtt{socket}_1, 5\rangle$ from $\mathtt{ip}_1 : \mathtt{socket}_1$ with 5 pieces of data. The application updates its $\mathcal{R}$ buffer to the following:

$$\mathcal{P}_0 = [\langle \mathtt{ip}_1 : \mathtt{socket}_1, 10\rangle, \langle \mathtt{ip}_2 : \mathtt{socket}_2, 5\rangle, \langle \mathtt{ip}_3 : \mathtt{socket}_3, 2\rangle]$$
$$\mathcal{R}_1 = [\langle \mathtt{ip}_1 : \mathtt{socket}_1, 5\rangle, \langle \mathtt{ip}_2 : \mathtt{socket}_2, 0\rangle, \langle \mathtt{ip}_3 : \mathtt{socket}_3, 0\rangle]$$

Notice that it does not immediately process this new request. It waits for the round to finish. Say now that a new request arrives: $\langle \mathtt{ip}_4 : \mathtt{socket}_4, 7\rangle$ with a fresh identifer $\mathtt{ip}_4 : \mathtt{socket}_4$. Since the number of elements in $\mathcal{P}$ is less than $k = 4$, it can safely add this request to the buffers as follows, where @ is the concatenation operator:

$$\mathcal{P}_1 = \mathcal{P}_0@[\langle \mathtt{ip}_4 : \mathtt{socket}_4, 0\rangle] \quad \mathcal{R}_2 = \mathcal{R}_1@[\langle \mathtt{ip}_4 : \mathtt{socket}_4, 7\rangle]$$

That is, it has received 7 pieces of data from $\mathtt{ip}_4 : \mathtt{socket}_4$ but not processed any piece of information.

Assume now that another request $req_\nu = \langle \mathtt{ip}_5 : \mathtt{socket}_5, 1\rangle$ has arrived. Since the number of elements in $\mathcal{P}$ is equal to $k = 4$, then it must decide whether to process req_ν or not. It first sets $\mathtt{PMod} := \mathtt{PMod} + 1$ and decides to keep req_ν with probability

$$\mathtt{Prob} = \frac{k}{k + \mathtt{PMod}}$$

Notice that the probability of accepting new incomming requests reduces progressively once the buffer contains k elements. For more about this choice we refer to [Khanna et al. 2008, Khanna et al. 2012, Dantas et al. 2014]. Say that after throwing the coin, it accepts to process the request req_ν. It should now decide which one

of the requests in $\mathcal{P}$ and $\mathcal{R}$ it should drop, because it cannot process more than k requests by assumption. In this thesis, such a decision is done by an uniform probability, however SeVen can be instantiate by different probability distributions, as you can see in [Dantas 2015, Dantas et al. 2016, Lemos et al. 2016]. Say that it chooses to drop the request with identification $\texttt{ip}_2 : \texttt{socket}_2$, sending an appropriate message to the requesting agent. The resulting buffers are as follows where $\setminus$ is the operator that removes an element from a list:

$$\mathcal{P}_2 = (\mathcal{P}_1 \setminus \{\langle \texttt{ip}_2 : \texttt{socket}_2, 5\rangle\})@[\langle \texttt{ip}_5 : \texttt{socket}_5, 0\rangle]$$
$$\mathcal{R}_3 = (\mathcal{R}_2 \setminus \{\langle \texttt{ip}_2 : \texttt{socket}_2, 0\rangle\})@[\langle \texttt{ip}_5 : \texttt{socket}_5, 1\rangle]$$

Consider now the end of this round. The application processes the requests that survived in $\mathcal{R}$, *e.g.*, the 5 pieces of data received for request identified $\texttt{ip}_1 : \texttt{socket}_1$ are processed, that is, they are added to the initial 10 pieces. The resulting buffer $\mathcal{P}$ is:

$$\left[\begin{array}{c} \mathcal{P}_3 = \langle \texttt{ip}_1 : \texttt{socket}_1, 15\rangle, \langle \texttt{ip}_3 : \texttt{socket}_3, 2\rangle, \\ \langle \texttt{ip}_4 : \texttt{socket}_4, 7\rangle, \langle \texttt{ip}_5 : \texttt{socket}_5, 1\rangle \end{array}\right]$$

The data component of the requests in $\mathcal{R}$ are all set to 0, as well as `PMod`.

Finally, the application checks which requests have been completed. Say that this is the case for the request identified $\texttt{ip}_1 : \texttt{socket}_1$, then it sends the appropriate web application' data to agent ip_1.

Intuition for the Effectiveness of Selective Strategies The strategy that we just described above is not tailored to mitigate any particular attack. We are simply selecting requests by using traffic rate which incoming requests to select and by random with uniform probability which requests to drop. It might seem puzzling that such a simple strategy works, but we have two rational assumptions for that: 1) the goal of a Low-Rate ADDoS attack is to occupy for long periods of time the workers of the web-server. 2) once the application is at its maximum capacity, it is likely that it is under attack and it is likely that there is larger number of attackers consuming the web-server's resource than legitimate clients. Thus although the requests to be dropped are selected at random, there is a greater chance of selecting a request from an attacker.

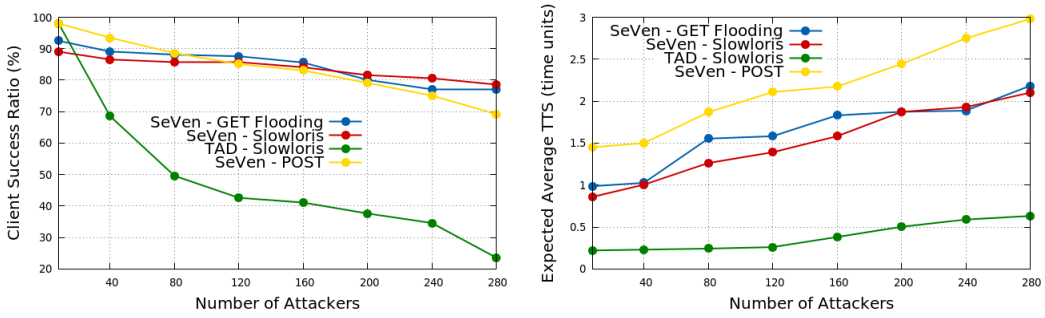
4. SIMULATION AND EXPERIMENTAL RESULTS

This section describes our main simulation and experimental results. [Dantas 2015] contains further details of the Maude formalization and C++ implementation of SeVen.

4.1. Simulation Results

In our simulations, we compare our results with a defense similar to the ones in the literature based on Traffic Analysis. At a glance, this defense keeps track of the request that it received, in particular, the number of pieces of data and the id. For example, if two consecutive Slowloris requests, i.e., "x-1:aa", are received by the application with the same id and the same number of pieces of data, the defense believes that it is an attack and simply blocks subsequent requests from this id. We call this simple defense as Traffic Analysis Defense (TAD). As shown in [de Almeida 2013], TAD is capable of mitigating Slowloris attacks generated by a small number of attackers. However, once there is a great number of attackers, this simple defense mechanism does not have a good performance any longer, as predicted by [Kumar and Selvakumar 2013]. For more details of such a defense, we refer to [de Almeida 2013, Dantas et al. 2014, Dantas 2015].

For our simulations, we used the following scenarios:



(a) Client Success Ratio. (b) Average TTS

Figure 2. Simulations of Client Success Ratio and Average TTS where $k = 12$.

1. SeVen – GET FLOOD: we simulated an HTTP GET flooding attack when the target application used our defense SeVen;
2. SeVen – Slowloris: we simulated an Slowloris attack when the target application used our defense SeVen;
3. SeVen – POST: we simulated an HTTP POST attack when the target application used our defense SeVen;
4. TAD – Slowloris: we simulated an HTTP Slowloris attack when the target application used the defense TAD.

Figure 2 contains the results of the simulations we carried out using different number of attacker that we measure how many (of the 200) clients were successful in receiving an acknowledgment from the target application stating that their request has been completed and how much time in average that it takes for a successful client to receive an acknowledgment that his request was successfully processed by the application. Figure 2(a) depicts the evolution of Client Success Ratio when we increase the number of attackers. For all three attacks, the performance of SeVen is similar. When there are 280 attackers, as opposed to 200 clients, the application running SeVen still can maintain a high level of availability, namely superior to 70%. On the other hand, TAD has similar or even superior results when there is a small number of attackers, but then it falls drastically with the increase of attackers. This is in conformance with the evaluation in [Kumar and Selvakumar 2013] that filtering defenses do not perform well when there is a great number of attackers.

Figure 2(b) depicts the average TTS when we vary the number of attackers. All scenarios involving SeVen have a higher TTS than the scenario using TAD. This is expected, as SeVen only answers request when the round of t_S elapses, while TAD answers immediately. Moreover, as expected, there is an increase on the average TTS. In all scenarios involving SeVen the TTS doubles when we increase the number of attackers from 8 to 280, which seems reasonable. The same happens to the TTS when using TAD. Notice, however, that when using TAD the number of clients that actually receive an acknowledgment reduces considerably with the number of attackers, as depicted in Figure 2(a) and the TTS is only computed using the Successful Clients. So despite TAD having a smaller TTS, the number of clients that have their request completely processed is much less.

4.2. Experimental Results

We carried out a number of experiments on the network to check the robustness of SeVen against Slowloris and HTTP POST attacks. In order to have a realistic DDoS attack, we set both target application and SeVen in Vitória-ES/BR and carried out the attack using

Table 1. Result Summary

	Without SeVen		With SeVen	
	Success Rate	TTS	Success Rate	TTS
No Attack	100.0%	0.01s	100.0%	0.03s
POST	0.0%	∞	97.25%	0.05s
Slowloris	0.0%	∞	94.47%	0.06s

distributed machines from João Pessoa-PB/BR. Moreover, we used a benchmark tool, named Siege [Siege 2014], to generate legitimate client requests also from João Pessoa.

We used an Apache web-server 2.4.7 running in a Ubuntu machine, which was configured with 200 sockets, i.e., the maximum number of connections that the web-server can process. The timeout for the web-server was set to 40 seconds. This means that if a socket remains silent for longer than 40 seconds, then the request arrived at that socket is dropped. We generated (using Siege) 50 clients each 5 seconds, which means in average a rate of 10 clients per second. The goal is to keep 1/4 of the buffer occupied by legitimate clients. For generating the Slowloris and HTTP POST attacks, we used Slowloris [slowloris 2013] and Switchblade [SwitchBlade 2015] tools respectively. We generated 250 attackers every 35 seconds, i.e., a rate of 7.14 attackers per second, which is smaller that our client rate but enough for denying service to a small/medium size web-server. It supports our claim that ADDoS do not need to generate a large amount of traffic to deny a target service. Finally, we set SeVen as a proxy and the round time of SeVen was set to 10 ms.

Table 1 summarizes the results on the Success Rate and TTS for when the web-server is not under an attack, and for the Slowloris and HTTP POST attacks for the scenarios when using SeVen and without SeVen. We observe that the web-server maintains great levels of availability when using SeVen for both attacks. On the other hand, when SeVen is not running the web-server is not able to respond any requests from legitimate clients. In addition, we also measured the slight overhead added by SeVen working as a proxy, which in our experiments was 0.02 seconds.

Our experiments demonstrate that SeVen is indeed effective for mitigating Slowloris and HTTP POST attacks as already showed in our simulations using formal methods. This indicate that formal methods is a suitable tool for specifying defense for mitigating Distributed Denial of Services attacks allowing to increase our confidence on the proposed defense before actual implementation.

5. Conclusions and Future Work

The outcomes of the Thesis We consider the outcome of the whole project a success. At the time this paper was written, we have three published papers [Dantas et al. 2014, Dantas et al. 2016, Lemos et al. 2016], two concluded bachelor theses and two ongoing master theses on the topic. Due to such a success, we also have a solid ongoing project funded by RNP (www.rnp.br) in which we are currently deploying SeVen to protect the web-services of different educational institutions. In addition, we are also investigating how SeVen can be incorporated into existing services such as those used by Fone@RNP.

We propose a novel defense for Application Layer DDoS attacks (ADDoS), called

SeVen. We demonstrated by simulations and experiments that SeVen can be used to mitigate Slowloris and HTTP POST attacks. We already have carried out experiments over the network for the GET Flooding attack with great level of availability, however in order to stick with the thesis results we did not mention it in this paper. We have tested it with other web-servers, e.g., nginx. We are implementing the alternative selective strategies, e.g., using different probability distributions. In order to protect multiple servers, we are incorporating load balancing strategies into SeVen. We are also investigating whether SeVen can be used to mitigate second order DoS attacks [Olivo et al. 2015].

References

[AlTurki and Meseguer 2011] AlTurki, M. and Meseguer, J. (2011). Pvesta: A parallel statistical model checking and quantitative analysis tool. In *CALCO*, pages 386–392.

[Clavel et al. 2007] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Talcott, C. (2007). *All About Maude: A High-Performance Logical Framework*.

[Dantas 2015] Dantas, Y. G. (2015). Estratégias para tratamento de ataques de negação de serviço na camada de aplicação em redes ip – http://tede.biblioteca.ufpb.br/bitstream/tede/7841/2/arquivototal.pdf. Master Thesis in Portuguese.

[Dantas et al. 2016] Dantas, Y. G., Lemos, M. O. O., Fonseca, I., and Nigam, V. (2016). Formal specification and verification of a selective defense for tdos attacks. In *11th International Workshop on Rewriting Logic and its Applications (WRLA)*, LNCS.

[Dantas et al. 2014] Dantas, Y. G., Nigam, V., and Fonseca, I. E. (2014). A selective defense for application layer DDoS attacks. In *IEEE Joint Intelligence and Security Informatics Conference, JISIC 2014*, pages 75–82. IEEE.

[de Almeida 2013] de Almeida, L. C. (2013). Ferramenta computacional para identificação e bloqueio de ataques de negação de serviço em aplicações web. Master Thesis in Portuguese.

[Khanna et al. 2008] Khanna, S., Venkatesh, S. S., Fatemieh, O., Khan, F., and Gunter, C. A. (2008). Adaptive selectiveverification. In *INFOCOM*, pages 529–537.

[Khanna et al. 2012] Khanna, S., Venkatesh, S. S., Fatemieh, O., Khan, F., and Gunter, C. A. (2012). Adaptive selective verification: An efficient adaptive countermeasure to thwart dos attacks. *IEEE/ACM Trans. Netw.*, 20(3):715–728.

[Kumar and Selvakumar 2013] Kumar, P. A. R. and Selvakumar, S. (2013). Detection of distributed denial of service attacks using an ensemble of adaptive and hybrid neuro-fuzzy systems. *Computer Communications*, 36(3):303 – 319.

[Lemos et al. 2016] Lemos, M. O. O., Dantas, Y. G., Nigam, V., and Sampaio, G. (2016). A selective defense for mitigating coordinated call attacks. In (SBRC).

[Olivo et al. 2015] Olivo, O., Dillig, I., and Lin, C. (2015). Detecting and exploiting second order denial-of-service vulnerabilities in web applications. In CCS, 2015.

[r-u-dead yet 2013] r-u-dead yet (2013). https://code.google.com/p/r-u-dead-yet/.

[RFC 1999] RFC, H. (1999). https://tools.ietf.org/html/rfc2616.

[Shankesi et al. 2009] Shankesi, R., AlTurki, M., Sasse, R., Gunter, C. A., and Meseguer, J. (2009). Model-checking DoS amplification for VoIP session initiation. In *ESORICS*, pages 390–405.

[Siege 2014] Siege (2014). https://www.joedog.org/siege-home/.

[slowloris 2013] slowloris (2013). http://ha.ckers.org/slowloris/.

[SwitchBlade 2015] SwitchBlade (2015). http://www.proactiverisk.com/switchblade/.

[Zargar et al. 2013] Zargar, S. T., Joshi, J., and Tipper, D. (2013). A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. *IEEE Communications Surveys and Tutorials*, 15(4):2046–2069.

RIP-ROP: uma proteção contra ataques de execução de código arbitrário baseados em *Return-Oriented Programming*[*]

Mateus F. Tymburibá Ferreira[1], Eduardo Luzeiro Feitosa[2]

[1]Autor – Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

[2]Orientador – Instituto de Computação – Universidade Federal do Amazonas – Manaus, AM – Brasil

mateustymbu@dcc.ufmg.br, efeitosa@icomp.ufam.edu.br

Abstract. *Return-Oriented Programming (ROP) is currently the main technique used by attackers to allow the execution of arbitrary code on vulnerable applications. While protections have been extensively studied, a definitive solution is still not known. This dissertation shows that ROP attacks can be prevented by controlling the frequency of indirect branch instructions executed by applications. Experiments with public exploits confirmed the feasibility of this model. This work also provides: (i) metrics for evaluation and comparison of protections against ROP attacks and (ii) taxonomies to classify those protections.*

Resumo. *Return-Oriented Programming (ROP) é atualmente a principal técnica usada por atacantes para permitir a execução de códigos arbitrários em aplicações vulneráveis. Apesar de proteções terem sido extensamente estudadas, ainda não se conhece uma solução definitiva. Esta dissertação demonstra que, através do controle da frequência de instruções de desvio indireto executadas pelas aplicações, é possível impedir ataques ROP. Experimentos realizados com exploits públicos confirmaram a viabilidade desse modelo. Este trabalho provê também: (i) métricas para avaliação e comparação de proteções contra ataques ROP e (ii) taxonomias para classificação dessas proteções.*

1. Introdução

Dados disponibilizados pelo NIST (*National Institute of Standards and Technology*) apontam para o crescimento do número de vulnerabilidades críticas identificadas em softwares [NIST 2013]. Entre elas, em decorrência do poder que proporcionam ao invasor, as vulnerabilidades que permitem a execução de código arbitrário conquistaram a preferência dos atacantes, conforme estatísticas catalogadas pelo CVE [CVE 2013].

---

* Apesar de atualmente o autor estar vinculado ao Departamento de Ciência da Computação da Universidade Federal de Minas Gerais, esta dissertação foi elaborada enquanto o autor era aluno de mestrado do Instituto de Computação da Universidade Federal do Amazonas. Sua versão completa está disponível em:

http://homepages.dcc.ufmg.br/~mateustymbu/Dissertacao_2014.pdf

1.1. Motivação

A técnica denominada *Return-Oriented Programming* (ROP) tem despertado grande interesse da comunidade científica e da indústria de segurança de sistemas, em função da sua larga utilização em ataques recentes de execução de código arbitrário. Entre muitos exemplos de códigos maliciosos de grande impacto que empregam ROP, pode-se citar os malwares Stuxnet e Duqu [Callas 2011].

Por ter se tornado o principal mecanismo utilizado por atacantes para desenvolver códigos maliciosos, mitigações contra ROP têm sido amplamente estudadas. O Windows 8, por exemplo, agrega um mecanismo de proteção contra ROP. No entanto, poucos dias depois do lançamento da versão preliminar do sistema, pesquisadores demonstraram estratégias relativamente simples capazes de burlar essa defesa [Son 2011]. Outro ponto de concentração de esforços na busca por mecanismos de contenção de ataques tem sido a ferramenta EMET (*Enhanced Mitigation Experience Toolkit*), que inclui uma defesa contra ROP premiada no concurso *BlueHat Prize*. Essa implementação foi superada duas semanas após o seu anúncio [Snake 2012]. De fato, diversas proteções contra ataques ROP já foram propostas, mas ainda não há uma solução efetiva.

1.2. Objetivos

Este trabalho tem dois objetivos: (i) introduzir métricas e taxonomias destinadas à avaliação de proteções contra ataques de execução de código arbitrário baseados em ROP e (ii) apresentar um método inédito de proteção contra esses ataques. A eficácia desse método no bloqueio de ataques reais é demonstrada através de testes com códigos maliciosos disponíveis em repositórios públicos de *exploits*. São analisadas também a ocorrência de falsos positivos e a eficiência computacional do modelo proposto, comparando-o com soluções correlatas.

1.3. Contribuições

Para atingir esses objetivos, as seguintes contribuições foram alcançadas:

- Definição de métricas para avaliação e comparação de proteções contra ataques ROP.
- Elaboração de duas taxonomias para classificação das proteções em função das estratégias de proteção contra ataques ROP e das abordagens de implementação utilizadas em cada solução.
- Demonstração da eficácia do controle da frequência de desvios indiretos como estratégia para detecção de ataques ROP.
- Desenvolvimento de um protótipo de proteção contra ataques ROP destinado a ambientes Windows e Linux em um *framework* de instrumentação binária dinâmica.

A estratégia de proteção baseada no controle da frequência de desvios indiretos, idealizada e validada neste trabalho, é suportada por elementos suficientes para sustentar a crença de que ela irá impactar na criação de melhores proteções contra

ataques ROP. Além disso, o protótipo desenvolvido, denominado RIP-ROP[1], viabiliza a proteção imediata de aplicações críticas executadas nos ambientes mencionados.

2. Return-Oriented Programming

Os ataques ROP foram criados com o intuito de superar a proteção oferecida pelo bit de execução (NX/XD), que impede a execução de instruções oriundas de áreas destinadas a dados, como a pilha de um processo [Shacham 2007]. *Exploits* ROP são baseados na capacidade de um atacante controlar, ao mesmo tempo, a pilha de execução de um programa e o conteúdo armazenado no registrador que indica o endereço da próxima instrução a ser executada. A partir dessas premissas, o atacante constrói uma entrada maliciosa que força o encadeamento de pequenos trechos de código da própria aplicação, conhecidos como *gadgets*.

Por limitações de espaço, este artigo não detalha a técnica ROP. No entanto, instruções detalhadas sobre o minucioso processo de construção de um *exploit* ROP podem ser encontradas na Seção 2.3 da dissertação e em um capítulo de livro publicado pelo autor deste trabalho no SBSeg 2012 (vide Seção 7.2). Esse capítulo de livro apresenta o conteúdo de um minicurso ministrado no evento e reflete parte dos ensinamentos colhidos durante a preparação para a elaboração da dissertação resumida neste artigo.

3. Trabalhos Relacionados

As estratégias de proteção contra ataques ROP são discutidas e classificadas segundo uma nova taxonomia (Seção 3.2). Além de classificar as principais estratégias, também introduzimos uma proposta de classificação segundo a abordagem utilizada pelas soluções para implementar as estratégias escolhidas (Seção 3.3). Essas inéditas taxonomias, propostas nesta dissertação, agrupam as proteções de acordo com características semelhantes, o que facilita a compreensão dos fatores que influenciam a qualidade dos esquemas de proteção.

3.1. Métricas de avaliação

As métricas propostas foram utilizadas durante a análise das características, virtudes e deficiências inerentes a cada um dos trabalhos avaliados. Da mesma forma, acredita-se que elas constituem valiosa ferramenta no processo de análise dos benefícios e limitações inerentes a novas proteções contra ataques ROP. As seguintes métricas foram consideradas relevantes:

- Tipos de ataques bloqueados: indica os tipos de *gadgets*[2] detectados pela proteção. Em ataques ROP, *gadgets* podem ser de três tipos: finalizados por instruções “Ret”, “Jump” indireto ou “Call” indireto [Checkoway et al. 2010].
- *Overhead* médio: indica a perda de desempenho que uma proteção impõem.
- Exceções: indica as situações em que códigos autênticos são classificados equivocadamente (falsos positivos).

1 http://homepages.dcc.ufmg.br/~mateustymbu/RIPROP.zip

2 *Gadgets*: pequenos segmentos de código finalizados por uma instrução de desvio indireto.

- Viabilidade prática do protótipo: indica se existe implementação da solução para proteção imediata de aplicações reais, em ambientes de produção.

3.2. Estratégias de Proteção

Observou-se que as proteções contra ROP têm utilizado estratégias semelhantes de detecção do ataque. Em função disso, propôs-se o agrupamento das soluções em classes, a fim de facilitar a compreensão de aspectos comuns às proteções pertencentes a um mesmo segmento. A taxonomia sugerida é composta pelos seguintes elementos: randomização, construção de uma pilha-sombra, checagem da instrução anterior ao endereço de retorno, checagem dos endereços autênticos para desvios, checagem da posição de entrada nas funções, controle da frequência de instruções de retorno e checagem do apontador para o topo da pilha. Em função do espaço limitado para resumir todo o conteúdo da dissertação, essas estratégias não serão detalhadas neste artigo. Todavia, cada uma dessas classes, bem como as proteções pertencentes a elas, são destrinchadas na Seção 3.2 da dissertação.

3.3. Abordagens de Implementação

As diversas estratégias de proteção, elencadas na Seção anterior, podem ser aplicadas aos sistemas que se pretende defender através de várias abordagens. Essas abordagens de implementação impactam diretamente na qualidade da solução proposta e apresentam características peculiares. Diante dessa percepção, foi proposta a seguinte taxonomia para classificação das proteções segundo as abordagens escolhidas para aplicar as medidas de prevenção contra ataques ROP: compilação, instrumentação binária estática, instrumentação binária dinâmica, adaptação do hardware, virtualização e emulação de código. Novamente, por falta de espaço, essas classes não são pormenorizadas neste artigo. Contudo, cada uma dessas abordagens é detalhada na Seção 3.3 da dissertação.

3.4. Discussão

A Tabela 3.1 da dissertação exibe um resumo da análise comparativa de 27 proteções contra ataques ROP à luz das métricas propostas e das taxonomias sugeridas. Dessa tabela, que não será reproduzida neste artigo por premência de espaço, depreende-se que os trabalhos que empregam a abordagem de Instrumentação Binária Dinâmica acarretam *overheads* maiores do que os trabalhos que utilizam as demais abordagens. Em contrapartida, as soluções que empregam Instrumentação Binária Estática e Adaptações do Hardware tendem a apresentar um *overhead* menor. Outra observação interessante reside no fato de que apenas 5 soluções parecem não apresentar problemas de compatibilidade com algum tipo de aplicação.

Tendo em vista que ainda não há uma proteção eficaz contra ataques ROP, espera-se que esse assunto continue atraindo a atenção de pesquisadores em busca de uma solução que alie eficácia no bloqueio de todas as variantes de ataques ROP, reduzido *overhead* computacional e baixo índice de ocorrência de exceções. A análise das proteções elaboradas até o presente momento permite, inclusive, que se formule afirmações mais gerais quanto ao futuro das proteções contra ataques ROP. Todas as soluções baseadas em verificações efetuadas em pontos específicos da execução de uma

aplicação, como as ocasiões em que chamadas de sistema são efetuadas, podem eventualmente ser superadas por estratégias que, após preparar o ataque, forcem a execução de códigos sem efeito simplesmente para iludir as checagens futuras. Por isso, acredita-se que soluções efetivas contra o ROP deverão checar o status do fluxo de execução durante toda a execução do processo.

4. Controle da frequência de desvios indiretos

Normalmente, as sequências de instruções que compõem cada *gadget* usado em um ataque ROP são extremamente curtas, dificilmente contendo mais do que cinco instruções. Essa é uma característica inerente aos ataques ROP, porque quanto maior a sequência de instruções, maior a probabilidade de existir entre essas instruções uma operação que altere o status da memória ou de um registrador de forma a comprometer o ataque. Essa alteração de status é comumente chamada por atacantes de "efeito colateral" de um *gadget*. A fim de evitar esses efeitos colaterais, quase sempre os *gadgets* escolhidos pelos atacantes são extremamente curtos.

Para evitar os mencionados "efeitos colaterais", ataques ROP apresentam uma elevada concentração de instruções de desvio indireto em um curto espaço de tempo. Diante dessa constatação, a estratégia de proteção proposta neste trabalho consiste em checar se a contagem do número de instruções de desvio indireto em uma determinada "janela de instruções" é maior do que um determinado limiar. Ao invés de medir a frequência apenas das instruções de retorno, esse novo esquema supervisiona a frequência de qualquer tipo de desvio indireto, incluindo aqueles efetuados através de instruções CALL indireto ou JMP indireto. Dessa forma, é possível evitar os três tipos de ataques ROP. Conforme pode ser observado na Tabela 3.1 da dissertação, poucas proteções oferecem esse benefício.

A fim de comprovar a eficácia dessa estratégia, elaborou-se uma metodologia que contemplou a análise do comportamento dos desvios indiretos tanto em aplicações normais quanto em ataques ROP. Através dessa avaliação, foi possível confirmar que a diferença de comportamento entre essas duas classes é suficiente para ratificar a hipótese de que o controle da frequência de desvios indiretos é uma estratégia eficaz na detecção de ataques ROP (Seção 5.3 da dissertação). A análise da frequência de desvios indiretos foi dividida em duas etapas:

- Avaliação teórica (Seção 4.1.2 da dissertação): foram estudados os cenários em que o comportamento de aplicações autênticas mais se aproxima do padrão apresentado por ataques ROP.
- Avaliação empírica (Seção 4.1.3 da dissertação): foram registrados os valores de frequência máxima de desvios indiretos observados durante a execução dos *benchmarks* que compõem a suíte SPEC CPU2006, comparando-os com as frequências máximas de desvios indiretos observadas durante a execução de 15 *exploits* ROP reais. Os valores ideais do tamanho da janela de instruções, bem como do limiar, também foram definidos através desses experimentos.

Além de demonstrar a eficácia da proteção no bloqueio a ataques ROP, foram executados procedimentos para a avaliação do impacto da solução no tempo de execução das aplicações protegidas. Para isso, comparou-se o desempenho dos

benchmarks que compõem o SPEC CPU2006 com o tempo de CPU observado quando esses mesmos aplicativos são protegidos pelo RIP-ROP e por soluções correlatas. Os detalhes de implementação e as otimizações agregadas ao protótipo estão descritas na Seção 4.2 da dissertação e não serão apresentados neste artigo por restrição de espaço.

5. Resultados

Ao analisar os resultados dos experimentos realizados (Capítulo 5 da dissertação), constata-se que a densidade máxima de instruções de desvio indireto é maior nos *exploits* ROP do que em aplicações autênticas. Ao mesmo tempo, não foi identificado qualquer caso de falso positivo entre todos os *benchmarks* e ambientes de experimentação. Além disso, o RIP-ROP foi testado com sucesso na proteção de 15 (quinze) aplicações para as quais existem *exploits* ROP publicamente disponíveis.

A Tabela 1 exibe uma comparação do RIP-ROP com outras proteções que utilizam instrumentadores binários dinâmicos para implementar soluções baseadas no controle da frequência de instruções. Nessa tabela, estão expressos os *overheads* reportados pelos autores de cada solução, que remontam a diferentes conjuntos de teste e ambientes de experimentação. Apesar disso, pode-se dizer que o custo computacional apresentado pelo RIP-ROP (727%) é comparável ao do DROP [Chen et al. 2009] (530%), que utiliza o mesmo *framework* de instrumentação binária, mas mediu o desempenho através da execução de uma seleção de aplicações, ao invés da suíte de *benchmarks* SPEC. As únicas aplicações utilizadas tanto nos testes executados com o DROP quanto nos experimentos realizados com o RIP-ROP (bzip2 e gcc), que podem oferecer uma comparação mais realista, indicam que o protótipo desenvolvido neste trabalho impõe um *overhead* menor. Nos experimentos com o bzip2, o DROP acarretou em um custo computacional de 1.540%, consideravelmente superior aos 809% registrados pelo RIP-ROP. Nos testes com o gcc, o DROP impôs um *overhead* de 960%, enquanto o RIP-ROP elevou o tempo de CPU em 729%.

Tabela 1: Proteções que controlam a frequência de instruções de retorno via instrumentação binária dinâmica

Proteção	***Abordagens***	***Estratégias***	***Ataques bloqueados***	***Overhead (%)***	***Exceções***	***Viabilidade prática***
DynIMA [Davi et al. 2009]	2 e 3	6	R	*	4	Não
DROP [Chen et al. 2009]	3	6	R	530,0	1	Sim
RIP-ROP	**3**	**8**	**R, J e C**	**727**	**0**	**Sim**

Abordagens: 2-Instrumentação Binária Estática ; 3-Instrumentação Binária Dinâmica.
Estratégias: 6-Controle da frequência de instruções de retorno ; 8-Outras.
Ataques bloqueados: R-encadeamento via RET; J-encadeamento via JMP; C-encadeamento via CALL.
*O *overhead* não é informado na publicação.

Outro fator de comparação entre as proteções, ilustrado na Tabela 1, recai sobre os tipos de ataques ROP bloqueados. Nesse caso, apenas o RIP-ROP oferece uma proteção contra todos os tipos de *exploits* ROP. Essa capacidade está diretamente relacionada à mudança na estratégia de detecção dos ataques adotada neste projeto, que amplia o escopo de monitoramento para abarcar todas as instruções de desvio indireto.

6. Conclusão

Levando-se em conta a grande quantidade de proteções contra ataques ROP publicadas

e a complexidade do processo de comparação dessas soluções, este trabalho contribui com a pesquisa nessa área ao apresentar uma visão geral dos trabalhos, estruturar as soluções em duas classificações e propor métricas para a análise e comparação de proteções. Além disso, este trabalho demonstrou que a imposição de um limite para o uso de instruções de desvio indireto acarreta em severas limitações à capacidade de criação de um *exploit* ROP efetivo, impossibilitando-a em todos os casos testados. Finalmente, também foi desenvolvido neste trabalho um protótipo que pode ser facilmente adotado em ambientes de produção. A análise de desempenho desse protótipo indicou que a proteção é obtida a um custo computacional comparável à performance de soluções correlatas, superando-as em alguns casos.

7. Produção científica

Esta Seção relaciona a produção científica resultante da dissertação.

7.1. Trabalhos premiados

- Tymburibá, Mateus, Moreira, Rubens e Pereira, Fernando (2016). Inference of peak density of indirect branches to detect ROP attacks. *Proceedings of the ACM International Symposium on Code Generation and Optimization*, p. 150-159. http://dl.acm.org/citation.cfm?id=2854049

Esse artigo apresenta uma extensão do trabalho desenvolvido na dissertação. Ele descreve um algoritmo capaz de inferir o limiar máximo de frequência de instruções de desvio indireto previsto para uma aplicação à partir do seu código-fonte. Recebeu o prêmio de melhor trabalho de pesquisa de estudante (*ACM Student Research Competition*) em um dos principais eventos internacionais para divulgação de resultados de pesquisas ligadas à análise e otimização de códigos (CGO 2016), evento com classificação A2 no sistema Qualis da CAPES.

- Tymburibá Ferreira, M., Filho, A. e Feitosa, E. (2014). Controlando a Frequência de Desvios Indiretos para Bloquear Ataques ROP. *Anais do XIV SBSeg*, p. 223–236. http://www.lbd.dcc.ufmg.br/colecoes/sbseg/2014/0017.pdf

Esse artigo discute parte dos resultados apresentados na dissertação. Recebeu menção honrosa para melhor artigo no SBSeg 2014, evento com classificação B4 no sistema Qualis da CAPES.

7.2. Outros trabalhos

- Emilio, R., Tymburibá, M. ; Pereira, F. (2015). Inferência Estática da Frequência Máxima de Instruções de Retorno para Detecção de Ataques ROP. *Anais do XV SBSeg*, p. 2–15. http://sbseg2015.univali.br/anais/SBSegCompletos/artigoCompleto01.pdf

Esse artigo apresenta uma versão preliminar do algoritmo publicado em CGO 2016. Foi publicado no SBSeg 2015, evento com classificação B4 no sistema Qualis da CAPES.

- Tymburibá, M., Moreira, R. e Pereira, F. (2015). RipRop: A Dynamic Detector of ROP Attacks. *Anais da Sessão de Ferramentas do Congresso Brasileiro de Software*. p. 9–16. http://cbsoft.org/articles/0000/0529/Ferramentas.pdf

Esse artigo descreve as otimizações e os detalhes de implementação do RIP-ROP. Foi publicado no CBSoft 2015, congresso sem classificação no sistema Qualis da CAPES que agrega 4 eventos tradicionais: SBES (Qualis B2), SBLP (Qualis B3), SBMF (Qualis B4) e SBCARS (Qualis B4).

- Tymburibá, M. (2015). Counting the Frequency of Indirect Branches to Detect Return-Oriented Programming Attacks. *Student Forum Supplementary Volume of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).* http://homepages.dcc.ufmg.br/~mateustymbu/DSN-2015.pdf

Esse artigo descreve uma solução em hardware baseada na estratégia de proteção desenvolvida nesta dissertação. Foi apresentado no fórum de estudantes da 45ª *IEEE/IFIP International Conference on Dependable Systems and Networks* (DSN), uma das conferências internacionais de maior prestígio para apresentar trabalhos na área de segurança da computação, com classificação A1 no sistema Qualis da CAPES.

- Tymburibá Ferreira, M., Rocha, T. de S., Martins, G. B., Feitosa, E. e Souto, E. (2012). Análise de vulnerabilidades em Sistemas Computacionais Modernos: Conceitos, Exploits e Proteções. *Minicursos do XII SBSeg*. p. 2–51. http://sbseg2012.ppgia.pucpr.br/@docs/SBSeg2012Minicursos.pdf

Esse capítulo de livro descreve as principais técnicas modernas de desenvolvimento de *exploits*, incluindo ROP. Foi publicado e apresentado no formato de minicurso no SBSeg 2012, evento com classificação B4 no sistema Qualis da CAPES.

Referências

Callas, J. (2011). Smelling a RAT on Duqu. http://blogs.entrust.com/enterprise-authentication/?p=236.

Checkoway, S., Davi, L., Dmitrienko, A., et al. (2010). Return-oriented programming without returns. *CCS*, p. 559–572.

Chen, P., Xiao, H., Shen, X., et al. (15 nov 2009). DROP: Detecting Return-Oriented Programming Malicious Code. *ICISS*, Lecture Notes in Computer Science. v. 5905, p. 163–177.

CVE (2013). Vulnerability distribution of cve security vulnerabilities by types. http://www.cvedetails.com/vulnerabilities-by-types.php.

Davi, L., Sadeghi, A. e Winandy, M. (2009). Dynamic integrity measurement and attestation: towards defense against return-oriented programming attacks. In *Proceedings of the 2009 ACM workshop on Scalable trusted computing*.

NIST (2013). National Vulnerability Database (NVD) CVE Statistics. https://web.nvd.nist.gov/view/vuln/statistics-results?adv_search=true&cves=on&cwe_id=CWE-119.

Shacham, H. (2007). The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). *CCS*,

Snake (2012). Bypassing EMET 3.5′s ROP Mitigations. https://repret.wordpress.com/2012/08/08/bypassing-emet-3-5s-rop-mitigations/.

Son, N. H. (2011). ROP chain for Windows 8. http://blog.bkav.com/en/rop-chain-for-windows-8/.

Um IdM e Método de Autenticação baseado em chaves para prover autenticação única em Internet das Coisas

MSc. Adriano Witkovski, Prof. Dr. Altair O. Santin (Orientador)

Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná de Informática – PUCPR
Curitiba – PR – Brasil

wadriano@gmail.com, santin@ppgia.pucpr.br

Abstract. *Abstract. The Internet of Things (IoT) brings significant challenges to authentication schemes in a scenario with several appliances in a smart house. An Identity Management (IdM) can be applied to easily authenticate a technician that intends to access the appliances from the Internet. However, the Internet context is different from the IoT, demanding context adaptation. Therefore, integrating these contexts to allow the authentication on the Internet and provide Single Sign-On (SSO) in IoT is a challenge. The goal is to allow a technician to access an appliance that is not directly reachable from the Internet, using IdM, and without creating a single point of vulnerabilities in the gateway. The proposal works with two key-based schemes, one for Internet and the other for IoT. A proof-of-concept implementation shows the proposal is feasible and does not present significant overhead for messages with up to 4096 bytes and with 50 appliances.*

Resumo. *Resumo. A IoT (Internet das Coisas) traz desafios significativos para esquemas de autenticação em cenários com múltiplos appliances em uma Smart House. O Gerenciamento de Identidades (IdM) pode ser aplicado para autenticar um técnico que pretende acessar os appliances a partir da Internet. O contexto Internet é diferente da IoT, exigindo adaptação do contexto. Assim, integrar estes contextos para permitir a autenticação na Internet e fornecer Single Sign-On (SSO) em IoT é um desafio. O objetivo é permitir que um técnico possa acessar um appliance, que não é acessível diretamente a partir da Internet, usando um IdM e sem que criar o ponto de vulnerabilidades no gateway. A proposta interage com dois esquemas baseados em chave, um para a Internet e outro para a IoT. O protótipo mostra que a proposta é viável e não apresenta um overhead significativo para mensagens com tamanho máximo de 4096 bytes e com 50 appliances.*

1. Introdução

O principal objetivo da IoT é interligar diversas "coisas" ou *appliances*, provendo conexão à Internet para fornecer serviços como monitoramento, gerenciamento e automação [Atzori, Iera and Morabito 2010]. No contexto de IoT os *appliances* geralmente possuem restrições de recursos computacionais. Por esse motivo, inserir mecanismos de segurança com recursos limitados pode representar um grande desafio [Babar, Mahalle, Stango, Prasad 2010]. Os *appliances* podem ser passíveis de problemas de segurança como confidencialidade, autenticidade e integridade dos dados.

Assim, alguns requisitos de segurança necessitam ser garantidos como comunicação segura dos dados, acesso seguro à rede e Gestão de Identidades [Babar, Mahalle, Stango, Prasad 2010].

Um cenário da IoT é a *Smart House* que contém diversos *appliances* contectados à Internet (e.g. eletrodomésticos). A empresa fabricante dos *appliances* acessa remotamente os dispositivos para efetuar manutenção, atualização de *firmware*, etc. Por questões de segurança é importante que os *appliances* localizados dentro da *Smart House* não estejam expostos na Internet. Assim, gerir a identidade dos técnicos do fabricante é imprescindível para manter gestão de acesso aos *appliances*. Integrar um IdM da Internet com a IoT não é trivial, devido à restrição de recursos dos *appliances* e também pela ausência de um canal de comunicação segura entre os dois contextos.

Abordagens apresentadas para autenticação e autorização de acesso em IoT utilizam chave única em todos os *appliances* [Babar, Mahalle, Stango, Prasad 2010 e Miorandi, Sicari, De Pellegrini, Chlamtac 2012]. A utilização de uma chave única tem como vantagem a redução de recursos, porém, torna-se inviável caso a chave seja descoberta. Outra desvantagem é a falta de controle do usuário está acessando um determinado *appliance*, além da dificuldade de manutenção da chave em todos os *appliances* [Miorandi, Sicari, De Pellegrini e Chlamtac 2012]. A utilização de um serviço de autenticação possibilita a autorização de acesso que resolve as deficiências da chave única. As abordagens se limitam à autenticação e autorização de acesso localizado na Internet ou exclusivamente na IoT. Outras propostas buscam integrar os dois contextos, mas não apresentam implementação e não consideram um canal de comunicação seguro fim-a-fim entre os dois contextos [Van Thuan, Butkus 2014 e Fremantle, Aziz, Kopecky, Scott 2014 e Leo, Battisti, Carli e Neri 2014]. Nenhuma proposta aborda a autenticação única utilizando SSO (*Single Sign-On*).

O trabalho está estruturado da seguinte forma. A seção 2 apresenta a motivação e contribuição do trabalho. A seção 3 apresenta a proposta. A seção 4 apresenta o protótipo desenvolvido e os resultados obtidos. E por fim a seção 5 traz as conclusões do trabalho.

2. Motivação e Contribuição

Ao acessar remotamente os dispositivos o fabricante dos *appliances* pode monitorar e coletar informações para prover serviços. Uma máquina de lavar conectada à Internet pode submeter os dados de rotação e determinados parâmetros para um servidor na nuvem computacional do fabricante, por exemplo. O fabricante pode fornecer um serviço pró-ativo de substituição de peças. Apesar dos *appliances* da *Smart House* serem endereçáveis e, portanto, estarem acessíveis a partir da Internet, não devem estar expostos para acesso direto, pois vulnerabilidades podem exploradas ou injetadas nos mesmo e toda a segurança da casa pode ser comprometida.

Segundo [Chadwick 2009] Gestão de Identidade, comunicação segura, e resistência à violação de dispositivos são requisitos no contexto de segurança que devem ser observados na IoT. Para que o técnico do fabricante possa acessar os *appliances* é importante que exista autenticação para o acesso. Um dos desafios é integrar um IdM da Internet com os dispositivos de baixa capacidade de processamento, armazenamento e memória com a pilha de protocolos de IoT, além de fornecer

comunicação segura fim-a-fim. Os dispositivos da IoT geralmente não suportam os protocolos tradicionais como HTTP (*Hypertext Transfer Protocol*) e SSL (*Secure Socket Layer*). Por esta razão foram desenvolvidos o CoAP (*Constrained Application Protocol*) e DTLS (*Datagram Transport Layer Security*) para atender os requisitos da IoT [Shelby, Hartke, Bormann e Rescorla, Modadugu]. O *gateway* necessita de um *parser* para que o protocolo HTTP no contexto da Internet e o CoAP no contexto da IoT possam se comunicar. Na literatura, a maioria das propostas considera apenas o contexto da IoT, outros trabalhos assumem que o *gateway* é um elemento confiável. O recurso de SSO para IoT é praticamente inexplorado na literatura.

A principal contribuição deste trabalho é que o esquema proposto é capaz de operar um IdM de forma integrada entre o contexto de Internet e IoT. A comunicação fim-a-fim, fornece proteção por mensagem (*per-message*) no trânsito entre o *appliance* e o *gateway*, e entre este e o site do fabricante, sem que o *gateway* se torne um elemento vulnerável. O uso do SSO, provendo autenticação única para que o técnico do fabricante se autentique uma única vez por dia, por exemplo, e possa acessar múltiplos *appliances*, facilitando a administração.

3. Proposta

A proposta envolve seis componentes, conforme apresentado na Figura 1: *Appliance*, *Customer Service*, *Gateway*, *Appliance Technician*, *Authentication Server* e *Access Authorization Server*. O *Appliance* (App) é uma "coisa" utilizado na IoT com um atributo identificador e uma chave simétrica fornecida em sua fabricação. O número de série e a chave simétrica estão armazenados também no *Customer Service* (CS). O CS é um serviço fornecido pelo fabricante do App e que faz a comunicação entre o App e o técnico da empresa. O *Gateway* (GW) é o elemento responsável pela intermediação da comunicação entre a Internet e os App da *Smart House*.

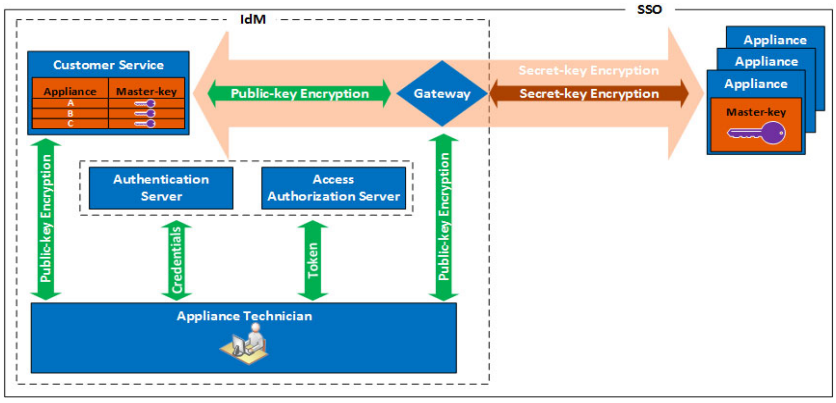


Figura 1 - Esquema de autenticação baseado em chaves para SSO em IoT.

O *Appliance Technician* (AppTec) é um sistema operado por um técnico do fabricante para responder às exigências dos consumidores e as necessidades dos Apps no período de pós-venda. O *Authorization Server* (AS) e *Access Authorization Server* (ASS) fazem parte de um IdP (*Identity Provider*). O AS fornece um serviço de autenticação capaz de validar as credenciais do AppTec e também para o fornecer o recurso de SSO. O AAS é um serviço de autorização de acesso que fornece *tokens* para que o AppTec já autenticado possa acessar o CS e também vários GWs. O GW é

acessado de forma segura, caso alguém tente acessá-lo e não esteja autenticado e autorizado, visto que as entidades devem estar previamente registadas no AAS para acessar o GW. No primeiro nível, CS e App utilizaram criptografia de chave simétrica baseada em uma chave mestre, fornecida pelo fabricante durante a fabricação do *appliance*, denominada KKM (*Master Key Encrypting Key* – ANSI X.9.17). Esta chave é utilizada para distribuir posteriormente a chave de sessão KEK (*Key Encryption Key*). No segundo nível, CS, AppTec, GW, AS e AAS utilizam a criptografia de chave pública. O GW e App utilizam a criptografia de chave secreta para proteger os dados transmitidos, incluindo KEK, por mensagem. Dessa forma, CS e App compartilham a chave mestre KKM que é armazenada manualmente no CS e no firmware do App. A chave KKM está relacionada com o número de série do App registrada no sistema do CS. As mensagens trocadas entre CS e App são criptografadas por KKM, não permitindo qualquer acesso intermediário ao conteúdo da mensagem. A mesma proteção é obtida quando a chave KEK criptografa os dados da mensagem.

3.1 Fluxo de Mensagens

A Figura 2 apresenta a comunicação iniciada por um App. Um sujeito solicita para o App um serviço prestado pelo CS (evento 1) com conteúdo de uma requisição (requestValue). O App gera e utiliza uma chave de sessão KEK para criptografar o conteúdo das mensagens trocadas durante a vida útil da requisição. O App usa a chave mestre KKM para criptografar a chave KEK e um *nonce* (e.g. *timestamp*). O App envia o valor criptografado (*encryptedValue*) ao GW com o número de série (*serialNumber*) e o endereço do CS (*costumerURL*). O GW realiza o *parser* da mensagem da IoT para Internet e encaminha a mensagem para o CS (evento 1.1.1).

O CS recupera a chave KKM do App através do número de série e decodifica os dados (encryptedValue). O CS valida a KEK do App pelo *nonce,* cujo objetivo é evitar o ataque de *replay*. O CS armazena a KEK do App e a utiliza para cifragem e decifragem de mensagens futuras do App na mesma sessão. O CS responde ao App o índice (ID) da chave de sessão e um valor do *nonce* incrementado de 1 (*nonce* + 1). O *nonce* é usado para garantir a autenticidade do CS, garantindo que apenas o titular da KKM pode a decifrar e recifrar uma mensagem. O GW recebe a resposta e mapeia o índice da KEK e o endereço do App. O GW traduz a mensagem de Internet para contexto da IoT e encaminha o App. O App recebe uma mensagem cifrada com KEK, decifra a mensagem e valida o *nonce.* O App cifra a requisição com um novo *nonce* e encaminha para o GW (caso 1.2), fornecendo o índice (ID) da chave de sessão que será utilizada para se comunicar com CS. O CS recupera a chave de sessão com base no ID, decifra e armazena a requisição para ser respondida posteriormente pelo técnico. O CS retorna o índice da chave de sessão e o *nonce* incrementado de 1. O App recebe a requisição, valida o *nonce*, e informa ao sujeito o status da mesma.

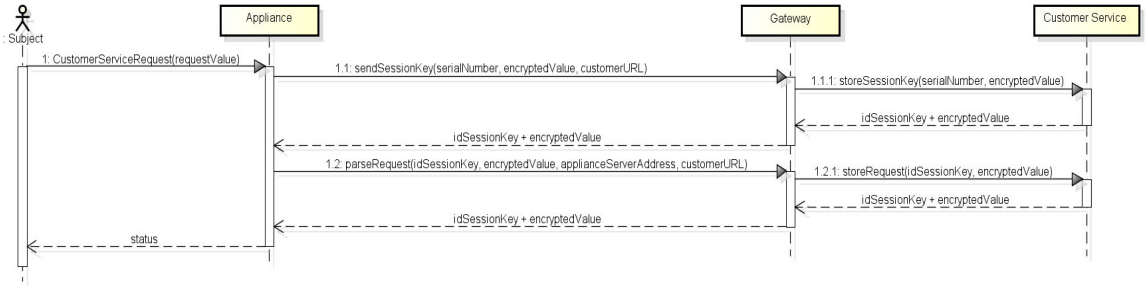


Figura 2 - Fluxo de mensagens para uma requisição iniciado pelo Appliance.

A Figura 3 mostra o processo de autenticação e autorização de acesso de um técnico do fabricante para acessar o AppTec. O técnico requisita acesso ao AppTec (evento 1) sendo redirecionado para o AS com suas credenciais solicitadas (evento 2). O AS valida as credenciais do técnico e responde com um código (com validade de tempo) a ser utilizado para solicitar um *token* de acesso ao AAS (evento 3.1). O AAS devolve um *token* para ser utilizado pelo técnico para atender a uma requisição do App.

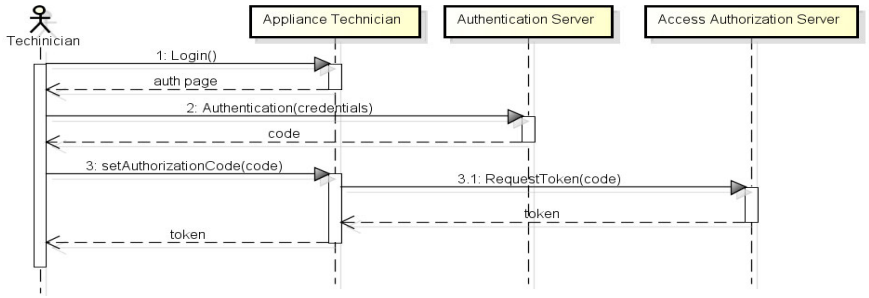


Figura 3 - Autenticação e autorização de acesso para o técnico da fabricante

O AppTec recupera a KEK através de seu índice (ID) e também possui a capacidade de decifrar a requisição, processar e encaminhar a resposta cifrada ao GW. O GW recebe a resposta cifrada juntamente com um *token* de acesso que permite ao GW fazer o *parser* e retransmitir a mensagem criptografada para o App. O App recebe a resposta cifrada com a KEK, usando o índice, decifra a resposta e processa a requisição.

O segundo tipo de comunicação iniciada por um técnico é para coletar informações ou realizar uma tarefa de manutenção, por exemplo, uma atualização de *firmware* no App, então é executado o procedimento de *Call Back* (Figura 4).

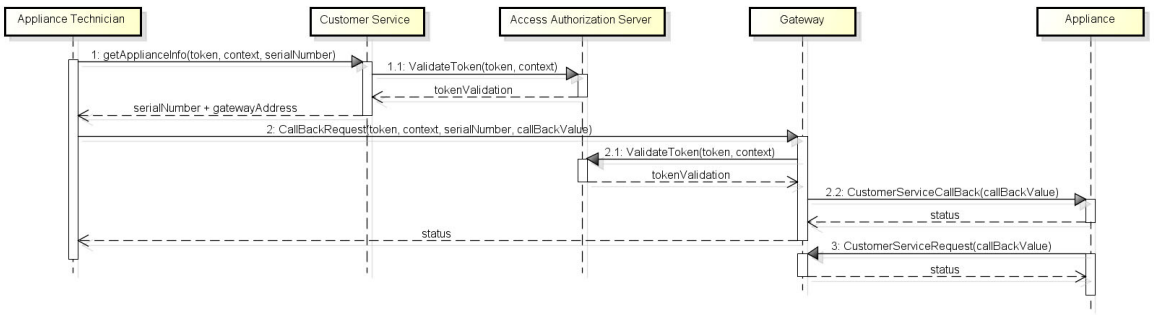


Figura 4 - Diagrama de sequência do procedimento de *Call Back*

4. Protótipo

O protótipo utiliza padrões de tecnologias conhecidas e bibliotecas de código aberto. O *Manufactor Domain* consiste em dois componentes, AppTec e CS. O AppTec foi implementado usando o framework Vaadin. O CS foi implementado como um *Web Service RESTful* usando a API JAX-RS. O *Customer Domain* consiste em um GW e vários Apps. As interfaces com o GW que representam o contexto Internet foram implantadas usando Java em um servidor HTTP. A interface da IoT foi construída num servidor CoAP usando a biblioteca Californium, que também foi usada para o App. Para cifragem foi utilizado o algoritmo AES de 128 bits, a fim de obter as chaves KKM e KEK, usando o Scandium, um subprojeto do Californium. O Scandium com suporte ao DTLS na versão 1.2 no contexto da IoT. O servidor de autenticação (AS) foi implementado seguindo a especificação do *OpenID* 2.0 usando a biblioteca Nimbus. O Nimbus fornece um IdM para AppTec, CS e GW, e também assegura que somente usuários autenticados e autorizados acessam os App. A autorização de acesso (AAS) foi implementada seguindo especificação do protocolo OAuth 2.0 e Nimbus, a fim de emitir *tokens* de acesso para o AppTec autenticado.

A pilha de protocolos usada na comunicação entre os contextos de Internet e IoT, feita usando HTTPS e CoAPs, é mostrada na Figura 5.

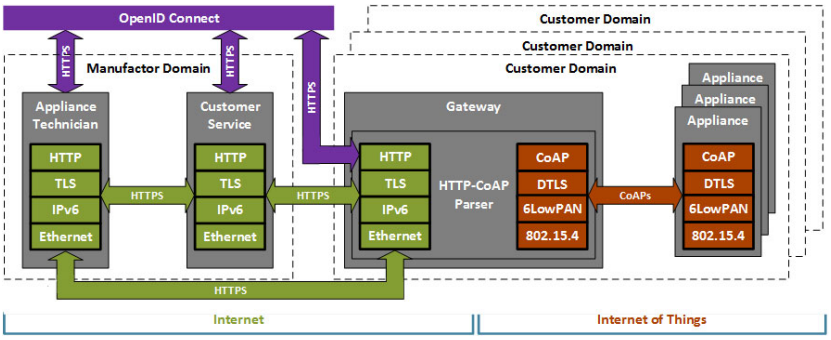


Figura 5 - Arquitetura do protótipo

4.1. Resultados

Os testes mediram o impacto do tamanho da mensagem na requisição e o tempo de resposta com e sem cifragem no ambiente de IoT. Também foi medido o tempo de resposta com e sem ativação do recurso de SSO com 50 Apps. Foi feito teste de stress para avaliar a quantidade de Apps suportados pelo protótipo. Na parte IoT utilizou-se Traffic Management (CBQ) para reduzir a banda de rede e imitar as limitações de 6LoWPAN.

A Figura 6 mostra que para requisições de tamanho de 32 a 1024 bytes, o tempo médio de resposta fica abaixo de 900 ms por requisição e o *overhead* pelo uso do CoAPs permanece abaixo de 200 ms. É possível observar um aumento de *overhead* para requisições com mensagens maiores de 1024 bytes, atingindo um *overhead* de 350 ms, para mensagens de 2048 bytes e 10 Apps. O tempo de resposta permanece abaixo de 1250 ms nas demais requisições, e as requisições CoAPs estão abaixo de 200 ms com 4096 bytes, adequado para o número Apps. Esta observação indica que a proposta funciona bem, com quase 15% de *overhead*, mesmo para 50 Apps e com tamanho de mensagens de 4096 bytes. Com o resultado do teste é possível concluir que o GW tem

um bom desempenho quando se leva em conta um número realista de Apps, no contexto de uma *Smart House*. Além disso, os resultados mostram a integração adequada entre os dois contextos, sem impacto significativo para a IoT. Ou seja, não existe uma diferença significativa no tempo de resposta de 10 a 50 Apps, que é de cerca de 7% para CoAPs (CoAP + DTLS) e 6% para CoAP.

Considerando o tamanho das requisições, é possível observar que existe um *overhead* considerável para mensagens entre 1024 e 2048 bytes. No entanto, desde que foi utilizada a chave simétrica, CoAP ou CoAPs, a proposta apresentou um pequeno *overhead* sobre 4096 bytes, sem afetar o tempo médio de resposta.

Para validar o SSO foram executados testes com CoAPs com e sem autenticação utilizando 50 App. Devido aos scripts de automação não foram visualizadas diferenças em relação ao tempo de respostas para mensagens até 2048 bytes. É possível concluir que não se ganha quantitativamente, mas sim qualitativamente. Com objetivo de avaliar o limite operacional da proposta, foram realizados testes com CoAPs e autenticação SSO (mensagens de 32 a 4096 bytes), iniciando com 10 Apps e incrementando o mesmo número, concluímos que 120 Apps é o limite suportado.

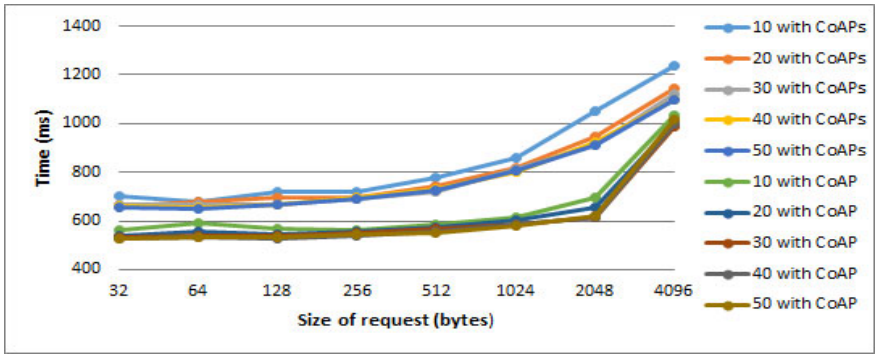


Figura 6 - Avaliação do Protótipo

5. Conclusão

O trabalho apresentou um método de autenticação para integrar um IdM do contexto da Internet à IoT. A ligação entre os contextos é fornecida por um *gateway* incapaz de visualizar o conteúdo das mensagens. As chaves simétricas são adequadas para a IoT para a proteção de mensagens fim-a-fim (a partir dos *appliances* até o *Customer Service*), evitando que o *gateway* seja um ponto único de falha. A autenticação do técnico da fabricante no *gateway* tem o objetivo de mitigar os possíveis ataques provenientes de Internet. O *gateway* fornece isolamento do *appliance* na Internet.

SSO "encapsulado no IdM" é adequado para a IoT, pois retira a necessidade do *appliance* precisar um esforço extra para interagir com um servidor de Internet, tal como proposto na literatura. O técnico pode acessar múltiplos *appliances* com uma única autenticação, sem necessitar saber uma senha diferente para cada *appliance,* e também sem necessitar utilizar a mesma senha para todos os *appliances* - prática que submete os *appliances* ao risco. O protótipo mostrou-se viável no seu tempo de resposta, variando o número de *appliances* e o tamanho das mensagens. A abordagem proposta não possui *overhead* significativo para o tempo de resposta de 10 a 50 Apps e mensagens de 32 a 4096 bytes por mensagem. O tempo de resposta, em média, fica

abaixo de 1250 ms por requisição, um *overhead* aceitável, levando em conta que é apresentado um IdM baseado em chave para a segurança fim-a-fim em IoT.

6. Publicações

Witkovski, A., Santin, A., Abreu, V., Marynowski, J.. **An IdM and Key-Based Authentication Method for Providing Single Sign-On in IoT**. In proc. of IEEE Global Communications Conference, 2015, San Diego, pp.1-6.

Artigo: https://secplab.ppgia.pucpr.br/files/papers/2015-2.pdf

Dissertação: https://www.ppgia.pucpr.br/pt/arquivos/mestrado/dissertacoes/2015/Adriano_Witkovski_Dissertacao.pdf

Patente: Witkovski, A. ; Santin, A. ; Abreu, V. . **Mecanismo baseado em chaves e Gestão de Identidades para autenticação unificada em Internet das Coisas**. 2015, BR1020150306326, INPI - Instituto Nacional da Propriedade Industrial. Depósito: 07/12/2015; Depósito PCT: 07/12/2015. Brasil.

Referências

Atzori L., Iera A., and Morabito G. (2010), "The Internet of Things: A survey," Computer. Networks, vol. 54, pp. 2787–2805.

Babar S., Mahalle P., Stango A., Prasad N., and Prasad R. (2010), "Proposed security model and threat taxonomy for the Internet of Things (IoT)," in Proc. of the CCIS - Communications in Computer and Information Science, pp. 420–429.

Miorandi D., Sicari S., De Pellegrini F., and Chlamtac I. (2012), "Internet of things: Vision, applications and research challenges," Ad Hoc Networks, pp. 1497–1516.

Belapurkar A., Chakrabarti A., Ponnapalli H., Varadarajan N., Padmanabhuni S., and Sundarrajan S. (2009) "Distributed Systems Security: Issues, Processes and Solutions", John Wiley & Sons.

Shelby Z., Hartke K., and Bormann C., "The Constrained Application Protocol (CoAP)", IETF RFC 7252.

Rescorla E., Modadugu N., "Datagram Transport Layer Security Version 1.2.", IETF RFC 6347.

Van Thuan D., Butkus P., and Van Thanh D. (2014) "A user centric identity management for Internet of things," in Proc. of the ICITCS - IT Convergence and Security, pp. 1–4.

Fremantle P., Aziz B., Kopecky J., and Scott P. (2014), "Federated Identity and Access Management for the Internet of Things," in Proc. of the Int. Workshop on Secure Internet of Things, pp. 10–17.

Leo M., Battisti F., Carli M., and Neri A., (2014) "A federated architecture approach for Internet of Things security," in Proc. of the EMTC - Euro Med Telco, pp. 1–5.

Chadwick, D. W. (2009) "Federated Identity Management. Foundations of Security Analysis and Design" V. Heidelberg: Springer-Verlag Berlin, p. 96-120.

XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

Niterói, RJ, novembro de 2016

WTICG – Workshop de Trabalhos de Iniciação Científica e de Graduação

Implementação compacta em software do algoritmo Ketje

Carlos Gregoreki[1], **Diego F. Aranha**[1]

[1]Instituto de Computação (IC) - Universidade Estadual de Campinas (Unicamp)

cgregoreki@gmail.com, dfaranha@ic.unicamp.br

Abstract. *The advances in mobile computing brought to light the information security problem, imposing new requirements for cryptosystems for increasingly smaller hardware, with reduced computation and storage capacity. This work presents a compact implementation of the authenticated encryption scheme Ketje performed on the programming language C, according to version 1.1 of its specification, for keys and nonces with length multiple of 8 bits. The present work aimed at improvements in code readability and resulted in a 69% smaller compiled code, while behaving similarly from the point of view of performance.*

Resumo. *O avanço do desenvolvimento de dispositivos móveis trouxe à tona o problema de segurança da informação, impondo novos requisitos de sistemas criptográficos para dispositivos cada vez menores, com capacidades de computação e armazenamento reduzidas. Este trabalho apresenta uma implementação compacta do esquema de cifração autenticada Ketje realizada em linguagem C, conforme a versão 1.1 de sua especificação, para chaves e nonces de tamanhos múltiplos de 8 bits. Visando uma melhoria na legibilidade do código, o presente trabalho teve como resultado uma redução no tamanho final do código compilado de até 69% comportando-se de maneira semelhante à referência do ponto de vista de desempenho.*

1. Introdução

Uma das formas mais utilizadas para armazenar informações confidenciais ou trocar mensagens de maneira segura é a criptografia simétrica. Essa técnica exige que exista uma chave secreta que deve ser compartilhada entre o cifrador da mensagem e o decifrador, e que o compartilhamento dessa chave seja dada por meios seguros para garantir sua confidencialidade. Desta forma, uma mensagem pode ser decifrada por uma chave em conjunto com um algoritmo se eles forem os mesmos que foram utilizados na cifração.

Ao longo do desenvolvimento de sistemas criptográficos, muitas formas de ataques por canais laterais – que objetivam identificar características de um sistema criptográfico através de, por exemplo, padrões de consumo de energia e ondas eletromagnéticas – apareceram e se tornaram técnicas cada vez mais frequentes de ataque. Além disso, com a portabilidade sendo cada vez mais desenvolvida, segurança em dispositivos pequenos e móveis tem sido cada vez mais exigida.

Neste contexto, implementações eficientes de novas primitivas criptográficas voltadas à dispositivos com restrição de memória e espaço e com resistência a esses ataques são um tema presente em competições de segurança como a *CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness*[1].

[1]https://competitions.cr.yp.to/caesar.html

No presente trabalho, foi produzida uma implementação[2] alternativa e mais compacta do algoritmo *Ketje* em suas versões recomendadas pela referência, *KetjeJr* e *KetjeSr*, para funcionamento com chaves e *nonces* de tamanho de bits múltiplos de 8 – casos comuns – visando a legibilidade do código, o espelhamento entre a implementação produzida deste trabalho e sua especificação, além da redução do tamanho do código compilado.

É importante citar que este algoritmo foi submetido à CAESAR por tratar dos requisitos da competição, a citar: é um cifrador autenticado com suporte à *associated data* (AEAD), é possível recuperar os dados originais com o criptograma e o *associated data*, possui flexibilidade para aceitar chaves de tamanho não comuns, isto é, 80, 128 ou 256 bits, e especifica dois casos particulares formando assim uma "família" de cifradores autenticados. Também propõe vantagens em relação ao AES-GCM. De acordo com a especificação, *Ketje* possui melhor desempenho e menor código e se comporta de maneira segura perante a ataques de canais laterais, além de apresentar suporte à sessões, isto é, autenticação de sequência de mensagens em série ao invés de apenas uma mensagem.

Para observar a execução da implementação proposta, escolhemos como plataformas alvo processadores que integrassem dispositivos móveis e que fossem conhecidos pela comunidade em geral, que nesses processadores estivesse sendo executado um sistema operacional de código aberto, e que fosse possível coletar dados de desempenho de maneira simples e confiável. Portanto, escolhemos o ArchLinux como sistema operacional e processadores ARM Cortex-A.

2. Preliminares

2.1. Cifração Autenticada

De acordo com [Bellare, Rogaway and Wagner 2003], um esquema de cifração autenticada é um mecanismo de cifração simétrica, no qual uma mensagem M é transformada em um criptograma CT com o objetivo que CT proteja a autenticidade e a privacidade de M. Segundo [Bertoni et al 2014], podemos entender cifração autenticada como um conjunto de operações, constituída de uma função que recebe um cabeçalho A e uma cadeia de caracteres B como entradas, e produz um criptograma C e uma *tag* T como saída – processo chamado de *cifração* –, e do processo inverso, que recebe um cabeçalho A e um criptograma C, produzindo uma cadeia de caracteres B e uma *tag* T, chamado de *decifração*.

2.2. Notação

Neste trabalho, utilizaremos a notação exposta na Tabela 1.

3. Ketje

O esquema de cifração autenticada *Ketje* se baseia na construção *duplex* – variante da construção esponja – na sua versão *MonkeyDuplex*. De acordo com sua definição [Bertoni et al 2011b], esponja é uma forma geral para a geração de funções de *hash* com entradas e saídas de tamanho arbitrário, baseada numa transformação de tamanho fixo que opera em cima de um número fixo de bits. *Duplex*, cuja segurança é semelhante, permite que blocos únicos de saída sejam gerados a medida que blocos de entrada são incorporados ao estado da construção. Assim, permite-se construir esquemas criptográficos requisitando apenas uma chamada dessa transformação por bloco.

[2]Disponível em http://github.com/cgregoreki/ketje-impl-v2

Tabela 1. Símbolos e seus significados

Símbolo	Significado
$\oplus$	Xor bit a bit
$\lfloor . \rfloor l$	Truncagem para l bits
$\lvert K \rvert$	Tamanho de K
$a \lvert\rvert b$	Concatenação de a e b
$f \circ g$	Composição de funções f e g: $f(g)$
$GF(p^n)$	Bits. $GF(2) = \{0, 1\}$

3.1. As construções esponja e duplex

Citando [Bertoni et al 2007], a operação da esponja é descrita em duas partes: *absorbing* e *squeezing*. A primeira é a etapa em que a esponja absorve e incorpora a entrada ao seu estado interno, e a segunda é quando uma saída de tamanho desejado é produzida baseada em tal estado. Desta forma, ao final de sua operação, a esponja terá produzido um *hash* pseudoaleatório de sua entrada.

A construção esponja está detalhada na Figura 1. A entrada M é divida em blocos após um *padding* e é absorvida para depois, então, gerar a saída Z. A função f é uma função de permutação.

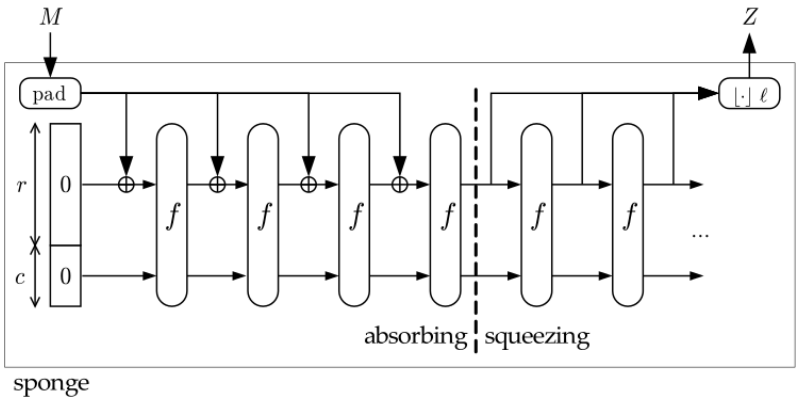


Figura 1. A construção de esponja, adaptado de [Bertoni et al 2011a]

A Figura 2 mostra a construção *duplex*. Aqui, ao invés de gerar uma saída integralmente, a saída é gerada bloco a bloco à medida que os blocos de entrada são incorporados.

3.2. MonkeyDuplex e MonkeyWrap

MonkeyDuplex usa uma função de permutação f com uma quantidade regulável de rodadas, aceita uma *cadeia de bits* como entrada e produz uma *cadeia de bits* como saída. Esta saída depende de todas as entradas recebidas até então. Diferentemente de *duplex*, na qual foi baseada, essa construção aceita dois tipos de chamadas que são diferentes em número de rodadas que são executadas pela função de permutação entre a entrada e a saída. A construção $MonkeyDuplex[f, r, n_{start}, n_{step}, n_{stride}]$ funciona de acordo com o seguinte:

- Uma instância de *MonkeyDuplex*, denotada por *D*, possui um estado de *b* bits igual ao tamanho da largura da função de permutação. Essa instância é inicializada com

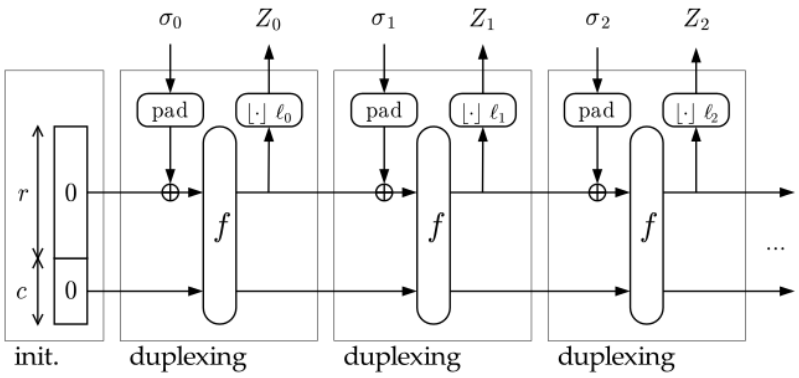


Figura 2. A construção duplex, adaptado de [Bertoni et al 2011a]

uma *string I*, que pode ser de tamanho quase igual a *b*, sendo que *I* é estendida até *b* bits por uma função *multi-rate padding*. Posteriormente, a função de permutação é aplicada n_{start} vezes.

- Em seguida, com chamadas para $D.step(\sigma, l)$ e $D.stride(\sigma, l)$, pode-se injetar no estado uma palavra σ de até $r - 2$ bits. Depois que os bits forem injetados, tanto $f[n_{step}]$ quanto $f[n_{stride}]$ são aplicadas ao estado e os primeiros l bits são extraídos.

Para melhor ilustrar, a construção *MonkeyDuplex* está exposta na Figura 3. O algoritmo é descrito na Figura 4.

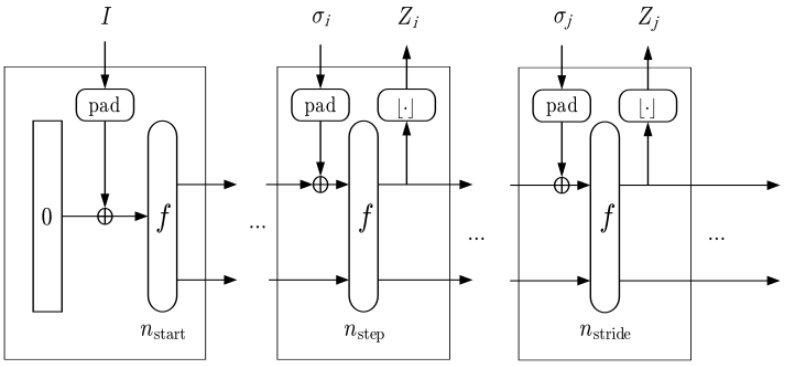


Figura 3. A construção MonkeyDuplex, adaptado de [Bertoni et al 2014]

Por sua vez, *MonkeyWrap* é um modo de cifração autenticada que é construído com a construção *MonkeyDuplex*. Assim, *MonkeyWrap* se vale das operações de *MonkeyDuplex – step* e *stride* – para manipular seu estado, utilizando-as nas suas funções de cifração e decifração. Portanto, depois de inicializar seu estado com a chave secreta e com um *nonce*, e adicionar os dados de cabeçalho ao estado fazendo chamadas para a função *step* de *MonkeyDuplex*:

- Para cifrar uma mensagem, *MonkeyWrap* extrai o texto cifrado do estado à medida que vai adicionando blocos de bits da mensagem original e fazendo novas chamadas a *step*. A cada bloco da mensagem original inserida, um bloco da mensagem cifrada é gerada.

Algorithm 1: The $MonkeyDuplex[f, r, n_{start}, n_{step}, n_{stride}]$ construction:

Require: $2 < r < b, n_{step} < n_{stride}$
Require: $s \in \mathbb{Z}_2^b$

Interface: $D.start(I)$ with $I \in \mathbb{Z}_2^{\leq b-2}$
$s = I||pad10*1[b](|I|)$
$s = f[n_{start}](s)$

Interface: $Z = D.step(\sigma, l)$ with $l \leq r, \sigma \in \mathbb{Z}_2^{\leq r-2}$ and $Z \in \mathbb{Z}_2^l$
$P = \sigma||pad10*1[r](|\sigma|)$
$s = s \oplus (P||0^{b-r})$
$s = f[n_{step}](s)$
return $\lfloor s \rfloor_l$

Interface: $Z = D.stride(\sigma, l)$ with $l \leq r, \sigma \in \mathbb{Z}_2^{\leq r-2}$ and $Z \in \mathbb{Z}_2^l$
$P = \sigma||pad10*1[r](|\sigma|)$
$s \oplus (P||0^{b-r})$
$f[n_{stride}](s)$
return $\lfloor s \rfloor_l$

Figura 4. Algoritmo para MonkeyDuplex, adaptado de [Bertoni et al 2014]

- Para decifrar uma mensagem, *MonkeyWrap* recupera o texto original à medida que insere blocos de bits do texto cifrado ao estado (com chamadas a *step*).
- Para cada uma das funções de cifração e decifração, uma *tag* é gerada ao final fazendo chamadas às funções *stride* e *step*.

O algoritmo de *MonkeyWrap* está representado na Figura 5.

3.3. As permutações *Keccak-p*

Como vimos, na construção *MonkeyDuplex*, é necessária a presença de uma função de permutação. Para o *Ketje*, a função de permutação escolhida é a função $Keccak\text{-}p$ que, de acordo com [Bertoni et al 2011a], é uma derivação da permutação $Keccak\text{-}f$ com um número regulável de rodadas. Uma permutação $Keccak\text{-}p$ é definida pela sua largura *b* em bits e seu número n_r de rodadas. $Keccak\text{-}p[b, n_r]$ consiste da aplicação das últimas n_r rodadas de *Keccak-f[b]*, onde $b = 25$ x 2^l e $b \in \{25, 50, 100, 200, 400, 800, 1600\}$.

A permutação $Keccak\text{-}p[b, n_r]$ é descrita como uma sequência de operações realizadas num estado *a* que é uma matriz tridimensional de elementos GF(2), da forma $a[5, 5, w]$, com $w = 2^l$, formada a partir de 5 passos. Isto é, podemos admitir que o estado a é uma matriz tridimensional com 5 linhas, 5 colunas e w células de profundidade – células estas que podem ter valores 0 ou 1 – sendo que w depende diretamente do parâmetro b, sendo mais especificamente $w = (b/25)$. Por exemplo, esse estado, para $Keccak\text{-}p[400, n_r]$, seria uma matriz com elementos $\in \{0, 1\}$ de dimensões $[5, 5, 16]$.

Uma rodada *R* sobre essa matriz pode ser definida como:

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

onde ι, χ, π, ρ e θ são funções definidas na Figura 6, tal que *RC* – conjunto de constantes de rodada – é dado por:

$$RC[i_r][0, 0, 2^j - 1] = rc[j + 7i_r], 0 \leq j \leq l$$

Algorithm 2: The $MonkeyWrap[f, \rho, n_{start}, n_{step}, n_{stride}]$ construction:

Require: $0 < \rho \leq b - 4$
Require: $D = MonkeyDuplex[f, \rho + 4, n_{start}, n_{step}, n_{stride}]$

Interface: W.initialize(K, N) with $K \in \mathbb{Z}_2^{\leq b-18}$, $|K|$ mod $8 = 0$ and $N \in \mathbb{Z}_2^{\leq b-|K|-1}$, D.start(keypack$(K, |K| + 16)||N)$

Interface: $(C, T) = W$.wrap(A, B, l) with $A, B, C \in \mathbb{Z}_2^*, l \geq 0$, and $T \in \mathbb{Z}_2^l$
for $i = 0$ to $||A|| - 2$ **do**
    D.step$(A_i||00, 0)$
$Z = D$.step$(A_{||A||-1}||01, |B_0|)$
$C_0 = B_0 \oplus Z$
for $i = 0$ to $||B|| - 2$ **do**
    $Z = D$.step$(B_i||11, |B_{i+1}|)$
    $C_{i+1} = B_{i+1} \oplus Z$
$T = D$.stride$(B|||B|| - 1||10, \rho)$
while $|T| < l$ **do**
    $T = T||D$.step$(0, \rho)$
$T = \lfloor T \rfloor_l$
return (C, T)

Interface: $B = W$.unwrap(A, C, T) with $A, B, C, T \in \mathbb{Z}_2^*$
for $i = 0$ to $||A|| - 2$ **do**
    D.step$(A_i||00, 0)$
$Z = D$.step$(A_{||A||-1}||01, |B_0|)$
$B_0 = C_0 \oplus Z$
for $i = 0$ to $||C|| - 2$ **do**
    $Z = D$.step$(B_i||11, |C_{i+1}|)$
    $B_{i+1} = C_{i+1} \oplus Z$
$T' = D$.stride$(B_{||C||-1}||10, \rho)$
while $|T'| < |T|$ **do**
    $T' = T'||D$.step$(0, \rho)$
$T' = \lfloor T \rfloor_{|T|}$
if $T = T'$ **then**
    return B
else
    return error

Figura 5. Algoritmo para MonkeyWrap, adaptado de [Bertoni et al 2014]

onde i_r é o índice da rodada. Todos os outros valores de $RC[i_r][x, y, z]$ são iguais a zero. Os valores de $rc[t]$ são definidos como o resultado de um *binary linear feedback shift register* (LFSR):

$$rc[t] = (x^t \pmod{x^8 + x^6 + x^5 + x^4 + 1}) \pmod{x}$$

$\theta : a[x, y, z] \leftarrow a[x, y, z] + \sum_{y'=0}^{4} a[x-1, y', z] + \sum_{y'=0}^{4} a[x+1, y', z-1]$,
$\rho : a[x, y, z] \leftarrow a[x, y, z-(t+1)(t+2)/2]$,
with satisfying $0 \leq t < 24$ and $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$ in GF$(5)^{2x2}$,
or $t = -1$ if $x = y = 0$,
$\pi : a[x, y] \leftarrow a[x', y']$, with $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$,
$\chi : a[x] \leftarrow a[x] + (a[x+1] + 1)a[x+2]$,
$\iota : a \leftarrow a + RC[i_r]$.

Figura 6. Funções de Keccak-f, adaptado de [Bertoni et al. 2011].

3.4. *KetjeJr* e *KetjeSr*

Como visto na sua especificação, são recomendadas duas instâncias concretas de *Monkey-Wrap* com função de permutação $Keccak\text{-}p$. A principal proposta é chamada de *KetjeSr*, sendo uma segunda chamada de *KetjeJr*:

$$KetjeSr = MonkeyWrap(Keccak{-}p[400], \rho = 32, n_{start} = 12, n_{step} = 1, n_{stride} = 6)$$

$$KetjeJr = MonkeyWrap(Keccak{-}p[200], \rho = 16, n_{start} = 12, n_{step} = 1, n_{stride} = 6)$$

KetjeSr suporta chaves de até 382 bits, e nonces de até $382 - |K|$. A adoção de chaves de tamanhos maiores que 128 bits (tamanho recomendado) pode ser feita como uma possível contra-medida contra ataques do tipo *multi-target*, onde um agente procura atacar um conjunto de chaves ao invés de uma chave apenas. *KetjeJr* suporta chaves de até 182 bits e nonces de até $182 - |K|$ bits. É recomendada a adoção de chaves de 96 bits.

4. Implementação

A implementação foi feita em C, assim como na referência. Além de manter a fidelidade com a especificação, o presente trabalho se preocupou com a legibilidade do código e seu resultado em tamanho final, a ponto de que ficasse menor que o tamanho do executável gerado pela referência sem, é claro, ter muito a perder com relação à eficiência.

Em termos de legibilidade, é notável que o código de referência não acompanha de forma trivial a especificação, podendo-se notar várias operações condicionais que tornam a leitura onerosa; operações estas que, por decisões de projeto, evidentemente estão presentes em função dos testes que foram criados e do propósito de evitar a chamada de operações em ordem incorreta ou inválida. Por exemplo, executar a cifração antes de adicionar os dados de cabeçalho ao estado. Dessa forma, depois da análise da leitura da referência, foi decidido que as operações condicionais citadas deveriam ser refeitas e até mesmo excluídas para evitar prolixidade na leitura, mantendo a correção do código.

A fim de detalhar o estilo de código adotado que impacta na leitura, um trecho de código da presente implementação está disposto na Listagem 2, enquanto o trecho de

código que realiza as mesmas ações na referência está colocado na Listagem 1. Nestas listagens, fica claro a não linearidade de leitura da referência devido a presença de operações condicionais e de retornos em caso de erro, enquanto a implementação compacta segue suas operações sem tais oscilações.

```
1
2  ...
3    if ( (instance->phase & Ketje_Phase_FeedingAssociatedData) != 0){
4          Ket_Step( instance->state, instance->dataRemainderSize, ↩
               FRAMEBITS01 );
5          instance->dataRemainderSize = 0;
6          instance->phase = Ketje_Phase_Wrapping;
7      }
8      if ( (instance->phase & Ketje_Phase_Wrapping) == 0){
9          return 1;
10     }
11
12     if ( (instance->dataRemainderSize + dataSizeInBytes) > ↩
           Ketje_BlockSize ) {
13   ...
14     if ( dataSizeInBytes > Ketje_BlockSize )
15         {
16           for ...
17         }
18 ...
```

Listagem 1. Trecho de código da referência

```
1  ...
2     Ketje_step(instance->state, instance->dataRemainderSize, ↩
          FRAMEBITS01);
3     instance->dataRemainderSize = 0;
4
5     int nblocks_B = return_ketjeJrSize(dataSizeInBytes_B)/↩
          Ketje_BlockSize;
6     if (nblocks_B > 0){
7           for (i = 1; i <= nblocks_B;i++){
8              ...
9  ...
```

Listagem 2. Trecho de código da implementação compacta

Ademais, a especificação descreve a inserção dos dados de cabeçalho no estado como parte interna dos algoritmos de cifração e decifração. Todavia, a implementação de referência externaliza esse trecho numa função separada, chamando-a antes da execução da cifração e decifração. Assim, ainda prezando pela legibilidade e espelhamento do código com a especificação, decidimos seguir a especificação colocando tal inserção dentro dessas funções.

O código da função de permutação $Keccak\text{-}p$ utilizado foi o mesmo código de referência com modificações para que o conjunto de constantes de rodada fossem pré carregadas na memória ao invés de gerá-las dinamicamente, visto que essas constantes

são fixas para cada tamanho do estado da função. Dessa forma, é possível prever que o processador execute uma quantidade de ciclos menor.

Quanto aos testes, é evidente que um esquema diferente da referência fosse elaborado pois nela há um gerador de conteúdo para as chaves e *nonces* com aspectos diferentes dos que foram adotados aqui. Isto posto, os testes utilizados neste trabalho, para garantir a correção do objeto final, seguiram a mesma estrutura dos que estão presentes na referência, tendo a parte dos parâmetros do gerador modificada para que funcionasse apenas para tamanhos múltiplos de 8 bits. Para garantir a integridade e confiabilidade dos dados coletados ao executar os testes, o mesmo esquema foi reproduzido no arquivo de referência, substituindo os testes originais, para que executassem os mesmos que a implementação aqui proposta. Foram sobre estes testes que os dados foram coletados.

Assim, os testes para correção e benchmarking são constituídos de chaves e *nonces* dos seguintes tamanhos, respectivamente, para *KetjeJr*: 176 bits e 0 bits, 152 bits e 24 bits, 128 bits e 48 bits, 112 bits e 64 bits, 96 bits e 80 bits, e 96 bits e 16 bits. Para *KetjeSr*, respectivamente: 376 bits e 0 bits, 280 bits e 96 bits, 184 bits e 192 bits, 160 bits e 216 bits, 136 bits e 240 bits, 112 bits e 264 bits, 96 bits e 280 bits, 96 bits e 120 bits, 96 bits e 56 bits. Para cada um desses pares, dados de cabeçalho e palavras a serem cifradas foram geradas por uma função semelhante a que a referência utiliza para o mesmo propósito, sendo que seus comprimentos são variáveis, sempre respeitando o limite do estado. Portanto, um total de 408 testes são executados para *KetjeJr* e 5922 para *KetjeSr*.

5. Resultados Experimentais

Os códigos, tanto de refêrencia quanto da implementação proposta, foram compilados de duas formas: com a flag *-Os*, que é para otimizar tamanho e, separadamente, com a flag *-O3* que serve para otimizar desempenho. O compilador utilizado foi o GCC 6.1.1.

Para observar o desempenho dos arquivos gerados pelo compilador (GCC), utilizamos a *Performance Monitoring Unit* (PMU)[3] disponível nos processadores ARM. Assim, pudemos acessar um registrador de contagem de ciclos por meio de um módulo carregado diretamente no *kernel* do sistema operacional utilizado (ArchLinux). Para o tamanho dos arquivos finais, o comando *size -B <nome_do_arquivo>* acessível nas plataformas Linux foi utilizado, observando sempre a sessão *text* sua saída, a qual traduz o tamanho do código em instruções que o processador irá executar. Tais medições foram realizadas em quatro processadores: Cortex-A8, Cortex-A15, Cortex-A53 com plataforma 32-bit e Cortex-A53 com plataforma 64-bit.

Expomos os dados coletados de tamanho do código compilado na Tabela 2 e os dados de desempenho nas Tabelas 3 e 4. Tanto nessas tabelas quanto nos gráficos desse trabalho, os dados para a implementação de referência possuem o prefixo *Ref*.

Observando a Tabela 2, podemos notar uma diferença significativa entre as reduções de espaço para os dois procedimentos. Isto é, enquanto tivemos uma redução de tamanho de apenas 21% utilizando a flag *-Os*, obtivemos uma redução de até 69% para *KetjeJr* com a flag *-O3*, sendo que o tamanho absoluto do primeiro ainda continua menor que o segundo. Para *KetjeSr*, a redução foi de até 18% com *-Os* e 69% com *-O3*. Em todos os processadores, os tamanhos obtidos foram semelhantes.

[3] Veja http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0388g/BABDEBHF.html

Tabela 2. Comparação: Tamanho de texto de código (código compilado) do executável

Utilizando Flag -Os				
Algoritmo	Ref Jr	Jr	Ref Sr	Sr
Tamanho	7186	5690	7066	5796
Utilizando Flag -O3				
Algoritmo	Ref Jr	Jr	Ref Sr	Sr
Tamanho	30106	9348	32402	10086

Tabela 3. Comparação: Dados da contagem de ciclos - flag -Os

	Cortex-A8	**Cortex A15**	**32-bit Cortex-A53**	**64-bit Cortex-A53**
Ref Jr	103432748	75544229	99213259	58861866
Jr	103382644	70264745	97515256	57050181
Ref Sr	1473854399	1016027233	1362340599	827482656
Sr	1423607979	985182579	1343514261	826662312

Tabela 4. Comparação: Dados da contagem de ciclos - flag -O3

	Cortex-A8	**Cortex A15**	**32-bit Cortex-A53**	**64-bit Cortex-A53**
Ref Jr	36394258	27237526	28214047	25694832
Jr	28168087	24243611	24184896	23212240
Ref Sr	489483314	457071392	484010978	396564278
Sr	453691027	425575715	441179836	401887092

Em termos de desempenho, também podemos notar uma diferença entre os resultados dos dois procedimentos. Enquanto para *-Os* tivemos uma melhora de até 7% com *KetjeJr* e de até 3% com *KetjeSr* no Cortex-A15, para *-O3* podemos observar um ganho de 11% e 7% para o mesmo processador, respectivamente. Portanto, pode-se admitir que o resultado ficou num patamar muito próximo, tendo ganhos pequenos em relação a referência. O gráfico da Figura 7 expõe essa comparação para o procedimento de compilação com *-O3*.

6. Conclusão

Este trabalho propõe uma implementação mais compacta do algoritmo de criptografia *Ketje* para chaves e *nonces* de tamanhos múltiplos de 8 bits, que possui menor tamanho do código compilado comparado ao da referência, sem perder em desempenho. Além disso, contribui para a comunidade em geral ao expôr um código de leitura mais fluída.

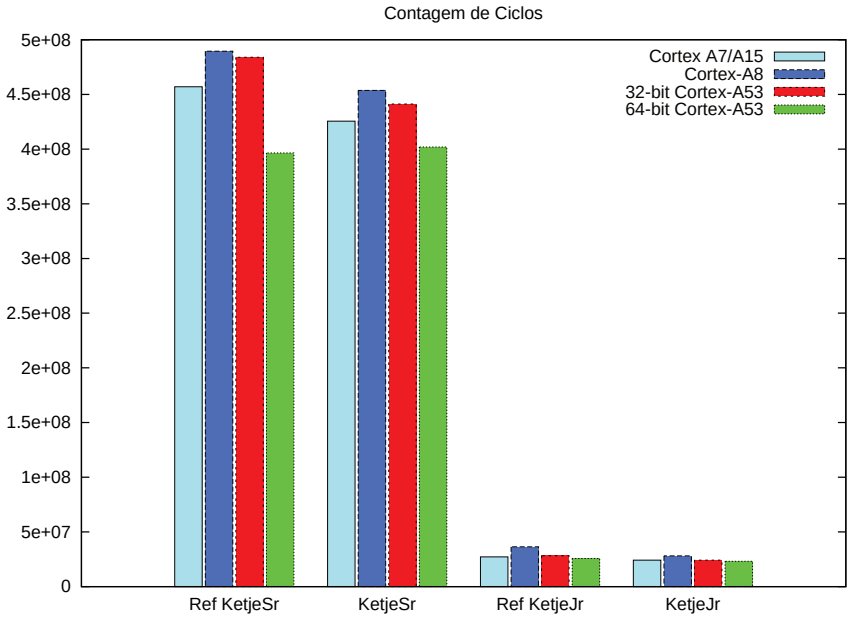


Figura 7. Contagem de Ciclos para KetjeJr e KetjeSr (flag -O3)

Referências

Bertoni, G., Daemen, J., Peeters, M., and Assche, G. (2011a). The Keccak Reference. http://keccak.noekeon.org/.

Bertoni, G., Daemen, J., Peeters, M., and Assche, G. (2011b). Cryptographic sponge functions. http://sponge.noekeon.org/.

Bertoni, G., Daemen, J., Peeters, M., and Assche, G. (2007). Sponge Functions. ECRYPT Workshop on Cryptographic Hash Functions, May 24-25, 2007.

Bertoni, G., Daemen, J., Peeters, M., Assche, G. and Keer, R. (2014). CAESAR submission: Ketje v1. http://ketje.noekeon.org/.

Bellare, M., Rogaway, P. and Wagner, D. (2003). A conventional authenticated-encryption mode. http://eprint.iacr.org/2003/069/

Implementação eficiente do algoritmo Keyak para ARMv8

André C. de Moraes[1], **Diego F. Aranha**[1]

[1]Instituto de Computação - Universidade Estadual de Campinas (UNICAMP)

andre.moraes@students.ic.unicamp.br, dfaranha@ic.unicamp.br

Abstract. *Keyak is an authenticated encryption scheme based on sponge construction. Cryptographic sponges provide a good amount of security at a high computational cost, which hinders its performance, specially in resource-constrained devices. The main contributions of this work aimed to improve the performance of the Keyakv2 in platforms using the ARM architecture, by implementing the algotrithm's core functions in assembly language with the NEON vector instruction set.*

Resumo. *Keyak é um algoritmo de cifração autenticada baseada na construção esponja. Esponjas criptográficas fornecem um bom nível de segurança por um custo computacional elevado, o que prejudica seu desempenho, especialmente em dispositivos com recursos limitados. As contribuições deste trabalho objetivaram aprimorar o desempenho do Keyakv2 nas plataformas baseadas na arquitetura ARM através da implementação em linguagem de montagem das principais rotinas do algoritmo utilizando com o conjunto de instruções vetoriais NEON.*

1. Introdução

Cifração autenticada é um algoritmo que providencia confidencialidade e autenticidade de dados, onde a decifração é feita de maneira conjunta com a verificação de integridade [K. Paterson 2013]. Tais propriedades protegem o algoritmo de ataques sob o cenário de *Man-in-the-Middle* [F. Callegati et al 2009], como *bit-flipping* [K. Graves 2010].

Um padrão muito utilizado para esse tipo de algoritmo é conhecido como **construção esponja**, que possui propriedades relevantes para a cifração autenticada como tamanho de entrada e saída arbitrários [G. Bertoni et al. 2011a]. **Keccak** é uma família de algoritmos de construções esponja, utilizada em diversas aplicações criptográficas como cifração autenticada, função *hash* e gerador de números aleatórios [G. Bertoni et al. 2013a]. Como destaque, o algoritmo **SHA-3**, homologado pelo *NIST*, faz parte da família *Keccak* [NIST 2014].

Keyak [G. Bertoni et al. 2015a] é um algoritmo baseado na construção esponja conhecida como **Keccak-**p [G. Bertoni et al. 2013a], proposta pelos mesmos autores do *Keccak*, e é uma das submissões para a competição **CAESAR**. A competição busca promover um novo algoritmo padrão na área de cifração autenticada. O **Keyakv2**, segunda versão da primitiva *Keyak*, cumpre todos os requisitos estabelecidos na terceira e mais recente especificação da competição [G. Bertoni et al. 2015b]. Qualquer menção no texto ao algoritmo *Keyak* a partir de então refere-se ao *Keyakv2*.

Neste trabalho, buscamos desenvolver uma implementação de alto desempenho do *Keyak* para dispositivos com **ARMv8**, a ser explicada em detalhes na **Seção 2**. O

código desenvolvido [A. Moraes 2016] foi aceito pelos autores do *Keyak* e pode ser visto no repositório oficial [G. Bertoni et al. 2015d]. Durante os testes, foi possível notar um erro na implementação em *assembly* do *ARMv7*, e notificar os autores do *Keyak* com a correção necessária [A. Moraes 2016].

2. Proposta

O algoritmo *Keyak* fornece um bom nível de segurança na troca de dados, sendo uma solução válida para desenvolvedores de *software* que busquem prover segurança para suas aplicações. Entretanto, sua baixa eficiência dificulta o seu uso em dispositivos com recursos limitados, como telefones celulares e *tablets*.

Uma maneira de contornar essa restrição é implementar as rotinas mais custosas do algoritmo em línguagem de máquina otimizada para as principais arquiteturas do mercado. No repositório oficial [G. Bertoni et al. 2015d] dos autores do *Keyak*, constam implementações para diversas arquiteturas, como por exemplo *ARMv7*.

A nova geração de processadores *ARMv8* introduz várias novidades em relação à geração antiga *ARMv7*, como o processador operar em 64 *bits* e não mais 32 *bits*, o pacote instruções vetoriais *NEON* tornando-se padrão e não mais opcional, uma quantidade maior de registradores vetoriais e uma nova síntaxe de linguagem de máquina. Entretanto tais mudanças acabam deixando a arquitetura não compatível com código legado de gerações anteriores.

Em vista desses fatos, é necessário escrever uma nova implementação para a arquitetura *ARMv8*, utilizando suas novas capacidades. Neste trabalho, procurou-se utilizar de instruções vetoriais sempre que possível, em oposição à implementação existente do *ARMv7* que as utiliza apenas em alguns casos.

3. Fundamentação Teórica

3.1 Construção Esponja

A **construção esponja** [G. Bertoni et al. 2011a] é definida como uma máquina de estados finita que consome dados de entrada de tamanho arbitrário, resultando em uma saída de um outro tamanho arbitrário. A esponja possui um **estado interno** de **largura** b *bits*, composta de duas partes, o ***bitrate*** de tamanho r a **capacidade** de tamanho c, sendo $b = c + r$. A esponja utiliza de uma **função de permutação** f que é aplicada em todos os b *bits* do seu estado interno.

Durante a fase de absorção, a esponja insere *bits* da entrada através de operações de ***XOR*** com os dados já existentes no estado, aplicando a função f a cada r *bits* inseridos. Após todos os dados da entrada serem inseridos no estado, a esponja entra na fase de compressão, e *bits* são extraídos do estado, aplicando f a cada r *bits* extraídos. A quantidade c de *bits* correspondente à capacidade do estado não é relacionada diretamente com os dados de entrada ou saída, sendo apenas um resultado de aplicações de f sobre o estado interno.

A única restrição sobre os dados de entrada e saída é que eles devem estar particionados em **blocos** $\sigma_0, \sigma_1, ..., \sigma_n$ de tamanho W *bits* cada (chamado de unidade de alinhamento), sendo r um múltiplo de W, o que é facilmente obtido utilizando alguma regra de preenchimento (*padding*).

3.2 Construção Duplex

A **construção duplex** [G. Bertoni et al. 2011b] é um caso particular de uma construção esponja em que não existe diferença entre a fase de absorção e a fase de compressão, uma vez que dados são lidos da entrada e produzidos na saída junto com cada aplicação da função de permutação f sobre o estado, em uma única fase de operação denominada ***duplexing***. O nível de segurança continua equivalente, porém há um aumento considerável do desempenho uma vez que o número de execuções da função de permutação f cai consideravelmente.

Uma **construção duplex de estado total** (*FDS*) é um caso particular de construção *duplex*, onde o *bitrate* de absorção passa a ser o tamanho total do estado, ou seja $b = r$, aumentando o número de *bits* inseridos e consequentemente o desempenho, sem custos significativos para a segurança [B. Mennink et al 2015], uma vez que a saída não contém *bits* da capacidade.

A **construção duplex de estado total chaveada** (*FSKD*) por sua vez é obtida quando se inicializa uma **FDS** com uma chave K de tamanho k e então se executa a função f antes de inserir qualquer dado da entrada. É possível inserir apenas a chave ou inserir a chave e inicializar os $(b - k)$ *bits* restantes com os primeiros blocos σ_i da entrada. Em ambos os casos, é necessário garantir que a chave também possa ser particionada em blocos de mesmo tamanho W que a entrada, através do uso de alguma regra de preenchimento.

A **Figura 1** representa uma construção duplex de estado total chaveada, onde a chave inserida é denominada K, a entrada é inserida em blocos σ_i, e a saída é obtida em blocos Z_i.

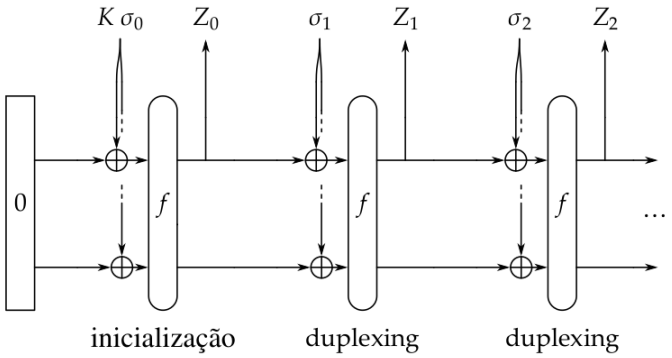


Figura 1. Exemplo de um *FKSD* - [G. Bertoni et al. 2011b]

3.3 Keccak

A família de esponjas **Keccak** utilizam-se de uma mesma função de permutação f conhecida como **Keccak-**f. Existem diversas variações da **Keccak-**$f[b]$ quanto ao tamanho b *bits* aceito, sendo necessário que $b = 25 * 2^l$ com $0 \leq l \leq 6$, ou seja, $b \in (25, 50, 100, 200, 400, 800, 1600)$ *bits*. Esse requisito é uma consequência do algoritmo, que requer mapear o estado recebido para um vetor de permutação de **25 posições**, cada qual de tamanho 2^l *bits*, para então realizar operações neste vetor e finalmente retornar o vetor de permutação para o estado resultante.

As operações se resumem a cinco transformações internas [J.P. Aumasson et al 2009], θ, ρ, π, χ e ι, que são executadas um número N_r de vezes, que depende do tamanho b através da relação $N_r = 12 + 2l$. As transformações envolvem operações lógicas, rotações e permutações entre diversos elementos do vetor de permutação.

Keccak-p é uma função de permutação [G. Bertoni et al. 2013a] que consiste na execução **das últimas n_r rodadas** da função de permutação *Keccak-f* associada, portanto $n_r \leq N_r$. No caso particular em que $n_r = N_r$, temos que *Keccak-$p[b, n_r]$* = *Keccak-$f[b]$*. A razão de se executar apenas algumas rodadas se dá pelo fato de que não é necessário realizar todas as rodadas possíveis para se obter um bom nível de segurança e devido ao custo elevado de cada rodada adicional.

4. Keyak

Keyak é um algoritmo composto de esponjas internas, onde a função de permutação é necessariamente alguma variação da *Keccak-$p[b, n_r]$*, de estado b *bits* e n_r rodadas. Em particular, os autores do *Keyak* recomendam cinco instâncias nomeadas: ***River Keyak***, ***Lake Keyak***, ***Sea Keyak***, ***Ocean Keyak*** e ***Lunar Keyak***. Todas possuem mesma capacidade $c = 256$, tamanho de rótulo $\tau = 128$ e número de rodadas $n_r = 12$. As instâncias se diferem no tamanho da largura b, o qual é 800 para *River* e 1600 para as demais, e ao número de esponjas com *River* e *Lake* possuindo apenas uma única instância, e *Sea*, *Ocean* e *Lunar* possuindo duas, quatro e oito instâncias respectivamente.

O *Keyak* opera em cima do chamado modo ***Motorist***, que é uma construção [G. Bertoni et al. 2015a] baseada em *FSKD* para cifração autenticada em sessões. Cada instância pode ter múltiplas construções *duplex* internas, denominadas ***Pistons***, que são gerenciadas pela ***Engine***, como desmontrado na **Figura 2**.

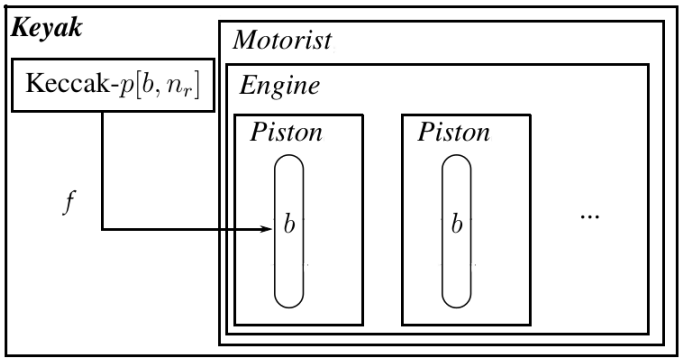


Figura 2. Diagrama esquemático do *Keyak*. A função de permutação f das esponjas é a *Keccak-$p[b, n_r]$*, de mesma largura b que as esponjas internas do *Motorist*.

4.1 Motorist

O papel do *Motorist* é cifrar ou decifrar mensagens, que são compostas de texto claro e metadados, e um possível rótulo (*tag*) associado para ser verificado. Note que é possível enviar somente texto claro, somente metadados ou ainda uma mensagem nula, apenas para verificar um rótulo fornecido.

A mensagem é distribuída entre os vários *Pistons*, cada qual recebendo uma parte diferente da mensagem. Para garantir que as esponjas dependam da mensagem inteira, o *Motorist* realiza o chamado ***knot***, em que concatena parte do estado de todos os *Pistons* em uma mensagem resultante que é inserida por inteira em cada *Piston*.

Uma sessão pode ser iniciada com uma *Session Unique Value*(SUV) composta de uma **chave secreta** e de um ***nonce***. A chave possui um tamanho máximo, porém não há limite para o tamanho do *nonce*. Uma interface de referência [J. Wetzels et al 2015] para o *Motorist* é:

$$\text{Motorist}[f, \Pi, W, c, \tau]$$

Sendo f a função de permutação a ser aplicada nos estados internos dos *Pistons*, Π o número de *Pistons*, W a unidade de alinhamento para o estado, capacidade c e tamanho τ de rótulo. *Motorist* possui quatro rotinas, sendo elas:

- *Motorist.StartEngine*: Inicializa a sessão com um *SUV* fornecido.
- *Motorist.Wrap*: Cifra ou decifra uma mensagen fornecida.
- *Motorist.MakeKnot*: Realiza um *knot* com o estado atual de todos os *Pistons*. No caso de haver mais de um *Piston*, ou a propriedade de esquecimento da sessão for requisitada, este procedimento será executado pelas rotinas *Motorist.StartEngine* e *Motorist.Wrap*.
- *Motorist.HandleTag*: Produz um novo rótulo ou verifica um rótulo fornecido. No caso de um novo rótulo ter sido requisitado ou um ter sido requisitada a validação de um rótulo fornecido, este procedimento será executado pelas rotinas *Motorist.StartEngine* e *Motorist.Wrap*.

4.2 Engine

Engine é uma construção suporte para o *Motorist*, responsável por gerenciar todos os *Pistons*, separando de maneira correta os dados de entrada e concatenando as várias saídas de cada *Piston*. Uma interface de referência [J. Wetzels et al 2015] para a *Engine* é:

$$\text{Engine}[\Pi, \textit{Pistons}]$$

Sendo Π o número de instâncias no vetor *Pistons*. *Engine* possui cinco rotinas, sendo elas:

- *Engine.Crypt*: Induz todo *Piston* a cifrar ou decifrar dados da entrada e retornar a saída correspondente.
- *Engine.Inject*: Induz todo *Piston* a injetar metadados da entrada.
- *Engine.Spark*: Induz todo *Piston* a aplicar a função de permutação f.
- *Engine.GetTag*: Induz todo *Piston* a extrair parte de seu estado para ser usado como rótulo.
- *Engine.InjectCollective*: Induz todo *Piston* a injetar dados de entrada de forma integral, sem particionar entre os Π *Pistons*.

4.3 Piston

Piston é uma construção *duplex* do tipo *FKSD* com um estado de largura b *bits*, equipado com uma função de permutação f. O estado é inicialmente zerado, até ser inicializado com uma *SUV*. Uma interface de referência [J. Wetzels et al 2015] para o *Piston* é:

$$\text{Piston}[f, r_s, r_a]$$

Sendo f a função de permutação a ser aplicada no estado, r_s o índice da taxa de compressão e r_a o índice da taxa de absorção, com $r_s \leq r_a < b$.

O estado deve ser preenchido com dados do texto claro até o indice r_s, e depois preenchido com metadados até o indíce r_a. Os *bytes* restantes correspondem à capacidade, e alguns são utilizados para armazenar informações do estado como a *flag* de fim de mensagem (EOM), o índice do estado contendo o último *byte* inserido (*Crypt End*), e o início e fim da injeção (*Inject Start* e *Inject End*) que indicam o início e o fim do trecho de *bytes* inseridos de metadados no estado.

Sempre que o *Piston* tiver o estado inteiro preenchido por dados, ele deverá aplicar a função de permutação f e voltar a absorver mais dados. O *Piston* possui quatro rotinas, sendo elas:

- *Piston.Crypt*: Cifra texto claro inserindo o criptograma resultante no estado, ou decifra criptograma inserindo o texto claro resultante. Em ambos os casos o indíce *Crypt End* é atualizado informando onde foi o último *byte* inserido, que será sempre menor que r_s.
- *Piston.Inject*: Insere metadados no estado, atualizando os indíces *Inject Start* e *Inject End*. A quantidade inserida pode ser no máximo r_a.
- *Piston.Spark*: Aplica a função de permutação f no estado, e atualiza a *flag* de *EOM* indicando se a mensagem terminou ou não.
- *Piston.GetTag*: Extrai parte do estado para ser usado como rótulo de autenticação.

5. Implementação

As rotinas do *Motorist* e *Engine* são de controle e são executadas poucas vezes durante uma sessão, portanto otimizá-las resultaria em um pequeno ganho de desempenho. De fato, a própria interface de implementações no repositório oficial [G. Bertoni et al. 2015d] recomenda apenas que as **funções que interagem diretamente com o estado interno** e a ***Keccak-***p em si sejam implementadas de maneira otimizada. São elas:

- *Initialize*: Inicializa o estado interno com *bytes* zero.
- *AddByte* e *AddBytes*: Adiciona *bytes* no estado interno através da operação **XOR**.
- *OverwriteBytes* e *OverwriteWithZeroes*: Sobrescreve *bytes* no estado interno com *bytes* zero ou com *bytes* fornecidos.
- *ExtractBytes* e *ExtractAndAddBytes*: Extrai *bytes* do estado interno, podendo também adicionar *bytes* ao mesmo tempo.
- *Permute*: A função de permutação *Keccak-*p.

Tais funções foram então implementadas em ***ARM assembly*** e também em ***NEON intrinsics*** [ARM Holdings 2015]. Como as instâncias nomeadas fazem uso de ambas ***Keccak-***$p[800]$ e ***Keccak-***$p[1600]$, foi necessário implementá-las para cada especificação, totalizando quatro implementações de alto desempenho.

Keyak e *Motorist* foram implementadas de maneira integral como mostrado na teoria, com exceção de alguns paramêtros que foram fixados por serem comuns entre as instâncias nomeadas, como o número n_r de rodadas e tamanho τ do rótulo. A entidade *Engine* foi separada em duas entidades de acordo com o tamanho de *bits* com qual ela

opera, a fim de evitar executar operações de condição (*branches*). Já a entidade *Piston* foi implementada dentro da entidade *Engine* para diminuir o número de chamadas internas, e facilitar uma possível implementação paralela futura.

Para verificar a correção do código, foi usado uma mesma **bateria de testes** fornecida pelo código de referência dos autores do algoritmo [G. Bertoni et al. 2015c]. Da mesma forma, foi integrado um **código já existente** de *Assembly* de *ARMv7* no projeto [G. Bertoni et al. 2015d], a fim de verificação de desempenho.

O algoritmo foi implementado **a partir do zero** em ***C***, devido ao maior desempenho da linguagem e fácil integração com linguagem de montagem (*assembly*). Todas as implementações foram compiladas com *flag* de otimização ***O2*** em todos os arquivos fonte, tanto *C* quanto *assembly*.

O código produzido em linguagem de montagem *ARMv8* utilizou dos novos recursos da arquitetura ***AArch64***, como o maior número de registradores vetoriais e as novas operações do pacote de instruções ***SIMD***, como transposição de vetores para otimizar movimentação de dados entre registradores. Cada registrador vetorial possui tamanho de 128 *bits* o que permite guardar quatro e duas palavras quando a largura b for 800 e 1600 respectivamente. Os registradores gerais possuem 64 *bits*, podendo suportar uma palavra por registrador no caso de $b = 1600$, que não era possível em registradores de 32 *bits* em *ARMv7*.

6. Resultados

Os resultados mostrados na **Tabela 1** foram obtidos por média simples de cem execuções para cada especificação. Os números de ciclos foram obtidos utilizando operações em linguagem de montagem específicas para cada arquitetura. As implementações para *ARMv7* foram executadas em um *Cortex-A8* e as implementações para *ARMv8* foram executadas em um *Cortex-A53*. Ambos dispositivos executavam sistema operacional ***ArchLinux***, o que permitiu compilar nativamente as implementações.

Os resultados da instância nomeada *Lunar Keyak* foram omitidas por serem de ordem muito maior que as outras e por apresentarem diferenças de desempenho entre implementações redundantes com as outras instâncias. Observa-se pela **Tabela 1** que o algoritmo fica naturalmente mais rápido sendo implementado em uma arquitetura *64 bits* como *ARMv8* do que em uma arquitetura *32 bits* como o *ARMv7*, como é possível observar na redução de número de ciclos na implementação em *C* do *ARMv7* para o *ARMv8*.

As implementações em *C* e *NEON intrinsincs* para ambas arquiteturas foram desenvolvidas neste trabalho, junto com a implementação *assembly* para *ARMv8*, aparecendo em itálico na **Tabela 1**. **A implementação *assembly* para *ARMv7* foi retirada do código de referência dos autores [G. Bertoni et al. 2015d] para nível de comparação**, e aparece em negrito na **Tabela 1**.

Tabela 1. Desempenho entre implementações - *Keyak Wrap*

River Keyak			
Processador	Implementação	Ciclos/*byte*	Ganho
ARMv7	*C*	249.88	-
	NEON Intrinsics	237.42	5,25%
	Assembly	26.87	829,95%
ARMv8	*C*	232.05	-
	NEON Intrinsics	63.45	256,72%
	Assembly	36.87	529,37%

Lake Keyak			
Processador	Implementação	Ciclos/*byte*	Ganho
ARMv7	*C*	165.35	-
	NEON Intrinsics	84.53	95,61%
	Assembly	15.10	995,03%
ARMv8	*C*	106.12	-
	NEON Intrinsics	19.19	453,00%
	Assembly	15.52	583.76%

Sea Keyak			
Processador	Implementação	Ciclos/*byte*	Ganho
ARMv7	*C*	178.20	-
	NEON Intrinsics	93.89	89.79%
	Assembly	15.97	1015.84%
ARMv8	*C*	116.03	-
	NEON Intrinsics	20.27	472.42%
	Assembly	16.37	608.79%

Ocean Keyak			
Processador	Implementação	Ciclos/*byte*	Ganho
ARMv7	*C*	202.29	-
	NEON Intrinsics	106.68	89.62%
	Assembly	17.79	1037.09%
ARMv8	*C*	132.45	-
	NEON Intrinsics	22.78	481.43%
	Assembly	18.45	617.89%

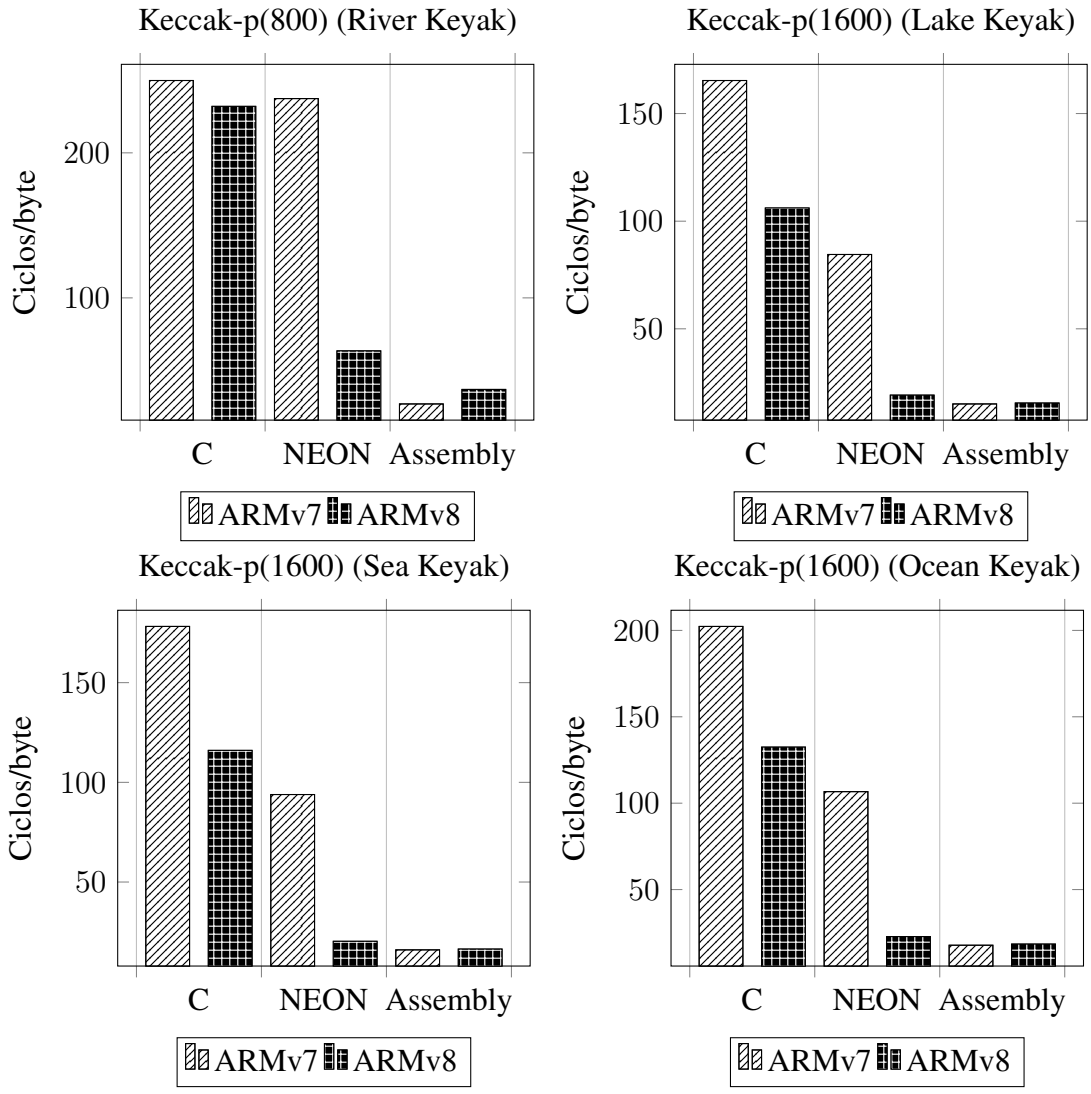


Figura 3. Gráficos dos resultados da Tabela 1.

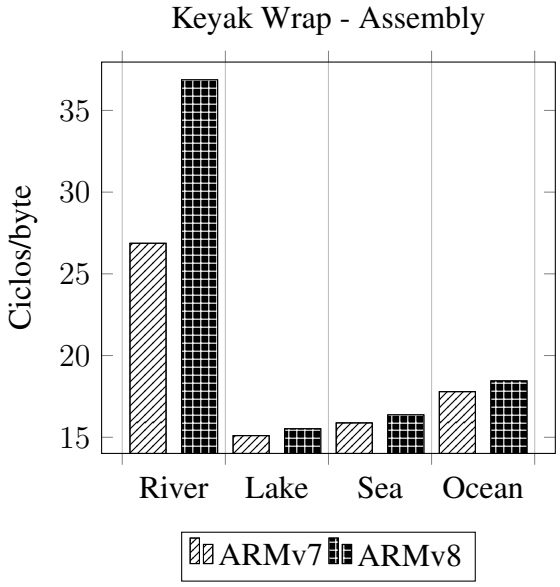


Figura 4. Gráfico comparando implementações em linguagem de montagem.

7. Conclusão

Os resultados da **Figura 3** mostram que a implementação em *NEON Intrinsics*, comparado à implementação em *C*, mostrou um alto desempenho em *ARMv8* nas instâncias que utilizam da *Keccak-*$p[1600]$, porém o desempenho em *ARMv7* foi relativamente inferior. Os resultados comprovam a melhor eficiência do código escrito em *NEON Intrinsics* para *ARMv8*. Já a instância que utiliza da *Keccak-*$p[800]$, *River Keyak*, teve uma melhora insignificativa em *ARMv7* e regular em *ARMv8*.

Para a implementação em código *assembly*, os resultados da **Figura 4** mostram que o desempenho obtido em *ARMv8* foi sempre marginalmente pior do que em *ARMv7* para as instâncias que utilizam da Keccak-$p[1600]$. Novamente na *River Keyak*, é possível perceber que ambas implementações possuem desempenho ineficiente, mas com a implementação em *ARMv8* gastando quase 40% mais ciclos que a de referência em *ARMv7*, apesar de ainda ser melhor do que as implementações em *C* e a *NEON Intrinsics* para *ARMv8*.

Como dito anteriormente na **Seção 2**, a implementação de referência [G. Bertoni et al. 2015d] utiliza do conjunto de instruções vetoriais apenas para a Keccak-$p[1600]$, sendo a Keccak-$p[800]$ implementada apenas com instruções regulares de montagem da arquitetura *ARMv7*, uma vez que é projetada para dispositivos com recursos limitados que não possuem suporte a *NEON*. As operações vetoriais acabam não sendo eficientes na Keccak-$p[800]$ devido à constante manipulação de palavras para montar os registradores vetoriais.

Assim, é possível que uma implementação que **não** use de operações vetoriais para Keccak-$p[800]$ tenha desempenho superior comparada à desenvolvida neste trabalho. Resta também implementar rotinas específicas para as instâncias com múltiplas esponjas, utilizando **paralelização** para otimizar seu desempenho. Tais tarefas podem vir a ser realizadas em trabalhos futuros.

Referências

G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche (2011). *Cryptographic sponge functions*. http://sponge.noekeon.org/

G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche (2013). *The Keccak sponge function family*. http://keccak.noekeon.org/

G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche (2013). *The Keccak implementation overview*. http://keccak.noekeon.org/Keccak-reference-3.0.pdf

G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer (2015). *Keyak Specification*. http://keyak.noekeon.org/

G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer (2015). *Keyakv2 CAESAR Submission*. http://keyak.noekeon.org/Keyak-2.0.pdf

G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche (2011). *Duplexing the sponge: single-pass authenticated encryption and other applications*. Selected Areas in Cryptography (SAC).

G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer (2015). *Keccak Tools software*. https://github.com/gvanas/KeccakTools

G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer (2015). *Keccak Code Package*. https://github.com/gvanas/KeccakCodePackage

National Institute of Standards and Technology (2014). *SHA-3 Standard: Permutation-Based Hash And Extendable-Output Functions*. http://www.nist.gov/manuscript-publication-search.cfm?pub_id=919061

J.P. Aumasson and D. Khovratovich (2009) *First analysis of Keccak*. http://131002.net/data/papers/AK09.pdf

K. Graves (2010) *Certified Ethical Hacker Study Guide, 1st Edition*.

F. Callegati, W. Cerroni and M. Ramilli (2009). *IEEE Xplore - Man-in-the-Middle Attack to the HTTPS Protocol*

J. Wetzels and W. Bokslag (2015) *Sponges and Engines: An introduction to Keccak and Keyak*.

K. Paterson (2013) *Authenticated Encryption in TLS*.

H. Krawczyk (2001) *The Order of Encryption and Authentication for Protecting Communications (Or: How Secure is SSL?)*.

B. Mennink, R. Reyhanitabar, and D. Vizar (2015) *Security of Full-State Keyed and Duplex Sponge: Applications to Authenticated Encryption*.

ARM Holdings (2015) *A64 Advanced SIMD Vector Instructions*. http://infocenter.arm.com

GitHUB (2016) *KeccakP-800 and KeccakP-1600 implementation for the ARM AArch64*. https://github.com/gvanas/KeccakCodePackage/pull/23

GitHUB (2016) *KeccakP1600 ARMv7A possible bug*. https://github.com/gvanas/KeccakCodePackage/issues/22

Mineração individual de *bitcoins* e *litecoins* no mundo

Guilherme Albuquerque B. Silva[1], Carlo Kleber da S. Rodrigues[1]

[1]Faculdade de Tecnologia e Ciências Sociais Aplicadas – Centro Universitário de Brasília (UniCEUB) – Brasília – DF – Brasil.

guilhermealbuquerque1@hotmail.com, carlokleber@gmail.com

Abstract. *This paper analyzes the solo mining process of the bitcoin and litecoin cryptocurrencies all over the world. The goal is to identify which one turns out to be the most profitable one. To this end, the metric named as energy efficiency is used and three continents are taken as mining regions: America, Asia and Europe. Three variables are considered to compute the above metric: the hash-processing rate of the hardware, the electrical energy consumption of the hardware, and the price of the kilowatt-hour of the mining region. The final results suggest that mining litecoins turns to be more profitable than mining bitcoins. Lastly, general conclusions and future avenues come at the end of this paper.*

Resumo. *Este artigo analisa a mineração individual das criptomoedas bitcoin e litecoin no mundo. O objetivo é identificar aquela que é economicamente mais rentável. Para tanto, utiliza-se a métrica denominada eficiência energética e consideram-se três continentes como regiões de mineração de cada criptomoeada: América, Ásia e Europa. O cálculo dessa métrica considera três variáveis: a taxa de processamento de hashes do hardware, o consumo de energia elétrica do hardware em watts, e o valor do quilowatt-hora da região de mineração. Os resultados finais indicam que minerar litecoins é mais rentável que minerar bitcoins. Conclusões gerais e possíveis trabalhos futuros finalizam este artigo.*

1. Introdução

Criptomoedas são formas de dinheiro virtual que contam com métodos criptográficos para assegurarem a geração e a distribuição de valores de forma segura por meio de uma rede de computadores [Bank for International Settlements, 2015]. As criptomoedas são livres de interferências cambiais de governos e dão relativa segurança às transações financeiras realizadas com elas. Duas criptomoedas bem populares e comercialmente atrativas são *bitcoin* e *litecoin* [Murphy, 2013].

O principal processo de obtenção de criptomoedas não advindo de uma transação financeira é denominado de *mineração*. Esse processo consiste em adquirir digitalmente a posse de um certo número de criptomoedas a partir de um esforço computacional despendido pelo *hardware* de uma pessoa (i.e., *minerador*) que realiza esse processo [Nakamoto, 2008].

O *bitcoin* foi criado em 2008 [Nakamoto, 2008] e alcançou a expressiva marca de USD 4,05 bilhões em seu mercado de transações no ano de 2015 [White, 2015]. Esse mercado é, no entanto, ainda instável. Por exemplo, o *bitcoin* teve uma valorização de 6.000% em 2013, alcançando a marca de USD 1.250,00 por *bitcoin*. No ano seguinte, porém, perdeu 2/3 do seu valor. Atualmente, um *bitcoin* (ou 1,0 BTC) é cotado em USD 454,90.

Dentre as outras criptomoedas que surgiram desde a criação do *bitcoin*, destaca-se a criptomoeda *litecoin*. Em 2013, essa criptomoeda teve um crescimento de 400% e, em seu ápice, chegou a ser cotada em USD 48,05. Trata-se de uma alternativa que possui um processo de mineração mais fácil e mais rápido que aquele do *bitcoin*. Além disso, como ainda não existem muitas opções de *hardware* especializado para a mineração de *litecoins*, há uma competição mais equânime entre os possíveis mineradores, pois estes tendem a utilizar o mesmo *hardware*. Diante disso, surge então a seguinte pergunta: a despeito da complexidade computacional e sob o ponto de vista da rentabilidade econômica, é mais vantajoso minerar *litecoins* que *bitcoins*?

Dentro deste contexto, este trabalho objetiva analisar a rentabilidade econômica da mineração individual das criptomoedas *bitcoin* e *litecoin*. Essa análise baseia-se na avaliação da métrica *eficiência energética* nos países com maiores Produtos Internos Brutos (PIBs) das seguintes regiões: América, Europa e Ásia, almejando-se uma rentabilidade de 0,10 USD/hora. Três variáveis são consideradas para o cálculo dessa métrica: a taxa de processamento de *hashes* do *hardware* empregado, o consumo de energia elétrica em watts, e o valor em dólares do quilowatt-hora (kWh).

A organização do restante deste texto é descrita a seguir. A Seção 2 apresenta resumidamente o protocolo *Bitcoin* e as suas principais diferenças com relação ao protocolo *Litecoin*. Os trabalhos relacionados são brevemente discorridos na Seção 3. Na Seção 4, são apresentados e discutidos os experimentos e os resultados. Por fim, conclusões gerais e trabalhos futuros aparecem na Seção 5.

2. Fundamentos

2.1 Protocolo Bitcoin

O protocolo *Bitcoin* foi originalmente anunciado em um artigo publicado em novembro de 2008, o qual definiu uma forma de criptomoeda que funciona de forma pseudônima e sem depender da confiança em qualquer usuário do sistema [Nakamoto, 2008]. Esse protocolo foi desenvolvido considerando o paradigma de uma rede *peer-to-peer* (P2P) de alcance mundial, resultando em um sistema de transações financeiras de escala global.

A arquitetura do protocolo *Bitcoin* é baseada na implementação de um livro-razão ou *ledger* público, onde todos os usuários têm acesso a todas as transações. No entanto, como são utilizados números em vez de nomes, a privacidade é preservada [Nakamoto, 2008]. Sempre que ocorre uma transação na rede, tanto o *ledger* do beneficiário quanto o do pagador vão ser alterados. De forma semelhante, o restante dos nós da rede atualiza seus *ledgers* para conter essa transação.

2.2 Conceito de Block-chain

Quando o pagador de uma transação em *bitcoins* envia certa quantia para outro usuário é como se esse pagador estivesse assinando um documento público que atesta a transferência de posse daquela quantia para o outro usuário. A assinatura dessa transação é realizada pelo conceito de par de chaves assimétricas, onde um mesmo usuário da rede possui duas sequências de dígitos únicos que formam as chaves pública e privada, respectivamente.

O sistema de chaves criptográficas assimétricas garante a autenticidade de quem e para quem a transação é realizada, mas não garante que um mesmo usuário não possa gastar aquela mesma quantia mais de uma vez em outras transações. Dado que se decorre um tempo para propagar-se o *ledger* atualizado, qualquer usuário poderia teoricamente realizar outra transação enquanto a antiga não se propagou por toda a rede. Nesse caso, a rede teria dificuldades em diferenciar qual transação seria a legítima.

A forma encontrada pelo protocolo para solucionar o problema anteriormente descrito se dá pelo emprego do conceito de *block-chain*, explicado a seguir. Propagam-se as transações recém realizadas, mas ainda não validadas, por toda a rede. Essas transações são então agrupadas em blocos. Cada bloco é validado pelos mineradores no processo de mineração.

Após validado, o bloco é então adicionado a uma corrente (cadeia) de blocos, que leva desde o primeiro bloco, contendo a primeira transação já realizada na história do *Bitcoin*, até a transação mais recentemente validada. Os blocos que não forem validados pelos mineradores são descartados. Essa corrente é denominada de *block-chain* e é a base de informação para implementar o *ledger* público.

2.3 Conceito de Proof-of-work

Nakamoto (2008) definiu que o processo matemático realizado pelo minerador ocorre por meio do algoritmo *hash* criptográfico *SHA-256*. Esse algoritmo faz com que minerador precise descobrir um número inteiro de 4 bytes, denominado de *nonce*, capaz de satisfazer a uma desigualdade (inequação) expressa em função desse algoritmo.

O método de descobrimento usado pelo minerador é baseado em múltiplas tentativas, e a condição de desigualdade é estabelecida considerando-se um valor máximo, denominado de *target difficulty*, que é ajustado pelo algoritmo para garantir que, em média, apenas um bloco de transações válidas seja adicionado à *block-chain* a cada 10 minutos.

O *nonce* descoberto, também chamado de *golden nonce*, é a *proof-of-work* (prova de trabalho) que o cálculo iterativo por tentativas foi de fato realizado e que o bloco pode então ser adicionado à *block-chain* [Rocha e Rodrigues, 2016]. Para incentivar os mineradores a validar os blocos, o protocolo prevê uma *reward* (recompensa) em *bitcoins* para quem primeiro conseguir encontrar o *golden nonce*. Essa recompensa não tem um valor fixo, sendo alterada de tempos em tempos [Rosenfeld, 2016].

Quando o *Bitcoin* foi lançado, o seu *reward* era de 50,0 BTC. Esse valor de *reward* é dividido por 2 a cada 230.000 blocos minerados na rede ou, aproximadamente, a cada quatro anos, já que cada bloco leva cerca de dez minutos para ser minerado. Esse ajuste é conhecido como *halving*. O *Bitcoin* já passou por um *halving* e, atualmente, cada bloco minerado é recompensado com 25,0 BTC.

2.4 Protocolo Litecoin

O protocolo *Litecoin* foi desenvolvido por Charles Lee, então funcionário da empresa Google, em outubro de 2011. É um projeto de código aberto que, na época, foi lançado em plataformas de desenvolvimento colaborativas [Bradbury, 2013]. Esse protocolo possui o tempo de mineração de cada bloco estimado em 2,5 minutos, sendo o valor da *reward* atual igual a 25,0 *litecoins*. Semelhantemente ao protocolo *Bitcoin*, o *Litecoin* também passa pelo

evento de *halving* a cada quatro anos aproximadamente. O algoritmo usado pelo *Litecoin* para estabelecer o processo matemático da mineração é o *Scrypt* [Percival, 2009].

É preciso ressaltar que a ideia do protocolo *Litecoin* não foi a de substituir a mineração de *bitcoins*, mas permitir que ambas criptomoedas coexistam nos mercados financeiros. Comparativamente, a principal diferença entre os algoritmos de *proof-of-work* dos protocolos é que o *SHA-256* (do *Bitcoin*) é focado em uso intensivo do processador, enquanto que *Scrypt* (do *Litecoin*) é focado no uso intensivo de memória. Disto resulta a necessidade de *hardwares* especializados para mineração de *bitcoins*, e de *hardwares* de uso geral para mineração de *litecoins*.

Para encerrar esta seção, cabe ainda mencionar que existem basicamente três formas de se obter criptomoedas: por meio de uma transação entre criptomoedas; comprando-se criptomoedas com dinheiro real; e, por último, por meio do processo de mineração. Esta última pode ser ainda realizada de duas formas: de maneira coletiva (i.e., *pools* de mineradores) e por meio da *mineração individual*, i.e., utilizando-se um único *hardware*. O foco deste trabalho é a mineração individual.

3. Trabalhos Relacionados

A análise formal da rentabilidade da mineração individual de criptomoedas ainda é um tema relativamente recente na literatura. Tendo-se conhecimento desta pouca exploração científica, esta seção busca discorrer brevemente sobre alguns dos trabalhos mais recentes da literatura que contribuíram ou se relacionaram, mesmo de forma indireta, com o objetivo deste trabalho.

Rosenfeld (2011) propõe a compreender os aspectos da rentabilidade por mineração em *pools*. O trabalho conclui que, por causa da alta variância nas recompensas da mineração individual, a necessidade de mineração em *pools* se faz importante e não pode ser desconsiderada.

O trabalho de Kiayias e Panagiotakos (2016) compara a eficiência da tecnologia de *block-chain* sob a ótica da segurança, considerando as alternativas mais modernas para realizar transações. O trabalho consegue apontar que, nos piores cenário de segurança, os protocolos mais modernos, como o denominado GHOST, têm um desempenho pior ou na melhor das hipóteses similar à *block-chain*.

Luther (2015) investiga o emprego de criptomoedas nos distintos mercados financeiros. O trabalho conclui que o *Bitcoin* representa um real avanço tecnológico no processamento de pagamentos, embora seja também verdade que o desenvolvimento de outras criptomoedas possa vir, em algum momento, tornar o *Bitcoin* obsoleto, passando este a ser reconhecido apenas como o precursor das criptomoedas.

O trabalho de Chávez e Rodrigues (2015) apresenta formas de decidir analiticamente quando é mais vantajoso trocar de *pools* a fim de se manter a rentabilidade considerada. O trabalho conclui que a mineração considerando o salto entre *pools* (i.e., troca dinâmica) é mais eficiente que a mineração considerando um único *pool*.

Pazmiño e Rodrigues (2015) avaliam o tempo de verificação de transações e, neste contexto, propõe um esquema para a divisão da base de dados de um nó da rede *bitcoin*, considerando o *hardware* disponível localmente no usuário. Os resultados finais são

numericamente atrativos, resultando em otimizações de até 71,42% no tempo de verificação de transação.

Por fim, o trabalho de Rocha e Rodrigues (2016) tem um viés de análise baseado em metodologias de desenvolvimento de *software* e, precipuamente, apresenta a modelagem do processo de negócio do sistema *Bitcoin*. Para tanto, são empregadas técnicas de *Engenharia de Software* e de *Business Process Model and* Notation. Os modelos desenvolvidos permitem mapear problemas críticos do sistema, explicitando em quais etapas do processo de negócio eles incidem.

4. Rentabilidade para Bitcoin e Litecoin

4.1 Definição de rentabilidade

Tanto para o *Bitcoin* quanto para o *Litecoin*, a *target difficulty*, *D*, refere-se à complexidade para se minerar um bloco, sendo ajustada de forma que qualquer número de tentativas para encontrar o *golden nonce* tenha sempre a razão de $\frac{1}{2^{32}*D}$ de chance de sucesso.

O valor atual de *D* para o protocolo *Bitcoin* é 194.254.820.283, e para o protocolo *Litecoin* é 47.760. A taxa com que um *hardware* consegue testar valores para descobrir o *golden nonce* é denominada de *hashrate*, representada por *h*.

Considerando-se então o tempo de mineração *t*, um *hardware* pode realizar um total de *h*t* tentativas (ou *hashes*) para descobrir o valor do *golden nonce*. Ainda, a quantidade de *bitcoins Q* a ser recebida, em um certo intervalo de tempo *t* de mineração, pode ser estimada pela Equação 1 [Rosenfeld, 2011].

$$Q = \frac{h * t}{2^{32} * D} * reward \qquad (1)$$

O valor da *Receita total* pode ser obtido pela Equação 2, aqui definida assim como as demais equações que seguem. Esse valor é calculado levando-se em conta o valor da *reward* então recebido, bem como o valor de cotação atual em dólares da criptomoeda considerada, representado por V_c.

$$Receita\ total = reward * V_c \qquad (2)$$

A rentabilidade total esperada é calculada como *Receita total* menos *Despesa*. Por sua vez, o valor de *Despesa* é obtido pela multiplicação da *Despesa por hora*, D_h, pelo tempo *t* de mineração em horas. Explica-se que D_h está relacionada ao valor do kWh da região, V_{kWh}, onde a mineração em si é realizada. Para esse cálculo (vide Equação 3), também se leva em conta a potência do equipamento (*hardware*) de mineração, *P*, dada em watts, que serve para o cômputo de quanta energia o equipamento precisa utilizar em uma hora.

$$D_h = \frac{P}{1000} * V_{kWh} \qquad (3)$$

Seja V_r = USD 0,10 por hora (i.e., USD 0,10/h) a rentabilidade almejada por hora. Esse valor é escolhido por ser compatível com o cenário atual da atividade de mineração, observado em *sites* populares da Internet que reportam sobre valores estimados para esse tipo de atividade. No entanto, é preciso esclarecer que esse valor absoluto não é importante

para efeito das conclusões e observações a serem alcançadas neste trabalho, pois aqui tenciona-se uma análise comparativa relativa e não absoluta.

Agora, a partir de V_r e de D_h, pode-se escrever a Equação 4 para obter-se o valor da *Receita por hora*, representada por R_h.

$$R_h = V_r + D_h \qquad (4)$$

Ainda, seja Q_h a quantidade de moedas por hora suficiente para alcançar a *Receita por hora*, representada por R_h. O cálculo de Q_h é obtido pela Equação 5.

$$Q_h = \frac{R_h}{V_c} \qquad (5)$$

Como mencionado, para se encontrar o *golden nonce* é necessário observar a medida de *hashrate* do equipamento. Isso para que o valor de *hashrate* seja suficiente para encontrar um bloco no tempo necessário para preservar a *Receita total*. Para tanto, é aqui considerado um valor de *hashrate* mínimo, representado por h_{min}. A Equação 6 apresenta a fórmula para obter esse valor. Essa equação é derivada a partir das Equações 1, 2, 3, 4 e 5, sendo t o período de tempo considerado de mineração.

$$h_{min} = \frac{2^{32} * D}{Q_h * t * reward} \qquad (6)$$

A métrica *eficiência energética EE*, definida neste trabalho, é medida em hash/J e representa quantos *hashes* um *hardware* padrão precisa calcular, utilizando 1,0 joule de energia, para que se preserve a rentabilidade almejada V_r. Pode-se então calcular *EE* a partir do valor de h_{min} (vide Equação 6) da região considerada e do valor de potência do equipamento empregado, conforme Equação 7.

$$EE = \frac{h_{min}}{\mathrm{P}} \qquad (7)$$

4.2 Mineração no Brasil

Segundo a Agência Nacional de Energia Elétrica (ANEEL), a taxa de kWh mais barata do Brasil é a da concessionária de energia de Santa Catarina (SC), conhecida como Cooperativa Pioneira de Eletrificação (COOPERA), no valor de R$ 0,25/kWh, conforme ilustrado na Tabela 1 [Agência Nacional de Energia Elétrica, 2016].

Tabela 1. As três operadoras mais caras e as mais baratas de energia no Brasil.

Sigla	Tarifa (kWh/R$)	Unidade da Federação
CEDRI	0,604	SP
UHENPAL	0,583	RS
CHESP	0,582	GO
CERSUL	0,293	SC
CEA	0,273	AP
COOPERA	0,250	SC

Considerando a cotação do dólar em R$ 3,61, tem-se então o valor de 0,069 USD/kWh. Por fim, pelo Sistema Internacional de Unidades (SI), sabe-se que: 1,0 watt = 1,0 J/s (Joule/segundo) e, ainda, 1 hora = 3.600 segundos.

Admitindo-se então o custo de energia de kWh dado anteriormente e um equipamento que não consuma mais do que 100,0 W e, ainda, substituindo-se os valores do cenário brasileiro na Equação 6, chega-se a um valor de h_{min} de aproximadamente 2.178,4 Ghash/s para garantir a rentabilidade almejada de 0,10 USD/h. Aplicando-se este valor de $h_{\min}$ na Equação 7, chega-se então ao seguinte resultado: EE = 21,78 Ghash/J, conforme pode ser visto na Figura 1.

Considerando-se agora o protocolo *Litecoin* e um raciocínio análogo ao que acabou de ser descrito para o protocolo *Bitcoin*, tem-se os seguintes resultados: h_{min} = 63,28 Mhash/s e EE = 0,632 Mhash/J, para o país Brasil, conforme é apresentado na Figura 2.

Ante o exposto, evidencia-se então que a eficiência energética para o *Litecoin* no Brasil é menor que a do *Bitcoin*. Portanto, pode-se concluir que a mineração de *litecoins* é mais rentável que a de *bitcoins* no Brasil.

4.3 Mineração no mundo

Para analisar o cenário mundial, escolhem-se três importantes regiões econômicas: América, Ásia e Europa. Dessas regiões, são escolhidos os países de maiores PIBs. Os dados dos PIBs dos países selecionados estão na Tabela 2 [Fundo Monetário Internacional, 2016]. Os resultados da EE estão nas Figuras 1 e 2. A partir dessas figuras, é possível perceber que os valores resultantes para o protocolo *Bitcoin* são bem superiores àqueles obtidos para o protocolo *Litecoin*.

Tabela 2. Países considerados e seus respectivos valores de PIB e kWh.

País	PIB (milhões USD)	Valor kWh (USD)
Brasil	1.722.589	0,069
México	1.144.334	0,108
Estados Unidos	17.947.000	0,094
Canadá	1.552.386	0,070
Alemanha	3.357.614	0,015
França	2.421.560	0,089
Reino Unido	2.849.345	0,141
Japão	4.123.258	0,177
China	10.982.829	0,113
Indonésia	858.953	0,164

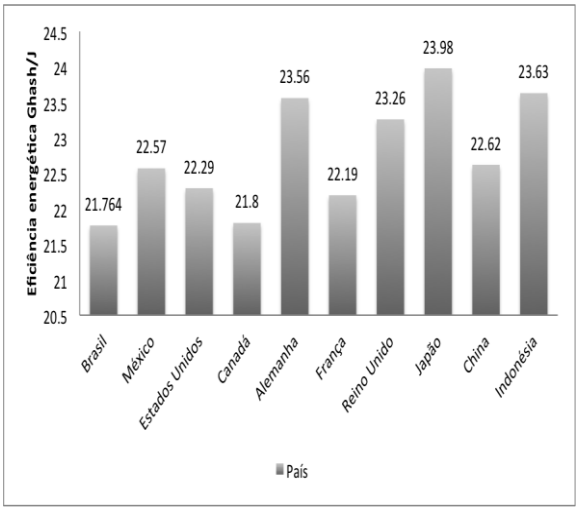


Figura 1. Eficiência energética do *Bitcoin.*

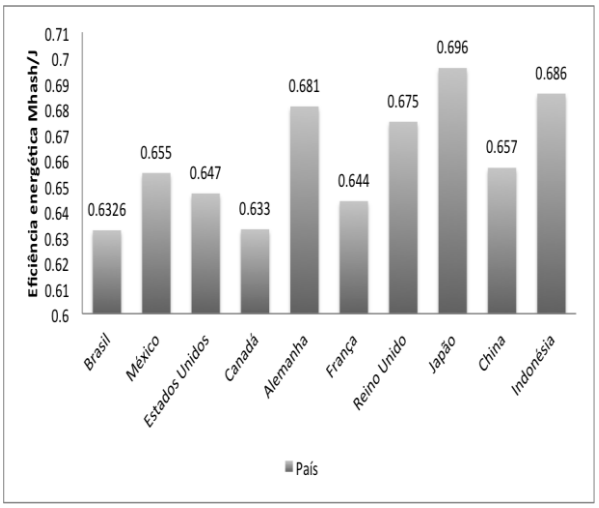


Figura 2. Eficiência energética do *Litecoin.*

Considerando então os resultados obtidos, pode-se concluir que no cenário mundial, assim como observado para a região brasileira anteriormente, a mineração de *litecoins* é mais rentável que a de *bitcoins*.

4.4 Tempo de mineração por bloco

Ressalta-se que, para mineração individual, o valor da *reward* é pago de uma só vez e apenas ao final do processo de mineração, quando o valor do *golden nonce* é encontrado. Ou seja, apenas ao término da mineração é que o minerador recebe sua *reward,* a qual corresponde atualmente a 25 moedas, tanto para o *Bitcoin* quanto para o *Litecoin*.

Assim, o tempo médio, $t_{médio}$, necessário para conseguir minerar um bloco se faz de importante consideração na mineração individual. Para se estimar $t_{médio}$ é necessário considerar a *target difficulty*, *D*, da criptomoeda considerada, e o valor da *hashrate, h,* do equipamento, resultando na Equação 8.

$$t_{médio} = \frac{2^{32} * D}{h} \qquad (8)$$

Para o país Brasil e o valor de *h* sendo a *hashrate* mínima, i.e., $h = h_{min} = 2.178\text{x}10^9$ hash/s, e com $D = 194.254.820.283$, a resolução da Equação 8 diz então que um bloco do protocolo *Bitcoin* seria minerado em aproximadamente $3{,}83\text{x}10^8$ s. Convertendo-se esse valor em dias, tem-se aproximadamente 4.323 dias, ou seja, cerca de 12 anos, para se minerar um bloco e receber 25 moedas (*reward* atual). Com a cotação atual da criptomoeda, seriam USD 11.372,5 de lucro.

Considerando agora o mesmo cenário de análise e substituindo-se os valores de *D* e h_{min} para o protocolo *Litecoin*, $t_{médio}$ seria de apenas 37,5 dias, e o valor das 25 moedas seria de USD 96,25. Ou seja, o tempo necessário pelo *Bitcoin* é cerca de 115,28 vezes maior para obter as 25 moedas de *reward*. Nesse tempo, o *Litecoin* obteria como *reward* 2.882 moedas, ou USD 11.095,7 de lucro, ficando bem próximo daquele provido pelo *Bitcoin*.

Além disso, o resultado do cálculo de $t_{médio}$ alerta para o fato de que, no período de 12 anos, tanto a criptomoeda *bitcoin* como a criptomoeda *litecoin* teriam passado por três *halvings*, o que faria o valor de 25 moedas se tornar um oitavo do valor original, ou seja,

3,12 moedas apenas. Nesse sentido, observe que o cálculo de *EE* proposto não leva em conta os eventos de *halvings* ao longo do tempo de mineração.

Para efeito de análise, no caso do protocolo *Bitcoin*, com a intenção de evitar-se o evento de *halving* e, portanto, garantir-se as 25 moedas de *reward*, admita o processo de mineração durando no máximo quatro anos (tempo limite para evitar-se o evento de *halving*), ou seja, um bloco deve ser minerado a cada 1.460 dias em média. A partir da manipulação das equações anteriores, é possível mostrar que a *EE* tem que ser aumentada em aproximadamente 296%, fazendo a taxa de hash/s do Brasil igual a 6.449 Ghash/s (ou 64,49 Ghash/J de eficiência energética).

Observe ainda que, ao longo de 12 anos, a dificuldade *D* e o valor da moeda podem sofrer variações significativas. Isso faz com que projeções em longo prazo sejam difíceis e geralmente imprecisas. Por outro lado, tendo em vista que, no cenário considerado, o ganho obtido pelo uso do protocolo *Litecoin* seria alcançado em cerca de apenas 37 dias, o evento de *halving* não influenciaria o lucro obtido no final do período estimado para minerar um bloco.

Sendo assim, as projeções para o *Bitcoin* de longo prazo tornam-se mais imprecisas que a do *Litecoin*. Isso leva à conclusão de que o retorno monetário advindo da mineração individual é bem mais incerto no caso do protocolo *Bitcoin*.

5. Conclusões e trabalhos futuros

Este artigo teve o objetivo de analisar a rentabilidade econômica da mineração individual de criptomoedas *bitcoins* e *litecoins*. Essa análise foi baseada na avaliação da métrica *eficiência energética* de cada criptomoeda. Três variáveis foram consideradas para o cálculo da métrica: a taxa de processamento de *hashes* do equipamento, o consumo de energia elétrica do equipamento, e o valor do quilowatt-hora da região da mineração.

Os seguintes resultados podem ser destacados: (1) para uma mesma rentabilidade econômica do processo de mineração individual, o protocolo *Litecoin* demanda menos *eficiência energética* que o protocolo *Bitcoin* para o país Brasil; (2) em países de outros continentes, ainda que os valores de kWh variem, a demanda inferior de *eficiência energética* do protocolo *Litecoin* em relação ao protocolo *Bitcoin* se mantém; (3) devido ao tempo necessário para mineração de um bloco de transações ser menor no protocolo *Litecoin*, as suas projeções de investimento a longo prazo são mais precisas e confiáveis que no caso do protocolo *Bitcoin*. Esses resultados levam à conclusão geral de que a mineração individual de *litecoins* é mais rentável e previsível que a de *bitcoins.*

Por fim, como trabalho futuro, sugerem-se o estudo e o desenvolvimento de métodos de *proof-of-work* para o protocolo *Bitcoin* que não sejam amparados exclusivamente no poder de processamento de *hashes* dos *hardwares* utilizados [Poon e Thaddeus, 2016]. Conjectura-se que, ao reduzir-se a demanda por esse poder de processamento, o consumo de energia elétrica diminui e, ainda, *hardwares* mais simples e, consequentemente, mais economicamente acessíveis, podem ser utilizados, resultando em um maior grau de competitividade do protocolo *Bitcoin* frente a outros protocolos de mesma finalidade.

Referências

Agência Nacional de Energia Elétrica (2016). "Ranking nacional de tarifas residenciais (grupo B1)". Disponível em: < http://www.aneel.gov.br/ranking-das-tarifas>. Acesso em: 21 jun. 2016.

Bank for International Settlements (2015). "Digital currencies". Disponível em: <https://www.bis.org/cpmi/publ/d137.pdf>. Acesso em: 9 jun. 2016.

Bradbury, D. (2013). "Litecoin founder Charles Lee on the origins and potential of the world's second largest cryptocurrency". Disponível em: <http://www.coindesk.com/litecoin-founder-charles-lee-on-the-origins-and-potential-of-the-worlds-second-largest-cryptocurrency/>. Acesso em: 20 de fev. 2016.

Chávez, J. J. G.; Rodrigues, C. K. S. (2015). A simple algorithm for hopping among Pools in the Bitcoin Mining Network. *The SIJ Transactions on Computer Networks & Communication Engineering (CNCE)*, v. 3, n. 2., p. 22-27.

Fundo Monetário Internacional (2016). "Report for Selected Country Groups and Subjects". Disponível em: <http://www.imf.org/external/pubs/ft/weo/2016/01/weodata/index.aspx>. Acesso em: 19 jun. 2016.

Kiayias, A.; Panagiotakos, G. (2016). "On Trees, Chains and Fast Transactions in the Blockchain". Disponível em: <https://eprint.iacr.org/2016/545>. Acesso em: 09 de jul. 2016.

Luther, W. J. (2015). "Bitcoin and the Future of Digital Payments". Disponível em: <http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2631314>. Acesso em: 10 fev. 2016.

Murphy, R. P. (2013). "The Economics of Bitcoin". *Library and Liberty*. Disponível em: <http://www.econlib.org/library/Columns/y2013/Murphybitcoin.html>. Acesso em: 10 fev. 2016.

Nakamoto, S. (2008). "Bitcoin: A Peer-to-Peer Electronic Cash System". Disponível em: <https://bitcoin.org/bitcoin.pdf>. Acesso em: 30 dez. 2015.

Pazmiño, J. E.; Rodrigues, C. K. S. (2015). Simply Dividing a Bitcoin Network Node May Reduce Transaction Verification Time. *The SIJ Transactions on Computer Networks & Communication Engineering (CNCE)*, v. 3, n. 2., p. 17-21.

Percival, C. (2009). "Stronger key derivation via sequential memory-hard functions". Disponível em: <http://www.tarsnap.com/scrypt/scrypt.pdf>. Acesso em: 30 jan. 2016.

Poon, J.; Thaddeus, D. (2016). "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments". Disponível em: <http://lightning.network/lightning-network-paper.pdf>. Acesso em: 25 jan. 2016.

Rocha, J. G.; Rodrigues, C.K.S. (2016). O processo de negócio do sistema de transações financeiras Bitcoin. *Universitas: Gestão e TI*, v.6, n. 1., p. 1-14.

Rosenfeld, M. (2011). "Analysis of Bitcoin Pooled Mining Reward Systems". Disponível em: <https://bitcoil.co.il/pool_analysis.pdf>. Acesso em: 30 jan. 2016.

White, L. H. (2015). The Market for Cryptocurrencies. *Cato Journal*, v. 35, n. 2., p. 383-402.

Caracterização Remota de Comportamento de Roteadores IPv6

**Rafael Almeida, Elverton Fazzion, Osvaldo Fonseca,
Dorgival Guedes, Wagner Meira, Ítalo Cunha**

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

{rlca,elverton,osvaldo.morais,dorgival,meira,cunha}@dcc.ufmg.br

Abstract. *Even though the IETF standardizes protocols and provides implementation guidelines, many implementation decisions are left to vendors and developers. The behavior of network devices are also dependent on their configuration. Over the years, researchers have developed many measurement techniques to characterize network devices. These characterization techniques are useful for network management, operation, troubleshooting, and security. With the depletion of IPv4 address space and increasing IPv6 adoption, characterization techniques for IPv6 devices are increasingly important. Unfortunately, characterization techniques for IPv6 devices remain incipient. In this work we propose and evaluate a new characterization technique for IPv6 devices along a network path that takes the behavior of intermediate devices into account.*

Resumo. *Apesar da IETF padronizar protocolos e prover diretivas de implementação, muitas decisões de implementação são deixadas a cargo de fabricantes e desenvolvedores. O comportamento de dispositivos também depende de suas configurações. Ao longo dos anos, pesquisadores desenvolveram várias técnicas de medição para caracterizar dispositivos de rede. Estas técnicas de caracterização de dispositivos são úteis para gerenciamento, operação, resolução de falhas e segurança de redes. Com o esgotamento dos endereços IPv4 e a crescente adoção do protocolo IPv6, técnicas de caracterização de dispositivos na Internet IPv6 são cada vez mais importantes. Porém, técnicas de caracterização para dispositivos IPv6 ainda são incipientes. Neste trabalho propomos e avaliamos uma nova técnica de caracterização de dispositivos IPv6 ao longo de um caminho na Internet que considera características de dispositivos intermediários.*

1. Introdução

A Internet é formada por milhares de redes autônomas interligadas. Essas redes, denominadas sistemas autônomos, podem ter diferentes políticas internas, que impactam decisões como a escolha de enlaces físicos, de dispositivos de interconexão e da configuração destes dispositivos. As políticas internas de um sistema autônomo, em geral, são privadas e consideradas segredos de negócios. No entanto, técnicas de caracterização e *scanning* (e.g. Nmap) podem expor tais políticas. Neste cenário, podemos destacar algumas questões relativas à segurança das redes: (1) A topologia interna de um sistema autônomo pode ser reconstruída através de técnicas de traceroute e IP *aliasing*. Essa

informação pode ser utilizada por um atacante para identificar pontos de falhas ou gargalos na rede. (2) Um atacante pode expor características dos dispositivos utilizados dentro de um sistema autônomo através de técnicas de coleta de assinaturas (*fingerprinting*) e utilizar tais informações para explorar vulnerabilidades conhecidas nos dispositivos identificados. Técnicas de coleta de assinaturas de dispositivos de rede devem ser estudadas, pois tem grande potencial de expor informações consideradas sigilosas pelos sistemas autônomos. O Nmap [Lyon 2009], por exemplo, é uma ferramenta que constrói a assinatura de um dispositivo de rede e através dessa informação infere o sistema operacional em execução no dispositivo.

Grande parte das técnicas de coleta de assinaturas levam em conta os diferentes comportamentos dos dispositivos de rede em diferentes situações, por exemplo, podem considerar as diferentes características dos cabeçalhos da camada de rede e da camada de transporte dos pacotes enviados pelo dispositivo alvo. O IETF (Internet Engineering Task Force) é uma entidade que propõe padrões de tecnologias e protocolos para a Internet, através dos RFCs (Request for Comments). No entanto, os padrões propostos pelo IETF podem deixar algumas decisões a cargo dos fabricantes de dispositivos, desenvolvedores e administradores de redes. Existe, portanto, uma heterogeneidade de comportamento entre os diferentes dispositivos de rede. Essa heterogeneidade é explorada por técnicas de coleta de assinaturas.

A coleta de assinaturas para dispositivos na Internet IPv4 é bastante estudada, no entanto existem poucos trabalhos (e ferramentas) que focam no protocolo IPv6. Segundo a Google, o tráfego de pacotes IPv6 na Internet passou dos 12% em Junho de 2016 [Google 2016]. Dessa forma, estudos que revelam aspectos da segurança da Internet, em particular da Internet IPv6, são de grande importância, dado que nosso entendimento sobre as peculiaridades de segurança da Internet IPv6 ainda é limitado.

Neste trabalho propomos um método para construção de assinaturas de dispositivos de rede para a Internet IPv6 que leva em conta não só informações do dispositivo alvo como também de seus vizinhos. Nossa assinatura é construída através de medições ativas e leva em conta aspectos do protocolo IPv6 que não foram explorados pelos poucos trabalhos existentes sobre o assunto. Consideramos os diferentes comportamentos dos dispositivos de rede em relação aos campos Traffic Class e Flow Label do cabeçalho IPv6. Em nossos experimentos, medimos e classificamos dispositivos de rede em cinco clusters diferentes. Aproveitamos para avaliar a viabilidade da técnica de coleta de assinaturas proposta por Vanaulbel et al. [Vanaubel et al. 2013] que é baseada nos TTLs iniciais de pacotes enviados pelos dispositivos.

Nosso método pode ser utilizado para identificar problemas de vazamento de informações confidenciais dos sistemas autônomos e auxiliar os administradores de redes na mitigação destes problemas. Por exemplo, se um atacante consegue associar um tipo de assinatura a um determinado modelo, tipo ou configuração de dispositivo de rede ele pode utilizar essa informação para explorar alguma vulnerabilidade previamente conhecida do dispositivo. Nosso método pode ser útil para obtenção da topologia de uma rede no nível de roteadores, complementando técnicas custosas de associação de endereços (IP *aliasing*), uma vez que interfaces que possuem o mesmo comportamento podem ser associadas a um mesmo roteador.

2. Fundamentação Teórica

O cabeçalho IPv6 trouxe muitas novidades em relação ao cabeçalho IPv4. Aqui introduzimos duas delas, o campo Flow Label e o campo Traffic Class. Em nosso trabalho estamos interessados no comportamento dos dispositivos de rede em relação a estes campos.

Alguns dispositivos de rede podem levar em conta um identificador de fluxo durante o processamento dos pacotes. Um balanceador de carga, por exemplo, pode utilizar o identificador de fluxo para encaminhar todos os pacotes de um mesmo fluxo para a mesma interface de saída, evitando que estes pacotes cheguem fora de ordem no destino final. O identificador de fluxo é tradicionalmente formado pela quíntupla: endereço de origem, endereço de destino, protocolo, porta de origem e porta de destino. Em um pacote IPv4, as informações da camada de transporte (como porta de origem e de destino) podem ser facilmente alcançadas através do campo Internet Header Length (IHL). No entanto, o campo IHL não existe no cabeçalho IPv6. Se um pacote IPv6 possui cabeçalhos de extensão, é necessário seguir a cadeia de Next Headers para alcançar o cabeçalho da camada de transporte. Isso impede que o tradicional identificador de fluxo seja calculado de forma eficiente. Para contornar este problema, o campo Flow Label foi introduzido no cabeçalho IPv6 [Amante et al. 2011]. O campo Flow Label permite a identificação do fluxo sem que as informações do cabeçalho da camada de transporte sejam necessárias. O campo Flow Label é especificado pela RFC 6437 [Amante et al. 2011]. Segundo a RFC, quando a este campo é atribuído um valor diferente de zero, é esperado que o pacote chegue ao destino final sem que o valor do Flow Label seja alterado. A RFC indica ainda que um dispositivo que encaminha um pacote cujo Flow Label é zero, pode inicializar o campo para um valor diferente de zero.

Além do Flow Label, o protocolo IPv6 também introduziu o campo Traffic Class. O campo Traffic Class é utilizado por dispositivos de rede para identificar diferentes classes de pacotes IPv6. Segundo a RFC 2460 [Deering and Hinden 1998], que especifica o protocolo IPv6, é esperado que o Traffic Class dê suporte aos serviços diferenciados (DiffServ), assim como o campo TOS do cabeçalho IPv4. Dispositivos de rede podem utilizar este campo de diferentes maneiras, inclusive de forma experimental e não padronizada. A RFC em questão determina que dispositivos de rede que suportam o campo Traffic Class podem modificar os valores do campo ao criar, encaminhar ou receber um pacote IPv6, sem restrições. Um dispositivo não deve assumir que o valor do campo Traffic Class é o mesmo de quando o pacote foi criado, ou seja, ele pode ter sido modificado por um dispositivo intermediário.

3. Determinando a Assinatura de um Dispositivo

Para construir a assinatura de um dispositivo alvo de rede IPv6 levamos em conta seu comportamento, dos dispositivos antecessores a ele (que estão entre o ponto de medição e o dispositivo alvo), e também do dispositivo sucessor a ele. O dispositivo sucessor tem papel importante pois utilizamos os pacotes com origem nele para descobrir se o dispositivo alvo faz ou não modificações nos pacotes enviados do ponto de medição. Já os dispositivos antecessores são levados em conta pois, se entre o ponto de medição e o dispositivo alvo há um dispositivo que modifica campos do cabeçalho IPv6, então precisamos saber isso para diferenciar modificações realizadas pelo dispositivo alvo de modificações realizadas por dispositivos intermediários.

Como exemplo vamos considerar a rota de Internet IPv6 $[R_1, R_2, R_3, R_4]$. Essa rota possui 4 dispositivos de rede. Vamos considerar R_3 como o dispositivo alvo. Neste caso, R_1 e R_2 são os dispositivos antecessores e R_4 o dispositivo sucessor ao dispositivo alvo. Para determinar sua assinatura, nossa metodologia leva em conta pacotes com origem em R_1, R_2, R_3 e em R_4. Estamos particularmente interessados em 2 tipos de informações do dispositivo alvo R_3: (i) as modificações que ele faz nos campos Traffic Class e Flow Label ao encaminhar um pacote para R_4 e (ii) os valores que ele utiliza para inicializar os campos Traffic Class e Flow Label quando cria pacotes ICMP Time Exceeded e ICMP Echo Reply.

3.1. Sondas de Medição

Inicialização dos campos Flow Label e Traffic Class. Enviamos sondas com TTL limitado utilizando a técnica do Paris traceroute, mantendo o identificador de fluxo constante. Para cada dispositivo alvo, em ordem crescente de distância a partir do ponto de medição, enviamos duas sondas ICMP Echo Request com TTL limitado. Na primeira sonda mantemos os campos Traffic Class e Flow Label zerados, e na segunda inicializamos estes campos com valores diferentes de zero. Essas duas sondas fazem com que os dispositivos alvo respondam com pacotes ICMP Time Exceeded. Com essas respostas inferimos como o dispositivo alvo inicializa os campos Flow Label e Traffic Class para pacotes Time Exceeded. Para cada dispositivo alvo, enviamos uma terceira sonda ICMP Echo Request destinada ao endereço IP identificado no dispositivo alvo pelas sondas acima. A sonda ICMP Echo Request faz com que o dispositivo alvo responda com um pacote ICMP Echo Reply. Com essa última sonda sabemos como o dispositivo alvo inicializa os campos Flow Label e Traffic Class para pacotes Echo Reply.

Modificação dos campos Flow Label e Traffic Class. Os pacotes ICMP Time Exceeded copiam também o cabeçalho do pacote recebido pelo dispositivo alvo (o pacote cujo TTL expirou e resultou na criação da resposta ICMP Time Exceeded). O cabeçalho copiado permite identificar qual Flow Label e Traffic Class foram observados pelo dispositivo alvo, em particular, se o Flow Label e Traffic Class foram modificados por algum dispositivo antecessor.

Impacto de rotas assimétricas. Rotas na Internet são assimétricas: um pacote de resposta criado pelo dispositivo pode trafegar de volta até o ponto de medição (*reverse path*) por um caminho diferente do trafegado pela sonda (*forward path*). Neste caso não podemos determinar qual dispositivo fez a modificação (ou inicialização) nos campos Flow Label e Traffic Class do pacote de resposta, apenas detectar a inicialização ou modificação dos campos. Note que o cabeçalho encapsulado por pacotes ICMP Time Exceeded não é modificado no caminho de volta até o ponto de medição, de forma que podemos identificar quais dispositivos modificam os campos Flow Label e Traffic Class no caminho do ponto de medição até um destino (*forward path*).

Impacto da perda de pacotes. Nossa metodologia depende fortemente das informações dos pacotes de respostas dos dispositivos considerados para a composição da assinatura. Nossa ferramenta reenvia as sondas de medições (faz até 3 reenvios) para tentar obter uma resposta do dispositivo. Dispositivos que não respondem impedem identificar precisamente qual dispositivo realizou as modificações nos campos do pacote. Neste caso, associamos qualquer modificações observadas ao dispositivo sucessor.

3.2. Assinatura dos dispositivos

Levamos em conta uma assinatura para os dispositivo de rede IPv6 que é formada pela dupla $\langle \text{MOD}, \text{INI} \rangle$. A entrada MOD guarda informações sobre as modificações que o dispositivo alvo faz nos campos Traffic Class e Flow Label ao encaminhar um pacote para o dispositivo sucessor. A entrada INI guarda informações sobre como o dispositivo alvo inicializa (e como dispositivos no caminho de volta até o ponto de medição modificam) os campos Traffic Class e Flow Label nos pacotes ICMP Time Exceeded e Echo Reply que ele envia. Cada entrada da assinatura é uma quádrupla. A tabela 3.2 sumariza a assinatura. A seguir, detalhamos os valores que cada um dos campos pode assumir e como determinamos cada campo.

Campo	Valor	Significado
MOD	$\langle \text{FL}_\text{Z}, \text{TC}_\text{Z}, \text{FL}_\text{N}, \text{TC}_\text{N} \rangle$	Campos que o dispositivo modifica
INI	$\langle \text{FL}_\text{TE}, \text{FL}_\text{ER}, \text{TC}_\text{TE}, \text{TC}_\text{ER} \rangle$	Campos que o dispositivo inicializa

Tabela 1. Campos da assinatura de dispositivos.

O campo MOD da assinatura guarda as modificações que o dispositivo alvo faz ao encaminhar um pacote. Ele é formado pela quádrupla $\langle \text{FL}_\text{Z}, \text{TC}_\text{Z}, \text{FL}_\text{N}, \text{TC}_\text{N} \rangle$. Cada entrada assume valor booleano e indica se o dispositivo modifica os campos Traffic Class (TC) e Flow Label (FL) quando estes campos têm valor zero (Z) e diferente de zero (N).

O campo INI da assinatura guarda como o dispositivo alvo inicializa os campos de seus pacotes. Ele é formado pela quádrupla $\langle \text{FL}_\text{TE}, \text{FL}_\text{ER}, \text{TC}_\text{TE}, \text{TC}_\text{ER} \rangle$. Cada entrada indica como o dispositivo inicializa os campos Flow Label (FL) e Traffic Class (TC) para pacotes ICMP Time Exceeded (TE) e pacotes ICMP Echo Reply (ER). Cada campo pode assumir o valor ZERO, quando não inicializa o campo; COPY, quando o valor do campo é o mesmo da sonda que provocou a resposta ICMP; ou DIFF quando o valor é diferente de zero e diferente do valor do campo na sonda que provocou a resposta.

A medição é feita salto a salto, como no traceroute. Inicialmente sabemos o comportamento do ponto de medição e conseguimos calcular a assinatura de seu sucessor (primeiro salto) enviando sondas para ele e para o segundo salto. Fazemos isso até alcançar um destino.

No nosso exemplo, para detectar se o dispositivo alvo R_3 faz modificações em um pacote IPv6 ao encaminhá-lo para R_4 verificados os valores do Traffic Class e do Flow Label no cabeçalho encapsulado por pacotes ICMP Time Exceeded criados por R_4. Para verificar como dispositivo R_3 inicializa os campos Traffic Class e Flow Label, verificamos o cabeçalho IPv6 dos pacotes que ele envia. Note que esta verificação é impossível caso um dos dispositivos entre R_3 e o ponto de medição também faça modificações. Para respostas DIFF, inferimos que algum dispositivo no caminho do dispositivo alvo até o ponto de medição (*reverse path*) modifica os campos Traffic Class e Flow Label, ou que o dispositivo alvo inicializa os campos. Para respostas COPY, inferimos que os campos foram inicializados pelo dispositivo alvo e não foram modificados no caminho de volta até o ponto de medição, pois apenas o dispositivo alvo tem acesso aos valores utilizados ao inicializar a sonda. Para respostas ZERO, inferimos que o dispositivo alvo não inicializa os campos Flow Label e Traffic Class e que nenhum dispositivo no caminho de volta até o ponto de medição modifica os campos.

Id.	País	Cidade	ASN
cph-dk	Dinamarca	Ballerup	59469
dac-bd	Bangladesh	Dhaka	24122
hnl-us	Estados Unidos	Honolulu, HI	6360
lax-us	Estados Unidos	Los Angeles, CA	2152
san-us	Estados Unidos	San Diego, CA	1909
sin-sg	Singapura	Singapore	37989
ylk-ca	Canadá	Barrie, ON	19764

Tabela 2. Pontos de medição utilizados para coleta

4. Coleta e caracterização dos dados

Coletamos assinaturas para um total de 19199 interfaces diferentes durante um período de 5 dias. Nossas medições foram realizadas em sete pontos de medição da infraestrutura do Ark, mantidos pela CAIDA (Center for Applied Internet Data Analysis) [Hyun 2016]. Os dispositivos estão localizados em redes distintas de cinco países, como é mostrado na tabela 2. Nossa lista de destinos possui 51928 endereços IP e foi formada a partir da *hitlist* IPv6 disponibilizada por Gasser et al. [Gasser et al. 2016]. A lista original possui cerca de 700 mil endereços IPv6 e construímos a nossa lista escolhendo aleatoriamente dois endereços IP para cada prefixo /48 da lista original.

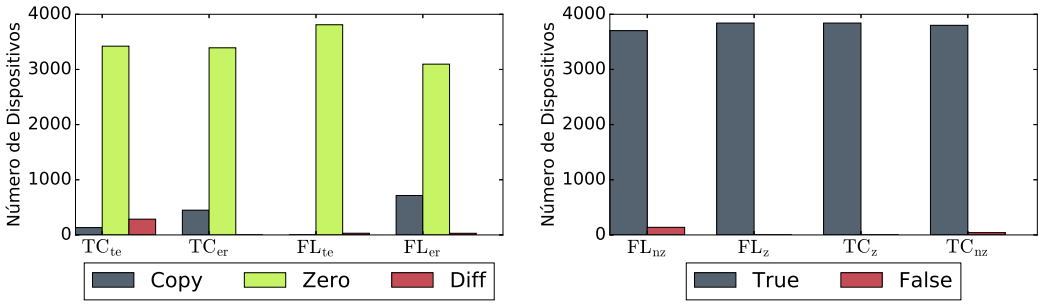
Para selecionar os dispositivos alvos, realizamos medições de traceroute partindo dos sete pontos de medição com destino a cada um dos 51928 endereços IP selecionados. Assim, cada um dos dispositivos que apareceram nesses caminhos são dispositivos alvos. Os dispositivos para os quais não conseguimos obter uma assinatura completa são desconsiderados da análise.

5. Resultados

Para identificar dispositivos com comportamento similar entre os dispositivos mensurados durante nossas medições, realizamos um agrupamento utilizando o algoritmo K-means, que recebe a assinatura de cada dispositivo e os agrupa em K classes definidas. É conhecido que um dos maiores desafios desse algoritmo é determinar o valor de K corretamente dado que é bastante subjetivo [Han et al. 2011]. Em nosso problema, não sabemos quais são os diferentes comportamentos presentes em nossas medições. Para contornar esse problema, utilizamos o método de Elbow para estimar o número de classes ideal que minimize o erro interno de cada grupo (i.e., consiga agrupar bem dispositivos similares) e, ao mesmo tempo, não especialize demasiadamente os mesmos (onde exista um grupo para cada dispositivo) [Kodinariya and Makwana 2013]. O método revelou que o valor de $K = 5$ é o melhor compromisso entre baixo erro interno e generalização dos grupos.

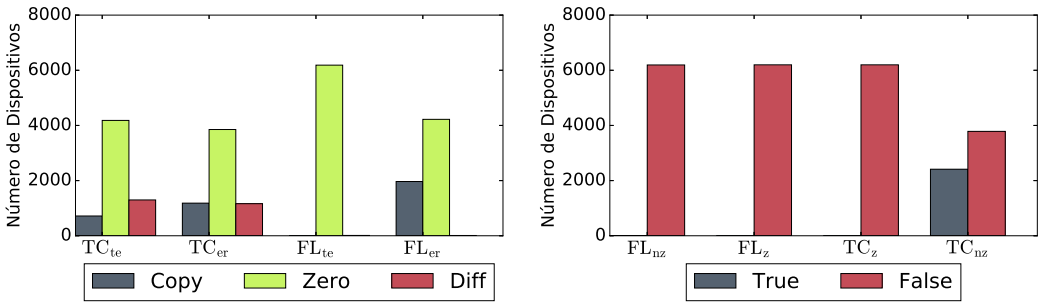
Dado a descoberta de cinco diferentes classes de dispositivos em nossa base, buscamos entender quais características as diferenciam. Através das figuras 1–5 é possível comparar os atributos de inicialização e modificação dos dispositivos em cada cluster, respectivamente. A seguir, descrevemos em detalhes cada um dos clusters identificados.

Cluster 1 (20%): A característica dos dispositivos do cluster 1 é que eles sempre modificam o valor do campo Flow Label e do campo Traffic Class ao encaminhar um pacote IPv6, independente do valor encontrado nesses campos. Além disso, a maior parte dos



(a) Atributos inicializados (b) Atributos modificados

Figura 1. Características do Cluster 1



(a) Atributos inicializados (b) Atributos modificados

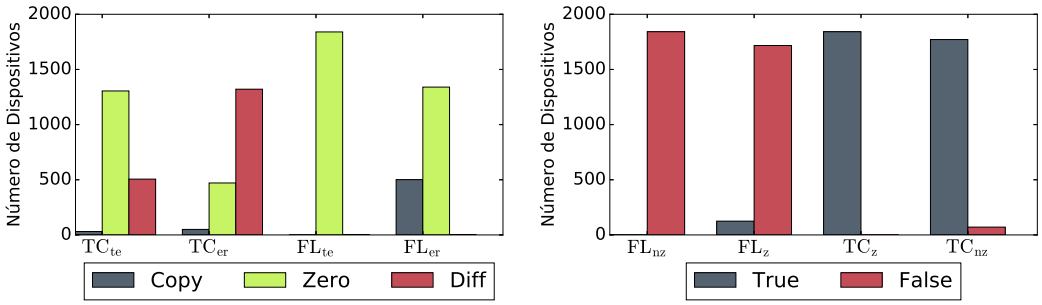
Figura 2. Características do Cluster 2

dispositivos nesse cluster inicializam os campos Traffic Class e Flow Label dos pacotes ICMP Time Exceeded e ICMP Echo Reply com ZERO.

Cluster 2 (32%): O cluster 2 é formado em sua maioria por roteadores que não modificam o campo Flow Label ao encaminhar um pacote. Os dispositivos deste cluster também não modificam o campo Traffic Class quando este está zerado, mas alguns dos dispositivos (menos da metade) modificam o Traffic Class quando este é diferente de zero. Além disso, grande parte dos dispositivos do cluster 2 inicializa os campos Traffic Class e Flow Label dos pacotes ICMP Time Exceeded e ICMP Echo Reply com ZERO.

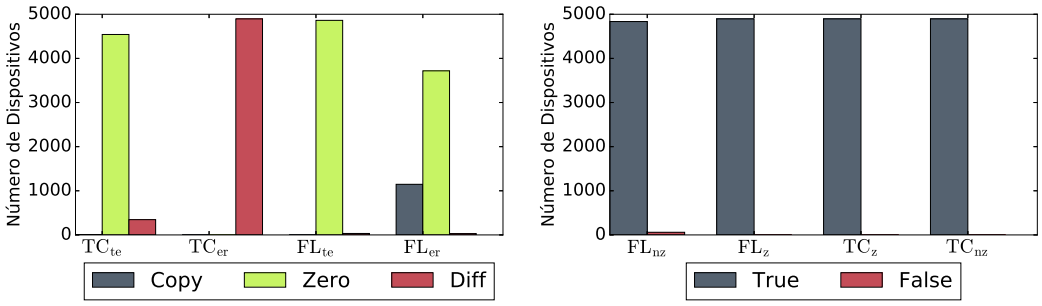
Cluster 3 (10%): O cluster 3 é formado, em geral, por dispositivos que modificam o valor do campo Traffic Class mas não modificam o valor do campo Flow Label ao encaminhar um pacote IPv6, independente do valor encontrado nesses campos. Quanto a inicialização dos campos dos pacotes ICMP, todos dispositivos deste cluster inicializam o Flow Label de pacotes Time Exceeded com ZERO, e grande parte inicializa o Flow Label dos pacotes ICMP Echo Reply também com ZERO, mas existe uma parte considerável de dispositivos que inicializa o Flow Label com o valor encontrado no pacote que causou a resposta ICMP (COPY). Em pacotes Time Exceeded, grande parte dos dispositivos inicializam o Traffic Class com ZERO, já em pacotes Echo Reply, a grande maioria inicializa com valores diferentes de 0 e diferentes dos pacotes que causaram a resposta.

Cluster 4 (25%): O cluster 4, por sua vez, é formado por dispositivos que modificam o campo Traffic Class e Flow Label dos pacotes independente do valor destes campos. A diferença para o cluster 1 fica em como estes dispositivos inicializam os campos do



(a) Atributos inicializados (b) Atributos modificados

Figura 3. Características do Cluster 3



(a) Atributos inicializados (b) Atributos modificados

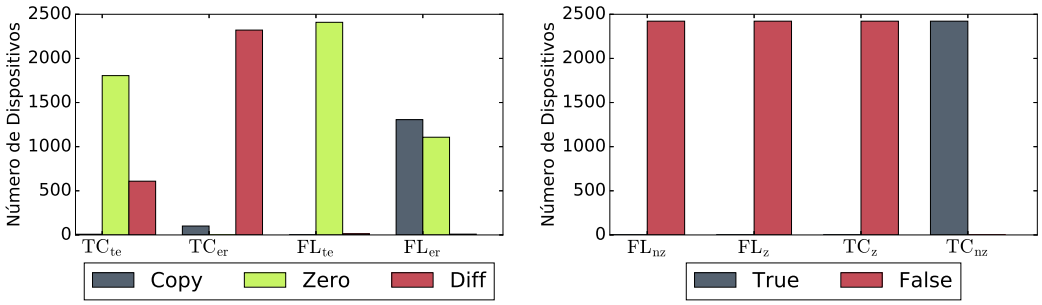
Figura 4. Características do Cluster 4

cabeçalho IPv6 nas mensagens ICMP de resposta. Todos os dispositivos do cluster 4 inicializam o Traffic Class para pacotes Echo Reply com valores diferentes de ZERO e diferentes das sondas que causaram a resposta (DIFF). No caso de pacotes Time Exceeded, os dispositivos, em geral, inicializam o Traffic Class com ZERO. Para o Flow Label, a característica dos dispositivos do cluster 4 é que eles inicializam este campo com ZERO tanto para pacotes com mensagens Time Exceeded quanto para mensagens Echo Reply.

Cluster 5 (13%): O cluster 5 é composto por dispositivos que modificam o Traffic Class quando este é diferente de zero mas não modificam quando é zero. Além disso, os dispositivos do cluster 5 não modificam o campo Flow Label ao encaminhar um pacote IPv6. Já as características de inicialização dos dispositivos do cluster 5 é que eles inicializam com ZERO o campo Flow Label dos pacotes com mensagem Time Exceeded, no entanto para pacotes com mensagem Echo Reply, estes dispositivos copiam o valor do pacote que causou a resposta.

5.1. Inicialização do Campo TTL

Durante nossos experimentos, verificamos a possibilidade de levar em conta também a inicialização do campo Hop Limit (TTL no IPv4) dos pacotes ICMP Time Exceeded e ICMP Echo Reply, como foi proposto por Vanaubel et al. [Vanaubel et al. 2013]. O trabalho de Vanaubel et al. mostrou que é possível classificar dispositivos de redes de forma bem definida levando em conta o TTL inicial de pacotes ICMP Time Exceeded e ICMP Echo Reply. No entanto, percebemos que tal classificação não é possível na Internet IPv6. Durante nossas medições, percebemos um comportamento que acreditamos ser



(a) Atributos inicializados (b) Atributos modificados

Figura 5. Características do Cluster 5

exclusivo do protocolo IPv6. Diferente do que Vanaubel et al. mostraram para a Internet IPv4, encontramos que 98,63% dos dispositivos na Internet IPv6 inicializam o Hop Limit com 64, tanto para pacotes ICMP Time Exceeded quanto para ICMP Echo Reply. De fato, a RFC 1700 recomendou em 1994 que o valor padrão para o TTL deveria ser 64, no entanto isso não era seguido. Acreditamos que este comportamento mostra maior aderência dos padrões e recomendações nas implementações IPv6 se comparado com o comportamento mostrado por Vanaubel et al.

6. Trabalhos Relacionados

Técnicas de coleta de assinaturas de dispositivos de rede são estudadas há algum tempo. No entanto, grande parte possui foco no protocolo IPv4 [Shu and Lee 2006, Vanaubel et al. 2013, Paxson 1997]. Técnicas específicas para o protocolo IPv6 começaram a ser estudadas recentemente, com a crescente popularização deste protocolo. O Nmap [Lyon 2009], por exemplo, é uma ferramenta de detecção de assinatura de dispositivos de rede bastante popular. A ferramenta trabalha enviando sondas destinadas ao dispositivo alvo, construindo uma assinatura do dispositivo e comparando essa assinatura com outras já conhecidas previamente armazenadas em um banco de dados. O Nmap possui um módulo de identificação para o protocolo IPv6. Nosso trabalho vai além no sentido de que considera informações não só do dispositivo alvo como também de um dispositivo sucessor para formar a assinatura, desta forma conseguimos identificar os campos do protocolo IPv6 que o dispositivo alvo modifica.

O trabalho de Vanaubel et al. [Vanaubel et al. 2013] utiliza uma assinatura baseada no TTL inicial de pacotes ICMP Time Exceeded e Echo Reply e mostra que com essa assinatura é possível inferir características de dispositivos de redes, inclusive a marca do dispositivo. Inspirados pelos resultados de Vanaubel et al., verificamos que dispositivos IPv6 inicializam o campo Hop Limit com valor 64.

7. Conclusões e Trabalhos Futuros

Neste trabalho desenvolvemos um método de construção de assinatura para dispositivos de rede IPv6. Nosso método utiliza uma assinatura que é baseada nas características dos dispositivos ao processar diferentes pacotes IPv6. Nossa assinatura leva em conta o comportamento não só do dispositivo alvo como também dos dispositivos antecessores e de seu dispositivo sucessor. Durante cinco dias coletamos assinaturas de 19199 interfaces

diferentes que foram classificadas em cinco clusters bem definidos através do algoritmo K-means. Mostramos também que 98.63% dos dispositivos IPv6 inicializam o campo TTL com o valor 64, limitando a utilidade deste campo na identificação de dispositivos IPv6.

Em trabalhos futuros pretendemos melhorar nossa técnica de identificação de roteadores adicionando mais campos à assinatura. Por exemplo, esperamos que os cabeçalhos de extensão de um pacote IPv6 possam fornecer boas informações para compor a assinatura do dispositivo. Outro aspecto que pretendemos estudar é a quantidade de dados (o tamanho do *payload*) de pacotes ICMP Time Exceeded. Estas características podem servir, portanto, como mais campos de assinatura de um dispositivo de rede.

Referências

Amante, S., Carpenter, B., Jiang, S., and Rajahalme, J. (2011). Ipv6 flow label specification. RFC 6437, RFC Editor.

Deering, S. E. and Hinden, R. M. (1998). Internet protocol, version 6 (ipv6) specification. RFC 2460, RFC Editor.

Gasser, O., Scheitle, Q., Gebhard, S., and Carle, G. (2016). Scanning the ipv6 internet: Towards a comprehensive hitlist. *Traffic Monitoring and Analysis Workshop (TMA)*.

Google (2016). Ipv6 adoption statistics in the internet. (2016, June 29) Retrieved from https://www.google.com/intl/en/ipv6/statistics.html.

Han, J., Pei, J., and Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

Hyun, Y. (2016). Archipelago (ark) measurement infrastructure. (2016, July 15) Retrieved from http://www.caida.org/projects/ark/.

Kodinariya, T. M. and Makwana, P. R. (2013). Review on determining number of cluster in k-means clustering. *International Journal*, 1(6):90–95.

Lyon, G. F. (2009). *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure.

Paxson, V. (1997). Automated packet trace analysis of tcp implementations. In *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '97, pages 167–179, New York, NY, USA. ACM.

Shu, G. and Lee, D. (2006). Network protocol system fingerprinting-a formal approach. In *Proc. IEEE International Conference on Computer Communications (INFOCOM)*.

Vanaubel, Y., Pansiot, J.-J., Mérindol, P., and Donnet, B. (2013). Network fingerprinting: Ttl-based router signatures. In *Proc. ACM Internet Measurement Conference (IMC)*, pages 369–376. ACM.

Elaboração e Avaliação de um Modelo de Comunicação Direta Criptografada Entre Dispositivo Móvel e Servidor

Juliana G. de Souza[1], **Luciana A. F. Martimiano**[1]

[1]Departamento de Informática – Universidade Estadual de Maringá (UEM)
CEP 87.020-900 – Maringá – PR – Brazil

juulianags@gmail.com, lafmartimiano@uem.br

Abstract. *The increasingly frequent use of mobile devices to remote access information highlights the need for ongoing studies on the security of data in transit. This paper seeks to establish and evaluate a model of communication between a server and a mobile device (client), considering the processing and memory limitations, that ensures data confidentiality through encryption. The model uses the cryptographic algorithms RSA and AES and evaluation thereof - in particular the performance in the generation of secret keys - and performed by an application developed using the Android platform.*

Resumo. *A utilização cada vez mais frequente de dispositivos móveis para acesso remoto de informações destaca a necessidade de contínuos estudos relativos à segurança dos dados em trânsito. O presente trabalho busca estabelecer e avaliar um modelo de comunicação entre um servidor e um dispositivo móvel (cliente), considerando as limitações de processamento e memória contidas neste, que garanta a confidencialidade dos dados mediante criptografia. O modelo utiliza os algoritmos criptográficos RSA e AES e a avaliação deste - em especial o desempenho na geração de chaves secretas - é realizada por meio de uma aplicação desenvolvida na plataforma Android.*

1. Introdução

Pesquisas afirmam que, dentre os domicílios brasileiros com acesso à Internet no ano de 2014, 80,4% o realizavam por meio de aparelho celular [IBGE 2016]. Esse acesso, tanto no Brasil como em todo o mundo, ocorre com aplicativos que realizam cada vez mais numerosas e diversificadas funções.

Muitos desses aplicativos armazenam e transmitem informações pessoais sigilosas. Cabe ao desenvolvedor proporcionar o máximo de confidencialidade, restringindo o acesso às informações apenas aos usuários devidos; integridade, impedindo a alteração indevida das informações; e disponibilidade, fornecendo os serviços propostos de maneira satisfatória. Além disso, é responsabilidade do desenvolvedor conhecer e compreender tecnologias diversas, afim de que as interações entre o sistema desenvolvido e outros sistemas, e até mesmo as interações internas, sejam realizadas de maneira eficiente e segura.

Em busca de vantagem competitiva e/ou proteção às suas operações, a maioria das organizações que realizam tráfego de informações sigilosas por meio de dispositivos móveis não divulga suas diretivas de segurança. Todavia, acompanhando o avanço da tecnologia e o aumento do poder de processamento e armazenamento dos computadores,

as estratégias de ataque se aperfeiçoam e se diversificam constantemente, exigindo que o tema continue sendo estudado e avaliado.

No decorrer deste trabalho, a principal característica de segurança abordada é a confidencialidade, garantida por meio de criptografia. O objetivo geral consiste em estabelecer e avaliar um modelo de comunicação entre um servidor e um dispositivo móvel, considerando as limitações de processamento e memória contidas neste. O modelo utiliza os algoritmos criptográficos RSA e AES e a avaliação deste, em especial o desempenho na geração de chaves secretas, é realizada por meio de uma aplicação *Android*.

O artigo está organizado da seguinte forma: a Seção 2 apresenta uma visão geral dos aspectos envolvidos na segurança, incluindo considerações sobre segurança no sistema *Android*; na Seção 3 são descritos o modelo de comunicação desenvolvido e a aplicação utilizada como estudo de caso; os resultados e a análise referentes à geração das chaves de criptografia são apresentados na Seção 4; e a Seção 5 apresenta as conclusões e as sugestões de trabalhos futuros.

2. Visão Geral da Segurança

O Ministério da Justiça define segurança como a "proteção dos ativos de informação contra perda, corrupção, destruição, acesso, uso e alteração indevidos ou não autorizados", e Segurança da Informação e Comunicações (SIC) como "ações que objetivam viabilizar e assegurar a disponibilidade, a integridade, a confidencialidade e a autenticidade das informações" [Brasil 2013]. Porém, chegar a um estado perfeito de segurança é impraticável [Six 2012], mas é possível identificar os riscos, amenizá-los e tornar viável a comunicação segura.

As ameaças podem estar relacionadas a diversos aspectos. *Softwares* maliciosos podem desencadear eventos não esperados. Existem ainda as falhas humanas, em que acidentes ou enganos das pessoas que interagem com o sistema desencadeiam consequências negativas. Além disso, existe a figura do agressor deliberado, que se torna uma ameaça tanto na tentativa de invasão e espionagem, quanto por objetivos de sabotagem e vandalismo [Marciano 2006].

Na busca pelo nível ideal de segurança é preciso pensar no maior número possível de características do *software* em questão. É preciso saber o que exatamente se quer proteger; as ameaças mais prováveis; a importância de cada parte do sistema; o grau desejável de segurança que esses dados exigem; os recursos disponíveis adequados a cumprir essa meta em se tratando de tempo, pessoas e recursos financeiros; e a expectativa dos indivíduos envolvidos com o projeto [Coelho et al. 2014]. Dessa maneira, é aconselhável que as preocupações com a segurança sejam de responsabilidade de todos os envolvidos, acrescentando, se possível, a realização de consultoria com especialistas [ABNT 2005].

Conforme a norma da ABNT [ABNT 2005], "convém que a política de computação móvel inclua os requisitos de proteção física, controles de acesso, técnicas criptográficas, cópias de segurança e proteção contra vírus". É de suma importância, juntamente, a análise da conformidade das operações realizadas no aplicativo com a legislação vigente.

Criptografia é uma técnica utilizada para a troca de mensagens fim a fim sem influência ou visualização de terceiros. Os algoritmos de criptografia são divididos en-

tre algoritmos de criptografia simétrica, em que o emissor e o receptor conhecem a chave utilizada para codificação e, por meio dela, realizam a decodificação da mensagem [Schneier 2000], como o AES (*Advanced Encryption Standard*); algoritmos de criptografia assimétrica, em que um par de chaves é gerado, uma privada e uma pública, para codificar e decodificar uma mensagem [Diffie and Hellman 1976], como o RSA (*Rivest - Shamir - Adleman*); e algoritmos de *hash*, que "mapeiam uma mensagem de tamanho variável em um valor de *hash* de tamanho fixo, ou um resumo da mensagem", sem a possibilidade de restaurar a mensagem inicial, como o SHA (*Secure Hash Algorithm*) [Stallings 2008].

2.1. Segurança na Plataforma *Android*

Desde o início do projeto *Android* considerou-se a segurança um ponto de suma importância. O planejamento ocorreu após a observação do funcionamento de outras plataformas, se aproveitando de padrões já existentes e procurando suprir necessidades detectadas. A arquitetura multicamadas é a principal estratégia de segurança da plataforma. Com uma estrutura semelhante a uma pilha, em cada nível é assumido que os anteriores estão seguros. No nível mais baixo são oferecidas as funções de segurança do *kernel* do *Linux* e definidas as permissões de comunicação entre os processos. Caso uma aplicação maliciosa queira prejudicar o sistema, a falta de interação com outros processos a impedirá de causar grandes estragos [ANDROID 2014].

O modelo de permissões é baseado no usuário. A cada aplicativo sendo executado é associado um número de identificação, juntamente com as informações do usuário, gerando um processo único, diferentemente do caso em que todos as aplicações são executadas com as mesmas permissões de usuário. Essa área da execução do processo é chamada de *sandbox* da aplicação e restringe as ações em seu próprio contexto, já que limita o contato com o sistema operacional e impede que outras aplicações e regiões de memória sejam afetadas.

3. Desenvolvimento da Aplicação

Com o objetivo de avaliar a geração de chaves dos algoritmos criptográficos e testar um modelo de comunicação seguro, foi realizado um estudo de caso em uma empresa real, que desejava o desenvolvimento de um aplicativo com as funcionalidades básicas de um sistema já existente. A funcionalidade principal consiste em receber informações pessoais do usuário e enviá-las a um servidor, que realiza uma análise de crédito baseada em inteligência artificial.

O projeto é composto por três módulos: aplicação cliente, comunicação e aplicação servidora. Os dados, e a tramitação desses, devem ser analisados de maneira apropriada em cada um dos módulos. Primeiramente, é preciso tratar a comunicação direta com o usuário, a recepção das informações e as primeiras ações necessárias a fim de preparar os dados, evitando informações incompletas, inconsistentes e dados faltantes. Essa análise é crucial para impedir que um agente malicioso realize injeção de código. Em um segundo momento, esses dados devem ser protegidos e transmitidos ao servidor sem intervenção de terceiros. Na aplicação servidora, por fim, as informações são processadas de acordo com os serviços solicitados e os resultados são enviados ao cliente.

Quanto ao ambiente de desenvolvimento, além da adição de senhas nas máquinas

envolvidas no projeto, estas devem ser mantidas em salas com restrição de acesso. Na empresa em que o estudo de caso foi realizado não é permitido que visitantes tenham acesso ao servidor local, localizado em sala separada e resfriada. As versões do projeto são mantidas no sistema *Apache Subversion*, um sistema para controle de versão de código livre. Desse modo, o projeto se mantém consistente e com possibilidade de recuperação. Os servidores são armazenados em dois lugares fisicamente distantes e possuem dados replicados, assegurando a existência de cópias de segurança. Todas as máquinas possuem sistemas de proteção a programas maliciosos e acessos indevidos.

3.1. Desenvolvimento da Comunicação

A comunicação entre a aplicação cliente (no dispositivo móvel) e a aplicação servidora (no servidor) é um ponto crítico em relação à segurança das informações. Os dados trocados partem dos extremos da comunicação e estão sujeitos a ameaças. À vista disso, com o intuito de garantir confidencialidade e integridade, técnicas criptográficas são utilizadas.

Na literatura, é possível identificar vários tipos de criptografia e algoritmos. Porém, definir como a criptografia será realizada não consiste apenas em implementar um algoritmo, é preciso selecioná-lo de acordo com alguns critérios e definir os passos do processo. Deve-se levar em consideração que as operações de codificação e decodificação serão realizadas em dispositivos móveis, que não possuem o mesmo poder de processamento e memória que os computadores em que esses algoritmos geralmente são implementados. Outra análise necessária é com relação aos avanços na tentativa de "quebrar" o algoritmo, ou seja, de obter a mensagem inicial ou texto claro sem possuir a chave, ou de deduzir a mesma.

Como o enfoque do modelo de comunicação situa-se na comunicação direta entre dispositivo móvel e servidor, a empresa decidiu não utilizar certificação digital e não envolver outras organizações. Essa condição imposta pela empresa em que o estudo de caso foi realizado, trouxe ao modelo uma possível vulnerabilidade a ataques de Man-in-The-Middle e semelhantes. Desse modo, não seria possível manter uma organização intermediária no sentido de auxiliar no estabelecimento da comunicação e troca de chaves. Caso a chave seja sempre a mesma e esteja armazenada tanto no servidor como no dispositivo, a troca de mensagens se torna mais simples. Existe, porém, a vulnerabilidade de manter a mesma chave em todos os dispositivos que utilizam o aplicativo, excluindo a possibilidade de alteração sem atualização total do aplicativo. Por outro lado, para operar uma chave variável é preciso tornar conhecida essa informação antes do início da comunicação, o que pode inutilizar todo o processo de codificação.

Denomine-se a chave da criptografia simétrica como chave secreta e o par de chaves da criptografia assimétrica como chave pública e chave privada. A solução proposta para a comunicação codificada consiste nas seguintes etapas, mostradas nas Figuras 2 (a), 2 (b) e 1:

1. No momento em que o usuário acessa o aplicativo, por meio da aplicação cliente, e realiza alguma operação que necessite acesso ao servidor, a mensagem A, a ser enviada, é criada.
2. Uma chave secreta é gerada e a mensagem A é codificada com ela, gerando uma mensagem codificada B.

3. A chave secreta é codificada com a chave pública não certificada do servidor, gerando uma mensagem C.
4. É enviada à aplicação servidora uma mensagem D composta de duas partes: a mensagem B e a mensagem C.
5. A aplicação servidora recebe a mensagem D e utiliza sua chave privada para decodificar a mensagem C e obter a chave secreta.
6. Por meio da chave secreta, a aplicação servidora decodifica a mensagem B, obtendo a mensagem A inicial.
7. A aplicação servidora processa a mensagem A e gera uma mensagem E de resposta.
8. A aplicação servidora utiliza a chave secreta para codificar a mensagem E, gerando uma mensagem F codificada, e a envia à aplicação cliente.
9. A aplicação cliente recebe a mensagem F e a decodifica com a chave secreta criada no início no processo.
10. A aplicação cliente interpreta a mensagem e exibe os resultados para o usuário.

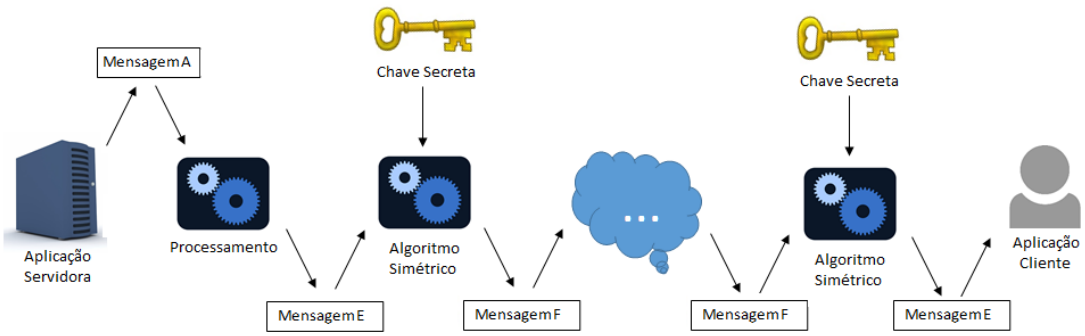


Figura 1. Procedimento de resposta da aplicação servidora à aplicação cliente.

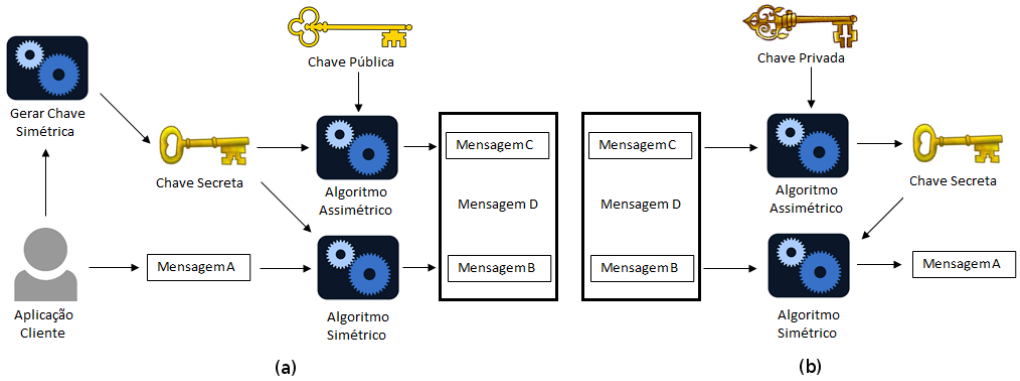


Figura 2. (a) Procedimento de envio de mensagem da aplicação cliente para a aplicação servidora. (b) Procedimento de recebimento da mensagem enviada pela aplicação cliente para a aplicação servidora.

Definido o modelo de criptografia, foram utilizados dois algoritmos considerando os seguintes requisitos:

- Não existirem registros ou previsões de "quebra" do algoritmo em um tempo razoável (todos os algoritmos podem ser quebrados por meio da geração de todas as chaves possíveis, porém, no caso dos algoritmos considerados seguros, isso levaria milhares de anos).
- Possibilidade de implementação nas linguagens Java e PHP, preferencialmente por meio de bibliotecas já existentes.
- Utilização de memória e processamento em quantidade razoável e com tempos de resposta aceitáveis em um dispositivo móvel.

Por meio das classes do pacote *Security*, implementou-se o algoritmo de chave assimétrica RSA, por ser um dos mais comuns atualmente e, portanto, mais testado, oferecendo maior confiabilidade. A desvantagem desse algoritmo, como é comum entre os algoritmos assimétricos, é o alto nível de processamento exigido. Isso ocorre devido à exigência de uma chave de no mínimo 2018 bits [Barker et al. 2016]. Porém, como o texto a ser codificado é pequeno, isso não inviabilizou o processo. Escolheu-se para a codificação simétrica o algoritmo AES, cujo uso é aconselhado pelo NIST (*National Institute of Standards and Technology*) e sua utilização constante atesta alto nível de segurança.

As funcionalidades da aplicação servidora incluem a codificação e decodificação realizadas pelo algoritmo AES-256 e decodificação assimétrica realizada pelo algoritmo RSA. No primeiro caso, a chave utilizada é recebida juntamente com mensagem da aplicação cliente. No caso do algoritmo RSA, é preciso analisar as questões concernentes à manutenção da chave privada, pois esta precisa ser mantida segura por mais tempo.

4. Análise dos Resultados

Por meio das definições de segurança, observou-se que não basta focar na ideia de assegurar o tráfego de dados na rede, mas, para um resultado positivo, se faz necessário analisar todos os componentes envolvidos no desenvolvimento da aplicação. O conceito de segurança abrange tanto as pessoas quanto os ambientes, as práticas seguras dentro de uma empresa, ameaças dentro do próprio dispositivo, acidentes ou intervenções intencionais, uso incorreto da criptografia, e diversos outros fatores e elementos que, de alguma forma, estão envolvidos com o sistema.

Tratando-se do *Android*, várias medidas de segurança foram embutidas na arquitetura da plataforma. Para um aplicativo malicioso ser capaz de acessar outros aplicativos dentro do dispositivo, sem as devidas permissões do usuário, por exemplo, seria preciso "driblar" as configurações do *kernel* do *Linux*. Por conseguinte, se o usuário se policiar e ler atentamente as solicitações de permissão dos aplicativos instalados em seu dispositivo, a probabilidade de acesso indevido às informações são mínimas. Quando a informação sai do dispositivo, porém, está sujeita a todo tipo de ameaça e condições adversas. O desenvolvedor se torna o responsável por estabelecer as medidas de segurança cabíveis, de acordo com o nível exigido pela aplicação. Não convém utilizar-se de todas as técnicas possíveis, gastar muitos recursos financeiros e computacionais, com a finalidade de garantir o máximo de segurança a informações que não necessitam de todas essas precauções. Em contrapartida, o descuido com algum dado sigiloso pode trazer consequências ruins a todos os envolvidos.

No tocante à escolha dos algoritmos adequados à criptografia, um dos fatores é a compatibilidade da linguagem empregada na implementação do aplicativo e a utilizada no

servidor em relação aos padrões de criptografia implementados. É preciso realizar testes com algoritmos diferentes em cada ponta da comunicação para definir os algoritmos que se comportam da mesma maneira em ambas, já que, dependendo das configurações dos *hardwares* e *softwares* utilizados no projeto, o desempenho pode sofrer alterações.

4.1. Geração da Chave Secreta

Elencaram-se duas abordagens para geração da chave secreta utilizada pelo algoritmo AES-256 na aplicação cliente: geração automática por meio da classe *SecretKey*, pertencente ao pacote *Javax.Crypto*, e geração manual por meio da geração de *bytes* aleatórios e um algoritmo de *hash*. Durante a geração da chave manual, é realizado um processamento em quatro etapas:

1. Um número aleatório N pertencente à uma determinada faixa é gerado.
2. É gerado um vetor V de *bytes* com N *bytes* aleatórios.
3. O vetor V é transformado em uma *string* S (cadeia de caracteres).
4. Um algoritmo de *hash* codifica S de modo a gerar uma chave secreta de tamanho padronizado de 256 *bits*.

Em uma chave composta por 256 *bits*, têm-se 7,2 * 10^{16} combinações possíveis. Esse número de possíveis chaves necessitaria, para ser gerado por meio de força bruta, um total de 2300 anos de processamento em um computador que realize um milhão de tentativas por segundo. Portanto, esse tamanho de chave pode ser considerado seguro para a aplicação.

Para identificar o número de *bytes* iniciais adequado, com baixo custo de implementação, para cada uma das etapas foram realizados testes de execução em dois dispositivos, doravante denominados dispositivos Z e M, respectivamente:

1. ZTE GV821 com processador *MediaTek* MT6516 / ARM926EJ-S rev 5 (*Single-Core*), memória interna de 256MB (100MB acessível pelo usuário) e tela de 2,4".
2. Moto G 2015 XT1544 Dual DTV com processador *Qualcomm® SnapdragonTM* 400 com 1,2 GHz *Quad-Core CPU*, memória interna de 16GB e tela de 5".

A medição de tempo foi realizada por meio do método *System.nanoTime()*, dessa forma, é possível registrar dois instantes de tempo dentro da aplicação e contabilizar a diferença entre eles. Em cada caso de teste, contabilizou-se o tempo em microssegundos da operação 12 vezes, eliminaram-se o maior e o menor valor e o resultado consistiu na média simples dos 10 valores restantes. A Tabela 1 contém os resultados da execução de três casos de teste para cada etapa e subetapa nos dispositivos Z e M, de acordo com o número de bytes definido no início do processo.

Para avaliação do comportamento da geração manual de chave secreta, os testes foram realizados com os respectivos valores de N: 2^4, 2^6, 2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, em ambos os dispositivos. Porém, foi considerado suficiente para avaliar esse comportamento, os testes realizados com os valores de N contidos na Tabela 1.

É possível observar que a operação de geração de um número aleatório, que compreende a etapa 1, permanece com valores relativamente constantes, portanto, não influenciará na escolha do tamanho adequado da *string* aleatória a ser gerada. Na segunda etapa são necessárias duas operações: são gerados um vetor de *bytes* de tamanho N (etapa 2.1) e os N *bytes* aleatórios que o populam (etapa 2.2).

Tabela 1. Tempo em microssegundos das operações relacionadas à geração manual da chave secreta e codificação da mesma por meio do algoritmo RSA

	Dispositivo M					Dispositivo Z				
N bytes	2^6	2^{10}	2^{14}	2^{16}	2^{18}	2^6	2^{10}	2^{14}	2^{16}	2^{18}
Gerar um número aleatório entre 0 e o valor indicado										
Tempo	0.83	0.84	0.84	0.84	0.84	7277	7430	8715	7507	7661
em μs	0.84	0.85	0.85	0.85	0.84	8207	7638	9192	7938	8338
Etapa 1	0.84	0.85	0.87	0.85	0.85	8661	8784	9230	9038	9038
Gerar um vetor *bytes*, de acordo com o número de *bytes*										
Tempo	1.23	5.84	79.79	75.14	114.31	11.75	18.18	65.74	10192	95890.28
em μs	1.27	7.37	86.82	90.84	158.14	12.15	19.36	67.12	11309.82	95966.67
Etapa 2.1	1.5	8.52	132.17	111.98	811.47	12.27	28.47	68.91	13507.58	97948.43
Definir o vetor de *bytes* aleatórios de acordo com número de bytes										
Tempo	3.52	44.21	710.65	3004.86	11455.72	82.85	1260.64	27996.84	91493.45	373906.72
em μs	3.54	45.08	868.26	3581.59	12477.51	82.98	2375.29	30816.97	92373.43	376966.35
Etapa 2.2	3.9	45.88	931.14	4198.14	15635.38	83.51	2586.58	32369.28	100232.24	378060.39
Gerar uma *string* aleatória utilizando o comando 'for'										
Tempo	655.14	451441.85	-	-	-	40358.39	10741667.65	-	-	-
em μs	960.05	419311.64	-	-	-	44238.66	11393083.78	-	-	-
Etapa 3.1	1113.98	380543.89	-	-	-	45549.44	11401467.43	-	-	-
Gerar uma *string* aleatória utilizando o comando *new String(bytes)*										
Tempo	8.19	86.22	1278.77	4272.84	17801.98	2131.2	3434.25	31710.46	144877.58	617196.65
em μs	10.3	88.39	1474.02	4312.33	22366.36	2281.96	3732.75	32193.75	146171.34	619690.14
Etapa 3.2	10.37	93.69	1699.11	5172.67	23566.48	2310.92	4041.75	33157.32	154199.73	619850.81
Codificar com o algoritmo SHA-256, de acordo com o tamanho do vetor gerador da *string* aleatória										
Tempo	189.82	308.26	2727.65	7626.91	32833.57	6547.74	8262.78	17875.96	73675.34	298858.29
em μs	249.28	334.95	2850.53	8454.3	33647.35	6622.55	10398.92	22989.62	75996.06	300123.62
Etapa 4	326.33	366.41	2999.22	8646.07	38079.17	8068.58	10981.87	23160.88	77819.77	302769.41
Gerar a chave e codificar com o algoritmo RSA, utilizando *strings*										
Tempo em μs	4549.83	4634.15	7802.76	20117.01	69649.73	23304.95	18339.34	88381.48	287582.66	1210236.5
	4893.42	4715.01	8382.43	22608.76	70304.01	32927.24	19246.14	88651.31	288838.65	1226622.01
	6294.24	6801.11	9170.47	24188.35	71579.13	42622.91	28435.67	94475.22	298755.13	1228257.14
Gerar a chave e codificar com o algoritmo RSA, utilizando vetor de *bytes*										
Tempo em μs	2926.76	3145.29	4368.19	8565.3	22842.56	16597.35	14444.13	40750.84	111597.06	434900.58
	3501.89	4207.16	4823.47	9611.95	25318.18	17728.02	16420.09	42241.4	117569.09	437382.59
	4210.57	5208.31	5614.41	10589.31	29438.92	19477.22	26698.86	44506.43	122697.44	438112.93

Durante a terceira etapa, tendo o vetor de N *bytes*, é preciso montar a *string* aleatória a ser enviada à função *hash*. O primeiro método testado implementou um laço de repetição "for" que, a cada iteração, acrescenta um *byte* à *string* (etapa 3.1). Esse método foi mais custoso, levando mais de 10 segundos para o processamento de 1024 *bytes* no dispositivo Z. O segundo método testado foi a criação de uma *string* aleatória através do comando *new String(bytes)* (etapa 3.2), sendo *bytes*, o vetor de N *bytes* gerado anteriormente. Esse método apresentou melhores resultados que o anterior.

Tendo a *string* aleatória é realizada a quarta etapa e o resultado da função de *hash* é codificado por meio do algoritmo RSA. Por fim, também na Tabela 1, tem-se o tempo em microssegundos da operação completa de geração da chave e criptografia RSA. Calculou-se cada tempo com a utilização de um vetor de *bytes* de tamanho fixo, mas a operação de geração de um número aleatório N contabilizou-se variando entre 0 e 2^{12}. Isso porque, como já mostrado, o valor da operação se mantém semelhante até o limite de *bytes*.

Observou-se que as operações relativas à criação da *string* aleatória são tão custosas quanto ou mais custosas que as operações de criptografia. À vista disso, as operações de criptografia foram modificadas para trabalhar apenas com vetores de *bytes*, possibilitando a remoção da terceira etapa. Assim, o vetor de N *bytes* gerado é enviado à criptografia *hash* diretamente, bem como a chave gerada em *bytes* é enviada diretamente à criptografia RSA. Foram obtidos os resultados exibidos na Tabela 1.

Tabela 2. Informações estatísticas dos dados relativos ao tempo de execução (em μs) das operações de geração automática de chave e geração manual de chave nos dispositivos M e Z.

	Dispositivo M			Dispositivo Z		
Geração de Chave	**Média**	**Mediana**	**Desvio Padrão**	**Média**	**Mediana**	**Desvio Padrão**
Manual	16719,23	17121,79	1131,3	385397,9	386249,1	12709,8
Automática	53,37	52,7	7,4	488,1	486,773	13,3

Dessa forma, é possível realizar todas as operações em menos de meio segundo no dispositivo com menor poder de processamento e com o maior valor de *bytes* mostrado. Definiu-se, então, que a faixa de valores possíveis para N compreenderia de 2^{16} a 2^{18}, o que resulta em 196608 possibilidades de tamanhos para o vetor de *bytes*. É possível observar que, no dispositivo M, utilizando apenas *bytes*, obtém-se um ganho de tempo de aproximadamente 57%, no caso da geração de um vetor de *bytes* do tamanho mínimo possível e de aproximadamente 63,3% no caso de um vetor de tamanho máximo. No dispositivo Z o ganho é de aproximadamente 59,8% e 64,3%, respectivamente.

Seguindo a segunda abordagem, a geração automática da chave é feita por meio de quatro comandos:

```
1 KeyGenerator kg = KeyGenerator.getInstance("AES");
2 kg.init(256);
3 SecretKey sk = kg.generateKey();
4 byte[] chave = sk.getEncoded();
```

Foram realizadas 10 execuções da operação de geração automática de chave e 10 execuções da operação de geração manual de chave em cada dispositivo. A Tabela 2 contém informações estatísticas referentes a esses testes. É possível observar que, em ambos os dispositivos e abordagens, média e mediana são valores muito próximos e o desvio padrão é relativamente pequeno. Assim, podemos concluir que os resultados possuem pouca variação e a média se torna uma medida expressiva para o conjunto de dados. De acordo com a Tabela 2, a geração automática de chave tem um ganho médio em relação à geração manual de aproximadamente 99,87% para o dispositivo Z e 99,68% para o dispositivo M. Em vista disso, após todas as análises relativas à geração de chave, decidiu-se por aplicar a geração automática.

Esse resultado retrata que, apesar de ser possível a geração manual em tempo moderado, a utilização de bibliotecas e métodos nativos da linguagem podem poupar tempo tanto na implementação do aplicativo, quanto na execução. Apesar disso, é importante ter conhecimento do comportamento de soluções alternativas e das possibilidades dentro das limitações dos recursos disponíveis. Apesar de outros testes relativos à técnicas criptográficas terem sido realizados, escolheu-se a análise de geração de chaves para ser descrita, por compor um dos conjuntos de teste que melhor descreve o impacto de cada operação. Nela é possível observar que uma mudança de abordagem ou tipo de variável pode causar efeitos com alta significância na execução do processo.

5. Conclusão e Trabalhos Futuros

Na era da informação, a todo momento, dados estão sendo gerados e consumidos. Informações armazenadas, enviadas e recebidas por meio da Internet são pontos funda-

mentais para a tomada de decisão em todas as partes do planeta. Esse processo está cada vez mais intenso e rápido. Nesse cenário, a análise da segurança dessas informações se torna imprescindível.

Para o uso seguro de uma informação é preciso que esta tenha partido do emissor e chegado ao receptor nas condições devidas, podendo ser armazenada por tempo indeterminado nesse caminho. Com a popularização dos dispositivos móveis, essa movimentação da informação passou a ser feita, em grande parte, por meio deles. Surgem então questionamentos sobre como a segurança da informação deve ser tratada nessas tecnologias e quais seriam os problemas envolvidos.

Deste ponto parte o presente trabalho. Em princípio, buscou-se conhecer o que é preciso saber, em relação à segurança, antes de avaliar ou implementar um aplicativo para dispositivo móvel. Tendo as noções gerais, a segurança da plataforma de desenvolvimento é avaliada. A plataforma utilizada na aplicação desenvolvida proporciona alto nível de segurança das informações processadas localmente no dispositivo, devido ao funcionamento de seu sistema operacional e esquema de permissões, o que rende à comunicação externa maior foco de preocupação do desenvolvedor.

Observou-se que as características próprias da linguagem influenciam diretamente no desempenho das operações. É preciso definir uma maneira adequada para o desenvolvimento que viabilize a execução em tempo aceitável. Na linguagem Java, utilizada pela plataforma *Android*, realizar operações com dados no formato *string* pode ser muito custoso, como constatado nos testes de geração de chaves. Utilizar bibliotecas nativas é uma boa estratégia para otimizar os processos, pois, na maior parte dos casos, possuem implementação eficiente e segura, economizando tempo de desenvolvimento e testes mais detalhados, conferindo ao sistema mais credibilidade e facilidade de manutenção.

O trabalho descreveu possibilidades para a implementação da comunicação direta entre um dispositivo móvel e um servidor, sem a interferência de outras organizações. A confidencialidade da informação é proporcionada pelo modelo de criptografia desenvolvido, que emprega criptografia simétrica, por meio do algoritmo AES, e criptografia assimétrica, por meio do algoritmo RSA.

Como possíveis trabalhos futuros podem-se apontar o maior detalhamento comparativo do desempenho dos algoritmos de criptografia em dispositivos móveis, além do acréscimo de medidas de segurança ao projeto, como certificação digital. Outras possibilidades envolvem a análise de segurança de aplicativos utilizados em outros sistemas operacionais, como IOS, desenvolvido pela empresa *Apple*, e *Windows Phone*, desenvolvido pela *Microsoft*.

Referências

ABNT (2005). Nbr iso/iec 27002: Tecnologia da informação - técnicas de segurança - código de prática para a gestão da segurança da informação. *ABNT*.

ANDROID, O. S. P. (2014). Android security overview.

Barker, E., Barker, W., Burr, W., Polk, W., and Smid, M. (2016). *Computer Security: Recommendation for Key Management – Part 1: General (Revised)*. NIST, 4 edition.

Brasil, M. D. J. (2013). Política de segurança da informação e comunicações do ministério da justiça: Portaria nº. 3.530, de 3 de dezembro de 2013.

Coelho, F. E. S., de Araújo, L. G. S., and Bezerra, E. K. (2014). Gestão da segurança da informação: Nbr 27001 e nbr 27002. *Rnp/esr*.

Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654.

IBGE (2016). Pnad tic: em 2014, pela primeira vez, celulares superaram microcomputadores no acesso domiciliar à internet.

Marciano, J. L. P. (2006). Segurança da informação: Uma abordagem social.

Schneier, B. (2000). *Secrets & Lies: Digital Security in a Networked World*. Wiley Computer Publishing, United States of America.

Six, J. (2012). *Segurança de Aplicativos Android*. Novatec, São Paulo.

Stallings, W. (2008). *Criptografia e Segurança de Redes: Princípios e Práticas*. Pearson Prentice Hall, São Paulo, 4 edition.

LETTY: Uma implementação de *Website Fingerprinting*

Jordan S. Queiroz[1], **Eduardo L. Feitosa**[1]

[1]Instituto de Computação (IComp) – Universidade Federal do Amazonas (UFAM)
CEP 69.077-000 – Manaus – AM – Brasil

{jsq,efeitosa}@icomp.ufam.edu.br

Abstract. *Website Fingerprinting techniques are capable of identify, with high probability, uniquely a device and consequently its user, by extracting device-related characteristics. Normally, those techniques harms user's privacy on the Web and the reason for this is that the Websites which embed Website Fingerprinting don't warn users of its existence nor of its functions. However, these techniques are used for authentication and validation purposes. Controversially, they can be used for malicious purposes (user activity tracking). With the goal to show that it is easy and even simple, this work has the objective to implement a Website Fingerprinting with JavaScript and to evaluate the real world possibilities of identifying uniquely a large number of devices.*

Resumo. *As técnicas de Website Fingerprinting são capazes de identificar, com alta probabilidade, unicamente um dispositivo e consequentemente seu usuário, através da extração de dados relacionados ao dispositivo. Normalmente, tais técnicas ferem a privacidade dos usuários na Web ao não avisar ou deixar claro a sua existência ou função. Contudo, elas são usadas para fins de autenticação e validação de usuários. Por outro lado, também podem ser usadas para fins maliciosos (rastrear atividades dos usuários). Como forma de mostrar que é fácil e até mesmo simples, este trabalho tem como objetivo implementar um Website fingerprinting usando códigos em JavaScript e avaliando a real possibilidade de se identificar unicamente um grande número de dispositivos.*

1. Introdução e Motivação

A Web tornou-se o principal meio para a realização de diversas atividades, graças a sua capacidade de oferecer várias aplicações e serviços, tais como pesquisas, compras, pagamentos online, interação com outras pessoas através de redes sociais, dentre outras [Khademi 2014].

Contudo, essa diversidade tem permitido aos interessados (empresas de propaganda, segurança digital e até mesmo atacantes) utilizar características intrínsecas e diferenças de implementação para obter informações relacionadas ao navegador e ao dispositivo (sistemas operacionais e listas de extensões e plugins, por exemplo) e, consequentemente, identificar o usuário por trás da máquina. Essa identificação é feita com o uso de técnicas de *website fingerprinting*, um conjunto de métodos e algoritmos capazes de identificar um usuário/navegador, com base nas informações do dispositivo [Eckersley 2010].

Em linhas gerais, *website fingerprinting* emprega um conjunto amplo de tecnologias e técnicas usadas para identificar (ou reidentificar) um usuário ou um dispositivo, através de um conjunto de configurações, atributos (tamanho da tela do dispositivo,

versões de software instalado, entre muitos outros) e outras características observáveis durante as comunicações [Saraiva et al. 2014]. A questão é: o quão fácil é obter informações sobre o navegador e o dispositivo de um usuário, a fim de gerar uma identificação única? Assim, o objetivo deste artigo é desenvolver um mecanismo de *website fingerprinting* aplicando técnicas simples, mas capazes de identificar usuários na Internet, comprovando que métodos e atributos específicos do *JavaScript* podem ser usados para implementar tal mecanismo. Para tanto, um protótipo funcional denominado LETTY (*LeT me idenTify You*) foi implementado e aplicado na identificação de usuários. Os resultados mostram que os atributos mais comuns passíveis de coleta em um dispositivo tem a capacidade de gerar um identificador.

O restante do artigo é organizado como segue: na Seção 2 é apresentada uma breve descrição sobre *website fingerprinting*, incluindo seu funcionamento e tecnologias atualmente utilizadas. Na Seção 3 é descrito o projeto e a implementação da LETTY, explicando as tecnologias utilizadas e a dinâmica de funcionamento. Na Seção 4 são apresentados os resultados dos experimentos realizados com a ferramenta. Por fim, a Seção 5 traz uma visão geral do trabalho e melhorias futuras que podem ser feitas.

2. *Website Fingerprinting*

Website fingerprinting é o processo em que as informações do dispositivo e do navegador são coletadas, a princípio, para autenticação e identificação do usuário [Mowery et al. 2011, Khademi 2014, Eckersley 2010], para evitar fraudes de transações [Nikiforakis et al. 2013], para rastrear usuários [Nikiforakis and Acar 2014], para evitar roubo de sessão [Unger et al. 2013], dentre outras aplicações. Também é possível fazer *fingerprinting* apenas do navegador, obtendo informações estritamente relacionadas ao mesmo (versão, família, etc) [Mulazzani et al. 2013] e assim ajustar os conteúdos da página que serão exibidos, a fim de proporcionar uma boa experiência de navegação.

De acordo com a RFC 6973 [Cooper et al. 2013], *fingerprinting* é o processo no qual o observador ou atacante, com uma alta probabilidade, identifica unicamente um dispositivo ou aplicação, baseado na comunicação que ocorre entre os elementos que estão sendo observados.

Alguns autores na literatura consideram *website fingerprinting* como o método usado para descobrir o navegador, sua família e versão [Mulazzani et al. 2013]. Já outros consideram o termo como técnicas e tecnologias utilizadas para identificar um usuário [Eckersley 2010, Khademi 2014]. Neste trabalho, *website fingerprinting* é considerado como um conjunto de técnicas e tecnologias que são capazes de extrair informações de um navegador e usar essas informações para identificar tanto o dispositivo quanto o usuário.

2.1. Classificação

A RFC 6973 [Cooper et al. 2013] preconiza a existência de três tipos de *fingerprinting*: passivo, ativo e *cookie-like*. O **passivo** é baseado nas características observáveis das solicitações Web (cabeçalhos HTTP, endereço IP e outras informações do nível de rede), sem utilização de qualquer código executando no lado do cliente. O **ativo** utiliza códigos dinâmicos (JavaScript, ActionScript ou por qualquer outro tipo de código) no lado do cliente para observar características sobre o navegador. Dentre as características observáveis podem-se destacar a lista de fontes disponível no navegador, o tamanho da

tela, lista de *plugins*, padrões de renderização de gráficos, entre outros. O terceiro e último tipo é o ***cookie-like***, onde um código ou aplicativo é instalado no dispositivo do usuário, permitindo sua (re)identificação e a geração de inferências sobre o mesmo.

Neste trabalho, o tipo **ativo** foi utilizado na construção da ferramenta LETTY, pois as observações das características são realizadas no lado do cliente, com o uso da linguagem JavaScript.

2.2. Tecnologias

Para criar um mecanismo ou algoritmo de *website fingerprinting* é necessário utilizar alguma tecnologia Web disponível. Dentre as comumente empregadas estão:

- JavaScript, que tem acesso fácil e direto a diferentes APIs que fornecem informações relevantes sobre navegador e dispositivo em que o navegador está sendo executado (por exemplo, lista de *plugins* e sistema operacional) [Khademi 2014].
- Canvas, que é empregado para desenhar elementos 2D, mas que também permite a recuperação do conteúdo desenhado e a consequente identificação do navegador e do sistema operacional [Khademi 2014].
- Flash, que assim como JavaScript, possui métodos que permitem o acesso a várias informações que estão estritamente relacionadas ao sistema em que o navegador está sendo executado. Por exemplo, pode ser obtido o nome do sistema operacional, arquitetura da CPU, número de DPI (*Dots Per Inch*) da tela, dentre outras características [Khademi 2014].
- CSS, que se vale das diferentes versões e implementações dos navegadores para avaliar diferenças na *engine* de *layout* a fim de identificar o navegador [Unger et al. 2013].
- Cabeçalho HTTP, uma vez que existem uma série de informações que caracterizam um navegador e são suficientes para sua identificação [Unger et al. 2013].

3. Projeto e Implementação

Esta Seção apresenta os atributos coletados e a visão geral do projeto, incluindo detalhes da implementação da ferramenta LETTY.

3.1. Objetos e Atributos

Os dois principais objetos empregados na identificação do navegador são o *screen* e o *navigator*. O primeiro possui informações sobre as configurações de tela em que as páginas estão sendo renderizadas pelo navegador, enquanto o segundo possui informações sobre o ambiente em que o navegador está sendo executado, bem como suas propriedades.

Os atributos obtidos através dos objetos *screen* e *navigator* são exibidos nas Tabelas 1 e 2, respectivamente.

3.2. Visão geral

A ferramenta LETTY foi projetada de forma a ser integrada a qualquer *website* com bastante facilidade. Por isso, sua implementação fez uso da linguagem JavaScript, que permite coletar informações de ambos objetos e armazená-las em qualquer estrutura de dados. Além disso, segundo [Khademi 2014], o *fingerprint* gerado com o uso do JavaScript

Tabela 1. Atributos do objeto *screen*.

Propriedade	Descrição
colorDepth	Contém a profundidade de cores utilizadas para renderizar imagens na tela do dispositivo em que o navegador está sendo executado.
pixelDepth	Contém a resolução de cores da tela do dispositivo em que o navegador está sendo executado.
width	Contém a largura da tela do dispositivo em que o navegador está sendo executado.
height	Contém a altura da tela do dispositivo em que o navegador está sendo executado.

Tabela 2. Atributos do objeto *navigator*.

Propriedade	Descrição
userAgent	Contém informações sobre nome, versão e plataforma do navegador.
product	Contém o nome da *engine* do navegador.
productSub	Contém informação sobre o número da versão de desenvolvimento da *engine* do navegador.
cookieEnabled	Retorna *true* ou *false* se os *cookies* são permitidos ou se não são permitidos.
vendor	Informa quem é o desenvolvedor do navegador.
platform	Contém informação sobre a plataforma para a qual o navegador foi compilado.
language	Contém informação sobre a linguagem atual do navegador.
languages	Contém informação sobre as linguagens disponíveis do navegador.
javaEnabled	Informa se o navegador possui ou não o Java ativado.
appName	Contém o nome do navegador.
appCodeName	Contém o codinome do navegador.
appVersion	Contém a versão do navegador.
oscpu	Contem a arquitetura do processador do computador em que o navegador está sendo executado.
maxTouchPoints	Contém o número máximo de toques suportados pela tela do dispositivo em que o navegador está sendo executado.
plugins	Contém os *plugins* instalados no navegador.
mimeTypes	Contém a lista de tipos MIME suportados pelo navegador web.

possui alta entropia (podendo chegar até 9.97 bits). Quanto maior a entropia, maiores as chances de um dado dispositivo ser identificado unicamente. A Figura 1 ilustra a dinâmica de funcionamento da LETTY.

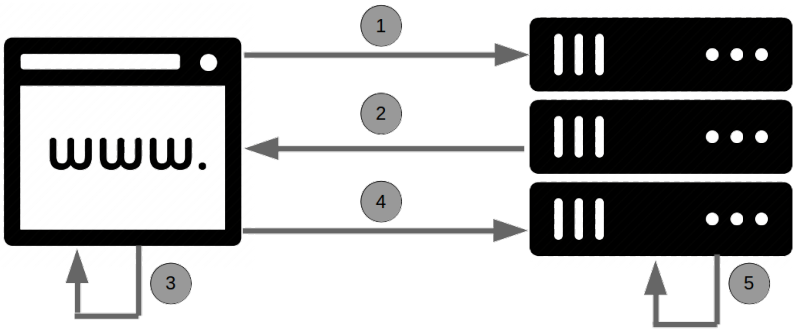


Figura 1. *Workflow* da LETTY.

No **passo 1**, o cliente (usuário) solicita acesso a uma página Web (contendo a LETTY). Como resposta (**passo 2**), o servidor Web envia a página solicitada (conteúdo), incluindo o código de *fingerpinting*. Ao chegar no navegador (**passo 3**) do cliente, a LETTY começa a executar, coletando os dados disponíveis no navegador. Primeiramente, os atributos do objeto *navigator* ($userAgent$, $product$, $productSub$, $cookieEnabled$, $vendor$, $platform$, $language$, $languages$, $javaEnabled$, $appName$, $appCodeName$, $appVersion$, $oscpu$ e $maxTouchPoints$) são coletados. Em seguida, é a vez dos atributos do objeto *screen* ($colorDepth$, $pixelDepth$, $width$ e $heigth$). De-

pois os *plugins* presentes no navegador ($navigator.plugins$) e os *mimetypes* suportados ($navigator.mimeTypes$) são coletados. Embora ambos estejam ligados ao objeto *navigator*, sua coleta é diferenciada visto que ambos são representados por vetores de atributos (e não características únicas como os demais).

No **passo 4**, a LETTY devolve ao servidor os dados coletados. No servidor, os dados são processados e a chave de identificação única do usuário/dispositivo é gerada (**passo 5**). Para gerar a chave, uma função de *hashing*, da biblioteca Cripto.js[1], foi utilizada. Embora se saiba que podem ocorrer colisões em tais funções, as chances dependendo da função escolhida podem ser extremamente baixas. Para tanto, foi empregada uma função *hash* MD5 de 128 bits, cuja probabilidade de colisão é estimada em ocorrer quando existirem 2^{64} entradas [Krawczyk et al. 1997]. Vale ressaltar que um banco de dados pode ser empregado para guardar os dados coletados dos clientes.

4. Avaliação e Resultados Experimentais

Para comprovar sua efetividade, a LETTY foi incorporada em uma página Web hospedada em um servidor da Universidade (https://200.17.49.135/pesquisa/letty). Os experimentos foram realizados durante 4 semanas, entre os dias 6 de outubro e 9 de novembro de 2015. No final, foram realizados 230 acessos à página com 119 dispositivos identificados unicamente.

Em linhas gerais, quando a página é acessada, é exibida uma mensagem explicando rapidamente o experimento e o propósito do mesmo. A LETTY só realiza o *fingerprinting* do usuário caso o mesmo concorde em participar do experimento. Uma vez aceito, os dados do navegador são recolhidos, enviados ao servidor (onde o identificador é gerado) e os valores são então exibidos para o usuário que visitou a página. A Figura 2 ilustra um resultado final exibido para o usuário.

4.1. Resultados

Antes de apresentar de fato o que foi descoberto, é importante ressaltar que como alguns atributos sempre apresentam os mesmos valores, seus resultados não são apresentados. Por exemplo, todos os navegadores que acessaram a página permitem o uso de *cookies* (*cookieEnabled* = *yes*) e também do Java (*javaEnabled* = yes). *appCodeName* representa o codinome do navegador e, por questões de compatibilidade, todos os navegadores modernos retornam Mozilla [W3Schools 2016].

Plataformas e navegadores

A Tabela 3 ilustra as plataformas e os navegadores dos dispositivos que foram usados para acessar a página Web. Todos os dados coletados foram obtidos através dos métodos *userAgent* e *appVersion*, do objeto *navigator*.

Percebe-se na Tabela 3 que uma variedade de navegadores foram utilizados para acessar a página da ferramenta LETTY. Segundo os dados apresentados, o navegador Chrome é o mais prevalente (63 aparições), seguido do Firefox (27 aparições) e Safari (10 aparições). Sobre o navegador Chrome, foram encontradas tanto versões mais recentes (50, 51 e 52) quanto versões mais antigas (36, 41, 42 e 43), todas com diferentes *releases*.

[1]https://www.npmjs.com/package/crypto-js

Attribute	Content
User Agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.134 Safari/537.36
Product	Gecko
Product Sub	20030107
Cookie Enabled	true
Vendor	Google Inc.
Platform	Linux x86_64
Language	pt-BR
Languages	pt-BR,pt,en-US,en
Java Enabled	function javaEnabled() { [native code] }
App Name	Netscape
App CodeName	Mozilla
App Version	5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.134 Safari/537.36
Oscpu	undefined
Mime Types	application/pdf,application/x-shockwave-flash,application/futuresplash,application/vnd.chromium.remoting-viewer,application/x-ppapi-widevine-cdm,application/googletalk,application/o1d,application/x-nacl,application/x-pnacl,application/x-google-chrome-pdf
Device	desktop
Key	8d13d3322e556fb2669ffe220537da05

Figura 2. Exemplo de resposta dada ao usuário sobre os dados coletados com a LETTY.

Tabela 3. Plataformas e Navegadores

Plataformas x Navegadores	Chrome	Firefox	Safari	Opera Coast	Samsung Browser	Blackberry Browser	LG Netcast 3	Internet Explorer	Edge	Epiphany	Miui Browser	Chromium	Amazon Silk 3	Desconhecido
Windows 10	13	9	0	0	0	0	0	1	3	0	0	0	0	0
Widows 8.x	3	1	0	0	0	0	0	0	0	0	0	0	0	0
Windows 7	9	3	0	0	0	0	0	1	0	0	0	0	0	0
Windows Vista	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Linux	11	8	0	0	0	0	0	0	0	0	0	1	0	0
Android 6.x	6	0	0	0	0	0	0	0	0	0	0	0	0	0
Android 5.x	10	0	0	0	1	0	0	0	0	0	0	0	0	0
Android 4.x	5	2	0	0	1	0	0	0	0	0	1	0	1	1
Android 2.x	0	0	0	0	1	0	0	0	0	0	0	0	0	0
iOS 9.3	1	1	4	1	0	0	0	0	0	0	0	0	0	0
iOS 9.0	0	1	2	0	0	0	0	0	0	0	0	0	0	0
Mac OS X	4	2	4	0	0	0	0	0	0	0	0	0	0	0
Blackberry	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Raspberry Pi	0	0	0	0	0	0	0	0	0	1	0	0	0	0
LG WebOS	0	0	0	0	0	0	1	0	0	0	0	0	0	0
TOTAL	**63**	**27**	**10**	**1**	**3**	**1**	**1**	**2**	**3**	**1**	**1**	**1**	**1**	**1**

Além dos dados mais frequentes, a Tabela 3 também apresenta plataformas e navegadores não tão usuais (BlackBerry, Raspberry Pi, Opera Cost e Amazon Silk 3). Essas características peculiares destacam ainda mais o dispositivo que passou pelo processo

de *fingerprinting*. Um exemplo mais visual é apresentado com a *string* do *userAgent* retornado pelo navegador Epiphany, utilizado pelo Raspberry Pi: **Mozilla/5.0 (Macintosh; ARM Mac OS X) AppleWebKit/538.15 (KHTML, like Gecko) Safari/538.15 Version/6.0 Raspbian/8.0 (1:3.8.2.0-0rpi27rpi1g) Epiphany/3.8.2**.

A primeira vista, pode-se pensar que o acesso foi oriundo de algum dispositivo rodando Mac OS ou iOS. No entanto, Epiphany é um navegador que não está disponível para esses sistemas.

Por fim, embora o experimento tenha coletado 119 *fingerprints* no total, apenas 116 são exibidos na Tabela, uma vez que três dos *fingerprints* coletados pertencem a robôs indexadores de página. Além disso, um único navegador foi classificado como desconhecido, pois os métodos empregados (*userAgent* e *appVersion*) não foram capazes de identificá-lo. Explicando melhor, a *string* retornada pelo método *userAgent* o apresenta como sendo Safari, mas o acesso foi feito a partir de um dispositivo Android. Embora não seja conclusivo, pode-se atribuir tal comportamento a alguma solução de segurança capaz de mudar a *string* do *userAgent*, retornando valores diferentes a cada requisição.

Resolução

Na coleta realizada com a LETTY, foram encontradas diferentes resoluções de tela dos dispositivos que acessaram a página, conforme ilustrado na Figura 3. Vale ressaltar que a resolução retornada representa o que é exibido pelo navegador, não incluindo resoluções de fotos e vídeos.

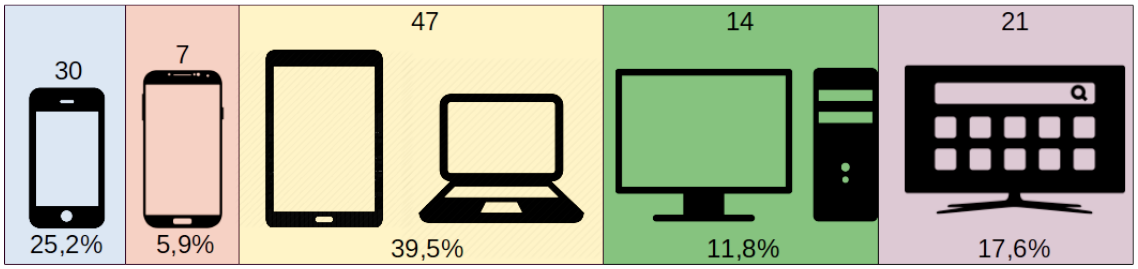


Figura 3. Quantidade, dimensão de tela e percentagem de cada tipo de dispositivo.

Foram identificados 30 dispositivos com telas de pequenas dimensões, variando de 320x534 até 384x640, representando 25,2% do total encontrado. Também foram identificados 7 dispositivos com dimensões entre 412x732 e 800x1270, perfazendo 5,9%. Esses 37 dispositivos são telefones celulares e smartphones executando Android, iOS ou Blackberry OS.

Os *notebooks* e *tablets* formam a maior faixa dos identificados, totalizando 47 dispositivos (39,5%) e com variação de tela indo de 1000x600 até 1366x768. Neste intervalo se enquadra o *smartphone* BlackBerry apresentado na seção de plataformas e navegadores. Os *desktops* identificados representam 11,8% dos dispositivos (14), apresentando resoluções entre 1438x808 e 1824x984.

Por fim, a última categoria é composta por *SmartTVs* e *desktops* ou *notebooks* de alta resolução ou usando mais de uma tela. Foram identificados 21 dispositivos com resolução entre 1916x979 e 1920x1200. Nesta última categoria também enquadram-se

os robôs indexadores que visitaram a página da LETTY (não considerados na seção de plataformas e navegadores).

Vale ressaltar que os valores das resoluções foram obtidos através dos métodos *width* e *height* do objeto *screen*.

Plugins

Além de obter dados sobre as plataformas, navegadores e resoluções, foi possível também obter a lista de *plugins* que os usuários utilizam em seus navegadores. Os *plugins* coletados durante o experimento estão ilustrados na Figura 4. Quanto maior for o tamanho da fonte, maior o número de ocorrências do *plugin* no conjunto dos resultados analisados.

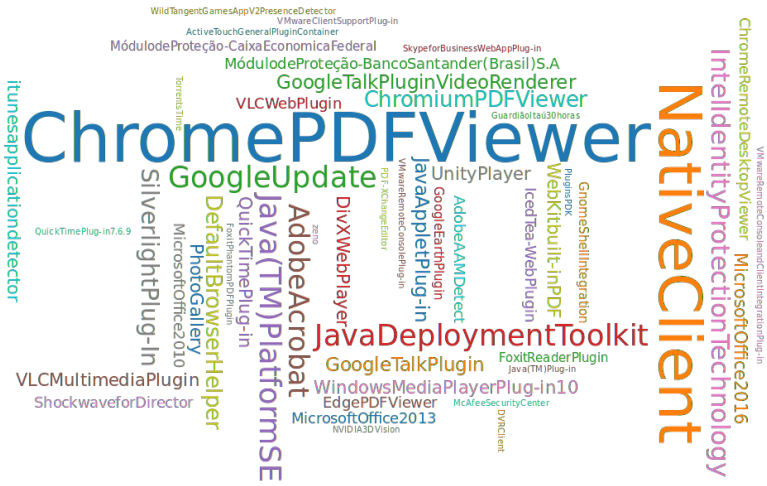


Figura 4. Nuvem de palavras com a lista de *Plugins*.

Obviamente, percebe-se que na Figura 4 existe uma diversidade de *plugins* nos navegadores modernos. Contudo, o preocupante é a questão da privacidade e segurança. Embora, a primeira vista, saber os *plugins* que o navegador de um usuário suporta não represente qualquer ameaça, o processo de coleta da LETTY foi capaz de encontrar *plugins* específicos relacionados à *Internet banking*. Foram encontrados os módulos de proteção dos bancos Caixa Econômica Federal e Santander.

Outros Resultados

Uma vez que a função da LETTY é testar a capacidade de coleta de informações sobre o navegador e o dispositivo, foi realizado um experimento para avaliar a resposta da LETTY em um ambiente com máquinas clonadas (máquinas com as mesmas configurações de *hardware* e contendo os mesmos *softwares* instalados de uma única vez). Neste experimento foi realizado em um laboratório contendo 16 máquinas idênticas.

Como resultado, apenas a primeira máquina a acessar a página contendo a LETTY foi identificada e teve seu identificador gerado. Todas as outras 15 máquinas foram vistas como sendo acessos repetidos da primeira, demonstrando que apesar de representativos, os atributos empregados no *fingerprinting* precisam ser melhorados.

5. Conclusões

Fingerprinting é um conjunto de técnicas que podem ser usadas tanto para fins benignos (autenticação, prover melhor experiência de uso, etc) quanto maliciosos (invasão de privacidade, explorar vulnerabilidades, dentre outros), uma vez que permitem obter, com certo nível de facilidade e relativa precisão, informações sobre o dispositivo e o usuário. Além disso, técnicas de *fingerprinting* voltadas a Web não precisam, a princípio, armazenar nenhum tipo de informação no dispositivo do usuário, diferentemente dos *cookies*. Grande parte dessa facilidade é devida a disponibilidade de APIs capazes de retornar dados relacionados à máquina do usuário. Dentre as tecnologias utilizadas em *fingerprinting*, a linguagem JavaScript é sem dúvida a mais usada por ser capaz de fornecer um variado número de informações (versão do navegador, lista de *plugins*, resolução de tela, número máximo de toques suportados pela tela, sistema operacional, entre outras) e assim identificar unicamente um dispositivo.

Este trabalho implementou um *fingerprinting* através dos objetos *navigator* e *screen*, utilizando JavaScript, e provou que é possível tirar vantagem de informações que todo navegador precisa responder, e ainda mais, mostrou que é simples obter tais dados e consequentemente identificar unicamente um dispositivo. A ferramenta desenvolvida neste trabalho (LETTY) foi construída com média quantidade de esforço, sendo capaz de descobrir características de todos os dispositivos que passaram pelo processo de *fingerprinting*. Além disso, um identificador único para cada dispositivo foi gerado e uma base com dados coletados também.

Contudo, este trabalho também mostrou que existem casos que precisam ser tratados, pois a identificação do usuário pode mudar por diversos fatores (presença de duas telas, atualização na versão do navegador, uso de um mecanismo de segurança, entre outros).

5.1. Trabalhos futuros

Embora o *fingerprinting* implementado neste trabalho tenha servido aos seus propósitos, é possível melhorá-lo para deixar a solução mais robusta. Um primeiro melhoramento é tentar descobrir se houve mudanças no dispositivo. Já foi provado em [Eckersley 2010] que é possível criar um algoritmo para verificar se um dispositivo com um novo *fingerprint* na verdade é um antigo dispositivo que teve a sua chave mudada por algum motivo. No entanto, achar uma solução ótima ainda é um trabalho em aberto.

Um segundo trabalho é de fato escolher atributos mais robustos e que retornem informações únicas para geração do identificador. O *fingerprinting* implementado neste trabalho utiliza o atributo *userAgent*, que pode ser mudado facilmente com o uso de extensões, alterando, assim, sua identificação. Por exemplo, [Nakibly et al. 2015] propuseram um *fingerprinting* baseado nos dispositivos de entrada e saída de áudio. A justificativa dos autores é que o microfone e fone possuem frequências diferentes, dependendo de como são projetados. Essas lacunas, combinadas a alguma inconsistência durante o processo de fabricação, por menor que seja, contribuem para identificar e rastrear um usuário e suas atividades.

5.2. Disponibilidade da ferramenta

O repositório da LETTY está disponível no seguinte link: https://github.com/Jordan-Queiroz/Letty.git.

5.3. Agradecimentos

Este trabalho teve apoio do Programa de Capacitação de Recursos Humanos em Tecnologias da Informação e Computação e em Sistemas e Aplicativos para Plataformas Tecnológicas Portáteis, Móveis e Distribuídas; número 930000, financiado pela Samsung Eletrônica da Amazônia.

Referências

Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., M., H., and Smith, R. (2013). Privacy Considerations for Internet Protocols. RFC 6973, RFC Editor.

Eckersley, P. (2010). How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 1–18, Berlin, Heidelberg. Springer-Verlag.

Khademi, A. F. (2014). Browser fingerprinting: Analysis, detection, and prevention at runtime. Master's thesis, Queen's University.

Krawczyk, H., Bellare, M., and Canetti, R. (1997). Hmac: Keyed-hashing for message authentication. RFC 2104, RFC Editor.

Mowery, K., Bogenreif, D., Yilek, S., and Shacham, H. (2011). Fingerprinting information in JavaScript implementations. In Wang, H., editor, *Proceedings of W2SP 2011*. IEEE Computer Society.

Mulazzani, M., Reschl, P., Huber, M., Leithner, M., Schrittwieser, S., Weippl, E., and Wien, F. (2013). Fast and reliable browser identification with javascript engine fingerprinting. In *Web 2.0 Workshop on Security and Privacy (W2SP)*, volume 5.

Nakibly, G., Shelef, G., and Yudilevich, S. (2015). Hardware fingerprinting using HTML5. *CoRR*, abs/1503.01408.

Nikiforakis, N. and Acar, G. (2014). Browse at your own risk. *IEEE Spectrum*, 51(8):30–35.

Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., and Vigna, G. (2013). Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 541–555, Washington, DC, USA. IEEE Computer Society.

Saraiva, A., Elleres, P., Carneiro, G., and Feitosa, E. (2014). Device Fingerprinting: Conceitos e Técnicas, Exemplos e Contramedidas. In dos Santos, A., van de Graaf, J., Nogueira, J. M., and Oliveira, L. B., editors, *Livro de Minicursos do XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg2014*, chapter 2, pages 49–98. SBC.

Unger, T., Mulazzani, M., Frühwirt, D., Huber, M., Schrittwieser, S., and Weippl, E. (2013). Shpf: Enhancing http (s) session security with browser fingerprinting. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pages 255–261. IEEE.

W3Schools (2016). Navigator appcodename property. `http://www.w3schools.com/jsref/prop_nav_appcodename.asp`. Acesso 20/06/2016.

Análise Formal de Requisitos de Consistência para Atribuições de Valoração

Frederico P. Santiago[1], **Mário S. Alvim**[2]

[1] Universidade Federal de Minas Gerais, Departamento de Engenharia Elétrica
frede-stg@ufmg.br

[2]Universidade Federal de Minas Gerais, Departamento de Ciência da Computação
msalvim@dcc.ufmg.br

Abstract. *Quantitative information flow aims to assess and control the leakage of sensitive information by computer systems. The community has recently turned its attention to semantic-based measures of information, which employ worth assignments to attribute a different worth to each portion of the secret, depending on its sensitivity. Algorithms for automatically deriving worth-assignments from a scenario of interest must be guided by a set of consistency requirements, which are expensive to compute and enforce. In this paper we analyze several candidates for consistency requirements, identifying subsets of redundant requirements and reducing, hence, the computational effort required by the algorithm for automatically deriving appropriate worth assignments.*

Resumo. *O campo de fluxo de informação quantitativo (QIF, do inglês* quantitative information flow*) concerne à mensuração da informação sigilosa que vaza por meio de sistemas computacionais. A comunidade recentemente tem se focado em métricas de informação baseadas em semântica, que empregam atribuições de valoração para atribuir a cada porção do segredo uma valoração correspondente à sua importância. Algoritmos para derivar atribuições de valoração a partir de um cenário de interesse devem ser guiados por um conjunto de requisitos de consistência, que são caros de se computarem. Neste artigo analisamos vários candidatos a requisitos de consistência, identificando subconjuntos de requisitos redundantes e reduzindo, assim, o esforço computacional para o algoritmo de derivação automática de atribuições de valoração.*

1. Introdução

O campo de *fluxo de informação quantitativo* (QIF, do inglês *quantitative information flow*) concerne à mensuração da informação sigilosa que vaza, para um adversário não-autorizado, por meio de sistemas computacionais. Assume-se que o adversário possui *informação a priori* sobre os segredos antes de a execução do sistema ser iniciada—modelada como uma distribuição de probabilidades sobre os segredos—, e que ele tem acesso aos *efeitos observáveis* do sistema em execução. Combinando a informação *a priori* e os observáveis públicos, o adversário obtém *informação a posteriori* sobre os segredos—modelada como uma coleção de distribuições de probabilidades sobre segredos condicionadas às observações feitas. O *vazamento de informação* de uma execução é computado como a diferença entre a informação *a posteriori* e a informação *a priori*.

Esta definição de vazamento depende de como informação é medida, e diferentes *métricas de informação* quantificam diferentes aspectos de interesse do cenário analisado.

Métricas populares no campo de QIF incluem *entropia de Shannon* [Clark et al. 2005, Malacaria 2007, Moskowitz et al. 2003, Chatzikokolakis et al. 2008, Alvim et al. 2012a], que mede o número esperado de tentativas que um adversário precisa para identificar o segredo usando pesquisa binária; *guessing entropy* [Massey 1994, Malacaria 2015], que mede o número esperado de tentativas que um adversário precisa para identificar o segredo usando pesquisa linear; e *probabilidade de acerto* [Smith 2009, Braun et al. 2009], que mede a probabilidade de o segredo ser inferido em um certo número de tentativas.

Todas estas métricas, entretanto, possuem a limitação de considerar todos os segredos como atômicos e de igual valor, e recentemente a comunidade tem considerado cenários mais ricos. O arcabouço de *g-leakage* [Alvim et al. 2012b, McIver et al. 2014, Alvim et al. 2014a, Alvim et al. 2016] utiliza funções de ganho para quantificar o benefício de diferentes prognósticos do adversário sobre o segredo. Escolhendo uma função de ganho adequada é possível modelar a importância de cada porção do segredo. Entretanto, identificar funções de ganho suficientemente expressivas, porém não complexas demais, é uma tarefa desafiadora. Em [Alvim et al. 2014b] são identificadas várias dimensões utilizadas na interpretação operacional de métricas clássicas de informação—o *tipo* e a *quantidade* de questões que o adversário pode utilizar, assim como a sua *probabilidade de acerto*—e propõem a inclusão de uma nova dimensão: a *valoração* de cada parte do segredo. Os autores consideram que um segredo possa ser particionado em *campos*, que podem ser combinados para formar *estruturas*. Como o vazamento de estruturas distintas corresponde a ameaças distintas, uma *atribuição de valoração* é utilizada para associar uma *valoração* a cada estrutura. Por exemplo, o segredo correspondendo à conta bancária de um cliente pode conter duas estruturas de 8 dígitos: sua senha de internet e seu número de telefone. Como o vazamento da senha tem o potencial de causar um dano considerável, a estrutura correspondente teria uma valoração alta; como o número de telefone é público, a estrutura correspondente teria uma valoração baixa.

Usando atribuições de valoração, Alvim et al. generalizaram métricas tradicionais para refletir a semântica dos segredos, e provaram diversas propriedades destas novas métricas. Entretanto, tais generalizações assumem a existência uma atribuição de valoração adequada para cada cenário de interesse. A fim de sintetizar semi-automaticamente atribuições de valoração a partir de um cenário de interesse, Alvim et al. propuseram um algoritmo iterativo que recebe como entrada informações básicas sobre o cenário de interesse—i.e., a distribuição de probabilidaade a priori sobre os segredos e quais estruturas são consideradas intrinsecamente sensíveis—e deriva a valoração das demais estruturas condicionadas a um conjunto de *requisitos de consistência*, que são propriedades matemáticas que garantem o bom comportamento das atribuições de valoração. Vários requisitos de consistência podem ser considerados, como, por exemplo, a não-negatividade, a monotonicidade, e a propriedade de que se uma estrutura "carrega informação sobre uma outra", então sua valoração deve refletir esta "informação compartilhada". O algoritmo de Alvim et al., entretanto, é indiferente aos requisitos de consistência adotados, e simplesmente tenta propagar a valoração entre estruturas garantindo um requisito por vez, em sequência, o que pode levar a dois problemas durante a execução do algoritmo: (i) se os requisitos adotados não forem consistentes entre si, o algoritmo pode divergir; e (ii) mesmo que o conjunto de requisitos seja consistente, se ele for redundante, o algoritmo se torna desnecessariamente lento por testar individualmente requisitos que já são implicados por outros requisitos já verificados.

Neste trabalho realizamos uma análise formal de requisitos de consistência para atribuições de valoração que pode guiar o projetista do sistema a escolher um conjunto de requisitos auto-consistentes e, a partir deste conjunto, decidir se existe uma ordem de verificação entre estes requisitos que torne o algoritmo de síntese de atribuições de valoração mais eficiente, ou se os requisitos escolhidos devem ser necessariamente checados individualmente. Mais especificamente, nossas principais contribuições são:

1. Uma análise da relevância de diversas propriedades matemáticas candidatas a requisitos de consistência, incluindo: não-negatividade, monotonicidade, princípio da inclusão-exclusão, princípio da correlação e separabilidade estatística;
2. Uma análise da auto-consistência entre tais requisitos. Em particular, demonstramos que a negatividade e a correlação produzem sempre distribuições triviais incompatíveis com o requisito de separabilidade estatística.
3. Uma análise da redundância entre conjuntos de requisitos, identificando se a satisfabilidade de um conjunto de requisitos implica necessariamente a satisfabilidade de outros requisitos. Em particular, demonstramos que à parte de casos triviais: correlação implica não-negatividade e monotonicidade; correlação e monotonicidade implicam não-negatividade; e correlação e não-negatividade implicam monotonicidade.
4. Relacionado ao item acima, uma análise de quais requisitos de consistência não permitem otimização na checagem. Em particular, demonstramos que correlação deve ser checada independentemente de monotonicidade, não-negatividade e separabilidade estatística.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta os conceitos fundamentais de fluxo de informação quantitativo e o arcabouço de Alvim et al. de métricas baseadas em atribuições de valoração. A Seção 3 formaliza candidatos a requisitos de consistência e analisa a relevância prática de cada um, enquanto na Seção 4 investiga a relação entre conjuntos de requisitos, identificando subconjuntos inconsistentes e subconjuntos redundantes. Por fim, a Seção 5 discute trabalhos futuros e conclui.

2. Preliminares

Nesta seção revisamos os fundamentos do campo de fluxo de informação quantitativo e o arcabouço de Alvim et al. [Alvim et al. 2014b] de métricas baseadas em atribuições de valoração.

2.1. Segredos, estruturas e atribuições de valoração

Seguindo [Alvim et al. 2014b], consideramos que cada segredo pode ser decomposto em partes atômicas chamadas de *campos*. O conjunto finito desses campos é dado por $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$, e o domínio de cada campo é o conjunto de todos os possíveis valores que o campo pode assumir, denotado por $domain(f_i)$, $1 \leq i \leq m$. Uma *estrutura* é um subconjunto $\mathfrak{f} \subseteq \mathcal{F}$. Se $\mathfrak{f} = \{f_{i_1}, f_{i_2}, \ldots, f_{i_k}\}$ então $domain(\mathfrak{f}) = domain(f_{i_1}) \times \cdots \times domain(f_{i_k})$. O conjunto de todas as possíveis estruturas é o conjunto potência $\mathcal{P}(\mathcal{F})$ dos campos. Chamamos $\mathfrak{f}$ de *estrutura máxima* quando $\mathfrak{f} = \mathcal{F}$.

Um *segredo* s é um mapeamento da estrutura máxima para valores na forma $s = \langle s[f_1], \ldots, s[f_m] \rangle$, onde $s[f_i] \in domain(f_i)$ é o valor assumido pelo campo f_i.

Desta forma, o conjunto $\mathcal{S}$ de todos os possíveis segredos é igual a $domain(\mathcal{F})$. Quando fazemos a projeção de um segredo s em uma estrutura $\mathfrak{f}$, obtemos o *subsegredo* $s[\mathfrak{f}]$. O conjunto de todos os subsegredos associados a uma estrutura $\mathfrak{f}$ é $\mathcal{S}[\mathfrak{f}] = domain(\mathfrak{f})$.

Exemplo 1. Considere um sistema bancário em que o segredo é composto pelos campos $\mathcal{F} = \{c, s, d\}$, representando, respectivamente, o número da conta c, a senha s da conta, e a data de nascimento d de cada correntista. O domínio $domain(c)$ de contas é o conjunto de todos os identificadores de contas, o domínio $domain(s)$ de senhas é o conjunto de todos os números de 4 dígitos, e o domínio $domain(d)$ de datas de nascimento são todos os dias de $01/01/1\,900$ até hoje. Como o conjunto de estruturas é o conjunto potência $\mathcal{P}(\mathcal{F})$ do conjunto de campos, podemos representar todas as estruturas em um reticulado em que a relação de ordem parcial é derivada da relação de inclusão em conjuntos. A Figura 1 representa o reticulado correspondente ao exemplo bancário. Note que a estrutura $\{c, s\}$ (conta e senha do correntista) está incluída na estrutura $\{c, s, d\}$ (a informação completa sobre o correntista), mas ela não tem relação de ordem com a estrutura $\{d\}$ (a data de nascimento do correntista). Considere o segredo $s = \langle \texttt{CONTA-ABC}, 2510, 11/06/1996\rangle$, representando um correntista nascido em 11/06/1996, dono da conta "CONTA-ABC", e tendo como senha "2510". Podemos isolar o subsegredo $s[\{c, s\}] = \langle \texttt{CONTA-ABC}, 2510\rangle$ correspondente à sua conta e senha, e o subsegredo $s[d] = \langle 11/06/1996\rangle$ correspondente à sua data de nascimento. □

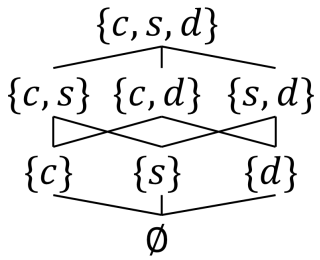


Figura 1. Reticulado de estruturas do Exemplo 1.

As estruturas podem ter valores associados com a informação que carregam. Uma *atribuição de valoração* é uma função $\omega : \mathcal{P}(\mathcal{F}) \to \mathbb{R}$ do conjunto de todas as estruturas para os reais. A valoração pode ser interpretada como o ganho do adversário ao tomar conhecimento de uma estrutura ou como o dano que o protetor do segredo sofre caso o adversário tome conhecimento desta estrutura. Por exemplo, no Exemplo 1 para refletir o fato de que a senha de um correntista é mais sensível que sua data de nascimento, podemos atribuir uma valoração tal que $\omega(\{s\}) > \omega(\{d\})$.

A atribuição de valoração deve representar apropriadamente a sensibilidade da estrutura no cenário em que está inserida, levando em consideração três aspectos relevantes: (i) o quanto o protetor do segredo se preocupa em proteger o conteúdo da estrutura em questão; (ii) qual é o ganho do adversário em descobrir esta estrutura; e (iii) o quanto esta estrutura revela a respeito de outras. Adotando uma atribuição de valoração apropriada, também é possível modelar outros aspectos de interesse, tais como: *vazamento baseado em semântica* (o impacto do vazamento é um reflexo da qualidade da informação vazada, e não do número de bits vazados); *compartilhamento de segredos* ou *secret sharing* (estruturas que isoladamente carregam pouca informação podem carregar muita informação se reveladas em conjunto); e *correlação de estruturas* (se uma estrutura carrega informação sobre outra, sua valoração deve refletir isso).

2.2. Métricas de informação baseadas em atribuições de valoração

Usando atribuições de valoração é possível generalizar métricas clássicas, como entropia de Shannon, *guessing entropy* e probabilidade de acerto para considerar segredos não-

atômicos e seus aspectos semânticos. [Alvim et al. 2014b] classifica as métricas com base em valoração em três categorias:

- W*-Measures* medem a valoração extraída do sistema em um ataque com número fixo de tentativas e uma determinada probabilidade de sucesso do adversário que deve ser atingida. E.g., *worth of certainty*, *W-vulnerability* e *worth of expectation*.
- P*-Measures* medem a probabilidade de sucesso do adversário em um ataque com um valor determinado de valoração extraída e número de tentativas fixo. E.g., *W-probability of guessing*.
- N*-Measures* medem quantas tentativas são necessárias para que o adversário extraia uma certa quantidade de valoração do sistema e tenha uma determinada probabilidade de sucesso. E.g., W*-guessing entropy* e W*-Shannon entropy*.

2.3. Um algoritmo para sintetizar atribuições de valoração

Todas as métricas baseadas em atribuição de valoração assumem que exista uma atribuição de valoração que capture os aspectos relevantes do cenário de interesse. [Alvim et al. 2014b] propõe um algoritmo para a sintetização de atribuições de valoração a partir de três aspectos relevantes do cenário de interesse: o conhecimento do adversário, requisitos de segredo e os requisitos de consistência.

- **Conhecimento do adversário** é o conjunto de informações relevantes ao adversário sobre o sistema que será atacado. Essas informações vêm de fontes externas e constituem o conhecimento *a priori* do adversário. A configuração do conhecimento do adversário é modelada como uma distribuição de probabilidades sobre as estruturas.
- **Requisitos de segredo** transmitem as preocupações do protetor do segredo em relação às estruturas que o compõem. Toda estrutura pode ser classificada como *intrinsecamente sensível*, isto é, possui valor por si só, ou como *contingencialmente sensível*, que possui valor apenas pela informação que revela sobre as intrinsecamente sensíveis. Os requisitos de segredo mapeiam todas as estruturas intrinsecamente sensíveis a uma valoração considerada apropriada pelo projetista do sistema. A relação entre as valorações de estruturas intrinsecamente sensíveis norteia o algoritmo na atribuição de valoração a todas as outras de maneira consistente com os interesses do projetista. No Exemplo 1 a senha do cliente deve ser considerada intrinsecamente sensível, já que é informação chave para o acesso à conta. Já a data de nascimento do correntista não é imprescindível para o acesso, mas sabe-se que existe uma parcela razoável de usuários de sistemas protegidos por senha que utilizam suas datas de nascimento como senha, o que faz da data de nascimento uma estrutura contingencialmente sensível.
- **Requisitos de consistência** são propriedades consideradas essenciais para garantir o significado operacional da valoração e para guiar o algoritmo na atribuição das valorações pelo reticulado.

Provido com as entradas acima, o algoritmo proposto segue os seguintes passos.

1. Construir o reticulado de estruturas completo.
2. Usar os requisitos de segredo para atribuir as valorações adequadas para as estruturas intrinsecamente sensíveis.

3. Derivar a distribuição de probabilidade p_S. Pode-se considerar que as partições do reticulado são variáveis aleatórias e usar p_S para encontrar as distribuições de probabilidades das estruturas.
4. Considerar uma métrica ν bem estabelecida e consistente com o cenário em questão e, para toda estrutura do reticulado, atualizar sua valoração respeitando os requisitos de consistência. Repetir até que todas as estruturas possuam valorações consistentes com os requisitos de consistência impostos.

O ponto crítico do algoritmo acima é o passo 4, em que se verificam iterativamente os requisitos de consistência. E.g., para testar se toda a distribuição é não negativa, basta uma comparação por elemento de $\mathcal{P}(\mathcal{F})$, mas para testar correlação deve-se garantir que cada elemento está devidamente relacionado a todos os outros do reticulado. O que pode se tornar uma atividade bastante custosa computacionalmente. Nas próximas seções vamos reduzir o número de requisitos de consistência para os quais devemos testar as distribuições sem abdicar das propriedades de interesse.

3. Os requisitos de consistência

Nesta seção formalizamos e avaliamos a significância prática de várias propriedades matemáticas candidatas a serem requisitos de consistência.

Os requisitos de consistência impostos à atribuição da valoração podem ser variados, contanto que sejam de interesse de quem constrói o sistema e que não haja conflitos entre as propriedades desejadas. [Alvim et al. 2014b] propõe algumas propriedades possivelmente relevantes. São elas: não-negatividade, monotonicidade, inclusão-exclusão e separabilidade estatística (no artigo original, chamado de independência). Além destes, consideraremos o princípio da correlação em nossos estudos.

- **Não-negatividade (NNG).** Nenhuma estrutura possui valoração negativa,

$$\forall \mathfrak{f} \in \mathcal{P}(\mathcal{F}) : \omega(\mathfrak{f}) \geq 0.$$

- **Monotonicidade (MNT).** Uma estrutura deve carregar, no mínimo, tanta informação quanto qualquer uma de suas subestruturas.

$$\forall \mathfrak{f}, \mathfrak{f}' \in \mathcal{P}(\mathcal{F}) : \mathfrak{f} \subseteq \mathfrak{f}' \Rightarrow \omega(\mathfrak{f}') \geq \omega(\mathfrak{f}).$$

- **Separabilidade Estatística (SEP).** Se duas estruturas $\mathfrak{f}$ e $\mathfrak{f}'$ forem *estatisticamente independentes* (denotado por $\mathfrak{f} \perp \mathfrak{f}'$), isto é, não compartilham informação entre si, então:

$$\omega(\mathfrak{f}, \mathfrak{f}') = \omega(\mathfrak{f}) + \omega(\mathfrak{f}').$$

- **Correlação (COR).** A correlação condiciona a valoração de uma estrutura à quantidade de informação que ela fornece a respeito de outras estruturas. A métrica de correlação que utilizaremos neste estudo é a *vulnerabilidade condicional*. A vulnerabilidade $V(\mathfrak{f})$ nos fornece a probabilidade de acerto de uma estrutura $\mathfrak{f}$ em apenas uma tentativa. Formalmente,

$$V(\mathfrak{f}) = \max_{x \in domain(\mathfrak{f})} p(s[\mathfrak{f}] = x).$$

A vulnerabilidade condicional $V(\mathfrak{f}' \mid \mathfrak{f})$, portanto, nos fornece a probabilidade de acerto da estrutura $\mathfrak{f}'$ dado que sabemos o conteúdo da estrutura $\mathfrak{f}$. Formalmente,

$$V(\mathfrak{f}' \mid \mathfrak{f} = x) = \max_{x' \in domain(\mathfrak{f}')} p(s[\mathfrak{f}'] = x' \mid \mathfrak{f} = x). \quad (1)$$

A propriedade da correlação é dada por

$$\forall \mathfrak{f}, \mathfrak{f}' \in \mathcal{P}(\mathcal{F}) : \omega(\mathfrak{f}) \geq V(\mathfrak{f}' \mid \mathfrak{f}) \cdot \omega(\mathfrak{f}').$$

Esta propriedade é interessante por creditar às estruturas suas influências nas valorações de todo o reticulado.

- **Inclusão-Exclusão (IE).** Utilizado largamente em outras métricas, o princípio da inclusão-exclusão proposto é da forma:

$$\forall \mathfrak{f}, \mathfrak{f}' \in \mathcal{P}(\mathcal{F}) : \omega(\mathfrak{f} \cup \mathfrak{f}') = \omega(\mathfrak{f}) + \omega(\mathfrak{f}') - \omega(\mathfrak{f} \cap \mathfrak{f}').$$

Percebeu-se, porém, que esta não é uma propriedade desejável no sistema, visto que sua imposição contraria propriedades mais elementares. E.g., sejam o número da conta bancária de um indivíduo e sua senha os dois campos que constituem um segredo. É razoável condicionar que o conhecimento da conta e da senha tenha uma valoração consideravelmente maior que a soma das valorações das partes. Mas pela definição da inclusão-exclusão, a maior valoração que a junção de duas estruturas pode atingir é a soma das valorações de cada estrutura.

4. As relações entre os requisitos

Nesta seção reportamos os principais resultados do nosso estudo: as relações de auto-consistência e redundância identificadas entre requisitos de consistência. Em particular, mostramos que para todo caso não trivial, o requisito de correlação implica não-negatividade e monotonicidade. Mostramos, também, que a separabilidade estatística não implica nenhuma das outras propriedades e que a correlação não é derivável das demais consideradas.

Começaremos definindo funções distribuição de valoração triviais.

Definição 2. *Uma atribuição de valoração $\omega : \mathcal{P}(\mathcal{F}) \to \mathbb{R}$ é trivial (TRIV) se, e somente se $\forall \mathfrak{f}, \mathfrak{f}' \in \mathcal{P}(\mathcal{F}) : \omega(\mathfrak{f}) = \omega(\mathfrak{f}')$.*

Nas demonstrações que se seguem, faremos referência constante ao cenário-base apresentado no exemplo abaixo..

Exemplo 3. Considere o cenário em que haja dois campos $\mathcal{F} = \{f_1, f_2\}$, gerando o conjunto de estruturas $\mathcal{P}(\mathcal{F}) = \{\emptyset, \{f_1\}, \{f_2\}, \{f_1, f_2\}\}$; Por simplicidade, vamos chamar $\mathfrak{f} = \{f_1\}$, $\mathfrak{f}' = \{f_2\}$ e $\mathfrak{f}'' = \{f_1, f_2\}$. A Figura 2 representa o reticulado de estruturas para este exemplo. Assuma que os campos são binários, i.e., que $domain(f_1) = domain(f_2) = \{0, 1\}$, de forma que o conjunto de todos os segredos possíveis seja $\mathcal{S} = \{00, 01, 10, 11\}$. A Tabela 1(a) apresenta uma distribuição de probabilidades sobre os

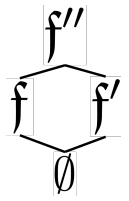


Figura 2. Reticulado das estruturas do Exemplo 3.

Tabela 1. Campos estatisticamente independentes para o Exemplo 3.

(a) Distribuição de probabilidades do segredo.

$s[f_1]$	$s[f_2]$	p_s
0	0	$1/6$
0	1	$1/3$
1	0	$1/6$
1	1	$1/3$

(b) Vulnerabilidades condicionais $V(a \mid b)$.

$b \backslash a$	$\emptyset$	$\mathfrak{f}$	$\mathfrak{f}'$	$\mathfrak{f}''$
$\emptyset$	1	$1/2$	$1/2$	$1/3$
$\mathfrak{f}$	1	1	$2/3$	$2/3$
$\mathfrak{f}'$	1	$2/3$	1	$2/3$
$\mathfrak{f}''$	1	1	1	1

segredos, e a Tabela 1(b) apresenta as vulnerabilidades condicionais (veja Equação (1)) entre cada par de campos. Note que neste exemplo temos que os campos são estatisticamente independentes ($f_1 \perp f_2$). □

Também precisaremos do seguinte resultado auxiliar.

Lema 4. *Se uma atribuição de valoração ω satisfaz o requisito de correlação, então não pode haver duas estruturas com valoração de sinais diferentes. Formalmente, temos que COR $\Rightarrow \forall \mathfrak{f}, \mathfrak{f}' \in \mathcal{P}(\mathcal{F}) : \omega(\mathfrak{f}) \cdot \omega(\mathfrak{f}') \geq 0$.*

Demonstração. Considere estruturas $\mathfrak{f}_1, \ldots, \mathfrak{f}_n$ e $\mathfrak{f}$ de um segredo tais que $\mathfrak{f}_1 \cup \ldots \cup \mathfrak{f}_n \subseteq \mathfrak{f}$ (Note que fazendo $n \geq 1$, temos que sempre podemos encontrar tal conjunto de estruturas). Se a atribuição de valoração ω satisfizer COR, então para $i = 1, \ldots, n$ temos que $\omega(\mathfrak{f}_i) \geq V(\mathfrak{f} \mid \mathfrak{f}_i) \cdot \omega(\mathfrak{f})$ e que $\omega(\mathfrak{f}) \geq V(\mathfrak{f}_i \mid \mathfrak{f}) \cdot \omega(\mathfrak{f}_i)$. Lembrando que a vulnerabilidade condicional $V(a \mid b)$ é sempre não-negativa, note que há dois casos a se analisar para o sinal de $\mathfrak{f}$: se $\omega(\mathfrak{f}) \geq 0$, por COR temos que $\omega(\mathfrak{f}_i) \geq 0$; e se $\omega(\mathfrak{f}) < 0$, por COR temos que $\omega(\mathfrak{f}_i) \leq \omega(\mathfrak{f}) < 0$. Isso implica que todas as estruturas $\mathfrak{f}_1, \ldots, \mathfrak{f}_n$ devem ter o mesmo sinal que a estrutura $\mathfrak{f}$. Escolhendo $\mathfrak{f}$ como a estrutura máxima do reticulado, que contém todas as demais, segue que todas as estruturas de um reticulado devem possuir sinal igual à estrutura máxima. □

Agora estamos prontos para enunciar nossos resultados principais.

Proposição 5 ($\neg$TRIV $\wedge$ COR $\Rightarrow$ NNG)**.** *Se uma atribuição de valoração ω não-trivial satisfaz o requisito de correlação, então ω necessariamente também satisfaz o requisito de não-negatividade.*

Demonstração. Vamos demonstrar a proposição equivalente de que se uma atribuição de valoração ω satisfizer a propriedade de correlação e possuir ao menos uma estrutura de valor negativo, então ω deve necessariamente ser trivial (i.e., COR $\wedge$ $\neg$NNG $\Rightarrow$ TRIV).

Considere um reticulado que contenha ao menos uma estrutura de valoração negativa. Pelo Lema 4, todas as demais estruturas devem também possuir valorações negativas. Assumindo a correlação, $\forall \mathfrak{f}, \mathfrak{f}' \in \mathcal{P}(\mathcal{F})$ temos que $V(\mathfrak{f}' \mid \mathfrak{f}) \cdot \omega(\mathfrak{f}') \leq \omega(\mathfrak{f}) < 0$ e que $V(\mathfrak{f} \mid \mathfrak{f}') \cdot \omega(\mathfrak{f}) \leq \omega(\mathfrak{f}') < 0$. Como $\omega(\mathfrak{f})$ e $\omega(\mathfrak{f}')$ são negativos, é verdade que $\omega(\mathfrak{f}') \leq \omega(\mathfrak{f}) < 0$ e que $\omega(\mathfrak{f}) \leq \omega(\mathfrak{f}') < 0$. Mas note que como temos $\omega(\mathfrak{f}') \leq \omega(\mathfrak{f})$ e $\omega(\mathfrak{f}) \leq \omega(\mathfrak{f}')$, então $\omega(\mathfrak{f}) = \omega(\mathfrak{f}')$. Como a igualdade é válida para quaisquer $\mathfrak{f}$ e $\mathfrak{f}'$, a distribuição é trivial. □

Proposição 6. *(COR $\wedge$ NNG $\Rightarrow$ MNT) Se uma atribuição de valoração ω não-trivial satisfaz os requisitos de correlação e não-negatividade, então ω necessariamente também satisfaz monotonicidade.*

Demonstração. Se $\mathfrak{f}' \subseteq \mathfrak{f} \Rightarrow V(\mathfrak{f}' \mid \mathfrak{f}) = 1$. Lançando mão de COR e assumindo a NNG, temos que $\omega(\mathfrak{f}) \geq V(\mathfrak{f}' \mid \mathfrak{f}) \cdot \omega(\mathfrak{f}') = \omega(\mathfrak{f}')$, que é exatamente a definição de monotonicidade.

Note que a condição de não-negatividade é essencial, pois o fator $V(\mathfrak{f}' \mid \mathfrak{f})$ reduz $\omega(\mathfrak{f}')$ em módulo. Se $\omega(\mathfrak{f}') < 0$, a redução de seu módulo significa em aumento de seu valor de forma que $V(\mathfrak{f}' \mid \mathfrak{f}) \cdot \omega(\mathfrak{f}') \geq \omega(\mathfrak{f}')$. □

Proposição 7. *(MNT∧NNG∧SEP ⇏ COR) A propriedade da correlação não é derivável de monotonicidade, não-negatividade ou separabilidade estatística, e nem de nenhuma combinação entre estas.*

Demonstração. Como contra exemplo, tome o cenário do Exemplo 3 e considere a atribuição de valoração ω tal que $\omega(\emptyset) = 0$, $\omega(\mathfrak{f}) = 5$, $\omega(\mathfrak{f}') = 5$ e $\omega(\mathfrak{f}'') = 10$. Note que NNG e MNT são satisfeitas em ω, assim como SEP (por construção, os campos f_1 e f_2 são independentes e ω satisfaz $\omega(\mathfrak{f}'') = 10 = 5 + 5 = \omega(\mathfrak{f}) + \omega(\mathfrak{f}')$). Entretanto, COR não é satisfeita: note que $\omega(\emptyset) < V(\mathfrak{f} \mid \emptyset) \cdot \omega(\mathfrak{f})$. □

Proposição 8. *(MNT∧NNG∧COR ⇏ SEP) A propriedade da separabilidade estatística não é derivável de monotonicidade, correlação ou não-negatividade e nem de combinações entre estas.*

Demonstração. Já sabemos pela Proposição 6 que COR ∧ NNG ⇒ MNT. Então basta demonstrar que NNG∧COR ⇏ SEP, o que faremos com o contra-exemplo a seguir. Tome novamente o cenário do Exemplo 3, e considere uma atribução de valoração ω tal que $\omega(\emptyset) = 4$, $\omega(\mathfrak{f}) = 7$, $\omega(\mathfrak{f}') = 5$ e $\omega(\mathfrak{f}'') = 10$. Note que COR, NNG (e, consequentemente, MNT) são satisfeitas, porém SEP é violada, pois $\omega(\mathfrak{f}'') \neq \omega(\mathfrak{f}) + \omega(\mathfrak{f}')$. □

Proposição 9. *(SEP ⇏ MNT ∨ NNG ∨ COR) A separabilidade estatística não implica monotonicidade, correlação nem não-negatividade.*

Demonstração. Como contra-exemplo, tome o cenário do Exemplo 3 e considere a atribução de valoração ω tal que $\omega(\emptyset) = -5$, $\omega(\mathfrak{f}) = -5$, $\omega(\mathfrak{f}') = -5$ e Perceba que SEP é satisfeita, pois $\omega(\mathfrak{f}'') = \omega(\mathfrak{f}) + \omega(\mathfrak{f}')$, mas a distribuição é negativa (NNG desrespeitada) e não-trivial, o que implica, pela Proposição 5, que COR não é satisfeita. Além disto, $\omega(\mathfrak{f}'') < \omega(\mathfrak{f})$. Ou seja, a monotonicidade também é desrespeitada. □

Neste ponto observamos que apesar de apenas as Proposições 5 e 6 apresentarem resultados positivos (i.e., determinarem uma ordem de precedência entre requisitos), todas as proposições apresentadas são úteis para auxiliar o projetista do sistema na atribuição de valoração. Resultados negativos demonstram que certos conjuntos de requisitos são inconsistentes (e.g., a Proposição 5 demonstra que se houver ao menos uma valoração negativa na distribuição, assumindo correlação, esta distribuição deve ser trivial) ou que, mesmo se consistentes, alguns conjuntos de requisitos demandam que a checagem seja feita individualmente (e.g., a Proposição 7 demonstra que correlação deve ser checada independentemente de monotonicidade, não-negatividade e separabilidade estatística).

5. Conclusões e trabalhos futuros

Neste trabalho analisamos a relevância de vários requisitos de consistência para atribuições de valoração, e identificamos vários conjuntos de requisitos inconsistentes entre si ou redundantes.

Os resultados obtidos levam a dois pontos fundamentais na otimização do algoritmo de síntese de atribuições de valoração a partir de um cenário de interesse: Em primeiro lugar, é suficiente que testemos apenas COR, NNG e SEP para garantir a validade destas e de MNT, o que aumenta a eficiência do algoritmo. Em segundo lugar, não podemos abrir mão de NNG na distribuição sem gerar trivialidade, se considerarmos COR ou gerar inconsistência, se considerarmos COR e SEP.

Como trabalho corrente estamos investigando as consequências de assumir a independência dos campos na definição de separabilidade estatística, buscando maiores reduções no esforço computacional para este caso particular. Também propomos métricas que captam a relação entre as estruturas de maneiras diferentes como funções de correlação e estudamos suas consequências na atribuição de valoração. Por fim, pretendemos analisar quantitativamente a redução do esforço computacional atingida com a eliminação das redundâncias entre requisitos de consistência identificadas.

Referências

Alvim, M., Chatzikokolakis, K., McIver, A., Morgan, C., Palamidessi, C., and Smith, G. (2014a). Additive and multiplicative notions of leakage, and their capacities. In *Computer Security Foundations Symposium (CSF), 2014 IEEE 27th*, pages 308–322.

Alvim, M., Chatzikokolakis, K., McIver, A., Morgan, C., Palamidessi, C., and Smith, G. (2016). Axioms for information leakage. In *Computer Security Foundations Symposium (CSF), 2016 IEEE 29th*, pages 77–92.

Alvim, M. S., Andrés, M. E., and Palamidessi, C. (2012a). Quantitative information flow in interactive systems. *Journal of Computer Security*, 20(1):3–50.

Alvim, M. S., Chatzikokolakis, K., Palamidessi, C., and Smith, G. (2012b). Measuring information leakage using generalized gain functions. In *Proc. 25th IEEE Computer Security Foundations Symposium (CSF 2012)*, pages 265–279.

Alvim, M. S., Scedrov, A., and Schneider, F. B. (2014b). When not all bits are equal: Worth-based information flow. In *Proc. 3rd Conference on Principles of Security and Trust (POST 2014)*, pages 120–139.

Braun, C., Chatzikokolakis, K., and Palamidessi, C. (2009). Quantitative notions of leakage for one-try attacks. In *Proceedings of the 25th Conf. on Mathematical Foundations of Programming Semantics*, volume 249 of *Electronic Notes in Theoretical Computer Science*, pages 75–91. Elsevier B.V.

Chatzikokolakis, K., Palamidessi, C., and Panangaden, P. (2008). Anonymity protocols as noisy channels. *Inf. and Comp.*, 206(2–4):378–401.

Clark, D., Hunt, S., and Malacaria, P. (2005). Quantitative information flow, relations and polymorphic types. *J. of Logic and Computation*, 18(2):181–199.

Malacaria, P. (2007). Assessing security threats of looping constructs. In Hofmann, M. and Felleisen, M., editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007*, pages 225–235. ACM.

Malacaria, P. (2015). Algebraic foundations for quantitative information flow. *Mathematical Structures in Computer Science*, 25(2):404–428.

Massey (1994). Guessing and entropy. In *Proceedings of the IEEE International Symposium on Information Theory*, page 204. IEEE.

McIver, A., Morgan, C., Smith, G., Espinoza, B., and Meinicke, L. (2014). Abstract channels and their robust information-leakage ordering. In *Proc. 3rd Conference on Principles of Security and Trust (POST 2014)*, pages 83–102.

Moskowitz, I. S., Newman, R. E., Crepeau, D. P., and Miller, A. R. (2003). Covert channels and anonymizing networks. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*, WPES '03, pages 79–88, New York, NY, USA. ACM.

Smith, G. (2009). On the foundations of quantitative information flow. In de Alfaro, L., editor, *Proc. 12th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS '09)*, volume 5504 of *Lecture Notes in Computer Science*, pages 288–302.

Forense computacional em discos de estado sólido: desafios e perspectivas

João Vitor A. Ribeiro[1], **João Gondim**[1]

[1]Departamento de Ciência da Computação – Universidade de Brasília (UnB)
Campus Darcy Ribeiro Edifício CIC/EST Asa Norte Caixa Postal 4466

jaavrr@gmail.com,gondim@unb.br

Abstract. *An analysis of the usability and the effectiveness of the standard practices in Data Recovery and Digital Forensics in a SSD - solid state disk using common data and file recovery tools and a black-box methodology is presented. The process of recovering data from a SSD using common tools of Data and File Recovery, Data Carving and with background knowledge in File Systems and in SSD technology, provided the basis for a comparative analysis with the results found in the bibliography. Results suggest that the technology used in SSDs can hinder or even prevent Forensic Analysis on Linux and Windows systems.*

Resumo. *Este trabalho apresenta uma análise da usabilidade e da eficácia de técnicas usuais de recuperação de dados e de forense digital em um SSD - disco de estado sólido utilizando ferramentas usuais de recuperação de dados e uma metodologia black-box. É feito um estudo de caso em um SSD utilizando ferramentas usuais de recuperação de dados, data carving e informações pertinentes a tecnologia dos dispositivos e sistemas de arquivos. Também é realizada uma análise comparativa com os resultados da bibliografia. Os resultados obtidos sugerem que as tecnologias utilizadas em SSDs podem dificultar ou até inviabilizar a análise forense nos Sistemas Operacionais Windows e Linux.*

1. Introdução

Nos últimos tempos, tem-se verificado um crescimento no uso dos discos de estado sólido - SSD *Solid State Disk* em detrimento dos discos rígidos (*Hard Disk Drive*). No entanto, os SSDs apresentam diversos desafios à forense computacional em comparação com os discos rígidos tradicionais. Entre eles, mecanismos internos que podem dificultar ou inviabilizar a recuperação de dados excluídos nessas mídias, podendo levar à perda de informação que pode constituir potenciais evidências.

São dois os principais fatores que podem levar a essa perda: o processo de coleta de lixo e o comando TRIM. Isso se deve em parte por conta do fato de os SSDs poderem fazer uso desses mecanismos de forma autônoma, ou seja, sem a necessidade da ação de um Sistema Operacional (SO) ou do usuário, sendo também possível que esses mecanismos sejam acionados pelos agentes citados. [Bell and Boddington 2010].

O trabalho se constitui num estudo de habilidade da aplicação das técnicas e ferramentas forenses abertas utilizadas em HDD's quando aplicadas a SSD's. Esse estudo levou a uma análise comparativa com os resultados mais recentes da área.

2. Discos de Estado Sólido

Nos últimos anos, os SSDs têm se tornado uma tecnologia emergente. SSDs são baseados em circuitos integrados, fornecendo diversas vantagens como menor consumo de energia, menor tamanho, resistência a choques e a característica mais impotante de alto desempenho no acesso a dados aleatórios [Chen et al. 2009].

A maioria dos SSDs atuais utiliza a arquitetura NAND planar, ou seja em duas dimensões. Porém já se encontram no mercado dispositivos com a tecnologia V-NAND (*Vertical* NAND) na qual tem-se o empilhamento de camadas de células na vertical, proporcionando maior densidade e capacidade de armazenamento [Samsung 2016].

2.1. Arquitetura de um SSD

Os SSDs são normalmente construídos utilizando arranjos de memória *Flash*, que são ligados por um barramento serial a uma **controladora** [Chen et al. 2009]. A controladora é um dos elementos mais importantes de um SSD, pois é responsável pela execução de rotinas e controle de mecanismos internos. A controladora recebe e processa requisições do Sistema de Arquivos por meio de uma interface de conexão como, por exemplo, a interface **SATA** - *Serial AT Attachment*.

É possível pensar a arquitetura de um SSD em três camadas (abstrações). São elas: HIL - *Host Interface Layer*, FTL - *Flash Translation Layer* e FIL - *Flash Interface Layer*. A Figura 1 ilustra essa divisão.

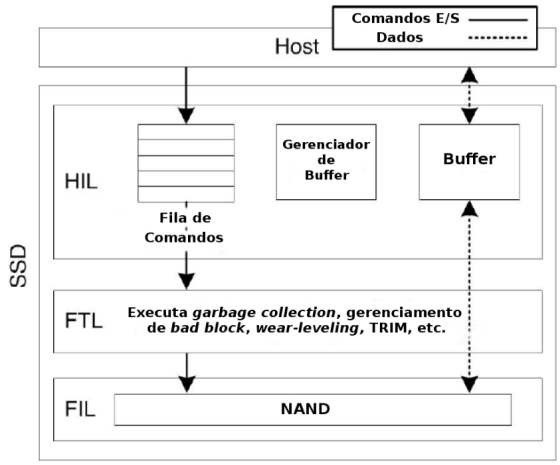


Figura 1. Arquitetura de um SSD
[Junior and Queiroz 2015b]

Na *Host Interface Layer*, comandos de E/S (Entrada/Saída) fornecidos pelo Sistema de Arquivos e/ou SO são enfileirados e atendidos na ordem em que chegam. Na *Flash Translation Layer* ocorre gerência de certos mecanismos internos utilizados para otimizar o desempenho e por emular o funcionamento de um HDD em termos lógicos, possibilitando a compatibilidade e a interoperabilidade entre os sistemas computacionais. Como sumariza [Chen et al. 2009], essa camada desempenha três papeis fundamentais: **Mapeamento lógico de blocos, Nívelamento de desgaste e Coleta de lixo**.

Por fim a *Flash Interface Layer*, camada de nível mais baixo, é constituída em essência pelas memórias NAND. Nela, são realizadas as operações de leitura, escrita e apagamento. Sendo que na leitura a menor unidade é a página. A escrita também é geralmente realizada em termos de páginas, e o apagamento é feito em nível de blocos [Chen et al. 2009].

2.2. Mapeamento lógico de blocos

Os discos de estado sólido operam tipicamente pelo armazenamento de dados em blocos de 512 kB, subdivididos em páginas de 4 kb, resultando em 128 páginas por bloco nessa configuração [Olson and Langlois 2008]. Tais blocos são constituídos de arranjos de transístores *NAND* semelhantes aos chips utilizados em processadores.

2.3. Nivelamento de desgaste

Geralmente as operações de escrita apresentam certa localidade, ou seja, frequentemente as mesmas regiões de armazenamento são escritas (princípio da localidade espacial). Naturalmente, esse princípio se aplica às requisições de blocos do SSD, onde um conjunto de blocos (lógicos) é constantemente referenciado para escrita pelo Sistema de Arquivos.

Como as memórias *Flash* possuem um número limitado de ciclos *erase-write*, foi concebido o mecanismo de nivelamento de desgaste (*wear leveling*) que uniformiza essas operações pelos blocos físicos do disco. Logo, escritas subsequentes em um bloco lógico na realidade correspondem a escritas em blocos diferentes (a controladora copia o conteúdo de um bloco físico para outro de modo a evitar que o primeiro seja sobrescrito). A Figura 2 ilustra a atuação do nivelamento de desgaste nas células de um SSD.

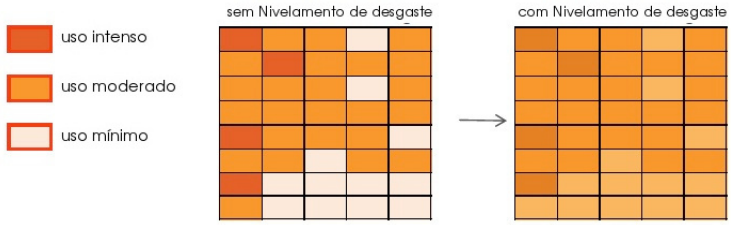


Figura 2. Nivelamento de desgaste
Adaptada de [Nazarian and Dubois 2013]

2.4. Coleta de lixo

Os dados são escritos na granularidade de páginas, porém em geral apenas blocos inteiros podem ser apagados [Bell and Boddington 2010]. À medida em que o SSD vai sendo utilizado, restam menos blocos sem páginas livres por conta do nivelamento de desgaste.

Dessa forma, quando se deseja reutilizar uma página que contém dados desatualizados, é necessário copiar as páginas válidas desse bloco para outro bloco na mídia, para que o primeiro seja apagado. Esse processo é custoso e é um dos grandes limitadores de desempenho no longo prazo.

Para amenizar esse problema, foram concebidas rotinas para a controladora do SSD que identificam blocos contendo páginas não utilizadas ou contendo dados desatualizados, antes de uma operação de escrita. Após a identificação, a controladora realiza

o procedimento explicado acima sempre que possível. Essa rotina recebe o nome de ”coleta de lixo”, porém na verdade a rotina faz a ”coleta” tanto de dados dados válidos quanto inválidos. Entretanto há um problema central pois a controladora do SSD ”não sabe” o que são dados inválidos (ou lixo), já que essa informação é dependente do Sistema de Arquivos e do SO e cada combinação desses define um conjunto de regras para o remanejo dos blocos. O SO não comunica à controladora os blocos passíveis de apagamento. [Bell and Boddington 2010] conjecturam que após a cópia de uma página pelo mecanismo do *wear leveling*, esta está sujeita a coleta de lixo em um momento oportuno. Nesse cenário, seria possível o apagamento automático sem a comunicação do SO.

Existem, portanto, dois tipos principais de coleta de lixo: a *background garbage collection* e a *filesystem-aware garbage collection*. A primeira trata do caso mais geral, onde foi enviada uma requisição de escrita a um bloco em uso. Já a segunda se refere à coleta de lixo onde a controladora ”ciente” do Sistema de Arquivos em uso, verifica o histórico de arquivos apagados e informações globais sobre o Sistema de Arquivos. Para lidar com o cenário mais geral, foi criado o comando TRIM que será abordado na Seção 3.

2.5. Provisionamento

Nos SSDs, o provisionamento é um mecanismo que objetiva aumentar o tempo de vida do dispositivo, atuando junto com o nivelamento de desgaste e com a coleta de lixo. Algumas áreas (blocos) são reservados para a controladora, sendo estas chamadas de área de sobre-provisionamento ou em inglês, *over-provisioning area* (OP). Esses blocos podem ser utilizados quando uma operação de escrita é enviada, requisitando bloco(s) para escrita. Esta é realizada na OP, enquanto que um bloco *dirty* (inválido para escrita, ou apagado) entra na área de provisionamento. Os blocos *dirty* são apagados (coleta de lixo) quando o SSD encontra-se em estado de baixa atividade [Gubanov and Afonin 2014].

Por conta do provisionamento, os SSDs possuem uma quantidade de memória física maior do que o especificado (ou reconhecido pelo SO), porém a memória efetiva continua a mesma.

3. Comando TRIM

O comando TRIM (em português aparar ou podar) é um comando no nível de SO que sinaliza à controladora que existem blocos não utilizados (que foram excluídos) para que esta apague previamente os blocos de modo a antecipar posteriores escritas, pois estas requerem um apagamento prévio. A Figura 3 ilustra o funcionamento do TRIM.

Cada SSD implementa de forma diferente o comando TRIM. Alguns implementam o que é chamado **DRAT** - *Deterministic Read After Trim* (Leitura Determinística Após o Trim) ou **DZAT** - *Deterministic Zeroes After Trim* - (”Zeros” Determinísticos Após o Trim) [Gubanov and Afonin 2014]. No primeiro, após o envio do comando TRIM pelo SO à controladora seguido de um comando de leitura desses blocos, o SSD retorna blocos contendo valores fixos. Já no segundo, esses valores são 0s. Outra implementação possível do comando retorna o conteúdo dos blocos até que as rotinas internas de coleta de lixo entrem em ação.

Cada fabricante define como a controladora implementa o comando TRIM, pois não há um padrão. Dessa forma, no caso de um certo par SO e Sistema de Arquivos que

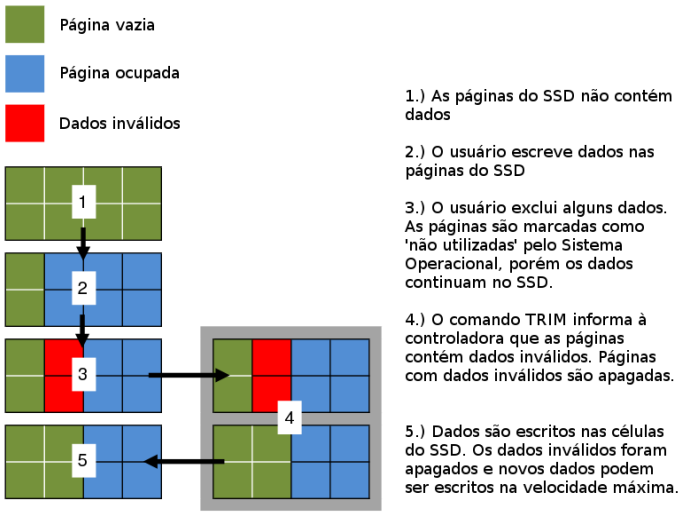


Figura 3. Funcionamento do comando TRIM
Adaptada de [Corsair 2010]

suporte o comando TRIM, o processo de exclusão e posterior leitura dos blocos lógicos ”podados” poderá apresentar diferentes comportamentos quando comparado a outro par SO e Sistema de Arquivos. Em alguns sistemas operacionais, o TRIM é enviado automaticamente. Em outros, é dependente de configuração, podendo ser necessária uma demanda explícita.

3.1. Suporte ao TRIM nos Sistemas Operacionais

Entre os sistemas operacionais que oferecem suporte ao comando TRIM encontram-se: Windows 7 e sucessores [Microsoft 2008] , Mac OS X (a partir da versão 10.6.8) [Slivka 2011] e o Linux [KernelNewbies 2008]. O suporte não é pleno, pois depende também do Sistema de Arquivos e poderá depender de habilitação ou configuração prévia.

4. Trabalhos relacionados

Antonellis [Antonellis 2008] realiza um experimento de exclusão de imagens e de um arquivo de texto, sob o ambiente *Windows* com o Sistema de Arquivos **NTFS** obtendo apenas arquivos com 0s. O autor não menciona a presença ou a influência do comando TRIM, assim como também não menciona a marca e o modelo do SSD.

Bell e Boddington [Bell and Boddington 2010] realizam experimentos de recuperabilidade no sistema *Windows XP* que não possui suporte ao comando TRIM, obtendo resultados ínfimos para a recuperação dos arquivos.

Gebremaryam [Gebremaryam 2011] realiza experimentos de recuperabilidade no sistema *Windows 7* na presença do TRIM, não obtendo sucesso. Já os experimentos na ausência do TRIM foram bem sucedidos.

King e Vidas [King and Vidas 2011] realizam experimentos nos sistemas *Windows* e *Linux* para uma vasta gama de SSDs obtendo resultados variados.

Bonetti et al. [Bonetti et al. 2011] realizam experimentos nos sistemas *Windows*

e *Linux*, na presença do TRIM, não obtendo sucesso. Já os experimentos na ausência do TRIM foram bem sucedidos.

Nisbet et al. [Nisbet et al. 2013] realizam experimentos nos sistemas *Windows*, *Linux* e no *Mac OS X* com resultados variados. Quanto ao uso do TRIM como uma medida anti-forense, os autores concluem que o TRIM por si só não é uma medida adequada para a sanitização de um SSD, relatando que em alguns casos, 185 a 214 MB remanesceram em um disco após a instrução do comando.

Júnior e Ruy [Junior and Queiroz 2015a] em 2015 afirmam que foi possível recuperar arquivos excluídos em um SSD da marca ADATA, modelo Premier SP800 de 32GB que possui suporte ao TRIM. Os autores não entram em detalhes sobre até que ponto se dá essa recuperabilidade nem sobre o procedimento utilizado.

5. Estudo de caso

A partir de um SSD da marca Samsung modelo 850 EVO, foi realizado um estudo de caso. O SSD em questão encontrava-se em estado seminovo, onde foi sujeito ao uso de um usuário comum, utilizando certas aplicações como navegador, *softwares* de edição de texto, terminal, visualizadores de mídia (aúdio, texto) etc. configurando-se, em maioria, uso de baixo impacto para o sistema de arquivos. A metodologia utilizada é a *black-box*. Dessa forma, não está no escopo deste trabalho avaliar a eficácia ou a usabilidade de ferramentas específicas. Também há que se mencionar que provavelmente há cache no dispositivo.

5.1. Cenários

Foram realizados cinco experimentos, com base em cinco cenários, com o intuito de comprovar ou contradizer os resultados atuais e realizar uma análise comparativa. Os cenários foram divididos em dois grupos: sistema *Linux* com Sistema de Arquivos **ext4** e sistema *Windows* com o Sistema de Arquivos **NTFS**.

No sistema *Linux*:

1. Exclusão de arquivos, sem a atuação do comando TRIM em qualquer forma e posterior tentativa de recuperação, com o objetivo de verificar a atuação da coleta de lixo;
2. Exclusão de arquivos, seguida do envio do comando TRIM manual após um tempo t e posterior tentativa de recuperação com o objetivo de verificar o efeito do comando TRIM e atuação da coleta de lixo.

Já no sistema *Windows*:

3. Exclusão de arquivos, sem a atuação do comando TRIM em qualquer forma e posterior tentativa de recuperação, com o objetivo de verificar a atuação da coleta de lixo;
4. Exclusão de arquivos, com o comando TRIM automático habilitado e posterior tentativa de recuperação, com o objetivo de verificar o efeito do comando TRIM quando enviado de forma autônoma pelo SO, além de verificar a atuação da coleta de lixo;
5. Exclusão de arquivos, seguida do envio do comando TRIM manual após um tempo t e posterior tentativa de recuperação com o objetivo de verificar o efeito do comando TRIM e atuação da coleta de lixo. O TRIM automático está desabilitado durante esse experimento.

5.2. Arquivos

Foram selecionados oito arquivos de diferentes formatos conforme a Tabela 1:

Tabela 1. Arquivos selecionados

Nro	Nome	Tamanho	Tipo
1	(Aya Higuchi) Chopin - Waltz in A minor, B. 150.mp3	2.1 MB	Arquivo de música (.mp3)
2	campo01.mp4	35.6 MB	Vídeo (.mp4)
3	folhas01.png	28.8 kB	Imagem (.png)
4	milho01.jpg	933.5 kB	Imagem (.jpg)
5	terra01.gif	927.0 kB	Imagem (.gif)
6	texto01.txt	61 B	Texto (.txt)
7	texto02.rtf	120.0 kB	Texto (.rtf)
8	relatorio01.pdf	2.5 MB	Documento (.pdf)

Para cada caso de teste no sistema *Linux* foi criada uma imagem de partição contendo apenas 0s e esta foi montada. No caso do sistema *Windows* foi criada um volume para cada caso de teste, também formatado. Os arquivos selecionados foram copiados para os volumes e logo após excluídos. Após o decorrer de certo período de tempo após a exclusão, a imagem de disco no sistema *Linux* foi desmontada, e no sistema *Windows* foi criada a partir do volume. Sob esta imagem foram aplicadas algumas das ferramentas usuais de recuperação de dados constantes na Seção 5.5.

5.3. Criação das imagens

Para a criação das imagens de disco no sistema *Linux* foi utilizado o utilitário *dd* versão 8.23. As imagens possuiam 250000 blocos de tamanho 4096 *bytes* totalizando 1Gb de tamanho total e foram particionadas com o utilitário *fdisk* versão 2.25.2 e formatadas com o Sistema de Arquivos **ext4** utilizando o *mkfs.ext4*.

Os volumes criados no sistema *Windows* possuiam o mesmo formato de número de blocos e tamanho dos blocos, logo, mesmo tamanho de 1Gb. O volume foi formatado com o Sistema de Arquivos **NTFS**. As imagens dos volumes foram criadas utilizando o programa *ProDiscover Basic* versão 7.0.

5.4. Casos de teste

Para cada caso de teste foram realizados os seguintes procedimentos:

1. Criação da partição formatada contendo apenas 0s;
2. Cópia dos arquivos de teste para a partição;
3. Exclusão ”permanente”dos arquivos;
4. Envio do comando TRIM manualmente (Cenários 2 e 5);
5. Aguardam-se t segundos e a imagem de disco é criada (*Windows*) ou desmontada (*Linux*);
6. Tentativa de recuperação dos arquivos.

5.5. Recuperação

Sob as imagens forenses foram aplicadas as seguintes ferramentas:

No *Linux*:

- *extundelete* versão 0.2.4 com a biblioteca libext2fs versão 1.42.12 por N E Case;
- *foremost* versão 1.5.7 por Jesse Kornblum, Kris Kendall, e Nick Mikus;
- *magicrescue* versão 1.1.9 por Jonas Jensen;
- *scalpel* versão 1.60 por Golden G. Richard III;
- *autopsy* versão 2.24 por Brian Carrier.

No *Windows*:

- *The Sleuth Kit* versão 4.2.0 por Brian Carrier;
- *FTK Imager* versão 3.4.2 por ACCESS DATA.

Para cada caso de teste, o objetivo era avaliar a viabilidade em recuperar os arquivos excluídos. Dessa forma, se uma ferramenta recuperou todos os arquivos de forma completa, não foram utilizadas as demais. Caso contrário, foram utilizadas todas as ferramentas sucessivamente. No escopo do trabalho não coube avaliar a eficácia das ferramentas na recuperação dos arquivos. Foi utilizado um comparador hexadecimal (*diff*) nos casos de recuperação parcial.

6. Resultados

No Cenário 1, no sistema *Linux* com o TRIM desabilitado, foi possível a recuperação total dos arquivos excluídos.

No Cenário 2, com o TRIM manual, foi possível recuperar os arquivos até um tempo t de 5s, que é a janela de oportunidade para a recuperação forense.

Já nos experimentos no sistema *Windows*, no Cenário 3, com o TRIM desabilitado, obteve-se a recuperação total dos arquivos, exceto um caso espúrio onde $t = 5$. Nesses caso, um dos arquivos encontrava-se com seu conteúdo ”zerado”e o outro com conteúdo parcialmente ”zerado”. Levanta-se a hipótese de que o dado espúrio represente uma execução aparentemente aleatória das rotinas de coleta de lixo ou um erro na formação da imagem de disco ou algum outro evento não previsto, como a presença prévia do comando TRIM na controladora.

No Cenário 4, na presença do TRIM automático (agendado), obteve-se uma janela de oportunidade de 3s. Para $4s \leq t \leq 15s$, recuperação foi parcial. Para $t \geq 16s$, obteve-se a recuperação do arquivo de texto (arquivo número 6 na Tabela 1).

Por fim, no Cenário 5, na atuação do TRIM manual, verifcou-se uma janela de oportunidade menor que 1s. Para $t \leq 20s$, a recuperação foi parcial. Para $t \geq 20s$, obteve-se a recuperação do arquivo de texto (arquivo número 6 na Tabela 1).

7. Comparação com outros trabalhos

A Tabela 2 contém um resumo da análise comparativa com os resultados de outros trabalhos.

8. Conclusão e trabalhos futuros

Foram estudadas as características e mecanismos internos dos SSDs que podem inviabilizar ou dificultar a prática usual de recuperação forense e de forense computacional em dois Sistemas Operacionais, *Windows* e *Linux*.

Tabela 2. Comparação com outros trabalhos

Autores	Resultados obtidos pelos autores	Comparação com o trabalho realizado
Bell & Boddington	Experimentos de recuperabilidade realizados no sistema *Windows XP* sem o TRIM, sem sucesso	Resultados obtidos pela dupla refutados pelos resultados dos experimentos dos Cenários 1 e 3
Gebremaryam	Experimentos no sistema *Windows 7* na presença (sem sucesso) e na ausência do TRIM (bem sucedidos)	Resultados se alinharam
King & Vidas	Experimentos nos sistemas *Windows* e *Linux*, vasta gama de SSDs com resultados variados	Confirmados para SSDs de algumas marcas e refutados por outros de outras
Bonetti et al.	Experimentos nos sistemas *Windows* e *Linux*, na presença do TRIM (sem sucesso) e na ausência do TRIM (bem sucedidos)	Resultados se alinharam, exceto para um modelo de SSD
Nisbet et al.	Experimentos nos sistemas *Windows*, *Linux* e no *Mac OS X* com resultados variados	Não coube comparação ($1hr \leq t \leq 5hrs$)

Os resultados foram variados, sendo que o fator determinante em ambos os sistemas é o comando TRIM. No sistema *Linux* a atuação do TRIM mostrou-se agressiva, inviablizando a recuperação dos arquivos após 5s. No sistema *Windows*, tanto a execução automática quanto a manual do comando TRIM apresentaram resultados negativos para a recuperação forense, à exceção de um arquivo de texto de tamanho na faixa dos *bytes*.

Em trabalhos futuros poderá ser estudada como se dá a interação do SO com a controladora do SSD nos instantes que se seguem após o envio do comando. Dessa forma poderão ser estudadas formas de impedir o envio do comando, retirá-lo da fila de comandos, impedir a sua execução, anular ou amenizar seus efeitos.

Trabalhos futuros também poderão ser direcionados no estudo das diferenças entre as implementações do TRIM em cada sistema bem como o estudo da granularidade dos arquivos e seus efeitos na recuperação forense, no sistema *Windows* com Sistema de Arquivos **NTFS**, dado que nesse sistema foi possível recuperar, em todos os testes, um arquivo com tamanho na faixa dos *bytes*.

Referências

Antonellis, C. J. (2008). *Solid State Disks and Computer Forensics* issa journal, pgs 36-38.

Bell, G. B. and Boddington, R. (2010). *Solid state drives: The beginning of the end for current practice in digital forensic recovery? Journal of Digital Forensics, Security and Law, 5(3):1-20.*

Bonetti, G., Viglione, M., Frossi, A., Maggi, F., and Zanero, S. (2011). *A Comprehensive Black-box Methodology for Testing the Forensic Characteristics of Solid-state Drives* journal of computer virology and hacking techniques.

Chen, F., Koufaty, D. A., and Zhang, X. (2009). *Understanding intrinsic characteristics and system implications of flash memory based solid state drives In ACM SIGMETRICS Performance Evaluation Review, volume 37, pages 181-192. ACM.*

Corsair, I. (2010). *How to Check That TRIM is active* `http://www.corsair.com/media/cms/manual/How_To_Check_That_TRIM_is_Active.pdf`.

Gebremaryam, F. Y. (2011). *Solid State Drive (SSD) Digital Forensics Construction* master of science in computing system engineering, politecnico di milano.

Gubanov, Y. and Afonin, O. (2014). *Recovering Evidence from SSD Drives in 2014: Understanding TRIM, Garbage Collection and Exclusions*.

Junior, M. A. C. C. and Queiroz, R. J. G. B. (2015a). *Apos um processo de formatacao, e possivel recuperar dados em um SSD?* universidade federal de pernambuco (ufpe).

Junior, M. A. C. C. and Queiroz, R. J. G. B. (2015b). *Forense em Solid State Drive: entendendo os mecanismos internos que podem inviabilizar a recuperacao de dados* universidade federal de pernambuco (ufpe).

KernelNewbies (2008). *Linux 2 6 28* `http://kernelnewbies.org/Linux_2_6_28\#head-a1a9591f48fe0cf8fde1f0d1f637c7ae54ad0bfa`.

King, C. and Vidas, T. (2011). *Empirical analysis of solid state disk data retention when used with contemporary operating systems* digital investigation journal.

Microsoft, C. (2008). *Windows 7 Enhancements for Solid-State Drives* `http://download.microsoft.com/download/F/A/7/FA70E919-8F82-4C4E-8D02-97DB3CF79AD5/COR-T558_Shu_Taiwan.pdf`.

Nazarian, H. and Dubois, S. (2013). *The drive for SSDs: What's holding back NAND flash?* `http://www.edn.com/design/systems-design/4424905/2/The-drive-for-SSDs--What-s-holding-back-NAND-flash--`.

Nisbet, A., Lawrence, S., and Ruff, M. (2013). *A forensic analysis and comparison of solid state drive data retention with trim enabled file systems* in proceedings of the 11th australian digital forensics conference. sri security research institute, edith cowan university, perth, western australia.

Olson, A. and Langlois, D. (2008). *Solid State Drives Data Reliability and Lifetime*.

Samsung, C. (2016). *SSD 850 EVO Overview* `http://www.samsung.com/global/business/semiconductor/minisite/SSD/global/html/ssd850evo/overview.html`.

Slivka, E. (2011). *Mac OS X 10.6.8 Brings TRIM Support for Apple SSDs, Graphics Improvements [Updated]* `http://www.macrumors.com/2011/06/27/mac-os-x-10-6-8-brings-trim-support-for-apple-ssds-graphics-improvements/`.

Um Estudo Acerca das Fraudes na Comunicação por Campo de Proximidade

Emilio R. R. Tubino e Juliano F. Kazienko

[1]Curso de Ciência da Computação - Universidade Federal do Pampa (UNIPAMPA)
Prédio A1 – CEP 97.546-550 – Alegrete – RS – Brasil

emiliotubino@alunos.unipampa.edu.br, kazienko@unipampa.edu.br

Abstract. *The Near Field Communication (NFC) has recently attracted the interest of the scientific community, industry, and users. It makes the NFC a promising communication technology. However, for its advance there are issues that need to be addressed. This work aims to study the frauds regarding to the NFC technology, specially ones related to the data modification, tag cloning, and data relay. Hence, we accomplish three study cases identifying vulnerabilities, attacks, and countermeasures. In addition, the proposed countermeasure in each study case is evaluated in order to demonstrate its efficiency.*

Resumo. *A Comunicação por Campo de Proximidade, do inglês, Near Field Communication (NFC) tem recentemente atraído o interesse da comunidade científica, da indústria e usuários, tornando-se uma tecnologia de comunicação promissora. Contudo, apesar de gradativa popularização, tal tecnologia ainda possui questões de segurança em aberto que devem ser tratadas. Este trabalho tem por objetivo estudar as fraudes no âmbito da tecnologia NFC, especialmente envolvendo a adulteração, a clonagem e a retransmissão. Para tanto, são realizados três estudos de caso nos quais se identificam vulnerabilidade, ataque e contramedida. Adicionalmente, a contramedida proposta através de cada estudo de caso é avaliada a fim de demonstrar sua efetividade.*

1. Introdução

A tecnologia de Identificação por Radiofrequência, do inglês, *Radio Frequency Identification* (RFID), realiza sua comunicação através de ondas de radiofrequência entre dispositivos. Essa tecnologia é comumente utilizada na identificação de itens, controle de ambientes e comunicação de dispositivos. Diversos padrões da tecnologia RFID são criados e adaptados dependendo das necessidades de cada aplicação. Esse é o caso da tecnologia de Comunicação por Campo de Proximidade, do inglês, *Near Field Communication* (NFC), que possui um campo de comunicação reduzido a uma distância máxima entre os dispositivos de aproximadamente 10 cm e opera em uma frequência de 13,56 MHz [Want 2011] [Coskun et al. 2013]. Suas aplicações consistem no controle de acesso em fechaduras eletrônicas, identificação de objetos através de etiquetas NFC, na área de pagamentos eletrônicos e bilhetagem de ônibus e metro, no Japão, Europa e Brasil [RFIDJournalBrasil 2015].

Apesar da tecnologia NFC já ser utilizada para diversas aplicações, ainda trata-se de uma tecnologia relativamente nova, possuindo vulnerabilidades que permitem ataques como os de adulteração de dados, clonagem e retransmissão [Chen et al. 2014]

[Madlmayr et al. 2008] [Hancke et al. 2009] [Roland et al. 2013] [Oh et al. 2015], necessitando assim, de aprimoramentos no âmbito da segurança [Chen et al. 2014] [Chattha 2014]. A necessidade de resolver as vulnerabilidades presentes nessa tecnologia torna-se ainda mais evidente por ser comumente aplicada, por exemplo, a área médica e de pagamentos eletrônicos, tendo sido essa última, alvo constante de fraudes.

Relacionado às vulnerabilidades citadas anteriormente, a tecnologia RFID já possui mecanismo de segurança que evitam e/ou mitigam os ataques citados anteriormente. Entretanto, muitos dos mecanismos existentes e aplicados na tecnologia RFID não são adequados à tecnologia NFC, mesmo sendo essa última um subconjunto da tecnologia RFID. Isso ocorre pela diferença entre protocolos e dispositivos presentes na tecnologia NFC, como o caso da comunicação par-a-par e dispositivos como os *smartphones* que estão presentes apenas na tecnologia NFC. Logo, os mecanismos de segurança existentes na tecnologia RFID requerem adaptação, ou, novos mecanismos devem ser desenvolvidos baseados nas características do NFC.

Neste trabalho, é realizado um estudo sobre fraudes existentes na tecnologia NFC. Desse modo, realiza-se, através de estudos de caso, a identificação de vulnerabilidades, ataques e contramedidas. Particularmente, trata-se através de tais estudos de caso os ataques de adulteração de dados, clonagem de etiquetas e retransmissão de dados. Adicionalmente, as contramedidas propostas são avaliadas a fim de demonstrar sua efetividade. Na Seção 2, são discutidos os trabalhos relacionados. A Seção 3 apresenta os estudos de caso desenvolvidos. Por fim, na Seção 4, são tecidas as considerações finais.

2. Trabalhos Relacionados

Os trabalhos apresentados pelos autores [Chen et al. 2014] e [Madlmayr et al. 2008] realizam um estudo acerca das fraudes aplicáveis na tecnologia NFC. Tais trabalhos apresentam o ataque de negação de serviço, do inglês, *Denial of Service* (DoS), em etiquetas NFC como um problema a ser tratado, sugerindo a necessidade um mecanismo de controle da função de leitura e escrita na etiqueta a fim de evitar que usuários não autorizados adulterem os dados armazenados nela. Tais trabalhos citam o ataque de adulteração de dados como um problema em aberto.

O trabalho apresentado pelos autores [Hameed et al. 2014] apresenta um mecanismo de segurança contra conteúdos maliciosos armazenados em etiquetas NFC, baseado em lista branca (*white list*) e lista negra (*black list*). Através dessas listas, o mecanismo intermediário confronta a *URL* lida, e verifica a confiabilidade da mesma. Neste tipo de ataque, uma *URL* pode ser inserida em um etiqueta para que um *smartphone* ao ler o conteúdo, seja direcionado para um *link* malicioso, como *link's* para *download* de aplicativos prejudiciais ou sites de empresas concorrentes a proprietária da etiqueta. Esse mecanismo apresenta vulnerabilidades como *URL's* não cadastradas na lista, e a necessidade do *smartphone* estar sempre atualizado com as listas, havendo assim o risco do conteúdo armazenado na etiqueta ser acessada pelo usuário pelo fato de não poder verificar sua confiabilidade.

No trabalho apresentado por [Spruit and Wester 2013], os autores propõem um mecanismo de segurança contra o ataque da clonagem no âmbito da tecnologia RFID. A contramedida proposta consiste no método *Public Key Re-Encryption*, que está baseado

no envio de dados cifrados pela etiqueta, impedindo que o atacante consiga clonar seu serial. Entretanto, esse método não permite sua aplicação em etiquetas capazes somente de realizar operações de leitura e escrita de dados, devido à incapacidade de realizar computações como a cifragem de dados.

O trabalho apresentado por [Lehtonen et al. 2009] aborda um mecanismo de detecção do ataque da clonagem em etiquetas que permitem apenas operações de leitura e escrita. Esse mecanismo é baseado em um número randômico compartilhado entre o terminal e a etiqueta. Com a comparação do valor randômico da etiqueta e terminal, verifica-se a sincronia entre os valores para a detecção da adulteração. A detecção da clonagem só ocorre quando duas etiquetas com o mesmo serial tentam acessar o sistema, assim a primeira a acessar o sistema dessincroniza o valor de autenticação randômico armazenado no terminal em relação a outra etiqueta.

Entretanto, apesar de o trabalho de [Lehtonen et al. 2009] citar ataques DoS, o mecanismo proposto pelos autores não resolve esta vulnerabilidade ocasionada pela clonagem das etiquetas. Nesse ataque, o objetivo principal é inviabilizar o acesso ao serviço a um usuário que tente autenticar-se perante um terminal utilizando a etiqueta legítima. Por utilizar números randômicos, este mecanismo também não possibilita a obtenção do histórico do atacante, não sendo possível verificar quais foram as utilizações da etiqueta clonada (ilegítima) e quais suas ações.

Outro ataque ao qual a tecnologia NFC está vulnerável consiste no ataque de retransmissão, ou do inglês, *Relay Attack*. Os autores [Hancke et al. 2009], além de estudar o ataque de retransmissão e realizarem uma crítica às contramedidas existentes a este ataque, também citam a existência de um mecanismo de tempo na comunicação NFC especificada pela ISO [14443 ISO 2008] para evitar o ataque de retransmissão. De acordo com os autores, seu tempo de tolerância muito prolongado tende a deixar uma grande margem para que um atacante realize o ataque de retransmissão.

O trabalho apresentado pelos autores [Ranganathan et al. 2015] propõem um mecanismo de segurança contra o ataque de retransmissão baseado na distância entre os dispositivos. Esse mecanismo utiliza o sinal de comunicação para detectar a distância entre os dispositivos, verificando a intensidade e ruído do sinal recebido através do uso de um osciloscópio. Com a comparação do sinal recebido e o sinal legítimo conhecido, é possível identificar a diferença entre os ruídos e intensidade do sinal recebido caso ele tenha sido gerado de um ataque de retransmissão. Esse mecanismo tem um custo mais elevado pois necessitar de *hardwares* mais robustos, não sendo viável utilizá-lo para o uso pessoal da tecnologia NFC, como em dispositivos *smartphones*.

Pelo fato da tecnologia NFC e a *Wi-fi* utilizarem uma frequência de comunicação diferente uma da outra, o trabalho de [Oh et al. 2015] propõe um mecanismo de segurança para evitar o ataque da retransmissão através da interferência do canal de comunicação, do inglês, *Jamming*, utilizado pela tecnologia *Wi-Fi*, não interferindo na comunicação NFC. Entretanto, o mecanismo apresentando em [Oh et al. 2015] se mostra ineficiente caso a tecnologia de comunicação utilizada para o ataque for diferente da *Wi-Fi* ou utilize um meio cabeado para a retransmissão dos dados. Outra limitação dessa contramedida é a interferência em qualquer comunicação *Wi-Fi*. Assim, todos os dispositivos que estiverem utilizando tal tecnologia, sejam legítimos ou atacantes, serão afetados por tal interferência.

3. Estudos de Caso

Esta seção apresenta três estudos de caso relacionados aos ataques de adulteração de dados, clonagem de etiquetas e retransmissão de dados, apresentando propostas de contramedidas avaliadas ao final de cada estudo. Para fins de experimentação e avaliação, foram utilizados: um *notebook Sony vaio* modelo *svf15213cbw* com NFC integrado, um *notebook LG* modelo *A530* e um leitor NFC modelo *ACR 122U* com conexão *USB*. Também foram utilizados um *Smartphone Samsung E7* e *Smartphone Sony Xperia M*, ambos com NFC integrado. Por fim foram utilizadas etiquetas NFC do modelo *MIFARE DESfireEV1* tipo 4, com a capacidade de armazenamento de 4.094 *Bytes*. Entretanto, qualquer tipo de etiqueta NFC poderia ser utilizada, principalmente modelos mais baratos com processamento limitado, uma vez que as propostas foram projetadas para tais etiquetas.

3.1. Estudo de Caso 1 (EC1): Detecção a adulteração de etiquetas NFC

As etiquetas NFC mais comumente utilizadas são as capazes de realizar apenas operações de leitura e escrita. Tais etiquetas são mais vulneráveis a ataques de adulteração por não possuir um controle de escrita que evite que dispositivos não autorizados realizem a adulteração dos dados armazenados. O ataque de adulteração utiliza uma etiqueta legítima, ou seja, uma etiqueta pertencente ao ambiente para causar prejuízos ao sistema [Chen et al. 2014]. Através desse ataque, um atacante pode modificar os dados contidos em uma etiqueta legítima, alterando preços de uma mercadoria e suas características, trazendo assim, vantagens ao atacante e prejuízos ao sistema ao qual essa etiqueta pertence. É possível também que um atacante modifique uma etiqueta legítima e adicione conteúdos maliciosos ou de empresas concorrentes, dependendo da aplicação.

Portanto, a contramedida proposta aqui tem por objetivo detectar ataques de adulteração de dados em etiquetas que possibilitam apenas operações de leitura e escrita. Particularmente, tal tipo de etiqueta é abordado neste trabalho pelo fato de possuir menor custo monetário e ser comumente encontradas na área comercial. Para fins de validação do mecanismo de contramedida desenvolvido, é abordado aqui um cenário de uma loja de produtos eletrônicos. Nesse sistema, todos os produtos possuem uma etiqueta NFC com seus dados técnicos e preço gravados nela, sendo cada etiqueta previamente cadastrada através de um leitor NFC, com seus dados, como o número serial da etiqueta armazenados em um banco de dados (BD). Para consultar informações e preços dos produtos do estabelecimento, os usuários do sistema (clientes) utilizam seu *Smartphone* com NFC habilitado, que deve estar atualizado com o software da loja previamente disponibilizado.

O esquema de autenticação é baseado na abordagem de obtenção de assinatura digital *Rivest, Shamir e Adelman* (RSA) [Stallings 2010], através de chaves assimétricas. Nesse sistema o terminal recebe os dados que serão armazenados na etiqueta, realiza um resumo criptográfico (*hash*) da mensagem M, $hash(M)$, e através da chave privada, E_{KR}, à cifra a fim de computar a assinatura digital da mensagem $\sigma = E_{KR}(hash(M))$, sendo que M consiste nos dados armazenados na etiqueta como preço e dados técnicos de um produto. Assim, são gravados na etiqueta M e a assinatura σ. O dispositivo leitor ao comunicar-se com a etiqueta, recebe a mensagem e a assinatura armazenados nela, que através da chave pública do terminal, E_{KP}, previamente distribuída entre os usuários do sistema, decifra o *hash* contido na etiqueta, $D_{KP}(\sigma)$, e gera um novo *hash* da mensagem recebida, $hash'(M)$. Tais resumos são comparados: $hash'(M) = D_{KP}(\sigma)$, se forem

iguais, a mensagem é aceita como integra e autêntica, finalizando assim o processo de autenticação.

3.1.1. Avaliação

Preliminarmente, o protótipo construído permitiu verificar que o sistema é funcional. Para testar o sistema, 5 etiquetas foram utilizadas, das quais 3 foram adulteradas. Nos três casos, a detecção da adulteração foi efetuada. O tempo médio de execução do protocolo de autenticação dos dados na etiqueta é de 3 ms. Tal tempo de execução é baixo comparado a um sistema sem autenticação, que demandou 2 ms para a leitura da etiqueta.

É importante ressaltar que o mecanismo proposto aqui não evita a adulteração dos dados, entretanto, ainda permite a detecção do ataque da adulteração, protegendo os usuários das etiquetas contra tal ataque. Nesse mecanismo é possível também detectar a inserção de etiquetas falsas (ilegítimas), pelo fato de que os atacantes não possuem a chave privada para a assinatura dos dados armazenados nas etiquetas.

3.2. Estudo de Caso 2 (EC2): Detecção de etiquetas clonadas

Um dos grandes desafios da tecnologia NFC consiste na detecção e invalidação de etiquetas clonadas que possibilitam apenas operações de leitura e escrita de dados. Nessas etiquetas, toda a informação armazenada pode ser lida por qualquer dispositivo [Chen et al. 2014] [Lehtonen et al. 2009], sendo essa a vulnerabilidade que possibilita a um atacante a leitura e cópia de todo o conteúdo para uma etiqueta ilegítima, requerendo tratamento apropriado.

O ataque da clonagem consiste na cópia dos dados de uma etiqueta (legítima) e posterior gravação desses dados em outras etiquetas (ilegítima). Com isso, o atacante, possuidor de uma etiqueta clonada, pode obter vantagem indevida identificando-se como terceiro, acessando ambientes, etc. [Chen et al. 2014]. Em um ambiente de controle de acesso que utiliza uma etiqueta NFC como "chave" de abertura de uma fechadura NFC, é possível que um atacante clone a etiqueta, e com isso, possa acessar um ambiente restrito. Considerando o acesso a um sistema de informação mediante a apresentação de uma etiqueta, o acesso indevido não será percebido, pois os dados da etiqueta ilegítima serão os mesmos da legítima.

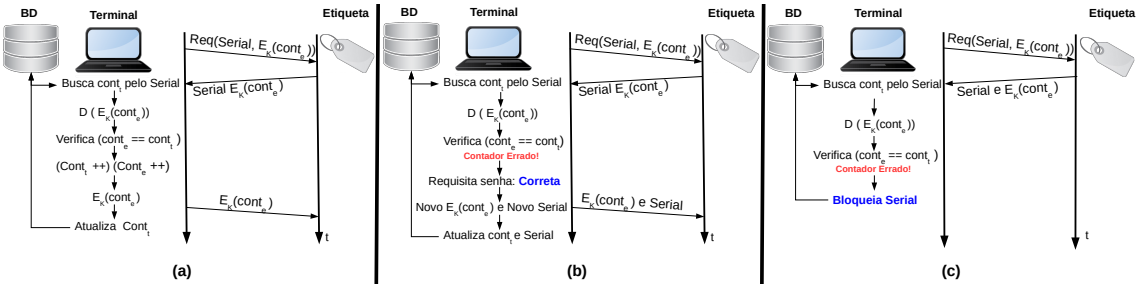
Assim, é proposto aqui como contramedida um mecanismo de detecção e invalidação de etiquetas clonadas (ilegítimas) que permitem apenas operações de leitura e escrita com o uso terminais NFC responsáveis pela detecção do ataque. Esses terminais estão conectados a um BD responsável pelo armazenamento das informações contidas nas etiquetas, como seu número serial, dados do usuário e um contador de leituras $cont_e$. Tal contador é incrementado a cada autenticação e atualizado no BD ($cont_e$) e armazenado na etiqueta de modo cifrado, $E_K(cont_e)$, utilizando uma chave simétrica K. Tal cifragem é necessária para que um atacante não tenha conhecimento do valor de $cont_e$.

Um terminal, ao realizar a leitura dos dados da etiqueta, decifra o contador $cont_e$ com a chave K da seguinte forma $D_K(E_K(cont_e))$. Em seguida, através da busca pelo serial da etiqueta no BD, o terminal resgata o valor de $cont_t$, e confere se $cont_e = cont_t$. Se tais contadores forem iguais, o terminal incrementa os contadores, $cont_t = cont_t + 1$

e $cont_e = cont_e + 1$. Em seguida, cifra $cont_e$ através da operação $E_K(cont_e)$ gravando o novo valor de $cont_e$ na etiqueta e atualizado o valor de $cont_t$ no banco de dados, conforme ilustrado na Figura 1(a). A detecção da clonagem acontece quando o usuário da etiqueta legítima utiliza o terminal novamente, uma vez que o terminal detecta que $cont_t \neq cont_e$.

Conforme o ambiente no qual o mecanismo for utilizado, duas versões do mecanismo são propostas. A Figura 1(b) ilustra a Versão 1, que envolve o uso de senhas. Nessa versão, um terminal ao identificar uma etiqueta como clonada, ou seja $cont_e \neq cont_t$, requisita uma senha previamente cadastrada pelo usuário legítimo da etiqueta. Tal senha é utilizada para fins de autenticação e fica armazenada no BD. Ao inserir sua senha, o usuário terá o serial de sua etiqueta alterado e atualizado no BD, fazendo com que a etiqueta clonada (ilegítima) não seja mais válida em nenhum terminal. É importante destacar que o uso de uma senha permite que o usuário não sofra uma negação de serviço, ou seja, DoS. Desse modo, o usuário autêntico poderá continuar utilizando o sistema. Nesse caso, há a possibilidade do usuário utilizar a etiqueta legítima antes do atacante. Assim, o atacante ao utilizar a etiqueta ilegítima não possuirá o contador correto. Esta solução apropriada para ambientes que possibilitam o uso de terminais com senha, como, por exemplo, estabelecimentos comerciais, mercados, restaurantes e cafeterias.

Figura 1. (a) Mecanismo de detecção de etiquetas clonadas, (b) versão com senha e (c) versão sem senha.



Já para ambientes que não possibilitam o uso de senhas, como no controle de mercadorias ou mesmo o controle de gado via etiqueta NFC, a Versão 2 mostrada na Figura 1(c) é mais apropriada. Nesta versão, uma etiqueta ao ser identificada como clonada, ou seja, $cont_e \neq cont_t$, terá seu serial bloqueado no BD. Assim, tanto a etiqueta legítima quanto a ilegítima são invalidadas, sendo necessário cadastrar novamente os dados em uma etiqueta para a mercadoria ou usuário legítimo. Como o serviço de identificação é negado, esse método está sujeito ao ataque de DoS. Vale lembrar que o foco do método proposto é retirar a etiqueta clonada de circulação para evitar maiores prejuízos ao usuário.

3.2.1. Avaliação

Para fins experimentais, utilizou-se o *Data Encryption Standard* (DES) como algoritmo de cifragem. Entretanto, qualquer cifra simétrica poderia ser utilizada. O protótipo construído permitiu verificar que o sistema é funcional. O tempo médio para a execução do mecanismo na Versão 1, com uso de senha, é de 7 ms, desconsiderando

o tempo que o usuário leva para digitar a senha. Já o tempo médio para execução do mecanismo na Versão 2, sem senha, é de 5 ms. A avaliação experimental revelou a funcionalidade do mecanismo proposto e uma baixa sobrecarga de tempo em relação a não aplicação do mecanismo, que possui um tempo médio de 2 ms para comunicação. Esse mecanismo mostra-se mas eficiente visto que o tempo demandado para autenticação em [Lehtonen et al. 2009] de 864 ms. Adicionalmente, o mecanismo proposto aqui abrange uma maior variedade de cenários, sendo capaz de evitar prejuízos causados pelo ataque DoS na Versão 1 do mecanismo, que faz uso de senha. Entretanto, a Versão 2, idealizada para cenários que não possibilitam o uso de senha, ainda admite esse ataque.

É importante ressaltar que, embora o método permita uma interação do atacante com o terminal, ele se mostrou eficaz em detectar a clonagem e invalidar uma etiqueta. Comparando à detecção da clonagem de cartões de crédito, que em geral fica por conta do próprio usuário, o mecanismo proposto aqui detecta a clonagem da etiqueta na sua próxima utilização posterior a da etiqueta clonada. Nesse mecanismo é possível identificar os danos causados pelo atacante utilizando o histórico obtido através da comparação entre o contador da etiqueta legitima e armazenado no BD. Adicionalmente, o método proposto aqui é simples uma vez que requer a troca de poucas mensagens, além de o terminal usar criptografia simétrica, de menor custo computacional quando comparado à criptografia assimétrica.

3.3. Estudo de Caso 3 (EC 3): Detectando o ataque de retransmissão de dados

Os dados transportados entre dois dispositivos são passíveis de interceptação por um dispositivo operando na mesma frequência da transmissão e dentro da área de cobertura, sendo essa uma vulnerabilidade que implica na possibilidade de que os dados ao serem recebidos por um dispositivo atacante, possam ser retransmitidos para os demais dispositivos que não fazem parte de tal comunicação, mesmo na presença de mecanismos de autenticação mútua entre os dispositivos legítimos. Na prática, a comunicação entre os dispositivos só ocorrerá caso esta seja a intenção do usuário, dado pela sua curta distância de comunicação. Com essa forma de ataque, um atacante pode forçar a comunicação entre dois dispositivos que estão distantes um do outro para realizar comunicações indesejadas, aproveitando-se da distração do usuário legítimo.

O ataque de retransmissão, do inglês, *Relay Attack*, pode ser utilizado para fraudar uma transmissão NFC, criando uma "ponte" de comunicação entre dois dispositivos camuflando a distância entre eles. Nesse caso, um dispositivo atacante A comunica-se com um terminal via NFC, que transmite sua comunicação via *Bluetooth* para o dispositivo atacante B que, por fim, retransmite a mensagem para o dispositivo legítimo (*smartphone*) via NFC novamente. Assim os dados enviados pelo terminal são exatamente os mesmos recebidos pelo dispositivo legítimo com o uso do ataque de retransmissão. Desta forma um atacante pode realizar a comunicação entre os dispositivos a uma distância maior, fraudando a presença de um dispositivo em um ambiente.

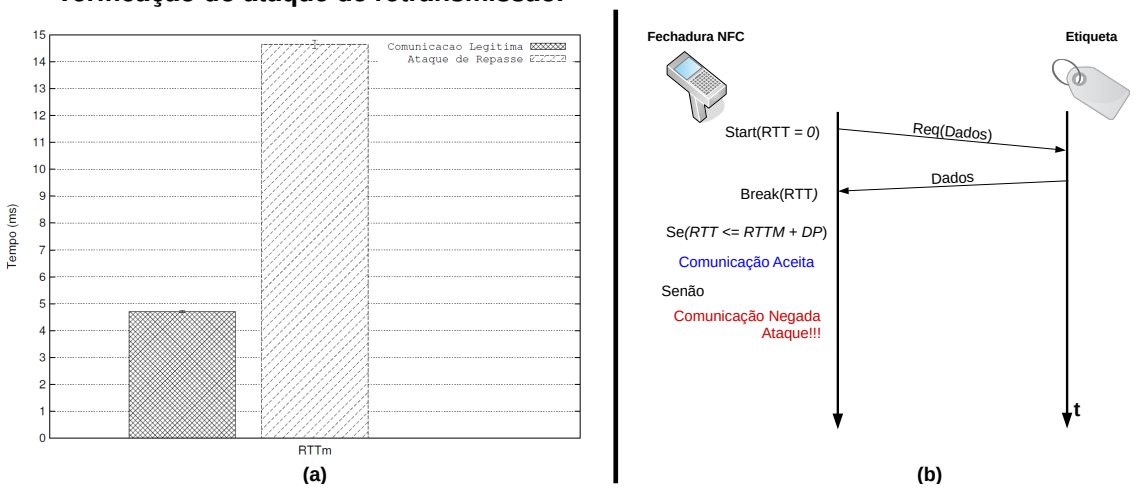
A contramedida proposta aqui tem por objetivo detectar o ataque de retransmissão de dados. Para isso, baseia-se em um ambiente de acesso controlado por uma fechadura NFC. Nesse mecanismo um terminal NFC é responsável pela autenticação dos dispositivos e etiquetas NFC, que serão as chaves de acesso. Esse mecanismo de segurança baseia-se na comparação de tempo de resposta de uma comunicação para a detecção do ataque

de retransmissão. Para isso, o terminal requisita os dados de autenticação ao dispositivo, disparando um contador de tempo de resposta, do inglês, *Round Trip Time* (RTT), que é automaticamente encerrado após o recebimento da resposta vinda do dispositivo. Tal tempo de comunicação médio da etiqueta, $RTTm$, é conhecido pelo terminal. Por fim, o sistema compara o valor resultante do tempo de resposta, RTT, com o tempo médio de comunicação, $RTTm$, como apresentado na Figura 2(b). Caso o RTT esteja dentro do Desvio Padrão (DP) aceitável, esta comunicação é dada como livre de retransmissão. Caso o valor seja superior ao DP aceitável esta comunicação é dada como resultante do ataque de retransmissão.

3.3.1. Avaliação

Para avaliar o mecanismo de segurança contra o ataque de retransmissão, implementou-se um sistema de comunicação entre um terminal e uma etiqueta NFC baseado em um cenário de autenticação de etiquetas para abertura de fechaduras eletrônicas. Posteriormente, implementou-se o ataque de retransmissão semelhante ao apresentado pelos autores [Cavdar and Tomur 2015] e [Wang et al. 2012], utilizando dois *Smartphones* atacante, A e B, para realizarem a retransmissão dos dados via *Bluetooth*. Onde A comunica-se com o leitor autêntico e B comunica-se com a etiqueta autêntica. Assim os tempos médios, $RTTm$, da comunicação legítima entre leitor e etiqueta, e o tempo de ataque de retransmissão foram coletados a partir de 1000 testes de comunicação e apresentados na Figura 2(a). O tempo médio de comunicação do leitor NFC e da etiqueta de $4,70ms$ com um desvio padrão de $0,571ms$, e um tempo médio de ataque de $14,64ms$ com um desvio padrão de $2,081ms$. Essa figura tem por objetivo demonstrar que o tempo de comunicação pode ser utilizado para detectar o ataque de retransmissão. Apresenta-se um nível de confiança de 95%.

Figura 2. A Parte (a) ilustra a comparação do tempo entre comunicações legítimas e ataques de retransmissão. A Parte (b) apresenta o protocolo de verificação do ataque de retransmissão.



Com a intenção de testar o mecanismo de segurança de forma prática, realizou-se 1000 comunicações legítimas e 1000 ataques de retransmissão. Como resultado, em todos

os testes da comunicação legítima não houve a ocorrência de falsos positivos, ou seja, identificar uma comunicação legítima como ataque de retransmissão, sendo todas aceitas pelo sistema. Já nos testes de ataques de retransmissão houve 100% de identificação do ataque, onde todas as comunicações retransmitidas pelo atacante foram recusadas.

É importante ressaltar que o tempo médio de comunicação entre a etiqueta legítima e o leitor NFC é coletado para que o sistema tenha conhecimento do tempo necessário para que uma comunicação legítima ocorra. Desse modo, através do uso do desvio padrão dos tempos coletados, é possível encontrar um tempo de tolerância mais adequado para a comunicação legítima. Portanto, ao contrário do método já existente e criticado pelo autor [Hancke et al. 2009], esse mecanismo utiliza o tempo necessário para que uma comunicação não seja identificada como falso positivo, sem que permita a presença do ataque de retransmissão, ou seja, evitando também falsos negativos.

Por fim, os resultados experimentais apresentados mostram que esse mecanismo de segurança é eficaz contra esse ataque. É importante observar que o tempo de ataque até mesmo em seu melhor caso de $8,10ms$, continua sendo recusado. Sendo possível então, detectar e evitar um ataque de retransmissão que siga este modelo de ataque apresentado pelos autores [Cavdar and Tomur 2015] e [Wang et al. 2012], através do uso do tempo de comunicação entre dispositivos, conforme mostrado na Figura 2.

A união do ataque de clonagem e retransmissão poderia burlar o mecanismo de tempo. Nessa união, o ataque da clonagem seria utilizado para prover os dados mais rapidamente ao leitor, realizando a clonagem da etiqueta legítima dos dados recebidos através do ataque de retransmissão. Assim, o dispositivo ilegítimo, por possuir os dados da etiqueta legítima a pronta entrega, seria aceito por não possuir um tempo elevado de comunicação. Um mecanismo de contramedida foi desenvolvido para evitar que a união dos ataques burle a contramedida baseada em tempo. Para isso, é necessário realizar bloqueio de leitura das etiquetas, para assim forçar o atacante a realizar o ataque de retransmissão para a captura dos dados. Entretanto, até o presente momento não foi possível avaliar tal contramedida de forma prática.

4. Conclusão e Trabalhos Futuros

Através do estudo realizado sobre fraudes na tecnologia NFC, foi possível identificar a existência de vulnerabilidades e propor mecanismos de contramedida. Tais mecanismos foram desenvolvidos e avaliados através de estudos de caso, resultando na identificação da adulteração de dados, com uma baixa sobrecarga de tempo.

Foi possível também detectar o ataque da clonagem de forma mais eficiente que o apresentado pelo autor [Lehtonen et al. 2009], evitando o ataque de DoS e possibilitando a obtenção do histórico de uso do sistema pelo atacante através da etiqueta clonada. Entretanto, não foi possível evitar o ataque de clonagem, porém foram diminuídos os danos causados por esse ataque ao identificar o uso da etiqueta clonada após o uso da legítima.

Outro ataque que foi estudado consiste na retransmissão de dados. O estudo permitiu identificar e evitar tal ataque de uma forma mais adequada que o mecanismo de temporização existente na tecnologia NFC [14443 ISO 2008] e discutido por [Hancke et al. 2009]. Adicionalmente, o mecanismo de combate a retransmissão proposto no Estudo de Caso 3 não necessita de *hardwares* adicionais, diferentemente do que

se propõe no trabalho desenvolvido pelos autores [Ranganathan et al. 2015].

Para trabalhos futuros, objetiva-se evoluir as contramedidas desenvolvidas aqui, como no caso do EC 2, visando a identificação de uma etiqueta clonada de forma prévia, ou até mesmo evitando a clonagem de uma etiqueta de baixo custo. Também objetiva-se avaliar o mecanismo contra a união dos ataques de clonagem e retransmissão.

Agradecimentos

Trabalho Conclusão de Curso e Iniciação Científica executados com apoio financeiro da Universidade Federal do Pampa, recursos do Edital n. 08/2015 - Programa de Bolsas de Iniciação à Pesquisa (PBIP).

Referências

14443 ISO (2008). Identification cards - Contactless integrated circuit(s) cards - Proximity cards. Technical Report ISO/IEC JTC1/SC17 Número 1363, International Standard ISO/IEC.

Cavdar, D. and Tomur, E. (2015). A practical NFC relay attack on mobile devices using card emulation mode. In *38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015*.

Chattha, N. A. (2014). NFC Vulnerabilities and defense. In *2014 Conference on Information Assurance and Cyber Security (CIACS)*, pages 35–38.

Chen, C. H., Lin, I. C., and Yang, C. C. (2014). NFC Attacks Analysis and Survey. In *Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 458–462.

Coskun, V., Ozdenizci, B., and Ok, K. (2013). A Survey on Near Field Communication (NFC) Technology. *Wireless Personal Communication*, 71:2259–2294.

Hameed, S., Hameed, B., Hussain, S. A., and Khalid, W. (2014). Lightweight Security Middleware to Detect Malicious Content in NFC Tags or Smart Posters. In *IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 900–905. IEEE.

Hancke, G. P., Mayes, K., and Markantonakis, K. (2009). Confidence in smart token proximity: Relay attacks revisited. *Computers & Security*, 28(7):615–627.

Lehtonen, M., Ostojic, D., Ilic, A., and Michahelles, F. (2009). Securing RFID Systems by Detecting Tag Cloning. In Tokuda, H., Beigl, M., Friday, A., Brush, A. J. B., and Tobe, Y., editors, *7th International Conference on Pervasive Computing (Pervasive 2009)*, pages 291–308, Nara, Japan. Springer.

Madlmayr, G., Langer, J., Kantner, C., and Scharinger, J. (2008). NFC devices: Security and privacy. In *Third International Conference on Availability, Reliability and Security*, pages 642–647. IEEE.

Oh, S., Doo, T., Ko, T., Kwak, J., and Hong, M. (2015). Countermeasure of nfc relay attack with jamming. In *12th International Conference & Expo on Emerging Technologies for a Smarter World (CEWIT)*, pages 1–4. IEEE.

Ranganathan, A., Danev, B., and Capkun, S. (2015). Proximity verification for contactless access control and authentication systems. In *31st Annual Computer Security Applications Conference (ACSAC)*, pages 271–280. ACM.

RFIDJournalBrasil (2015). RFID Journal Brasil Website. Disponível em: http://brasil.rfidjournal.com/. Acessado em: 27/07/2015.

Roland, M., Langer, J., and Scharinger, J. (2013). Applying relay attacks to google wallet. In *5th International Workshop on Near Field Communication (NFC)*, pages 1–6. IEEE.

Spruit, M. and Wester, W. (2013). RFID Security and Privacy: Threats and Countermeasures. Technical report, Department of Information and Computing Sciences Utrecht University, Netherlands - Technical Report UU-CS- 2013-001.

Stallings, W. (2010). *Cryptography and Network Security: Principles and Practice*. Prentice Hall.

Wang, Z., Xu, Z., Xin, W., and Chen, Z. (2012). Implementation and Analysis of a Practical NFC Relay Attack Example. In *Second International Conference on Instrumentation, Measurement, Computer, Communication and Control (IMCCC), 2012*, pages 143–146.

Want, R. (2011). Near field communication. *IEEE Pervasive Computing*, 10(3):4–7.

Protótipo para exfiltração óptica de dados em máquinas fisicamente isoladas

Arthur Costa Lopes[1]*, **Diego F. Aranha**[1]

[1]Instituto de Computação (IC) – Universidade Estadual de Campinas (Unicamp)
arthur.lopes@students.ic.unicamp.com, dfaranha@ic.unicamp.br

Abstract. *We present a new approach for data exfiltration using a malicious storage device which subtly transmits data through blinking LEDs. This new approach may be used by an attacker trying to leak sensitive data stored in the device, such as credentials, cryptographic keys or a small classified document. An ideal application for this approach is when an attacker is capable of sneaking a malicious device inside a protected perimeter and has remote control over a camera inside such perimeter. We present several techniques for optimizing communication between transmitter and receiver, including application of error-correcting codes, achieving transmission rates up to 30 bits per second.*

Resumo. *Este trabalho apresenta uma nova abordagem para exfiltração de dados utilizando um dispositivo malicioso de armazenamento que transmite dados sutilmente utilizando LEDs infravermelho. A abordagem pode ser utilizada por um atacante para capturar dados armazenados no dispositivo, como credenciais ou chaves criptográficas. Um cenário ideal de aplicação ocorre quando o adversário é capaz de introduzir o dispositivo em um perímetro protegido e então coletar informação por meio de uma câmera sob seu controle remoto. Discutem-se técnicas para otimizar comunicação entre transmissor e receptor, incluindo a aplicação de códigos corretores de erro, obtendo taxas de transmissão de até 30 bits por segundo.*

1. Introdução

Com a ampla disponibilidade de sistemas de computação, organizações começaram a fazer uso extensivo de computadores para armazenar e trocar informações, muitas vezes de natureza extremamente sensível. A segurança da informação está se tornando cada vez mais essencial para proteger e impedir que esses dados vazem de diversas maneiras, como por exemplo, por atividades maliciosas de exfiltração. O conceito de exfiltração de dados consiste na extração de dados a partir de uma rede fechada depois de um *software* ou dispositivo malicioso se infiltrar em tal rede.

Nos últimos anos, as políticas de segurança começaram a considerar a ameaça da exfiltração de dados. Várias formas de prevenção têm sido recomendadas, incluindo o isolamento de uma máquina de qualquer rede externa (*air-gapping*) [Maas 2013]. Este trabalho tem como objetivo estudar novos métodos ópticos pouco intrusivos para exfiltração de dados, junto às suas limitações e eficiência. Ao antecipar o impacto desta ameaça, as organizações tornam-se capazes de proteger seus dados contra exfiltração, projetando contramedidas adequadas. Ao mesmo tempo, aprimorar os mecanismos de exfiltração é importante para informar ativistas e delatores sobre as formas plausíveis para recolher provas de comportamento desonesto e vazar informações de forma sutil e eficiente.

*Agradecimentos à FAPESP pela concessão da bolsa de iniciação científica, processo 2015/13876-7.

A abordagem proposta tem vantagens quando comparada com resultados recentes da literatura. Ao contrário de outras técnicas que exigem que um *software* malicioso seja instalado na máquina atacada para controle privilegiado do *hardware*, apenas um dispositivo de armazenamento portátil, *pendrive* ou HD externo, precisa ser inserido no ambiente alvo, possivelmente empregando técnicas de engenharia social [Tischer et al. 2016]. Depois que a unidade de armazenamento é ligada à máquina fisicamente isolada e os dados são recebidos, o dispositivo malicioso pode começar a transmitir dados capturados para o adversário, sem exigir um alto nível de privilégio ou interferir de qualquer forma com o sistema. Além disso, não há a necessidade do atacante recuperar o dispositivo mais tarde, uma vez que os dados são capturados por uma câmera ou por um *smartphone* no interior do ambiente. Dessa forma, não há intervenção física no sistema atacado, que poderia ser relevada posteriormente em uma investigação forense, em caso de suspeita. A exfiltração óptica de dados através de LEDs também oferece uma maior largura de banda em comparação com outros métodos, permitindo que o invasor obtenha dados de maneira mais eficiente e permitindo uma maior flexibilidade para ser usada em múltiplos cenários.

2. Trabalhos relacionados

Diversos trabalhos na área de exfiltração de dados foram realizados nos últimos anos utilizando diferentes abordagens, como LED do monitor [Sepetnitsky et al. 2014], som da ventoinha [Guri et al. 2016b], emissão de calor [Guri et al. 2015] e ondas de rádio do barramento USB [Guri et al. 2016a]. Tais abordagens são tipicamente demonstradas para máquinas sem conexão a redes externas, nas quais vazar dados é uma tarefa complexa. As velocidades de transmissão são comumente baixas e os mecanismos de recepção das informações são muitas vezes de difícil elaboração. Por exemplo, utilizar calor na exfiltração de dados impõe uma largura de banda extremamente limitada, devido ao tempo necessário para aquecer e esfriar o computador somente executando um componente de *software* em segundo plano. Outra desvantagem destes métodos é o elevado nível de intrusão, que leva à necessidade de modificar um periférico antes do mesmo ter sido usado no ambiente alvo, o que pode facilitar a detecção; ou que um *software* malicioso seja instalado na máquina atacada para um controle privilegiado do *hardware*. Do ponto de vista de desempenho, LEDs de um monitor impõem um limite de frequência de 25Hz e transmissão por calor enfrenta um limite ainda menor.

Embora haja propostas para exfiltração de dados usando dispositivos de armazenamento portátil [Zaddach et al. 2013, Clark et al. 2009], a abordagem óptica que buscamos é nova e não requer adulteração de *hardware* ou inserção de *malware* no dispositivo para que ele possa acessar e vazar dados confidenciais do computador. A Tabela 1 resume alguns aspectos de diferentes técnicas de exfiltração de dados encontradas na literatura com a proposta nesse projeto. O nível de intrusão alto se refere à necessidade de infecção com *software* privilegiado, enquanto níveis médio (dispositivo malicioso) e baixo (apenas programa em nível de usuário) relaxam esse requisito.

3. Protótipo para exfiltração óptica

No cenário considerado, assume-se primeiramente que o atacante é capaz de infectar uma máquina conectada (computador em rede ou *smartphone*) dentro do perímetro de segurança para controlar sua câmera. A câmera fica à espera que LEDs em seu campo de visão comecem a piscar de uma maneira específica, com uma sequência para sincronizar

Tipo	Velocidade	Intrusividade	Taxa de transmissão
LED do monitor (luz)	Alta	Alta	25Hz
Bitwhisper (calor)	Baixa	Baixa	1 a 8 bits/hora
Fansmitter (som)	Baixa	Alta	900 bits/hora
USBee (rádio)	Alta	Alta	20 a 80 bytes/segundo
Método apresentado	Alta	Médio	30 bits/segundo

Tabela 1. Comparação dos métodos de exfiltração estudados na literatura: luz [Sepetnitsky et al. 2014], som [Guri et al. 2016b], calor [Guri et al. 2015] e rádio [Guri et al. 2016a].

o transmissor e o receptor. Estes LEDs fazem parte de um dispositivo que funciona como uma unidade de armazenamento simples, embora tenha *firmware* modificado para transmitir dados específicos armazenados dentro da memória. O dispositivo previamente introduzido é então conectado a uma máquina isolada, da qual o atacante objetiva vazar informações. Após a câmera monitorar o LED e completar o protocolo de sincronização, os dados podem ser lidos e enviados para um servidor controlado pelo atacante. O projeto está dividido em duas partes: o algoritmo para transmitir dados através do meio óptico a partir do dispositivo de armazenamento e o sistema de captura e decodificação.

3.1. Transmissor

A plataforma escolhida foi uma placa Teensy2, compatível com Arduino. O *framework* LUFA[1] é utilizado para construir o *pendrive* com um cartão SD comum[2], que funciona como memória do dispositivo. Alguns LEDs finalizam o projeto.

A Figura 1 apresenta a versão atual do protótipo do transmissor. O protótipo é ligado a um computador através de uma porta USB comum e pode ser usado como um dispositivo de armazenamento regular, utilizando um protocolo de comunicação padronizado. A primeira versão do protótipo era equipada com um único LED, representando um indicador de atividade. Como a largura de banda ficou limitada, a solução adotada inclui múltiplos LEDs, ampliando substancialmente a capacidade de transmissão. Para evitar aparência suspeita, os LEDs emitem luz infravermelha, invisível ao olho humano. Além disso o dispositivo foi colocado em uma capa transparente, para parecer com um dispositivo real. As extensões ou nomes dos arquivos de interesse são colocados na memória do dispositivo antes de sua entrada no perímetro de segurança. Quando o dispositivo for utililizado, buscará os dados escolhidos e logo em seguida a transmissão será iniciada.

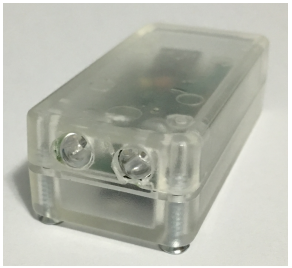


Figura 1. Protótipo do dispositivo.

3.2. Receptor

O receptor é representado por uma *webcam* comum, capturando 30 quadros por segundo na resolução de 720p. Para a parte de *software*, utilizamos a biblioteca gráfica

[1] `http://www.fourwalledcubicle.com/LUFA.php`

[2] `http://elasticsheep.com/2010/04/teensy2-usb-mass-storage-with-an-sd-card/`

OpenCV 3.0[3] para manipular as imagens da câmera e obter a informação transmitida pelos LEDs do transmissor. O componente de *software* identifica a fonte dos dados a serem transmitidos, mantendo o controle de cada LED e de seu comportamento. Após a sincronização, o *software* começa a decodificar a informação através de um código de correção de erros para detectar falhas na transmissão. A cadeia de bits corrigida pode ser armazenada na máquina remota ou pode ser enviada para um servidor web através da conexão móvel do *smartphone* ou da rede externa disponível no computador acoplado na *webcam*. Este servidor é controlado pelo atacante, proporcionando um refúgio seguro para os dados finalmente extraídos.

O algoritmo para a detecção de um LED é simples e se baseia na diferença entre dois quadros consecutivos para determinar quais as partes da imagem têm a mudança do estado do LED e se há ruído de fundo. Após o protocolo inicial terminar, o receptor sabe a localização de cada LED e pode monitorar os estados para receber mensagens. Os bits são transmitidos da maneira mais simples possível: o LED ligado equivale a um valor 1, e um valor 0 caso contrário.

3.3. Protocolo

O protocolo de transmissão começa a partir da sequência específica de bits transmitidos no início da mensagem. A sequência é usada para marcar quando a mensagem está começando novamente para que o receptor possa parar de decodificar novos bits e começar a procurar pedaços de informações que foram perdidas durante as mensagens anteriores. Outra finalidade é dar à câmera algum tempo para encontrar os LEDs na primeira leitura.

Após o protocolo inicial ser concluído, os bits da mensagem são transmitidos através dos LEDs. No entanto, este tipo de transmissão ruidosa introduz alguns erros nos dados que estão sendo enviados. Por causa do tempo que o LED leva para acender ou apagar, um bit às vezes é lido incorretamente, geralmente no final de uma sequência de bits iguais. A maneira mais fácil encontrada para lidar com este problema foi utilizar o código de Hamming que permite a correção de um bit por palavra. Esta escolha foi suficiente para acabar com os erros nos testes e tem resolvido também os casos em que o erro ocorre quando o LED começa a apagar e a câmera captura esse momento transitório, resultando em alguns bits trocados.

Uma característica importante sobre o protocolo escolhido foi a baixa quantidade de bits de paridade, não afetando a largura de banda significativamente. Tipos mais complexos de códigos de correção de erros foram consideradas, mas descartadas devido a um número muito maior de bits de paridade.

4. Resultados preliminares

O principal objetivo é transmitir o conteúdo do cartão SD, logo o dispositivo consegue escolher um arquivo específico da memória para enviar. Estes dados são transmitidos através de múltiplos LEDs, cada um representando um bit diferente. Em termos de velocidade, o protótipo é capaz de transmitir 10-15 bits por segundo por LED, sendo a principal limitação encontrada no tempo necessário para acender ou desligar o LED completamente. Como o protótipo faz uso de 2 LEDs diferentes simultâneos, velocidades de transmissão até 30 bits por segundo foram observadas.

[3] `http://opencv.org/about.html`

O método já pode ser utilizado para enviar dados pequenos como senhas e chaves criptográficas, devido à sua relativamente baixa velocidade quando comparada ao tamanho típico de arquivos, apesar de ser rápida o suficiente para transmitir essas informações em poucos segundos. Documentos pequenos também podem ser transmitidos se o atacante tiver tempo suficiente disponível. Um desafio permanente do projeto é conseguir a máxima largura de banda possível para maximizar a eficácia da exfiltração. A maior limitação é a taxa de quadros da câmera na recepção, porque cada LED é capaz de transmitir mais de 30 bits por segundo. Essa foi a principal razão que levou a usar vários LEDs simultâneos. Esta modificação permitiu que a taxa de quadros fosse muito mais elevada, com apenas um pouco mais de esforço. Os resultados preliminares não estão tão longe de alguns trabalhos relacionados publicados [Sepetnitsky et al. 2014]. Dadas as limitações da abordagem óptica, a velocidade atingida parece promissora, tornando possível enviar dados sensíveis em apenas alguns segundos, e alguns pequenos arquivos em minutos.

5. Conclusão

Com os resultados obtidos pretende-se continuar o projeto aprimorando o protótipo para torná-lo apto à uma aplicação real. O principal objetivo é aumentar a velocidade de transmissão para possibilitar a transferência de arquivos de forma discreta e eficaz. Como o intuito do projeto é estudar formas novas de exfiltração, o estudo das formas de prevenção para tal método se mostra muito importante, para evitar de forma eficiente que dados sensíveis sejam vazados.

Referências

Clark, J., Leblanc, S., and Knight, S. (2009). Hardware trojan horse device based on unintended USB channels. In *NSS*, pages 1–8.

Guri, M., Monitz, M., and Elovici, Y. (2016a). Usbee: Air-gap covert-channel via electromagnetic emission from USB. *CoRR*, abs/1608.08397.

Guri, M., Monitz, M., Mirski, Y., and Elovici, Y. (2015). Bitwhisper: Covert signaling channel between air-gapped computers using thermal manipulations. In *IEEE CSF*, pages 276–289.

Guri, M., Solewicz, Y. A., Daidakulov, A., and Elovici, Y. (2016b). Fansmitter: Acoustic data exfiltration from (speakerless) air-gapped computers. *CoRR*, abs/1606.05915.

Maas, P. (2013). How Laura Poitras helped Snowden spill his secrets. New York Times: `http://www.nytimes.com/2013/08/18/magazine/laura-poitras-snowden.html`.

Sepetnitsky, V., Guri, M., and Elovici, Y. (2014). Exfiltration of information from air-gapped machines using monitor's LED indicator. In *IEE JISIC*, pages 264–267.

Tischer, M., Durumeric, Z., Foster, S., Duan, S., Mori, A., Bursztein, E., and Bailey, M. (2016). Users Really Do Plug in USB Drives They Find. In *IEEE Security & Privacy*, San Jose, California, USA.

Zaddach, J., Kurmus, A., Balzarotti, D., Blass, E., Francillon, A., Goodspeed, T., Gupta, M., and Koltsidas, I. (2013). Implementation and implications of a stealth hard-drive backdoor. In *ACSAC*, pages 279–288. ACM.

Uma Revisão Sistemática sobre Privacidade do Usuário em Aplicações MCS

Guibson Souza[1], **Ingrid Santos**[1], **Karla Pereira**[1], **Eduardo Feitosa**[1]

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
CEP 69.077-000 – Manaus – AM – Brazil

{nedel,efeitosa}@icomp.ufam.edu.br

Abstract. *Mobile Crowd Sensing (MCS) applications allow mobile users to collect, share and comment on information provided by third parties or himself through different sensors (GPS, accelerometer, camera, microphone, etc.). As the handling of this information is carried out directly by end users, privacy has become a critical and important issue that should be evaluated. This paper presents a Systematic Literature Review on user privacy in MCS context. The goal is to help the end users understand their privacy and how to secure it in MCS applications.*

Resumo. *Aplicações de Mobile Crowd Sensing (MCS) permitem que usuários de dispositivos móveis coletem, compartilhem e comentem informações fornecidas por terceiros ou ele mesmo, através de diferentes sensores (GPS, acelerômetro, câmera, microfone, entre outros). Como a manipulação dessas informações são realizadas diretamente por usuários finais, sua privacidade tornou-se uma questão crítica e importante que deve ser avaliada. Neste trabalho é apresentada uma Revisão Sistemática da Literatura sobre privacidade do usuário no contexto de MCS. O objetivo é contribuir para que o usuário entenda o que é a sua privacidade e como garanti-la em aplicações MCS.*

1. Introdução e Conceituação

Mobile Crowd Sensing, ou simplesmente MCS, é um paradigma de sensoriamento que tira proveito do número de dispositivos que um usuário possui e de sua mobilidade para adquirir conhecimento local (informações ambientais, de infraestrutura e social) e compartilhar esse conhecimento dentro de uma esfera social. A ideia é que os dados coletados por um usuário comum, através dos diferentes sensores (GPS, acelerômetro, câmera, sensor de temperatura) do dispositivo móvel, podem contribuir de alguma forma para melhorar e/ou auxiliar a vida das outras pessoas.

Para Ganti et al. [Ganti et al. 2011], os pesquisadores responsáveis por cunharem o termo, *Mobile Crowd Sensing se refere a uma variada e ampla gama de modelos de sensoriamento no qual indivíduos com dispositivos computacionais e sensores são capazes de coletar e contribuir com dados valiosos para diferentes aplicações*. A principal diferença entre MCS em relação a outras plataformas de sensoriamento participativo reside no fato de que toda ação de registro (coleta de dados, informações e comentários sobre os dados) é feita a partir do dispositivo móvel do participante, podendo estar conectado a qualquer rede de acesso à Internet. Um grande benefício deste paradigma é o baixo

custo de implantação, pois não é preciso ter uma estrutura de hardware específica para ser participante ou usuário deste tipo de aplicação.

Contudo, devido a suas características únicas, a privacidade em MCS traz preocupações com a divulgação direta da identidade dos participantes bem como com a divulgação de atributos sensíveis, incluindo locais (por exemplo, endereço residencial ou do trabalho) e outras informações privadas como atividades pessoais ou condições (por exemplo, estilo de vida ou doença) que permitam inferir a identidade dos participantes [Pournajaf et al. 2014].

Neste contexto, este artigo visa investigar se a privacidade do usuários é considerada em aplicações MCS, através de uma Revisão Sistemática da Literatura (RSL).

2. Revisão Sistemática da Literatura

O objetivo desta Revisão Sistemática da Literatura (RSL), delineada a partir do paradigma GQM [Basili and Rombach 1988], é **Analisar** publicações científicas relacionadas à privacidade em MCS; com o **Propósito** de identificar conceitos, características, fatores e ferramentas; no que diz **Respeito** ao usuário; no **Contexto** de aplicações Móveis.

Com base neste objetivo, três (3) questões de pesquisas foram criadas: **Q1** - Existe preocupação com a privacidade do usuário em MCS? **Q2** - O que usuário percebe ou entende sobre privacidade em MCS? **Q3** - Quais técnicas são utilizadas na privacidade do usuário em aplicações MCS?

2.1. Execução e Resultados

A revisão sistemática da literatura foi realizada entre Março e Junho de 2016, utilizando a máquina de busca Scopus (`https://www.scopus.com`), resultando no retorno de 193 artigos. Inicialmente, como primeiro filtro, foram analisados e considerados apenas o título do artigo, o *abstract* e as palavras-chave. Ao final desta etapa, foram selecionados 71 artigos. No segundo filtro foi considerada e analisada a leitura completa dos artigos selecionados. Nesta fase foram selecionados 16 artigos. Também foram realizadas pesquisas manuais que retornaram 6 artigos, resultando em duas inclusões. Ao final do processo, 18 artigos foram selecionados por estarem relacionados a esse domínio.

Vale ressaltar que alguns dos trabalhos encontrados relacionam características empregadas em paradigmas fora ou além de *Mobile Crowd Sensing*, como Sensoriamento Participativo e *Mobile CrowdSourcing*.

3. Análise

Esta seção discute as questões de pesquisa com base nos trabalhos encontrados na RSL sobre Privacidade de Usuário em MCS.

3.1. Existe preocupação com a privacidade do usuário em MCS?

Sim. Existem alguns estudos [Pournajaf et al. 2014, Christin 2010, Brush et al. 2010] que correlacionam a preocupação com a privacidade com as diferentes modalidades de sensoriamento usadas, bem como a forma como os usuários entendem, selecionam e se comportam em relação a privacidade. Contudo, todos são unanimes em afirmar que os usuários não são considerados no processo de avaliação da usabilidade e utilidade das soluções de privacidade, mesmo sabendo que tais aplicações gerenciam grandes volumes de informações que tipicamente são e podem ser tornadas públicas.

3.2. O que usuário percebe ou entende sobre privacidade em MCS?

De forma geral, os dados obtidos e coletados pelos e dos usuários são controlados pela infraestrutura da aplicação e o usuário não tem conhecimento algum do que acontece com seus dados. Entretanto, alguns aspectos de segurança dependem unicamente do usuário, ou melhor, das suas preferências (intervenções realizadas para ajustar parâmetros de acordo com suas preferências).

Christin et al. [Christin 2010] afirmam que a privacidade no sensoriamento participativo está centrada em dois componentes estritos: pessoa e dados. Assim, o usuário deve entender, escolher e controlar os dados coletados que serão compartilhados, bem como filtrar quem receberá esses dados e a duração da disponibilidade destes. O objetivo é aumentar a consciência dos usuários acerca das questões de privacidade.

Gustarini et al. [Gustarini et al. 2016] definiram uma métrica (intimidade) para investigar a percepção do anonimato do usuário em MCS, os fatores que o influenciam a aceitar ser anônimo ou não, e o contexto em que tomam a decisão de compartilhar os dados. A intimidade concentra o contexto do usuário em três elementos principais: localização semântica dos usuários, o número e o tipo de pessoas que estão ao seu redor. Ao aplicarem um questionário sobre intimidade - escala de valores entre 1 (completamente) e 6 (não estou íntimo) - com usuários de MCS no contexto de onde se encontrava naquele momento (local), quantas pessoas estavam ao seu redor e se eles se sentiam a vontade de compartilhar o conteúdo da pesquisa anonimamente ou não, os autores perceberam que a maioria dos usuários tem dificuldades para inferir sobre os dados que estão sendo coletados, para qual finalidade e como são coletados.

3.3. Quais técnicas são utilizadas na privacidade do usuário em aplicações MCS?

De acordo com [Laurido and Feitosa 2015], as principais técnicas para garantir a privacidade do usuário em aplicações MCS são: (i) permitir que os usuários expressem suas preferências de privacidade; (ii) a distribuição de tarefas de sensoriamento anonimamente, para assegurar o anonimato e a privacidade de localização; e (iii) o uso de anonimato, para remover qualquer informação que possa identificar o usuário durante a distribuição e realização de tarefas.

Dentre essas, a mais relevante e com maior grau de interdisciplinaridade é permitir que o usuário tenha a capacidade de expressar suas preferências sobre privacidade em uma determinada aplicação. Das et al. [Das et al. 2010] propuseram um esquema binário, com acesso completo ou nenhum acesso aos dados do usuário. A ideia é que os usuários possam decidir ativar seletivamente as medições do sensor, dependendo de uma variedade de fatores como a presença em locais sensíveis (casa ou escritório) e de pessoas de seu convívio social (presença de amigos ou familiares). Por exemplo, amostras podem ser recolhidas a cada hora ao invés de a cada 15 segundos ou informações de localização podem ou não ser capturadas dependendo da localização.

Christin et al. [Christin et al. 2014] argumentam que existe a viabilidade de dar controle aos usuários sobre a proteção de sua privacidade, permitindo que personalizem suas preferências. Para tanto, elaboram um projeto para conscientizar os usuários sobre ameaças de privacidade em aplicações de sensoriamento móveis, a fim de motivar a necessidade de configurar uma aplicação para preservação da privacidade. Selecionaram um aplicativo que monitora ruídos no ambiente através de smartphones e realizaram um

experimento em dois (2) passos. O primeiro foi projetar interfaces explicando a funcionalidade da privacidade no aplicativo e como o usuário pode fazer sua proteção escolhendo as configurações de acordo com as suas preferências. O segundo foi realizar um estudo com 20 participantes, testando e avaliando diferentes interfaces de privacidade, completando tarefas guiadas e tarefas livres de acordo com as definições dadas e personalizando suas próprias configurações. O resultado do estudo mostrou que os participantes gostaram deste controle adicional oferecido, mas alguns se complicaram devido à complexidade do problema. O experimento foi realizado no sistema iOS e as interfaces projetadas foram integradas ao aplicativo "NoiseCapture".

4. Conclusões

Neste trabalho foi apresentada uma Revisão Sistemática da Literatura sobre privacidade do usuário em aplicações MCS. Dos 193 artigos encontrados na pesquisa, 18 artigos foram selecionados e estudados. A partir dos estudos pode-se observar que existem poucos trabalhos visando melhorar a privacidade do usuário nos diversos ambientes de sensoriamento usando dispositivos móveis. Além disso, constatou-se que nem mesmo os maiores interessados, os usuários dessas aplicações, sabem o significado de privacidade e como é feito esse processo para garantir a integridade dos seus dados.

References

Basili, V. R. and Rombach, H. D. (1988). The tame project: Towards improvement-oriented software environments. *IEEE Trans. on software engineering*, 14(6):758–773.

Brush, A. B., Krumm, J., and Scott, J. (2010). Exploring end user preferences for location obfuscation, location-based services, and the value of location. In *Proceedings of the 12th ACM UbiComp*, pages 95–104. ACM.

Christin, D. (2010). Impenetrable obscurity vs. informed decisions: privacy solutions for participatory sensing. In *2010 8th IEEE PERCOM*, pages 847–848.

Christin, D., Engelmann, F., and Hollick, M. (2014). Usable privacy for mobile sensing applications. In *IFIP International Workshop on Information Security Theory and Practice*, pages 92–107. Springer.

Das, T., Mohan, P., Padmanabhan, V. N., Ramjee, R., and Sharma, A. (2010). Prism: platform for remote sensing using smartphones. In *Proceedings of the 8th MobiSys*, pages 63–76. ACM.

Ganti, R., Ye, F., and Lei, H. (2011). Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39.

Gustarini, M., Wac, K., and Dey, A. K. (2016). Anonymous smartphone data collection: factors influencing the users' acceptance in mobile crowd sensing. *Personal and Ubiquitous Computing*, 20(1):65–82.

Laurido, J. and Feitosa, E. (2015). Segurança em Mobile Crowd Sensing. In *Livro de Minicursos do SBSeg2015*, chapter 2, pages 49–98. SBC.

Pournajaf, L., Xiong, L., Garcia-Ulloa, D. A., and Sunderam, V. (2014). A survey on privacy in mobile crowd sensing task management. Technical report, Emory University.

SimBo - Ambiente de Simulação dedicado ao Estudo de Botnets

Felipe Balabanian, Moisés Danziger, Marco Aurélio Amaral Henriques

Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (UNICAMP)
Campinas – SP – Brasil

[bala012,danziger,marco]@dca.fee.unicamp.br

Abstract. *The main objective of this project is to build an appropriate environment for the simulation of intrusions into computer networks caused by botnets. Such an environment should facilitate experimentation for researchers, allowing a better understanding of botnets operation and validation of new approaches to detect and disarticulate them. The discrete events system simulator Omnet++ was extended with several modules to facilitate the simulation of botnets and their countermeasures.*

Resumo. *O principal objetivo deste projeto é a construção de um ambiente apropriado para a simulação de invasões em redes de computadores causadas por botnets. Tal ambiente deverá facilitar a experimentação aos pesquisadores, permitindo um melhor entendimento do funcionamento de botnets e a validação de novas abordagens de detecção e desarticulação deste tipo de rede. O simulador de sistema de eventos discretos Omnet++ foi estendido com vários módulos para facilitar a simulação de botnets e de medidas de defesa.*

1. Introdução

O que são botnets? A princípio uma botnet é um conjunto de computadores infectados, bots, controlados por um operador denominado botmaster. As botnets têm crescido como poderosas ferramentas criminais [1], capazes de realizar roubos de dados sensíveis como cartões de crédito, quebra de senhas criptografadas e negação de serviço (DDoS) a empresas e governos.

Buscar os dados em ambientes reais é uma tarefa árdua visto que as maiores fontes de informação estão presentes nas empresas, que na maioria das vezes são contrárias em cedê-las, já que dados de tráfego de rede são tratados como dados sensíveis e por isso devem ser cuidadosamente controlados. A exigência de um alto número de máquinas para estudo das ameaças que as botnets representam resulta em um problema de escalabilidade (com consequente limitação financeira). Outro problema refere-se a segurança da própria infraestrutura usada pelo pesquisador que trabalha com código hostil, pois ela pode ser infectada e se tornar mais um vetor de propagação de bots pela rede de sua instituição.

Como uma forma alternativa para responder a tais desafios, propomos o uso de um simulador específico para testes de malwares em geral, botnets em particular. Sua utilização pode ser bem mais simples, já que não exige grandes quantidades de hardware e pode ser executado em equipamentos de baixo custo.

Este trabalho fez a implementação de um ambiente de testes (*framework*) que inclui novas plataformas e abordagens tecnológicas. Ele se baseia na plataforma

Omnet++ [2] que foi estendida com a criação de vários módulos voltados para a simulação de botnets.

2. Simulador de Eventos Discretos Omnet++

O ambiente Omnet++ é um simulador de eventos discretos baseado em C++ que permite simular nós conectados por uma rede para troca de mensagens. Como este trabalho focou na construção de ferramentas que permitam simular botnets e sistemas de defesa e detecção em redes de computadores, utilizou-se o Omnet++ em conjunto com a biblioteca Inet especializada em protocolos de redes, que fornece módulos para a simulação dos protocolos mais usados como o TCP, UDP, IP, Ethernet, etc.

Às bibliotecas-padrão foram adicionados diversos módulos construídos em C++ (linguagem base do Omnet++) com a finalidade de alcançar o objetivo de criar um framework de redes voltado para estudo de cibersegurança. Entre eles estão:

- módulo que simula uma botnet genérica simples;
- módulo de teste de conectividade (ping);
- módulo de geração de tráfego de rede TCP;
- módulo de captura de pacotes (sniffer);
- módulo de comunicação com programas externos, tais como o Matlab e o JaCaMo (plataforma de sistemas multi-agentes) [3] para a realização de simulações e análises mais sofisticadas.

2.1. Módulo Botnet Genérica

O módulo que simula o ciclo de vida de uma botnet realiza cinco operações típicas:

- varredura da rede;
- busca por vulnerabilidades;
- invasão de computadores vulneráveis;
- aviso informativo do seu status ao botmaster;
- escuta por novos comandos do botmaster.

Na Figura 1 temos um exemplo de simulação executando o Módulo Botnet no qual as máquinas infectadas são identificadas pela coloração vermelha. Com este módulo é possível verificar o comportamento de uma botnet genérica numa topologia de rede e verificar como se dá a sua evolução. Caso seja do interesse do pesquisador, é possível modificar os parâmetros e funções deste modelo de botnet para um modelo específico que se queira estudar.

A partir desse módulo é possível simular alguns eventos como, por exemplo, um ataque DDoS (Distributed Denial-of-Service) onde, em determinado instante, se deseja iniciar um elevado número de requisições a um servidor a partir dos diversos nós infectados na rede.

O módulo botnet foi desenvolvido como uma aplicação TCP contida no módulo host. Na Figura 2 é possível observar os componentes internos de um host com duas

aplicações TCP (sendo uma a bot) executando e interagindo com o bloco da camada de rede. Também se atenta ao módulo sniffer que não está conectado a camada de rede, pois dispõe de outra metodologia para capturar o fluxo de pacotes.

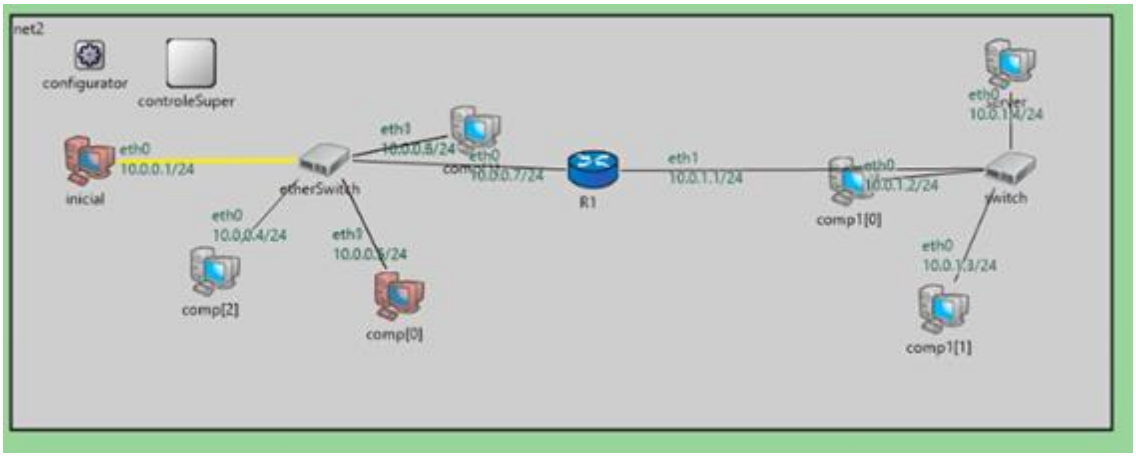


Figura 1. Simulação utilizando o Modulo de Botnet. Computador com nome “inicial” age como epicentro da contaminação (e centro de controle) e contamina sua primeira vítima (comp[0]).

2.2. Módulo de Comunicação Externa

A fim de estender os ferramentais disponíveis no Omnet++, optou-se por desenvolver um módulo específico para a comunicação do ambiente de simulação com outros ambientes como, por exemplo, o MatLab e o JaCaMo, o que abre caminho para a criação de experimentos com botnets e detectores mais sofisticados, baseados em técnicas mais recentes, como as de inteligência computacional.

A integração da plataforma com ferramentas poderosas como o Matlab permite a análise do cenário de simulação em tempo de execução pelos diversos recursos e gráficos disponibilizados pelas mesmas. Facilita-se assim a análise, visualização e controle da rede simulada.

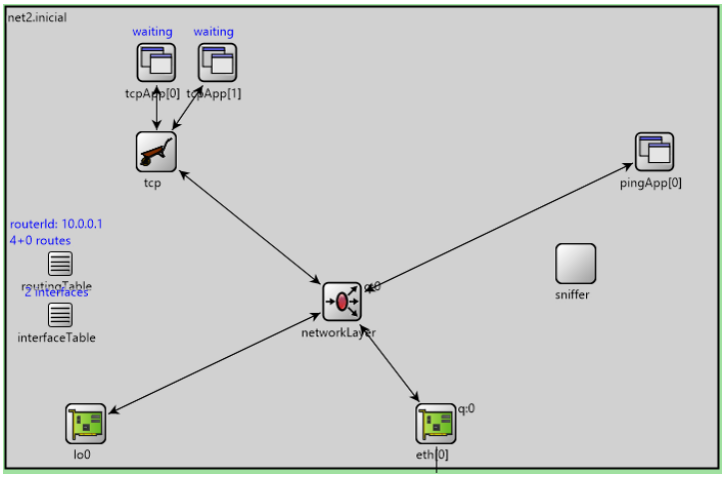


Figura 2. Estrutura interna de um módulo host.

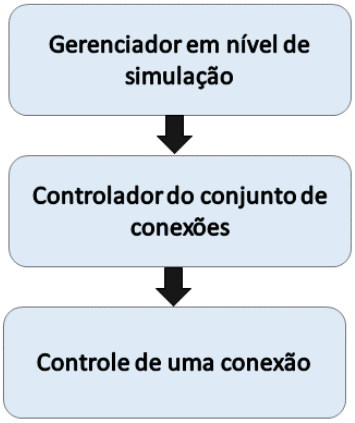


Figura 3. Hierarquia de controle.

O módulo de comunicação é flexível a ponto de gerenciar múltiplas conexões com diversos ambientes externos. Para isso utiliza-se da hierarquia ilustrada na Figura 3, onde o primeiro módulo cuida do momento apropriado de se verificar as conexões externas durante o fluxo de execução da simulação. O segundo módulo se preocupa em organizar

a execução do conjunto de controladores de conexão. E o terceiro módulo dedica-se ao controle da conexão e gerenciamento de tokens entre o Omnet++ e o programa externo a fim de se obter uma comunicação ordenada.

O controle de conexão é gerenciado por tokens conforme o protocolo da Figura 4. Neste gerenciamento são criados quatro arquivos, sendo dois para controle e dois para troca de dados. Nos arquivos de controle são colocados os tokens correspondentes à permissão de enviar ou receber instruções, enquanto os arquivos de dados correspondem aos *inputs/outputs* dos módulos que estão se comunicando. Todos arquivos são iniciados com o caractere zero, '0', para garantir um estado inicial consistente.

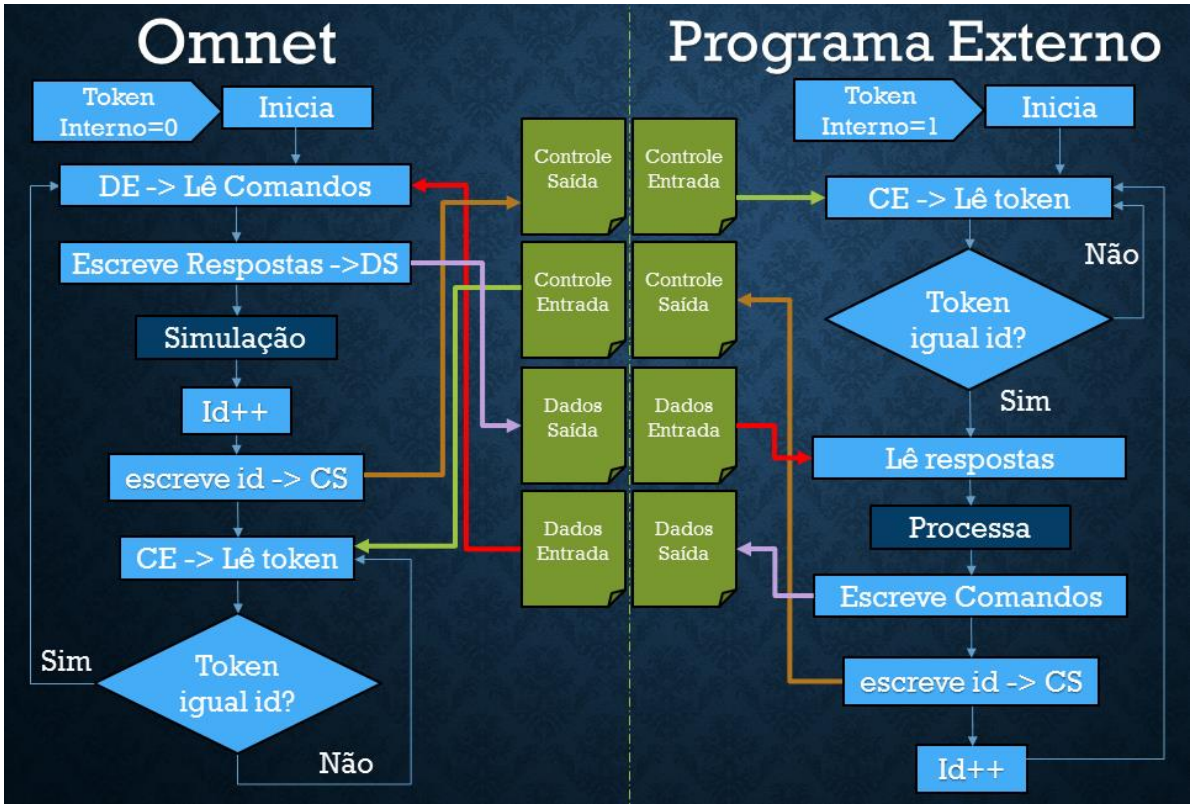


Figura 4. Fluxo de controle de conexão.

Do lado da aplicação Omnet++ o fluxo se inicia com a definição do token interno como zero. Prossegue lendo os comandos presentes no arquivo de dados de entrada (DE) e depositando no arquivo de dados de saída (DS) as requisições externas. Este procedimento garante que uma nova conexão possa ser aberta em qualquer momento da simulação, antes ou depois do programa externo ter sido iniciado. Após esse processo o Omnet++ volta à simulação, já com a conexão iniciada. A simulação continua até o módulo de gerenciamento informar que é o momento de uma nova iteração. O token interno é acrescido em uma unidade e escrito no arquivo Controle de Saída (CS). A partir desse momento entra-se em uma verificação cíclica do arquivo Controle de Entrada (CE) esperando uma resposta do programa externo e comparando o valor recebido com o token interno. Caso o valor recebido (correspondente ao token do programa externo) coincida com o token interno, continua-se para o tratamento dos dados. O Omnet++ recebe os dados requisitados e novos comandos do arquivo DE, processa o que for possível e escreve no arquivo DS os dados processados e novas requisições para o programa externo. Do lado do programa externo segue-se o mesmo procedimento, mas iniciando-se na etapa da verificação do token externo e com o token interno igual a um.

3. Código Fonte

Todos os módulos que foram desenvolvidos neste trabalho utilizam a linguagem C++, base do próprio Omnet++, o que conferiu ao ambiente a velocidade e escalabilidade desejadas. Para utilizar os módulos descritos neste trabalho, basta instalá-los com o Omnet++/Inet. O código encontra-se disponível para ambientes Linux e Windows [4].

4. Utilizações do SimBo

A plataforma aqui descrita está servindo de base para pesquisas envolvendo métodos de detecção de botnets. Tem sido usada também como principal ferramenta de simulação de novas botnets com novos tipos de paradigmas de comportamento e operação, tais como botnets que utilizam inteligência computacional. A ferramenta também serve para a análise de metodologias que visam prevenir, identificar e se recuperar de ataques DDoS.

Apesar dos usos descritos acima estarem em seu início e não terem ainda exigido tudo que SimBo tem a oferecer, eles já comprovaram a alta flexibilidade que o simulador oferece e já se beneficiam dos recursos extras oferecidos ao ambiente Omnet++.

Espera-se que, com uma gradual divulgação deste trabalho, novos usuários surjam e novos feedbacks ajudem a aprimorar cada vez mais os recursos oferecidos por este simulador. Acreditamos que deverão surgir também novos usos para SimBo além da simulação de botnets, já que os módulos que permitem a troca de dados com ferramentas externas deverão possibilitar que outros tipos de simulação sejam realizados.

6. Conclusões

A plataforma SimBo disponibiliza uma ferramenta para o estudo de botnets. Destaca-se que, com os módulos de comunicação externa desenvolvidos, é possível coletar e enviar dados em tempo de simulação para a plataforma, expandindo as possibilidades para análise de dados ou controle da simulação. Foi implementado com o Matlab o monitoramento de padrões de comunicação e detectores de intrusão simples, mas algoritmos mais complexos podem ser desenvolvidos. A possibilidade de controlar simulações externamente usando, por exemplo, o pacote JaCaMo que implementa sistemas multiagentes permite simular novos tipos de botnets baseados, por exemplo, em técnicas de inteligência computacional. Seja como for, percebe-se que a plataforma SimBo pode se tornar uma base importante para que os pesquisadores implementem e testem suas novas ideias por meio de simulações confiáveis e de baixo custo.

7. Referências

[1] TiirmaaKlaar, H., Gassen, J., GerhardsPadilla, E. and Martini, P.; Botnets. SpringerBriefs in Cybersecurity, book. Springer, 2013.

[2] Omnet++ (Acessado em julho de 2016). OMNeT++ Discrete Event Simulator. https://omnetpp.org/.

[3] Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. In: Science of Computer Programming (2011).

[4] Felipe Balabanian, Código-fonte dos módulos do simulador de botnets, https://github.com/regras/simbo/.

Extensão do conjunto de instruções para implementação segura de X25519

Antonio C. Guimarães, Edson Borin, Diego F. Aranha

Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Av. Albert Einstein, 1251 – 13.083-852 – Campinas – SP – Brasil
antonio.junior@students.ic.unicamp.br, {edson, dfaranha}@ic.unicamp.br

Abstract. *This papers presents the results of introducing a new conditional exchange instruction in the x86 architecture and its application in a side-channel resistant implementation of the X25519 protocol based on Diffie-Hellman. Building upon an implementation included in SUPERCOP, we developed 4 different versions of the algorithm: 1 using the new instruction and 3 restricted to the instructions already available. The MARSSx86 simulator was used for prototyping and validation of security and efficiency. Experimental results illustrate the vulnerability in an insecure implementation and the improvements in security and 1.4% in performance after the new instruction was introduced.*

Resumo. *Este artigo descreve os resultados da introdução de uma nova instrução de troca condicional para a arquitetura x86 e sua utilização na implementação resistente a ataques de canal lateral do protocolo X25519 baseado em Diffie-Hellman. Baseando-se na implementação presente no SUPERCOP, foram desenvolvidas 4 versões do algoritmo, sendo uma com a nova instrução e outras 3 utilizando apenas instruções já existentes. Para a prototipação da instrução e validação de segurança e desempenho foi utilizado o simulador MARSSx86. Os resultados experimentais permitiram demonstrar a vulnerabilidade em uma versão insegura, assim como o ganho em segurança e de 1.4% em desempenho com a introdução da nova instrução.*

1. Introdução

A preocupação com ataques de canal lateral em dispositivos embarcados cresce à medida que a Internet das Coisas se populariza. Devido à baixa capacidade computacional, tais dispositivos tiveram sua construção focada especialmente na eficiência, muitas vezes em detrimento da proteção contra vazamento de informações dos dados sensíveis ali processados. Dentre tais dados, os algoritmos criptográficos utilizados nesses dispositivos também estão vulneráveis, sendo, para diversas implementações comuns, como o AES e o RSA, a obtenção de informação secreta por meio de ataques de canal lateral amplamente descrita pela literatura [1-4].

De modo geral, as soluções em *software*, até agora propostas e utilizadas, encontram limitações na falta de suporte do *hardware* para execução segura. Ainda que obtenham sucesso em coibir alguns tipos de vazamento, muitas vezes acabam por ser incompletas, ao permitirem o vazamento por outros meios, ou até mesmo criar novas vulnerabilidades [15].

Neste artigo foi adotada uma abordagem baseada no projeto conjunto de

software e *hardware* com o objetivo de mitigar ataques de canal lateral de tempo. O foco dado foi ao combate de ataques de tempo em algoritmos de curvas elípticas, especificamente a função X25519. Com tal objetivo, foi desenvolvida uma nova instrução para a arquitetura x86, permitindo uma implementação mais segura e eficiente do algoritmo.

Com a escolha desta arquitetura, dado o foco em dispositivos embarcados, a validação de segurança e desempenho da nova instrução foi realizada a partir da simulação de um sistema similar ao Intel Galileo: um processador com núcleo único, com execução de instruções em ordem, com 16KB de *cache* L1 e 256MB de memória RAM. É preciso notar que a escolha da arquitetura impacta diretamente nos resultados da introdução da nova instrução. Assim, é possível que ao aplicar tal abordagem em outras arquiteturas, que não a x86, não se chegue as mesmas conclusões que este artigo.

Apesar de ter sua construção inspirada por um algoritmo criptográfico específico, a nova instrução proposta foi definida de maneira genérica o bastante para ter seu uso facilmente generalizado, podendo ser utilizada em outros algoritmos e aplicações, até mesmo fora do contexto criptográfico ou de segurança.

2. Fundamentação Teórica

2.1. Ataques de Canal Lateral

São considerados ataques de canal lateral aqueles que se baseiam em dados físicos da máquina que executa o criptossistema para obter acesso às informações secretas. Dentre os dados físicos mais utilizados nesse tipo de ataque estão o tempo de execução, o consumo de energia e o espectro eletromagnético. Para a maior parte dos casos, o atacante precisa ter acesso físico ao dispositivo, mas a principal exceção são ataques de tempo, que podem ser executados através de uma conexão remota com o dispositivo por meio, por exemplo, da Internet. Existem ainda ataques que exigem acesso não-privilegiado ao sistema que executa o algoritmo criptográfico, como é o caso do ataque por colisão de *cache* ao AES [3].

Apesar do foco deste artigo estar em dispositivos embarcados, processadores de outras classes, como estações de trabalho e servidores, também estão suscetíveis a ataques de canal lateral. Ainda que suas informações físicas sejam influenciadas por outros fatores, como, por exemplo, outros algoritmos executando simultaneamente e a variação da velocidade de conexão, é possível filtrar tais ruídos monitorando a máquina por tempo suficiente [14].

De modo geral, mesmo que as informações fornecidas a respeito da informação secreta de algoritmos criptográficos pelos canais laterais não sejam suficientes para determinar com exatidão a chave, elas podem fornecer informação suficiente para reduzir o domínio da chave secreta, reduzindo o número de combinações possíveis e, desse modo, possibilitando que um ataque por força bruta seja bem-sucedido.

A abordagem mais adotada no combate aos ataques de canal lateral está em tornar as informações físicas emitidas pela máquina hospedeira do sistema criptográfico independentes das informações secretas do algoritmo.

2.2. Ataques de Tempo

Bastante estudados, a principal característica que coloca os ataques de tempo entre os

mais utilizados é a possibilidade de serem executados de modo completamente remoto [8], utilizando-se apenas das vias tradicionais de acesso ao dispositivo alvo, por exemplo: fazendo requisições que envolvam procedimentos criptográficos e medindo o tempo de resposta.

Em geral, a maior fonte de variação do tempo de execução dos algoritmos em função da entrada é ocasionada por mudanças no fluxo de execução do algoritmo. Felizmente, para algoritmos criptográficos, este tipo de vulnerabilidade já foi amplamente estudada [9] e diversas técnicas estão disponíveis para se evitar esta falha.

Outros fatores também ocasionam tal variação no tempo de execução e, por serem mais complexos e dependentes da arquitetura, não tem soluções bem definidas. Dentre eles, vale destacar as falhas na predição de desvios [10] e as variações no tempo de carregamento e armazenamento dos dados devido à hierarquia de memória das máquinas [3].

2.3. A Função X25519

Apresentada em 2006 por Bernstein [5], a X25519 é uma função para a implementação do algoritmo de Diffie-Hellman utilizando curvas elípticas. Ela foi apresentada por seu autor como um novo recorde de velocidade para implementação segura de tal algoritmo. A Figura 1 demonstra o fluxo de tal implementação.

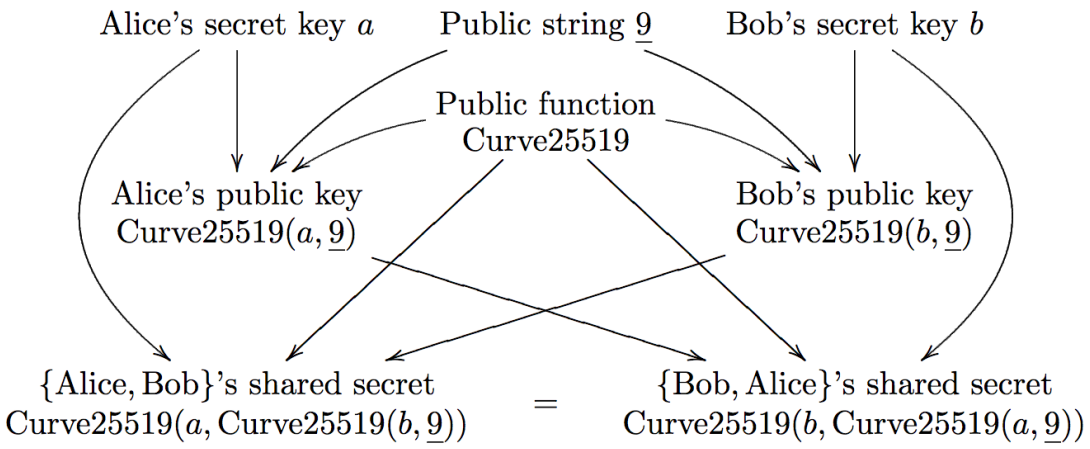


Figura 1. Fluxo de obtenção das chaves secretas compartilhadas no algoritmo de Diffie-Hellman utilizando a X25519. Retirado da referência [5].

A X25519 pode ser basicamente definida pela multiplicação de pontos na curva elíptica $E: y^2 = x^3 + 486662x^2 + x$, definida utilizando o primo $p = 2^{255} - 19$.

Além da velocidade, uma característica bastante atrativa da X25519 é que sua execução se dá em tempo constante, isto é, independente dos dados de entrada, o que implica, a princípio, na sua resistência contra ataques de canal lateral baseados no tempo.

Apesar de tal resistência, este artigo foca em fornecer um método alternativo, utilizando o suporte do *hardware*, para garantir esta proteção e ainda permitir certo ganho de desempenho. A principal razão que dá relevância a alternativa aqui proposta está na questão de que os artifícios utilizados pela implementação original para tornar a execução resistente a ataques de tempo acabam por criar novas vulnerabilidades e fontes

de vazamento de dados, como mostrado por Erick Nascimento, Lukasz Chmielewski, David Oswald e Peter Schwabe [15].

3. Metodologia

O ciclo básico de estudo adotado para este artigo foi:

1. Incluir no simulador de sistema MARSSx86 uma nova instrução, voltada a permitir a execução segura da X25519.
2. Instrumentar o simulador, de modo a permitir a obtenção de novas métricas que facilitem a detecção dos canais laterais e o entendimento dos dados já fornecidos.
3. Partindo da implementação original da X25519, presente no SUPERCOP [6], gerar 3 versões modificadas de tal implementação.
4. Executar as 4 versões da X25519 no simulador, analisar e tratar os dados de cada uma delas.

Cada uma das etapas descritas anteriormente será detalhada nas subseções seguintes.

3.1. Implementações da Função X25519

Foi tomada como base a implementação `ref10` do X25519 presente no SUPERCOP[6]. Nela, o ponto de vulnerabilidade escolhido para estudo foi a função *fe_cswap*, que tem por objetivo fazer ou não a troca de valores entre dois vetores a depender de um *bit* provido pela chave. Tal função, na implementação, é utilizada para substituir o condicional do algoritmo da cadeia de Montgomery [13]. Seu código é mostrado na Figura 2 de maneira simplificada, ocultando trechos não pertinentes à lógica do algoritmo, como declaração de variáveis e iteração sobre vetores.

```
void fe_cswap(fe f, fe g, unsigned int b)
{
  crypto_int32 f0 = f;
  crypto_int32 g0 = g;
  crypto_int32 x0 = f0 ^ g0;
  b = -b;
  x0 &= b;
  f = f0 ^ x0;
  g = g0 ^ x0;
}
```

Figura 2. Código, simplificado, da função fe_cswap do SUPERCOP [6].

As outras 3 versões implementadas a partir da original consistiram basicamente de alterações na maneira como esta função efetua a troca. Na versão original pode-se notar que a execução deve ocorrer em tempo constante, o que será demonstrado neste artigo. Nela, entretanto, é feita a utilização de variáveis auxiliares, o que ocasiona perda de desempenho, além de possível inserção de novas vulnerabilidades [15].

A primeira versão modificada é a versão com uma troca ingênua. Utilizando-se de um condicional simples para efetuar a troca, tal versão é absolutamente insegura e sua vulnerabilidade seria facilmente detectada por uma análise do fluxo de execução. Seu código é mostrado na Figura 3.


```
void ns_select(fe p, fe r, unsigned int b)
{
  crypto_int32 aux;
  if(b == 1){
    aux = p;
    p = r;
    r = aux;
  }
}
```

Figura 3. Código da função de troca insegura.

As versões anteriores foram implementadas utilizando a linguagem C, entretanto, uma vez que a presente pesquisa se dá especificamente para a arquitetura Intel x86, foram também implementadas duas trocas condicionais em *assembly*, de modo a otimizar este trecho de código e abrir espaço a introdução de uma nova instrução.

A segunda versão modificada utiliza-se da instrução *cmov* para efetuar a troca condicional. Seu código, simplificado, é mostrado na Figura 4. O sufixo *e* indica que a condição a ser verificada é a de igualdade.

```
cmpl $1, %edi
.irp i, 0,1,2,3,4,5,6,7,8,9
        mov 4*\i(%edx), %eax
        mov 4*\i(%ecx), %ebx
        mov %eax,-4(%ebp)
        cmove %ebx,%eax
        cmove -4(%ebp),%ebx
        mov %eax, 4*\i(%edx)
        mov %ebx, 4*\i(%ecx)
.endr
```

Figura 4. Código, simplificado, da função de troca utilizando a instrução cmove.

É preciso notar que em todas as implementações anteriores, inclusive na versão em *assembly*, foi necessário uma variável temporária para se efetuar a troca. No caso de uma eventual implementação em uma arquitetura de 64 *bits*, tal variável poderia ser alocada em um registrador. Porém, na arquitetura x86, por falta de registradores, ela acaba por ficar na memória, ocasionando perdas de desempenho.

Finalmente, a última versão implementada consiste na troca condicional através de uma nova instrução de troca condicional, a *cxch*. O funcionamento dessa nova instrução consiste basicamente em trocar os valores dos registradores alvos se a condição definida pelo sufixo for satisfeita pelas *flags* do processador. Tal verificação é similar à realizada pela instrução *cmov*. Este comportamento difere da já existente Compare-And-Swap (CAS) no ponto em que esta realiza a troca com base apenas na igualdade de seus próprios operandos.

O código da implementação com a nova instrução, simplificado, é mostrado na Figura 5. O sufixo *e* indica que a condição a ser verificada é a de igualdade.

Ao contrário das demais implementações, pode-se ver, além de uma redução no número de instruções, que não foi necessário o uso de qualquer variável auxiliar em memória. Desse modo tem-se um provável ganho de desempenho, que será também demonstrado neste artigo.


```
cmpl $1, %edi
.irp i, 0,1,2,3,4,5,6,7,8,9
        mov 4*\i(%edx), %eax
        mov 4*\i(%ecx), %ebx
        cxche %eax, %ebx
        mov %eax, 4*\i(%edx)
        mov %ebx, 4*\i(%ecx)
.endr
```

Figura 5. Código, simplificado, da função de troca utilizando uma nova instrução cxche.

3.2. O Simulador de Sistema MARSSx86

Para se efetuar a prototipação da nova instrução, assim como obter e comparar dados das execuções de cada uma das versões anteriores foi utilizado o simulador de sistema MARSSx86 [7].

Apresentado em 2011, o MARSSx86 é um simulador micro arquitetural e simulador de sistema baseado no QEMU [11] e no PTLSim [12]. Bastante utilizado em pesquisas na área de arquitetura, ele é dotado de alta acurácia em suas simulações e provê um grande número de dados que possibilitam uma análise profunda do comportamento dos algoritmos nele executados.

Além disso, a facilidade de acesso e modificação de seu código aliada à grande precisão que provê garantem a possibilidade de expandir seu conjunto de instruções e de inserir novos componentes de *hardware* com boa precisão em relação a um *hardware* real.

3.3. A prototipação de Instruções

Como citado nas subseções anteriores, a nova instrução inserida no MARSSx86 foi a troca condicional. O pseudocódigo que descreve sua implementação é mostrado na Figura 6. Nele, a operação *Seleciona* retorna o primeiro parâmetro a ela passado, caso o valor do terceiro parâmetro, a variável *Troca*, seja 0. Caso contrário, o segundo parâmetro é retornado. Já a operação *Verifica_Condicao* verifica se a condição definida por seu primeiro parâmetro é satisfeita pelas *flags* passadas por seu segundo parâmetro.

```
Cxch (A, B, Cond, Flags)
  bool Troca = Verifica_Condicao(cond, Flags)
  Temp = A
  A = Seleciona(Temp, B, Troca)
  B = Seleciona(B, Temp, Troca)
```

Figura 6. Pseudocódigo da instrução *cxch*.

Na implementação realizada no simulador, tal pseudocódigo gera 4 micro operações da arquitetura Intel. A primeira, chamada *OP_collcc*, é a responsável por possibilitar a operação *Verifica_Condicao*. Enquanto as outras 3, de mesmo tipo e chamadas *OP_sel*, realizam a operação *Seleciona*. Apesar de ser uma atribuição simples, a atribuição à variável *Temp* também é realizada pela micro operação *OP_sel*.

Foram utilizadas apenas as micro operações já existentes no processador e que

são utilizadas na implementação das demais instruções da arquitetura. Assim, o tempo de execução dessa nova instrução comparado às demais é bastante realista, tendo uma proporção bem próxima a que ocorreria num *hardware* real. As variáveis temporárias *Troca* e *Temp* são registradores temporários internos ao conjunto de instruções.

O simulador é capaz de simular a execução as micro operações em tempo constante, porém, no caso de uma implementação em *hardware* real seria necessário que essa propriedade fosse garantida para que se obtivesse, de fato, uma execução resistente a ataques de canal lateral por tempo.

3.4. O processo de Simulação

Para possibilitar uma melhor análise dos dados foram medidas estatísticas não só para a função criptográfica toda, mas também apenas para os pontos de interesse na execução, no caso, a função que efetua a troca. Apesar das estatísticas se referirem ao código isolado, o sistema operacional é capaz de influir de diversas formas na execução, assim, para se garantir a confiabilidade dos resultados cada simulação foi executada 10 vezes, a fim de se obter um desvio padrão de cada dado.

No total, o processo de simulação fornece cerca de 300 dados sobre a execução realizada. Dentre eles estatísticas sobre o número de ciclos, predições de desvio, acessos à *cache* e à memória, tipos de instruções, tipos de micro operações, dentre outras. Para apresentação e análise dos resultados na seção seguinte, serão apresentados apenas os dados mais relevantes.

4. Resultados Experimentais

O primeiro objetivo do processo de simulação foi o de demonstrar a capacidade da ferramenta de detectar o vazamento de informações por canais laterais. Isso foi feito através da exposição da vulnerabilidade da versão insegura a ataques de tempo. Neste processo também foi demonstrado a segurança das demais versões. Para alcançar tal objetivo foram executados dois testes em ambiente simulado.

A função de troca, descrita na subseção 3.1, recebe como parâmetro uma variável binária *b* e a utiliza para definir se efetuará ou não a troca. Tal variável é definida pela operação de ou exclusivo (*xor*) entre dois *bits* sequenciais da chave. Tais *bits* são percorridos sequencialmente do mais significativo para o menos significativo. Assim, temos que quanto mais transições de *bits* a chave tiver, mais trocas o algoritmo fará.

Com isso, o primeiro teste consistiu em executar as 4 versões para 3 chaves privadas diferentes, com o número de transições de *bits* crescente em cada uma delas. A chave pública permaneceu inalterada. São elas:

- K1: Uma chave composta por 128 *bits* zeros seguidos de 128 *bits* uns.
- KX: Uma chave fixa gerada aleatoriamente. Possui 120 transições aleatoriamente distribuídas.
- K256: Uma chave comporta por 128 pares de *bits* “01”.

O foco está nos ataques de tempo e é possível detectar variação diretamente através do número total de ciclos executados. O gráfico da Figura 7 mostra o número total de ciclos para uma execução completa do algoritmo, enquanto o gráfico da Figura

8 mostra o número de ciclos gastos apenas pelas funções de troca, também durante uma execução completa do algoritmo.

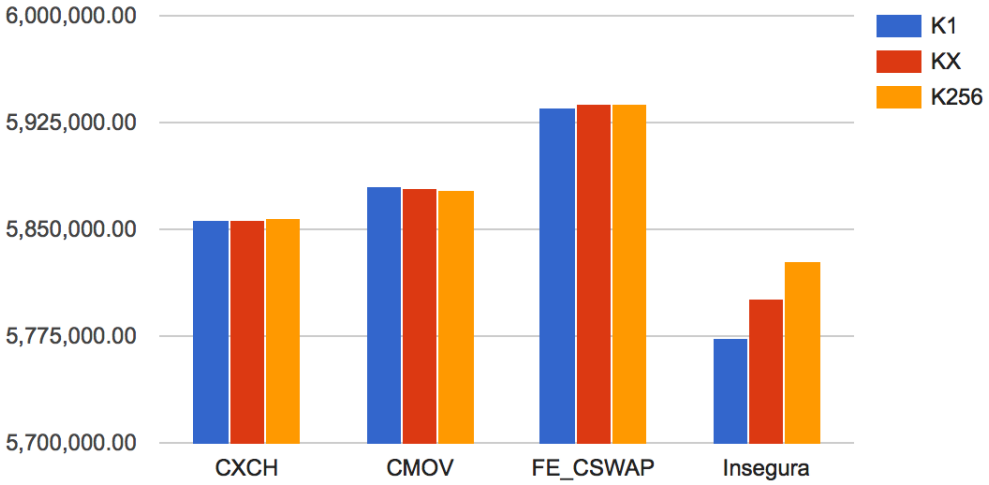


Figura 7. Gráfico do número de ciclos da execução do algoritmo completo.

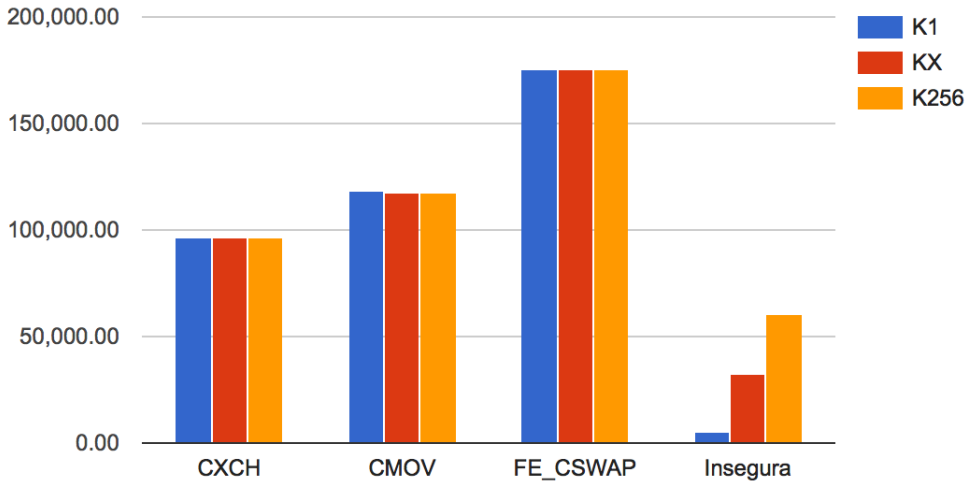


Figura 8. Gráfico do número de ciclos da execução da função de troca.

Analisando os dados do gráfico da Figura 7, é possível notar com clareza a vulnerabilidade da versão insegura. O tempo de execução dela varia diretamente de acordo com o número de transições da chave. Tal fato decorre de suas grandes variações no fluxo de execução da função de troca, o que garante a ela um desempenho bastante superior às versões seguras, mas que permitiria a um atacante adquirir facilmente a informação sobre o número de transições da chave. As demais versões, ao menos em relação a ataques baseados em tempo, neste teste, se mostram resistentes. O gráfico da Figura 8 permite confirmar a fonte da vulnerabilidade da versão insegura.

O segundo teste realizado foi a execução das 4 versões para duas chaves privadas com o mesmo número de transições, porém em uma delas, kB, as transições estão dispostas de maneira padronizada, facilmente predizíveis, enquanto na outra, kA, estão dispostas aleatoriamente. O gráfico da Figura 9 exibe a razão entre o número de ciclos das execuções das funções de troca com cada uma das chaves.

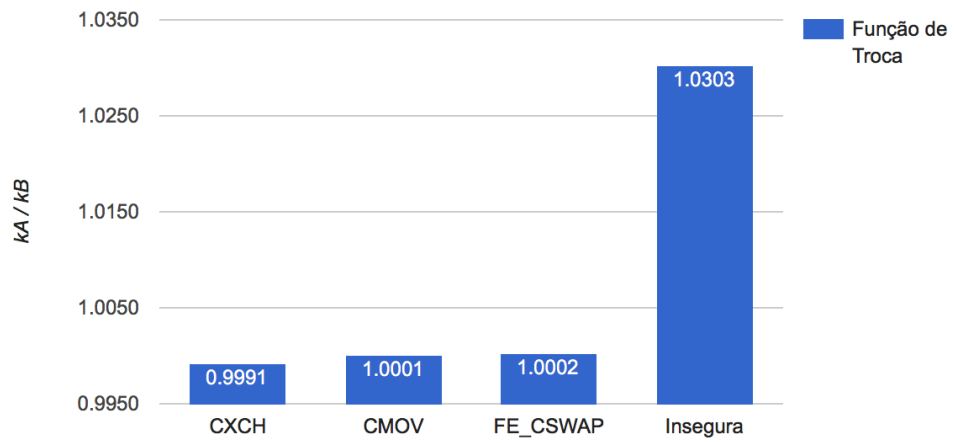


Figura 9. Razão entre o número de ciclos das execuções da funções de troca com as chaves kA e kB.

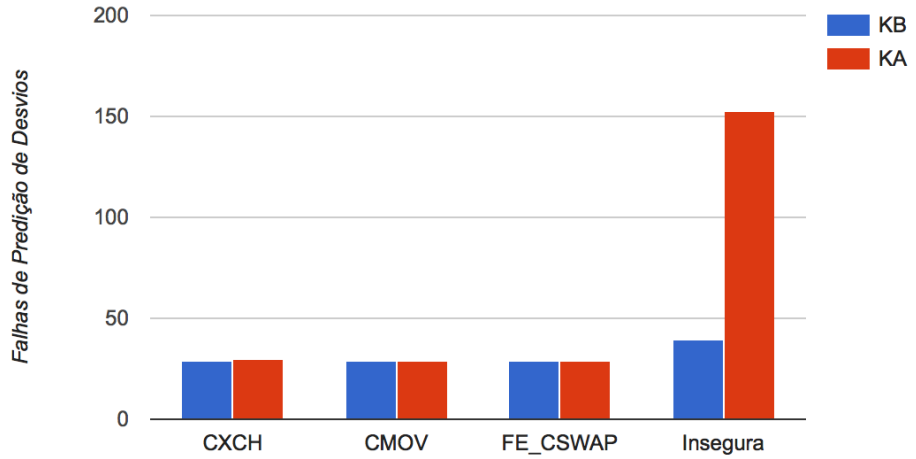


Figura 10. Gráfico do número de falhas de predição de desvios da execução do algoritmo completo.

O gráfico da Figura 9 permite notar que enquanto as versões seguras têm variação na ordem de 0.01% no número de ciclos, decorrente da influência do sistema operacional, a versão insegura tem uma variação de mais de 3% no número de ciclos, mesmo com as duas chaves tendo o mesmo número de transições. Esta variação se dá devido às falhas de predição de desvio no condicional da função de troca. Quando as transições seguem um padrão, o preditor consegue determiná-las corretamente e o tempo de execução cai. Já quando elas são aleatórias, ele passa a errar mais e, consequentemente, o tempo de execução aumenta. O gráfico da Figura 10 mostra o número de falhas de predição de desvio ocorridas durante toda a execução de cada uma das versões, para as duas chaves kA e kB. Seus resultados corroboram a explicação anterior.

Após este último teste, tem-se que a versão insegura revela, sob um ataque de tempo, não só a informação referente ao número de transições da chave, mas também à disposição de tais transições. Assim, demonstra-se a razoável capacidade de detecção de vazamento de informações por canais laterais da ferramenta utilizada, bem como o nível de segurança das versões implementadas.

O segundo objetivo da simulação está em demonstrar o ganho de desempenho da versão com a nova instrução *cxch* em relação às demais versões seguras. Considerando apenas as funções de troca, em relação à *fe_cswap*, presente na implementação original, o ganho de desempenho provido pelo uso da nova instrução chega a 45%, como é mostrado no gráfico da Figura 8. Enquanto para o algoritmo todo, ele fica em torno de 1.4%, também em relação à versão original. O gráfico da Figura 11 ajuda a entender o ganho alcançado. Ele representa dados de acesso à *cache*, número de instruções e micro operações executadas por cada uma das versões das funções de troca. Através do gráfico, é possível notar, por exemplo, que a versão original faz o dobro de acesso à *cache* de dados do que a versão com a nova instrução. Para facilitar a visualização, os dados são normalizados com relação aos dados medidos na versão com a instrução *cxch*.

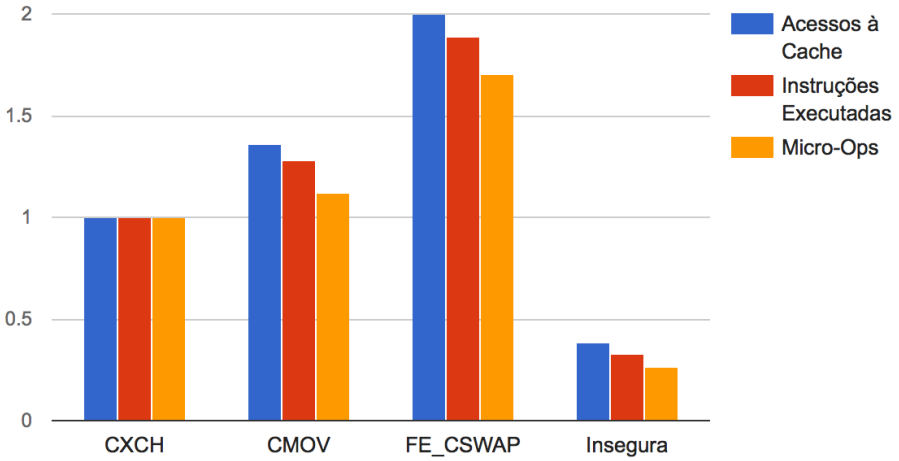


Figura 11. Gráfico de acessos à *cache*, número de instruções e micro operações executadas pelas funções de troca normalizados com relação aos dados medidos na versão com a instrução CXCH.

5. Conclusões

Neste artigo foram apresentados os resultados da prototipação de uma nova instrução de troca condicional para a arquitetura x86 e sua utilização na implementação resistente a ataques de canal lateral do algoritmo de Diffie-Hellman com a X25519. Através das simulações realizadas, foi possível demonstrar a capacidade de detecção de vazamento de informações por canais laterais do método utilizado, assim como demonstrar a segurança e comparar, em termos de eficiência, diferentes implementações da X25519.

A proposta da nova instrução de troca condicional se mostra bastante razoável, uma vez que tal instrução possibilitou um ganho de desempenho de 45% na função de troca da X25519, em relação a implementação original. Além de ajudar a evitar, por fazer todo o procedimento de troca internamente, que se encontrem outras vulnerabilidades naquele ponto. É preciso ressaltar, entretanto, que tal abordagem tem como desvantagens a necessidade de se fazer modificações no *hardware* e o fato de ter sido desenvolvida para uma arquitetura específica.

Pretende-se, como trabalho futuro, a replicação do método de estudo para outros algoritmos criptográficos, resultando na criação de novas instruções e componentes de *hardware*. Pretende-se também validar todo procedimento com a prototipação das instruções e execução dos algoritmos em *hardware* real, através da utilização de

FPGAs. Outros tipos de vazamento de dados por canais laterais, como, por exemplo, o consumo de energia, também serão estudados.

6. Agradecimentos

À Intel Semicondutores do Brasil Ltda e à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processo nº 2014/50704-7, pelo financiamento e apoio à pesquisa.

Referências

[1] Kocher, Paul C. "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems." *Annual International Cryptology Conference*. Springer Berlin Heidelberg, 1996.

[2] Bernstein, Daniel J. "Cache-timing attacks on AES." (2005).

[3] Tromer, Eran, Dag Arne Osvik, and Adi Shamir. "Efficient cache attacks on AES, and countermeasures." *Journal of Cryptology* 23.1 (2010): 37-71.

[4] Brumley, David, and Dan Boneh. "Remote timing attacks are practical." *Computer Networks* 48.5 (2005): 701-716.

[5] Bernstein, Daniel J. "Curve25519: new Diffie-Hellman speed records." *International Workshop on Public Key Cryptography*. Springer Berlin Heidelberg, 2006.

[6] Bernstein, Daniel J. "Supercop: System for unified performance evaluation related to cryptographic operations and primitives." (2009).

[7] Patel, Avadh, Furat Afram, and Kanad Ghose. "Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors." *1st International Qemu Users' Forum*. 2011.

[8] Brumley, David, and Dan Boneh. "Remote timing attacks are practical." *Computer Networks* 48.5 (2005): 701-716.

[9] Molnar, David, et al. "The program counter security model: Automatic detection and removal of control-flow side channel attacks." *International Conference on Information Security and Cryptology*. Springer Berlin Heidelberg, 2005.

[10] Acıiçmez, Onur, Çetin Kaya Koç, and Jean-Pierre Seifert. "Predicting secret keys via branch prediction." *Cryptographers' Track at the RSA Conference*. Springer Berlin Heidelberg, 2007.

[11] Bellard, Fabrice. "QEMU, a fast and portable dynamic translator." *USENIX Annual Technical Conference, FREENIX Track*. 2005.

[12] Yourst, Matt T. "PTLsim: A cycle accurate full system x86-64 microarchitectural simulator." *2007 IEEE International Symposium on Performance Analysis of Systems & Software*. IEEE, 2007.

[13] Montgomery, Peter L. "Speeding the Pollard and elliptic curve methods of factorization." *Mathematics of computation* 48.177 (1987): 243-264.

[14] Le, Thanh-Ha, et al. "Noise reduction in side channel attack using fourth-order cumulant." *IEEE Transactions on Information Forensics and Security* 2.4 (2007): 710-720.

[15] Erick Nascimento, Lukasz Chmielewski, David Oswald and Peter Schwabe. "*Attacking embedded ECC implementations through cmov side channels.*" *23rd Conference on Selected Areas in Cryptography* (SAC 2016)

Implementação eficiente e segura de cifras de fluxo em software

Daniel P. Cunha[1], **Rafael J. Cruz**[1], **Diego F. Aranha**[1]

[1]Instituto de Computação (IC) – Universidade Estadual de Campinas (Unicamp)

{dcunha,raju,dfaranha}@lasca.ic.unicamp.br

Abstract. *The number of devices able to connect to the Internet reaches new levels day after day, raising challenges about protecting the information they are capable of transmitting. Because of their smaller footprint, stream ciphers represent an ideal cryptographic primitive for confidentiality of in-transit data. This work presents efficient and constant time software implementations of two stream ciphers targeted at ARM processors. Our implementations of ChaCha20 and SOSEMANUK provide performance gains up to 30% and 43% against reference code, respectively, depending on the testing platform used.*

Resumo. *O crescente número de dispositivos que podem se conectar à Internet atinge novos patamares e traz consigo desafios ainda maiores quanto à segurança das informações transmitidas. Cifras de fluxo são uma primitiva criptográfica ideal para fornecer confidencialidade nesse cenário. Este trabalho apresenta implementações eficientes em software de duas cifras de fluxo na plataforma ARM, protegidas contra ataques de canal lateral de tempo. A implementações do ChaCha20 e SOSEMANUK apresentaram ganhos de desempenho de até 30% e 43% em relação ao código de referência, respectivamente, a depender da plataforma de testes utilizada.*

1. Introdução

Avanços tecnológicos permitem a progressiva inclusão de microprocessadores em praticamente qualquer objeto. Tal fato abre caminho para um cenário que representa uma nova era das comunicações, à medida que o número de dispositivos com capacidade de transmissão de dados aumenta rapidamente, a exemplo de eletrodomésticos e objetos de uso pessoal. Em linhas gerais, o cenário, batizado de "Internet das Coisas" ou *Internet of Things* – IoT, traz consigo a crescente necessidade de obtenção de ferramentas capazes de funcionar neste novo contexto. Desta forma, a proteção das informações pessoais coletadas por utensílios do cotidiano demanda a implementação eficiente de primitivas criptográficas. Alcançar esse objetivo pode ser desafiador, dado que a tendência de redução da recursos consumidos pelos dispositivos implica poder computacional restrito dos mesmos, especialmente aqueles alimentados por baterias.

O surgimento da criptografia de chave pública fomentou ativamente as pesquisas em implementação eficiente de primitivas criptográficas, onde a princípio o desempenho era essencialmente avaliado em termos do tempo de execução. O novo

paradigma de IoT reforça e expande requisitos de desempenho para incluir consumo de energia e motiva a necessidade de algoritmos leves de criptografia.

Neste trabalho foram propostas otimizações das implementações de referência de dois cifradores de fluxo: o ChaCha [Bernstein 2005a], que é baseado no algoritmo do Salsa20 [Bernstein 2008], e o SOSEMANUK [Berbain et al. 2008] que faz uso das funções de transformação linear baseadas no SERPENT [Biham et al. 1998] e princípios de projeto do cifrador SNOW 2.0 [Ekdahl and Johansson 2002]. Tanto o ChaCha quanto o SOSEMANUK foram finalistas da competição internacional eSTREAM, que buscou no Perfil 1 cifradores de fluxo com características mais apropriadas para aplicações em software, com requisitos de alta vazão de dados. A implementação do ChaCha é intrinsecamente protegida contra ataque de canal lateral de tempo, mas o SOSEMANUK possui duas variantes: uma em tempo variável e outra em tempo de execução constante. A existência de 2 variantes para o SOSEMANUK advém de características do próprio cifrador, que faz uso de uma tabela de 256 posições, conforme especificação [Berbain et al. 2008], que pode introduzir variações na latência de resposta da hierarquia de memória úteis para um adversário capaz de montar ataques de canal lateral de tempo.

2. Notação e terminologia

A Tabela 1 estabelece o significado dos símbolos utilizados neste documento.

Tabela 1. Símbolos e seu respectivo significado neste documento.

Símbolo	Significado
$\lll n$	Rotação de n bits à esquerda
$\oplus$ ou $\wedge$	`XOR` bit a bit
$+$ ou $\boxplus$	Adição módulo 32 bits
$\ll n$	Deslocamento de n bits à esquerda
$\gg n$	Deslocamento de n bits à direita

Dada a escolha de plataforma alvo, o termo "palavra de memória", ou simplesmente "palavra" se refere a um elemento de 32 bits. Sempre que mencionado implementação de referência deve-se entender que a implementação em questão refere-se àquela encaminhada como última submissão à competição eSTREAM. No caso do ChaCha20, ver Salsa20 [eSTREAM 2004].

3. O algoritmo ChaCha20

O ChaCha20 [Bernstein 2005a] é uma variante da cifra de fluxo Salsa20, e esta por sua vez é construída sobre uma função pseudo-aleatória definida pelo autor como uma função de *hash* (ou confusão) baseada em operações ARX (*Add-Rotate-XOR*), com adição realizada em 32 bits. Mais especificamente, a definição do Chacha20 segue todas as especificações do Salsa20 a menos da função quarto-de-rodada, ou "*quarter-round*", e do posicionamento de cada elemento da matriz de estado, que consiste em 16 palavras de 4 bytes. Na matriz, metade das palavras representam a chave secreta, 4 palavras representam constantes e as últimas 4 são palavras de entrada sob controle do usuário do cifrador, representando o *nonce* ou vetor de

inicialização. O número 20 que sucede o nome do algoritmo representa o número de iterações ou rodadas da função *hash*.

3.1. A função *quarter-round*

A função *quarter-round* (QR) é definida no Salsa20 considerando uma entrada constituída por uma sequência de 4 palavras de 32 bits. Para cada 4 ciclos finalizados desta função completa-se o equivalente a uma rodada. A *quarter-round* deve processar a entrada segundo a sequência de operações de 32 bits como no Algoritmo 1 apresentado a seguir para o Salsa20 e reprojetada para o ChaCha20 como no Algoritmo 2. Ambas as funções apresentam o mesmo número de operações de soma, XOR bit-a-bit e rotações.

Algoritmo 1 Função QR para o Salsa20

1: $b \leftarrow b \oplus (a + d) \lll 7$
2: $c \leftarrow c \oplus (b + a) \lll 9$
3: $d \leftarrow d \oplus (c + b) \lll 13$
4: $a \leftarrow a \oplus (d + c) \lll 18$

Algoritmo 2 Função QR para o Chacha20

1: $a \leftarrow a + b$, $d \leftarrow (d \oplus a) \lll 16$
2: $c \leftarrow c + d$, $b \leftarrow (b \oplus c) \lll 12$
3: $a \leftarrow a + b$, $d \leftarrow (d \oplus a) \lll 8$
4: $c \leftarrow c + d$, $b \leftarrow (b \oplus c) \lll 7$

A essência da cifração dos dados pode ser resumida em 3 funções: a função de *hash*, *quarter-round*, a função *row-round* e a função *column-round*. As duas últimas buscam selecionar os 16 elementos da matriz de estados para operação. A operação das duas funções ocorre de modo alternado entre linhas e colunas da matriz, de forma que a função *column-round* é composta de quatro chamadas à função *quarter-round*, cujos 4 parâmetros passados nesta chamada correspondem à posições na matriz de estados.

Analogamente, a função *row-round* realiza 4 chamadas à função *quarter-round*, onde os parâmetros agora correspondem a parâmetros selecionados um de cada linha da matriz de estados de modo que em cada chamada um único elemento de cada linha é utilizado. A aplicação de uma função *column-round* seguida de uma *row-round* é batizada de um *double-round*, que equivale a 2 rodadas do total de 20 rodadas propostas (uma rodada por função chamada).

Cabe mencionar que existem variações da implementação original, Salsa20/12, Salsa20/8 que correspondem a algoritmos que implementam, respectivamente, 12 e 8 rodadas da função, de forma que tais algoritmos representam duas dentre outras variantes com menor número de rodadas relativas ao ChaCha20. O ChaCha20/8 é também conhecido como ChaCha8, o ChaCha20/12 como ChaCha12 e assim por diante.

3.2. A matriz de estados do ChaCha20

A matriz do ChaCha20 possui 16 palavras de 32 bits ou um total de 64 bytes, dispostas conforme a Figura 1, que apresenta as mudanças propostas na matriz do Chacha20 em relação à matriz do Salsa20.

Após executadas as 20 rodadas da função de *hash* sobre a matriz de estados do ChaCha20 tais como descritas anteriormente, adiciona-se o resultado à uma cópia inalterada da matriz original, byte a byte, para compor os 64 bytes de

$$\begin{pmatrix} constant & constant & constant & constant \\ key & key & key & key \\ key & key & key & key \\ input & input & input & input \end{pmatrix} \qquad \begin{pmatrix} constant & key & key & key \\ key & constant & input & input \\ input & input & constant & key \\ key & key & key & constant \end{pmatrix}$$

(a) Matriz de estados ChaCha20 **(b) Matriz de estados Salsa20**

Figura 1. Matriz de estados dos algoritmos ChaCha20 e Salsa20

cadeia de chave, que por sua vez será combinada através de um `XOR` sobre a mensagem original para produzir 64 bytes da mensagem cifrada. A especificação do Salsa20 [Bernstein 2005b] requer que os bytes de interesse sejam processados no formato *little-endian*. Dessa forma ao definir $Z_i = (z_0, z_1, \cdots, z_{15})$ como as 16 palavras da cadeia de chave, tem-se que o texto claro, (M_j), é transformado em texto cifrado, (C_j), pelo Salsa20 após conversão de representação por $C_j = M_j \oplus Z_j$, com $Z_j = (Y_i + X_i)$, onde $X_i = (x_0, x_1, \cdots, x_{15})$ são os elementos da matriz de estados original, da qual é guardada uma cópia antes da execução da função de *hash* e $Y_i = (y_0, y_1, \cdots, y_{15})$ representa a matriz de estados após as 20 rodadas da função de *hash*.

4. Descrição da implementação do ChaCha20

A implementação proposta foi desenvolvida em linguagem C, tal como a implementação de referência, fornecendo portabilidade e eficiência. A proposta de implementação leva em consideração a legibilidade do código, bem como a busca por operações ou combinações das mesmas que, mantendo as propriedades do cifrador, resultem em tempo de execução reduzido.

Aliado ao foco em uma implementação mais eficiente buscou-se uma implementação regular do ponto de vista do número de operações realizadas em cada um dos possíveis casos de execução, isto é, reescrever execuções condicionais de modo a torná-las equivalentes no que tange ao número de instruções executadas ao tomar ou não um determinado desvio condicional.

Serão apresentadas a princípio as estratégias adotadas que contribuíram com o ganho de desempenho do algoritmo, e em seguida as modificações que tornaram o código protegido contra ataques de canal lateral, isto é, que permitiram a execução do mesmo ocorrer em tempo constante para mensagens do mesmo tamanho.

O primeiro ganho obtido decorreu da observação de que as arquiteturas dos sistemas atuais é em sua grande maioria construída sobre arquiteturas nativamente *little-endian*, fato este que tende a se consolidar dada a crescente difusão de processadores desenvolvidos sobre as arquiteturas ARM e x86. Assim é possível evitar funções de conversão das entradas e resultado.

A Figura 2 mostra a aplicação das 20 rodadas da função de hash tanto na implementação proposta sob a nova estratégia quanto na implementação de referência. Vale ressaltar que na implementação de referência, além de computar a conversão, a função especificada(`U32TO8_LITTLE`) converte elementos de 4 bytes em 4 elementos de um byte.

Adicionalmente, a segunda estratégia adotada na implementação proposta


```
static inline void salsa20_hash_chacha(
                        uint8_t vec[64],
                        uint32_t matrix[16]) {
  int i;
  uint32_t x[16] = { 0 };
  uint32_t sum;
  uint32_t *vec_32 = (uint32_t *)vec;

  for (i = 0; i < 16; i++)
    x[i] = matrix[i];

  for (i = 0; i < 10; i++)
    DOUBLE_ROUND(x);

  for (i = 0; i < 16; i++) {
    sum = x[i] + matrix[i];
    vec_32[i] = sum;
  }
}
```

(a) Chacha20 *hash* otimizado

```
static void salsa20_hash_chacha(
                uint8_t output[64],
                const uint32_t input[16]){
                uint32_t x[16];
  int i;

  for (i = 0;i < 16;++i)
    x[i] = input[i];

  for (i = 20;i > 0;i -= 2) {
    QUARTERROUND( 0, 4, 8,12)
    QUARTERROUND( 1, 5, 9,13)
    QUARTERROUND( 2, 6,10,14)
    QUARTERROUND( 3, 7,11,15)
    QUARTERROUND( 0, 5,10,15)
    QUARTERROUND( 1, 6,11,12)
    QUARTERROUND( 2, 7, 8,13)
    QUARTERROUND( 3, 4, 9,14)
  }

  for (i = 0;i < 16;++i)
    x[i] = PLUS(x[i],input[i]);
  for (i = 0;i < 16;++i)
    U32TO8_LITTLE(output + 4 * i,x[i]);
}
```

(b) Chacha20 *hash* referência

Figura 2. Comparação da implementação sob a nova estratégia com a implementação de referência.

refere-se ao modo como são realizadas as conversões das palavras (4 bytes) em elementos de um único byte. Ao contrário da implementação de referência que, a partir de uma palavra, executa deslocamentos, bitwise `AND`'s e `OR`'s para extração de cada um dos bytes de interesse, é proposta a inicialização de apontador de dimensão 32 bits (8 bits) por meio da conversão em um elemento de 8 bits (32 bits) de interesse caso a conversão seja de elementos de 1 byte (4 bytes) para elemento(s) de 4 (1) byte(s).

A segunda estratégia é apresentada na Figura 2, onde apesar de existir um consumo relativo maior de memória pela inicialização de 2 variáveis adicionais (`sum` e `*vec_32`), há o compromisso com o número de operações que é significativamente menor, visto que o parâmetro `vec` da função de *hash* pode, após aplicada a referida estratégia da inicialização dos auxiliares, ser visto como um vetor de 4 bytes, aproveitando melhor os acessos à memória realizados.

Ainda que esta estratégia tenha beneficiado as funções de expansão de chaves e inicialização, é na função de *hash* do algoritmo que se verifica o maior aproveitamento do uso destas estratégias, visto que as funções de inicialização podem ser utilizadas uma única vez para cada 2^{70} bytes a cifrar (período do cifrador), enquanto a função de *hash* pode ser utilizada sucessivamente a cada múltiplo de 64 bytes a cifrar, ainda sob o mesmo estado de inicialização caso respeitado o período mencionado. Por este motivo, as estratégias apresentadas, embora utilizadas nas funções de inicialização, não foram exibidas em termos de código implementado nas funções de expansão.

5. O algoritmo SOSEMANUK

O SOSEMANUK é uma cifra de fluxo construída com base na estrutura geral do SNOW 2.0 [Ekdahl and Johansson 2002], baseado em um registrador de deslocamento de retroalimentação linear, ou LFSR – *Linear Feedback Shift Register* – e em uma máquina de estados finitos ou FSM – *Finite State Machine*. Adicionalmente, o SOSEMANUK agrega a transformação linear do cifrador de bloco SERPENT [Biham et al. 1998], candidato do concurso AES, para a inicialização do estado interno deste cifrador. A expansão de chaves também se baseia no SERPENT. Diferentemente do ChaCha que aceita chaves de 128 ou de 256 bits, o SOSEMANUK aceita chaves de tamanhos de 1 a 256 bits, porém chaves maiores do que 128 bits são recomendadas por razões de segurança. Para a expansão da chave, o SOSEMANUK faz uso de 8 S-Boxes – *Substitution Boxes* ou caixas de substituição – herdadas do SERPENT. O objetivo do SOSEMANUK é superar o SNOW 2.0 em termos de segurança e eficiência.

5.1. A inicialização do estado interno: populando o LFSR e a FSM

A inicialização do SOSEMANUK consiste em expandir a chave secreta. Antes da expansão, é realizado o preenchimento da chave para que esta tenha 256 bits, de modo a completar chaves menores do que 256 bits com um 1 no primeiro bit não utilizado da chave fornecida e os demais bits com 0 até atingir os 256 restantes. Em seguida, a expansão é realizada de modo a gerar 100 elementos de 4 bytes, ou 100 sub-chaves, pela interpolação em sequência especificada por uso parcial da sequência de caixas de substituição definidas no SERPENT [Biham et al. 1998].

A população inicial da FSM e do LFSR compõe a etapa seguinte da inicialização do algoritmo. A inicialização transforma as sub-chaves geradas anteriormente por meio do uso das S-Boxes associadas à subsequente transformação linear do SERPENT em todas as sub-chaves. As S-Boxes operam a chave expandida no modo *bitsliced*. Vale mencionar que em pontos específicos desta inicialização é que são populadas individualmente as 10 posições da LFSR e os 2 registradores da FSM, segundo especificação do cifrador [Berbain et al. 2008].

Este trabalho dará maior ênfase sobre a definição do LFSR, pelo fato de trazer contribuições significativas nessa estrutura do cifrador a fim de proteger o algoritmo contra ataques de canal lateral. A resistência é aprimorada pela eliminação do uso das tabelas internas do SOSEMANUK, ao calcular seus coeficientes durante a execução e apenas quando forem necessários. Além disso, outra contribuição nesse sentido foi o uso de estratégias na implementação estrutural da FSM no SOSEMANUK, para que instruções sujeitas a desvios condicionais sejam evitadas ou reformuladas para atingir execução em tempo constante.

5.2. A construção do LFSR: os polinômios base

A estrutura geral do LFSR é representada na Figura 3, onde o novo elemento do registrador de deslocamento, s_{t+9}, é gerado a partir de operações sobre α e α^{-1} e o valor que é deslocado para fora é utilizado para calcular com um `XOR` a saída do algoritmo.

Antes de mencionar o papel do coeficiente α e seu inverso, é necessário mencionar que os valores destes coeficientes são obtidos por operações envolvendo o

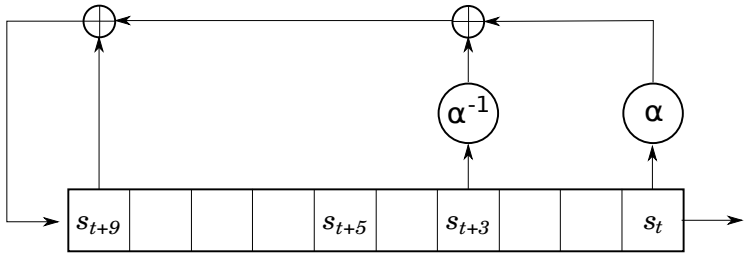


Figura 3. Esquema do LFSR

polinômio primitivo $Q(X) = X^8 + X^7 + X^5 + X^3 + 1$, que define um corpo binário de elementos em $GF(2)$ (os coeficientes de cada termo de $Q(X)$ são elementos iguais a 1 ou 0) cujas raízes, $\beta_i = (\beta^7, \beta^6, \cdots, \beta, 1)$, produzem os elementos inversíveis do corpo de extensão, representado por um segundo polinômio, $P(X) = X^4 + \beta^{23}X^3 + \beta^2 45X^2 + \beta^{48}X + \beta^{239}$. Esse último é composto por elementos em $GF(2^8)$ (coeficientes de $P(X)$ são bytes de valor entre 0 e 255) de modo que as raízes de $P(X)$ são representadas pela base $(\alpha_3, \alpha_2, \alpha, 1)$, e cada elemento representado por $P(X)$ possui na realidade um correspondente de 32 bits na representação de números inteiros, isto é, cada elemento representado por $P(X)$ é um elemento de 32 bits que irá compor o próximo elemento de s_{t+9} no LFSR.

A definição destes 2 corpos é mencionada com mais detalhes na especificação do algoritmo [Berbain et al. 2008], onde é definido que uma multiplicação por β corresponde a um deslocamento para esquerda de um bit na representação de inteiros, seguido de um `XOR` com uma máscara fixa se o bit mais significativo eliminado pelo deslocamento for 1. Vale ressaltar que a adição de elementos em $GF(2^{32})$ corresponde a um `XOR`. A multiplicação de um elemento em $GF(2^{32})$ por α equivale a um deslocamento à esquerda de 8 bits seguido de um `XOR` com uma máscara de 32 bits que depende apenas do byte mais significativo do elemento em $GF(2^{32})$. Uma divisão por α (multiplicação por α^{-1}) representa um deslocamento à direita de 8 bits seguido de um `XOR` com uma máscara que depende apenas do byte menos significativo de s_{t+3} em LFSR.

5.3. A atualização da FSM durante o processo de cifração

A FSM é atualizada, após ter sido inicializada, por meio de operações sobre $R1_{t-1}$ e $R2_{t-1}$, onde $t = 0$ representa o momento da inicialização, de forma que t é sempre maior ou igual a 1. O cálculo de $R1_t$ é como segue: $R1_t = (R2_{t-1} + mux(lsb(R1_{t-1}), s_{t+1}, s_{t+1} \oplus s_{t+8}))$ mod 2^{32}, sendo $lsb(x)$ o bit menos significativo de x e $mux(w, y, z)$ igual a y caso w seja igual a 0 ou igual a z caso w seja igual a 1. O cálculo de $R2_t$ é dado por $(M \cdot R1_{t-1} \mod 2^{32}) <<< 7$, onde M é constante e igual a `0x54655307`. Finalmente, a saída da FSM é dada por $f_t = (s_{t+9} + R1_t \mod 2^{32}) \oplus R2_t$.

6. Descrição da implementação do SOSEMANUK

A implementação protegida consiste, além da implementação em tempo constante de modo análogo ao descrito ao final da seção 4, no cálculo dos coeficientes das tabelas `mulAlpha` e `divAlpha` sob demanda, ou seja, conforme a execução do algoritmo necessitar destes coeficientes. Estas tabelas representam as máscaras fixas

mencionadas ao final da seção 5.2, que são operadas por um `XOR` com o resultado do deslocamento de 8 bytes para a direita no elemento s_{t+3} do LFSR ou de 8 bytes para a esquerda no elemento s_t de LFSR.

A versão desprotegida contra ataques de canal lateral apresenta um compromisso com a diminuição no tempo de execução a custo de maior uso de memória interna associado ao estado do cifrador, visto que duas tabelas de 256 posições de 32 bits contendo as máscaras podem ser armazenadas localmente no algoritmo ou calculadas sob demanda, para as implementações desprotegidas ou protegidas, respectivamente. O cálculo das máscaras ocasiona, naturalmente, aumento bastante significativo no tempo de execução geral.

A Figura 4 apresenta o corpo de 2 das 3 funções utilizadas para o cálculo das máscaras (elementos das tabelas `mulAlpha` e `divAlpha`, que correspondem à multiplicação e divisão por alfa, respectivamente). A função `mul_galois8` opera no corpo binário, enquanto a `(mul/div)alpha_galois32` opera no corpo $GF(2^8)$ e concatena os resultados para obter elementos em $GF(2^{32})$, considerando o *carry*. A função `divAlpha_galois32`, não exibida na imagem, é análoga à função apresentada na função da Figura 4b, a menos das constantes passadas como primeiro parâmetro para a função `mul_galois8` e a menos da constante de verificação do byte menos significativo ao invés do mais significativo, que por sua vez implementa um simples `AND` lógico da constante `0xFF` com o valor b. Ainda na `divAlpha_galois32`, as constantes são: `0xCD`, `0x40`, `0x0F` e`0x18`, nessa ordem.

```
uint8_t mul_galois8(
            uint8_t a,
            uint8_t b,
            uint8_t *of) {
  uint8_t p = 0;
  int32_t i;

  for (i = 0; i < 8; i++) {
    // multiply  b_(i) * a
    p = p ^ ((b & 0x01) * a);
    /keeps every overflow bit before a *= 2
    *of |= ((a & 0x80) << (7 - i));
    // multiply a by 2
    a = (a << 1) ^ (((a & 0x80) >> 7) * 0xA9);
    // cat the  b_(i+1)
    b = (b >> 1);
  }

  b ^= *of;
  return p;
}
```

(a) Multiplicação no corpo $GF(2^8)$ operando sobre $Q(x)$.

```
static inline uint32_t mulAlpha_galois32(
                            uint32_t b,
                            uint8_t *overflow) {

  uint8_t p = ((b & 0xFF000000) >> 24);
  int32_t ans = 0;

  ans = ans | ((*overflow ^
        mul_galois8(0x13, p, overflow)) << 0);
  *overflow = 0;
  ans = ans | ((*overflow ^
        mul_galois8(0xCF, p, overflow)) << 8);
  *overflow = 0;
  ans = ans | ((*overflow ^
        mul_galois8(0x9F, p, overflow)) << 16);
  *overflow = 0;
  ans = ans | ((*overflow ^
        mul_galois8(0xE1, p, overflow)) << 24);
  *overflow = 0;

  return ans;
}
```

(b) Multiplicação no corpo $GF(2^{32})$ operando sobre $P(x)$.

Figura 4. Implementação das funções para cálculo dos coeficientes da tabela `mulAlpha`.

Para evitar execuções em tempo variável, foi adotada a definição da operação de seleção(mux) de acordo com uma implementação opcional sugerida na implementação de referência, que evita o desvio condicional que frequentemente recai em números distintos de operações executadas ou variância no tempo por interferência

da predição de desvios. A implementação adotada é:

```
#define XMUX(c, x, y) (((signed int) ((c) << 31) >> 31) & (y)) ^ (x))
```

A macro seleciona entre os valores x ou $x \oplus y$, de forma que todas as operações serão executadas incondicionalmente, e será selecionado o resultado apropriado, a depender do valor de c.

As modificações realizadas na função de expansão da chave consistem em evitar o uso das funções `memcpy` e `memset` no momento em que se efetua o preenchimento da chave fornecida para 256 bits, visto que o tempo de execução dessas funções pode ser variável em alguns casos [Morrow 2004]. Por fim, foram realizadas alterações nos contextos das funções de inicialização da FSM e do LFSR para realizar conversões analogamente à estratégia adotada no ChaCha20: operar simultaneamente em bytes individuais armazenados em uma mesma palavra quando possível, para propiciar considerável economia de acessos à memória ou ainda evitar rotinas de conversão de tipos.

7. Resultados Experimentais

Esta seção apresenta uma comparação de desempenho entre implementações propostas para ambos os cifradores apresentados neste trabalho e as respectivas implementações de referência. Os testes foram realizados em diferentes plataformas ARM, com o propósito de medir, em ciclos por byte cifrado, os tempos de execução em cada plataforma, conforme Tabela 2.

O *benchmark* de nossas implementações foi realizado da seguinte forma nas respectivas plataformas:

- **Cortex-M3:** Arduino Due com processador Atmel SAM3X8E ARM Cortex-M3 84MHz. O compilador fornecido em última versão pelo Kit de Desenvolvimento Arduino, GCC 4.8.4, foi utilizado com as flags `-fno-schedule-insns-nostdlib -mcpu=cortex-m3 -mthumb`. O tempo de execução foi medido pela conversão da saída da função *micros()* do Arduino que mede ciclos em micro segundos através da simples multiplicação da frequência nominal.
- **Cortex-M4:** Teensy 3.2 é uma placa que contém o processador MK20DX256VLH7 Cortex-M4 de 72MHz. Utilizamos o mesmo compilador da plataforma Cortex-M3, porém com as flags `-fno-schedule-insns -nostdlib -mcpu=cortex-m4 -mthumb`. O tempo de execução foi medido através de um registrador para contagem de ciclos nativo à plataforma e do uso de um trecho de código em *assembly*.
- **Cortex-A7/15:** ODROID-XU4 é uma placa equipada com um Samsung Exynos5422 Cortex-A15 e um Cortex-A7 2GHz de oito núcleos. Instalamos a distribuição oficial do Arch Linux para a placa, que está equipada com o GCC 6.1.1 para plataformas ARM, usando as flags `-fno-schedule-insns -march=native`. O tempo de execução foi medido ao habilitar a leitura do registrador para contagem de ciclos(CCNT - Cycle CouNT register) no módulo de monitoramento de desempenho(Performance Monitor Unit - PMU) a nível de usuário.

- **Cortex-A8:** BeagleBone Black é uma placa equipada com um AM335x 1GHz ARM Cortex-A8. Mesmas configurações de Sistema Operacional, compilador, flags e ativação do contador de ciclos pela PMU da ODROID-XU4.
- **Cortex-A53:** ODROID-C2 é uma placa contendo um processador Amlogic ARM Cortex- A53(ARMv8) com quadro núcleos de 2GHz. Mesmas configurações de Sistema Operacional, compilador, flags e ativação do contador de ciclos pela PMU da ODROID-XU4.
- **Core i7 Ivy Bridge:** Intel Core i7-3632Q de clock 2.20GHz. O GCC 6.1.1 foi utilizado novamente e com as flags `-fno-schedule-insns -march=native`. O registrador RDTSC foi utilizado para contagem de ciclos.

Para o SOSEMANUK habilitamos também a flag `-O3` em todas as plataformas, já para o ChaCha20 habilitamos apenas o parâmetro `-O1` pois demonstrou melhor custo benefício.

Tabela 2. Tempo de execução e tamanho de código para ChaCha20 e SOSEMANUK no Cortex-M3/M4/A7/A8/A15/A53 ARM e Core i7 x86 Ivy Bridge. As células apresentam o tempo médio para cifrar um único byte (CPB) para entradas de 128 bytes e 4096 bytes, com TC sendo a implementação em tempo constante. A letra N representa a implementação proposta nesse trabalho e R a implementação de referência.

		ChaCha20			SOSEMANUK			SOSEMANUK TC		
		Entrada de 128 bytes (CPB)	Entrada de 4096 bytes (CPB)	Tamanho (bytes)	Entrada de 128 bytes (CPB)	Entrada de 4096 bytes (CPB)	Tamanho (bytes)	Entrada de 128 bytes (CPB)	Entrada de 4096 bytes (CPB)	Tamanho (bytes)
Cortex - M3	**N**	**49,84**	**49,84**	1348	65,95	24,57	7696	212,30	168,07	13761
	R	**71,95**	**71,97**	1152	65,74	28,01	17428	–	–	–
Cortex - M4	**N**	36,56	34,96	1348	45,37	15,37	7836	153,47	123,42	14061
	R	47,53	44,98	1152	45,24	17,97	17856	–	–	–
Cortex-A7	**N**	47,77	30,08	1896	**37,09**	10,25	8760	133,80	92,65	16097
	R	51,31	38,96	1736	**51,49**	12,35	19251	–	–	–
Cortex-A8	**N**	24,72	23,80	1896	24,87	6,46	8760	71,66	52,66	16097
	R	31,49	31,01	1736	26,33	8,31	19251	–	–	–
Cortex - A15	**N**	14,96	14,64	1896	17,60	**4,00**	8760	56,79	43,29	16097
	R	19,00	18,68	1736	18,53	**7,02**	19251	–	–	–
Cortex - A53	**N**	24,54	24,02	1552	17,34	3,93	8504	51,59	37,98	12883
	R	31,95	30,54	1548	17,78	6,58	19779	–	–	–
i7 Ivy Bridge	**N**	9,37	8,88	1602	21,33	6,28	8716	67,38	50,62	15121
	R	12,54	11,93	1235	21,91	7,13	19160	–	–	–

Cabe mencionar que a plataforma Intel x86, Core i7 Ivy Bridge é utilizada apenas como um comparativo que faz alusão à uma plataforma externa ao contexto das plataformas ARM que foram utilizadas como plataformas alvo para medição, e, portanto, fora do foco central deste trabalho, servindo apenas como um ponto adicional de comparação.

Como a Tabela 2 nos permite constatar, a depender da plataforma obtivemos ganhos percentuais de até 43% para o SOSEMANUK executando em um Cortex-A15 cifrando 4096 bytes por vez, e 30% para o ChaCha20 executando num Cortex M3 cifrando 128 bytes ou 4096 bytes, de forma que esses processadores são característicos de sistemas embarcados. A penalidade causada pela implementação protegida do SOSEMANUK e consequente compromisso entre proteção e tempo de execução mencionados previamente neste trabalho é mensurado por picos de *perda* de 650% em ciclos por byte comparado à implementação de referência, não protegida, no Cortex-A7 cifrando 4096 bytes e para cifrar 128 bytes, uma *perda* de 239% no Cortex M4.

Não foram encontradas implementações alternativas além da referência do

SOSEMANUK para comparação. A plataforma FELICS[1] possui diversas implementações do Chacha20, com destaque para uma implementação no Cortex-M3 que toma 55,43 ciclos por bytes na cifração de 128 bytes consecutivos e consome 740 bytes de ROM. A implementação descrita nesse trabalho é 12% mais rápida, porém quase 2 vezes maior. A plataforma SUPERCOP[2] possui implementações em *Assembly* ARM da Chacha20, portanto não diretamente comparáveis.

8. Conclusão

Este trabalho propõe otimizações às implementações de referência do ChaCha20 e do SOSEMANUK, onde, nas plataformas testadas, as implementações superam a de referência ou se igualam a esta considerando ±2% como flutuações de medição. As implementações trazem proteção adicional contra ataques de canal lateral de tempo por executarem em tempo constante. Vale ressaltar que as otimizações propostas ao ChaCha20 podem se estender às suas variantes com menor número de rodadas.

Agradecimentos

Os autores agradecem o apoio financeiro da empresa LG para financiamento dessa pesquisa, sob o projeto "*Efficient and Secure Cryptography for IoT*"

Referências

Berbain, C., Billet, O., Canteaut, A., Courtois, N., Gilbert, H., Goubin, L., Gouget, A., Granboulan, L., Lauradoux, C., Minier, M., Pornin, T., and Sibert, H. (2008). Sosemanuk, a fast software-oriented stream cipher. In *The eSTREAM Finalists*, volume 4986 of *LNCS*, pages 98–118. Springer.

Bernstein, D. J. (2005a). Chacha20, a variant of salsa20. `https://cr.yp.to/chacha/chacha-20080120.pdf`.

Bernstein, D. J. (2005b). Salsa20 specification. `https://cr.yp.to/snuffle/spec.pdf`.

Bernstein, D. J. (2008). The Salsa20 Family of Stream Ciphers. In *The eSTREAM Finalists*, volume 4986 of *LNCS*, pages 84–97. Springer.

Biham, E., Anderson, R. J., and Knudsen, L. R. (1998). Serpent: A new block cipher proposal. In *FSE*, volume 1372 of *LNCS*, pages 222–238. Springer.

Ekdahl, P. and Johansson, T. (2002). A new version of the stream cipher SNOW. In *Selected Areas in Cryptography*, volume 2595 of *LNCS*, pages 47–61. Springer.

eSTREAM (2004). the ECRYPT Stream Cipher Project. `http://www.ecrypt.eu.org/stream/`.

Morrow, M. (2004). Optimizing Memcpy improves speed. `www.embedded.com/design/configurable-systems/4024961/Optimizing-Memcpy-improves-speed`. Acessado: 09-09-2016.

[1] `https://www.cryptolux.org/index.php/FELICS_Stream_Ciphers_Brief_Results`
[2] `https://bench.cr.yp.to/results-stream.html`

XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

Niterói, RJ, novembro de 2016

WGID – VI Workshop de Gestão de Identidades Digitais

Estudo Comparativo das Soluções de eID Móvel para Governo Eletrônico

Glaidson Menegazzo Verzeletti[1,2], Emerson Ribeiro de Mello[2]*, Victor Hugo Barbosa de Oliveira[3], Michelle Silva Wangham[1]

[1]Universidade do Vale do Itajaí (UNIVALI) – SC – Brasil

[2]Instituto Federal de Santa Catarina – SC – Brasil

[3]Universidade de Brasília – DF – Brasil

{glaidson.verzeletti, mello}@ifsc.edu.br,

hugo.victoor77@gmail.com, wangham@univali.br

Abstract. *The Mobile Electronic Identity (Mobile eID) is an important tool to connect citizens and governments securely. Thus, this tool improves citizens' access to electronic government services, as well as to allow a greater interaction of people with public policies and decisions. This article aims to describe and analyze eID solutions described in the literature which have been selected after a systematic review.*

Resumo. *A Identidade Eletrônica Móvel (eID Móvel) é uma importante solução para conectar cidadãos e governos de forma segura. Dessa forma, esta solução melhora o acesso dos cidadãos aos serviços de governo eletrônico, além de permitir uma maior interação das pessoas com as políticas e decisões públicas. Este artigo tem por objetivo descrever e analisar as soluções de identidade eletrônica móvel descritas na literatura e que foram selecionadas após a condução de uma revisão sistemática da literatura.*

1. Introdução

Segundo [ONU 2014], o governo eletrônico constitui-se de uma importante ferramenta para revitalizar a administração pública tanto no nível nacional quanto local. Como estratégia, muitos países indicam a necessidade de oferecer soluções robustas de autenticação do ponto de vista da segurança, que proporcionem mobilidade aos cidadãos e que tenham, preferencialmente, baixo custo. Sendo assim, o desenvolvimento de uma Estratégia Nacional de Gestão de Identidade Eletrônica (GId) é fundamental para a realização de Programas de Governo Eletrônico (e-Gov).

Em [OECD 2011], foi apresentado um estudo sobre as estratégias nacionais de Gid para e-Gov que foram adotadas por alguns países. Constatou-se que tais estratégias se basearam em práticas e regulamentos dos sistemas de identidade convencional existentes. Ou seja, os países presentes no estudo optaram por lançar uma carteira de identidade eletrônica (cartão eID), de forma que a distribuição e entrega dos mesmos para os cidadãos seguisse aquilo que já era feito com as carteiras de identidade convencionais com suporte em papel.

*Bolsista CNPq - 200995/2015-4

O uso de cartão eID aumenta a robustez dos processos de identificação e autenticação do cidadão perante serviços do governo eletrônico, uma vez que esse cartão possui elementos criptográficos e a habilidade de executar algumas pequenas aplicações dentro do próprio ambiente seguro do cartão. A identidade eletrônica móvel (eID móvel) pode ser vista como uma evolução dos cartões eID, tendo em vista que o cidadão poderia fazer uso do seu próprio telefone para acessar de forma segura os serviços de governo eletrônico. Conforme observado por [Ruiz-Martínez et al. 2007], a adoção do eID móvel pode ser motivada pelo crescente uso de serviços de telecomunicações e principalmente pela baixa aceitação das eIDs baseadas em cartão.

Como as soluções de eID Móvel oferecem a possibilidade de autenticação usando a eID a partir do próprio telefone móvel do usuário, o uso do cartão de eID pode ser substituído nos processos de autenticação ou funcionar como complemento às estratégias nacionais de GId. Neste último cenário, as soluções de eID Móvel atuam como tecnologias de inclusão, permitindo que mais pessoas se beneficiem do uso de uma eID para interagir com o governo. Resumidamente, as soluções de eID Móvel tem como potencial aumentar significativamente o fator usabilidade e inclusão digital, sem perder o nível de segurança oferecido pelas soluções baseadas em cartão de eID [Rath et al. 2014].

O objetivo deste artigo é analisar as estratégias e soluções de identidade eletrônica móvel (eId Móvel) presentes na literatura acadêmica. Este artigo descreve os resultados obtidos (trabalhos selecionados) de uma revisão sistemática executada em agosto de 2016 e está organizado da seguinte forma: Na Seção 2, são apresentados os conceitos e características referentes a eID Móvel. A Seção 3 descreve como a revisão sistemática da literatura foi conduzida, bem como a descrição dos trabalhos relacionados. Na Seção 4 é feita a análise dos trabalhos relacionados. Por fim, na Seção 5 são apresentadas as considerações finais e trabalhos futuros.

2. Identidade Eletrônica Móvel (eID Móvel)

A identidade de uma pessoa pode ser descrita como sendo um conjunto de informações pessoais, utilizado a fim de caracterizar corretamente um indivíduo. Essas informações podem se constituir do nome da pessoa, registro biométrico, altura, cor do cabelo, entre outros. Para realizar essa caracterização, a identidade pode também estar associada a outros atributos ligados à pessoa como o nome da mãe, data e local de nascimento. Dependendo do uso e do contexto, a identidade pode ser composta somente por algumas destas informações pessoais [NSTC 2008]. Para [Clauß and Köhntopp 2001], a identidade eletrônica (eID) é criada quando informações pessoais são utilizadas para se caracterizar uma pessoa no meio digital.

Por definição, a identidade eletrônica móvel se refere ao uso da eID através de dispositivos móveis, como telefones inteligentes (*smartphones*) e *tablets*. Segundo [World 2015], eID Móvel é um conjunto formado pela eID do usuário e pelas tecnologias utilizadas para gerenciá-la. Já para [Telia 2016], é como se o cartão de eID estivesse dentro de um telefone móvel.

Durante as últimas décadas, algumas soluções de eID Móvel foram testadas e implementadas em alguns países. Enquanto algumas soluções continuam em uso, outras foram abandonadas ou substituídas. Atualmente, um conjunto heterogêneo de soluções encontra-se em produção, englobando um número crescente e contínuo de diferentes con-

ceitos tecnológicos e implementações [Zefferer and Teufl 2015].

Para [Rath et al. 2014], apesar da diversidade de implementações da eID Móvel, o modelo de confiança advém do uso de algum elemento seguro (*secure hardware element* – SE) presente no dispositivo móvel. Este SE deve possuir a capacidade de armazenar e realizar operações de criptografia. De acordo com esse autor, as implementações atuais de eID Móvel podem ser classificadas em dois tipos de soluções, dependendo da localização do elemento seguro:

- Soluções baseadas em cliente: soluções baseadas neste modelo, normalmente, fazem uso de um cartão SIM (*subscriber identity module*) especial, capaz de armazenar a eID de forma segura e prover operações de criptografia como assinatura de documentos. Tipicamente, o acesso aos dados e funcionalidades deste cartão SIM está protegido por uma senha secreta (*Personal Identification Number* – PIN), proporcionando assim o segundo fator de autenticação.

- Soluções baseadas em servidor: essas soluções utilizam as técnicas de computação em nuvem para implementar um elemento de *hardware* central, como, por exemplo, um Módulo de Hardware Seguro (HSM). Assim, o dispositivo móvel do usuário não armazena os dados da eID e não implementa as funcionalidades de criptografia. Entretanto, o dispositivo móvel participa do processo de autenticação, como um dos componentes obrigatórios. Isto porque este dispositivo é o responsável por acessar a eID no HSM e destravar as funcionalidades de criptografia, o que geralmente é feito com o recebimento de uma senha descartável (*one-time password* – OTP).

Para [Mantoro and Milišić 2010], nos processos de autenticação, as chances de forjar uma identidade devem ser minimizadas, utilizando algum elemento seguro como apresentado acima. É possível utilizar o telefone móvel pessoal para identificar e autenticar o cidadão de forma segura durante o acesso a um provedor de serviço (*Service Provider* – SP) governamental, por exemplo. Por outro lado, conforme apresentado por [Stamp 2011], é recomendável utilizar mais de um fator de autenticação de forma a reforçar a segurança.

Implementar uma solução baseada em cliente ou em servidor depende da infraestrutura e da estratégia nacional de GId para e-Gov adotada em cada país. Como exemplificado por [Zefferer and Teufl 2015], os países Bálticos e Escandinavos passaram a adotar arquiteturas baseadas em cliente, com o uso de cartões SIM como elementos de hardware seguros. A Áustria, por outro lado, preferiu a adoção de uma arquitetura baseada em servidor, utilizando o HSM como hardware seguro.

A popularização e o aumento da adoção de uma solução de eID Móvel pelos governos tem atraído muitas oportunidades de negócios. Empresas privadas começaram a oferecer soluções de eID Móvel prontas para uso, de forma a facilitar a implantação pelos governos. Dentre estas, podem ser citadas as empresas *Valimo Wireless* Ltd[1] e a *Methics Ltd*[2], que oferecem soluções baseadas em cartão SIM; e a ServerBKU[3] e PkBox[4], que

[1] http://www.valimo.com/
[2] http://www.methics.fi/
[3] http://www.valimo.com/
[4] http://www.pksuite.it/eng/pr_pkbox.php

oferecem soluções baseadas no uso de HSM.

Porém, ao abordar a questão identidade eletrônica móvel tanto em soluções oferecidas por empresas privadas quanto públicas, uma preocupação constante se refere à privacidade das pessoas. [Laurido and Feitosa 2015] descrevem privacidade como sendo o direito de cada pessoa de manter e controlar suas informações pessoais, decidindo quem pode ter acesso a elas. Para [Pournajaf et al. 2014], a quebra de privacidade é considerada um problema ao expor atributos sensíveis de uma pessoa como um simples endereço de residência, até informações privadas e particulares como doenças ou estilos de vida.

Como a identidade digital de uma pessoa pode estar atrelada a um conjunto de informações pessoais, conforme afirmado por [Jøsang et al. 2007], o sistema de gestão de identidade desempenha um papel importante na preservação da privacidade. Para [Hansen et al. 2008], há uma grande variedade de princípios e guias que visam estabelecer e manter a privacidade do sistema de gestão de identidade. Os princípios mais amplamente aceitos são as Práticas para Informação Justa (*Fair Information Practice* – FIPs), que foram desenvolvidos na década de 1970.

A OECD[5] publicou um conjunto de diretrizes baseado no FIPs que objetiva garantir a privacidade para a troca de dados pessoais [Hansen et al. 2008]. Em 2013, o conjunto de diretrizes criadas pela OECD foi atualizado, de forma a garantir aderência ao avanço tecnológico dos últimos anos. Ao adotá-lo como guia, um sistema de gestão de identidade passa a operar no modelo centrado no usuário, oferecendo a esse usuário o controle sobre suas informações pessoais [OECD 2013].

De forma geral, a privacidade pode ser entendida como a possibilidade do usuário determinar quais atributos pessoais serão divulgados para os SPs e se concorda com os termos de uso estabelecidos pelo provedor. Segundo os princípios definidos pela OECD, o usuário tem o direito de garantir seu anonimato perante o provedor de serviço, bem como divulgar as informações pessoais que julgar adequadas.

Contudo, apesar de existirem diretrizes para garantir a privacidade dos usuários, quando se trata de soluções de eID Móvel para governo eletrônico, também deve-se considerar que o direito à privacidade deve ser algo previsto em lei.

3. Revisão Sistemática da Literatura

A execução de uma Revisão Sistemática da Literatura (RSL) teve como objetivo principal identificar as publicações relacionadas ao uso da Identidade Eletrônica Móvel (eID Móvel), dentro e fora do contexto de governo eletrônico. [Kitchenham and Charters 2007] propõe quatro passos para conduzir uma revisão sistemática da literatura (RSL): (1) identificação dos recursos (questão de pesquisa, palavras chaves e fontes); (2) seleção dos estudos; (3) extração dos dados e (4) análise dos dados.

Segundo [Kitchenham and Charters 2007], o estabelecimento das questões de pesquisa se constitui a parte mais importante de qualquer revisão sistemática. O processo de revisão visa, portanto, encontrar os estudos candidatos capazes de responder às questões de pesquisa formuladas e a análise destes estudos deverão ser capazes de trazer as respostas esperadas. Para identificar e selecionar estes estudos, foi elaborado e executado

[5]Disponível em http://oecdprivacy.org/.

um protocolo de busca para encontrar nas bases científicas os estudos primários que respondem à seguinte questão: quais soluções de identidade eletrônica móvel (eID Móvel) propostas podem ser aplicadas no contexto de governo eletrônico? Como questão adicional, definiu-se ainda: quais tecnologias estão sendo utilizadas para prover a eID Móvel? Através da análise das publicações, buscou-se analisar as soluções de eID Móvel propostas de forma a identificar como ocorre o armazenamento dos atributos do usuário e como este usuário é identificado pelo provedor de serviço (SP) ao fazer uso do identificador eletrônico (eID).

Na revisão sistemática, executada em agosto de 2016, definiu-se que apenas os estudos publicados na língua inglesa seriam considerados, de acordo com a *string* de busca: ("MOBILEID" OR "MOBILE ID" OR "MOBILE-ID" OR "MOBILE EID" OR "M-ID" OR "MOBILE IDENTIFICATION" OR "MOBILE IDENTITY" OR "IDENTIFICATION ON MOBILE").

O protocolo de busca foi executado considerando as cinco fontes mais relevantes da área, a saber: ACM Digital Library [6], IEEE Explore [7], Science Direct [8], Scopus [9] e SpringerLink [10]. Para extrair a primeira lista de trabalhos, executou-se a string de busca nas fontes citadas, considerando o título e resumo, e utilizou-se como filtro trabalhos publicados nos últimos 10 anos. A Tabela 1 apresenta o total de publicações retornadas de cada uma das cinco bases científicas, o que representou um total de 275 trabalhos.

Tabela 1. Resultados da Execução do Protocolo de Busca e Seleção de Trabalhos

	ACM	IEEEXplorer	ScienceDirect	Scopus	Springer	Total
Resultado da string	25	27	10	52	161	275
Trabalhos lidos	3	6	0	7	2	18
Trabalhos selecionados	2	2	0	3	1	8

Ainda na etapa de seleção dos trabalhos (etapa 2 da RSL), os seguintes procedimentos foram executados para inclusão e exclusão de trabalhos: (1) após verificar o título, autores e filiação, procurou-se identificar os trabalhos repetidos (foi mantido o trabalho mais recente). Trinta e nove (39) trabalhos repetidos foram excluídos da lista; (2) após a leitura do título e do resumo de cada artigo, foram excluídos trabalhos que não tratam exatamente de eID Móvel ou cujos títulos e resumos apresentavam informações conflitantes. Em caso de dúvida, a conclusão do artigo foi analisada. Ainda nesta etapa, trabalhos com até 3 páginas, que caracterizavam resumo estendido e trabalhos sem resumo foram excluídos. Ao final desta etapa 2, duzentos e dezoito trabalhos foram excluídos. Como resultado (3), dezoito artigos foram incluídos na lista de artigos pré-selecionados, pois estes respondiam à questão primária de pesquisa, ou seja, os trabalhos descrevem uma solução para prover eID Móvel e não apenas descrevem o problema (ver Tabela 1). Finalmente, após a leitura completa dos dezoito trabalhos, foram selecionados oito trabalhos que apresentam de forma consistente uma solução para prover eID Móvel. Estes trabalhos estão indicados na Tabela 2.

[6] http://dl.acm.org/
[7] http://ieeexplore.ieee.org/
[8] http://www.sciencedirect.com/
[9] http://link.springer.com/search
[10] http://link.springer.com/

Tabela 2. Resultados da Execução do Protocolo de Busca e Seleção de Trabalhos

Referência	Título	Tipo de Publicação	Fonte
MARTENS (2010)	Electronic identity management in Estonia between market and state governance	Periódico	Springer
EN-NASRY E KETTANI (2011)	Towards an open framework for mobile digital identity management through strong authentication methods	Evento Internacional	Springer
BICAKCI (2014)	Mobile authentication secure against man-in-the-middle attacks	Evento Internacional	IEEE
WU ET AL. (2014)	Research of eID Mobile Identity Authentication Method	Evento Internacional	Springer
KRIMPE (2014)	Mobile ID: Crucial element of m-Government	Evento Internacional	ACM
PRUSA (2015)	E-identity: Basic building block of e-Government	Evento Internacional	IEEE
ZEFFERER E TEUFL (2015)	Leveraging the Adoption of Mobile eID and e-Signature Solutions in Europe	Evento Internacional	Springer
KETTULA (2015)	A novel federated strong mobile signature service: The Finnish case	Periódico	Elsevier

3.1. Descrição dos Trabalhos Selecionados

O trabalho apresentado por [Martens 2010] descreve o Sistema de Gestão de Identidade Eletrônica (*electronic Identity Management System* – eIDMS) da Estônia para o contexto de e-Gov do país. A eID Móvel foi introduzida no governo estoniano em maio de 2007 pela maior operadora móvel (EMT) em cooperação com a SK[11] (Autoridade Certificadora Estoniana). Para poder utilizar a eID Móvel, o usuário precisa trocar o seu cartão SIM por um cartão com capacidade PKI (*Public Key Infrastructure*). Porém, mesmo após realizar a troca do cartão SIM por outro com funcionalidade PKI, o usuário ainda precisa ativar a sua eID Móvel, o que é feito através do cartão de eID.

A PKI adotada na Estônia, prevê o uso de dois pares de certificados digitais para o cidadão: um par utilizado para autenticação e um par utilizado para assinatura eletrônica. Com validade de cinco anos, os dois pares são considerados qualificados, ou seja, são certificados reconhecidos legalmente pelo Ato de Assinatura Digital[12]. Cada uma das chaves privadas possui um código PIN diferente. Os certificados contêm o nome completo do cidadão, seu código pessoal[13] (*Personal Identification Code* – PIC) e um endereço de email reconhecido pelo governo. Entretanto, nenhuma informação biométrica é utilizada.

Em [En-Nasry and El Kettani 2011], os autores propõem um *middleware* para ser integrado em provedores de identidade, de forma a oferecer vários métodos de autenticação para o usuário. Segundo os autores, entre os métodos mais comuns de autenticação dos sistemas de eID Móvel, destacam-se: autenticação baseada em cartão SIM, autenticação baseada em chaves públicas (PKI) e autenticação baseada em senhas descartáveis (*one time password* – OTP).

Independente do mecanismo de eID Móvel adotado, o modelo proposto em [En-Nasry and El Kettani 2011] se baseia em uma camada de software (*middleware*) para

[11]Disponível em https://sk.ee/en/about/.

[12]Disponível em http://www.legaltext.ee/text/en/X30081K4.htm

[13]O PIC é atribuído a todo cidadão do país desde a independência do país em 1992.

integrar todos os métodos de autenticação. Este *middleware* deve ser capaz de gerenciar perfis de usuário e contexto, garantindo que a autenticação possa ocorrer a partir de diversos métodos. Isto significa que o usuário poderá utilizar um, dentre vários métodos, para se autenticar no SP. A proposta prima pela usabilidade, desobrigando o usuário de lembrar exatamente qual o método de autenticação deve utilizar para acessar um serviço *online* protegido.

O protocolo proposto por [Bicakci et al. 2014] tem por objetivo garantir a segurança contra ataques do tipo homem-do-meio (*Man-in-The-Middle* – MITM) em ambientes de autenticação, utilizando soluções tradicionais de eID Móvel. Para os autores, as soluções tradicionais de eID Móvel são as que geram um par de chaves criptográficas assimétricas (pública e privada) para autenticação do usuário, armazenando essas chaves no cartão SIM e protegendo-as com uma senha pessoal (PIN). Contudo, para autenticação no SP com uma eID Móvel, o protocolo proposto leva em consideração o contexto no qual o usuário e o dispositivo estão inseridos. O servidor de assinatura móvel ganha a capacidade de autenticar o usuário, dispensando o uso de uma entidade certificadora (CA). Assume-se também como premissa que o sistema operacional móvel é seguro, portanto o dispositivo móvel deve estar equipado com o ambiente de execução confiável (*Trusted Executed Environment* – TEE).

O método de autenticação em eID Móvel proposto em [Wu et al. 2014], é baseado na utilização da identidade eletrônica em conjunto com a tecnologia NFC (*Near Field Communication*). Os experimentos realizados pelo autor demonstraram que o método proposto pode ser utilizado como uma solução de eID Móvel universal. De forma geral, o método de eID Móvel proposto visa prover a autenticação segura e a privacidade do usuário.

[Krimpe 2014] descreve a implementação da eID Móvel na República do Azerbaijão. Conhecida no país por *Asan Ìmza* ou "Assinatura Simples", a solução de eID Móvel faz parte do projeto de assinatura eletrônica (*e-Signature*) móvel, sendo o principal componente das políticas de e-Gov daquele país. A eID Móvel foi lançada como projeto piloto em 2013 e, oficialmente, entrou em operação no ano de 2014.

A implementação da identidade móvel no Azerbaijão é baseada em uma parceria público-privada (*Public-Private-Partnership* – PPP), realizada com as maiores operadoras de telefonia móvel do país. Um cartão SIM diferenciado, que armazena os certificados digitais pessoais no dispositivo móvel é fornecido ao cidadão. Com base em uma infraestrutura de chave público-privada (PKI), os certificados pessoais emitidos são armazenados no cartão SIM, evitando que a chave privada possa ser copiada. Os certificados também só podem ser destravados com o uso de senhas pessoais (códigos PIN).

O trabalho de [Prusa 2015] trata dos aspectos ligados à confiança relacionada com a eID e sua integração com o governo eletrônico. Foi feita uma discussão sobre os projetos europeus STORK e STORK 2.0 e a integração entre os países da União Europeia, como forma de motivar o avanço da interoperabilidade entre países, especialmente os países da União Africana. O uso da identidade eletrônica é o foco principal da publicação, abordando de forma rápida o uso da eID através de dispositivos móveis e citando superficialmente a Áustria, Estônia e Islândia como países que disponibilizaram a solução de eID Móvel para seus cidadãos.

O trabalho desenvolvido por [Zefferer and Teufl 2015] apresenta uma pesquisa e analisa as soluções de eID Móvel e assinatura eletrônica, adotadas por vários países europeus, com o intuito de servir como referência para outros governos. Na Estônia, a solução de eID Móvel e assinatura eletrônica é conhecida "Mobiil-ID". Nessa solução, os dados do usuário são armazenados no cartão SIM fornecido pelas operadoras de telefonia móvel. Com abordagem semelhante, a solução chamada de "BankID" na Noruega indica também a participação do setor bancário. Outras soluções que seguem a abordagem com uso do cartão SIM também são encontradas na Finlândia e Lituânia, o que demonstra que os Estados Escandinavos e Bálticos estão juntamente com a Áustria, entre os países líderes com relação à adoção da eID Móvel e assinatura eletrônica na Europa.

Com o nome "MobilImza", a Turquia desenvolveu a solução de eID Móvel e assinatura eletrônica, tendo nas operadoras de telefonia móvel Turkcell e Avea os tomadores de decisões centrais. Assim como nos países Escandinavos e Bálticos, a solução da Turquia segue a abordagem de arquitetura baseada em cliente, porém, utilizando um cartão SIM melhorado. Soluções utilizando o cartão SIM foram adotados pela Moldova e Suíça. Na Moldova, a solução foi introduzida em 2012 com a participação das operadoras de telefonia móvel e tendo o governo como entidade certificadora raiz. Na Suíça, com o nome de "Swiss Mobile ID", a solução foi colocada em produção em 2013.

O trabalho de [Kerttula 2015] tem por objetivo descrever questões legais, arquitetura e padrões do serviço de eID Móvel da Finlândia. O serviço é baseado um uma estrutura PKI móvel e na federação de asserções de segurança, usando padrões ETSI MSS[14]. As credencias pessoais de segurança são armazenadas no UICC (*Universal Integrated Circuit Card*) e as operações de criptografia rodam neste SE a prova de violação do operador de telefonia móvel.

Os operadores de telefonia móvel participam ativamente do sistema de eID Móvel naquele país. Com serviço lançado em junho de 2011[15], atualmente, as operadoras TeliaSonera Finland, Elisa e DNA são as responsáveis pela implementação e desenvolvimento da eID Móvel, atuando em cooperação com a FiCom[16]. Entretanto, o sistema é aberto ao ponto de permitir a entrada de novos operadores de telefonia.

Como o modelo faz uso de um elemento seguro (USIM), as chaves RSA podem ser geradas de uma forma simples e segura dentro deste elemento. Da mesma forma, um conjunto de aplicativos (*SIM Application Toolkit* – SAT) é inserido no SE para permitir a assinatura móvel.

4. Análise dos Trabalhos

Na Tabela 3, são apresentadas algumas informações sobre a implementação das soluções de eID Móvel, obtidas na análise dos trabalhos relacionados, a saber (1) os cenários de aplicação da eID Móvel, ou seja, se a solução de eID Móvel foi implementada ou projetada para uso pelos SPs governamentais e SPs privados ou trata-se apenas de um modelo teórico; (2) os países para os quais a solução foi proposta ou implementada; (3) o ano de início da operação (data de lançamento) da solução de eID Móvel; (4) se houve o estabelecimento de parcerias público-privada para desenvolvimento da solução; (5) o tipo de

[14]Disponível em http://www.etsi.org/

[15]Disponível em www.mobiilivarmenne.fi/en/index.html.

[16]Finnish Federation of Communications and Informatics.

elemento seguro adotado pela solução de eID Móvel; e (6) o que é preciso que o cidadão informe para que o processo de autenticação ocorra, como por exemplo, senha numérica, senhas descartáveis (OTP) ou fatores biométricos.

Tabela 3. Comparação dos trabalhos relacionados

Referência	Cenário	País	Ano	PPP	SE	Autenticação
MARTENS (2010)	e-Gov	Estônia	2007	Sim	Cartão SIM	PIN
EN-NASRY E KETTANI (2011)	Teórico	-	-	Sim	Cartão SIM	OTP
BICAKCI (2014)	Teórico	-	-	Sim	C. SIM / TEE	PIN
WU ET AL. (2014)	Teórico	-	-	-	Cartão de eID	PIN
KRIMPE (2014)	e-Gov	Azerbaijão	2014	Sim	Cartão SIM	PIN
PRUSA (2015)	e-Gov	Islândia	2014	Sim	Cartão SIM	PIN
ZEFFERER, TEUFL (2015)	e-Gov	Austria	2010	-	HSM	Senha / OTP
	e-Gov	Noruega	2009	Sim	Cartão SIM	PIN
	e-Gov	Lituânia	2007	Sim	Cartão SIM	PIN
	e-Gov	Turquia	2007	Sim	Cartão SIM	PIN
	e-Gov	Moldova	2012	Sim	Cartão SIM	PIN
	e-Gov	Suíça	2013	Sim	Cartão SIM	PIN
KERTTULA (2015)	e-Gov/Priv	Finlândia	2010	Sim	Cartão USIM	PIN / OTP

Alguns países foram citados em mais de um trabalho relacionado. Assim, foi considerada apenas uma referência (linha) na Tabela, como é o caso da Áustria que aparece nos trabalhos de [Prusa 2015], [Zefferer and Teufl 2015] e [Martens 2010], da Estônia que é descrita nos trabalhos de [Prusa 2015] e [Zefferer and Teufl 2015], e da Finlândia que é citada por [Kerttula 2015] e [Zefferer and Teufl 2015]. Dados incompletos, imprecisos, não informados ou irrelevantes foram deixados intencionalmente sem preenchimento (marcação "-").

Nas publicações de [Martens 2010], [Krimpe 2014] e [Kerttula 2015], apenas um país em cada artigo foi analisado pelos autores, respectivamente, a Estônia, a República do Azerbaijão e a Finlândia. Embora a primeira solução de eID Móvel tenha sido implementada em 2007 (Estônia), observou-se que as implementações posteriores seguem a mesma abordagem, com a adoção do cartão SIM como elemento seguro e utilização da eID tanto pelo governo quanto pelo setor privado.

Os autores [En-Nasry and El Kettani 2011], [Bicakci et al. 2014] e [Wu et al. 2014] optaram por apresentar um modelo teórico de autenticação móvel. [En-Nasry and El Kettani 2011] propõem um protocolo extensível para operar de forma independente em plataformas de autenticação (ex. IdP), de forma que essas plataformas sejam capazes de reconhecer e trabalhar com todos os tipos de autenticação existentes, como biometria, senhas descartáveis, identificação de contexto, entre outras. [Bicakci et al. 2014] apresentam o protocolo "Mobile-ID" utilizando padrões abertos, com uma abordagem que procura mitigar problemas com ataques do tipo homem-no-meio. [Wu et al. 2014] foram os únicos a propor a utilização do *chip* do cartão de eID do cidadão como elemento seguro. Este modelo sugere a utilização do leitor NFC dos dispositivos móveis, que em conjunto com os módulos idealizados, visam promover uma forma segura de autenticação móvel.

Em [Prusa 2015], quatro países são supervisualmente citados: Áustria, Estônia, Islândia e República Tcheca. Destes, a Islândia traz mais informações se destacando com uma das implementações mais recentes de eID Móvel. Por outro lado, o trabalho apresen-

tado por [Zefferer and Teufl 2015] descreve o cenário de eID Móvel em diversos países europeus, dos quais a Áustria se destaca como o único país a apresentar uma solução baseada em servidor.

Permitir a participação de instituições privadas aparenta ser uma prática comum, visto que uma mesma eID pode ser utilizada tanto para autenticação em provedor de serviço governamental tanto quanto o SP privado. Apesar das semelhanças encontradas, a data de implementação varia muito. Países como a Estônia, Lituânia e Turquia implementaram a eID Móvel em 2007; já a Islândia, República do Azerbaijão e Suíça implementaram a solução recentemente.

Ao observar na Tabela 3, o ano de início de operação das soluções, constata-se a afirmação de [Zefferer and Teufl 2015] ao citar que os Estados Escandinavos e Bálticos estão, juntamente com a Áustria, entre os países líderes com relação à adoção da eID Móvel na Europa.

Apesar da maioria das soluções apresentarem parcerias entre governo e iniciativa privada, não foi apresentado pelos trabalhos relacionados o grau de envolvimento do governo ou da iniciativa privada. Ou seja, não foi possível identificar até que ponto o governo dita as regras de negócios e determina quais padrões tecnológicos devem ser seguidos. Da mesma forma, não foi possível determinar como o direito à privacidade é garantido aos cidadãos e usuários das soluções de eID Móvel.

Pela simplicidade de implementação, observa-se que a utilização do cartão SIM como elemento seguro tem sido a solução de eID Móvel mais bem aceita pelos governos. Portanto, considerando a questão de pesquisa elencada como objetivo principal da revisão sistemática, esta solução aparenta ser a mais aderente às políticas de governo eletrônico.

Contudo, considerando a questão adicional de pesquisa, é importante destacar que o HSM também foi citado como solução viável de eID Móvel. Da mesma forma, o trabalho teórico apresentado por [Bicakci et al. 2014], sugere a utilização do TEE como fator complementar à segurança.

De forma geral, conforme apresentado na Tabela 3, como solução baseada no cliente o cartão SIM é identificado como o mais adotado, bem como o HSM é considerado quando o governo pretende adotar uma solução baseada em servidor.

5. Conclusão

Com avanço tecnológico dos dispositivos móveis, as capacidades de *hardware* têm se ampliado e os softwares estão se tornando cada vez mais intuitivos, fazendo destes aparelhos o elemento perfeito para estar junto do usuário em qualquer lugar. Ao combinar elementos seguros de armazenamento e processamento, estes dispositivos passam a permitir que transações *online* seguras possam ser realizadas.

Neste contexto, as soluções de eID Móvel se destacam como soluções viáveis pela usabilidade e pela alta segurança oferecida. No geral, os custos para os cidadãos são relativamente baixos, uma vez que o componente mais caro destas soluções é o dispositivo móvel e normalmente o cidadão já o possui. Além disso, as demais funcionalidades providas por esses dispositivos permitem ao cidadão outras facilidades, que não o uso exclusivo como identificador eletrônico.

Conforme observado no estudo dos trabalhos relacionados, a adoção de soluções de eID Móvel se mostra cada vez mais frequente. Oferecer serviços a uma gama maior de pessoas tem motivado sua adoção por governos, além de ter atraído o interesse de empresas privadas. Trabalhos científicos, em sua maioria, descrevem as soluções já implementadas e prontas para uso. Poucos trabalhos realmente propõem uma solução de eID nova ou oferecem resultados de uma prova de conceito. Mesmo assim, a maioria das soluções foi concebida através da utilização de cartões SIM, justamente pela facilidade de implementação.

Por fim, este estudo demonstrou que as soluções de eID Móvel estão gradativamente sendo adotadas por diferentes países por se constituírem uma grande ferramenta para aumentar a interação dos cidadãos e governos. Em trabalhos futuros, será estudado o cenário de e-Gov no Brasil e será proposto um sistema de eID Móvel que seja aderente à estratégia de GId nacional.

Referências

Bicakci, K., Unal, D., Ascioglu, N., and Adalier, O. (2014). Mobile authentication secure against man-in-the-middle attacks. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, pages 273–276. IEEE. http://dx.doi.org/10.1109/MobileCloud.2014.43.

Clauß, S. and Köhntopp, M. (2001). Identity management and its support of multilateral security. *Computer Networks*, 37(2):205–219. http://dx.doi.org/10.1016/S1389-1286(01)00217-1.

En-Nasry, B. and El Kettani, M. D. E.-C. (2011). Towards an open framework for mobile digital identity management through strong authentication methods. In *FTRA International Conference on Secure and Trust Computing, Data Management, and Application*, pages 56–63. Springer. http://dx.doi.org/10.1007/978-3-642-22365-5_8.

Hansen, M., Schwartz, A., and Cooper, A. (2008). Privacy and identity management. *IEEE Security & Privacy*, 6(2):38–45. http://dx.doi.org/10.1109/MSP.2008.41.

Jøsang, A., Zomai, M. A., and Suriadi, S. (2007). Usability and privacy in identity management architectures. In *Proceedings of the fifth Australasian symposium on ACSW frontiers-Volume 68*, pages 143–152. Australian Computer Society, Inc.

Kerttula, E. (2015). A novel federated strong mobile signature service—the finnish case. *Journal of Network and Computer Applications*, 56:101–114. http://dx.doi.org/10.1016/j.jnca.2015.06.007.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report.

Krimpe, J. (2014). Mobile id. In *Proceedings of the 2014 Conference on Electronic Governance and Open Society: Challenges in Eurasia*, pages 187–194. ACM. http://dx.doi.org/10.1145/2729104.2729133.

Laurido, J. J. d. V. and Feitosa, E. L. (2015). Segurança em mobile crowd sensing. In *V WGID - Workshop de Gestão de Identidades – XV Simpósio Brasileiro*

em Segurança da Informação e de Sistemas Computacionais - SBSeg, pages 51–92. http://sbseg2015.univali.br/anais/WGID/artigoWGID02.pdf.

Mantoro, T. and Milišić, A. (2010). Smart card authentication for internet applications using nfc enabled phone. In *Information and Communication Technology for the Muslim World (ICT4M), 2010 International Conference on*, pages D13–D18. IEEE. http://dx.doi.org/10.1109/ICT4M.2010.5971895.

Martens, T. (2010). Electronic identity management in estonia between market and state governance. *Identity in the Information Society*, 3(1):213–233. http://dx.doi.org/10.1007/s12394-010-0044-0.

NSTC (2008). Identity management task force report. *Subcomiittee on Biometrics and Identity Management*. https://goo.gl/2iIYOG.

OECD (2011). *National strategies and policies for digital identity management in OECD countries*. OECD Publishing. dx.doi.org/10.1787/5kgdzvn5rfs2-en.

OECD (2013). *OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data*. OECD Publishing. https://goo.gl/Y0yxmw.

ONU (2014). *e-Government Survey*. United Nations. https://goo.gl/jccKDV.

Pournajaf, L., Xiong, L., Garcia-Ulloa, D. A., and Sunderam, V. (2014). A survey on privacy in mobile crowd sensing task management. Technical report, Technical Report TR-2014-002, Department of Mathe-matics and Computer Science, Emory University. http://www.mathcs.emory.edu/~lpourna/papers/pournajaf-survey14.pdf.

Prusa, J. (2015). E-identity. In *IST-Africa Conference, 2015*, pages 1–10. IEEE. http://dx.doi.org/10.1109/ISTAFRICA.2015.7190586.

Rath, C., Roth, S., Schallar, M., and Zefferer, T. (2014). A secure and flexible server-based mobile eid and e-signature solution. In *The Eighth International Conference on Digital Society*, pages 7–12. http://dx.doi.org/10.1145/2684103.2684142.

Ruiz-Martínez, A., Sánchez-Martínez, D., Martínez-Montesinos, M., and Gómez-Skarmeta, A. F. (2007). A survey of electronic signature solutions in mobile devices. *Journal of Theoretical and Applied Electronic Commerce Research*, 2(3):94. http://www.jtaer.com/dec2007/ruiz_sanchez_martinez_gomez_p7.pdf.

Stamp, M. (2011). *Information security: principles and practice*. John Wiley & Sons.

Telia (2016). Mobile id. https://www.telia.ee/en/era/muud-teenused/mobiil-id.

World, M. I. (2015). What is mobile id? http://mobileidworld.com/what-is-mobile-id/.

Wu, X., Fan, Y., Zhang, X., and Xu, J. (2014). Research of eid mobile identity authentication method. In *International Conference on Trustworthy Computing and Services*, pages 350–358. Springer. http://dx.doi.org/10.1007/978-3-662-47401-3_46.

Zefferer, T. and Teufl, P. (2015). Leveraging the adoption of mobile eid and e-signature solutions in europe. In *Electronic Government and the Information Systems Perspective*, pages 86–100. Springer. http://dx.doi.org/10.1007/978-3-319-22389-6_7.

Estrutura e Funcionamento de um Sistema de Identificação Biométrico *Peer-to-peer* baseado no Protocolo Chord

Clayton G. C. Santos[1], Rafael T. de Sousa Júnior[1]
[1]ENE – Departamento de Engenharia Elétrica, UnB – Brasília, DF – Brasil
clayton.cova@gmail.com, desousa@unb.br

Abstract. *This paper presents a solution for sharing ISO/IEC 19794-2:2011 templates between distinct databases and Automated Fingerprint Identification Systems (AFIS). Using an interoperation infrastructure based on a peer-to-peer (P2P) architecture, the Chord protocol provides the distribution and location of biometric records on the network by means of its distributed hash table (DHT). Forming an AFIS network overlaying Chord, a P2P system is built for sharing biometric data over the network, enabling efficient, flexible and balanced distributed operations for searching and comparing fingerprint data. With the developed system, its validation is performed by simulating the interoperation among AFIS instances related to each State of the Brazilian Federation.*

Resumo. *Este artigo apresenta uma solução para compartilhamento de templates ISO/IEC 19794-2:2011 entre distintas bases de dados e sistemas AFIS. Utilizando uma infraestrutura de interoperação baseada em arquitetura peer-to-peer (P2P), o protocolo Chord efetua a distribuição e localização de prontuários na rede através da sua tabela de hash distribuída (DHT). Com a formação de uma rede de AFIS sobreposta ao Chord é estabelecido um sistema peer-to-peer que compartilha dados biométricos em rede, viabilizando uma maneira eficiente e flexível para localizar e confrontar impressões digitais de forma balanceada. Com o sistema desenvolvido, a validação das funcionalidades é feita por simulação da interoperação entre instâncias de AFIS vinculadas a cada Estado da Federação Brasileira.*

Palavras-chave: Identificação e autorização, sistemas automatizados de identificação, interoperabilidade, sistemas *peer-to-peer*.

1. Introdução

Considerando as prerrogativas e a autonomia das unidades membros da Federação Brasileira, a ausência de um protocolo de comunicação entre bases biométricas coloca em evidência a necessidade de uma alternativa tecnológica para a troca destas informações e viabilizar pesquisas entre bases de identificação criminal dos Estados. Sem uma via padronizada de comunicação a ser seguida, cada órgão de segurança adota hoje um procedimento distinto para recebimento de solicitações de informações e consultas de outros órgãos, usando procedimentos administrativos, manuais e burocráticos em sua maioria.

Destarte, uma vez definido um padrão de comunicação entre os órgãos de segurança destes Estados, a troca de informações biométricas de forma automatizada permitiria agilidade e eficiência no processamento de requisições entre governos, aumentando a capacidade de investigação dos seus órgãos de segurança e resolução de crimes. Tal formato maximiza o uso das informações biométricas e agrega eficiência na persecução criminal e à Justiça.

Consideradas tais condicionantes, este artigo apresenta uma solução distribuída para compartilhamento de dados e interoperação balanceada e justa entre distintas bases de dados e sistemas AFIS (*Automated Fingerprint Identification System*) heterogêneos, apresentando o sistema e a validação de suas funcionalidades.

Este artigo está organizado da seguinte forma, na Seção 2 apresenta-se a descrição geral da abordagem utilizada. Na Seção 3 é detalhada a arquitetura e são descritas as operações *peer-to-peer*. Na Seção 4 se valida o sistema, considerando os requisitos de prioridade e de balanceamento de carga nas operações de verificação e identificação. A Seção 5 apresenta as considerações finais e propostas de trabalhos futuros.

2. Abordagem Distribuída para Interoperação de AFIS

Uma forma de compartilhar as informações biométricas, oferecendo operações de verificação e identificação entre Estados seria a disponibilidade de um sistema para troca de informações entre os diferentes sistemas AFIS dos governos estaduais. Por motivos políticos e de gestão a organização destas bases de identificação precisaria ser moldada num formato não hierárquico, alinhando autonomia e garantia de independência de cada participante. Com a manutenção dos dados em suas próprias estruturas de tecnologia, servidores e ambientes de TI, o armazenamento dos dados biométricos do Estado estará sobre a gestão local e independente.

Para interoperar sistemas AFIS, compartilhando, por exemplo, bases de dados criminais, a proposta aqui apresentada visa conceber um sistema de localização de *templates*[1] biométricos em uma rede P2P utilizando o protocolo Chord do MIT [Stoica *et al*, 2001], permitindo a distribuição de um índice de pesquisa em uma estrutura lógica descentralizada. Também define uma maneira de manter a rede sustentável, escalonando as filas de confrontos biométricos entre os nós e atribuindo custos baseados em informações dos *peers* participantes.

Para um Estado operar tal sistema como nó, em um primeiro momento, implantado o cliente do AFIS, o participante da rede constrói uma sua base de dados de identificadores, alimentando uma nova tabela com os dados de registros existentes na sua base de dados criminais. Cada indivíduo existente no banco de identidades é mapeado para o ambiente em rede, recebendo uma chave *k*, ID único na rede *peer-to-peer* e identificador este baseado no valor numérico do CPF (Cadastro de Pessoas Físicas) ou RCN (Registro Civil Nacional), que associa o prontuário do indivíduo no banco ao Estado a que ele pertence. A partir desta difusão as funcionalidades da rede peer-to-peer são desenvolvidas para permitir a interoperação entre os inúmeros AFIS participantes (Figura 1).

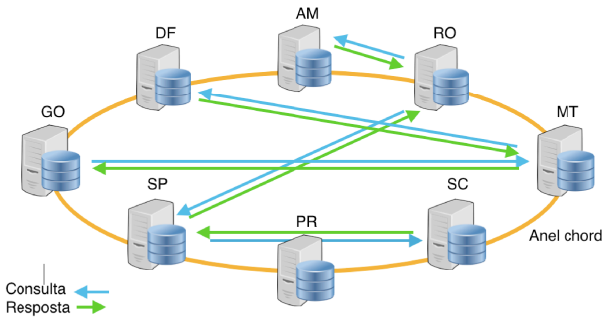


Figura 1. Estrutura lógica de comunicação da rede AFIS *peer-to-peer*.

1 *Template*: Arquivo contendo as coordenadas das minúcias e particularidades de uma impressão digital, utilizado nas operações de verificação e identificação de impressões digitais em sistemas AFIS.

2.1. Chord

Em uma arquitetura distribuída, sem uma organização hierárquica ou controle centralizado, a viabilização do compartilhamento de dados locais demanda de um formato descentralizado de comunicação e uma maneira eficiente de localizar os recursos em rede. No caso de *templates* biométricos, foco do presente estudo, trata-se de localizar o prontuário atribuído ao ID do indivíduo e permitir acesso a estes dados no servidor final.

O protocolo Chord em sua versão original efetua o mapeamento de uma dada chave para um determinado nó de rede. Seu modelo simplifica o design de sistemas e aplicações P2P atendendo as necessidades de balanceamento de carga, descentralização, escalabilidade, disponibilidade e flexibilidade na definição das chaves [Stoica *et al*, 2001]. Este último é de singular relevância para a aplicação aqui proposta, vez que os requisitos de um sistema distribuído em um modelo federativo incluem contar com uma maneira de localizar os recursos, baseando-se em dados de identificação dos indivíduos cadastrados. No caso do trabalho aqui tratado, foi utilizado o número de CPF de cada pessoa identificada para definição das chaves.

Em seu cerne, o protocolo Chord provê uma maneira rápida de mapeamento entre as chaves e os nós responsáveis por eles, usando o chamado espalhamento consistente (*consistent hashing)* para que um mínimo de nós da rede precise mover suas chaves mediante saída de um nó da rede. Com um mínimo de informações sobre seus *O(log N)* nós vizinhos a escalabilidade de uma rede com N nós é garantida, através da comunicação indireta entre os nós sequenciais.

A aplicação aqui proposta utiliza Tabelas de Hash Distribuídas e, para tanto, cada nó entrante na rede recebe uma chave, ou seja, um *hash SHA-2* que é produzido a partir da concatenação do FQDN (*Fully Qualified Domain Name*) do nó com a respectiva porta de operação do Chord. Ao passo que outros nós se registram no anel, tais nós recebem chaves semelhantes, baseadas em suas configurações de nome e porta, posicionando-se conceitualmente em uma sequência circular definida como *identifier circle*, um anel lógico de identificação dentro da extensão de chave de *m* bits. Para cada chave inserida no anel define-se o nó sucessor para esta chave, o nó que receberá a entrada e deverá estar posicionado adiante (no sentido horário) ou na posição da chave recém-inserida (Figura 2).

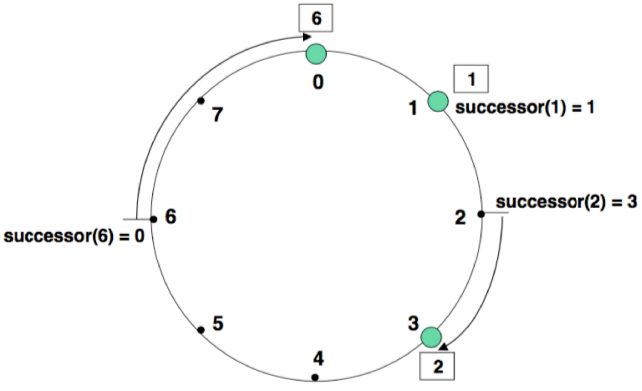


Figura 2. Anel lógico de identificação com *m* igual a *3 bits*, estando os nós 0, 1 e 3 inseridos na rede. Neste exemplo a chave 1 é posicionada no nó 1, a chave 2 no nó 3 e a chave 6 no nó 0, sempre buscando o nó sucessor à chave inserida [Stoica *et al*, 2001].

No momento em que é estabelecido e normalizado o conjunto de templates e IDs para o banco de identidades local, o módulo AFIS desenvolvido insere estes registros na rede Chord.

É feito o cálculo de *hash* dos respectivos números de CPFs dos prontuários existentes e propagadas estas informações em conjunto com a identificação de origem para os demais nós na rede. Esta propagação distribuirá as chaves pelos diversos nós do anel Chord, permitindo que qualquer outro nó participante possa pesquisar por um CPF específico e localizar este a partir da sua respectiva chave.

A proposta é que o protocolo Chord suporte o índice de pesquisa, permitindo maior flexibilidade na operação dos nós independentes, mantendo de forma distribuída a listagem dos CPFs e prontuários presentes no anel. Na medida que outros nós entram ou saem deste ambiente de comunicação, os dados de localização das bases estaduais são propagados para os nós vizinhos, garantindo a manutenção do índice de pesquisa em caso de falha na comunicação ou desconexão.

A ocorrência de uma colisão de *hash* de um mesmo ID em dois ou mais nós pode ocorrer, representando um CPF com prontuários em nós distintos (uma pessoa com identidades em Estados distintos). Esta exceção é tratada ao se criar uma lista encadeada a partir do registro já existente. O prontuário inserido passa a ser referenciado pelo registro mais antigo na rede, indicando os nós que possuem prontuários para um mesmo ID.

Estando os IDs propagados pelos nós na rede, a pesquisa de um determinado prontuário se dará da forma como estabelecida no protocolo Chord, por meio do número CPF o nó de origem processa a informação solicitada e busca com auxílio da sua *fingertable* o host sucessor ao valor da chave k, correspondente ao *hash* do CPF pesquisado.

A busca de um nó sucessor dentro da rede é feita consultando a *fingertable* armazenada em cada nó, uma tabela com m bits de entrada, onde a primeira entrada é o nó vizinho imediato à origem e as demais são nós posicionados no dobro da distância de cada entrada subsequente presente. Este recurso do *Chord* viabiliza uma redução significativa do custo de pesquisa, encurtando à metade a distância para a chave alvo. Recebendo a resposta da rede, relativa a chave consultada, o nó de origem recebe a resposta informando em que nó na rede o *template,* referente ao CPF pesquisado, esta armazenado.

Definido o servidor proprietário do prontuário relativo ao CPF buscado, o nó de origem aciona o módulo AFIS de comunicação entre servidores, enviando o *template* para ser pesquisado no destino, na base de *templates* do proprietário. A pesquisa executada consistirá em uma operação de verificação entre o *template* enviado e o *template* constatado no destino. Este modelo salvaguarda a autonomia dos participantes da rede com o processamento dos confrontos solicitados, processados pelo proprietário da informação, executando a propagação posterior somente dos dados compatíveis com o *template* apresentado. Com uma rede *peer-to-peer* operante, os IDs de todos os prontuários armazenados nos nós participantes ficam disponíveis para serem pesquisados através das chaves distribuídas nos nós, entretanto o acesso ao prontuário dependerá da disponibilidade do Servidor proprietário no anel.

Apesar de o modelo aqui apresentado não propor executar a carga direta dos *templates* biométricos, para a distribuição destes dados na rede *P2P,* a aplicação pode ser modificada de forma a permitir anexar tais conjuntos de dados biométricos, permitindo, por exemplo, encaminhar e disponibilizar no anel Chord os *templates* dos usuários cadastrados junto aos índices da rede *Chord*. A decisão de não executar tal carga de *templates* teve como determinante o tempo de replicação em sistemas com bases muito grandes. O volume de dados de *templates* a serem compartilhados poderia comprometer a eficiência da pesquisa em rede diante das limitadas condições de conexão e infra-estrutura de muitos órgãos de segurança dos Estados, especialmente aqueles com menor orçamento para investimentos na área de tecnologia.

Apesar do exposto, a disponibilidade de compartilhamento do próprio conjunto de *templates* junto à chave fica disponível como um campo junto à chave *k*, o que possibilitaria agilidade em confrontos de verificação ou identificação com as mais diversas estratégias. Um detalhe a mais nesta forma de execução é o fato de que o nó que armazena a chave com o conjunto de *templates* não possui informação que indique a quem pertence esse conjunto de dados, ofuscando a origem do *template* uma vez que a chave é o resultado do *hash* do ID único, no caso aqui em tela, o CPF.

Questões como autenticidade, sigilo e confiabilidade no canal de comunicação não foram tratadas no protótipo e tangenciam o escopo do artigo. Entretanto, podem ser atendidos com protocolos e *APIs* criptográficas assimétricas por exemplo. A distribuição de carga de processamento e pesos nas consultas são questões tratadas nas seções seguintes.

3. Operações *peer-to-peer* no Sistema Proposto

Nesta seção, são descritas as operações projetadas e desenvolvidas para a interoperação entre diferentes AFIS em modelo *peer-to-peer*, incluindo a inicialização da rede sobreposta, as trocas de dados de identificação dos nós, o intercâmbio de *templates* de informações biométricas, bem com as operações de verificação de identidades em confrontos 1:1 e 1:n.

3.1. Protocolo de comunicação

Para o estabelecimento do índice de pesquisa foi escolhido o protocolo Chord devido a suas características já apresentadas acima. A implementação do protótipo utilizado nos testes foi executada com a adaptação e adição de funcionalidades ao código do projeto *Nchord* [Cencini, 2009], modificado durante este estudo para carga de *templates* e metadados, que disponibiliza as funções *peer-to-peer* para a formação de um anel Chord e a propagação de chaves criadas pelo módulo de software desenvolvido neste trabalho.

A comunicação entre os nós se dá por meio de um módulo de comunicação do protótipo AFIS, sobre o protocolo TCP/IP, utilizando as informações de nome do host em conjunto com a porta de operação da aplicação para conexão entre um novo *peer* e um nó inicial da rede Chord. Estabelece-se um canal de comunicação entre um novo nó e o nó em operação. Ao passo que outros nós adentram ao anel Chord e se conectam a um dos nós, as chaves de cada nó são propagadas para os demais nós participantes, efetuando a distribuição do índice de pesquisa.

Para a troca dos dados de identificação entre os nós da rede, cria-se um canal de comunicação entre o cliente nó de origem, e o servidor nó de destino. Para tanto, utilizam-se os nomes de domínio configurados em cada nó, associados à porta de comunicação. A participação de um nó na rede de servidores entre os Estados poderia ser definida mediante termo de cooperação entre os entes participantes, por motivos de interesse entre as partes, definindo parâmetros de segurança e procedimentos administrativos. Com o canal de comunicação estabelecido, é efetuado o envio do *template* ao nó proprietário do prontuário a ser pesquisado, obtendo como retorno o *template* compatível ou uma mensagem de não correspondência.

O formato utilizado na transmissão de *templates* segue os padrões ISO, XML e binário compacto, este último nativo do projeto utilizado para confronto de *templates*, a solução SourceAFIS, base do protótipo AFIS utilizado, detalhado em Santos et al (2015). Poderão ser implementadas em trabalhos futuros formatações adicionais como estruturas JSON ou de objetos mais complexos, também baseadas em *Web Services*.

Seguindo a transmissão do *template* encaminhado, uma vez recebido no nó de destino, proprietário do prontuário, o processo AFIS deste nó insere o *template* na fila de processamento correspondente, atribuindo um identificador e um peso à entrada, quando for o caso.

3.2. Processamento de *templates*

Ao estabelecer-se como nó da rede no AFIS *peer-to-peer*, o ente participante inicia a função interna do nó local processando toda a sua base biométrica, gerando os respectivos *templates* no formato ISO/IEC 19794-2:2011.

Caso a base de *templates* já exista no formato em questão, ou for possível a criação de uma *view,* estando o conjunto dos dados normalizado, para cada prontuário cadastrado no sistema é utilizado um ID único associado ao indivíduo, CPF ou título de eleitor por exemplo. No software implementado durante os estudos utilizou-se o número do CPF como ID.

Os IDs atribuídos para cada prontuário do banco serão inseridos no índice de pesquisa do Chord, por meio do cálculo do *hash* do valor do ID, gerando uma chave k e sendo distribuído entre os nós do anel para viabilização da pesquisa e localização dos prontuários.

Em cada nó, ao passo que as solicitações de verificação ou identificação são recebidas, estes registros e *templates* são persistidos em banco de dados, em uma tabela utilizada para armazenar os *templates* recebidos, a serem pesquisados. Caso o *template* questionado possua correspondência atestada em algum sistema AFIS dos nós participantes, este *template* é associado ao prontuário compatível, passando a integrar a base de dados biométrica nos nós de origem e destino.

3.3. Confronto de verificação

Conforme mostra a Figura 3, em um processo de verificação de *template* (confronto 1:1), quando os nós existentes executam pesquisas na rede, ao localizar um nó proprietário para um determinado ID, o nó solicitante estabelece um canal de comunicação TCP com o nó proprietário e envia uma mensagem com o ID a ser confrontado e o *template* de referência (questionado), a ser confrontado na base remota.

Após o recebimento correto do *template* no módulo do destino o servidor proprietário insere este *template* na fila de pesquisa de verificação. Na aplicação desenvolvida, a fila consiste em uma tabela identificada com o nome “fila_ver” no MySQL, SGBD utilizado pelo protótipo. Após persistir tais dados, o *template* questionado é confrontando por meio do algoritmo SourceAFIS [Važan, 2012] com o *template* correspondente no prontuário (padrão), correspondente ao ID especificado na mensagem do nó de origem.

O módulo AFIS do nó proprietário atribui um identificador temporário para esta nova entrada, registra a ocorrência em um log de operações e processa a entrada recém-inserida na chamada fila de verificação. Dentro desta fila de processamento, a ordem de chegada, em conjunto com um peso, detalhado mais adiante, atribuído ao nó solicitante, será utilizada para organização das prioridades na execução dos confrontos. Sequencialmente e sob demanda, o processo AFIS do servidor efetua o processo de verificação de cada entrada na fila de verificação, confrontando os *templates* recebidos (questionados) da rede com os *templates* locais (padrões), disponíveis em sua base de dados.

Após persistir os dados do *template* recebido, o nó proprietário responde ao nó de origem enviando o conjunto de dados relativos ao prontuário compatível, da seguinte maneira: confrontados os *templates*, o servidor proprietário armazena o resultante da operação em um *log* do sistema.

Ocorrendo o *matching* (correspondência) no registro processado, o *template* confrontado será então associado ao prontuário do indivíduo ao qual ocorreu a positivação, com uso de uma chave estrangeira criando uma relação para ligação com a tabela de prontuários do nó proprietário. Sendo negativa a correspondência entre os *templates*, o servidor proprietário descarta o *template* enviado inicialmente e retorna uma mensagem de não correspondência para o nó de origem.

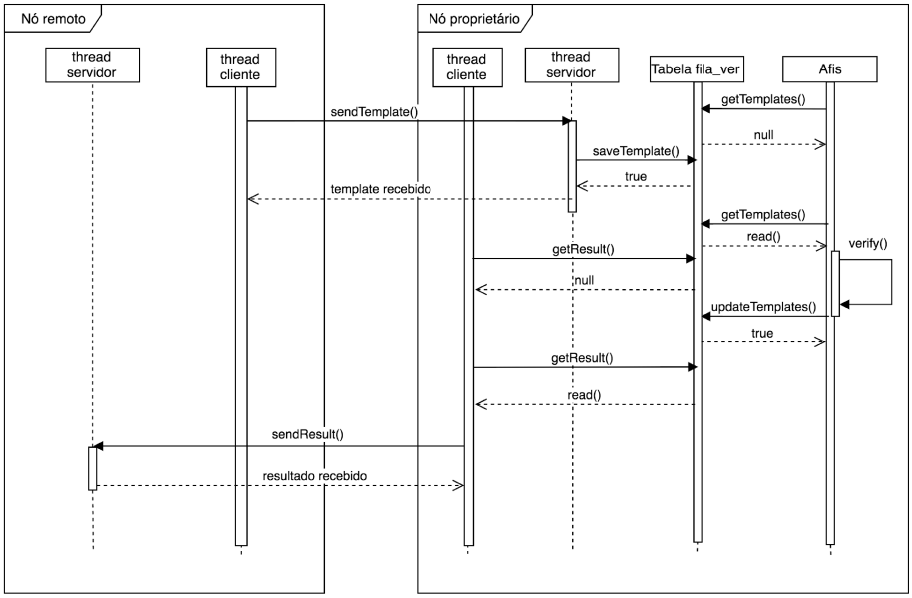


Figura 3. Sequência lógica de comunicação entre os nós remoto e proprietário na operação de verificação da rede AFIS *peer-to-peer*.

3.4. Confronto de identificação

Na ocorrência de um processo de identificação de *template* (pesquisa 1:n), aquele em que o nó participante solicita o confronto de um *template* com a totalidade da base de dados de um nó proprietário, o nó originário da pesquisa busca o ID do nó destino na rede. Envia na sequência uma solicitação de pesquisa genérica, em vez de informar o ID do prontuário a ser confrontado, ele informa o ID do nó proprietário.

Recebido o *template* da solicitação com sucesso, ao perceber a requisição genérica por meio do uso de seu ID do nó, o módulo AFIS atribui um identificador temporário ao *template*, registra o recebimento em um log de operações e armazena esta entrada em uma fila de processamento distinta, chamada fila de identificação "fila_id". Nesta fila de processamento, o conjunto do peso do nó solicitante e a ordem de chegada do item na fila é o que determina a sequência de processamento e de submissão do *template* à pesquisa 1 para n. De forma escalonada e assíncrona, o processo AFIS segue a fila de prioridades buscando em sua base de *templates* aquele compatível com o registro testado. Após o envio do *template* por parte do nó de origem, o servidor proprietário responde à origem com uma mensagem de sucesso no recebimento do arquivo e encerra a conexão.

O *template* enviado, presente na fila de identificação, aguarda o processamento das entradas com maior prioridade e na sequência é submetida ao confronto 1:n. Após a conclusão do processamento desta entrada, o módulo de comunicação do AFIS acessa o nó de remetente do *template* testado e retorna o resultado da pesquisa.

Caso o resultado do confronto de identificação retorne *matching* com um *template* compatível, o nó proprietário associa o *template* testado com uma chave estrangeira ao prontuário correspondente e envia na resposta ao nó de origem com os dados deste prontuário, garantindo assim a identificação do *template* testado.

Caso o resultado seja negativo, o nó proprietário descarta o *template* processado e retorna uma mensagem de não correspondência para o nó de origem. A sequência lógica de operação para o caso do processo de identificação é semelhante ao de verificação, com as mesmas duas *threads* cliente e servidor em cada nó participante.

Entretanto, é utilizada uma tabela distinta como fila de identificação, definida como "fila_id" no MySQL, e também o processo de confronto de *templates* executa a operação da forma 1:n, confrontando o *template* questionado com os demais presentes no banco de *templates* do nó local por meio do metodo Identify().

Para efeito de validação através de protótipo experimental, todo o trabalho de processamento e confronto de *templates* foi efetuado utilizando uma solução de software baseada no projeto *SourceAFIS*, com o algoritmo adaptado para trabalhar com *templates* de teste, amostra da base de prontuários biométricos da Secretaria de Segurança do Estado de Rondônia, sistema apresentado em [Santos et al, 2015].

4. Métricas de Prioridade, Balanceamento nos Nós e Validação do Sistema

É previsível que existirão entre os nós participantes variações quanto às capacidades de processamento e memória [Karger *et al,* 2004], assim como diferentes níveis de robustez, significando que distribuir igualmente a responsabilidade não é necessariamente a melhor opção [Epsztejn *et al*., 2006]. Faz-se necessária então a definição de um formato justo de balanceamento nas atividades de verificação e identificação dos *templates* questionados junto às bases de identificação dos nós da rede Chord.

Dentre os parâmetros levantados nos *peers* da rede, com objetivo de manter uma justa distribuição do processamento dos *templates*, abordagens como a apresentada em Garofalakis et al. (2009) e Chawachat et al. (2014) definem estratégias para construção de métricas e custos para organização das filas de identificação, semelhante a algumas apresentadas em Mondal *et al*, (2003). No trabalho proposto são consideradas algumas das variáveis utilizadas nestas referências como índices de distribuição de carga, exemplos são: disponibilidade para requisições, custo para armazenar os dados, carga de processamento, etc.

A forma de organização das filas de processamento dos *templates* deve estabelecer parâmetros de priorização de solicitações, de maneira justa, mantendo a rede sustentável ao estimular nós da rede ofertarem seus recursos computacionais e dados biométricos. As filas de verificação e de identificação são tabelas utilizadas para organizar a chegada das requisições de consultas dos nós participantes da rede. Estes registros objetivam a organização e o escalonamento equilibrado do processo de confronto entre os *templates* questionados e os padrões do banco pertencente ao nó proprietário. Em um cenário com nós mais requisitados que outros é previsível a necessidade de balanceamento de carga, por exemplo um estado como São Paulo receberá mais requisições de confrontos do que um estado como Rondônia. Nesta conjuntura, torna-se imperativo um ajuste nas prioridades de processamentos para cada nova solicitação. Em ambientes AFIS são demandados capacidade de processamento, número de requisições, volume de *templates* encaminhados, número de registros no banco, etc. Tais variáveis são determinantes para a correta operação do sistema [Komarinski, 2005].

Conforme as requisições chegam no módulo de transmissão do AFIS, o registro recebido entra na fila correspondente ao tipo de confronto requisitado, verificação ou identificação, recebendo inicialmente o número da sua posição nela, ordem de chegada. Durante o período de operação do nó na rede, a ocorrência de consultas e interações com os demais entes participantes permite ao nó receptor coletar informações e número de operações entre nós, permitindo o levantamento do número de requisições, tamanho das bases de dados, solicitações e respostas de confronto com os nós contatados.

O sistema proposto utiliza um algoritmo de escalonamento de múltiplas filas, o processo dentro do AFIS implementado mantém organizada uma fila de verificação e outra de identificação. O processamento da fila de verificação representa um ônus menor para o sistema do que o processamento da fila de identificação. Devido à necessidade de resposta imediata em operações 1:1, entradas desta fila são processadas assim que recebidas, sempre que existirem entradas e com prioridade maior do que a fila de identificação. Dentro desta estrutura a ordem de chegada é definida como a ordem de processamento, seguindo o conceito de fila, FIFO (*first in, first out*).

Sabendo que o custo para o processamento da fila de identificação é maior devido a natureza da pesquisa de 1 para n, a organização da sequência neste caso é mais dispendiosa e deve considerar outra estratégia, favorecendo aqueles que atendem e respondem melhor à rede e mantendo a sustentabilidade do sistema. As seguintes variáveis são consideradas para o cálculo de prioridade considerando a demanda dos serviços AFIS:

- Ordem de chegada (OC): na sequência da ordem de chegada é atribuído um valor crescente ao item recebido para processamento;
- Disponibilidade do nó (DN): em cada tentativa de consulta de verificação ou identificação em outros nós, a estes são atribuídas pontuações, cumulativas.
- Número de requisições do nó (NQ): cada requisição representa um custo de operação que é atribuído ao nó solicitante, este valor também é cumulativo;
- Número de respostas do nó (NR): em consultas a outros nós na rede, as respostas fornecidas por estes nós constituem pontuação para o nó consultado. Valor cumulativo.
- Número de prontuários disponíveis no nó (NP): o número de itens em sua base sendo disponibilizado para consulta representa pontuação para o nó participante. Valor proporcional ao tamanho da base.

Através da fórmula apresentada em *(1)*, calcula-se a prioridade para cada item da tabela, considerando nesta função os demais pesos, quando disponíveis. O menor o valor de custo representará maior prioridade de processamento.

$$\textit{Custo:}\quad \mathrm{OC} - \left(\frac{1}{2} * \mathrm{DN}\right) - \left(\frac{1}{2} * \mathrm{NR}\right) + \left(\frac{1}{10^2} * \mathrm{NQ}\right) - \left(\frac{5}{10^6} * \mathrm{NP}\right) \quad (1)$$

A fórmula proposta foi concebida a partir da avaliação das variáveis de controle na função objetivo, com foco em definir como resultado um custo menor para os nós que disponibilizam mais seus recursos na rede. A definição dos coeficientes na equação foi feita considerando a proporção entre a Ordem de Chegada e as demais variáveis, a razão entre o número de requisições, disponibilidade, consultas e tamanho da base de dados foram ajustadas para influenciar proporcionalmente o resultado final do custo. Cada variável na fórmula linear possui uma constante que representa o peso desta variável na função.

Foi avaliado o número de templates a serem processados nos nós participantes em uma aplicação real, com fundamento em dados reais. No caso do Estado de São Paulo, o maior em números populacionais segundo dados oficiais da Secretaria de Segurança Pública do Estado no ano de 2015. Neste ano foram cadastradas cerca de 20 milhões de pessoas, valor próximo a 48% da população do Estado que é de 41.262.199 segundo dados projetados para o ano pelo IBGE [SESP, 2014][IBGE, 2010]. Uma base de prontuários com esse volume de dados proporcionalmente terá uma carga maior de solicitações a ser atendida pelos nós da rede, o que demanda um custo mais elevado para o processamento destas requisições.

Ao se exigir uma maior infraestrutura para suportar as operações no sistema AFIS, define-se um menor custo para o processamento de solicitações do nó São Paulo, proporcionalmente definido com influência do tamanho se sua base de *templates*. Para o cálculo do custo, é considerada a ordem de chegada (OC) com um fator de peso maior, seguida da disponibilidade do nó (DN), número de respostas (NR), como fatores que diminuem o custo. O número de requisições (NQ) também é utilizado, mas para incrementar o custo com peso relativo menor, é necessário neste caso um alto índice de requisições do nó para alterar influenciar o aumento do custo.

Didaticamente, considerando a ordem de chegada como custo inicial, ficou estabelecido para esta aplicação que, a cada 2 pontos de Disponibilidade do Nó, 1 ponto é subtraído do custo. Da mesma forma, a cada 2 pontos de Número de Respostas atribuída ao nó, mais 1 ponto é subtraído do custo. Na sequência, 100 pontos no Número de Requisições do Nó representarão um ponto a mais na Ordem de chegada, aumentando o custo nesse caso. Ao final, considerando o número de prontuários disponibilizados pelo nó avaliado, a cada 2 milhões de prontuários disponibilizados, um ponto vai ser subtraído do custo. Os valores inteiros apresentados foram utilizados a fim de facilitar a ilustração, mas no algoritmo da fórmula em questão, os resultados do custo para cada valor inserido será considerado, ainda que num formato decimal.

Tais valores coeficientes foram selecionados após ciclos de testes com simulações de interações entre nós, randomizando os valores DN, NQ, e NR, em condições semelhantes ao de operação real, avaliando-se sempre o rearranjo das posições das requisições dentro da fila de prioridades. Após testes e ajustes, ao perceber um escalonamento justo das entradas processadas, baseando-se na proposta e nos princípios acima definidos, de forma empírica, a fórmula foi escolhida.

Tabela1. Apresenta uma fila de identificação simulada, para as métricas definidas na aplicação proposta, ordenadas neste caso pelo valor do Custo.

Nó de Origem	Ordem	Custo	DN	NQ	NR	NP
PA	11	-57,87	634	701	523	3638904
MS	17	-52,2	601	492	475	1175532
PR	13	-52,06	547	744	485	5013372
AL	2	-51,72	519	77	506	1497837
RR	6	-49,37	598	289	405	216230
MA	23	-44,34	509	185	396	3155899
RS	20	-43,09	539	673	325	5133086
DF	1	-33,96	454	305	221	1233677
PE	19	-32,4	390	359	261	4222295
CE	8	-32,08	530	351	94	4057143
BA	12	-31,12	331	299	254	6728115
AC	7	-30,64	470	291	159	352108
PI	21	-28,71	484	43	118	1496813
RJ	24	-26,32	366	314	138	7675166
MT	18	-25,82	508	361	37	1456859
SP	22	-21,25	187	599	96	19805856
TO	14	-20,11	283	726	155	664054
PB	26	-19,79	321	562	120	1807933
ES	10	-19,39	238	302	159	1687177
MG	15	-13,38	134	570	81	9406718
SC	5	-11,48	148	373	69	2999249
AP	25	-10,92	159	344	113	321372
RN	3	-8,55	105	360	64	1520653
AM	9	-4,1	58	241	30	1672313
GO	4	-4	40	745	34	2881818
SE	27	-3,85	77	642	57	992648

A partir da prioridade, resultado da função formulada em *(1)*, a lista de identificação é definida e cada entrada na lista é processada, submetendo o *template* encaminhado pelo nó de origem ao confronto com todos os outros *templates* do banco local. A Tabela 1 é resultante de uma simulação, pela inserção de 26 entradas em uma fila de identificação com aplicação da fórmula proposta, representando solicitações de confronto dentro do sistema AFIS, uma solicitação para cada um dos nós, representando estados brasileiros e processados pelo nó RO.

Para fins de teste o valor relativo ao número de prontuários de cada nó foi definido como a parcela de 48% do montante populacional de cada Estado, seguindo a mesma taxa de cadastramento biométrico do Estado de São Paulo e considerando dados populacionais dos estados segundo censo do IBGE [IBGE, 2010]. Para cada estado, considerando seus tamanhos de bases de dados, foram produzidos valores aleatórios dentro de uma faixa proporcional para a disponibilidade do nó (DN) e para o número de requisições (NQ). O número de respostas (NR) foi definido dentro da faixa do valor de disponibilidade do nó (DN), representando as respostas recebidas das consultas.

Analisando os valores resultantes podemos observar que, na forma aqui proposta, somente a Ordem de Chegada não é suficiente para definir a prioridade de processamento, visto que os nós com grandes bases de prontuários e *templates* terão prioridades de processamento influenciadas pela sua ativa participação na rede. Um exemplo é o caso das entradas dos estados DF e MS, em que a entrada do MS (Tabela 1) recebeu melhor custo apesar de ter entrada depois do registro atribuído ao DF na fila de processamento. Com melhores valores de disponibilidade e mais que o dobro do número de respostas à rede, a entrada do MS é privilegiada com o cálculo de custo considerando também estas variáveis. É possível observar na simulação que o tamanho da base disponibilizada, associado com a participação ativa do nó na rede, são fatores que melhoraram a posição das entradas de processamento e garantiram uma organização mais justa de posicionamento de entradas de confronto na fila AFIS de cada nó.

5. Conclusão

O trabalho aqui descrito apresenta a concepção de um sistema de localização, compartilhamento e confronto de *templates* biométricos em uma rede *peer-to-peer*, utilizando o protocolo Chord para distribuição dos identificadores. Também apresenta uma proposta de escalonamento para filas de confronto baseada em custos, com métricas e pesos calculados para cada entrada na fila, considerando parâmetros como ordem de chegada, disponibilidade, número de requisições, número de respostas e tamanho da base de *templates*.

Tal solução é percebida como uma alternativa para confronto entre *templates* de nós distintos, podendo ser utilizado em consultas de verificação (1:1) ou pesquisas de identificação (1:n) entre *peers*. Esta aplicação, uma vez finalizada, poderia integrar Estados brasileiros, permitindo consultas entre bases biométricas e possibilitando a identificação de indivíduos registrados em outras unidades da federação.

Foi possível constatar através de simulações entre os nós, compondo filas de identificação aleatoriamente, que a fórmula proposta permite um reordenamento dos itens recebidos de maneira satisfatória, escalonando o processamento dos *templates* no sistema AFIS dos nós participantes. O protocolo Chord se mostrou eficiente para obter a localização dos *templates*, mantendo um índice de pesquisa com base nos CPFs dos prontuários existentes em cada nó da rede. Fica proposta para trabalhos futuros a avaliação do impacto da carga dos dados biométricos junto ao índice de pesquisa do Chord, permitindo o compartilhamento de *templates* no formato ISO/IEC 19794-2:2011 diretamente no *peer* da rede. Poderia ser avaliado também o aspecto de segurança desta abordagem, que em tese manteria a privacidade dos prontuários compartilhados.

Agradecimentos

Os autores agradecem às Agências brasileiras de pesquisa, desenvolvimento e inovação CAPES (Projeto FORTE 23038.007604/2014-69) e FINEP (Projeto RENASCI/PROTO 01.12.0555.00), bem como ao DITEC/DPF/MJ (Contrato 36/2010 DPF-FUB Mestrado Profissional em Informática Forense), à DIPLA/MP (TED 005/2016 DIPLA-FUB), pelo suporte a este trabalho.

Referências

Ayyasamy, S. e Sivanandam, S. N. (2010) A Cluster Based Replication Architecture for Load Balancing in Peer-to-Peer Content Distribution, International Journal of Computer Networks & Communications (IJCNC) Vol.2, No.5.

Cencini, A. (2009) NChord 1.0.1.0, Technical Documentation and Tutorial.

Chawachat, J. e Fakcharoenphol, J. (2014) A simpler load-balancing algorithm for range-partitioned data in Peer-to-Peer systems. Department of Computer Engineering Kasetsart, University Bangkok, Thailand.

Epsztejn, G., Carlos, O., Duarte, M. B. (2006) Proposta De Rede P2p Organizada Por Índices, Grupo de Teleinformática e Automação, Universidade Federal do Rio de Janeiro, UFRJ, Rio de Janeiro, Brasil.

Garofalakis, J., e Michail, T.A. (2009) Load Balancing in a Cluster-Based P2P System, Proceedings of Fourth Balkan Conference in Informatics, 133-138, Thessaloniki Greece.

IBGE. Censo Demográfico 2010 - Resultados do universo. (2010) Disponível em: http://www.censo2010.ibge.gov.br/sinopse/index.php?dados=4&uf=00, Acesso em: 02/06/2016.

Karger, D. e Ruhl, M. (2004) Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems, Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures, pp. 36-43.

Komarinski, P. (2005). Automated fingerprint identification systems (AFIS). Amsterdam, Elsevier Academic.

Mondal, A., Goda, K., e Kitsuregawa, M. (2003) Effective load-balancing of peer-to-peer systems. Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid, pp.81-88.

Patange V., Gatade D. D. (2013) Survey of Load Balancing Approaches in Peer-To-Peer Network, International Journal of Soft Computing and Engineering (IJSCE), Volume-3, Issue-2.

Santos, C. G. C., Sousa Jr., Rafael T. de, Deus, F. E. G (2015). Prototipação e validação de um AFIS de código aberto utilizando a Base de Identificação Criminal do Estado de Rondônia. Anais do WGID - V Workshop de Gestão de Identidades Digitais, 2015, XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais SBSeg 2015, v. 1. pp. 575-586

Stoica, I., Morris, R., Karger, D., Kaashoek, M., e Balakrishnan, H. (2001) Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 1-12.

SESP - Secretaria de Estado da Segurança Pública. (2014) São Paulo ganha nova carteira de identidade, Governo do Estado de São Paulo, http://www.ssp.sp.gov.br/noticia/lenoticia.aspx?id=33409, acesso em 02/06/2016.

Važan, R. (2012) SourceAFIS Tutorial, Wikibooks.org, https://en.wikibooks.org/wiki/SourceAFIS/Tutorial.

Serviço de Gerenciamento de Organizações Virtuais entre Federações SAML

Maykon Chagas de Souza, Jucélio Jair Silva, Michelle Silva Wangham

[1]Universidade do Vale do Itajaí – (UNIVALI)

{mchagas,jucelio}@edu.univali.br, wangham@univali.br

Abstract. *Collaborative environments are based on the establishment of virtual organizations (VOs) and have specific rules and policies to allow access to researchers. The use of federated identity model simplifies management identity to the user and provides single sign-on authentication. However, to create VOs that go beyond the barriers of a Federation, it is necessary to establish trust relationship among the domains of different Federations.This work aims to describe a service responsible for (1) providing a trust relationship establishment between the VO providers that are in different SAML federations and (2) supporting the authentication and authorization of VO users.*

Resumo. *Ambientes colaborativos são baseados na formação de organizações virtuais (OVs) e possuem regras e políticas específicas para permitir o acesso dos pesquisadores. O uso do modelo de identidades federadas simplifica para o usuário a gestão de identidades, proporcionando a funcionalidade de autenticação federada e única (SSO). No entanto, para a formação de OVs, que ultrapassam o domínio de uma única Federação, é necessário o estabelecimento de relações de confiança entre os domínios de diferentes Federações. O objetivo deste trabalho é descrever um serviço responsável por (1) prover o estabelecimento das relações de confiança entre os provedores usados em uma OV formada por domínios administrativos que estão em diferentes federações SAML, e (2), contribuir com a autenticação e autorização dos usuários da OV.*

1. Introdução

Universidades, institutos de pesquisas e empresas estão gerando uma grande quantidade de dados que precisam ser acessados através de ambientes colaborativos de pesquisa que ultrapassam os domínios de uma única organização [Broeder et al. 2012]. Instituições, normalmente geograficamente distribuídas, se conectam através da Internet e estabelecem relações de confiança entre si para desenvolverem pesquisas colaborativas, pesquisas estas chamadas de *e-science* [Schroeder 2008].

Nestes ambientes colaborativos, nos quais pesquisadores fazem uso de recursos computacionais distribuídos, as relações de confiança estabelecidas devem permitir a interação entre pesquisadores que não participam de uma mesma instituição (mesmo domínio administrativo), mas que têm interesse em compartilhar recursos e informações sobre uma determinada linha de pesquisa ou projeto [Zhang et al. 2012]. Como resultado deste cenário, tem-se um grupo sem fronteiras que atua como uma rede de pessoas e instituições conectadas que trabalham juntas com o objetivo de resolver problemas complexos e fazer ciência. Na literatura, este grupo é chamado de Organização Virtual (OV) [Gemmill et al. 2009, Foster et al. 2001].

Um ponto chave para os ambientes colaborativos das OVs é prover a gestão de identidades (GId) por meio da criação de um sistema de identificação, de autenticação e de autorização de usuários. Estes ambientes precisam ser protegidos contra acessos não autorizados. É preciso definir quem tem acesso a quais recursos e em quais circunstâncias e também definir quem terá permissões para gerenciar as regras de acesso de outros usuários [Foster et al. 2001].

A Gestão de Identidade (GId) pode ser entendida como o conjunto de processos e tecnologias usados para garantir a identidade de uma entidade, garantir a qualidade das informações de uma identidade (identificadores, credenciais e atributos) e para prover procedimentos de autenticação, autorização e auditoria [ITU-T 2009]. Para prover a GId, é necessária a construção de um sistema integrado de políticas e processos para validação e troca de credenciais entre os envolvidos, além das definições, certificação e gerenciamento do ciclo de vida das identidades digitais que permitam o tratamento e manipulação de identidades (atributos de identidades) [Jøsang et al. 2005, Chadwick 2009].

Dentre os modelos de GId utilizados em ambientes colaborativos, destaca-se o de identidades federadas, no qual uma federação é uma forma de associação entre instituições parceiras (domínios administrativos) de uma rede colaborativa que usa um conjunto comum de atributos, práticas e políticas para trocar informações e compartilhar serviços, possibilitando a cooperação entre os membros e usuários da federação [Carmody et al. 2005]. A adoção de um modelo de GId federadas, tem por objetivo remover a complexidade do usuário, no que se refere a administrar um nome de usuário e senha para cada serviço que deseja acessar, permitindo que uma mesma identidade possa ser utilizada no acesso a diferentes serviços que podem estar em domínios administrativos diferentes [Jøsang et al. 2005, Bhargav-Spantzel et al. 2007].

Em pesquisas colaborativas (*e-science*), a utilização de GId federadas se mostra vantajosa, pois permite que os pesquisadores envolvidos não necessitem de um novo usuário para acesso aos recursos que serão compartilhados, possibilitando que os responsáveis pela OV adicionem somente as regras de acesso aos recursos, deixando a autenticação a cargo da instituição de origem de cada pesquisador. Motivados por essas necessidades, representantes de diferentes instituições têm se organizado para discutir como resolver os desafios relacionados a definição de uma política comum para gestão de identidades nas estruturas, federações e tecnologias existentes com o objetivo de desenvolverem pesquisas colaborativas [Broeder et al. 2012].

Este trabalho tem por objetivo descrever um serviço de gerência de segurança de OVs, que visa auxiliar as etapas do ciclo de vida de uma OV, em especial, que possibilite o estabelecimento das relações de confiança entre provedores que estão em federações SAML distintas e contribua com a autenticação e autorização em ambiente colaborativos. De forma a avaliar a aplicabilidade do serviço proposto, um protótipo foi desenvolvido e um cenário de uma OV foi concebido para os experimentos de avaliação.

Este artigo está organizado em cinco seções. Na Seção 2, são apresentados os conceitos referentes a OVs, *e-science* e gestão de identidades federadas. A solução proposta (serviço de gerência de segurança para OVs) é descrita na Seção 3. Na Seção 4, os trabalhos relacionados são analisados e comparados com o serviço proposto. Por fim, a conclusão e os trabalhos futuros são apresentados na Seção 5.

2. Organizações Virtuais - OV

O responsável pelo programa de *e-science* do governo Inglês, John Taylor, define *e-science* como: "*uma forma de colaboração global em determinadas áreas da ciência e a infraestrutura que irá suportá-la*" [Taylor 2001]. Além disto, *e-science* utiliza ferramentas computacionais para a troca de informações, permitindo que pesquisadores possam compartilhar recursos com outros pesquisadores ou, em outros projetos, instituições e até entre diferentes áreas de pesquisa [Schroeder 2008].

A colaboração entre as diferentes disciplinas, áreas de conhecimento, não se aplicam somente às tecnologias mas também às instituições, nas quais as políticas de coordenação dos projetos de *e-science* ultrapassam as fronteiras de países e apresentam obstáculos para o processo, não só técnicos, mas organizacionais [Schroeder 2008]. Estas barreiras incluem políticas restritivas de acesso aos recursos computacionais das *grids* acadêmicas providas por um país, diferentes leis de direitos autorais e de propriedade intelectual, além de barreiras comerciais como a interconexão de redes seguras.

Nos ambientes colaborativos, um desafio apontado por [Foster et al. 2001] é o compartilhamento de recursos para resolução de problemas de pesquisa de forma dinâmica e multi-institucional. Para permitir este compartilhamento é necessário prover um ambiente controlado, definindo claramente quem é o provedor do serviço, o que é compartilhado, quem tem acesso a estes recursos e quem pode compartilhá-los. Uma Organização Virtual (OV) é um grupo formado por universidades, empresas e indivíduos que compartilham recursos e conhecimentos para alcançar um objetivo e que atuam por acordos estabelecidos [Foster et al. 2001].

Um dos desafios apresentados em uma OV é criar e gerenciar um ambiente seguro e federado entre domínios administrativos autônomos, de forma a garantir a separação entre o provimento, o gerenciamento da aplicação fornecida, o gerenciamento operacional da infraestrutura da OV, o descobrimento dos recursos disponíveis e o estabelecimento das relações de confiança [Capuano et al. 2010].

O ciclo de vida de uma OV é um processo composto dos seguintes estágios: (1) **criação**, identifica as competências necessárias para desenvolver um projeto de pesquisa, modela o projeto com base nestas competências e identifica os parceiros que melhor se enquadram neste projeto; (2) **operação**, visa a execução do projeto cooperativo de forma eficiente, sendo que os mecanismos de cooperação e as medidas de desempenho têm papel importante neste estágio; (3) **evolução**, permite uma pequena alteração ou redistribuição de competências entre os membros[1]; e (4) **dissolução**, trata da dissolução das relações de cooperação estabelecidas para operação da OV.

Nos ambientes de *e-science* e OVs, o conceito de identidades federadas tem se popularizado justamente pelo fato de permitir maior flexibilidade no uso das identidades dos usuários e no gerenciamento destas pelos administradores de serviços que participam da Federação. Para prover estes ambientes de federação, o SAML tem se mostrado o protocolo mais utilizado, uma vez que possui uma especificação robusta e que foi projetada com o intuito de atender as necessidades para a formação e gerenciamento de Federações. Outros protocolos também podem ser usados em Federações, como o *WS-Federation* e o OpenID Connect.

[1]Mudanças em objetivos ou mudanças de muitos parceiros levam a uma nova formação.

3. Serviço de Gerência de Segurança para Organizações Virtuais

Para gerenciar o ciclo de vida de uma OV e fazer uso dos benefícios das identidades federadas e da transposição de autenticação e de atributos para outros domínios fora da federação de origem, foi preciso neste trabalho enfrentar as seguintes questões de pesquisa: como estabelecer as relações de confiança quando os membros da OV (instituições e pesquisadores) não fazem parte da mesma federação? E ainda, quando se trata de relações entre federações que utilizam diferentes sistemas de GId (baseados em SAML), como realizar a transposição de autenticação e de atributos?

O serviço proposto de gerência de segurança atua como um *proxy*, uma terceira parte confiável, que intermedia os acessos de usuários entre os domínios administrativos das instituições que participam do ambiente de pesquisa colaborativa (*e-Science*). Este serviço está baseado na especificação SAML e estende o conceito de autenticação federada ao possibilitar que os membros da OV estejam em federações distintas. A solução proposta contempla ainda a definição de um metadado de atributos comuns para ambientes de *e-Science*, para que o mapeamento dos atributos dos usuários entre os diferentes domínios administrativos possa acontecer.

A habilidade de atuar como um *proxy* possibilita que o serviço se comunique com diferentes entidades (IdPs ou SPs) que podem estar em federações SAML distintas. Para isto, o serviço estabelece as relações de confiança com as entidades de forma independente (não com toda a federação que esta entidade pertence). Ou seja, um IdP ou um SP que pertence a uma Federação pode possuir uma relação de confiança direta com o serviço. Para exemplificar esta funcionalidade, a Figura 1 ilustra uma OV formada pelos domínios A, C e F de diferentes federações. Os IdPs A e C da Federação A, o IdP F da Federação B e os SPs 1 e 2 possuem relações de confiança com o serviço, o que possibilita que os usuários da OV que estão nestes IdPs possam acessar serviços do SP1 e do SP5. O primeiro SP faz parte da Federação C e o segundo é um serviço independente, que não faz parte de nenhuma Federação, mas que participa da OV.

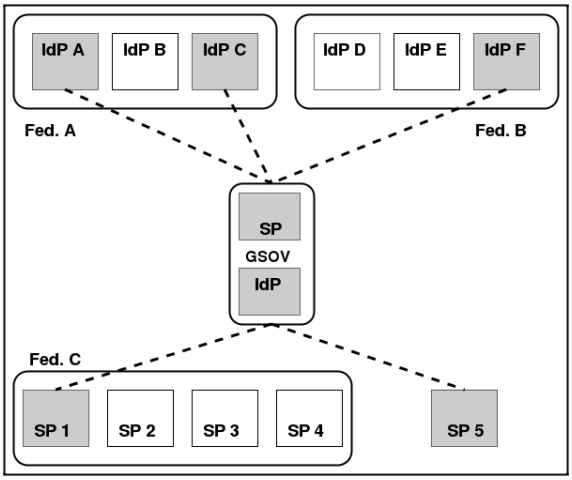


Figura 1. Visão geral do serviço proposto como Proxy

O serviço proposto não tem a necessidade de estar inserido em um ambiente de uma federação, e esta característica torna a formação de OVs mais flexível, pois permite que os envolvidos na pesquisa colaborativa estabeleçam a relação de confiança para a

formação da OV somente entre si. Na solução proposta, é possível ainda ter diferentes serviços gerentes de OV, para diferentes OVs, o que garante a escalabilidade e distribuição da solução. O serviço de gerência de segurança é capaz de estabelecer as relações de confiança entre as entidades de diferentes domínios administrativos que participarão de uma OV, permitindo que os atributos dos usuários sejam trocados entre as entidades da OV, tendo como base as relações de confiança e políticas de uso previamente estabelecidas entre os provedores participantes.

Com estas relações de confianças estabelecidas, a transposição de autenticação e de atributos ocorrerá dentro da OV e os provedores de serviços da OV poderão ser acessados de forma segura e transparente. Desta forma, o usuário não terá de se preocupar com detalhes específicos de como acessar os provedores de serviços ou com o gerenciamento de múltiplas identidades para cada SP que desejar acessar na OV.

O processo para estabelecimento de relações de confiança entre os participantes da OV não se difere muito do processo realizado para a entrada de um participante em uma Federação SAML. No entanto, diferentemente de outros trabalhos, como [Vullings et al. 2007] ou [Gemmill et al. 2009], em que as relações de confiança precisam ser estabelecidas com toda a federação, na solução proposta, o serviço deve ser uma entidade confiável apenas para os domínios envolvidos na OV, sem a necessidade de participar da federação de cada um dos domínios envolvidos. Logo, é necessário estabelecer relações de confiança apenas com os IdPs e os SPs participantes da OV.

Após a criação da OV, o serviço está apto a processar as requisições de acesso aos SPs registrados na OV dos usuários de diferentes domínios administrativos, sejam de domínios de uma mesma federação ou de federações distintas, mesmo quando estes domínios utilizam diferentes sistemas de GId, porém, baseados no padrão SAML. A solução proposta pode ser utilizada em dois cenários de OVs, a saber:

- OV intrafederada, nos quais todos os membros de uma OV participam de uma mesma federação baseada no padrão SAML. Esta é a forma mais simples para criação, operação e dissolução da OV, uma vez que boa parte das relações de confiança já estão estabelecidas e não há necessidade de transposição de autenticação e de atributos entre federações.
- OV interfederada, nos quais os membros da OV são de federações distintas.

A Figura 2 apresenta os dois cenários, sendo que a Org. Virtual B apresenta a OV intrafederada e a Org. Virtual A apresenta a OV interfederada. A Figura 3 apresenta o fluxo de mensagens entre os domínios que participam de uma OV (seja intra ou interfederação). A seguir, o fluxo de mensagens entre as entidades é descrito.

No **passo 1** da Figura 3, o usuário, usando um navegador Web, tenta acessar o serviço no SP. O SP redireciona o navegador do usuário para o serviço gerente de segurança (**passo 2**). No **passo 3**, o usuário deve indicar o seu IdP de origem, que deve estar na lista apresentada pelo serviço gerente. No **passo 4**, o navegador do usuário é redirecionado pelo serviço gerente para a página de autenticação do IdP escolhido, que é apresentada para o usuário no **passo 5**. No **passo 6**, após o usuário inserir suas credenciais (usuário e senha), o IdP autentica o usuário e, caso autenticação seja bem sucedida, este gera asserção SAML de atributos (**passo 7**. O IdP envia a resposta de autenticação para o serviço gerente, conforme indicado no **passo 8**. No **passo 9**, o gerente agrega os

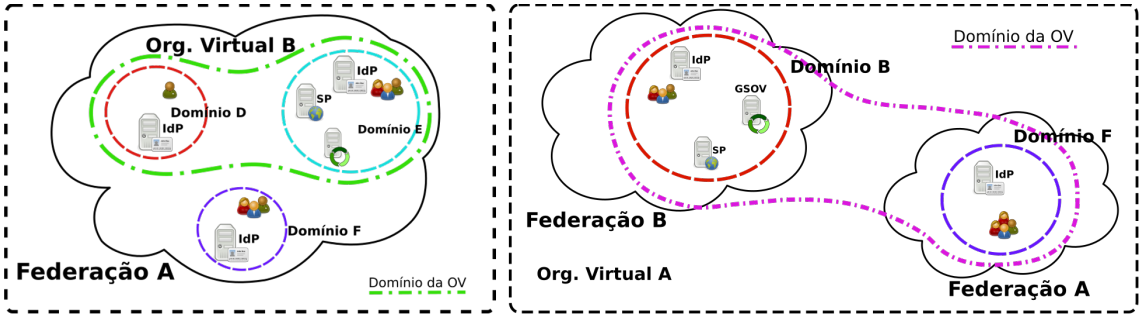


Figura 2. OV Intrafederada e interfederada

atributos vindos do IdP com os atributos já registrados no serviço referente ao usuário e apresenta para o usuário estas informações (**passo 10**). No **passo 11**, o usuário confirma a liberação de atributos para SP. No **passo 12**, o serviço envia os atributos resultantes do usuário para o SP e, com base nos atributos do usuário, decide se permite ou não o seu acesso ao serviço **passo 13**.

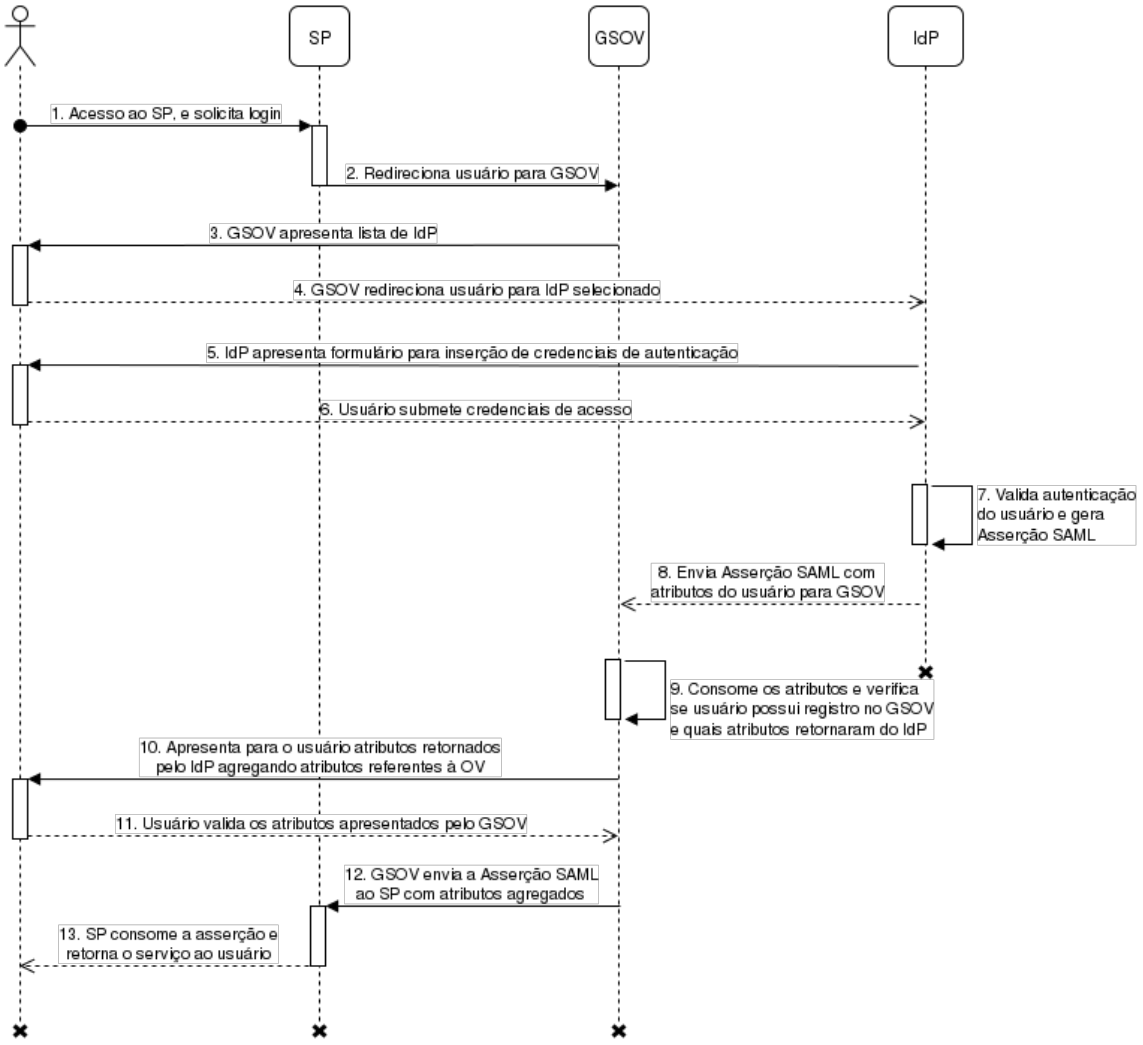


Figura 3. Diagrama de mensagens de comunicação entre as entidades

3.1. Componentes e funcionalidades

Para realizar o estabelecimento das relações de confiança entre os domínios e contribuir com a interoperabilidade entre os sistemas de GId, o serviço proposto possui os seguintes componentes:

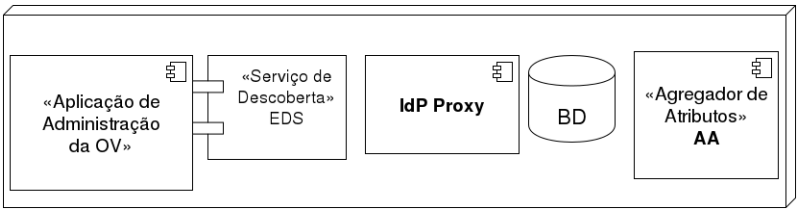


Figura 4. Componentes do serviço proposto

- Aplicação de administração: aplicação Web que possibilita ao administrador da OV criar, editar e excluir os membros da OV. Ou seja, uma aplicação com as funcionalidades para gerenciar o ciclo de vida das OVs. Além disto, este componente é responsável por auxiliar o estabelecimento das relações de confiança entre os IdPs que farão parte da OV e os SPs que hospedarão serviços colaborativos para a OV;
- *Embedded Discovery Service* (EDS): responsável por apresentar uma lista de IdPs para o usuário para que este indique qual seu IdP de origem. Com base na escolha do usuário, o EDS redirecionará o mesmo para o IdP escolhido para a autenticação;
- Agregador de Atributos: responsável por realizar a agregação de atributos do usuário. O agregador recebe os atributos liberados pelo IdP do usuário e agrega (concatena) novos atributos que foram definidos pelo administrador da OV no momento do registro deste usuário no serviço. Os atributos agregados são: o nome da OV que o usuário participa e o papel deste na OV;
- IdP Proxy: responsável pela mediação entre os IdPs e os SPs da OV e que está baseado na especificação IdP Proxy SAML [OASIS 2008]. Este componente permite que entidades de Federações diferentes ou que não participam de uma Federação possam interagir de forma a permitir que um usuário autenticado em um IdP cadastrado na OV possa acessar os serviços colaborativos que estão em outro domínio administrativo.

3.2. Agregação de atributos

Os atributos provenientes do IdP do pesquisador podem não ser suficientes para garantir que este pesquisador, que está tentando acessar o recurso, tenha permissão de acesso. Desta maneira, o serviço gerente de segurança possibilita que atributos específicos da OV sejam criados e concedidos aos membros da OV. Para isto, um mecanismos agregador de atributos, que é um mecanismo que coleta e une atributos de um usuário provenientes de diferentes provedores de identidades [Chadwick et al. 2011], foi concebido e integrado ao serviço proposto. Diante disto, o serviço proposto possibilita o registro de usuários para que os atributos específicos da OV sejam criados e atribuídos aos membros da OV. Vale destacar que o serviço proposto atua como um provedor e agregador de atributos.

3.3. Implementação e Resultados

De forma a avaliar a aplicabilidade do serviço proposto, uma prova de conceito foi desenvolvida utilizando o o ambiente de experimentação do GIdLab [Souza et al. 2014], com o objetivo de analisar os cenários de OVs intrafederada e interfederada. O serviço gerente de segurança para OVs, que gerencia o ciclo de vida da OV, foi desenvolvido em PHP e utiliza o *framework* SimpleSAMLphp[2] que implementa o protocolo SAML. As funcionalidades implementadas no serviço gerente estão indicadas no diagrama de casos de uso da Figura 5. Para armazenar as informações referentes as instituições, provedores de serviços e atributos agregados dos usuários que formam a OV, foi modelado um banco de dados MySQL para armazenamento.

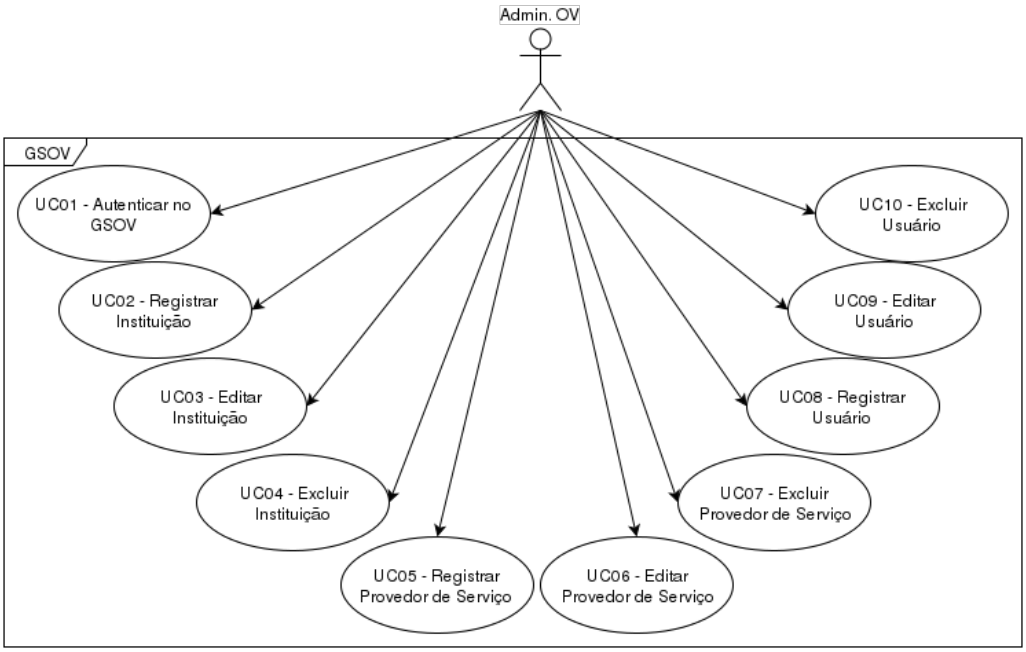


Figura 5. Casos de uso

O cenário desenvolvido, ilustrado na Figura 6, é composto por duas Federações. A aplicação colaborativa implantada no SP foi um ambiente de apoio para documentação de projetos (*Wiki*). Utilizou-se a o software MediaWiki que provê a criação destes ambientes. Para realizar autenticação via SAML, foi utilizada a extensão SimpleSAMLphp disponível neste software [3].

Para exemplificar a relação de confiança com IdPs de diferentes domínios administrativos (em diferentes federações), foram implantados três IdPs, utilizando dois *frameworks* distintos. IdPs1 e IdP2 utilizou-se o SimpleSAMLphp e no IdP3 o Shibboleth[4]. Em todos os IdPs foram definidos um conjunto de usuários para realização dos testes. Estes usuários foram armazenados em uma estrutura de diretórios OpenLDAP[5].

Para avaliar a solução do serviço gerente de segurança para organizações virtuais,

[2]https://simplesamlphp.org/
[3]https://www.mediawiki.org/wiki/Extension:SimpleSamlAuth
[4]https://shibboleth.net
[5]http://www.openldap.org/

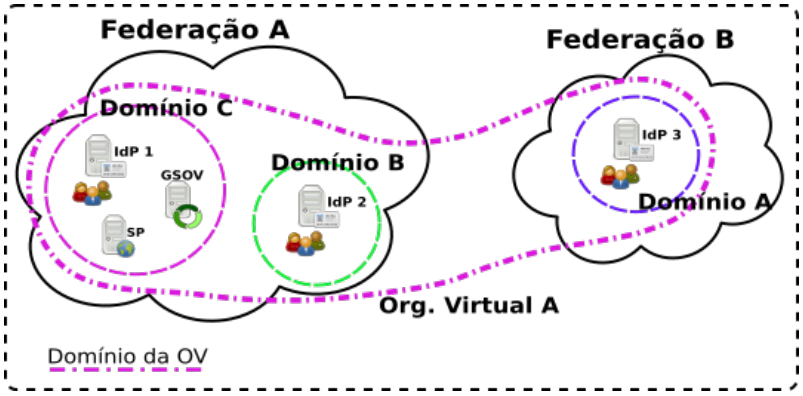


Figura 6. Cenário da Prova de Conceito

foram definidos e executados quatro casos de testes, tendo como base os principais casos de uso do serviço ilustrados na Figura 5.

O primeiro caso de testes teve como objetivo verificar a função de autenticação para acesso as funcionalidades de administração de uma OV providas pelo serviço. Neste caso de teste, o administrador da OV realizou a autenticação através da página de *login* da aplicação gerente de OV. No segundo caso de teste o objetivo era validar o processo de criação da OV, através do cadastro dos membros da OV (instituições e pesquisadores participante da OV) e dos serviços. Neste caso de teste, foram validados os casos de usos referentes a registro, edição e exclusão da OV e de seus participantes. No terceiro caso de testes, o objetivo foi validar o acesso de um pesquisador participantes da OV ao serviço colaborativo da OV. Além do acesso, foi validado ainda o processo de agregação de atributos (atributos vindos do IdP mais os atributos específicos da OV). O objetivo do último caso de teste foi verificar se usuários não registrados na aplicação gerente de OV, mas cadastrados em um IdP, seriam impedidos de acessar o serviço colaborativo. Todos os casos de testes executados obtiveram resultados positivos, logo é possível afirmar que as funcionalidades desenvolvidas estão de acordo com seus casos de uso.

4. Trabalhos relacionados

[Vullings et al. 2007] apresentam uma solução de Infraestrutura de Autenticação e Autorização (IAA) chamada *Identity Access Management Suite*[6], baseada nas especificações SAML e XACML para acesso a um ambiente de pesquisas virtuais (VRE). A IAA pode realizar a agregação de atributos provida por um IdP interno da OV (elemento VO AA), combinando atributos recebidos pelo IdP do usuário com os atributos específicos da VRE. A IAA também permite o acesso a serviços que estão em diferentes federações, baseado nas relações de confiança entre federações (p.e.: Federação InCommon e a Federação Australiana). A solução permite acessar os ambientes através de um portal web ou uma aplicação *desktop*, realizando autenticação via SAML ou através de certificados digitais X.509. A solução visa atender uma plataforma utilizada em ambientes de *Grid*, responsável pela criação e gerenciamento das OVs.

[Gemmill et al. 2009] apresentam um ambiente chamado myVOCS, que permite a criação e gerenciamento de OVs para ambientes de Grid de forma autônoma e flexível,

[6]Compatível com o portal do ambiente GridSphere `http://www.gridsphere.org/`

utilizando autenticação federada através do Shibboleth (SAML). myVOCS é uma alternativa ao uso de soluções como VOMS[7] e PERMIS[8], utilizados para gerenciamento de autorização das OVs. Desenvolvido como um serviço de gerenciamento dos atributos de afiliação de OV com a inserção dos atributos providos pelos IdPs dos usuários. A solução provê o acesso a contextos de segurança em domínios distribuídos, independentemente administrados permitindo ao usuário acessar diretamente e de forma transparente um recurso. A solução é implementada como um *proxy*, que se comporta como um IdP, recebendo as requisições dos SPs, e como um SP, enviando requisições de autenticação para os IdPs dos usuários. Os autores não descrevem como são estabelecidas as relações de confiança entre os serviços colaborativos e os IdPs (que realizam autenticação dos usuários participantes das OVs).

[Lopez Garcia et al. 2013] utilizam o VOMS para prover a autenticação e autorização em um ambiente multi-institucional para experimentos com nuvens, com diferentes provedores de recursos. No trabalho em questão, foi utilizado o OpenStack[9]. O suporte ao VOMS foi implementado através de um módulo adicionado ao servidor web que utiliza a funcionalidade de autenticação externa, permitida na arquitetura do OpenStack, que delega a autenticação a um terceiro. Neste trabalho, os autores apresentam a alteração realizada no mecanismo de autenticação e autorização do OpenStack, o Keystone, para permitir autorização usando certificados X.509 emitidos pelo VOMS, sendo que neste certificado um dos atributos identifica qual OV o usuário participa. Este trabalho não possui um mecanismo para criação e gerenciamento de de OVs.

[SILVA et al. 2015] propuseram o ACROSS (**A**ttribute-based access **C**ont**RO**l and di**S**tributed policie**S**), um arcabouço para autenticação e autorização em ambientes federados para criação de OVs. O ACROSS trata tanto da autenticação federada dentro da OV quanto do controle de acesso ao recursos através de políticas locais e globais. A arquitetura do ACROSS está organizada em módulos, sendo que os principais módulos são: Federação de Identidade, Provedor de atributos e de Controle de Acesso (ABAC). Neste trabalho, os membros da OV estão todos em uma única Federação baseada no SAML.

[Chard et al. 2016] apresentam a solução *Globus Nexus*, uma *Plaform as a Service* (PaaS) desenvolvida como parte do projeto Globus[10], que tem como foco disponibilizar uma plataforma para o ambiente colaborativo de *e-science* e uma série de serviços, como armazenamento de dados, provedores de recursos computacionais e outros serviços para *e-science*. O Globus Nexus funciona como um Provedor de Identidade (que agrega diferentes atributos de diferentes IdPs associados a uma identidade), além disso provê gerenciamento de grupos e suporta a criação de domínios customizados.

Em relação aos trabalhos [Vullings et al. 2007], [Gemmill et al. 2009] [SILVA et al. 2015], o presente trabalho tem como diferencial a possibilidade da formação de OVs de forma descentralizada, de acordo com a demanda dos participantes dos projetos envolvidos na OV. Outro diferencial da presente solução é criação de OVs com a participação de membros que estejam em diferentes federações SAML sem a necessidade de que estas federações estejam em uma confederação. Além disso, ao

[7]http://toolkit.globus.org/grid_software/security/voms.php
[8]http://sec.cs.kent.ac.uk/permis/index.shtml
[9]http://openstack.org/
[10]https://www.globus.org/

contrários das soluções apresentadas em [Lopez Garcia et al. 2013] e [Chard et al. 2016], focadas em nuvem ou em grids computacionais, respectivamente, a presente solução pode ser empregada em diferentes cenários de OVs para realiza *e-science*.

5. Conclusão

Este trabalho teve como objetivo apresentar uma solução para a criação de ambientes de OVs quando há a necessidade de colaboração em projetos e os participantes (instituições e usuários) que estão em diferentes federações SAML. O serviço de gerência de segurança para OVs provê ainda um mecanismo agregador de atributos capaz de complementar os atributos vindos dos IdPs das federações, de forma a contribuir com a granularidade e flexibilidade das políticas de controle de acesso baseadas em atributos dos serviços colaborativos das OVs.

Federações de serviços que adotam o modelo federado, baseadas o padrão SAML, emergiram com grande aceitação nas redes nacionais de ensino e pesquisa (NRENs) e outras redes colaborativas. Porém, outras tecnologias como *OpenID Connect*, *OAuth* e *WS-Federation* também podem ser utilizadas nestes ambientes. Como trabalho futuro, pretende-se estender a solução proposta de forma a contribuir com a interoperabilidade entre os sistemas de gestão de identidades federadas que podem ser utilizados nos administrativos dos membros que compõem uma OV. Ainda como trabalho futuro, pretende-se empregar a solução proposta em um cenário de e-Science mais complexo, como por exemplo um cenário que utiliza *testbeds* de experimentação para Internet do Futuro.

Referências

Bhargav-Spantzel, A., Camenisch, J., Gross, T., e Sommer, D. (2007). User Centric: A Taxonomy and Open Issues . *Journal of Computer Security*, page 493–527.

Broeder, D., Jones, B., Kelsey, D., Kershaw, P., Lüders, S., Lyall, A., Nyrönen, T., Wartel, R., e Weyer, H. J. (2012). Federated Identity Management for Research Collaborations, CERN-OPEN-2012-006. Technical Report CERN-OPEN-2012-006, CERN, Geneva.

Capuano, N., Gaeta, A., Gaeta, M., Orciuoli, F., Brossard, D., e Gusmini, A. (2010). Management of virtual organizations. In Dimitrakos, T., Martrat, J., e Wesner, S., editors, *Service Oriented Infrastructures and Cloud Service Platforms for the Enterprise*, pages 49–73. Springer Berlin Heidelberg.

Carmody, S., Erdos, M., Hazelton, K., Hoehm, W., Morgan, B., Scavo, T., e Wasley, D. (2005). Incommon Technical Requirements and Information. Technical report, Incommon.

Chadwick, D. (2009). Federated identity management. In Aldini, A., Barthe, G., e Gorrieri, R., editors, *Foundations of Security Analysis and Design V*, volume 5705 of *Lecture Notes in Computer Science*, pages 96–120. Springer Berlin Heidelberg.

Chadwick, D. W., Inman, G. L., Siu, K. W., e Ferdous, M. S. (2011). Leveraging social networks to gain access to organisational resources. In *Proceedings of the 7th ACM Workshop on Digital Identity Management*, DIM '11, pages 43–52, New York, NY, USA. ACM.

Chard, K., Lidman, M., McCollam, B., Bryan, J., Ananthakrishnan, R., Tuecke, S., e Foster, I. (2016). Globus nexus: A platform-as-a-service provider of research identity, profile, and group management. *Future Gener. Comput. Syst.*, 56(C):571–583.

Foster, I., Kesselman, C., e Tuecke, S. (2001). The anatomy of the grid - enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15:2001.

Gemmill, J., Robinson, J.-P., Scavo, T., e Bangalore, P. (2009). Cross-domain authorization for federated virtual organizations using the myvocs collaboration environment. *Concurr. Comput. : Pract. Exper.*, 21(4):509–532.

ITU-T (2009). NGN Identity Management Framework - Recommendation Y.2720. Technical report, ITU-T.

Jøsang, A., Fabre, J., Hay, B., Dalziel, J., e Pope, S. (2005). Trust Requirements in Identity Management. In *ACSW Frontiers*, volume 44 of *CRPIT*, page 99–108. Australian Computer Society.

Lopez Garcia, A., Fernandez-del Castillo, E., e Puel, M. (2013). Identity federation with voms in cloud infrastructures. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 42–48.

OASIS (2008). Security Assertion Markup Language (SAML). Technical Report Technical Overview, OASIS.

Schroeder, R. (2008). e-Sciences as research technologies: reconfiguring disciplines, globalizing knowledge. *Social Science Information*, 47(2):131–157.

SILVA, E. F., FERNANDES, N. C., e Muchaluat-Saade, D. (2015). Modelagem do across: Um arcabouço de aa baseado em políticas e atributos para organizações virtuais. In *Workshop de Gestão de Identidade (WGID), Anais do XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg2015)*, pages 1–12. Sociedade Brasileira de Computação.

Souza, M. C., Mello, E. R., e Wangham, M. S. (2014). Gidlab: Laboratório de experimentação em gestão de identidade. In *Workshop de Gestão de Identidade (WGID), Anais do XIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg2014)*, page 467–468. Sociedade Brasileira de Computação.

Taylor, J. (2001). News from the e-Science Programme, first phase. *Social Science Information*, 47(2):131–157.

Vullings, E., Dalziel, J., e Buchhorn, M. (2007). Secure federated authentication and authorisation to grid portal applications using saml and xacml. *Journal of Research and Practice in Information Technology*, 39(2):101–113. cited By 5.

Zhang, H., Wu, W., e Li, Z. (2012). Open social based group access control framework for e-science data infrastructure. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pages 1–8.

CoFee: Uma Federação de Identidade para IoT

Pedro Micael T. L. N. Pinto[1], Antonio L. Maia Neto [1], Maria Luiza B. A. Santos[1]
Artur Souza[1], Marco A. Amaral Henriques[2],Leonardo B. Oliveira[1]

[1]Universidade Federal de Minas Gerais (UFMG)
[2] Universidade Estadual de Campinas (Unicamp)
{pedromicael,lemosmaia,mburga,arturluis,leob}@dcc.ufmg.br
marco@dca.fee.unicamp.br

Abstract. *Due to the constraints of IoT elements, even widely adopted schemes such as digital certificates/PKI (Public Key Infrastructure) are inadequate to this context. Our goal is, therefore, to come up with an Federated Identity Management solution exclusively tailored to IoT. We also evaluate alternatives to traditional PKI and our results indicate that an approach using ECQV (Elliptic Curve Qu-Vanstone) is around 10% better than the traditional approach.*

1. Introdução

A Internet das Coisas (IoT – *Internet of Things*) é um tópico de pesquisa cada vez mais relevante (Atzori et al. 2010; Wangham et al. 2013). A ideia básica deste conceito é que o ambiente no entorno de usuários seja repleto de elementos computacionais chamados de "coisas" (*things*).

A área de Gestão de Identidade, por sua vez, tem recebido grande atenção por parte de empresas, provedores de serviço e pelas redes que dão suporte ao ensino ao redor do mundo (Torres et al. 2013; Horrow and Sardana 2012). Com a proliferação de provedores de serviços na Internet, aumentam os usuários e, por conseguinte, a necessidade de se estabelecer controle de acesso a serviços e/ou informações mais efetivos. Temos também uma maior incidência de fraudes a tais serviços, o que impõe uma infinidade de desafios técnicos, científicos e tecnológicos à comunidade de Segurança Digital.

Tais desafios são ainda maiores quando olhados pela perspectiva do mundo conectado de IoT. Isso porque algumas das estratégias para viabilizar a Gestão de Identidade na Internet tradicional não são adequadas à IoT. Por exemplo, autenticação na Internet tradicional é usualmente obtida por meio de uma Infraestrutura de Chaves Públicas (*Public Key Infra-Structure* – PKI), que muitas vezes não é adequada ao ambiente de IoT.

O objetivo deste trabalho é propor uma solução de Gestão de Identidade Federada para IoT, para tanto apresentamos uma Federação de Identidade para a Internet das Coisas chamada de **Co**isas **Fed**eradas (CoFee). A CoFee substitui criptossistemas assimétricos empregados numa PKI convencional por outros mais leves (*lightweight*) e, portanto, se adequa aos recursos computacionais de IoT.

2. CoFee

O esquema de gestão de identidade federada proposto, visa garantir acesso de um dispositivo fora do seu domínio de origem a um serviço provido em um domínio distinto do seu, utilizando-se para isso do processo de autenticação de seu próprio provedor de identidade. É assumido que foi realizado um cadastro do dispositivo D no seu provedor de

identidade. Utilizamos de premissas criptográficas para garantir a autenticação, integridade das mensagens e sigilo nas comunicações entre as entidades envolvidas, que neste caso são: dispositivo D, provedor de identidade IdP e o provedor de serviço SP. Para

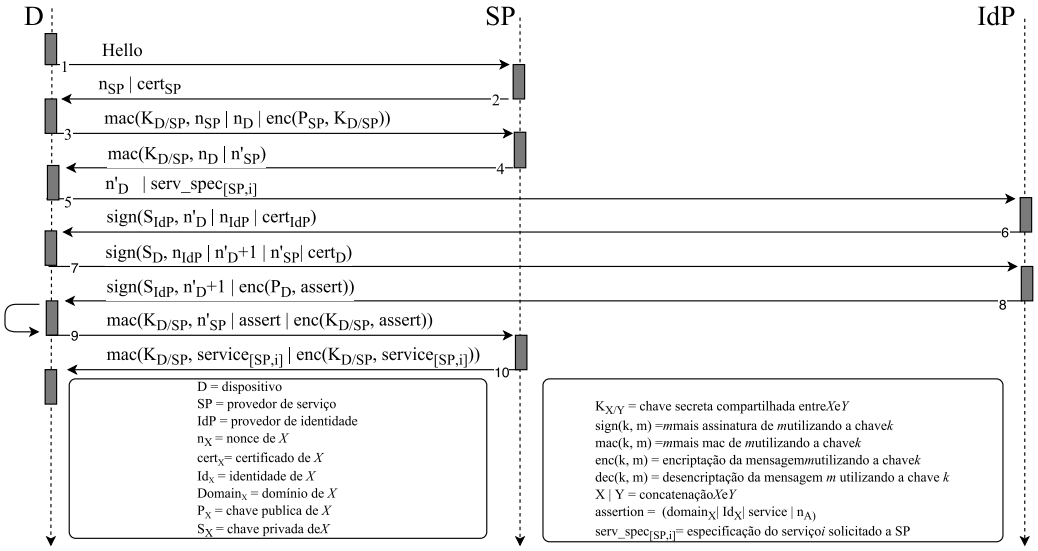


Figura 1. Ilustração de um Protocolo genérico de federação de identidade

que D utilize os serviços disponibilizados por SP é necessário que se autentique no seu provedor de identidade, que têm uma relação de confiança estabelecida com SP.

A utilização de certificados convencionais em conjunto com assinaturas para garantir a autenticidade de entidades em uma infraestrutura PKI tradicional pode não ser adequada para o ambiente de IoT.

2.1. Avaliação dos Protocolos

O alto custo de gerenciamento e verificação de certificados digitais tradicionais os tornam inadequados a estratégias de Gestão de Identidade no contexto de IoT. Note-se que não existe uma solução que atenda todos os requisitos desejáveis de um cenário de IoT. Abaixo apresentamos um levantamento de algumas características dos criptossistemas.

DSA+PKI: o Algoritmo de Assinatura Digital (*Digital Signature Algorithm* – DSA) é um esquema de assinaturas comumente utilizado na Internet em conjunto com certificados digitais tradicionais (PKI). A segurança do esquema é baseada no problema do logaritmo discreto (*Discrete Logarithm Problem* – DLP). O tamanho dos parâmetros e a necessidade dos certificados tradicionais tornam o esquema inviável para a Internet das Coisas.

ECQV: no esquema de certificados implícitos baseado em curvas elíptica de Qu-Vanstone (Brown et al. 2002), a chave pública de uma entidade é combinada com o certificado de forma a criar um elemento único, do qual a chave pública pode ser extraída sendo implicitamente verificada. Uma grande vantagem desse método reside no tamanho consideravelmente reduzido dos certificados implícitos, que os torna muito menores do que certificados tradicionais, o que torna o esquema muito interessante em cenários IoT.

Foram avaliados também os esquemas de Boneh e Franklin (Boneh and Franklin 2001), Cocks (Cocks 2001), Boneh-Gentry-Hamburg (Boneh et al. 2007) (BGH) e *Certificateless* com e sem emparalhemantos. De

forma qualitativa foram avaliados aspectos como custo computacional das operações utilizadas nos esquemas, tamanho de cifras gerados, ser baseado ou não em identidades e por fim a custódia de chaves. O custo das operações de emparalhamento presentes em alguns dos esquemas os tornam inviáveis pra o contexto de IoT. É importante ressaltar que que nenhum dos esquemas é totalmente adequado para o ambiente de IoT.

3. Resultados e Análise

Avaliação Analítica. Esta análise pretende determinar o custo analítico total para que D acesse um serviço provido por SP, o que envolve a autenticação de D por seu IdP (figura 1). Para tal, são avaliados os esquemas PKI e ECQV, ambos em conjunto com o ECDSA e o esquema de cifração (*Elliptic Curve Integrated Encryption Scheme* – ECIES).

Note-se que os esquemas avaliados são baseados em curvas elípticas, onde as operações que influenciam significativamente o custo de computação, especialmente quando se considera dispositivos com limitações de recursos, são a multiplicação de pontos de curva elíptica por escalar e os emparelhamentos bilineares. Dessa forma, considerar o número de execuções dessas operações no custo dos esquemas é uma métrica válida de avaliação analítica. Tanto PKI quanto ECQV não utilizam-se de operações de emparelhamentos, direcionando essa análise à quantidade de multiplicações de ponto por escalar. Como apontado em (Oliveira et al. 2009), o esquema *Certificateless*, ao fazer uso de emparelhamentos bilineares, incorre em uma alta sobrecarga computacional, o que justifica a sua exclusão desta avaliação.

O processo do acesso de D a um serviço de SP envolve as seguintes operações: geração e verificação de assinatura, cifração e decifração. O número de operações de (de)cifração é comum às duas estratégias, logo, o número de multiplicações de pontos, neste caso, é o mesmo. Para as operações que envolvem assinaturas, no entanto, há diferenças. No esquema PKI, a verificação de uma assinatura envolve, como passo complementar, a validação do certificado, o que, por sua vez, requer a verificação da assinatura contida no mesmo. Já na abordagem que usa ECQV, a verificação do certificado é realizada de maneira implícita, existindo a necessidade de uma operação de extração da chave pública. tabela 1 apresenta a quantidade de operações de multiplicação de pontos. Note-se que o PKI necessita de duas operações de multiplicação de ponto a mais que o ECQV, característica que deve estar refletida nos resultados experimentais, apresentados adiante.

Tabela 1. Custo Computacional e Tempos de Execução: PKI x ECQV

	Multiplicação de Pontos		**Tempo de Execução**	
	PKI	ECQV	PKI	ECQV
validação do certificado	4	n.a.	16.31 $\pm$0.48	n.a.
extração da chave pública	n.a.	2	n.a.	10.98 $\pm$0.49
geração de assinatura	1	1	6.80 $\pm$0.40	6.80 $\pm$0.40
verificação de assinatura	4	4	16.31 $\pm$0.48	16.31 $\pm$0.48
cifração	2	2	17.00 $\pm$0.55	17.00 $\pm$0.55
decifração	1	1	11.05 $\pm$0.38	11.05 $\pm$0.38

Avaliação Experimental. Os experimentos visaram avaliar os tempos de execução de cada um dos esquemas apresentados na avaliação analítica. Os códigos foram implementados na linguagem C, tendo como base a biblioteca criptográfica *RELIC*[1].

[1] `https://github.com/relic-toolkit`

Os testes foram executados na plataforma *Intel Edison Kit for Arduino*, com CPU *dual-core 500 MHz Intel Atom* e cada esquema foi executado 100 vezes. Avaliando os tempos totais de execução foi observado que, o menor tempo de execução obtido pelo PKI na amostra é maior do que o maior tempo de execução obtido pelo ECQV. A diferença média entre eles é de aproximadamente 10 ± 1.6675 ms, com o ECQV sendo, em média, 10% mais rápido. Calculando-se o intervalo de confiança, pode-se dizer que o ECQV executa em menor tempo que o PKI com 95% de confiança.

Os resultados experimentais confirmam os resultados analíticos esperados, com o tempo de execução do ECQV inferior ao PKI. A tabela 1 mostra o tempo em milisegundos necessário para execução de cada operação do PKI e do ECQV. As operações como geração e verificação de MAC, assim como as operações envolvendo criptografia simétrica não foram avaliadas nem analiticamente e nem experimentalmente pois tem tempos de execução desprezíveis frente ao custo demandado pelas operações em curvas elípticas.

Embora a diferença percentual entre a abordagem ECQV e PKI em relação as operações em multiplicação de pontos seja de 16,6%, isso não se reflete no percentual percebido no tempo de execução das abordagens a diferença é de 10%. Isso pode ser explicado pelo fato do ECDSA permitir o uso de primos de Mersenne, que possuem uma forma eficiente para computar reduções modulares, uma operação essencial no algoritmo.

4. Conclusão

A abordagem utilizando ECQV, foi cerca de 10% melhor que a PKI convencional, sendo mais rápida com 95% de confiança. Nossa proposta se mostra, portanto, atraente para o contexto de IoT, resultando em uma melhoria no desempenho e redução no consumo energético dos dispositivos utilizados.

Referências

[Atzori et al. 2010] Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805.

[Boneh and Franklin 2001] Boneh, D. and Franklin, M. (2001). Identity-based encryption from the weil pairing. In *CRYPTO*, pages 213–229. Springer.

[Boneh et al. 2007] Boneh, D., Gentry, C., and Hamburg, M. (2007). Space-efficient identity based encryptionwithout pairings. In *FOCS*, pages 647–657. IEEE.

[Brown et al. 2002] Brown, D. R. L., Gallant, R. P., and Vanstone, S. A. (2002). Provably secure implicit certificate schemes. In *FC*.

[Cocks 2001] Cocks, C. (2001). An identity based encryption scheme based on quadratic residues. In *Cryptography and coding*, pages 360–363. Springer.

[Horrow and Sardana 2012] Horrow, S. and Sardana, A. (2012). Identity management framework for cloud based internet of things. In *SECURIT*, pages 200–203.

[Oliveira et al. 2009] Oliveira, L. B., Kansal, A., Priyantha, B., Goraczko, M., and Zhao, F. (2009). Secure-tws: Authenticating node to multi-user communication in shared sensor networks. In *IPSN*, pages 289–300.

[Torres et al. 2013] Torres, J., Nogueira, M., and Pujolle, G. (2013). A survey on identity management for the future network. *Communications Surveys Tutorials, IEEE*, 15(2):787–802.

[Wangham et al. 2013] Wangham, M. S., Domenech, M. C., and de Mello, E. R. (2013). Infraestrutura de autenticação e de autorização para internet das coisas. In *Minicursos: SBSeg*, volume 1. SBC.

Protocolos de Autenticação de Dispositivos Móveis em Grupos para a Internet de Objetos (IoT)

Ana Paula Golembiouski Lopes[1], Lucas de Oliveira Hilgert[2], Luis Fernando Arias Roman[3], Paulo Roberto de Lira Gondim[4]

Departamento de Engenharia Elétrica – Universidade de Brasília (UnB)
Campus Universitário Darcy Ribeiro – 70910-900 – Brasília – Brazil

{anagolembiouski, lucas.hilgert, lfroman}@aluno.unb.br, pgondim@unb.br

Abstract. *The full establishment of Internet of Things (IoT) consists in connecting billions of devices securely and efficiently. The authentication of these devices will generate a large increase of signaling traffic, overloading the wireless access networks. To avoid this, a good solution is to authenticate groups of devices. The currently proposed protocols by the 3rd Generation Partnership Project (3GPP) are not suitable to large groups of devices. Consequently, group authentication protocols emerged, aiming for efficiency and the security required to the process. This paper presents a comparison involving recent authentication protocols, considering a set of criteria and metrics, allowing the evaluation of their characteristics.*

Resumo. *O pleno estabelecimento da Internet de Objetos (ou das Coisas) – IoT (do inglês, Internet of Things), requer a conexão de bilhões de dispositivos de maneira segura e eficiente. A autenticação destes dispositivos gerará grande aumento no tráfego de sinalização, sobrecarregando as redes de acesso sem fio. Para evitar isso, uma solução é autenticar grupos de dispositivos. Os protocolos propostos pelo 3rd Generation Partnership Project (3GPP) não são adequados para grandes grupos de dispositivos. Consequentemente, surgiram protocolos de autenticação em grupos, buscando eficiência e a segurança necessária para o processo. Este artigo compara protocolos de autenticação recentes, considerando um conjunto de critérios e métricas, permitindo avaliar suas características.*

1. Introdução

A IoT (*Internet of Things*) representa uma forte tendência atual, e seu emprego pode ser vinculado a diversas áreas de aplicação, tais como redes inteligentes (*Smart Grid),* cidades inteligentes e *m-health (Mobile Health)*. Uma das maiores preocupações para o pleno estabelecimento da IoT consiste em conectar bilhões de dispositivos de maneira segura e eficiente, sendo a rede LTE (*Long Term Evolution*) uma candidata natural para a integração desses dispositivos (comumente à base de grupos) à Internet. O protocolo 3GPP EPS-AKA (*Evolved Packet System – Authentication and Key Agreement*) é o protocolo de autenticação e acordo de chaves padrão para ser utilizado na E-UTRAN (*Evolved Universal Terrestrial Access Network*), que autentica cada um dos dispositivos individualmente. No âmbito da IoT, e também para a rede LTE, isso gera problemas, como o aumento do tráfego de sinalização, dos custos computacionais e de comunicação, além de vulnerabilidades de segurança que poderiam comprometer a comunicação e o funcionamento integrado desses dispositivos.

Este artigo apresenta, na seção 2, a descrição e a comparação de alguns protocolos de autenticação em grupo. Na seção 3 é feita uma análise de segurança e, na seção 4, tem-se a conclusão, trabalhos em andamento e futuros.

2. Descrição e Comparação dos Protocolos

Vários protocolos para autenticação de dispositivos em grupos têm sido propostos na literatura. Foram selecionados cinco protocolos para análise comparativa (incluindo um, EPS-AKA, representante dos padrões 3GPP atuais), com base em requisitos de segurança, resistência a ataques, custos de comunicação e computacionais, uso ou não de protocolo Diffie Hellman sobre curvas elípticas(ECDH) e de emparelhamento bilinear, dentre outros. As arquiteturas das redes LTE e 5G podem ser encontradas em [Firmin 2016] e [Cao 2016], respectivamente, incluindo entidades como MME (*Mobillity Management Entity*) e HSS (*Home Subscriber Server*).

SE-AKA [Lai 2013] utiliza criptografia assimétrica ECDH. Faz uso de identidades temporárias de MTCD (*Mobile Terminal Communication Device*) e de grupo para preservar suas identidades permanentes. Possui uma fase de inicialização, em que são definidos parâmetros para a autenticação. É usada chave de grupo e uma tabela de gerenciamento. Ao invés de definir um líder, trata de forma diferenciada o primeiro MTCD a se autenticar. Utiliza LAI (*Location Area Identification)* para que o HSS verifique se a *Base Station* indicada pelo MME é a mesma associada ao MTCD. O uso de ECDH permite que MTCD e SN (*Server Network*) gerem uma mesma chave, não compartilhada no canal de transmissão.

CHOI [Choi 2014] utiliza criptografia simétrica. Na fase de inicialização faz agrupamento dos MTCDs, sendo o gerenciamento do grupo feito usando árvore binária, com cada nó da árvore associado a um valor secreto derivado de seus nós pais, por funções hash. Após eleição do líder do grupo são definidos parâmetros como a chave de grupo, números aleatórios e funções hash. Os dispositivos do grupo são autenticados simultaneamente com o líder. Estabelece uma chave de sessão, obtida com a função hash dos valores secretos da árvore binária que os nós envolvidos possuem em comum.

GBAAM [Cao 2015] adota criptografia assimétrica ECDH. Na fase de registro ocorre formação de grupos e a eleição de líderes. Os MTCDs quando se registram pela primeira vez, devem ser provados autênticos para o KGC (*Key Generation Center*), recebendo chaves privadas. Um KGC define e publica os parâmetros do sistema e mantém uma chave mestre em segredo. O líder realiza a agregação das assinaturas de todos os MTCDs em uma única assinatura, que é enviada ao MME, e a autenticidade e aferida por emparelhamento bilinear. Se falhar na verificação, o MME divide o grupo de assinaturas em subgrupos e realiza emparelhamento em cada um deles, até detectar o subgrupo que apresenta assinatura inválida e, em seguida, por divisões sucessivas, encontrar a assinatura inválida.

GLARM [Lai 2016] utiliza criptografia simétrica; na fase de inicialização, forma grupos e cada MTCD calcula uma identidade temporária. Há a eleição de um líder. O processo de autenticação se baseia em desafio-resposta. Assim como o SE-AKA, utiliza identificador de localização para verificação de eNB e tabela de gerenciamento de grupo. Durante esta fase são derivadas chaves de cifragem, de integridade e a chave temporária de grupo. Ao final de uma autenticação com sucesso, cada MTCD compartilha com o MME uma chave K_{ASME} (*Key Access Security Management Entity*), essencial para

derivações de chaves futuras. Renova a chave de grupo a cada vez que um MTCD entra ou sai do grupo. Há ainda uma proposta para autenticação em grupo em redes não-3GPP. A Tabela 1 resume as características mais relevantes de cada um dos protocolos.

O custo computacional de [Cao 2015] é o mais alto dentre as propostas por utilizar emparelhamento bilinear, que exige muita capacidade de processamento. Seu custo de comunicação e o de [Lai 2016] são os mais baixos em relação aos demais protocolos.

Tabela 1. Principais características dos protocolos.

	EPS-AKA	SE-AKA	CHOI	GBAAM	GLARM
Autenticação em Grupo	Não	Sim	Sim	Sim	Sim
Tipo de Criptografia	Simétrica	Assimétrica ECDH	Simétrica	Assimétrica ECDH	Simétrica
MTC *Server*	Não possui	Dentro da EPC	Fora da EPC	Ambos	Fora da EPC
Eleição de Líder	Não	Não	Sim	Sim	Sim
Gerenciamento de Grupo	Não possui	Tabela	Árvore Binária	Dinâmico	Tabela
Emparelhamento Bilinear	Não	Não	Não	Sim	Não
Verificação de Localização (uso de LAI)	Não	Sim	Não	Não	Sim
Custo de Comunicação	Muito Alto	Alto	Baixo	Baixo	Baixo
Custo Computacional	Muito Alto	Baixo	Baixo	Alto	Baixo

3. Análise de Segurança

Algumas características estão presentes em todos os protocolos considerados: realizam autenticação mútua e acordo de chaves; usam chaves de sessão; ataques de repetição são evitados usando marcadores temporais (*Timestamps)* e números aleatórios durante autenticação; ataques *Man-in-the-Middle* são evitados, pois os parâmetros publicados no canal de comunicação não são suficientes para que um atacante gere mensagens e chaves de sessão válidas; segurança no futuro (*Forward Secrecy*) e segurança no passado (*Backward Secrecy*) são preservadas em relação a chaves de grupo, na entrada e saída de membros; todos conseguem se contrapor a ataques de personificação.

Tabela 2. Análise de segurança dos protocolos.

	EPS-AKA	SE-AKA	CHOI	GBAAM	GLARM
Autenticação Mútua e Acordo de Chaves	Sim	Sim	Sim	Sim	Sim
Confidencialidade	Não	Sim	Sim	Sim	Sim
Integridade	Não	Sim	Sim	Sim	Sim
Privacidade	Não	Sim	Não	Não	Não
Segurança no Futuro / no Passado	Não	Sim	Sim	Sim	Sim
Resistência ao Ataque de Repetição	Não	Sim	Sim	Sim	Sim
Resistência ao Ataque DoS	Não	Sim	Não	Não	Sim
Resistência ao Ataque *Man-in-the-Middle*	Não	Sim	Sim	Sim	Sim
Resistência ao Ataque de Redirecionamento	Não	Sim	Não	Não	Sim
Resistência ao Ataque de Personificação	Não	Sim	Sim	Sim	Sim

Para tais ataques, GBAAM [Cao 2015] busca impedir o adversário de forjar assinaturas válidas; CHOI-2014 atribui chaves de sessão únicas para cada MTCD; [Lai 2013] e [Lai 2016] usam números aleatórios diferentes em cada autenticação.

A privacidade é garantida no protocolo SE-AKA por proteger a identidade permanente dos MTCDs através de infraestrutura de chaves públicas. Os protocolos SE-AKA e GLARM evitam o ataque de negação de serviço (DoS) pelo uso de marcadores temporais e de LAI, permitindo detectar mensagens forjadas. O uso de LAI também evita os ataques de redirecionamento. A Tabela 2 sintetiza uma análise de segurança dos protocolos.

4. Conclusão e Trabalhos Futuros

É enorme a importância da autenticação de grupos de dispositivos, especialmente para o sucesso das implementações da IoT em áreas como *smart grid* e *m-health*. Após a análise, constatou-se que, em relação ao EPS-AKA [3GPP 2009], os demais protocolos avaliados realmente apresentam custos computacionais e de comunicação bastante reduzidos, atendendo requisitos de segurança e resistindo a diversos ataques. Outros protocolos foram também avaliados (tais como [Chen 2012], [Fu 2016], [LI 2016] e [Cao 2014]), mas não inclusos neste artigo por falta de espaço.

Trabalhos em andamento incluem o projeto de protocolo de autenticação para dispositivos móveis em grupo, com ênfase na utilização de protocolo Diffie Hellman sobre curvas elípticas e de emparelhamento bilinear. Trabalhos futuros incluem a simulação a eventos discretos e a validação formal, bem como avaliação dos custos de comunicação e computação.

Referências

3GPP (2009) 3GPP TS 33.401 V8.2.1, 3GPP System Architecture Evolution (SAE). Security Architecture.

Cao, J., Ma, M. and Li, H., (2014) “ABAAM: Access Authentication of Mass Device Connections for MTC in LTE Networks”, Smart Computing Review, vol. 4, no. 4, DOI: 10.6029/smartcr.2014.04.003

Cao, J., Ma, M. and Li, H. (2015) “GBAAM: Group-based Access Authentication for MTC in LTE Networks”, Security and Communication Networks, 8(17), 3282-3299. DOI: 10.1001/sec.1252.

Chen, Y., Wang, J., Chi, K. and Tseng, C., (2012) “Group-Based Authentication and Key Agreement”, Wireless Personal Communications. 62:965-979. DOI: 10.1007/s11277-010-0104-7.

Choi, D., Choi, H. and Lee, S. (2014) “A group-based security protocol for machine-type communications in LTE-advanced”, Wireless Networks 21:405. DOI: 10.1007/s112276-014-0788-9.

Firmin, F. (2016) “The Evolved Packet Core”. Acesso em Setembro 2016. http://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core

Fu, A., Song, J., Li, S., Zhang, G. and Zhang. Y. (2016) “A privacy-preserving group authentication protocol for machine-type communication in LTE/LTE-A networks”, Security and Communication Networks, 9:2002-2014 DOI: 10.1002/sec.1455.

Lai, C., Lu, R., Zheng, D., Li, H. and Shen, X. (2016) “GLARM: Group-based lightweight authentication scheme for resource-constrained machine to machine communications”, Computer Networks 99(2016) 66-81. DOI: 10.1016/j.comnet.2016.02.007.

Lai, C., Li, H., Lu, R. and Shen, X. (2013). “SE-AKA: A secure and efficient group authentication and key agreement protocol for LTE networks.” Computer Networks 57 (2013) 3492-3510. DOI: 10.1016/j.comnet.2013.08.003.

Li, J., Wen, M., and Zhang, T., (2016) "Group-Based Authentication and Key Agreement With Dynamic Policy Updating for MTC in LTE-A Networks”, *IEEE Internet of Things Journal*, vol.3, no.3, 408-417. DOI:10.1109/JIOT.2015.2495321.

Estudo preliminar sobre o uso dos *blockchains* de *Bitcoin*, *Litecoin*, *Ethereum* e *Namecoin* em gestão de identidades

Antônio Unias de Lucena[1], Marco Aurélio Amaral Henriques[1]

[1]Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
CEP 13083-852 – Campinas, SP, Brasil
{alucena, marco}@dca.fee.unicamp.br

Resumo. *The development of identity management systems using blockchain has changed their characteristics of being centralized structures and permited the creation of identity management systems that are distributed, descentralized and, even so, strong against frauds. There are already implementations of identity management systems using blockchain, Blockstack, for instance, but these systems still need to overcome some technical drawbacks in order to improve their performance. This article enumarates the main problems that Blockstack can face in a near future and evaluates the possibility of developing identity management systems using the blockchains of Bitcoin, Litecoin, Ethereum and Namecoin. Among the analysed blockchains, the Bitcoin is nowadays the most trustful blockchain, but Ethereum due to its characteristics of being designed to applications seems to be a good option for the implementation of identity management systems.*

1. Introdução

Blockchain é uma base distribuída de dados que mantém uma lista encadeada com todos os registros dos elementos de um conjunto, bem como os registros temporais de qualquer criação de novos elementos e modificação destes, impossibilitando assim revisão e adulteração dos mesmos.

O *blockchain* foi originalmente desenvolvido para ser o livro-razão da criptomoeda *Bitcoin*, mas por ser um modelo de sistema descentralizado e robusto a fraudes chamou a atenção de outras aplicações que necessitam de garantia de segurança e de descentralização de controle, tais como sistemas de gestão de identidade (GId) e infraestruturas de DNS (*Domain Name System*).

Antes do uso de *blockchain* em sistemas de GId, estes sistemas eram pautados pelo triângulo de Zooko (*Zooko's triangle*), ou seja, um nome só poderia ter no máximo duas das seguintes propriedades: ausência de um elemento centralizador, inteligibilidade e segurança. O emprego de *blockchain* nestes sistemas possibilitou que não houvesse mais a necessidade de um órgão central para gerenciar, mediar e controlar as interações realizadas entre os membros do sistema de gestão de identidade, possibilitando maior confiabilidade entre os usuários.

Já existem implementações de sistemas de GId em *blockchain* como por exemplo, o *Blockstack* [Muneeb et al 2016]. Porém, são necessárias melhorias em alguns aspectos técnicos dos *blockchains* como no processo de mineração (criação de

novos blocos), na capacidade e *throughput* de armazenamento de dados e no consumo de energia elétrica para melhorar o desempenho destas aplicações utilizando esta tecnologia.

Este artigo procura contribuir com a literatura através da análise de um sistema de GId, o *Blockstack,* implementado em dois *blockchains* diferentes*: Namecoin* e *Blockstack*. Também analisamos a possibilidade da implementação de sistemas de GId nos principais *blockchains* da atualidade: *Bitcoin*, Litecoin, *Namecoin* e *Ethereum*, por meio de uma análise comparativa destes *blockchains*.

2. Implementação de GId em *blockchains*

Nesta seção, apresentamos o exemplo de um sistema de GId implementado em *blockchain, o Blockstack,* e comparamos os principais *blockchains* da atualidade: *Bitcoin*, *Ethereum*, *Litecoin* e *Namecoin* quanto às características técnicas que mais influenciam o funcionamento e desempenho de sistemas de GId implementados em *blockchain*. Antes do exemplo e da comparação se faz necessário comentar um pouco sobre o funcionamento e características técnicas de um *blockchain*.

O *blockchain* foi originalmente concebido para armazenar transações financeiras e, por esta razão, são necessárias adaptações em sua estrutura de forma a armazenar informações relativas à gestão de identidades. A solução encontrada na adaptação de *blockchain* de criptomoedas é utilizar o campo reservado ao comentário de transações financeiras para armazenar nomes relativos aos sistemas de GId.

Por não terem sido concebidos para o armazenamento de dados, os blockchains de criptomoedas possuem baixa capacidade de armazenamento e seu *throughput,* relacionado com o intervalo de criação de novos blocos, é baixo. Além disso, para um *blockchain* ser considerado seguro, não pode haver concentração do processo de mineração e assim, impedir que alguns mineradores possam construir o *blockchain* em favorecimento próprio (*selfish mining*), desprezando outros usuários.

Desta forma, consideramos que as principais características que influenciam o funcionamento e desempenho de um sistema de GId implementado em *blockchain* são: concentração do processo de mineração em torno de poucos usuários, intervalo entre criação novos blocos e capacidade de armazenamento de dados.

2.1. *Blockstack*

O *Blockstack* foi originalmente implementado utilizando o *Namecoin*, uma ramificação do *blockchain* do Bitcoin, voltado especificamente para o armazenamento de nomes de domínio. Por terem a mesma origem, o *Namecoin* e o *Bitcoin* possuem as mesmas características técnicas (intervalo entre criação de novos blocos e mecanismo de mineração) [Namecoin 2016]. No entanto, a implementação do *Blockstack* utilizando o *Namecoin* não foi bem-sucedida, uma vez que ele possui concentração de mineração e os desenvolvedores do Blockstack perceberam que os mineradores criavam novos blocos seguindo conveniência e interesses próprios [Muneeb et al 2016].

Após as dificuldades enfrentadas com o *Namecoin,* o *Blockstack* foi então implementado no *blockchain* do *Bitcoin*. Ele passou a ganhar em confiabilidade, mas continuou com os seguintes problemas: baixo *throughput* e capacidade de armazenamento de dados.

Apesar da bem-sucedida implementação do *Blockstack* utilizando-se o *blockchain* do *Bitcoin*, consideramos que este pode vir a apresentar problemas de

concentração de mineração em um futuro próximo devido ao elevado consumo de energia elétrica [Malone 2013], queda na rentabilidade de seu processo de mineração e surgimento de *pools* de mineração com poder computacional com vez maior [Gautham 2016].

Além disso, serviços no *blockchain* do *Bitcoin* podem vir a se tornar inviáveis no futuro devido à competição com transações financeiras, pois o *Bitcoin* possui um tamanho máximo de bloco igual a 1MB, limitando o número de suas transações [Bitcoin Beta 2016]. Soma-se a isto o fato de as taxas das transações em *Bitcoin* serem cobradas de acordo com o tamanho do bloco [Antonopoulos 2014] e que blocos muito grandes são evitados pelos mineradores [Bitcoin Beta 2016]. Todos estes são os fatores que nos levam a crer que algumas características do *blockchain* do *Bitcoin* necessitam ser modificadas para que ele seja adotado como plataforma para o desenvolvimento de sistemas de gestão de identidades.

2.2. *Comparação das características de um blockchain que mais influenciam o desempenho um sistema de GId*

O primeiro *blockchain* a surgir foi o do *Bitcoin;* o *Namecoin*, por sua vez, surgiu com uma ramificação do *Bitcoin.* Já o *Litecoin* surgiu como uma proposta de melhoria de alguns aspectos do *Bitcoin*: menor intervalo entre blocos e mitigação ao problema de concentração de mineração em torno de poucos usuários, o *Ethereum*, por fim, surgiu como uma proposta de desenvolvimento de serviços e aplicações em *blockchain*.

Os *blockchains* do *Namecoin* e *Bitcoin* são muito parecidos, sendo que a única diferença entre eles é que o *Bitcoin* atualmente possui maior quantidade de mineradores, o que faz com que não haja concentração de mineração. O *blockchain* do *Litecoin* é interessante para o desenvolvimento de aplicações de gestão de identidades, pois possui menor intervalo de criação de blocos que o *Bitcoin*, proporcionando um maior *throughput* de armazenamento de dados. Já o blockchain do *Ethereum* também possui pequeno intervalo entre criação de novos blocos, tamanho de bloco um pouco menor quando comparado aos *blockchains* do *Bitcoin* e *Litecoin*, proposta de mineração voltada para evitar concentração de mineração e seu desenvolvimento foi direcionado para o desenvolvimento de aplicações em *blockchain*.

A seguir, as características de um *blockchain* que interferem na implementação de um sistema de gestão de identidade estão sumarizadas na Tabela 1.

Tabela 1 – Comparativo dos *blockchain* de *Bitcoin*, *Litecoin*, *Namecoin* e *Ethereum* quanto às características que influenciam o desempenho de sistemas de GId

	Bitcoin	*Litecoin*	*Namecoin*	*Ethereum*
Possibilidade de concentração de mineração	Sim	Sim	Sim	Não
Tamanho máximo do bloco	1 MB	1 MB	520 bytes	780kB
Intervalo entre criação de novos blocos	10 min	2,5 min	10 min	12 s

3. Conclusão

O uso de *blockchains* em sistemas de GId permite a construção de sistemas

descentralizados, no entanto, ainda é necessário vencer vários obstáculos técnicos para consolidar a implementação destes sistemas em *blockchain*.

Os principais obstáculos a serem solucionados são os baixos *throughput* e capacidade de armazenamento de dados, elevado consumo de energia elétrica durante o processo de mineração, possibilidade de concentração deste processo em torno de poucos usuários e competição com transações financeiras.

A implementação do *Blockstack* no *blockchain* do Namecoin mostrou o quanto a concentração de mineração influencia na segurança de um sistema de GId e que *blockchain* do *Bitcoin* pode ser considerado atualmente seguro para implementação de sistemas de GId, porém, devido à queda na rentabilidade do processo de mineração do Bitcoin, há a tendência de que seu processo de mineração se concentre cada vez mais, podendo trazer riscos à segurança e confiabilidade uso de sistemas de GId implementados neste blockchain.

Dentre os *blockchains* analisados, o do *Litecoin* oferece maior taxa de armazenamento que o *Bitcoin*, e o do *Ethereum*, por sua vez, oferece grandes possibilidades à implementação eficaz de gestão de identidades, pois possui características atraentes como baixo intervalo entre criação de novos blocos, política de evitar concentração de mineração e tamanho ilimitado de bloco.

Referências

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system..

Zyskind, Guy et al. Decentralizing privacy: Using blockchain to protect personal data. In: Security and Privacy Workshops (SPW), 2015 IEEE. IEEE, 2015. p. 180-184.

Stevenson, J. (2013). Getting started with Litecoins (after Bitcoin). John Stevenson.

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper.

Namecoin (2016). https://namecoin.info (acessado em 10 de Setembro de 2016).

Bitcoin Beta (2016). http://bitcoin.stackexchange.com/questions/21522/is-there-a-limit-on-number-of-transactions-included-in-a-block (acesso em 11 de Setembro de 2016).

Antonopoulos, Andreas M. Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.", 2014.

Ali, Muneeb et al. (2016) Blockstack: A Global Naming and Storage System Secured by Blockchains. In: 2016 USENIX Annual Technical Conference (USENIX ATC 16).

O'DWYER, Karl J.; MALONE, David. Bitcoin mining and its energy footprint. In: Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET. IET, 2013. p. 280-285.

Gautham, N. (2016). KnCMiner in Trouble, Declares Bankruptcy, 1 de Junho de 2016, disponível em: http://btcnews.com/kncminer-declares-bankruptcy (acessado em 3 de Julho de 2016).

XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

Niterói, RJ, novembro de 2016

WFC – V Workshop de Forense Computacional

HoneySELK: Um Ambiente para Pesquisa e Visualização de Ataques Cibernéticos em Tempo Real

Gildásio Antonio de Oliveira Júnior[1], Rafael Timóteo de Sousa Júnior[1], Robson Oliveira de Albuquerque[1], Edna Dias Canedo[2], André Grégio[3]

[1]Departamento de Engenharia Elétrica – Universidade de Brasília (UNB)
[2]Faculdade UnB Gama – Universidade de Brasília (UNB)
Campus Universitário Darcy Ribeiro, Asa Norte – 70910-900 –Brasília – DF – Brasil

[3]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Centro Politécnico – Cx. Postal 19081 – CEP 81531-980 – Curitiba – PR – Brasil

jrgildasio@gmail.com, desousa@unb.com, robson@redes.unb.br,
ednacanedo@unb.br, gregio@inf.ufpr.br

Abstract. *Due to the large number of vulnerabilities in information systems and the continuous activity of attackers, techniques for malicious traffic detection are required to identify and protect against cyber-attacks. Therefore, it is important to intentionally operate a cyber environment to be invaded and compromised in order to allow security professionals to analyze the evolution of the various attacks and exploited vulnerabilities. In this paper, we propose a security architecture deployed specifically for research and information gathering from the attackers' point of view. This architecture, named HoneySELK, is described and its results are evaluated to show its ability to collect, analyze, and visualize large amount of cyber-attacks in real time.*

Resumo. *Dado o grande número de vulnerabilidades em sistemas de informação e a atividade contínua dos atacantes, é cada vez mais necessário usar técnicas de detecção de tráfego malicioso para identificação e proteção contra ciberataques. Portanto, é importante operacionalizar intencionalmente um ambiente cibernético para ser invadido e comprometido, a fim de permitir que profissionais de segurança analisem a evolução dos diversos ataques e vulnerabilidades exploradas. Este trabalho propõe uma arquitetura projetada para a pesquisa e a obtenção de informações do ponto de vista dos atacantes. A solução, denominada HoneySELK, foi implementada e seus resultados avaliados para mostrar sua capacidade de coletar, analisar e visualizar uma grande quantidade de ataques cibernéticos em tempo real.*

1. Introdução

A rápida expansão do volume de informações acessadas através da Internet atraiu novas formas de atividades intrusivas. Assim, é fundamental desenvolver técnicas para acompanhar a evolução de ataques, bem como estruturar proteções contra diversas ameaças cibernéticas. Ressalta-se a necessidade de sempre atualizar as ferramentas e técnicas aplicadas nesse contexto, com a finalidade de combater ataques ou de proteger os dados, para assegurar a segurança de redes e sistemas, uma vez que a quantidade de incidentes de redes só aumenta, mesmo com a aplicação de variadas técnicas de defesa e vultosos gastos em segurança [CERT.br 2016] [CTIR.gov 2016].

Quanto à detecção de intrusão, empregam-se diversas tecnologias específicas em conjunto com outros mecanismos de segurança, buscando indícios da ocorrência de ataques. Segundo [Scarfone and Mell 2007], a detecção de intrusão é o processo de monitoramento de eventos que ocorre em um sistema de computador ou rede para detectar sinais de possíveis incidentes. Um IDS (*Intrusion Detection System*), sistema de suporte a tal processo tem como requisito natural a detecção de ataques que sejam conhecidos previamente, ou seja, ataques que tenham alguma assinatura verificável. De mesma importância é o requisito complementar de coletar, analisar e visualizar os dados de tráfego que incluam possíveis ataques ainda desconhecidos, preferencialmente em tempo real. Para tanto, o IDS deve detectar também indícios de ataques que correspondam a anomalias no tráfego ou na operação de sistemas. A detecção por anomalia, mesmo efetiva como método para bloquear os ataques, deixa em aberto a necessidade de analisar detalhadamente cada anomalia para descrever o ataque e então eventualmente obter a assinatura correspondente.

Nesse sentido, o presente trabalho propõe uma solução para a análise dos ataques, tanto daqueles conhecidos (com base em assinatura) quanto dos desconhecidos (com base em anomalia). O ambiente proposto, denominado *HoneySELK*, é deixado propositadamente ao alcance dos atacantes para que, quando comprometido, seja utilizado para controlar, capturar, analisar e visualizar em tempo real os diversos tipos de ataques e as correspondentes vulnerabilidades exploradas pelos intrusos. O ambiente é também de interesse forense, visto que tem a possibilidade de capturar o *modus operandi* das operações de intrusão, sejam elas por atividade de *malware* ou ataques direcionados. Com isso, é possível preservar evidências dos atos praticados por um atacante e que possam indicar a ocorrência de crimes cibernéticos, assim como caracterizar os meios tecnológicos empregados em tais delitos.

Este artigo está organizado da seguinte forma: na Seção 2 apresentam-se alguns trabalhos relacionados; na Seção 3, é detalhada a arquitetura do ambiente *HoneySELK* proposto propositadamente para ser invadido e comprometido; a Seção 4 descreve alguns ataques que foram capturados no ambiente *HoneySELK* e, por fim, a Seção 5 apresenta as considerações finais e propostas de trabalhos futuros.

2. Trabalhos Relacionados

Referências acerca de monitoração e análise de atividades intrusivas surgiram na década de 1980 e vêm constituindo uma área de intensa investigação científica, incluindo os trabalhos iniciais de [Stoll 1998] e [Cheswick 1990], o surgimento de *honeypots* baseados em *software* [Cohen 1998] e o Projeto *Honeynet* [Project 2002].

Métodos baseados na análise de componentes principais (PCA) [Almotairi et al. 2009] são uma alternativa promissora às técnicas tradicionais, por identificar os principais grupos de indicadores altamente correlacionados e que representam atividades maliciosas relevantes em dados de tráfego de rede coletados em *honeypots*. Considerando que padrões de tráfego de ataques são mais correlacionados do que o padrão de tráfego normal, métodos PC dependem apenas da análise estatística dos dados recolhidos, o que permite evitar ou reduzir a necessidade de conhecimento prévio sobre os ataques a serem detectados, ou seja, não necessitam de treinamento para reconhecer os ataques e separá-los do tráfego legítimo. Embora tais características os tornem adequados para a detecção automática de ataques, métodos PCA ainda exigem

intervenção humana, o que dificulta a análise automática e causa erros. Um método automático para identificar ataques de rede com coletas de *honeypot* é proposto em [David et al. 2011] e [Costa et al. 2012], usando esquemas de seleção de ordem do modelo, permitindo a implementação eficiente em *hardware* e em computação paralela.

Em todo caso, os *honeypots* coletam um substancial volume de tráfego e *logs* de atividade, colocando um verdadeiro desafio quanto à análise eficiente e automatizada desses dados, bem como a sua visualização para interpretação humana. Assim, [Siqueira et al. 2015] mostram uma arquitetura para extrair e visualizar informações de tráfego geradas por *malware* e conexões a *honeypots*, verificando que diversos *malwares* utilizam o método GET do HTTP para obtenção de poder sobre componentes da máquina infectada e que um serviço dos mais explorados nos *honeypots* é o SMB, na porta 445. Apesar de apresentar detalhes da extração dos dados, a análise do tráfego nesse trabalho é feita em apenas um dos componentes, tornando impossível a correlação entre duas fontes de dados diferentes. Já [Oliveira Jr et al. 2015] apresentam uma *honeynet* de alta interatividade projetada especificamente para pesquisa e preservação de evidências de ataques para efeito forense, possibilitando várias análises detalhadas em diversos tipos de ataques.

3. Arquitetura do Ambiente *HoneySELK*

Esta seção descreve os aspectos relacionados ao desenvolvimento do *HoneySELK*, uma *honeynet* virtual de alta interatividade, cuja arquitetura proposta (Figura 1) tem como objetivo controlar, capturar, analisar e visualizar ataques novos e desconhecidos, em ambiente de laboratório de pesquisa.

Como requisito fundamental, o endereço do *HoneySELK* não consta em nenhum mecanismo de resolução de nomes públicos nem é divulgado na Internet. Assim, para alcançar esse endereço o atacante tem de fazer varredura na Internet, o que por si já indica uma atividade não convencional. Encontrado tal endereço, os serviços podem ser descobertos pelo atacante por mapeamento de portas. O tráfego capturado no ambiente é suspeito, pois resulta de atividade maliciosa de mapeamento de serviços e conteúdo.

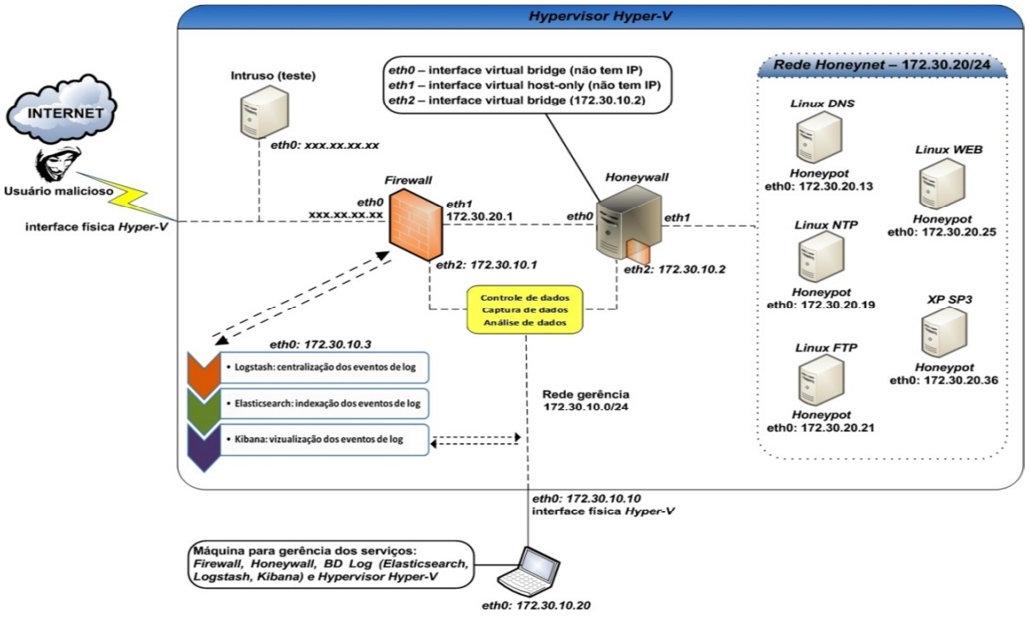


Figura 1. Arquitetura do ambiente *HoneySELK*.

O desenvolvimento da arquitetura ocorreu em quatro fases ([Project 2002], [Spitzner 2002] [Oliveira Jr et al. 2015]), detalhadas nas seções seguintes: a fase 1 aborda a arquitetura proposta e o modelo de solução; a fase 2 trata do controle de dados; a fase 3, da captura de dados; e a fase 4, dos aspectos da visualização dos ataques. A arquitetura física *HoneySELK* (Figura 1) emprega um hospedeiro (*host*) ligado a um roteador diretamente na Internet. Nesse hospedeiro, há nove convidados (*guests*).

3.1. Fase 1: Arquitetura Proposta

O *HypervisorHyper-V* [Carvalho 2012] foi implementado para criar a estrutura lógica e de roteamento do *HoneySELK*. Toda a estrutura do projeto foi feita em apenas um único *host* (*Dell PowerEdge* R610). A configuração e gerência dos sistemas convidados no *Hyper-V* é feita através de uma máquina dedicada para esta finalidade (Figura 2). Todas as interfaces de redes virtuais são criadas no *Hyper-V* e divididas conforme suas funções para *firewall*, gerência, *honeynet* e acesso externo.

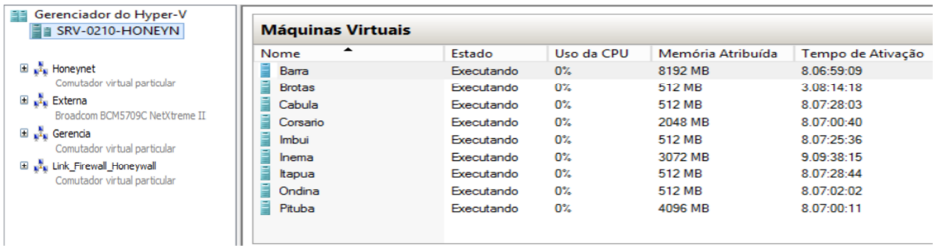


Figura 2. Gerência do *Hyper-V*.

O ambiente foi projetado para ter três redes distintas: i) a Internet, considerada não confiável por ser o lugar de onde os ataques são originados; ii) a rede *honeynet*, com endereço reservado, integrada por um conjunto de *honeypots* a serem comprometidos; e iii) a rede de gerência dos recursos de *honeywall*, *firewall* e da pilha formada por *Elasticsearch*, *Logstash* e *Kibana* (ELK) [Elastic 2016].

O *honeywall* [The Honeynet Project2008] é configurado com três interfaces virtuais. A primeira interface virtual *eth0* comunica-se com o *firewall*, a segunda interface virtual *eth1* é utilizada para se comunicar com a rede *honeynet*, e a terceira interface *eth2* é utilizada para gerência e coleta de dados do *honeywall*. As interfaces virtuais *eth0* e *eth1* estão configuradas como *bridge*, portanto não possuem endereço IP. O funcionamento deste dispositivo na camada 2 apresenta duas grandes vantagens: a primeira é que não há *hops* de roteamento nem decremento do *Time To Live* (TTL) no cabeçalho IP; a segunda é a dificuldade por parte dos atacantes em detectar o ambiente, já que não responde por requisições TCP/IP em tais interfaces.

O *firewall* é configurado com regras de direcionamento das portas para cada *honeypot*. Além disso, todo o *log* de entrada e saída deste dispositivo é enviado para o servidor ELK. O *Elasticsearch* é desenhado para suportar grandes volumes de dados e fazer sua indexação. Assim, o ELK tem por finalidade realizar o armazenamento distribuído da estrutura completa dos dados do monitoramento em tempo real dos ataques, com dados de georreferenciamentos, estatísticas e grafos, indicando relacionamentos diversos, que auxiliam na identificação e *modus operandi* de atacantes.

Todos os *honeypots* são implementados na rede *honeynet*, configurada como *host-only* (rede virtual privada) para fazer a comunicação entre os *honeypots* e a interface

virtual *eth1* do *honeywall*. A rede de gerência é considerada como confiável e é utilizada para coletar e analisar os dados. Esta rede administra ainda o *firewall*, o *honeywall* e o ELK. A gerência cabe a um *host* dedicado e exclusivo para esta finalidade.

Todos os *honeypots* são configurados com a instalação padrão do *Linux Debian Jessie 8.4* e *Windows* XP SP3. O uso do XP se justifica por ser um sistema ainda em uso em diversos ambientes, sendo alvo constante de *malware*. Foram feitas instalações e configurações de serviços tais como DNS, FTP, NTP, MySQL, HTTP, HTTPS, MSRPC, NETBIOS-SSN, Microsoft-DS, TELNET e SSH (todos com suas portas padrão). Não se aplicou nenhum processo de *hardening* para manter os sistemas mais seguros.

3.2. Fase 2: Controle de Dados

O controle de tráfego no ambiente *HoneySELK*, seja para dados recebidos ou enviados, tem como finalidade filtrar quais dados podem ir para qual destino. Foram definidas quatro regras básicas para controlar o fluxo do tráfego: a primeira - qualquer indivíduo poderá realizar uma conexão da Internet para a *honeynet*, pois isso permite que um atacante explore os *honeypots*; a segunda - o *firewall* controlará conexões feitas da rede *honeynet* com a Internet para evitar que os atacantes usem os *honeypots* comprometidos para acometer outros sistemas. Esta regra é replicada também no *firewall* do *honeywall*, para que haja uma redundância de controle de fluxo; a terceira - apenas os *honeypots* DNS e NTP poderão realizar conexões ilimitadas para fora do ambiente, uma vez que eles serão utilizados pelos outros *honeypots* para resolução de nomes e sincronismo de tempo; quarta - a rede *honeynet* e a rede gerência não poderão se comunicar. Isso evita que os *honeypots* comprometidos modifiquem ou destruam os dados coletados.

Além disso, o *honeywall* é utilizado para prevenir ataques de dentro da rede *honeynet* para outros sistemas. O principal objetivo é limitar o número de conexões (TCP, UDP e ICMP) que podem ser feitas para fora da rede *honeynet* em uma escala de tempo configurada conforme a necessidade da análise.

3.3. Fase 3: Captura de Dados

A captura de dados tem como finalidade coletar todas as atividades que ocorrem dentro da rede *honeynet*. Quanto maior o número de camadas (métodos de captura), maior a possibilidade de novos ataques serem detectados. Na arquitetura, o software detector de intrusões ([SNORT2016]), está no *honeywall* e configurado com regras atualizadas. Sua finalidade é capturar todo o tráfego da interface virtual *eth1* do *honeywall*, ou seja, todo o fluxo de entrada e saída da rede *honeynet*. O *snort* também foi configurado para converter quaisquer informações ASCII encontradas no *payload* do pacote. Este procedimento é fundamental para analisar rapidamente as seções de texto simples, tais como as seções de FTP, TELNET ou qualquer outra em texto claro.

3.4. Fase 4: Visualização dos ataques

A visualização dos ataques no ambiente *HoneySELK* tem como objetivo principal facilitar a interpretação de ataques para que se seja possível aplicar contramedidas eficientes. Além disso, o monitoramento em tempo real permite observar endereços IP de origem e destino, protocolo utilizado, porta, georreferenciamentos e fazer mapeamento dos ataques através de grafos.

Este processo é implementado no servidor ELK através de quatro ferramentas: *Logstash* (centralização dos dados), *Elasticsearch* (indexação e busca de grande volume de dados), *Kibana* (visualização dos dados) e *Graph* (relacionamento entre as entidades) [Elastic2016]. Para fazer o georreferenciamento dos ataques utilizou-se a base de dados *GeoLiteCity* [MAXMIND 2016].

4. Análise dos Ataques no Ambiente *HoneySELK*

A solução em produção indica como a arquitetura proposta pode ser utilizada como fonte de pesquisa para coletar, analisar, visualizar e estudar ataques cibernéticos e vulnerabilidades exploradas em sistemas de rede. Alguns dos resultados obtidos até a conclusão deste artigo são apresentados nos tópicos seguintes.

4.1. Visualização geográfica e estatística dos ataques

A solução foi implementada em laboratório de pesquisa da Universidade de Brasília, do qual os autores são pesquisadores ou atuam em conjunto. A coleta dos dados ocorreu entre os dias 08/junho e 09/agosto de 2016. Na Figura 3a pode-se verificar a quantidade total de conexões realizadas nos dias 14/junho (5.434), 27/junho (9.730) e 03/agosto (10.057), sendo detalhada na seção 4.2 as conexões do dia 14/junho. Observa-se uma queda entre os dias 18 e 28 de julho devido a janelas de manutenção na infraestrutura do laboratório. Nota-se também por outro lado que 163 países fizeram 89.991 tentativas de acessos através de 18.359 IPs diferentes (Figura 3b).

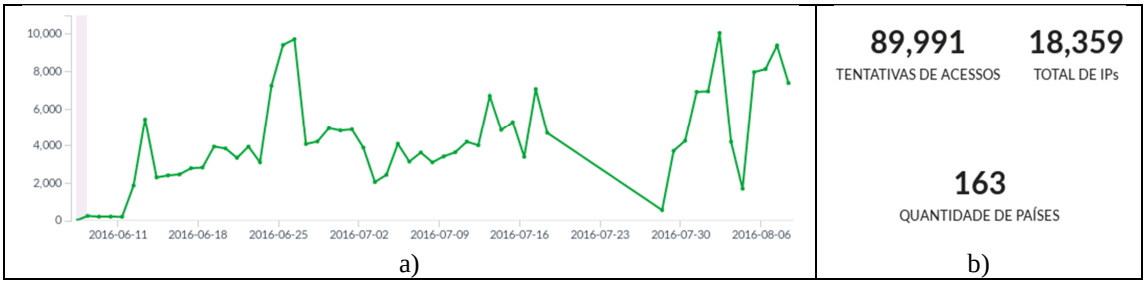


Figura 3. Captura de conexões entre 08/junho e 09/agosto.

Na Figura 4 pode-se verificar no mapa os ataques representados por círculos e divididos em cores (vermelha, laranja e amarela) com a localização geográfica dos IPs de origem. Tais informações podem ser detalhada em país, estado e cidade, por meio das coordenadas geográficas geradas pela centralização, indexação, busca e visualização dos *logs* obtidos do *firewall*. Percebe-se também um circulo vermelho e outro laranja no mapa indicando a origem dos IPs que mais atacaram o ambiente (China).

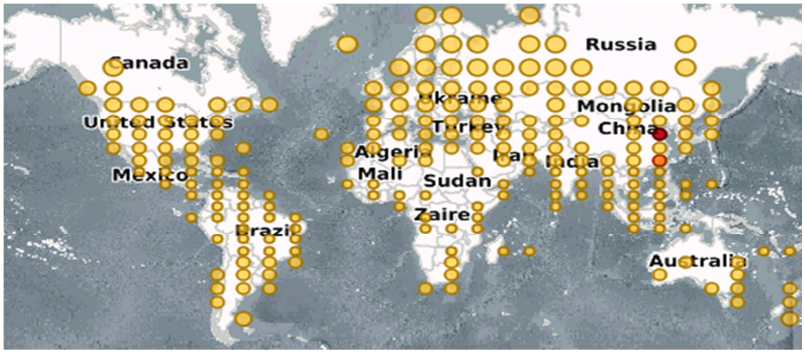


Figura 4. Visualização geográfica dos ataques.

A Figura 5 classifica os ataques por países, IPs e serviços. As Figuras 5a e 5d apresentam as estatísticas dos ataques por país de origem, sendo a China e o Brasil os países com maior quantidade de requisições, havendo 40.573 tentativas de ataques da China no dia da análise. As Figuras 5b e 5e representam as estatísticas dos ataques direcionados ao ambiente por IPs, sendo o IP 116.31.116.21 aquele com a maior quantidade de requisições (13.664). As Figuras 5c e 5f apresentam os ataques por serviços, especificamente 23/TELNET, 21/FTP, 22/SSH, 445/MICROSOFT-DS, 80/HTTP, 139/NETBIOS-SSN, 135/MSRCP, 443/HTTPS, 123/NTP, 3306/MYSQL e 53/DNS. É possível verificar através da Figura 5f que as portas 22 (42.528) e 23 (29.092) foram as mais visitadas.

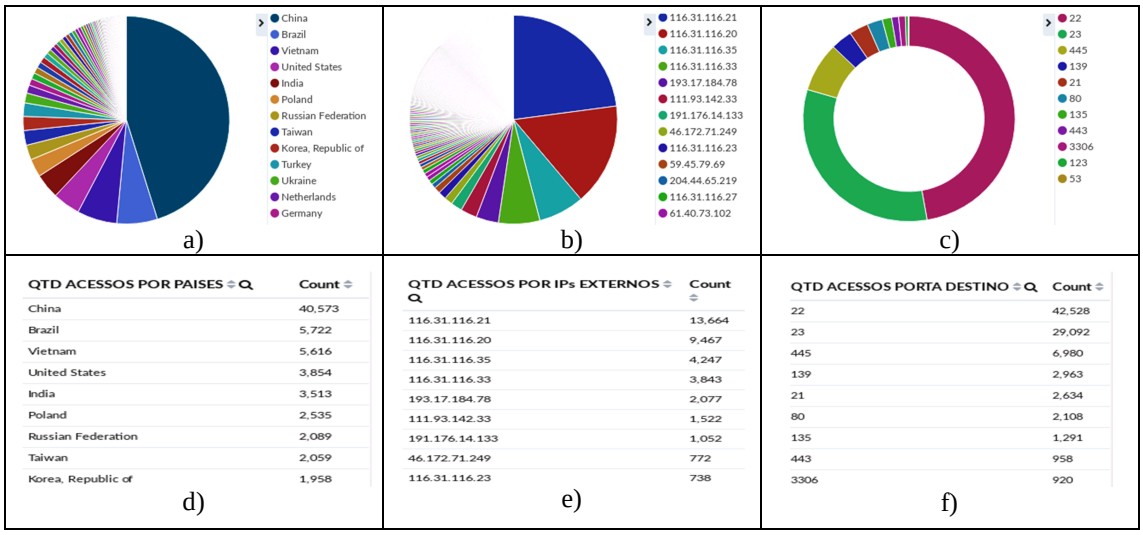


Figura 5. Distribuição de ataques por países, IPs e serviços.

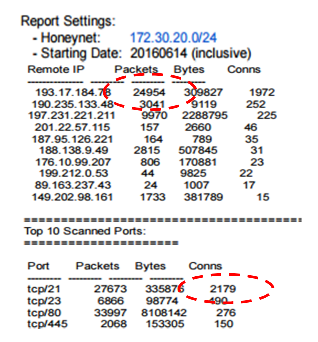
Figura 6. Nuvem de países e IPs que estão atacando o *HoneySELK*.

Informações sobre os países origem de ataques ao ambiente são observadas também na nuvem da Figura 6a, onde se destacam em vermelho (China) e laranja (Brasil, Vietnam, Estados Unidos e Índia) aqueles com maior quantidade de requisições ao ambiente. Com um filtro (*geoip.country_name:"china"*), podemos verificar também a nuvem de IPs (Figura 6b) que pertencem a China, inclusive os IPs (116.31.116.21, 116.31.116.20, 116.31.116.35 e 116.31.116.33) com maior quantidade de requisições. Tais informações podem ser postas em *blacklist* para bloquear a origem dos ataques.

4.2. Analisando os ataques direcionados ao ambiente

No dia 14 de junho, verificou-se um alerta recebido por *e-mail* pelo ambiente, indicando que o IP 193.17.184.78 (origem na Polônia) havia enviado sozinho 24.954 pacotes e que a porta 21 (FTP) estava com 2.179 requisições (Figura 7a). Para se obter informações

mais detalhadas, analisou-se o *payload* dos pacotes, permitindo confirmar que o usuário *admin* e a senha *carole* (Figura 7b) foram utilizados pelo atacante para tentar realizar a autenticação no servidor. Logo, o IP 193.17.184.78 realizara um ataque de força bruta na porta 21. O detalhamento da função de georreferenciamento permitiu verificar que este ataque veio da cidade de Varsóvia (Figura 8).

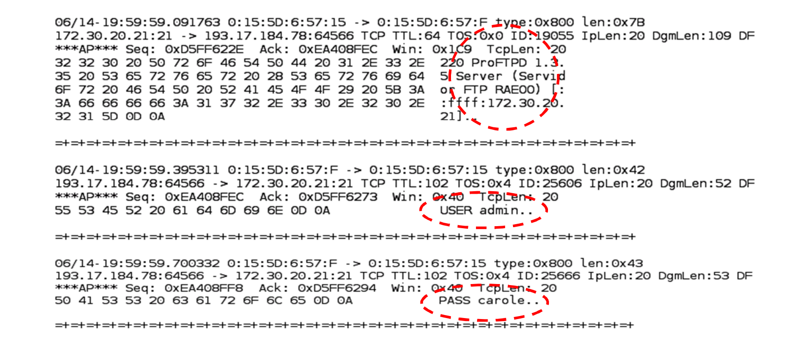


```
Report Settings:
  - Honeynet:       172.30.20.0/24
  - Starting Date:  20160614 (inclusive)
 Remote IP        Packets   Bytes     Conns
 ---------------  --------  --------  --------
 193.17.184.78     24954    309827     1972
 190.235.133.48     3041      9119      252
 197.231.221.211    9970   2288795      225
 201.22.57.115       157      2660       46
 187.95.126.221      164       789       35
 188.138.9.49       2815    507845       31
 176.10.99.207       806    170881       23
 199.212.0.53         44      9825       22
 89.163.237.43        24      1007       17
 149.202.98.161     1733    381789       15

==========================================
Top 10 Scanned Ports:
=====================

 Port      Packets   Bytes     Conns
 --------  --------  --------  --------
 tcp/21     27673    335876     2179
 tcp/23      6866     98774      490
 tcp/80     33997   8108142      276
 tcp/445     2068    153305      150
```

a)



```
06/14-19:59:59.091763 0:15:5D:6:57:15 -> 0:15:5D:6:57:F type:0x800 len:0x7B
172.30.20.21:21 -> 193.17.184.78:64566 TCP TTL:64 TOS:0x0 ID:19055 IpLen:20 DgmLen:109 DF
***AP*** Seq: 0xD5FF622E  Ack: 0xEA408FEC  Win: 0x1C9  TcpLen: 20
32 32 30 20 50 72 6F 46 54 50 44 20 31 2E 33 2E  220 ProFTPD 1.3.
35 20 53 65 72 76 65 72 20 28 53 65 72 76 69 64  5 Server (Servid
6F 72 20 46 54 50 20 52 41 45 4F 4F 29 20 5B 3A  or FTP RAEOO) [:
3A 66 66 66 66 3A 31 37 32 2E 33 30 2E 32 30 2E  :ffff:172.30.20.
32 31 5D 0D 0A                                   21]..

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

06/14-19:59:59.395311 0:15:5D:6:57:F -> 0:15:5D:6:57:15 type:0x800 len:0x42
193.17.184.78:64566 -> 172.30.20.21:21 TCP TTL:102 TOS:0x4 ID:25606 IpLen:20 DgmLen:52 DF
***AP*** Seq: 0xEA408FEC  Ack: 0xD5FF6273  Win: 0x40  TcpLen: 20
55 53 45 52 20 61 64 6D 69 6E 0D 0A              USER admin..

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

06/14-19:59:59.700332 0:15:5D:6:57:F -> 0:15:5D:6:57:15 type:0x800 len:0x43
193.17.184.78:64566 -> 172.30.20.21:21 TCP TTL:102 TOS:0x4 ID:25666 IpLen:20 DgmLen:53 DF
***AP*** Seq: 0xEA408FF8  Ack: 0xD5FF6294  Win: 0x40  TcpLen: 20
50 41 53 53 20 63 61 72 6F 6C 65 0D 0A           PASS carole..

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

b)

Figura 7. Detalhamento do ataque ao serviço FTP.

Figura 8. Visualização geográfica da origem do ataque no *Honeypot* FTP.

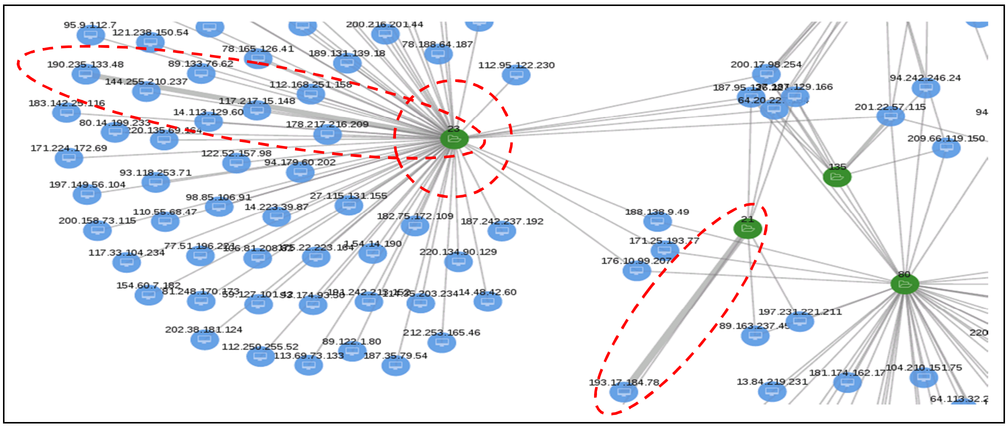


Figura 9. Grafo com relacionamento dos ataques entre IPs e serviços.

Para facilitar a visualização deste ataque, produziu-se um grafo com as informações dos documentos e termos do índice do dia 14/junho (Figura 9). Dessa forma, é possível verificar os IPs que tentaram atacar os serviços e as conexões mais significativas (IP 193.17.184.78 e IP 190.235.133.48). Observa-se uma quantidade maior de IPs atacando a porta 23 (telnet). Além disso, nota-se novamente pela largura da aresta que o IP 193.17.184.78 da Polônia foi quem fez o maior número de requisições direcionadas para a porta 21 (ftp).

4.3. Coleta e análise de *malware*

O ambiente também permite a detecção de artefatos maliciosos. No dia 20 de junho às 18h52mim foi encontrado um *script* malicioso com a finalidade única de fazer

download de arquivos tendo como origem o IP 185.22.172.238 (Figura 10a). A análise dos pacotes e do *payload* permitiu verificar esse *script* (*bin.sh*) capturado e suas funcionalidades básicas (Figura 10b). O detalhamento demonstrou que se tratava de um *script* para *download* de binários, um deles baixado via rede TOR para o ambiente para fins de análise forense. A ferramenta *ghex* mostrou que o binário estava em *Executable and Linking Format* (ELF) e empacotado com o *packer* UPX (Figura 10c).

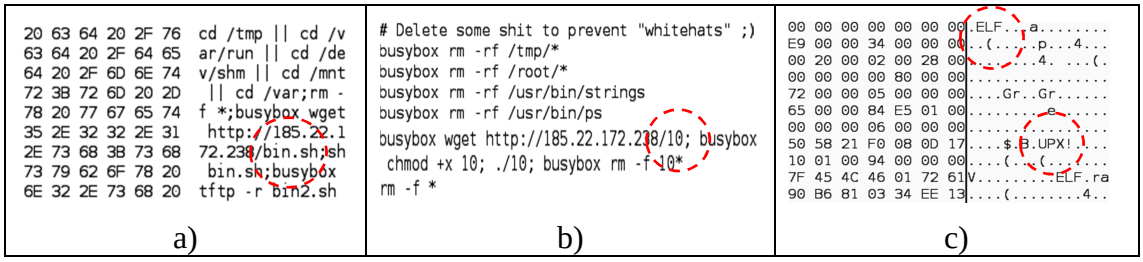


Figura 10. Análise do *script* e *malware* coletado.

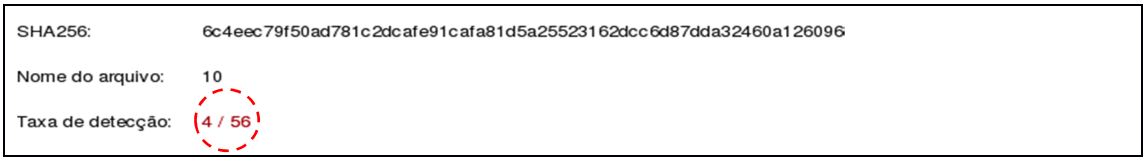


Figura 11. Análise básica do *malware* coletado.

De posse do binário, este foi submetido para análise pela plataforma virustotal (Figura 11). Interessante notar que a taxa de detecção do binário como sendo *malware* foi considerada baixa, já que apenas 4 de 56 plataformas de antivírus foram capazes de considerar que o binário era de fato *malware* no dia da análise.

5. Conclusão

Este trabalho propõe e descreve o ambiente *HoneySELK* como solução para analisar ataques e acompanhar novas formas de atividades de intrusos em redes de computadores.

O *HoneySELK* se mostrou capaz de capturar grande quantidade de ataques em tempo real, bem como *malware* difundido por atacantes. A solução permite análise georreferenciada dos endereços IPs de origem e extração de informações estatísticas de serviços atacados e dados quantitativos segundo critérios que podem ser configurados de acordo com a necessidade. Como resultados da solução, foi detalhado um ataque ao ambiente e a análise de um artefato malicioso capturado pelo ambiente. A análise dos resultados indica que tais técnicas permitem a utilização do ambiente proposto em diversas aplicações de rede, inclusive para análise forense de ataques. A solução permite ainda a visualização de detalhes de ataques, seja com o intuito de atualizar medidas de proteção, seja para efeito de demonstração de técnicas utilizadas pelos atacantes.

Como trabalhos futuros, objetiva-se replicar o ambiente através de *honeypots* distribuídos, estender a quantidade e tipos de *honeypots* para que se possa pesquisar outras formas de ataques e realizar análise forense mais detalhada de *honeypots* que possam ter sido comprometidos. Para fins de diferenciação de origem, é importante fazer o mapeamento de georreferenciamento dos IPs de redes anônimas como TOR e I2P. Também se entende como necessário extrair e analisar dados a partir de *dumps* de memória e incluir *honeypots* com sistemas operacionais utilizados por *smartphones*.

Agradecimentos

Os autores agradecem às Agências brasileiras de pesquisa, desenvolvimento e inovação, CAPES (Projeto FORTE 23038.007604/2014-69) e FINEP (Projeto RENASCI/PROTO 01.12.0555.00), bem como ao DITEC/DPF/MJ (Contrato 36/2010 DPF-FUB Mestrado Profissional em Informática Forense), à DIPLA/MP (TED 05/2016 DIPLA-FUB) e à DPGU (TED 66/2016 DPU-FUB), pelo apoio ao trabalho.

Referências

Almotairi S., Clark A., Mohay G., Zimmermann J. (2009). A technique for detecting new attacks in low-interaction honeypot traffic, Proceedings of the 2009 Fourth International Conference on Internet Monitoring Protection. IEEE Computer Society, pp. 7–13.

Carvalho, L. (2012). Windows Server 2012 Hyper-V Cookbook. ISBN 978-1-84968-442-2, Birmingham B3 2PB, UK.

Costa J. P. C. L. da, Freitas E. P. de, David B. M., Serrano A. M. R., Amaral D., Sousa Jr R. T. de (2012). Improved blind automatic malicious activity detection in honeypot data, The Sixth International Conferenceon Forensic Computer Science.

CERT.br (2016). Incidentes Reportados ao CERT.br – Janeiro a Dezembro de 2015, disponível em: http://www.cert.br/stats/incidentes/2015-jan-dec/analise.html.

Cheswick, B. (1990). An evening with berferd in which a cracker is lured, endured, studied. *Proc. Winter USENIX Conference*. [S.l.: s.n.], p. 163–174.

Cohen F. (1998). A Note on the Role of Deception in Information Protection, disponível em: http://all.net/journal/deception/deception.html/.

CTIR.gov (2016). Estatísticas de Tratamento de Incidentes de Rede na APF, disponível em: http://www.ctir.gov.br/estatisticas.html.

David B. M., Costa J. P. C. L. da, Nascimento A. C. A., Holtz M. D., Amaral D., Sousa Jr R. T. de (2011). Blind automatic malicious activity detection in honeypot data, pp. 02-04, The Fifth International Conference on Forensic Computer Science.

Elastic (2016). Elastic Stack Product Documentation, disponível em, https://www.elastic.co/guide/index.html.

Kali Linux (2016). Our Most Advanced Penetration Testing Distribution, Ever, disponível em https://www.kali.org.

Maxmind (2016). GeoIP Databases & Services: Industry Leading IP Intelligence, disponível em: https://www.maxmind.com/en/geoip2-services-databases.

Oliveira Jr, G. A., Sousa Jr, R. T.de, Tenório, D. F. (2015). Desenvolvimento de um Ambiente Honeynet Virtual para Aplicação Governamental. *In: The Ninth International Conferenceon Forensic Computer Science*. v. 1. p. 70-80.

Project, Honeynet. (2002). Conheça seu inimigo - O Projeto Honeynet. São Paulo: Pearson Education do Brasil.

Scarfone, K., Mell, P. (2007). Guide to Intrusion Detection Prevention Systems (IDPS). Recommendations of the National Institute of Stards Technology, Gaithersburg.

Siqueira, H. R. A., Baruque, A. C., Geus, P. L., Grégio, A. R. A. (2015). Uma Arquitetura para Análise e Visualização de Tráfego de Rede Malicioso. *XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg)*, Florianópolis, SC, Brazil.

SNORT (2016). Snort Users Manual, disponível em https://www.snort.org/documents.

Spitzner, L. Honeypots (2002). Tracking Hackers. Indianápolis, IN: Addison Wesley.

Stoll, C. (1988). Stalking the wily hacker. *Commun. ACM, New York, NY, USA*, v. 31, n. 5, p. 484–497.

The Honeynet Project (2008). User's Manual, disponível em: http://old.honeynet.org/index.html.

Sampling and similarity hashes in digital forensics: An efficient approach to find needles in a haystack

Vitor Hugo Galhardo Moia[1], **Marco Aurélio A. Henriques**[1]

[1]School of Electrical and Computer Engineering (FEEC)
University of Campinas (UNICAMP)
Campinas, SP, Brasil 13083-852

vhgmoia,marco@dca.fee.unicamp.br

Abstract. *The amount of data handled by forensics examiners has grown significantly over the past few years. Investigations involving digital devices used to store valuable information related to crimes are becoming very common. Methods aiming to reduce the time spent in each case and yet be effective in finding evidences are necessary. Hash-based functions appeared as a solution, but they are limited only to yes/no answers, failing in detecting similarities among objects. New approaches have been proposed to solve this problem as, for example, the similarity hash. This function can find small-scale similarities in objects with different contents. However, such flexibility comes with a price: The high cost compared to traditional approaches. In this paper, we propose and evaluate the combination of similarity hashes and sampling techniques, where only a small portion of a seized media is analyzed to produce reliable results for a triage process in a short period of time. We also propose to apply similarity hashes to the sector level and present a new approach of clustering sectors to reduce even more the investigation time. The proposed approach can reliably identify even small fragments of deleted files that still remains in a seized media.*

1. Introduction

With the increasing volume of data in today's world, forensics examiners face a significant challenge in analyzing seized devices. Cases where digital electronics are used to store data related to crime are increasing. Users store more and more data as storage capacity increases and prices get cheaper. On the other hand, forensic professionals normally do not have the resources to scale in the same way. Methods capable of evaluating seized devices in a reasonable time become necessary to deal with this trend.

Hash-based techniques are an efficient solution for identifying objects in investigations. Forensics examiners create hashes for the seized media objects and look up for matches in a database containing hashes of interest objects. However, hash functions fail in identifying similar objects. They only give yes/no answers. If two files are identical, their hashes will be precisely the same, but if they differ at least by one bit, their hashes will be completely different. This way, documents with small changes will not be detected, and examiners may lose important evidences.

To overcome such limitation, researchers have employed similarity hashes, which are similar to hash functions in using unique representations for data, but with the difference that small variations will reflect in small changes in the digest. They can create representations that allow us to compare two objects and give a similarity value for them.

In this work, we use sdhash as our similarity tool [Roussev 2010], although it is very expensive in relation to traditional hash functions. We present an approach to minimize its high cost, combining it with statistical sampling techniques to reduce the amount of data analyzed and hence the number of similarity hashes generated. Also, we work at the disk sector level (or equivalent) with the purpose of detecting even small fragments of objects that remained in the media after been deleted but not overwritten.

2. Related Work

A significant amount of work has been done with hash functions to reduce the volume of data analyzed when comparing data in a seized device with a database of known interest objects. This database can encompass bad objects, which can be a lead to follow, or good ones, eliminated from the analysis, like operating system data, well-known software objects etc. The comparison is in a hash level, using algorithms like MD5 or SHA-1. In this work, we will focus on the approach using a bad objects database for our experiments.

Hash functions are used to create a representation of an input with arbitrary size into a fixed-length size value. If a single bit of the input is changed, the output will be radically different. They are commonly used in integrity checks, digital signatures, and many other applications. Their security stands on the fact that it is computationally infeasible to find two different inputs with the same hash.

However, hashing data has some drawbacks in the forensics field. They only give binary answers. Malicious users can make small changes to objects to make this technique fail, since their hashes will be far different. To mitigate this problem, researchers adopted another approach to deal with it. This approach, known as piecewise hash, splits the object into fixed-size segments, generate hashes for each one of them and then make the comparison. Besides identifying whole objects, this approach could also be useful to deal with data in disk sectors. However, there are also some difficulties in using this technique. Choosing the appropriate object block size is one of them, due to alignment issues. If objects are modified, their stored fragments in the hard drive may not align with the ones in the database. This way, the hashes will be different, and the method will fail.

To overcome this issue, a new method was proposed to compare objects. This mechanism, called similarity hash or similarity digest, can detect commonality among the binary structure of objects, providing a confidence measure of the similarity among them. The first idea of using such method was proposed by Rabin [Rabin et al. 1981], using random polynomials to create data fingerprinting. More recent research include the ssdeep algorithm [Kornblum 2006], which combines piecewise and rolling hash techniques. Data is broken into pieces using a rolling algorithm, and for each piece, a hash function is used to produce a digest. In the end, all of these digests are concatenated to generate the representation of the object. Other methods with the same purpose can be found in Martínez et al.' work [Martínez et al. 2014].

A more recent method for creating similarity hashes, called sdhash, was presented by Roussev [Roussev 2010]. This method will be discussed in the next section and used in this work due to its interesting characteristics and because of its availability, performance, and documentation. In another work, Roussev presents a comparison between sdhash and ssdeep and shows that the former has a better performance than the latter [Roussev 2011].

Another technique used in the present work is statistical sampling, with the pur-

pose of reducing the amount of data analyzed in an investigation. Instead of examining all objects from the seized media, we extract random samples of it to make a statement about the entire population. We aim to implement the same idea of sampling discussed in Garfinkel's work [Garfinkel et al. 2010], but using similarity hashes instead of block hashes to increase the effectiveness of the search for evidences. We also use a sector level approach in order to detect even small parts of target data. This way, we expect to reduce the amount of time to pursue a triage process and increase the accuracy of the search, where evidences with small changes will not confuse the investigation.

3. SdHash: Similarity Digest Hash

The sdHash tool, developed by Roussev [Roussev 2010], has the purpose of selecting multiple features that are most likely to be unique to an object in order to produce a similarity representation. Fig. 1 illustrates the overall process performed by this tool. First, features (sequence of B bytes) are extracted from the object and have their entropy calculated. Then a filter selects the ones considered the best in representing the target. The next step is the creation of a sequence of Bloom Filters [Bloom 1970] to generate the object representation, as illustrated in Fig 2. In this process, up to f selected features are hashed (using SHA-1 algorithm) and the result is split into five sub-hashes. The 11 least significant bits of each sub-hash are taken to address the bits within the Bloom filter. The implementation described in the Roussev's paper uses B=64 and 256-byte Bloom filters, with up to $f = 128$ features mapped to each filter. New filters are created as needed.

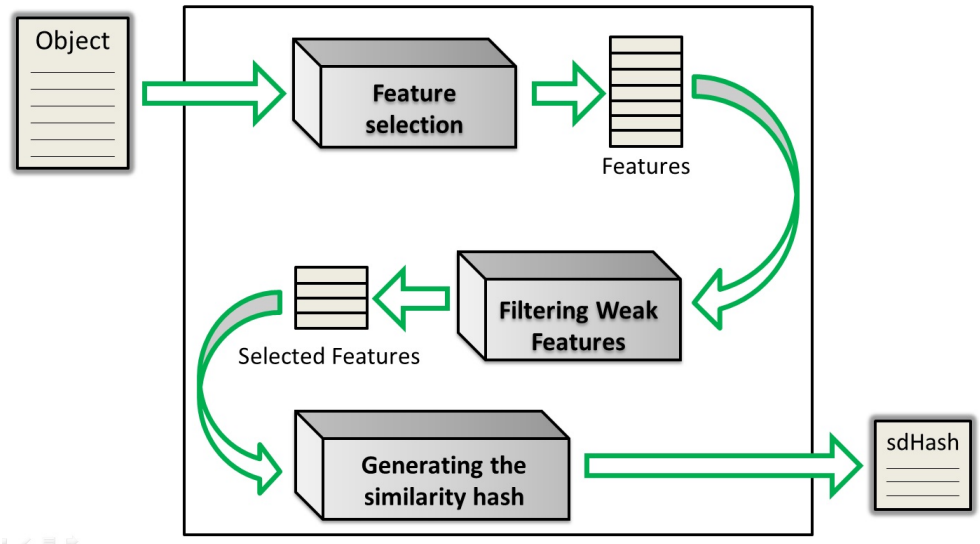


Figure 1. Overall process of generating a similarity hash with sdhash

Comparing two similarity hashes means comparing their Bloom filters. The first filter from the first object is compared to every filter from the second one. The maximum similarity score is selected. The process is repeated for all other features from the first object. Finally, an average of the results is calculated to produce a score which varies from -1 to 100 [Roussev 2010]. Roussev and Quates [Roussev and Quates 2012] point out that a result of 0 (zero) means that the objects are uncorrelated, while -1 (a rare occurrence) means that at least one of the digest have no enough features to produce a reliable comparison. Results ranging from 21-100 are reliable and indicate the existence

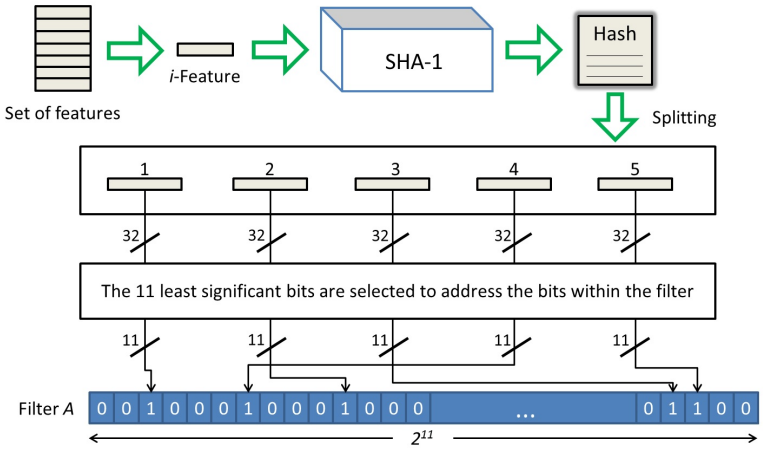


Figure 2. Process of mapping up to f features into each Bloom filter

of significant similarities between the objects. However, for simple object types, such as text, a 5 (five) score could be meaningful and worth further evaluation. The authors also highlight that a value of 100 does not guarantee that the objects are identical.

4. Similarity hashes and Digital Forensics

The sdhash tool has a great potential to be used in forensic investigations for identifying similar objects. Examiners can look in a seized media for objects of interest, being able to detect their targets even though they have been partially modified. Roussev [Roussev 2011] mention another application for this tool, which is finding embedded objects. In this case, we can look for a picture in a seized drive and find not only the picture itself but any occurrence of it even inside other objects.

However, besides the advantages and flexibility achieved with sdhash, its high cost compared to traditional approaches (hash functions) is a major problem. Although we can not estimate the cost of this tool easily, as it is file-dependent, we can calculate the number of features selected by looking at the metadata contained in the digest generated for the object [Roussev and Quates 2012]. For example, for a certain 1 MB object tested, a total of 15.981 features were used. As each feature needs to be hashed before inserted in the Bloom filter, a total of 15.981 hashes need to be performed. In this case, the sdhash will be 15.981 times more expensive than the traditional hash method (SHA-1). As the object size grows, so the number of hashes. Even small objects (1 MB) require thousands of hashes to build their representation. This shows how sdhash is expensive and the need for improving somehow its efficiency.

Another problem is the size of the similarity representation, as described by Roussev [Roussev and Quates 2012]. Hash functions produce a finite and well-known number of output bits, independent of object size. However, sdhash tool outputs a digest about 3% of the object size, according to empirical tests. This tool also requires more time to produce the digests, since it needs to create and consolidate several Bloom filters. Therefore, object size is a major problem in the use of such tool, as it can be a bottleneck in the triage process. For this reason, we propose a new way to deal with these problems by using statistical methods, presented in the next section.

5. Sampling in digital forensics

In order to reduce the time taken for an examiner to classify a seized media as worth or not for further and deeper analysis, we could reduce the amount of data actually processed. This could be done following the ideas presented by Garfinkel et al. [Garfinkel et al. 2010]. The authors show how to adapt the classical "Urn Problem without replacement" in the forensics context. They say that taking enough random samples from a set of objects, there is a good chance that this sample represents the entire dataset. The amount of samples required can be obtained from the following equation, which calculates the probability of missing one of the objects of interest.

$$p = \prod_{i=1}^{n} \frac{((N-(i-1))-M)}{(N-(i-1))} \tag{1}$$

Here N represents the total number of objects in the set, M the number of targets (objects of interest), and n the number of objects required to be sampled. We will use this idea combined with similarity hashes in our work to reduce the time for a triage in investigations. Our goal is to verify whether a media contains at least one object from a database of interest objects. If so, we can take this media for further analysis.

In the calculus, the variable N is the seized media size. As we work at the disk sector level, the value for N will be the number of media sectors. For determining the M value, we can use different approaches. The first one considers the case where we want to find a particular object, which is just using its size (in sectors) as M value. Other scenario consists in figuring out whether a seized drive contains any object of a database or not. In such case, we need to adapt the Urn Problem to this context, since we do not know whether the media contains evidences or not. We have to consider that at least one object of size M (estimated value) will be present in the media in order to comply with one of the basic assumptions of the problem: The presence of an object inside the "urn", or in our case, the media. Then, we calculate the formula using the estimated M value (in sectors) to obtain n for a chosen probability rate. This means that we need to take n random samples from the media and with probability $1 - p$ we will find objects with this size or larger (we can adjust this value as needed, but the n value will change). We highlight that the examiner can control the M value according to the investigation, using the average size of the objects from the database or a common object size for the type of searched objects for that particular scenario. High values for M increase the chances of missing smaller objects but decreases the required triage time. Therefore, prior knowledge about the search can potentially help in determining the M value.

Using the adapted Urn Problem equation and controlling its error rate, examiners can look for objects using random samples with high probabilities of success. For example, considering a 2 TB seized disk (which contains approximately 500 million 4 KB sectors) and that we want to find a 100 MB object (25,000 sectors) on it. Using the Eq. 1 with N = 500,000,000, M = 25,000 and fixing a success rate of 99%, we need to take around 92,500 samples from the hard disk drive in order to find at least one fragment from the desired object.

6. Using sampling to reduce sdhash cost

In order to minimize the time to analyze a large volume of data, we propose an approach to combine the use of similarity hashes, using the sdhash tool, with sampling techniques. We will also work with object fragments (disk sectors) instead of whole objects since we want to encompass scenarios where the file system is corrupted, and no metadata about its structure is available. Besides, taking sectors directly from a hard drive give us the possibility of finding data that were deleted but some of its fragments were not overwritten.

However, using the sampling technique with object fragments can also be costly. The number of similarity hashes generated and of comparisons between the datasets increase as their sizes grow, even though we use samples. This is also compounded by the fact that we are working with fragments instead of whole objects (it increases the number of objects). With this in mind, we propose the use of clusters to decrease this cost.

6.1. Clustering approach

We propose the use of clusters to reduce sdhash cost. We take a determined number of sectors from the seized media being analyzed and gather them in a single object, which will have the similarity hash generated and will be used in the comparison. The fragments can and most of the times will be from different objects and yet the sdhash tool will be able to identify similarity due to its characteristics.

Our objective is to figure out whether a seized media contains objects of interest based on a comparison to a database, in the shortest possible time. To this end, we randomly sample sectors from the disk, gather them into clusters of j sectors and generate their similarity hash representation using sdhash. Then, we compare these digests to a database in order to find common interest objects.

One question that stands is how to take the samples. We propose two different ways to perform this operation: diffuse or contiguous. In the first one (Fig. 3), we select objects (sectors) randomly around the seized media and group them into clusters, which will be sdhashed and compared to a database of interest objects. The second method consists in taking contiguous fragments around some randomly chosen sectors in order to build the clusters (Fig. 4). It is important to highlight that for fragmented disks both methods seems to work fine, while for a defragmented one, the second method will not get fragments from many different files. In this work, we will only use the diffuse mode for sampling. The other method will be covered in future studies.

6.2. Experiments

We conducted experiments to validate our ideas using sdhash tool and sampling techniques. To this end, we simulated specific scenarios to measure the efficiency of this approach in each one, in order to find evidences in a triage process. Our goal is to find at least one object fragment from a database of interest objects present in the seized media being analyzed. If so, this media will be selected for further and deeper analysis.

We first compared our approach of using similarity hashes and sampling to the common one where a similarity representation is generated for every object in seized media. Then, we made experiments with the first approach exploring different scenarios and evaluating the impact of using the clustering method.

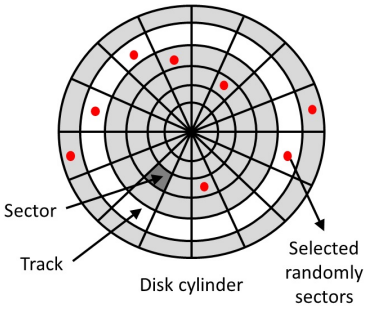


Figure 3. Diffuse method of selecting objects for a cluster.

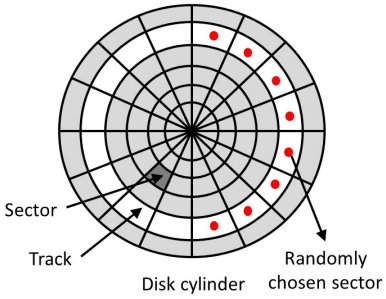


Figure 4. Contiguous method for clustering

In our experiments we selected two datasets and labeled them as DBIO (Database of Interest Objects) and SM (Seized Media). The DBIO is the largest one, with 2204 different files (218458 sectors), while the second have 549 (74929 sectors). The datasets were taken from the $t5$ corpus [Roussev 2011] and encompass different file formats: text, pdf, html, doc, ppt, jpg, xls and gif. For the purpose of our experiments, the DBIO and SM datasets have only four files in common (each one encompass several objects fragments), with the following format and size: F_1: doc (1039 sectors), F_2: ppt (469 sectors), F_3: text (87 sectors) and F_4: pdf (273 sectors). Our goal is to find at least one object in common in the shortest time possible. All files in the SM dataset were fragmented (broken into 4 KB objects) to simulate disk sectors. For the DBIO dataset, we disposed the objects in three ways to simulate different scenarios: fragmented (4 KB), whole file and image file.

We first evaluated the approach of using sdhash without sampling. For this end, we followed the steps below:

1. Creation of a similarity hash representation for every object in the DBIO.
2. Creation of a similarity hash representation for every object of the SM.
3. Comparison of similarity hashes between DBIO and SM objects.
4. Evaluation of the results.

The next experiment evaluated the inclusion of sampling in the process. The first step is to make a sample of the SM dataset. To this end, we used the approach described in section 5. We first estimated the object size we wanted to search and applied the Urn Problem equation using this value and the SM total size (74929 sectors). In our experiments, we chose as file size of 1039 sectors (the size of the greatest common file available in both datasets) and adopted a 99,9% success rate. Using the equation, we got a result of 550, which is the required number of samples we have to take from the SM.

Then, we made several experiments focusing on the described approach with different scenarios in order to evaluate the impact of clustering and determine the best conditions for using it. In general, we did our search following the steps below:

1. Creation of a similarity hash representation for every object in the DBIO.
2. Sampling the SM dataset.
3. Creation of a similarity hash representation for the samples/clusters of SM.
4. Comparison of similarity hashes between DBIO objects and SM samples/clusters.
5. Evaluation of the results.

Our first experiment involved comparing the objects from the SM with those in DBIO dataset in the fragment level, in order to simulate disk sectors (4 KB). We also conducted this experiment using our idea of clustering. We choose the cluster size c a value of 100 sectors since it was a reasonable value got from empirical tests. For larger values, we identified an increasing number of false positives (results pointing out similarity between unrelated files) while smaller ones did not present a significant reduction but only more comparisons to make. In the next experiment, we evaluated the use of whole files representing the database instead of their fragments. Again, we did it using clusters and without them. The last experiment compared the SM fragments to a DBIO image file. To this end, we created a similarity hash using the entire dataset in order to create a single representation of it. For default, the sdhash breaks large files in 128 MB pieces and creates a similarity hash for each one of them [Roussev and Quates 2013].

7. Discussion

The results of our experiments are presented in tables 1 and 2. In the first one, we present a comparison of sdhash with and without sampling, using DBIO and SM in the fragmented level. However, there is a difference between the maximum comparison number of both techniques (second column) to the expected value (16.368.839.482 without sampling and 120.151.900 with it). This is due to sdhash does not work with objects smaller than 512 bytes and the datasets have a few ones which do not have this minimum size.

When we analyze the impact of adding the sampling technique, a significant reduction in the number of comparisons is noticed, requiring about 136 times fewer operations. Also, we present the number of objects of interest found by each technique. Both found more objects than there really are. This exceeding value represents the false positives generated by them, minimized by the use of sampling.

The cost of each technique is presented in the last column of table 1. It is represented by the number of hashes functions (SHA-1) required to generate the representation of each dataset object. For the traditional approach using a hash-based method, the value is equal to the objects number: each one only requires a single hash to create its representation. However, a high cost is observed using sdhash (about 224 times more expensive than the traditional technique), which is minimized by the adoption of sampling (reduced to 167 times). This way, it is evident that using sdhash is very expensive and its combination with statistical methods becomes essential towards a practical approach. To this end, we focus on this combination and propose the use of clusters to reduce even more the costs, evaluating different scenarios in order to find the best ones in which we have the greatest reduction in costs.

Table 1. Experiments comparing sdhash with and without sampling on the fragment level, measuring their efficiency and the cost based on hash functions

Technique	#Max. Comparisons	#Objs of interest found	#Hashes functions (SHA-1)		
			DBIO	SM	Total
sdhash	16.343.700.471	9114 / 1868	48.947.005	16.791.207	65.738.212
sdhash & Sampling	120.031.450	62 / 22	48.947.005	124.979	49.071.984

We present in Table 2 the results of sdhash and sampling techniques for different scenarios, evaluating the use of our clustering approach. With this technique, the number of comparisons between DBIO and SM were reduced by a factor of c (size of the cluster). However, at the same time, in some experiments as cases #3 and #5, this technique is ineffective since no matches were found when comparing the object fragments with the DBIO in whole file and image file formats. The objects encompassing the cluster interfere in the creation of the similarity representation and thwart the comparison (the similarity among them will be slight). Only case #1 when the comparison is made considering the DBIO fragmented, we had significant results. Also, in the scenarios where we did not use cluster and consider the DBIO dataset in whole file and image file formats (#4 and #6), the comparisons decreased, but the number of false positives proportional to the comparisons is elevated in relation to the other scenarios (#1 and #2). Also, given an object fragment, it is harder to detect similarity comparing this small datum to a database image file as the size proportional between them increase. This makes our goal of identifying fragments of deleted data fail.

Table 2. Experiments with sdhash using different object formats

#Exp.	#Max. Comparisons	#Matches 20-100	Clusters	Database format	Objs of interest F_1	Objs of interest F_2	#Features generated
1	1.309.434	107	Yes	Fragmented	11(4)/4	20(15)/18	48.985.337
2	120.031.450	238	No	Fragmented	17(4)/4	45(15)/18	49.071.984
3	12.870	0	Yes	Whole file	0/4	0/18	14.363.578
4	1.179.750	149	No	Whole file	5(4)/4	15/18	14.450.225
5	42	0	Yes	Image file	0/4	0/18	10.587.068
6	3850	288	No	Image file	4/4	17(15)/18	10.673.715

Table 2 also presents the total number of matches between the datasets according to a range of scores got from sdhash, where the results express the similarity between objects in the SM sample to those in the DBIO. We knew beforehand that there were only 22 objects in common between the datasets after the sampling, where 18 belongs to file F_1 and 4 to F_2. The other two objects (F_3 and F_4) had no fragments selected in the sample. We also show the number of matches restricted to the objects selected and the precision in the experiments on finding them, as well as the number of false positives by showing the total number of matches and the number of true positive (TP) matches (parentheses). For example, in experiment #2, object F_2 had 45 matches for only 18 TP possible. In the parentheses, we show the number 15 which is the number of TP matches. We highlight that only scores of similarity ≥ 20 were counted, since they are the ones significant (reliable) [Roussev and Quates 2012]. Object F_1 had 100% of cover in experiments #1, #2, #4 and #6, while F_2 a total of 83.34% for the same cases.

Comparing the expected number of matches to the one got from the total matches (third column), we can see a high number of false positives results (considering results with score ≥ 20). The elevated number will be analyzed in future studies, as well as measures to properly address them, but we believe that it happens due to common structure between objects of the same type. Furthermore, Young et. al [Young et al. 2012] mention that the cause can also be due to common NULL (0x00) blocks.

8. Conclusions

Digital forensic investigation is becoming a critical field as its techniques needs to scale to follow the fast increase in media storage capacity. In this work, we presented a proposal for using similarity hash tools, such as sdhash, combined with sampling techniques to reduce the time of a triage process. We showed how expensive sdhash is and possible ways to reduce the number of similarity hashes generated and of comparisons, using sampling techniques and clustering disk sectors. Besides the reduction of the number of comparisons and features generated, this approach showed effective in finding similar objects of interest and presented a smaller number of false positives. Also, our technique gives examiners more flexibility and allow them to find even objects that were deleted but some of its fragments remain on disk. We also showed limitations of clustering by evaluating different scenarios, which proved to be ineffective when comparing clusters to database image files and to whole files. On the other hand, we had good results comparing them to a fragmented database.

Although our experiments have shown that sdhash could identify objects of interest with a small set of data taken randomly from a seized media, there is a lack of understanding about the false positives generated and how to treat them, a subject that will be covered in a future work.

References

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426.

Garfinkel, S., Nelson, A., White, D., and Roussev, V. (2010). Using purpose-built functions and block hashes to enable small block and sub-file forensics. *Digital investigation*, 7:S13–S23.

Kornblum, J. (2006). Identifying almost identical files using context triggered piecewise hashing. *Digital investigation*, 3:91–97.

Martínez, V. G., Álvarez, F. H., and Encinas, L. H. (2014). State of the art in similarity preserving hashing functions. In *Proceedings of the International Conference on Security and Management (SAM)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Rabin, M. O. et al. (1981). *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Lab., Univ.

Roussev, V. (2010). Data fingerprinting with similarity digests. In *IFIP International Conf. on Digital Forensics*, pages 207–226. Springer.

Roussev, V. (2011). An evaluation of forensic similarity hashes. *Digital investigation*, 8:34–41.

Roussev, V. and Quates, C. (2012). Content triage with similarity digests: The m57 case study. *Digital Investigation*, 9:S60–S68.

Roussev, V. and Quates, C. (2013). sdhash tutorial: Release 0.8. `http://roussev.net/sdhash/tutorial/sdhash-tutorial.pdf`. Accessed 2016 Set 13.

Young, J., Foster, K., Garfinkel, S., and Fairbanks, K. (2012). Distinct sector hashes for target file detection. *Computer*, 45(12):28–35.

Análise de Malware .NET para Identificação de Comportamentos Similares

Cicero Silva Luiz Junior[1], **Paulo Lício de Geus**[1], **André Grégio**[2]

[1] Instituto de Computação (IC)
Universidade Estadual de Campinas (Unicamp)
Campinas – SP – Brasil

[2]Departamento de Informática (DInf)
Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

Abstract. *Currently, the .NET Framework is one of the most popular platforms for software development. It was introduced around the year 2000, with the idea behind it that it should be safe and increase developers' productivity. However, it also attracted some malware developers, since the .NET framework could not only be used to improve their productivity but also to write multi-platform malware. More importantly, .NET is adopted widely across the industry, from large scale servers to embedded devices and micro-controllers. At the same time, however, the .NET framework enforces a strong richness in its metadata, which helps to statically analyze a compiled binary in a way that makes it possible to infer most of its behaviors.*

Resumo. *Hoje em dia, o .NET é um dos ambientes mais populares para desenvolvimento de Software. Ele foi introduzido por volta dos anos 2000, com o intuito de ser uma plataforma segura e que aumentasse a produtividade dos desenvolvedores. Contudo, esse aumento de produtividade também chamou a atenção de desenvolvedores de malware, que viram nele a possibilidade de se escrever programas maliciosos de forma rápida e que pudessem alcançar uma vasta gama de dispositivos, já que o .NET é executado em ambientes desde servidores de larga escala até microcontroladores ARM ou MIPS. Contudo, a estrutura do framework do .NET é muito rica em meta-informação, o que facilita a análise de binários para inferir seus possíveis comportamentos.*

1. Introdução

Estima-se atualmente que o sistema operacional Windows (família NT) é responsável pelo funcionamento de mais de 90% de todos os computadores pessoais existentes [for Internet Technologies 2016] . Nos anos 2000, com o intuito de melhorar a segurança, performance e confiabilidade do ecossistema MS-Windows, houve um esforço da parte da Microsoft em construir um ambiente de execução gerenciado e seguro. Esse ambiente deveria se tornar a principal plataforma de desenvolvimento de software para este ecossistema e é conhecido como .NET Framework, cuja primeira versão foi disponibilizada em 2002 e, desde então, sua importância dentro do ambiente MS-Windows só aumentou.

Embora o .NET tenha sido construído com o intuito de facilitar a construção de aplicativos legítimos, a sua capacidade de interagir de forma muito integrada com o sistema operacional também o tornou atraente para a construção de programas maliciosos

(*malware*). Além disso, o fato de ele estar disponível em praticamente qualquer instalação MS-Windows, desde as Desktop até as versões integradas e IoT o torna um bom meio de difusão de *malware*, pois um código escrito para uma plataforma pode ser executado em qualquer outra que também implemente o .NET [LLC 2015] .

Sabe-se, contudo, que programas .NET são compilados para uma linguagem intermediária (*CIL - Common Intermediate Language*) e que o *framework* exige que uma série de regras rígidas sejam cumpridas para que os binários executem. Dessa forma, é possível explorar um binário potencialmente malicioso compilado em *CIL* e obter estaticamente uma representação de alto-nível do seu código fonte. Tal representação pode ser usada para comparação entre diferentes programas de forma a se buscar similaridades, isto é, funções reutilizadas ou pequenas mudanças que indiquem que um *malware* é uma variante de outro, pertencendo assim à mesma família.

A contribuição deste artigo é introduzir um analisador estático automatizado de *malware* com enfoque em amostras compiladas para CIL, utilizar o algoritmo MOSS para detectar a similaridade entre trechos de um dado programa e comparar esses trechos a uma base de dados gerada pelos autores, a qual possui trechos de códigos maliciosos que representam funções que implementam comportamentos suspeitos conhecidos. Desta forma é possível detectar, em um compilado CIL arbitrário, se ele possui trechos com comportamento potencialmente malicioso. O texto está dividido da seguinte maneira: introduz-se na Seção 2 detalhes sobre o .NET Framework, técnicas de análise de *malware* e o algoritmo MOSS; na Seção 3, discute-se o sistema proposto (arquitetura e implementação); na Seção 4 mostram-se testes realizados e resultados obtidos e, por fim, na Seção 5 apresenta-se a conclusão do artigo.

2. Aspectos Técnicos e Trabalhos Relacionados

Nesta seção são apresentados conceitos básicos de como funciona o .NET Framework, alguns detalhes importantes para a execução da proposta deste trabalho, tais como a descrição resumida do que é uma análise estática, uma breve introdução ao algoritmo MOSS, bem como alguns trabalhos relacionados.

2.1. .NET Framework

O Microsoft .NET é um *framework* arquitetado e desenvolvido pela Microsoft[Microsoft 2016]. Seus principais componentes são a *FCL (Framework Class Library)* e o *CLR (Common Language Runtime)*. De maneira sucinta, pode-se dizer que a *FCL* é um conjunto de bibliotecas com funções prontas para uso por parte dos programas enquanto o *CLR* é uma combinação de compilador e sistema de suporte à execução[Wikipedia 2016b].

CLR - Common Language Runtime. O *CLR* é o alicerce de todo o ambiente .NET. Trata-se de um agente que gerencia código em tempo de execução e provê serviços essenciais, tais como gerenciamento de memória e *threads*. Adicionalmente, o *CLR* também é responsável por garantir a segurança de tipos de dados e várias outras tarefas ligadas à salvaguarda e robustez dos programas gerenciados por ele[Microsoft 2010]. Especialmente com foco em robustez, por exemplo, o *CLR* possui um módulo dedicado chamado *CTS (Common Type System)*, cujo objetivo é obrigar a todo e qualquer código compatível com o *CLR* ser auto-descritivo e se basearem em um conjunto pré-definido de tipos de dados

básicos. Devido a isso, códigos gerenciados podem consumir outros códigos gerenciados de forma completamente confiável, independente da linguagem de programação original em que eles foram escritos[Microsoft 2010, Microsoft 2015a, Microsoft 2015b]. Outras informações também contidas nessa "auto-descrição" incluem detalhes de todas as versões de bibliotecas que são usadas pelo programa em questão.

CIL / MSIL (Common Intermediate Language / Microsoft Intermediate Language). *MSIL* (ou *CIL*) é a linguagem intermediária produzida por qualquer compilador cujo alvo seja produzir programas que devem executar no *CLR*. Trata-se de uma linguagem orientada a objetos e que se assemelha com uma linguagem *bytecode*[Wikipedia 2016a].

2.2. Técnicas de Análise de Malware

Há vários métodos para se tentar determinar se a ação de um componente de *software* é benigna ou maligna. Os principais métodos, entretanto, podem ser divididos em dois grandes grupos:

- **Análise Estática:** a ideia é estudar o comportamento de um programa sem de fato precisar executá-lo. Para tanto, normalmente se utiliza ferramentas tais como descompiladores, analisadores de códigos-fonte, pesquisadores de *strings* e afins[Mehta 2015]. As principais vantagens desse método de análise são que ele é bastante rápido e pode possivelmente levar a descobertas sobre ramos de execução que nem sempre são exercitados em um ataque real. Por outro lado, sabe-se que na prática é impossível prever todos os caminhos de execução de um programa apenas com base na análise estática clássica, exceto para exemplares suficientemente pequenos e simples[Moser et al. 2007].
- **Análise Dinâmica:** trata de monitorar um certo componente de *software* enquanto ele está sendo executado. Para que a análise seja efetiva, o *software* (ou o ambiente de análise) precisa ser "instrumentalizado", ou seja, ligeiramente modificado para permitir ao analista obter informações bastante detalhadas sobre o programa analisado. Alguns dados que costumam ser de interesse são as saídas produzidas pelo programa, eventuais modificações que ele cause tanto no sistema de arquivos como no repositório central de configurações, tentativas de comunicação com servidores externos de forma periódica, alterações que ele provoca em outros programas executando ao mesmo tempo que ele, parâmetros enviados para *syscalls*, etc[Dan Farmer 2005, EGELE et al. 2012].

Análises feitas sobre diversos tipos de programas maliciosos mostraram que vários deles, na verdade, compartilham código. Com o intuito de facilitar o trabalho dentro das comunidades de *hackers*, alguns destes construíram "ferramentas de administração", que nada mais são do que programas maiores que contêm uma grande quantidade de ataques disponíveis e uma interface de uso intuitivo, permitindo a criminosos não tão experientes criarem seu próprio *malware* personalizado[Pontiroli 2014, McDaniel and Schultz 2014].

2.3. MOSS - *Measure of Software Similarity*

O *MOSS* é um sistema cujo principal objetivo é detectar plágio de software [Aiken 1994]. Dados dois ou mais programas como entrada ele é capaz de retornar a porcentagem que esses dois ou mais programas compartilham de códigos entre eles. Adicionalmente, o algoritmo por trás da implementação do *MOSS* não pode ser trivialmente enganado, de

tal forma que renomear (ou até mesmo suprimir) nomes de variáveis, alterar a ordem de um código (mantendo sua semântica) ou outras transformações de natureza semelhante não produzem alterações significativas da taxa de detecção [Saul Schleimer 2003] . Desta forma, códigos que estiverem ofuscados, mas que conservem as suas principais estruturas lógicas continuam sendo detectáveis como semelhantes pelo algoritmo.

2.4. Trabalhos Relacionados

[Filho et al. 2010] propõem um sistema de análise dinâmica de *malware* baseado em *SSDT Hooking*. Dentro do contexto de sistemas operacionais Windows, *SSDT* significa *System Service Dispatch Table*, ou seja, trata-se da tabela de *Syscalls* do sistema operacional. Através da técnica de *Hooking* de *Syscalls*, o sistema BehEMOT permite obter, em tempo de execução, várias informações sobre um certo processo ou conjunto de processos em execução em uma máquina instrumentalizada. Dessa forma é possível gerar um conjunto detalhado de *logs* que permite a um analista determinar se um dado programa é maligno ou benigno. Contudo, a partir da versão 6.0 do Windows NT em 64 bits, não é mais possível fazer alterações em estruturas críticas do *Kernel*. Portanto, a abordagem tomada pelo BehEMOT só pode ser utilizada no NT 5.2 ou inferior ou, para versões acima da 6.0, apenas para as versões de 32 bits.

[Botacin et al. 2014] propõem uma alternativa que elimina a limitação do artigo anterior, permitindo a avaliação de forma dinâmica de um exemplar de *malware* de 64 bits executando no Windows 8 (NT 6.2). Para que esse objetivo fosse alcançado, construiu-se um *filesystem filter* e um monitor de *callback*, ambos em modo *kernel*, que permitem a uma ferramenta externa capturar atividades executadas no sistema em questão, como por exemplo escritas ou leituras de chaves de registro, alterações no sistema de arquivos, tráfego de rede etc. Notar que a abordagem do trabalho é ligeiramente diferente das abordagens tradicionais que eram aplicáveis a versões anteriores ao NT6, já que o NT6 de 64 bits introduziu novas funcionalidades que impedem o funcionamento de tais abordagens.

Por outro lado, [Christodorescu and Jha 2003] focam na construção de um sistema genérico capaz de detectar padrões de ações maliciosas em arquivos executáveis. De forma resumida, para que isso seja alcançável, primeiro constrói-se uma representação do que define um "comportamento malicioso". Em seguida, constrói-se uma representação do executável que será analisado. De posse de ambas representações, o sistema gera um autômato que contém uma representação genérica das dependências entre as variáveis do programa analisado. A partir disso, para cada procedimento do executável que será analisado é gerado um *CFG - Control-Flow-Graph*. Dando prosseguimento, o sistema de detecção recebe como entrada a representação do comportamento malicioso e um *CFG* e responde se o comportamento malicioso pôde ser detectado no *CFG*.

Por fim, [Grégio et al. 2012] apresentam uma abordagem baseada na detecção de instruções de escrita em memória para identificar possíveis trechos de reuso de código. Para tanto, instrumentaliza-se um sistema com um *debugger* que o monitora por instruções baixo-nível específicas que alteram os valores de registradores ou de posições de memória. Em seguida, agrupa-se cada uma dessas instruções duas-a-duas em uma janela deslizante produzindo uma sequência de bigramas. Por fim, utiliza-se um algoritmo de clusterização sobre os bigramas produzidos, com o intuito de agrupar exemplares de *malware* que tenham gerado sequências parecidas de operações de escrita. Com base

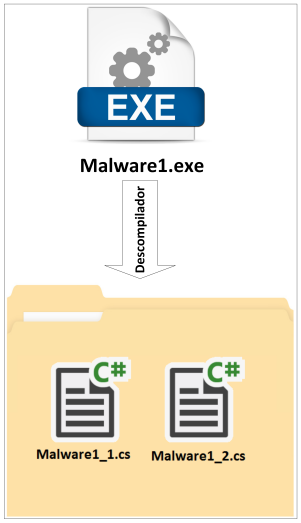


Figura 1. (à esq.) Visão geral da etapa de descompilação automática de uma amostra de *malware*

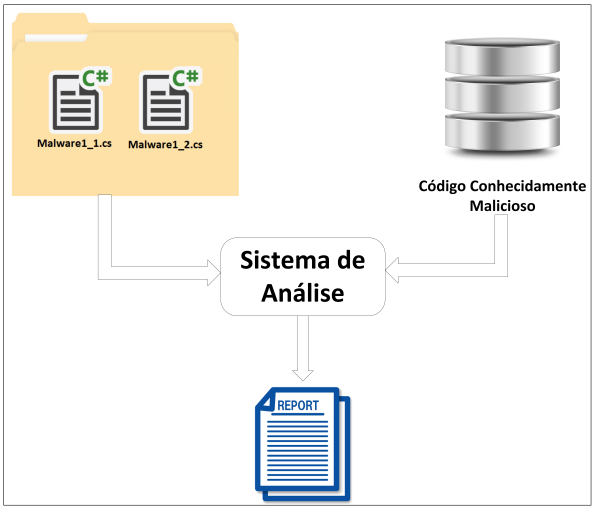


Figura 2. (à dir.) Esquema simplificado do sistema de análise heurística

nisso, o trabalho é capaz de afirmar quão similar um exemplar de *malware* é quando comparado a outro.

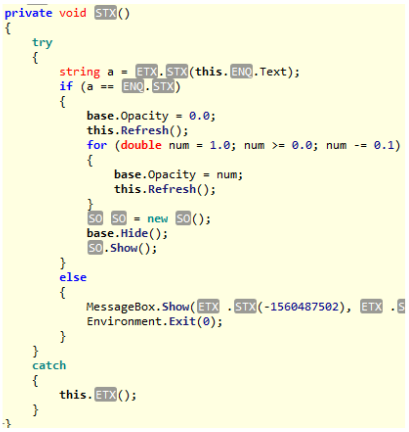
3. Arquitetura e Implementação

O sistema proposto é composto de duas partes: a primeira é responsável pela engenharia reversa, na qual códigos de alto nível são obtidos a partir dos binários .NET. A segunda faz uma análise sobre os códigos-fonte obtidos e tenta localizar trechos que contêm ações potencialmente maliciosas. As Figuras 1 e 2 ilustram, respectivamente, as entradas e as saídas esperadas para cada um dos módulos do sistema.

Módulo 1: Módulo de Engenharia Reversa. O Módulo de Engenharia Reversa é implementado por um programa autônomo, com o auxílio de bibliotecas capazes de analisar códigos *CIL* e transformá-los em linguagem de alto-nível. Neste caso, apenas por familiaridade, escolheu-se transformar código CIL em linguagem CSharp. [icsharpcode 2016, Jetbrains 2016, Gate 2016]. Exemplos dos tipos de código de alto-nível que podem ser extraídos utilizando-se bibliotecas como as mencionadas anteriormente podem ser observados nas Figuras 3 e 4. Na primeira delas trata-se de um código fracamente ofuscado, enquanto na outra trata-se de um código fortemente ofuscado.

Módulo 2: Módulo de Análise Heurística baseada no MOSS. O Módulo de Análise é composto por uma base de dados com trechos de código conhecidamente maliciosos e um sistema capaz de se utilizar destas informações para gerar comparações com os códigos descompilados obtidos na fase 1. Todos os códigos submetidos ao sistema são considerados, procurando-se por algum conjunto que se assemelhe ao que está armazenado no banco de dados. No final é gerado um relatório que indica, nos códigos descompilados, trechos de programa que têm alta probabilidade de serem maliciosos. No caso concreto é feita uma varredura por toda a base de dados e, utilizando o algoritmo MOSS, descobre-se quais trechos de código conhecidamente maliciosos aparecem, total ou parcialmente, dentro de cada um dos exemplares em análise.

```
private void InitializeComponent()
{
    this.components = new Container();
    this.axLUhrLLnI = new mWYMsmhuVo();
    this.bindingSource1 = new BindingSource(this.components);
    ((ISupportInitialize)this.bindingSource1).BeginInit();
    base.SuspendLayout();
    base.Controls.Add(this.axLUhrLLnI);
    base.AutoScaleDimensions = new SizeF(6f, 13f);
    base.AutoScaleMode = AutoScaleMode.Font;
    base.ClientSize = new Size(363, 248);
    base.Name = "Form1";
    this.Text = "Form1";
    base.Load += new EventHandler(this.Form1_Load);
    ((ISupportInitialize)this.bindingSource1).EndInit();
    base.ResumeLayout(false);
}
```



```
private void STX()
{
    try
    {
        string a = ETX.STX(this.ENQ.Text);
        if (a == ENQ.STX)
        {
            base.Opacity = 0.0;
            this.Refresh();
            for (double num = 1.0; num >= 0.0; num -= 0.1)
            {
                base.Opacity = num;
                this.Refresh();
            }
            SO SO = new SO();
            base.Hide();
            SO.Show();
        }
        else
        {
            MessageBox.Show(ETX .STX(-1560487502), ETX .S
            Environment.Exit(0);
        }
    }
    catch
    {
        this.ETX();
    }
}
```

Figura 3. (à esq.) Código-fonte levemente ofuscado extraído a partir de um compilado em *MSIL/CIL*: a maior parte das variáveis ainda possui seu nome original no código *CIL* e pode, portanto, ser reconvertida em código-fonte[MalwareTips 2014].

Figura 4. (à dir.) Código-fonte fortemente ofuscado extraído a partir de um compilado em *MSIL/CIL*: embora os nomes das variáveis e classes não estejam disponíveis, mesmo assim é possível obter código suficientemente de alto-nível de tal forma que seja possível inferir sua funcionalidade[Altman 2012].

3.1. Implementação

Todo o software foi implementado em linguagem C# no Microsoft Visual Studio. Por conveniência, decidiu-se utilizar o Microsoft SQL Azure como banco de dados para o armazenamento das informações, uma vez que isso contribui para a velocidade da análise. Adicionalmente, todos os processos de análise e teste são executados a partir de uma máquina virtual dentro do Microsoft Azure. Para a estrutura do sistema decidiu-se pela utilização do paradigma DDD (Desenvolvimento Dirigido a Domínio). Inicialmente optou-se por construir a Interface de Usuário utilizando-se o WPF (Windows Presentation Foundation), com o intuito de se economizar tempo de implementação.

3.1.1. Interface de Usuário

Tela Inicial. A tela inicial permite ao usuário escolher entre as principais funções do Programa. Em *Manage Code Snippets* ele pode gerenciar os comportamentos conhecidos pelo programa; em *Analyze New Malware* é possível analisar um único Malware e obter um relatório de comparação com a base de conhecimento; A função *Browse Analyzed* permite visitar análises antigas que foram guardadas na base de conhecimento e, por fim, a opção *Directory / Batch Analysis* permite selecionar um diretório inteiro contendo Malware e processar a análise de todos eles, gerando um relatório individual automatizado para cada um.

Tela de Gerenciamento de *Snippets*. Nesta tela é possível gerenciar os comportamentos maliciosos conhecidos pelo programa.

Tela de Análise de novo *Malware*. Nesta tela é possível submeter um único exemplar de Malware para Análise. Assim que a análise é concluída é exibido um relatório, conforme

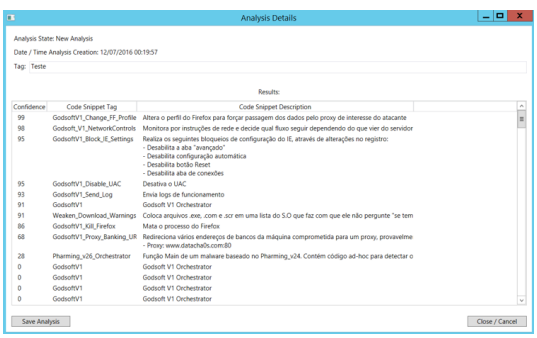


Figura 5. (à esq.) Exemplo de tela de relatório obtida após uma análise. *Confidence* representa a probabilidade de um certo comportamento estar presente no exemplar fornecido como entrada para o sistema.

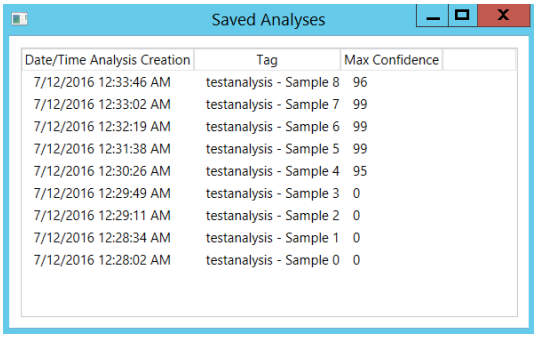


Figura 6. (à dir.) Relatório gerado a partir de uma análise em batch, demonstrando os *hits* que aconteceram para cada um dos exemplares. *Max Confidence* representa o maior valor de uma detecção dentro do exemplar sendo analisado. Caso a maior detecção seja baixa, quer dizer que o exemplar em análise não é semelhante a nada presente na base de dados. Caso seja alta, algum dos *snippets* da base de conhecimento se assemelha a alguma porção de código do programa em análise.

apresentando na Figura 5.

Análise de Malware em *Batch*. Neste modo é possível fornecer um diretório com vários executáveis. O programa faz a análise e guarda internamente o relatório para cada um dos exemplares. Para facilitar a busca, todos os exemplares analisados a partir de uma requisição via *batch* compartilham a mesma *tag*, e a isso é adicionado um contador, para indicar qual dos arquivos é de que se trata.

3.1.2. Engine

Para implementar a abordagem DDD, o *Engine* do sistema é dividido em quatro grandes módulos: Domínio (responsável por definir a lógica principal e os contratos que a implementação deve seguir), Repositório (responsável por interagir com o armazenamento de dados), Serviço (responsável por coordenar a interação entre o Repositório, os Modelos do Domínio e outros serviços) e o Util, que possui adaptadores que permitem ao programa acessar ferramentas externas (tais como o descompilador ILSpy [icsharpcode 2016] , que foi o utilizado para transformar *CIL* em C# para posterior análise e uma interface *REST* que comunica com o serviço online do MOSS disponibilizado pela Universidade de *Stanford*.)

3.1.3. Modelos do Domínio

Os Modelos do Domínio são responsáveis por representar os dados que são armazenados no sistema, além de conter métodos que operam sobre esses dados. Os modelos implementados para o programa em questão são:

BaseModel. Todos os modelos herdam deste modelo base. Seu único campo é um Id, composto de 128 bits que tem como objetivo identificar de forma única um objeto no sistema

Analysis. Representa uma análise completa. Suas informações são um *TimeStamp*, uma estrutura *FileDisassebly* (que representa a descompilação de um executável em *CIL* para C#), uma lista de estruturas do tipo *CodeCompareResult* (em que cada estrutura representa a comparação entre um arquivo de código fonte do Malware sendo analisado e um dos *snippets* de conhecimento do sistema), uma *Tag* (que é um campo texto, apenas para facilitar a busca dos dados na base) e um *MaxConfidence*, que não é armazenado na base, mas sim recalculado sempre se obtendo o maior valor de *Confidence* dentro da lista de *CodeCompareResult*.

CodeSnippet. Representa o conhecimento de um pedaço de código malicioso. Seus campos são um *CodeSnippetSource* (texto), que possui o código em C# que implementa o comportamento em questão, o campo *CodeSnippetDescription*, que é uma descrição em linguagem natural do que faz o comportamento em questão e uma *Tag*, utilizada para facilitar buscas e agrupamentos na base de dados.

DisassemblyResult. Representa a descompilação de um *malware* por um descompilador. Seus campos são um campo dizendo qual o descompilador que foi utilizado, uma lista de estruturas do tipo *SourceFile* (que representa um arquivo de código fonte) e um apontador para uma estrutura do tipo *FileDisassembly*, que é a entidade pai que guarda objetos do tipo *DisassemblyResult*

FileDisassembly. Representa a descompilação de um executável de *CIL* para C#. Como o programa permite que se pluguem vários descompiladores a ele, um *FileDisassembly* na verdade possui uma lista de *DisassemblyResult*, onde cada um deles está associado a um descompilador. Isso foi introduzido com o intuito de se comparar os resultados obtidos, caso mais de um descompilador seja utilizado. Além do campo mencionado também são guardados um *TimeStamp* da criação da descompilação, extensão do arquivo, comprimento do arquivo original em *bytes*, nome do arquivo original e um *snapshot* completo do arquivo, para futura reanálise.

SourceFile. Representa um arquivo de código fonte que foi descompilado a partir do binário em *CIL* fornecido. Seus campos são o *FileContent* (código fonte em C# propriamente dito), *FileName* (que na maioria das vezes também é recuperado) e o *FileLengthInBytes*, que representa o comprimento do arquivo extraído.

4. Testes e Resultados

Para validar o funcionamento do sistema, separou-se um conjunto de exemplares de *Malware* e seus códigos-fonte foram analisados utilizando a ferramenta ILSpy [icsharpcode 2016]. A partir destas análises foi possível determinar que vários destes programas maliciosos eram construídos a partir de *toolkits*, e isso ficava bastante evidente nos fontes descompilados. A partir disso, algumas das funcionalidades encontradas foram documentadas e introduzidas no programa como porções de código da base de conhecimento. Em seguida fez-se uma análise em *batch* sobre um segundo conjunto composto por 9 exemplares, dos quais 5 eram sabidamente construídos com a ferramenta GodSoft. Como o esperado, o sistema foi capaz de identificar os 5 exemplares que continham algum tipo de código parecido com o que havia sido introduzido em sua base de conhecimento e, ao mesmo tempo, não gerou falsos-positivos para exemplares que não devem ser detectados. Na figura 6 é possível verificar o relatório gerado para esta análise.

Adicionalmente também foi feita uma análise individual sobre um décimo exem-

plar que também havia sido construído com a ferramenta GodSoft. O relatório detalhado para esta detecção pode ser observado na Figura 5 e, como o esperado, todos os componentes do GodSoft foram identificados, ao mesmo tempo em que componentes alheios tiveram uma baixa taxa de detecção.

5. Conclusão

Como se pode observar, analisar exemplares de *malware* utilizando-se um descompilador e um analisador baseado no algoritmo do *MOSS* é viável. Adicionalmente, devido às propriedades deste algoritmo em questão, ofuscações mais simples (baseadas na alteração ou supressão do nome de variáveis) não alteram a detecção, já que o próprio MOSS é, por construção, imune a estes tipos de alterações.

Referências

Aiken, A. (1994). *A System for Detecting Software Plagiarism*. `http://theory.stanford.edu/~aiken/moss/`. Acessado em Setembro/2016.

Altman, T. (2012). Reverse-engineer an obfuscated .net application. `http://travisaltman.com/reverse-engineer-an-obfuscated-net-application/`. Acessado em Setembro/2016.

Botacin, M., Afonso, V., de Geus, P. L., and Grégio, A. (2014). Monitoração de comportamento de malware em sistemas operacionais windows nt 6.x de 64 bits. *Anais do SBSEG 2014*.

Christodorescu, M. and Jha, S. (2003). Static analysis of executables to detect malicious patterns. *In Proceedings of the 12th USENIX Security Symposium*.

Dan Farmer, W. V. (2005). Malware analysis basics. `http://www.porcupine.org/forensics/forensic-discovery/chapter6.html`. Acessado em Setembro/2016.

EGELE, M., SCHOLTE, T., KIRDA, E., and KRUEGEL, C. (2012). A survey on automated dynamic malware analysis techniques and tools. *ACM Computing Surveys*.

Filho, D. S. F., Grégio, A. R. A., Afonso, V. M., Santos, R. D. C., Jino, M., and de Geus, P. L. (2010). Análise comportamental de código malicioso através da monitoração de chamadas de sistema e tráfego de rede. *Anais do SBSEG 2010*.

for Internet Technologies, M. S. S. (2016). *Operating System Desktop Market Share*. `http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0`. Acessado em Setembro/2016.

Gate, R. (2016). .net reflector. `http://www.red-gate.com/products/dotnet-development/reflector/`. Acessado em Setembro/2016.

Grégio, A. R. A., de Geus, P. L., Kruegel, C., and Vigna, G. (2012). Tracking memory writes for malware classification and code reuse identification. *DIMVA'12 Proceedings of the 9th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*.

icsharpcode (2016). Ilspy - .net decompiler. http://ilspy.net/. Acessado em Setembro/2016.

Jetbrains (2016). Dotpeek. https://www.jetbrains.com/decompiler/. Acessado em Setembro/2016.

LLC, S. L. (2015). *Netduino*. http://www.netduino.com/. Acessado em Setembro/2016.

MalwareTips (2014). Malware analysis 3 - reverse engineering .net malware. http://malwaretips.com/threads/malware-analysis-3-reverse-engineering-net-malware.42338/. Acessado em Agosto/2015.

McDaniel, D. and Schultz, J. (2014). Reversing multilayer .net malware. http://blogs.cisco.com/security/talos/reversing-multilayer-net-malware. Acessado em Setembro/2016.

Mehta, L. (2015). Malware static analysis. http://resources.infosecinstitute.com/malware-analysis-basics-static-analysis/. Acessado em Setembro/2016.

Microsoft (2010). .net framework conceptual overview. https://msdn.microsoft.com/en-us/library/zw4w595w(v=vs.100).aspx. Acessado em Setembro/2016.

Microsoft (2015a). Common language runtime (clr). https://msdn.microsoft.com/en-us/library/8bs2ecf4(v=vs.110).aspx. Acessado em Setembro/2016.

Microsoft (2015b). Common type system. https://msdn.microsoft.com/en-us/library/zcx1eb1e(v=vs.110).aspx. Acessado em Setembro/2016.

Microsoft (2016). .net framework. https://www.microsoft.com/net/. Acessado em Setembro/2016.

Moser, A., Kruegel, C., and Kirda, E. (2007). Limits of static analysis for malware detection. *23rd Annual Computer Security Applications Conference*.

Pontiroli, S. (2014). Garfield garfield true, or the story behind syrian malware, .net trojans and social engineering. https://securelist.com/blog/virus-watch/59360/garfield-garfield-true-or-the-story-behind-syrian-malware-net-trojans Acessado em Setembro/2016.

Saul Schleimer, Daniel Wilkerson, A. A. (2003). *Winnowing: Local Algorithms for Document Fingerprinting*. http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf. Acessado em Setembro/2016.

Wikipedia (2016a). Common intermediate language. https://en.wikipedia.org/wiki/Common_Intermediate_Language. Acessado em Setembro/2016.

Wikipedia (2016b). .net framework (wikipedia). https://en.wikipedia.org/wiki/.NET_Framework/. Acessado em Setembro/2016.

Extração e análise de memória volátil em ambientes Android: uma abordagem voltada à reconstrução de trajetórias com base no protocolo NMEA 0183

João Paulo Claudino de Sousa[1], João José Costa Gondim[2]

[1]Seção de Perícias de Informática – Polícia Civil do Distrito Federal (PCDF)
Brasília, DF – Brasil

[2]Departamento de Ciência da Computação – Universidade de Brasília (UnB)
Brasília, DF – Brasil.

jpclaudino@gmail.com, joao.gondim@gmail.com

Abstract. *Android devices are widely used and can function as GPS receivers. Time and position information are very relevant in the investigative field, however, data stored in non-volatile media may be limited with regard to the reconstruction of trajectories. This paper proposes a method for recovering messages with GPS coordinates stored in RAM memory of Android mobile devices. Finally, it was analyzed the feasibility of reconstruction of trajectories based on the NMEA 0183 protocol messages retrieved from RAM memory.*

Resumo. *Dispositivos Android são amplamente utilizados e podem funcionar como receptores GPS. Informações de tempo e posicionamento possuem grande relevância no campo investigativo, todavia, os dados armazenados em mídia não-volátil podem ser limitados no que diz respeito à reconstrução de trajetórias. Este trabalho propõe um método de recuperação de mensagens com coordenadas GPS armazenadas na memória RAM de dispositivos móveis Android. Por fim, foi analisada a viabilidade da reconstrução de trajetórias de dispositivos móveis com base nas mensagens do protocolo NMEA 0183 recuperadas da memória RAM.*

1. Introdução

Sistemas Android possuem grande relevância para comunidade forense, sendo que nos últimos anos, diversos trabalhos sobre a aquisição e análise de dados provenientes tanto de memórias voláteis como não voláteis foram produzidos.

No campo investigativo, informações de posicionamento são importantes pois podem fornecer elementos para elucidação da autoria e do *modus operandi* de algum fato delituoso. Além disso, dados de tempo e velocidade também são providos por receptores GPS, podendo ter relevância na elucidação de crimes e acidentes de trânsito.

Este artigo propõe um método de recuperação de mensagens com coordenadas GPS armazenadas na memória RAM de dispositivos móveis Android, a fim de reconstruir o trajeto trilhado pelo dispositivo.

Nesta linha, as especificações da *National Marine Eletronics Association* (NMEA), em especial a NMEA 0183, têm importância fundamental na comunicação

dos receptores GPS com os diversos tipos de aparelhos, uma vez que provêm uma forma padronizada de transmissão dos dados de posicionamento.

O restante do artigo é organizado da seguinte forma: na seção 2 são apresentados trabalhos correlatos, tanto na área de aquisição de memória RAM, quanto na área de forense em dispositivos GPS; na seção 3 é mostrado o funcionamento da *engine* de localização (*GL engine*), sendo explicado como são geradas mensagens do protocolo NMEA e como estas mensagens podem ser recuperadas da memória RAM; na seção 4 são apresentados os testes realizados em diferentes ambientes, com múltiplos aparelhos, bem como os resultados obtidos a partir da análise de imagens (*dumps*) da memória volátil dos dispositivos. Por fim, trabalhos futuros e a conclusão são apresentados na seção 5.

2. Trabalhos correlatos

A recuperação de memória volátil em dispositivos Android vem sendo amplamente explorada nos últimos anos, dada a importância para o mundo forense das informações contidas neste tipo de memória. Em [Sylve, Case, Marziale e Richard 2012], foi produzida uma técnica que utiliza um módulo do *kernel* para extração de memória (conhecido como dmd ou LiME - *Linux Memory Extractor*). O módulo realizava a extração da memória percorrendo a estrutura *iomem_resource* do *kernel* e realizando a tradução de endereços. Entretanto, esta abordagem tem como empecilho a necessidade de compilação do módulo para cada versão específica de *kernel*.

Também existem outros trabalhos visando a aquisição de memória RAM, como em [Thing, Ng e Chang 2010], [Muller e Spreitzenbarth 2013] e [Stüttgen e Cohen 2014], porém, nesta pesquisa, optou-se por utilizar o módulo LiME para extração da memória volátil, visto que seu funcionamento, do ponto de vista forense, não causa grandes impactos na evidência. A documentação[1] do LiME detalha todo o processo de compilação do módulo *kernel*, levando em consideração a arquitetura ARM.

No que diz respeito à recuperação de informações de posicionamento em sistemas de navegação por satélite, há estudos principalmente associados a dispositivos das marcas Garmim, TomTom e Mappy, como pode ser visto em [Nutter 2008)], [Eijk e Roeloffs 2010], [Arbelet 2014] e [Lim, Lee, Park e Lee 2014]. Também foram desenvolvidos estudos relativos a recuperação de mensagens do protocolo NMEA, como pode ser observado em [Si e Aung 2011], bem como estudos relativos à precisão dos dados obtidos pelos mecanismos de GPS dos dispositivos móveis, como em [Guido, Gallelli, Saccomanno, Vitale, Rogano e Festa 2014] e [Spreitzenbarth, Schmitt e Freiling 2012].

Por fim, no que tange a análise de dados de GPS em dispositivos Android, em [Maus, Höfken e Schuba 2011] foi desenvolvido um trabalho onde coordenadas de GPS eram recuperadas de arquivos de log e bancos de dados de aplicações, todavia a análise de memória volátil não foi explorada.

Este trabalho possui como diferencial a abordagem voltada à reconstrução de trajetórias utilizando-se de dados de velocidade, geoposicionamento e tempo recuperados da memória volátil.

[1] https://github.com/504ensicsLabs/LiME - acessado em 07 jan. 2016

3. Sistemas de posicionamento global na plataforma Android

Para que sejam extraídas coordenadas GPS da memória volátil, primeiramente é necessário entender a arquitetura dos sistemas Android e como se dá a comunicação do chip receptor GPS com as diversas camadas do sistema. A seguir, são abordados o protocolo NMEA e sua relação com a reconstrução de trajetórias de dispositivos móveis.

3.1. Protocolo NMEA

O protocolo NMEA 0183 é o mecanismo de comunicação padrão entre os receptores GPS e os *drivers* dos dispositivos receptores na arquitetura Android. A ideia do protocolo, no que diz respeito a coordenadas de posicionamento, consiste em enviar uma linha de dados, codificada em ASCII, que pode conter informações de posição, velocidade, tempo, dentre outros elementos processados pelo receptor GPS. Cada mensagem é iniciada com o caractere "$" e terminada com o retorno de carro e a alimentação de linha (CR/LF). Além disso, as mensagens têm um prefixo de duas letras que define o tipo de dispositivo (chamado de *TALKER Identifier*), seguido de três letras que definem o conteúdo da mensagem. Para receptores de GPS o prefixo utilizado é o "GP" [*National Marine Electronics Association* 2002].

O formato da mensagem é simples, de forma que os dados contidos em uma linha são separados unicamente por vírgula (*comma-separated format*). Também há um campo, no fim de cada mensagem, para verificação de integridade dos dados (*checksum*) que é separado do restante da mensagem pelo caractere "*".

Há uma grande quantidade de tipos de mensagens NMEA, entretanto, no escopo desta pesquisa, cujo foco é a recuperação de coordenadas de posicionamento, de dados temporais e de velocidades, dois tipos de mensagens têm maior relevância: GPGGA e GPRMC. A Figura 1 mostra exemplos das referidas mensagens. Maiores informações sobre a estrutura das mensagens podem ser encontradas na especificação NMEA 0183.

```
$GPRMC,171925.000,A,1547.452057,S,04753.970342,W,27.4,104.7,060216,,,A*50 - Velocidade em km/h: 50.7448

$GPGGA,171959.000,1547.507968,S,04753.891545,W,1,08,0.8,1116.5,M,-9.8,M,,*7C
```

Figura 1. Mensagens GPRMC e GPGGA (coordenadas sublinhadas, velocidade, expressa em nós, circulada).

3.2. Arquitetura GPS em Sistemas Android

O sistema operacional Android possui o código fonte aberto (*open source*) o que facilita o entendimento de sua arquitetura. No que diz respeito à arquitetura GPS, o sistema pode ser dividido em camadas, que vão desde os aplicativos até o nível de hardware. Nas camadas mais acima estão os aplicativos e os *frameworks* que fazem uso de serviços de posicionamento.

Em nível de sistema, funcionando como base para os serviços de localização, existe toda uma *engine* para receber os dados do receptor GPS e fornecê-los para camadas superiores. A partir da versão 4.0 dos sistemas Android, alguns fabricantes começaram a utilizar o projeto GPSD para monitorar os receptores GPS embutidos nos aparelhos [GPSD Project 2015]. É importante ressaltar que alguns fabricantes não fazem

uso do projeto GPSD, possuindo outros mecanismos para gerenciar os dados de posicionamento. A *engine* também possui arquivos de configuração, geralmente em formato xml, que variam de nome e localização a depender da plataforma [*XDA Developers* 2012].

Nas camadas mais baixas existem os *drivers* e os chips receptores GPS. Os *drivers* utilizam API´s de baixo nível para se comunicar com o receptor GPS e geralmente consistem de um ou mais arquivos que se iniciam com o prefixo gps e terminam com a extensão ".so" [*XDA Developers* 2012].

4. Extração de Mensagens NMEA da memória volátil de sistemas Android

As mensagens NMEA perpassam praticamente todas as camadas da arquitetura GPS em ambientes Android, desde a comunicação *driver*/receptor até os *frameworks* em nível de aplicação. Como consequência, tais informações podem ser encontradas em diferentes regiões de memória, sendo de grande interesse forense, como é mostrado na seção 4.1. Na seção 4.2 é explicado o cenário de testes que foi montado para analisar a persistência dos dados na memória volátil, bem como quais técnicas podem ser utilizadas para preservar os dados.

4.1. Mensagens NMEA na memória RAM

As mensagens, como já dito anteriormente, são transmitidas por diversas camadas do sistema operacional. Por padrão, a especificação do protocolo NMEA estabelece que as mensagens são transmitidas em codificação ASCII. As camadas mais baixas da arquitetura GPS (*deamon gpsd* e os *drivers*) são escritas na linguagem de programação C, que geralmente trata *strings* como uma sequência de *bytes* codificados em ASCII. Desta forma, algumas mensagens serão encontradas na memória como uma simples sequência de caracteres codificada em ASCII, como pode ser visto na Figura 2.

Por outro lado, os aplicativos e *frameworks*, em ambientes Android, são escritos utilizando-se a linguagem de programação Java. Java é baseada no *charset* Unicode, o qual suporta a maioria dos símbolos utilizados no mundo. A especificação original do Unicode definia os caracteres como elementos de tamanho fixo, com 16 bits.

Entretanto, o padrão veio sendo alterado com o passar dos anos para permitir representações que necessitem de mais de 16 bits. Logo, as mensagens do protocolo NMEA, quando utilizadas nas camadas mais altas da arquitetura GPS, apresentam uma codificação diferente, utilizando mais *bytes*, como pode ser visto na Figura 2.

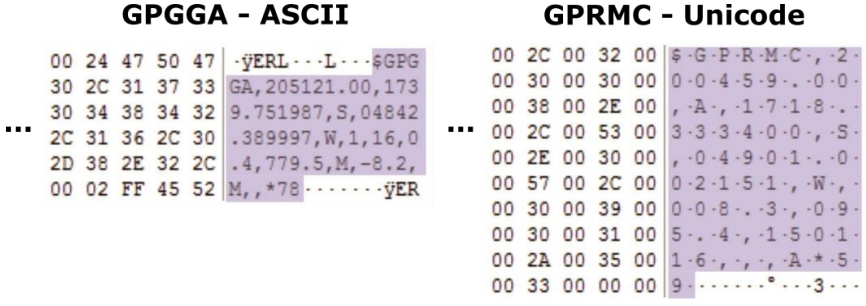


Figura 2. Mensagens GPGGA e GPRMC em ASCII e Unicode, respectivamente.

Para realizar a extração e análise dos dados, foi construída uma ferramenta que percorre o *dump* da memória RAM em busca de mensagens da especificação NMEA 0183, tendo como entrada um arquivo contendo a imagem da memória volátil e como saída arquivos de log e arquivos no formato GPX (*GPS Exchange Format*), que é um formato aberto para apresentação de dados de geoposicionamento. A ferramenta, que é de código aberto, está disponível no repositório Github[2].

4.2. Persistência das mensagens NMEA na memória volátil

O sistema operacional Android possui mecanismos diferenciados de gestão de memória que se adequam às especificidades dos dispositivos móveis. Consequentemente, a recuperação dos dados de posicionamento da memória volátil tem grande relação com o estado do sistema operacional no momento da coleta dos dados, ou seja, tem relação direta com o número de processos em execução, bem como a demanda por memória RAM por parte dos aplicativos nos momentos que antecedem a coleta dos dados.

Para melhor analisar a viabilidade da recuperação de dados de GPS em ambientes de reposta a incidentes, foram realizados testes em diferentes cenários com 4 dispositivos móveis. Os aparelhos escolhidos possuíam características diversas, utilizando diferentes versões do sistema Android e diferentes arquiteturas (ARM e x86). Os dispositivos utilizados podem ser visualizados na Tabela 1.

Tabela 1. Especificações dos dispositivos móveis

Fabricante/ Modelo	*Android*	Chipset	Memória
Samsung / GT-P5200 (*Tablet*)	4.2.2 (Jelly Bean)	Intel(R) Atom(TM) CPU Z2560 @ 1.60GHz	944372 kB
Samsung / SM-G3812B (*Smartphone*)	4.2.2 (Jelly Bean)	PXA1088 (ARMv7 Processor rev 3 (v7l))	804440 kB
Sony / LT22i (*Smartphone*)	4.1.2 (Jelly Bean)	NovaThor U8500 ARMv7 Processor rev 1 (v7l)	820588 kB
Sony / Z2 (*Smartphone*)	5.0.2 (Lollipop)	Qualcomm MSM8974PRO-AB (ARMv7 Processor rev 1 (v7l))	2846028 kB

Os experimentos foram realizados com o uso dos aplicativos Waze[3] e Google Maps[4], dada sua grande utilização pelos usuários Android. Os dispositivos foram inicialmente submetidos a *exploits* para escalada de privilégios e acesso como usuário administrador (*root*). Importante frisar que não foram utilizadas técnicas que envolvem substituição da partição *Recovery*, nem qualquer outra técnica que pudesse reiniciar o aparelho, visto que em casos reais o aparelho não pode ser desligado. Também foram previamente compilados módulos LiME para extração da memória RAM de acordo com as características de cada aparelho.

Para realização dos testes, todos os dispositivos foram colocados em modo avião, de forma a evitar o acesso a redes *wireless* e o acionamento do modo GPS assistido (A-GPS). Também foram removidos os cartões SIM (*simcards*). Desta forma, garantiu-se o uso dos aplicativos no modo *GPS only*. Para se obter os dados, foram realizados testes sob diferentes condições de tráfego e clima. Os dispositivos foram

[2] https://github.com/jpclaudino/nmeaSearch - acessado em 12 jun. 2016
[3] https://play.google.com/store/apps/details?id=com.waze - acessado em 17 jan. 2016
[4] https://play.google.com/store/apps/details?id=com.google.android.apps.maps - acessado em 17 jan. 2016

montados em um veículo de teste, que percorreu três rotas distintas. A primeira rota consistiu em um pequeno trecho de 4 km, a segunda em um trecho médio de aproximadamente 15 km e a terceira rota em um trecho longo de 150 km.

4.2.1 Experimento em rotas curtas

Os testes em rotas curtas consistiram em analisar o comportamento da memória RAM ao longo do tempo, verificando a viabilidade de reconstrução do caminho percorrido pelo dispositivo, mesmo já tendo passado um certo período de tempo. Foram realizados *dumps* da memória dos aparelhos durante um período de 30 minutos. Para cada dispositivo, após ser percorrido o trajeto de 4 km com um dos aplicativos ligados, o veículo foi parado e foi iniciada a recuperação dos dados da memória. A cada 5 minutos um novo *dump* de memória era produzido, com exceção do aparelho Sony Z2, que demandou 10 minutos, devido a maior quantidade de memória RAM.

Neste cenário, foram realizados dois tipos de testes. No primeiro, os serviços de localização do Android não foram interrompidos. Desta forma, os dados do receptor GPS continuaram sendo recebidos pelos aplicativos. No segundo caso, os serviços de localização foram desabilitados, no momento que o veículo parou, de forma que os aplicativos não mais conseguiam determinar rotas.

As imagens das memórias dos dispositivos foram então submetidas ao processamento pela ferramenta de extração e análise. A Tabela 2 mostra a quantidade de mensagens recuperadas de acordo com o tempo em que o *dump* foi realizado. O quadro quantitativo leva em consideração somente mensagens GPGGA e GPRMC válidas, ou seja, que continham coordenadas GPS.

Tabela 2. Quadro quantitativo de mensagens NMEA recuperadas a cada *dump*

Dispositivo	Serviço de Localização	Tempo transcorrido em minutos						
		0	5	10	15	20	25	30
Samsung GT-P5200	Habilitado	294	274	284	302	342	311	345
Samsung GT-P5200	Desabilitado	307	87	65	3	3	2	2
Samsung SM-G3812B	Habilitado	279	273	272	255	346	324	238
Samsung SM-G3812B	Desabilitado	86	46	26	24	22	12	8
Sony LT22i	Habilitado	275	364	355	382	396	431	316
Sony LT22i	Desabilitado	214	144	114	93	75	69	71
Sony Z2	Habilitado	1175	N/A	195	N/A	159	N/A	125
Sony Z2	Desabilitado	743	N/A	299	N/A	131	N/A	113

Observou-se que as coordenadas de posicionamento tendem a se agrupar no ponto de repouso do veículo, quando o serviço de localização estava ativo, como era de se esperar. Já a quantidade de mensagens recuperadas quando o serviço de localização estava inativo tende a reduzir com o passar do tempo, entretanto, em alguns dispositivos, as coordenadas de posicionamento recuperadas forneceram mais informações sobre o percurso completo, mesmo depois de transcorridos 30 minutos, se comparadas com aquelas recuperadas nos testes com o serviço de localização habilitado.

A Figura 3 mostra imagens, criadas com o software *Google Earth*[5], das coordenadas GPS recuperadas do dispositivo Sony Z2, nos testes realizados. É possível observar, na figura, que as coordenadas ao longo do trajeto foram perdidas depois de transcorridos 30 minutos, no cenário com os serviços de localização habilitados. Entretanto, no cenário com os serviços de localização desabilitados, foi possível reconstruir parcialmente a trajetória do veículo, mesmo depois de transcorridos 30 minutos.

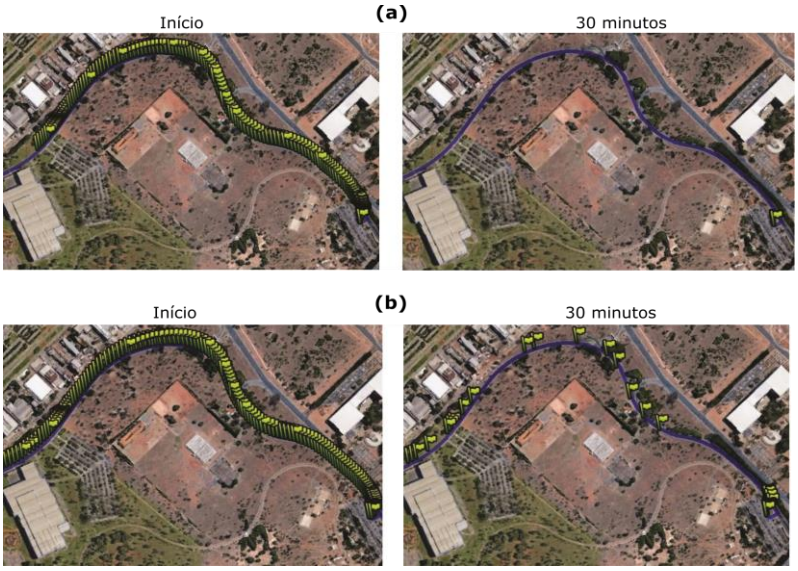


Figura 3. Coordenadas de posicionamento recuperadas do aparelho Sony Z2 nos testes com os serviços de localização habilitados (a) e desabilitados (b).

Os dados também mostram um comportamento diferenciado do aparelho Sony Z2, com uma alta taxa de mensagens recuperadas nos primeiros *dumps*, seguido de uma queda abrupta com o passar do tempo. Tal variação ocorreu devido ao fato de o aparelho em análise possuir a versão *Lollipop* do Android, que utiliza o *Android Runtime* (ART) como ambiente de execução, ao invés de seu predecessor *Dalvik*[6]. O sistema de gerenciamento de memória do ART foi aprimorado, principalmente no que diz respeito a *garbage collection*, sendo mais eficiente na liberação de memória.

Cabe ressaltar que, independentemente da abordagem utilizada (desligando ou não os serviços de localização), foi possível recuperar informações sobre determinados pontos do trajeto percorrido, inclusive com dados sobre velocidade e tempo.

4.2.2 Experimento em rotas médias

Para simular casos de condutores de veículos em ambientes urbanos, foi percorrido um trajeto de 15 km. Os dispositivos Samsung SM-G3812B e Sony Z2 foram testados em condições de clima desfavoráveis para o uso de GPS, com chuva intensa e nuvens carregadas. Assim como no caso anterior, o veículo foi parado e foi iniciada a recuperação dos dados da memória. Foram realizados *dumps* da memória dos aparelhos no período de uma hora. Nos primeiros 30 minutos, uma imagem era produzida a cada dez minutos. Depois deste período, foi feito um último *dump* depois de passados 60 minutos do veículo em repouso.

[5] https://www.google.com/earth - acessado em 02 jan. 2016
[6] https://source.android.com/devices/tech/dalvik - acessado em 07 jan. 2016

Neste cenário os serviços de localização não foram desligados, simulando casos de acidentes de trânsito, onde o tempo de reposta das equipes periciais geralmente é longo. A Tabela 3 mostra a quantidade de mensagens recuperadas ao longo do tempo. A Figura 4 mostra imagens das coordenadas GPS recuperadas do dispositivo Samsung GT-P5200, após transcorridos 60 minutos da primeira extração de dados. Importante observar que, mesmo transcorrido o intervalo de 60 minutos após o veículo entrar em repouso, foi possível recuperar informações do trajeto percorrido, bem como dados de velocidade, que podem ser muito importantes em investigações de acidentes de trânsito.

Tabela 3. Quadro quantitativo de mensagens recuperadas a cada *dump* (15 km)

Dispositivo	Tempo transcorrido em minutos				
	0	10	20	30	60
Samsung GT-P5200	177	243	337	315	330
Samsung SM-G3812B	11	293	336	303	377
Sony LT22i	200	363	177	333	272
Sony Z2	294	985	234	180	228

Os experimentos também mostraram que a precisão dos receptores GPS em condições climáticas adversas, como no caso de chuva intensa, pode ser altamente afetada. Os dados coletados do aparelho Samsung SM-G3812B mostram erros no cálculo de posicionamento que variaram entre 1200 m a 1500 m do percurso original, como pode ser visualizado na Figura 5. Neste sentido, deve-se levar em consideração outros aspectos das mensagens NMEA, as quais possuem informações sobre o número de satélites visíveis, bem como a precisão dos dados recebidos.

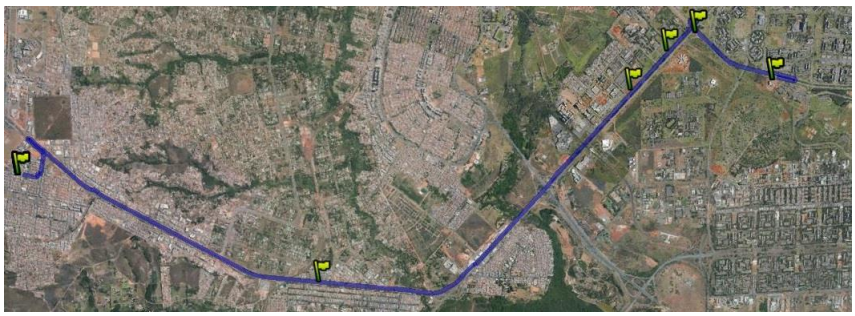


Figura 4. Plotagem dos dados recuperados do dispositivo Samsung GT-P5200, após transcorridos 60 minutos.

Figura 5. Erro de medição em condições climáticas adversas.

4.2.3 Experimento em rotas longas

Neste experimento, foi utilizado somente o Tablet Samsung GT-P5200. Foi percorrido um trajeto de 150 km em aproximadamente 2h. Ao final do trajeto, após 10 minutos com o veículo em repouso, foi realizado o processo de *dump* da memória RAM. Os dados foram então extraídos e analisados, tendo sido encontradas 392 mensagens NMEA na memória. Neste cenário, os dados foram coletados após uso contínuo e por um longo período do aplicativo Google Maps. Mesmo após o decurso de duas horas, foi possível recuperar informações de grande parte do trajeto, como mostrado na Figura 6.

Figura 6. Plotagem dos dados recuperados do dispositivo Samsung GT-P5200, após ser percorrido um trajeto de 150 Km.

O aplicativo fez uso de mais memória RAM do que nos testes anteriores, e mais dados persistiram por mais tempo. Algumas informações podem ter grande relevância forense e podem indicar o comportamento do usuário do dispositivo. Os dados recuperados, ainda que não forneçam precisamente todo o trajeto percorrido, contêm informações que permitem inferir aspectos como clima, tempo e velocidade média em determinados trechos.

5. Conclusão e Trabalhos Futuros

Neste artigo foi demonstrada a viabilidade da reconstrução de trajetórias de dispositivos móveis Android com base em coordenadas de posicionamento recuperadas da memória RAM. Além disso, foi feita uma análise da arquitetura GPS em sistemas Android, o que resultou na produção de uma ferramenta para recuperação de dados do protocolo NMEA 0183 relevantes no contexto forense.

Foi possível observar que diversos fatores podem ter influência na recuperação dos dados da memória RAM, como o estado do sistema operacional, a ferramenta de extração de dados e as configurações de software e hardware do dispositivo.

Desta forma, este trabalho se diferencia dos citados anteriormente uma vez que, além dos estudos na área de extração de memória volátil e forense em GPS, o estudo aborda a reconstrução da trajetória de dispositivos móveis com base no protocolo NMEA 0183.

Futuramente será construída uma ferramenta para analisar não somente as mensagens NMEA recuperadas da memória RAM, mas também outros dados com informações de GPS armazenadas no contexto das aplicações. Um dos grandes empecilhos para extração de dados da memória volátil é o procedimento utilizado atualmente, com o uso do módulo LiME, que depende da prévia compilação com base no *kernel* do dispositivo em evidência.

Trabalhos futuros podem explorar mecanismos de extração de memória RAM que independam de prévia compilação de *kernel*. Além disso, os dispositivos estão em constante evolução e periodicamente são lançadas correções de segurança que inibem a escalada de privilégios, o que demanda estudos constantes dos mecanismos de *rooting*.

Referências

Arbelet, A. (2014). Garmin satnav forensic methods and artefacts: an exploratory study. Doctoral dissertation, Edinburgh Napier University.

GPSD Project. (2015). gpsd - a GPS service daemon. Acessado em: 14 jan. 2016. http://www.catb.org/gpsd/index.html.

Guido, G., Gallelli, V., Saccomanno, F., Vitale, A., Rogano, D. and Festa, D. (2014). "Treating uncertainty in the estimation of speed from smartphone traffic probes." Transportation Research Part C: Emerging Technologies, 47, p. 100-112.

Lim, K. S., Lee, C., Park, J. H., and Lee, S. J. (2014). "Test-driven forensic analysis of satellite automotive navigation systems", Journal of Intelligent Manufacturing, 25(2), p. 329-338.

Maus, S., Höfken, H. and Schuba, M. (2011). "Forensic analysis of geodata in android smartphones.", International Conference on Cybercrime, Security and Digital Forensics, http://www. schuba. fh-aachen. de/papers/11-cyberforensics.pdf.

Müller, T. and Spreitzenbarth, M. (2013). "Frost", Applied Cryptography and Network Security. p. 373-388. Springer Berlin Heidelberg.

National Marine Electronics Association. (2002). NMEA 0183 - Standard for interfacing marine electronic devices, Version: 3.01. NMEA.

Nutter, B. (2008). "Pinpointing TomTom location records: A forensic analysis.", Digital Investigation, 5(1), p. 10-18.

Si, H. and Aung, Z. M. (2011). "Position data acquisition from NMEA protocol of global positioning system.", International Journal of Computer and Electrical Engineering, 3(3), p. 353.

Spreitzenbarth, M., Schmitt, S. and Freiling, F. (2012). "Comparing Sources of Location data From Android Smartphones.", Advances in Digital Forensics VIII. p. 143-157. Springer Berlin Heidelberg.

Stüttgen, J. and Cohen, M. (2014). "Robust Linux memory acquisition with minimal target impact", Digital Investigation, 11, S112-S119.

Sylve, J., Case, A., Marziale, L. and Richard, G. G. (2012). "Acquisition and analysis of volatile memory from android devices", Digital Investigation, 8(3), p. 175-184.

Thing, V. L., Ng, K. Y. and Chang, E. C. (2010). "Live memory forensics of mobile phones", Digital Investigation, 7, S74-S82.

Van Eijk, O. and Roeloffs, M. (2010). "Forensic acquisition and analysis of the Random Access Memory of TomTom GPS navigation systems.", Digital Investigation, 6(3), p. 179-188.

XDA Developers. (2012). Understanding Android GPS Architecture. (n.d.). Acessado em: 14 jan. 2016. http://forum.xda-developers.com/showthread.php?t=2063295.

Modelagem de ameaças antiforenses aplicada ao processo forense digital

Marcelo B. Maués[1], Bruno Werneck P. Hoelz[2*]

[1]Departamento de Engenharia Elétrica - Universidade de Brasília
[2]Instituto Nacional de Criminalística – Polícia Federal
Brasília, DF – Brasil

marcelomaues76@hotmail.com, werneck.bwph@dpf.gov.br

Abstract: *In digital forensics, the role of the expert is to collect and analyze digital evidence. However, anti-forensics actions threaten the forensic examination process and may compromise its conclusions. This work proposes a threat modeling process in order to reduce the* risks *associated with anti-forensics threats. The proposed process introduces risk management activities as a complement to digital forensic processes found in the literature, allowing a systemic approach for measuring risk and employing countermeasures and risk mitigation strategies.*

Resumo: *Na perícia forense digital, o papel do perito é coletar e analisar as evidências digitais. No entanto, ações antiforenses ameaçam o processo do exame pericial, podendo comprometer suas conclusões. Este trabalho propõe um processo de modelagem de ameaças com o objetivo de reduzir os riscos de ameaças antiforenses. O processo proposto introduz atividades de gestão de risco como um complemento aos processos de perícia digital encontrados na literatura, permitindo uma abordagem sistêmica para a avaliação de risco e o emprego de contramedidas e estratégias de mitigação de risco.*

1 Introdução

A perícia forense digital busca evidências digitais para o esclarecimento de um incidente a partir de informações existentes em diferentes tecnologias, por exemplo, computadores, telefones móveis e redes de computadores (SACHOWSKI, 2016). Esse trabalho é realizado, predominantemente, em ambientes fora do controle do perito, dos quais informações prévias podem ser escassas ou inexistentes. Por isso, a realização do exame pericial requer preparação prévia das ferramentas, técnicas e procedimentos a serem aplicados em cada cenário de atuação.

Um dos aspectos comumente ignorados nessa preparação, é a avaliação de riscos associados a ameaças antiforenses. Essas ameaças estão relacionadas a qualquer tentativa de comprometer a disponibilidade ou utilidade de evidências digitais. Segundo Conlan et al. (2016), meios antiforenses têm se tornado um grande obstáculo para a comunidade forense, exigindo novas iniciativas e estratégias de investigação para resolver esse problema crescente. O resultado antiforense pode ser obtido com o uso de ferramentas ou

* Os autores agradecem o apoio da Secretaria Nacional de Segurança Pública (SENASP), da Diretoria Técnico-Científica da Polícia Federal e da FINEP (Convênio 01.12.0433.01, Projeto: Defesa Nacional e Segurança Pública) na realização deste trabalho.

métodos maliciosos, muitas de conhecimento público, ou simplesmente pelo uso de proteções legítimas como senhas e criptografia.

Apesar da constatação de que ações antiforenses representam uma ameaça ao processo forense digital, tal preocupação não se reflete nos modelos de processos encontrados na literatura, que não consideram a gerência de riscos como uma fase do processo pericial, conforme pode ser observado em Sachowski (2016). Embora diversos estudos sejam realizados sobre detecção de ações antiforenses, como o uso de esteganografia, a ausência de um processo bem definido de identificação e avaliação de riscos contribui para que técnicas de detecção não sejam incorporadas ao processo pericial ou que sejam aplicadas desnecessariamente. No primeiro caso, evidências digitais cruciais ao exame podem deixar de ser recuperadas por falta de tratamento adequado, enquanto no segundo há desperdício de recursos, especialmente de tempo.

Diante dessa realidade, este trabalho propõe um processo de modelagem de ameaças antiforenses para complementar processos propostos na literatura que não consideram tais riscos. O processo proposto também permite que uma organização avalie continuamente seu nível de preparação, seja em termos de treinamento ou das ferramentas disponíveis, identificando lacunas que apresentem risco à realização adequada do exame pericial. Por fim, o processo proposto também auxilia na adoção do preceito de prontidão forense (ou *forensic readiness)*, segundo o qual deve-se maximizar o uso de evidências digitais e ao mesmo tempo minimizar custo de uma investigação forense digital (SULE, 2014).

O restante deste artigo está organizado da seguinte maneira: na Seção 2, é discutida a modelagem de ameaças antiforenses aplicada ao processo forense digital; na Seção 3 é demonstrada a aplicação do processo de modelagem proposto através de um estudo de caso e, por fim, na Seção 4, são apresentadas as conclusões e possibilidades de estudos futuros.

2 Modelagem de ameaças antiforenses aplicada ao processo pericial

A modelagem de ameaças é amplamente utilizada no desenvolvimento de software com o objetivo de analisar a segurança do aplicativo, identificando, quantificando e tratando riscos associados ao sistema de uma forma estruturada (OWASP, 2015). Na literatura, é possível encontrar diversas formas de conduzir um processo de modelagem de ameaças, conforme pode ser visto em Sindre e Opdahl (2005), Shostack (2014) e OWASP (2015). Entretanto, no âmbito da perícia digital, os processos propostos na literatura não consideram a gestão dos riscos associados às ameaças antiforenses. Por isso, não foram encontradas aplicações anteriores de técnicas de modelagem de ameaças no processo pericial.

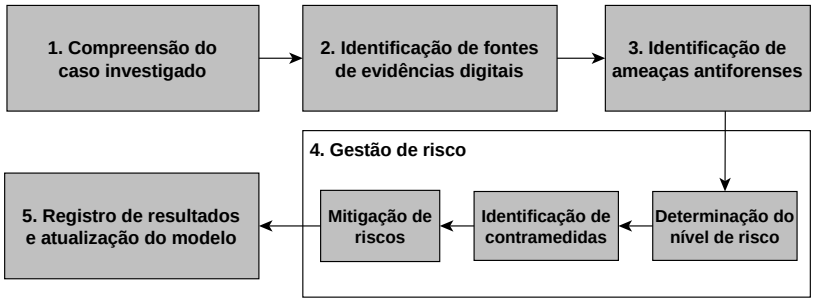


Figura 1 - Processo de modelagem de ameaças antiforenses

O processo de modelagem proposto neste trabalho foi desenvolvido tendo como referência os processos utilizados no desenvolvimento de software. Assim, o modelo de ameaças resultante visa identificar e tratar riscos de ameaças antiforenses que podem afetar a coleta de evidências digitais durante cada uma das fases do processo pericial. O modelo proposto é dividido em cinco etapas, detalhadas a seguir, conforme a Figura 1.

A primeira etapa é de compreensão do caso investigado, cujo objetivo é levantar informações para auxiliar a tomada de decisões em etapas futuras do processo de modelagem. Com base no conhecimento de especialistas, um questionário foi elaborado para nortear a coleta de informações. O questionário está voltado a aspectos relacionados ao suspeito, ao ambiente a ser periciado e a ação criminal, como, por exemplo: há suspeitos com conhecimento avançado de Informática? Que tipos de equipamentos estão envolvidos no incidente?

A identificação das fontes de evidências digitais é a segunda etapa. Nela, são identificados os meios de armazenamento de dados onde podem ser encontradas evidências digitais relacionadas ao incidente. As evidências digitais podem ser obtidas de diversas fontes tais como arquivos produzidos por usuários, *logs* do sistema operacional, históricos de navegadores da Internet ou atributos de arquivos. Também são consideradas fontes de evidências digitais, dispositivos como câmeras digitais, consoles de jogos ou GPS.

Em seguida, é realizada a identificação de ameaças antiforenses**.** Essa etapa consiste em analisar cada uma das fontes de evidências digitais e verificar quais ações antiforenses podem ser aplicadas para comprometê-las. Para auxiliar esta etapa, é proposto um catálogo com registros de ações antiforenses identificadas durante a pesquisa. Por exemplo, criptografia de disco, esteganografia, ocultação de dados em *slack space*[1], entre outros. Seguindo a ideia de busca de ameaças por categorias adotado no modelo STRIDE (SHOSTACK, 2014), amplamente utilizado no desenvolvimento de software, o catálogo é organizado em categorias. As categorias foram definidas com base na classificação de ações antiforenses sugerida por Harris (2006). Segundo o autor, ações antiforenses podem ser classificadas quanto à destruição, ocultação, falsificação e eliminação de fontes das evidências digitais. Cabe ressaltar que o catálogo precisa ser alimentado sempre que se tenha conhecimento de novas técnicas antiforenses.

A gestão de risco é realizada em seguida. Essa etapa tem como objetivo estimar os riscos que as ameaças antiforenses representam ao processo forense digital. Isso permitirá determinar quais delas devem ou não ser mitigadas, pois tratar todas nem sempre é viável tendo em vista os recursos, incluindo o tempo, disponíveis. Essa etapa é discutida em detalhes na seção 2.1.

Por fim, é realizado o registro dos resultados e a atualização do modelo. Nessa etapa, é gerado um relatório com o resultado das etapas anteriores. Assim, é possível recorrer posteriormente a essa documentação para revisar a avaliação realizada pelo perito ou verificar ameaças que porventura não foram consideradas, mas que foram detectadas durante o exame. Essa etapa também é utilizada para a atualização dos catálogos propostos anteriormente: ocorrências de ações antiforenses, ameaças e contramedidas.

[1] Sobras de espaços no disco rígido que não podem ser utilizadas pelo sistema de arquivos

2.1 Gestão de riscos

A gestão de riscos é a principal etapa do processo de modelagem proposto. Ela está divida em três partes: (1) determinação do nível de risco, (2) identificação de contramedidas e (3) mitigação de riscos. A Figura 2 apresenta os elementos envolvidos na gestão do risco. Os três elementos principais são o suspeito, a ameaça e o risco. A motivação, capacidade e oportunidade do suspeito são elementos determinantes na probabilidade da ameaça, enquanto o tipo da ameaça determina seu impacto. A probabilidade e o impacto são então utilizados para determinar o risco.

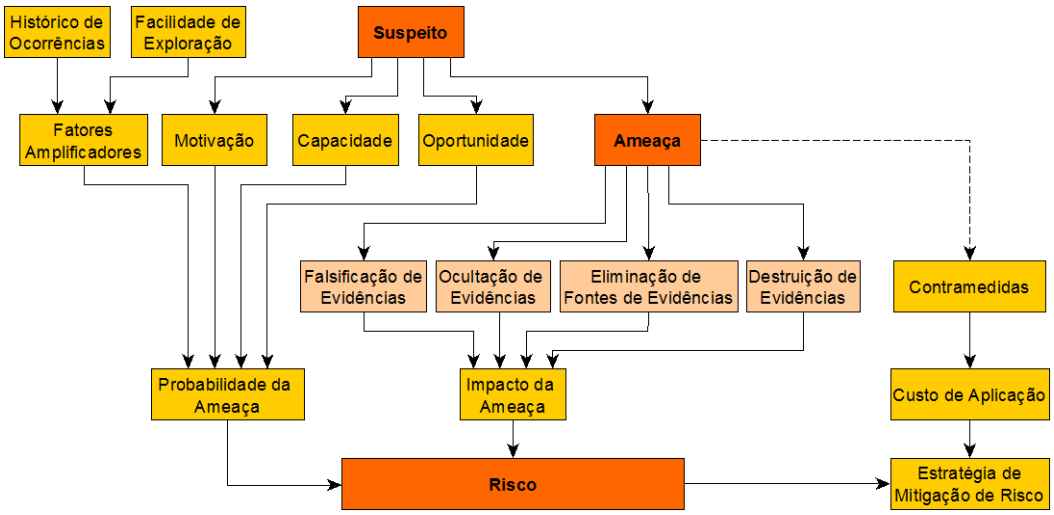


Figura 2 - Elementos envolvidos na gestão de riscos

2.1.1 Determinação do nível de risco

A metodologia proposta para determinação do risco tem como base a publicação especial 800-30 do NIST (*National Institute of Standards and Technology*), bem como as orientações da NBR ISO/IEC 31010:2012, sobre técnicas de avaliação de risco. O nível de risco é determinado pela combinação de fatores relacionados à probabilidade e impacto (NIST, 2002). Neste trabalho, a probabilidade está relacionada à possibilidade de ocorrência da ameaça antiforense e o impacto às consequências que a ameaça antiforense pode causar ao processo forense digital e seus resultados.

Para estimar a probabilidade, são considerados fatores relacionados ao agente da ameaça (suspeito) como motivação, capacidade e oportunidade, fatores considerados determinantes por Vidalis e Jones (2005). A capacidade está relacionada às condições que o suspeito possui para aplicação da ação antiforense. O suspeito tem conhecimento técnico para aplicar a ação antiforense ou conta com profissionais capacitados? Possui condições financeiras para compra de *software* e *hardware*, caso seja necessário? A motivação está relacionada ao custo/benefício do uso de ações antiforenses pelo suspeito. A aplicação de algumas técnicas antiforenses pode ser ou não compensadora para certas ações criminais. Já a oportunidade refere-se às circunstâncias que favorecerem a aplicação da ação antiforense pelo suspeito, como, por exemplo, a existência de uma técnica antiforense pouco documentada e divulgada, inexistência de *software* adequado para tratamento, dificuldade de detecção, etc. Para determinar os fatores de capacidade, motivação e oportunidade relacionados ao suspeito no cálculo da probabilidade da ameaça, são propostas as pontuações apresentadas na Tabela 1.

Além da capacidade, motivação e oportunidade, alguns fatores podem aumentar a probabilidade de ocorrência de uma ação antiforense. Esses fatores são chamados de fatores amplificadores. Segundo Jones (2002), fatores amplificadores são influências que podem contribuir na ocorrência de um incidente. Para o modelo proposto, são fatores amplificadores:

- histórico de ocorrências: relacionado ao emprego anterior da ação antiforense;
- facilidade de exploração: nível de recursos necessário para exploração da ação antiforense. A existência de ferramentas para execução da ação e de documentação do método são exemplos de facilitadores.

Tabela 1 - Pontuação associada à avaliação da capacidade, motivação e oportunidade

Pts.	Capacidade	Motivação	Oportunidade
20	O suspeito possui amplas condições de fazer uso da ação antiforense.	O uso da ação antiforense pelo suspeito compensa muito à prática do delito investigado.	As circunstâncias são altamente favoráveis para aplicação da técnica antiforense.
10	O suspeito possui moderadas condições de fazer uso da ação antiforense.	O uso da ação antiforense pelo suspeito compensa moderadamente à prática do delito investigado.	As circunstâncias são moderadamente favoráveis para aplicação da técnica antiforense.
5	O suspeito possui poucas condições de fazer uso da ação antiforense.	O uso da ação antiforense pelo suspeito compensa pouco à prática do delito investigado.	As circunstâncias são pouco favoráveis para aplicação da técnica antiforense.
0	O suspeito não apresenta condições de fazer uso da ação antiforense.	O uso da ação antiforense pelo suspeito não compensa à prática do delito investigado.	As circunstâncias não são favoráveis para aplicação da técnica antiforense.

Em relação ao "Histórico de Ocorrências", um modelo de catálogo é proposto para registrar ações antiforenses identificadas em exames anteriores ou conhecidas a partir de outras fontes de informação como outros órgãos periciais ou trabalhos científicos. Este catálogo é denominado de catálogo de "Ocorrências de Ações Antiforenses". Para sua eficácia, o catálogo precisa ser atualizado constantemente. Ferramentas e procedimentos utilizados na detecção e tratamento dessas ações são registrados em outro catálogo, de contramedidas, apresentado na Seção 2.1.2.

A Tabela 2 deve ser usada para determinar a pontuação dos fatores amplificadores. A pontuação para cada um dos fatores foi estabelecida de forma que fatores relacionados ao suspeito (capacidade, motivação e oportunidade) fossem suficientes para indicar uma alta probabilidade de ocorrência de uma ação antiforense.

Tabela 2 – Pontuação associada ao histórico de ocorrências e facilidade de exploração

Pts.	Histórico de ocorrências	Facilidade de exploração
20	A ação antiforense foi amplamente utilizada em situações anteriores.	Há diversos facilitadores para aplicação da técnica antiforense.
15	A ação antiforense foi moderadamente utilizada.	Há alguns facilitadores para aplicação da técnica antiforense.
10	A ação antiforense foi pouco utilizada.	Há poucos facilitadores para aplicação da técnica antiforense
0	Não há relatos da aplicação da ação antiforense.	Não há facilitadores para aplicação da técnica antiforense.

Para os fatores amplificadores, a pontuação utilizada visa potencializar a probabilidade de ocorrência, sendo incapaz de, isoladamente, estabelecer uma probabilidade alta de ameaça. No entanto, em ações que são fácil e frequentemente empregadas, a pontuação dos fatores amplificadores aumentaria a probabilidade de ameaça de baixa para média, baixa para alta e de média para alta.

Logo, a probabilidade de ocorrência da ameaça antiforense é estimada pela somatória dos pontos atribuídos a cada um dos fatores e poderá ser baixa (< 45), média (entre 45 e 70) e alta (> 70). Após a determinação da probabilidade de ocorrência da ameaça, é necessário determinar o impacto da ameaça antiforense aos procedimentos pericias. Segundo Beer, Stander e Belle (2014), as ações antiforenses impactam na recuperação e apresentação de evidências com valor probatório, podendo até levar a absolvição de um suspeito. Com base nessa afirmativa, são propostos três níveis de impacto: alto, médio e baixo, conforme a Tabela 3.

Tabela 3 - Níveis de impacto da ameaça

Impacto	
Alto	Pode comprometer totalmente a recuperação e apresentação de evidências digitais utilizáveis.
Médio	Pode comprometer parcialmente a recuperação e apresentação de evidências digitais utilizáveis.
Baixo	Pouco pode comprometer a recuperação e apresentação de evidências digitais utilizáveis.

Por fim, o risco da ameaça antiforense é obtido com a multiplicação da probabilidade pelo impacto, conforme apresentado na Tabela 4. O risco resultante poderá ser alto (entre 50 e 100), médio (de 10 a 50) e baixo (1 a 10).

Tabela 4 - Matriz de cálculo do risco da ameaça antiforense, adaptada de NIST (2002)

Risco = Probabilidade x Impacto			
Probabilidade	**Impacto**		
	BAIXO (10)	MÉDIO (50)	ALTO (100)
ALTO (1,0)	Baixo (1,0 x 10 = 10)	Médio (1,0 x 50 = 50)	Alto (1,0 x 100 = 100)
MÉDIA (0,5)	Baixo (0,5 x 10 = 5)	Médio (0,5 x 50 = 25)	Médio (0,5 x 100 = 50)
BAIXA (0,1)	Baixo (0,1 x 10 = 1)	Baixo (0,1 x 50 = 5)	Baixo (0,1 x 100 = 10)

2.1.2 Identificação de contramedidas

Após a determinação do nível de risco das ameaças, verifica-se a existência de medidas que podem minimizar os impactos nos exames periciais. É proposta a utilização de um catálogo de "Contramedidas", que mantém os procedimentos e ferramentas a serem utilizados para cada situação, bem como observações sobre o custo associado à sua aplicação. O catálogo deve ser atualizado com a descoberta de novas contramedidas.

2.1.3 Mitigação de riscos

Feito o levantamento de contramedidas que podem ser adotadas para minimizar riscos, esta etapa visa avaliar a aplicação ou não delas. A decisão mais adequada está

relacionada ao risco da ameaça e ao custo de aplicação da contramedida. O custo de aplicação pode ser alto, médio e baixo, segundo o esforço necessário para sua aplicação. Algumas medidas podem ser muito custosas e, dependendo do risco da ameaça, podem ser julgadas desnecessárias. Cabe ressaltar que a determinação do custo de aplicação da contramedida depende dos recursos, inclusive tempo, disponíveis para o perito ou órgão pericial.

Para auxiliar a decisão de aceitar ou não o risco da ameaça, uma matriz composta por valores do risco da ameaça e custo de aplicação da contramedida é proposta (Tabela 5). A combinação do risco e custo na matriz resulta em dois valores: mitigar ou aceitar. O primeiro valor (mitigar) significa que a aplicação da contramedida é indicada diante do risco da ameaça, ou seja, a contramedida deve ser aplicada. Já o segundo valor (aceitar) significa que o risco da ameaça deve ser aceito, já que o custo não compensa em face do risco avaliado. Como os custos de aplicação dependem dos recursos disponíveis, a estratégia de mitigação pode variar de uma aplicação para outra, em especial se vários riscos forem identificados. Algumas ameaças de baixo risco podem ser toleradas em face a uma ameaça simultânea de médio risco, caso os recursos para tratá-las sejam limitados. Portanto, o nível de risco não determina, isoladamente, a obrigatoriedade de qualquer ação por parte do perito. É necessário também avaliar o custo de mitigação desses riscos.

Tabela 5 - Matriz de estratégia de mitigação

Estratégia de Mitigação			
Risco	**Custo de aplicação**		
	BAIXO	MÉDIO	ALTO
ALTO	Mitigar	Mitigar	Mitigar
MÉDIO	Mitigar	Mitigar	Aceitar
BAIXO	Mitigar	Aceitar	Aceitar

3 Estudo de caso

Para validar o processo proposto, considerou-se um caso real de busca e apreensão realizado em uma empresa privada. A investigação apurava crime de exploração e abuso sexual de criança ou adolescente. Nos próximos parágrafos, são apresentados resumidamente os resultados obtidos em cada etapa da aplicação do processo de modelagem de ameaças antiforenses conduzidos pelo Perito designado para participar da operação.

Na primeira etapa (compreensão do caso investigado), com o suporte do questionário proposto, apurou-se que o caso envolvia um funcionário, lotado na Gerência de Informática, com conhecimentos avançados de informática, sem antecedentes criminais e com idade aproximada de 30 anos. Em termos de equipamentos computacionais, a operação previa apenas a apreensão do computador de uso diário do funcionário.

Na segunda etapa (identificação de fontes de evidências digitais), foram considerados como fontes potenciais de evidências digitais apenas arquivos de imagem e vídeo produzidos pelo usuário armazenados no disco rígido do computador.

Na terceira etapa (identificação de ameaças antiforenses), com suporte do catálogo de ameaças antiforenses, foram identificados diversos tipos de ameaças antiforenses que poderiam comprometer a recuperação de arquivos de imagem e vídeo. Contudo, neste trabalho, serão consideradas apenas as ameaças de ocultação de dados com uso de criptografia de disco (*full-disk encryption*) ou esteganografia.

Na quarta etapa (gestão de riscos), primeiramente foram estimados a probabilidade e o impacto e, em seguida, foi estimado o risco efetivo da ameaça utilizando a matriz de risco proposta no modelo. A Tabela 6 apresenta os resultados obtidos.

Tabela 6 - Estudo de caso: cálculo da probabilidade, impacto e risco

Ameaça antiforense	Probabilidade	Impacto	Risco (Probabilidade x Impacto)
Criptografia de disco	Capacidade + Motivação + Oportunidade + Histórico de Ocorrências + Facilidade de Exploração = 20 + 20 + 20 + 15 + 20 = 95 (Alta)	Alto (pode comprometer totalmente a recuperação e apresentação de evidências digitais utilizáveis)	Alta x Alto = Alto
Esteganografia	Capacidade + Motivação + Oportunidade + Histórico de Ocorrências + Facilidade de Exploração = 20 + 20 + 20 + 0 + 10 = 70 (Média)	Médio (pode comprometer parcialmente a recuperação e apresentação de evidências digitais utilizáveis)	Média x Médio = Médio

Estimado o risco, contramedidas foram sugeridas para cada ameaça, com apoio do catálogo de contramedidas. Cabe ressaltar que na inexistência de soluções no catálogo, outras fontes devem ser pesquisadas. Em seguida, foi feito uma análise do custo de aplicação das contramedidas para cada ameaça. A Tabela 7 apresenta as contramedidas propostas, seu custo e algumas observações sobre sua aplicação.

Tabela 7 - Estudo de caso: contramedidas identificadas e custo de aplicação

Ameaça antiforense	Contramedidas	Custo	Observações
Criptografia em disco	Obtenção dos volumes criptografados quando ainda estão montados (CM1).	Médio	Requer oportunidade adequada para realização da cópia dos dados (cujo tempo depende do volume de dados encontrado).
	Obtenção da chave criptográfica do volume (CM2).	Alto	Não há garantia de sucesso, mesmo com grande custo de tempo.
Esteganografia	Busca por ferramentas ou aplicações de esteganografia no computador do suspeito. Se forem encontradas, indagar o suspeito sobre sua utilização (CM3).	Baixo	A verificação de ferramentas conhecidas pode ser feita facilmente (caso não haja outro obstáculo), mas não é exaustiva.
	Analisar o conteúdo de arquivos com ferramentas de detecção automatizada, como s*tegdetect* (CM4).	Alto	O tempo de execução das ferramentas pode ser proibitivo dependendo do volume de dados.

Com a combinação do custo de aplicação e risco da ameaça na matriz de estratégia de mitigação sugerida no modelo, é definido se o risco da ameaça deve ser aceito ou se as contramedidas devem ser aplicadas (mitigar). A Tabela 8 apresenta esses resultados e

mostra que nem todas as contramedidas devem ser aplicadas. Por exemplo, para ameaça esteganografia foram sugeridas 02 (duas) contramedidas: CM3 e CM4. Contudo, a contramedida CM4 não deverá ser aplicada, visto que, seu custo de aplicação alto não compensa diante do risco médio da ameaça esteganografia.

Tabela 8 – Estudo de caso: estratégia de mitigação

Risco da ameaça antiforense	Custo de aplicação da contramedida	Estratégia de mitigação
Criptografia em disco: Alto	CM1: Médio	Mitigar
	CM2: Alto	Mitigar
Esteganografia: Médio	CM3: Baixo	Mitigar
	CM4: Alto	Aceitar

Na última etapa, um relatório foi gerado com todos os resultados da aplicação do processo de modelagem de ameaças antiforenses, bem como da constatação de fato dessas ameaças e do emprego das contramedidas. No caso em questão, não foram identificadas ameaças de esteganografia com o uso da busca por ferramentas ou aplicações conhecidas para esse fim. A ameaça de criptografia de disco também não foi confirmada, embora a contramedida CM1 tenha sido aplicada preventivamente, considerando que a confirmação da ameaça só seria possível após o acesso ao computador.

4 Conclusões

Este trabalho propôs a aplicação do processo de modelagem de ameaças no tratamento de riscos de ações antiforenses em processos forenses digitais. O modelo, dividido em cinco etapas (compreensão do caso investigado, identificação de fontes de evidências, identificação de ameaças, gestão de riscos e registro dos resultados), complementa as fases comumente utilizadas no processo forense digital, introduzindo a gestão de risco de ações antiforenses que possam prejudicar o resultado da atividade pericial. A proposta também contribui para complementar o processo pericial ao sugerir a adoção de um questionário para auxiliar o levantamento de informações sobre o caso investigado e de catálogos que servem de fontes de informações para auxiliar na tomada de decisões durante a aplicação do modelo de ameaça.

O estudo de caso demonstra que a incorporação do modelo proposto nas atividades do perito permite identificar e avaliar riscos de ameaças antiforenses ainda no início do processo e oferecer medidas de detecção e mitigação que podem ser aplicadas nas fases de coleta e análise de dados. Com isso, o processo pericial torna-se mais robusto, minimizando perdas de evidências digitais diante de riscos antiforenses.

Em nível organizacional, a utilização do processo proposto, incluindo a atualização constantes dos catálogos propostos, permite avaliar o nível de preparação para atender determinadas ameaças. Pode-se, por exemplo, identificar a indisponibilidade de ferramentas para detecção de esteganografia ou a necessidade de abordar um suspeito enquanto um volume criptografado está disponível no computador. Alguns riscos que podem parecer improváveis para determinadas organizações podem se apresentar críticos em outros (uso de esteganografia em casos associados a terrorismo). Portanto, a forma de aplicação da proposta depende da realidade objetiva encontrada pelos peritos. Cabe destacar que a gestão de risco no processo forense também é preventiva, em preparação às ameaças antiforenses, e não somente reativa. Portanto, algumas contramedidas

implicam em ações de planejamento anteriores à realização da coleta de dados e exame pericial, como foi possível observar no estudo de caso.

Trabalhos futuros devem ser realizados na expansão dos catálogos de ameaças e contramedidas, bem como em formas de compartilhamento desse conhecimento entre organizações. Naturalmente, a aplicação do processo proposto torna-se mais eficiente com uma base de conhecimento mais abrangente e com um histórico de ocorrências que permite ao perito avaliar com maior precisão os riscos envolvidos em determinado cenário.

Referências bibliográficas

BEER, R., STANDER, A. & BELLE, J. V. (2014). Anti-Forensic Tool Use and Their Impact on Digital Forensic Investigations : A South African Perspective. Department of Information Systems. University of Cape Town Private. Conference Paper.

CONLAN, K., BAGGILI, I. e BREITINGER, F. (2016). Anti-forensics: Furthering digital forensic science through a new extended, granular taxonomy. Digital Investigation, 18(December 2015), S66–S75

HARRIS, R. (2006). Arriving at an Anti-Forensics Consensus: Examining How to Define and Control the AntiForensics Problem. Digital Investigation, 3(S), S44-S49. Disponível em http://dfrws.org/2006/proceedings/6-Harris.pdf.

ISO/IEC 31010. (2012). ABNT NBR ISO/IEC 31010:2012. Gestão de riscos — Técnicas para o processo de avaliação de riscos. ABNT - Associação Brasileira de Normas Técnicas. 04/04/2012.

JONES, A. (2002). Identification of a Method for the Calculation of Threat in na Information Environment, 44(abril).

NIST. (2002). Risk Management Guide for Information Technology Systems. Recommendations of the National Institute of Standards and Technology (NIST). Special Publication 800-53. Department of Commerce.

OWASP.(2015). Application Threat Modeling. Open Web Application Security Project. Disponível em https://www.owasp.org/index.php/Application_Threat_Modeling.

SACHOWSKI, J. (2016). Implementing Digital Forensic Readiness: From Reactive to Proactive Process. Syngress.

SHOSTACK, A. (2014). Threat Modeling: Designing for Security. John Wiley & Sons, Inc., Indianapolis, Indiana.

SINDRE, G. e OPDAHL, A. L. (2005). Requirements Eng. Eliciting security requirements with misuse cases. Disponível em https://link.springer.com/article/10.1007/s00766-004-0194-4

SULE, D. (2014). Importance of Forensic Readiness. ISACA Journal. Disponível em http://www.isaca.org/Journal/archives/2014/Volume-1/Pages/JOnline-Importance-of-Forensic-Readiness.aspx.

VIDALIS, S. e JONES, A. (2005). Analyzing Threat Agents and Their Attributes. Disponível em https://www.researchgate.net/publication/220947230_Analyzing_Threat_Agents_and_Their_Attributes.

Técnicas baseadas em Grafos para Priorização de Investigações Policiais de Fraudes Bancárias Eletrônicas

Álex Moreira do Patrocínio[1], **Zenilton K .G. Patrocínio Jr.**[2]

[1]Universidade de Brasília (UnB)
Brasília, DF, Brasil

[2]Pontifícia Universidade Católica de Minas Gerais (PUCMinas)
Belo Horizonte, MG, Brasil

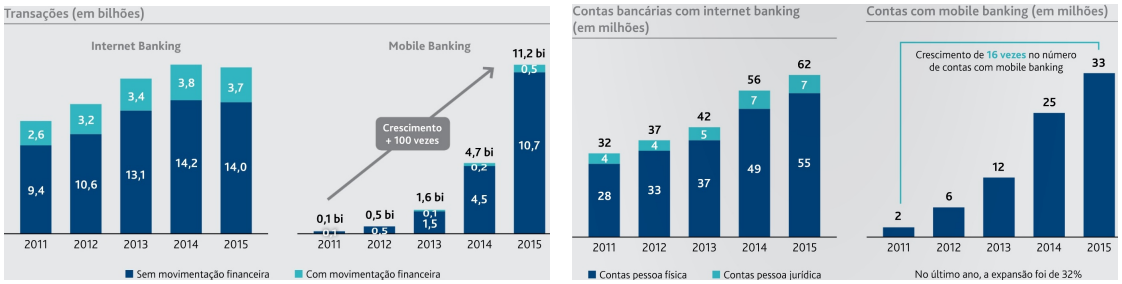
apf.alex.moreira@gmail.com, zenilton@pucminas.br

Abstract. *The project of Federal Police of Brazil named* ***Tentáculos*** *(Tentacles) gathers on a single database all the crime notifications related to bank fraud practices through Internet and mobile phones and also card cloning from clients of the brazilian bank Caixa Econômica Federal – CAIXA. Since 2008, Project* ***Tentáculos*** *has used link analysis between the entities involved in frauds, seeking to reduce the number of police investigations, avoiding investigating the same bank fraud in more than one location or police station, mapping the criminal groups and selecting the best place for the police action. But, the method adopted presently needs human interference (police investigator) to delimit the parameters (metrics) of research in police investigations. This work proposes a graph-based approach – named* ***Kraken*** *– to help prioritizing the investigations of bank frauds, in order to reduce human interference (police investigator), and, consequently, further accelerating the investigations of Project* ***Tentáculos****.*

Resumo. *O Projeto* ***Tentáculos*** *da Polícia Federal do Brasil centraliza em um banco de dados todas as notícias-crime das práticas de fraudes bancárias nas modalidades de Internet banking, mobile banking e clonagem de cartão bancário da instituição bancária Caixa Econômica Federal – CAIXA. Desde 2008, o Projeto* ***Tentáculos*** *utiliza-se da análise de vínculos entre as entidades envolvidas nas fraudes para reduzir o número de inquéritos policiais instaurados, evitando que a mesma fraude bancária seja investigada em mais de uma localidade ou delegacia, mapeando os grupos de criminosos e selecionando o melhor local para a atuação policial. Contudo, o método adotado atualmente necessita da interferência humana (investigador policial), para delimitar os parâmetros (métricas) das pesquisas nas investigações policiais. Este trabalho propõe uma abordagem baseada em grafos – denominada* ***Kraken*** *– que auxilie na priorização das investigações de fraudes bancárias, a fim de diminuir a necessidade de interferência humana (investigador policial) e, consequentemente, acelerando ainda mais as investigações no Projeto* ***Tentáculos****.*

1. Introdução

Hoje em dia, muitas das transações bancárias são realizadas através do *Internet banking*, *mobile banking* ou por meio de cartões bancários na modalidade débito ou crédito, o que faz aumentar o número de fraudes eletrônicas em desfavor das instituições financeiras. A



(a) Transações Bancárias (b) Número de Contas Bancárias

Figura 1. Volume de Transações Bancárias e Número de Contas no Brasil nas modalidades *Internet banking* e *mobile banking*. [Mompean 2016]

fim de diminuir o número de procedimentos policiais para apurar estes crimes e centralizar as informações das fraudes bancárias em desfavor da Caixa Econômica Federal – **CAIXA**, a Polícia Federal do Brasil – **PF** idealizou no ano de 2007 o Projeto **Tentáculos**.

Em 2008, o Projeto **Tentáculos** foi implementado em virtude da assinatura do termo de Cooperação Técnica entre a PF e a instituição bancária CAIXA, uma vez ser atribuição da PF investigar crimes contra a CAIXA. Hoje com o **Tentáculos** as notícias-crime são encaminhadas diretamente do órgão central da CAIXA para a PF, que as insere constantemente em um repositório único de dados centralizado, a chamada Base Nacional de Fraudes Bancárias e Eletrônicas – **BNFBE**. Segundo [Siqueira 2014], essa base de dados utiliza ferramentas de análise de vínculos, com modelagem própria, que busca identificar relações existentes entre fraudes bancárias do mesmo tipo, a fim de gerar um procedimento policial investigatório único, que reúna todas as vítimas de uma mesmo criminoso ou organização criminosa, evitando-se assim o retrabalho, resultando em um melhor aproveitamento dos recursos humanos e aprimorando a inteligência policial no combate às fraudes bancárias eletrônicas.

Porém desde sua implementação em 2008, o **Tentáculos** tem absorvido um aumento expressivo das fraudes bancárias eletrônicas. Segundo [Mompean 2016], somente as transações via Internet na modalidade *mobile banking* cresceram 138% entre os anos de 2014 a 2015, e mais de 100 vezes do ano de 2011 até 2015. Assim, verifica-se a necessidade imediata de se repensar as intervenções manuais dos investigadores policiais federais, que utilizam do modelo de análise de vínculos do Projeto **Tentáculos**, a fim de suportar a demanda crescente de combate as fraudes bancárias eletrônicas que estão por vir nos próximos anos. A Figura1(a) ilustra o crescimento das transações realizadas nas modalidades *Internet banking* e *mobile banking* no Brasil entre os anos de 2011 e 2015 [Mompean 2016]. Além disso, com o aumento das fraudes bancárias eletrônicas cresce em paralelo o mercado clandestino de dados, informações e artefatos maliciosos. Esse comércio é bastante intenso, organizado, interconectado e em franco processo de amadurecimento [Cavallaro 2014]. A abordagem proposta pelo **Kraken**, visa automatizar e auxiliar a PF na tomada de decisão referente a quais das investigações contidas na **BNFBE** devem ser priorizadas na investigação policial, devido à uma série de fatores, dentre eles:

- Escassez de recursos humanos;
- Na metodologia atual do **Tentáculos** o início da análise dos vínculos das fraudes

eletrônicas baseia-se em parâmetros de pesquisa que são inseridos manualmente por um investigador policial, o que demanda tempo, impossibilitando em um prazo aceitável, que o Policial Federal verifique todos os registros fraudulentos encaminhados pela CAIXA. Nesta metodologia a quantidade de vítimas, os montantes envolvidos nas fraudes e a localidade dos beneficiários somente são conhecidos no final da investigação, após várias intervenções manuais dos policiais federais;

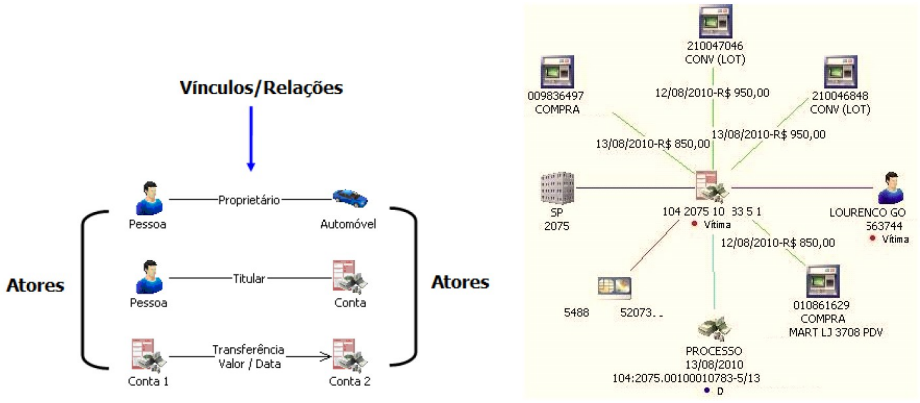
- A expressiva diminuição do tempo da elaboração do relatório de investigação policial, uma vez que com a abordagem proposta pelo **Kraken**, o policial poderá escolher previamente, qual grupo criminoso irá direcionar sua investigação, já com a certeza do resultado que irá alcançar baseado em métricas conhecidas e já preprocessadas pelo **Kraken**, como é demonstrado no item 3 deste artigo.
- Segundo a Fundação Getúlio Vargas – FGV, o crescimento do uso de computadores no Brasil dobra a cada 4 anos [FUNDAÇÃO GETÚLIO VARGAS 2016], aumentando a demanda de investigações policiais de fraudes bancárias eletrônicas;
- Por fim, segundo [Mompean 2016], o crescimento de 16 vezes do número de contas de *mobile banking* no período de 2011 a 2015, acarreta no aumento de fraudes bancárias nesta modalidade. A Figura 1(b) mostra o número de contas bancárias acessadas via *Internet banking* e *mobile banking* no Brasil.

Neste trabalho é proposta uma abordagem baseada em grafos – denominada **Kraken** – na qual as entidades e vínculos do **Tentáculos** são descritos através de grafos. A identificação dos componentes conexos é feita objetivando o correlacionamento automatizado das fraudes bancárias pertencentes a um criminoso ou grupo criminoso. Estes, por sua vez, podem ser filtrados por parâmetros como montante fraudado, período de datas, localidade, entre outros. O resultado dessa filtragem sofrerá uma análise de grupamento, de forma a produzir um roteiro que auxiliará os investigadores na geração de relatórios de análise das fraudes bancárias da **BNFBE** do Projeto **Tentáculos**.

O restante deste artigo está organizado da seguinte forma. A Seção 2 descreve o Projeto **Tentáculos**. Em seguida, o **Kraken** é apresentada na Seção 3. A Seção 4 relata e avalia os resultados obtidos pelos experimentos sobre a **BNFBE**. Por fim, a Seção 5 expõe as considerações finais deste trabalho.

2. Projeto Tentáculos

Uma vez constatada uma fraude bancária contra um correntista, a CAIXA ressarcirá os valores referentes às transações e a instituição financeira absorverá este prejuízo. Neste caso, é obrigatório que a CAIXA abra um processo de contestação, com todas as informações colhidas pelo seu setor de segurança bancária, referentes à fraude praticada contra aquela conta. Toda a informação relativa à esse tipo de fraude é consistida e organizada na **BNFBE** do Projeto **Tentáculos**. O objetivo desta análise é diminuir o número de procedimentos investigatórios (anteriormente uma vítima = um inquérito policial – IPL) e, assim, otimizar recursos humanos e materiais nas investigações. A redução do retrabalho dos policiais é possível, uma vez que uma quadrilha de fraudadores pode vitimar inúmeras contas bancárias ao longo do tempo e do território brasileiro. O Projeto **Tentáculos** usa ferramentas como o *Ibase* e *Analyst's Notebook* para o armazenamento, organização, controle e análise gráfica dos dados encaminhados pela CAIXA. Com essas ferramentas, a PF realiza de forma manual a investigação das fraudes. A modelagem e análise do Projeto **Tentáculos** baseiam-se nos conceitos de entidades e vínculos. As entidades são os atores



(a) Vínculos vs Atores (b) Representação no Grafo

Figura 2. Representação de vínculos diretos no Tentáculos. [Siqueira et al. 2014]

que estão sendo representados, tais como: processo-banco, conta, agência, cartão, entre outros. Já os vínculos representam as relações entre estas entidades, tais como: transferências bancárias, saques, pagamentos, data/hora de acesso, entre outros. Estabelecer vínculos pressupõe associar dados, condutas, eventos, entidades ou quaisquer outros elementos de um empreendimento criminal complexo, subsidiando a ação policial no sentido de permitir uma visão esclarecedora de um determinado comportamento ou ação delitiva [Dantas et al. 2007].

Segundo [Siqueira 2014], os tipos de vínculos numa abordagem investigativa podem ser diretos ou indiretos. Os vínculos diretos independem de complemento, como: transferência entre contas, registros de logs no servidor da instituição bancária, dispositivo (smartphone, tablet, computador, etc) utilizado na conexão com a conta vítima, conexão de internet (IP), pagamentos (transações bancárias) e recargas de celulares. A Figura 2(b) representa graficamente os vínculos diretos no **Tentáculos**. Já vínculos indiretos necessitam de outra técnica investigativa para serem comprovados, como: história-cobertura, infiltração, interrogatório, busca e apreensão, etc. Já os vínculos indiretos geralmente demonstram a Análise de Densidade de Ocorrência de Fenômenos (AD) ou *density analysis*, ou mancha criminal. São vínculos indiretos: localidade das agências das contas beneficiadas e os endereços das pessoas beneficiadas. Aplica-se AD no mapeamento criminal tendo como finalidade mostrar a densidade ou concentração de fatos, parte de um fenômeno, em um determinado espaço e tempo [Dantas et al. 2007].

3. Método Kraken

Existem diversos problemas da computação nos quais podemos lançar mão dos grafos para a sua solução. Segundo [West 2000], os grafos auxiliam na representação e na manipulação de conexões entre pares de objetos. A abordagem proposta neste trabalho – denominada **Kraken** – é baseada em grafos. Nessa abordagem, as entidades investigadas no **Tentáculos** (contas bancárias vitimas e beneficiadas, telefones que acessaram contas vitimas, pessoas beneficiadas, etc) são representadas pelos vértices dos grafos, enquanto os seus vínculos ou relacionamentos (transferências bancárias entre contas, ligações telefônicas para contas vitimas, titulares de contas beneficiadas, etc) existentes entre as entidades são representados no **Kraken** como sendo as arestas que ligam os vértices dos

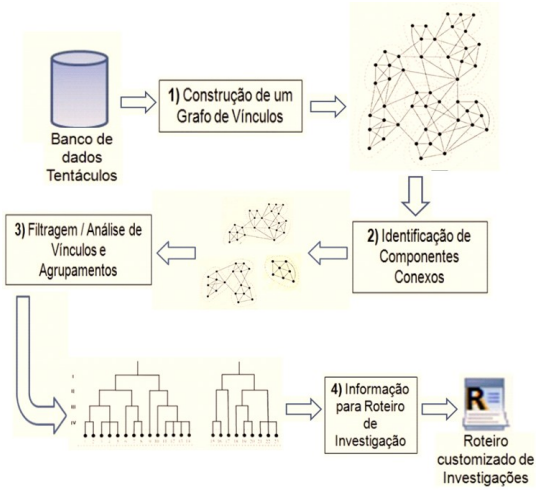


Figura 3. Representação gráfica da abordagem proposta – Kraken.

grafos. O resultado desta representação passa por uma análise de grupamento, onde são separados grafos conexos utiliza-se de um algoritmo de busca em profundidade – *Depth-First Searh* para tanto. Para o **Kraken** grafos conexos representam grupos criminosos. O **Kraken** possui uma interface gráfica – **IG** para apresentação dos seus resultados, que permite a PF, filtrar a **BNFE** por transações que ocorreram num período de datas. Ao final do processamento a **IG** exibe em forma de tabelas o resumo de quantidades de vítimas, de beneficiários, valor total das fraudes, estados do Brasil envolvidos para cada grafo conexo (entre outras informações), permitindo ao investigador realizar filtros ou ordenar os grafos baseados nas métricas acima, direcionando assim a sua investigação policial. Com base nos possíveis caminhos entre os vértices, o **Kraken** gera um roteiro com o fluxo do desvio do dinheiro desde sua origem até o destino final e comandos em *Structured Query Language* – SQL. Ambos auxiliarão a PF na geração dos relatórios de análise e diagramas de elos das fraudes bancárias existentes no **Tentáculos**, junto as ferramentas do *Ibase* e do *Analyst's Notebook* da IBM. A Figura 3 ilustra a abordagem proposta.

3.1. Construção de um Grafo de Vínculos

O grafo de vínculos $G = (V, E)$ em que V representa o conjunto de vértices (ou entidades) e E representa as arestas ou vínculos entre eles. O grafo é representado internamente no **Kraken** por uma lista de adjacência de cada vértice utilizando uma representação compacta (*compact forward and reverse star representation*) gerada a partir das tabelas existentes na **BNFBE** do **Tentáculos** e que por se tratar de um banco de dados relacional permite determinar o relacionamento (vínculos) entre os diversos tipos de entidades (Telefones, contas vítimas, contas beneficiárias, etc) existentes em suas tabelas. Com este modelo pode-se atribuir pesos p_e as arestas $e \in E$, tais como o montante da fraude. Além de ser possível recuperar informações sobre período de datas, a localidade, etc. Dessa forma, por exemplo, os caminhos entre vértices do grafo podem representar o fluxo do dinheiro das contas vítimas para as contas beneficiárias e pagamentos fraudulentos. De uma maneira simples pode-se definir o **BNFBE** como uma coleção de vários conjuntos (possivelmente disjuntos) envolvendo contas vítimas, contas beneficiárias, pessoas, localidades, transações, pagamentos, IPs ou telefones que acessaram contas bancárias, ATMs, etc. Estes conjuntos passam então a ser representados na abordagem proposta – **Kraken**


```
procedure TMain_Form.visita(k:integer);
var
 arc, arc1 : integer;
begin
   agora  := agora + 1;
   v[k]  := agora;
   for arc := No_ORIG[k] to (No_ORIG[k+1] - 1) do
   begin
      if (v[DESTINO[arc]] = 0) then
        visita(DESTINO[arc]);
   end;
   for arc := No_DEST[k] to (No_DEST[k+1]-1) do
   begin
      arc1 := IND[arc];
      if (v[ORIGEM[arc1]] = 0) then
        visita(ORIGEM[arc1]);
   end;
end;
```

(a) Seleciona 1o. Vértice do Grafo

```
agora := 0;
for k  := 1 to no_vertice do
begin
   if (v[k] = 0) then
   begin
     visita(k);
     v[k] := -v[k];  // Marco o Início do grafo
   end;
end;
```

(b) Sequência de Visitas no Grafo

Figura 4. Implementação em Pascal do algoritmo de busca em profundidade

através de um grande grafo (formado, eventualmente, por vários subgrafos conexos).

3.2. Identificação de Componentes Conexos

Um componente conexo é um subgrafo maximal conexo de um grafo. Cada vértice do grafo original pertence a exatamente um componente conexo, e o mesmo é valido para suas arestas. A identificação de componentes conexos do grafo de vínculos G permite o levantamento de forma automática dos grupos de criminosos. Além disso, segundo [Zhang and Yang 2008], agrupar dados representados sob forma de um grafo se resume a particionar o mesmo em subgrafos, de modo que cada subgrafo consistindo de vértices interconectados é referido como um grupo. Dessa forma, os dados armazenados no banco de dados do Projeto **Tentáculos** passam a ser conjuntos de vértices nos subgrafos. Assim, a abordagem proposta **Kraken** gera diversos subgrafos, de modo que cada um represente uma possível investigação criminal. Segundo [Siqueira 2014], o modelo de dados é estruturado com base nos vínculos diretos e indiretos. Portanto, a utilização de tais vínculos é importante no desenvolvimento de rotinas automatizadas de análise, em que os vínculos, principalmente os diretos, podem ter muitos níveis de profundidade. Faz-se então a necessidade da implementação de um algoritmo de **busca em profundidade** – *Depth-First Searh* para se automatizar a análise de vínculos diretos na **BNFBE**. Segundo [Szwarcfiter 1986], uma busca é dita em profundidade quando o critério de escolha de vértice marcado (a partir do qual será realizada a próxima exploração de aresta) obedecer ao princípio: *dentre todos os vértices marcados e incidentes a alguma aresta ainda não explorada, escolher aquele mais recentemente alcançado na busca*. Esse critério garante que a escolha de vértice torna-se única e sem margem de dúvidas. A Figura 4 apresenta a implementação em *ObjectPascal* do algoritmo para busca em profundidade.

3.3. Filtragem e Análise dos Vínculos e dos Agrupamentos

Os vértices de uma árvore de um subgrafo possuem geralmente vizinhos próximos (sobretudo no caso de fraudes bancárias na modalidade cartão bancário clonado) pois possuem ligações (arestas) fortes de vínculos diretos e até mesmo por que a fraude com cartões clonado acontece em uma área geográfica mais restrita. Porém, algumas análises podem indicar elos fracos entre os vértices (principalmente no caso de fraudes bancárias nas modalidades *Internet banking* e *mobile banking*), por exemplo quando a maioria das fraudes ocorreram em um determinado estado do Brasil e em apenas um caso isolado a fraude acontece outra localidade bem diferente. Esta situação pode levar a eliminação deste

vínculo e por consequência do subgrafo a ele ligado, caso este seja o único caminho de ligação entre o caso de fraude isolado com a maioria dos outros casos. Esta eliminação deve ser feita pelo investigador da PF baseado na visualização do grafo gerado pelo **Kraken**Contudo, apenas a análise de vizinhança local dos vértices pode não ser suficiente, devendo-se investigar a relação entre vértices distantes e, se verificado um elo forte, estes devem integrar o subgrafo de investigação. Assim a análise de vizinhança, não deve se restringir a uma abordagem local (vértices próximos). Após o processamento geral feito pelo **Kraken** na **BNFBE**, o investigador pode utilizar-se dos filtros da sua **IG** para analisar os vínculos (arestas dos grafos) de forma objetiva, visando o interesse da investigação. Filtros como: contas de uma mesma localidade, pagamentos realizados por uma mesma pessoa ou quadrilha, grupos de vítimas, grupos de beneficiários, entre outros podem ser feitos através da **IG** do **Kraken** pois o mesmo armazena o tipo de entidade relacionada a cada vértice do grafo.

3.4. Informação para Roteiro de Investigação

Após a etapa de filtragem e análise dos vínculos e dos agrupamentos, o método **Kraken** disponibiliza as informações necessárias para o investigador elaborar os relatórios de análise das fraudes bancárias da **BNFBE** do **Tentáculos**. Tais informações constituem um roteiro com as transações passo à passo com a cronologia dos fatos (transações) de como o dinheiro foi desviado da vítima até chegar no destinatário final. As informações desse roteiro são exportadas em formato *Excel*, e são aproveitadas como uma especificação de importação de dados da ferramentas *Analyst's Notebook*, gerando automaticamente o diagrama de elos da investigação policial em curso.

4. Experimentos e Análise de Resultados

A abordagem proposta **Kraken** foi implementada em Delphi versão 2010 para o sistema operacional Windows 7 professional e testada em uma máquina com processador i5-520M (Dual Core). de 2.4 GHz com memória RAM de 4 GBytes e disco de 500 GBytes. Utiliza-se o banco de dados MySql versão 5.7.12, que contém as tabelas dos dados do Projeto **Tentáculos** retiradas da BNFBE, além daquelas necessárias ao funcionamento do **Kraken**. Essa implementação focou inicialmente nas fraudes de **transferências bancárias** entre contas vítimas e beneficiárias, sendo que as demais transações como pagamento de boletos bancários, saques, compras pela internet ou com cartões bancários não foram consideradas neste trabalho. O **Kraken** é formado por dois módulos: o primeiro é o responsável pelo processamento e análise dos dados e outro pela exibição das informações (roteiros) e sua exportação em formato *Excel*. A Figura 5 exibe a tela da implementação feita para o método **Kraken**. Nessa tela o investigador pode lançar mão, caso deseje, de filtros por período de datas para o processamento das investigações. Após o processamento da BNFBE, a parte superior da tela exibe o subgrafos (ou investigações possíveis) levantadas pelo **Kraken**. Para cada uma delas se encontram informações sumarizadas sobre o número de vítimas e beneficiários, o valor total fraudado em reais, os estados da federação em que se encontram as vítimas e os beneficiários, além do período de tempo (data de início e fim) em que ocorreram as operações fraudulentas. Tais investigações podem ser ordenadas e/ou filtradas por quaisquer uma dessas informações sumarizadas descritas acima. Desse modo, o investigador pode priorizar suas ações, de acordo com os critérios que julgar mais adequados. Ainda na parte superior da tela o investigador

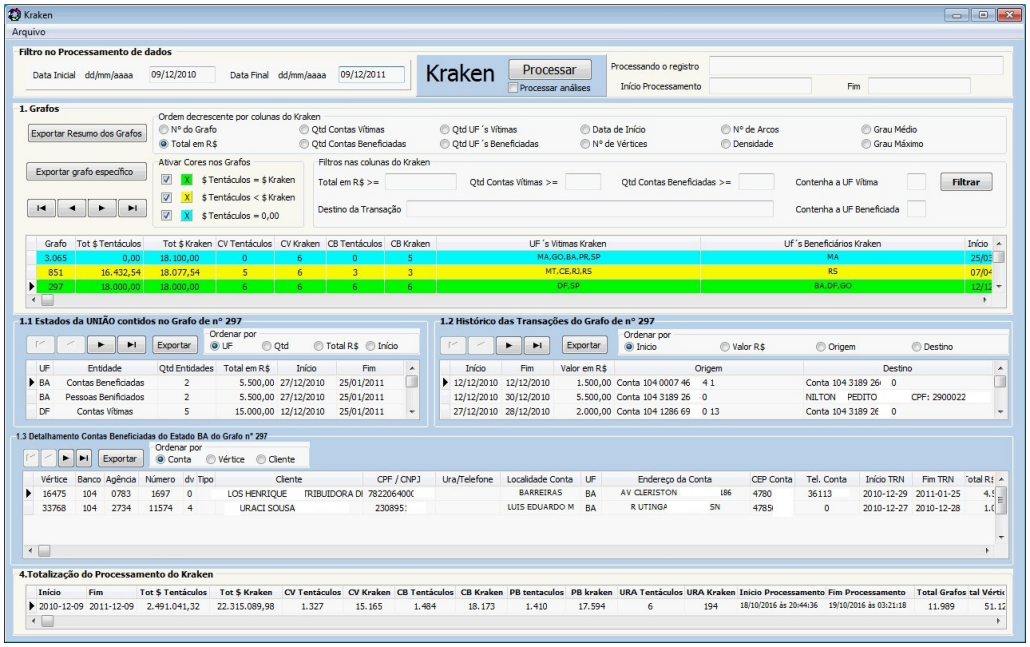


Figura 5. Tela da implementação do Kraken.

pode realizar comparações entre os grafos gerados pelo **Kraken** versus os grafos que já foram investigados pelo **Tentáculos** no que diz respeito a transferências bancárias. As comparações são baseadas no valor total fraudado de cada grafo, neste caso a cor verde sobre a linha de resumo do grafo mostra que o valor encontrado pelo **Kraken** é igual ao do **Tentáculos**, a cor amarela indica que o **Kraken** achou um valor de fraude maior que o **Tentáculos** demonstrando que a pesquisa em profundidade do **Kraken** localizou um número maior de vínculos e/ou vértices naquele grafo em específico em relação às pesquisas humanas feitas por Policiais Federais. Já a cor azul claro indica que o **Kraken** gerou um grafo que ainda não foi investigado pelo **Tentáculos**, como o grafo de número 3.065 na Figura 5. Já na parte central a esquerda, é exibido um detalhamento por estado do Brasil para uma dado grafo selecionado na parte superior. Dessa forma, é possível visualizar a quantidade de contas vítimas e beneficiárias e de pessoas beneficiárias por estado juntamente com os respectivos montantes fraudados (valores em reais) e períodos de tempo de ocorrência. Enquanto que na parte inferior, são exibidas informações detalhadas da entidade (telefone/URA, conta vítima, conta beneficiária e pessoa beneficiada) que tenha sido selecionada na parte central a esquerda. Por fim, na parte central a direita da tela, encontra-se uma descrição detalhada do histórico das transações fraudulentas do grafo selecionado na parte superior. Tais informações podem ser ordenadas por data, valor fraudado, origem e destino da operação. O ordem cronológica permite ao investigador descrever os fatos e transações fraudulentas na medida em que ocorreram fornecendo uma descrição da atuação como um todo do grupo criminoso, já a ordenação por destino permite que o investigador foque sua descrição em determinadas contas e/ou pessoas beneficiárias permitindo qualificar a participação das mesmas na fraude. Já a ordenação por valor auxilia o investigador em priorizar as operações de maior volume monetário e, provavelmente, relacionadas aos membros mais influentes do grupo investigado. A Figura 6 ilustra o resultado das ferramentas de análise do Projeto **Tentáculos** aplicado a um subgrafo obtido pela implementação do **Kraken**. De forma a avaliar o

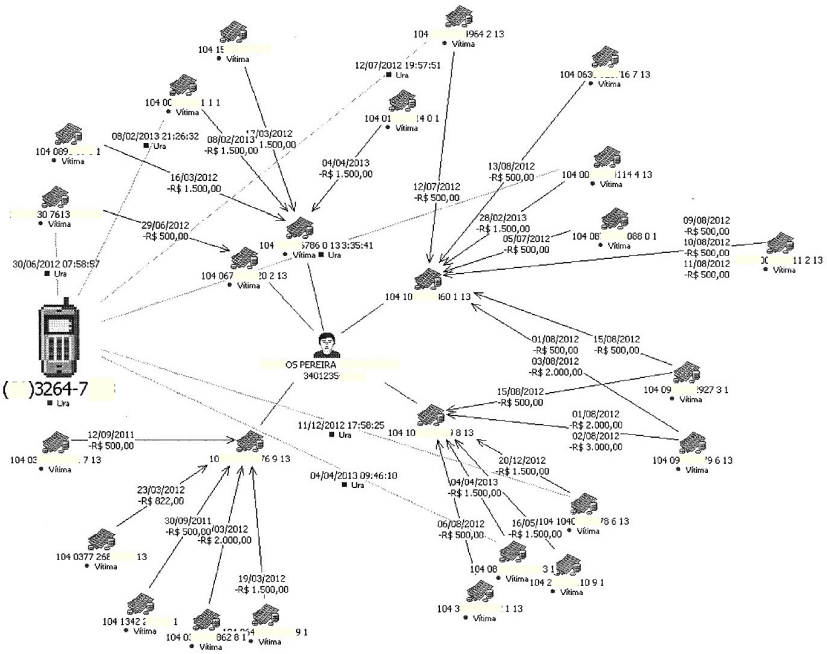


Figura 6. Exemplo de Rede de Entidades do Projeto Tentáculos.

método **Kraken**, foram carregados, processados e analisados diferentes volumes de dados extraídos da **BNFBE** do Projeto **Tentáculos** para períodos de 01, 02 e 03 meses. Os resultados encontram-se sumarizados na Tabela 1. Só para se ter uma ideia do potencial que o método **Kraken** representa em termos de auxílio aos investigadores, para o período de um mês, o número de contas analisadas pela PF foi de 649 contas vítimas (contra 2.059 feitas pelo **Kraken**), que representou um valor total de R$ 877.693,02 (contra R$ 3.038.467,57 obtidos pelo **Kraken**).

5. Conclusão

Este trabalho propõe uma abordagem – denominada **Kraken** – na qual as entidades e vínculos do Projeto **Tentáculos** vão ser descritos através de grafos. A aplicação de técnicas baseadas em grafos para priorização de Investigações Policiais de Fraudes Bancárias Eletrônicas traz um novo paradigma na elucidação de crimes contra instituições financeiras. Os experimentos realizados pelo **Kraken** demonstram nitidamente a velocidade e as vantagens de se processar, previamente, uma massa enorme de dados contidos na **BNFBE**, baseado em métricas conhecidas capazes de indicar com clareza quais as investigações devem ser priorizadas. Com o experimento fica claro que a complexidade da investigação aumenta de maneira absurda ao se aumentar o período de datas das investigações o que dificulta cada vez mais a análise humana. Deve-se ter em mente que a fraude bancária eletrônica, no seu modus operandi pode se arrastar por meses, ou anos,

Tabela 1. Resultados do Processamento do Método Kraken.

Período (meses)	Núm. de Grafos	Total Fraudado (Reais)	Núm. Total Vítimas	Núm. Total Beneficiários	Tempo processamento
01	1.594	3.038.467,59	2.059	2.786	1h43m
02	3.324	6.221.564,58	4.351	5.610	2h12m
03	5.502	10.095.944,34	7.103	9.052	3h50m

assim, para um efetivo combate as quadrilhas de criminosos que fazem uso deste tipo de ilícito, considera-se importante a janela temporal da investigação, a fim de atribuir o máximo possível de crimes para a mesma quadrilha de criminosos (grafos conexos), diminuindo assim o número de procedimentos investigativos instaurados pela PF e elucidando o maior número possível de contestações bancárias de uma só vez. A análise na totalidade dessa gigantesca **BNFBE**, pelo ser humano, torna-se improdutivo e aumenta o lapso temporal entre a ocorrência da fraude informada pela CAIXA e a análise por parte do policial federal. A aplicação do modelo **Kraken** reduz de forma significativa a intervenção humana na garimpagem dos dados iniciais da investigação, restando ao investigador apenas selecionar qual das investigações já processadas pelo **Kraken** devem ser priorizadas.

Trabalhos futuros devem considerar além das transferências bancárias entre contas vítimas e beneficiárias, outros vínculos diretos como pagamentos de boletos, saques em ATM, compras com cartão e através da internet, transferência de valores para recarregar planos de celulares, entre outros, além dos vínculos indiretos entre as transações fraudulentas, como: endereço dos beneficiários e contas destinatárias visando assim aglomerar ao máximo grafos ora desconexos em uma única investigação, o que resultará em um montante maior de vítimas, beneficiários e dinheiro envolvido em uma única investigação.

Referências

Cavallaro, L. (2014). Malicious software and its underground economy two sides to every story. Curso online. Disponível em: *https://pt.coursera.org/learn/malsoftware*.

Dantas, G. F. D. L., Bair, S., Felipe, A., and Magalhães, L. C. (2007). Análise criminal: Novas tendências em proveito da análise criminal avançada e da inteligência de segurança pública. FENEME – Federação Nacional de Entidades de Oficiais Militares Estaduais. Disponível em: *http://www.feneme.org.br/pagina/810/analise-criminalnovas-tendencias-em-proveito-da-analise-criminal-avancada-e-da-inteligencia-de-seguranca-publica*.

FUNDAÇÃO GETÚLIO VARGAS (2016). 27ª pesquisa anual de uso de TI. Pesquisa anual realizada pelo Centro de Tecnologia de Informação Aplicada da FGV-EAESP, Disponível em: *http://eaesp.fgvsp.br/sites/eaesp.fgvsp.br/files/pesti2016gvciappt.pdf*.

Mompean, A. (2016). Transações com mobile banking crescem 138% em um ano. *Revista Ciab FEBRABAN*, pages 16–23. Disponível em: *https://issuu.com/revistaciab/docs/revista_ciab_63_jun16/1*.

Siqueira, E. P. (2014). O Projeto Tentáculos da Polícia Federal: Da concepção à proposta de modelo aplicável na segurança pública brasileira. Monografia de Especialização. Universidade de Brasília.

Siqueira, E. P. et al. (2014). Caderno Didático: Crimes Cibernéticos. Polícia Federal. ANP – Academia Nacional de Polícia.

Szwarcfiter, J. (1986). *Grafos e algoritmos computacionais*. Campus.

West, D. B. (2000). *Introduction to Graph Theory*. Prentice Hall, 2 edition.

Zhang, X. X. and Yang, Y. M. (2008). Minimum spanning tree and color image segmentation. In *Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on*, pages 900–904.

Análises de linhas temporais contextuais em investigações digitais

Regis Levino de Oliveira[1], Bruno Werneck P. Hoelz[2*]

[1]Departamento de Engenharia Elétrica - Universidade de Brasília
[2]Instituto Nacional de Criminalística – Polícia Federal
Brasília, DF – Brasil

regislevino@yahoo.com.br, werneck.bwph@dpf.gov.br

Abstract. *Timeline analysis is a well-known technique in digital forensics. However, most studies and tools focus on the extraction of timestamps with less emphasis on how to visualize and analyze large volumes of such data. This work proposes a process to generate contextual timelines, where each timestamp is also associated with other four dimensions: place, person, subject and event. A clustering algorithm is then used to generate timelines that contain similar data, and are easier to visualize and interpret.*

Resumo. *A análise de linhas temporais é uma técnica bastante empregada em exames periciais em ambientes computacionais. No entanto, a maioria desses estudos concentra-se nos desafios da extração de registros temporais com menos ênfase em como visualizar e analisar um grande volume desses dados. Este trabalho propõe um processo para gerar linhas temporais contextuais, onde cada rótulo temporal é associado a outras quatro dimensões: local, pessoa, assunto e evento. Um algoritmo de clusterização é então utilizado para gerar linhas temporais com dados similares, que são mais fáceis de visualizar e interpretar.*

1. Introdução

Vestígios digitais podem ser a chave para a elucidação de um fato criminoso, como a existência de uma foto (ex.: comprovante de depósito), a troca de mensagens por e-mail ou aplicativos de mensageria ou o registro de ações realizadas em um sistema computacional (como *logs* de acesso). Para auxiliar na compreensão e análise dessa grande diversidade de vestígios, vários recursos de visualização são empregados, dentre eles o da visualização de linhas temporais. A partir da construção de uma linha do tempo, fica mais fácil analisar eventos em torno de pontos de interesse, como o momento em que um servidor de dados foi acessado indevidamente.

Muitas ferramentas estão disponíveis para a geração e análise de linhas temporais. Ferramentas como Zeitline (Buchholz e Falk, 2005), CyberForensics TimeLab (Olsson e Boldt, 2009) e log2timeline (Guðjónsson, 2010) lidam com o desafio de coletar e apresentar dados temporais de várias fontes. No entanto, uma das maiores dificuldades para a análise das linhas temporais é a grande quantidade de dados, o que requer outras abordagens. Nesse sentido, Chabot et al. (2014) propõem um modelo para a reconstrução de eventos que inclui definições formais das entidades envolvidas em um incidente e

* Os autores agradecem o apoio da Secretaria Nacional de Segurança Pública (SENASP), da Diretoria Técnico-Científica da Polícia Federal e da FINEP (Convênio 01.12.0433.01, Projeto: Defesa Nacional e Segurança Pública) na realização deste trabalho.

operadores que permitem que o conhecimento contido no modelo seja extraído, manipulado e analisado. Já Yu Jin (Jin, 2013) propõe o uso de mapas auto-organizáveis para analisar a relação de registros de atividades no sistema operacional Android.

De acordo com as circunstâncias do caso, pode ser necessário visualizar os eventos temporais segundo o contexto a que estão relacionados, reduzindo o ruído associado ao grande volume de registros temporais. Para isso, este trabalho propõe uma abordagem de contextualização e agrupamento de eventos temporais. Os registros temporais passam por um processo de contextualização, no qual um registro temporal (e seu arquivo de origem) é analisado e associado a entidades previamente definidas nas dimensões pessoa, local, evento e assunto. Posteriormente, algoritmos de agrupamento (clusterização) são aplicados sobre os registros, resultando em linhas temporais contextualizadas segundo essas entidades. Tal abordagem facilita a visualização, identificação e análise de eventos relacionados, que poderiam passar desapercebidos em meio ao grande volume de dados provenientes de diversas fontes, contribuindo, portanto, para uma melhor análise das linhas temporais.

O restante desse artigo está organizado da seguinte forma: a Seção 2 descreve a abordagem proposta e trabalhos correlatos; a Seção 3 apresenta a aplicação da abordagem e os resultados obtidos e, por fim, a Seção 4 apresenta as conclusões e trabalhos futuros.

2. Linhas temporais contextuais

Linhas temporais contextuais visam apresentar os dados temporais agrupados segundo outras dimensões que não somente o tempo. A construção de linhas temporais contextuais está dividida em quatro etapas, conforme ilustra a Figura 1. A primeira etapa é a determinação das fontes de evidência das quais serão extraídos os registros temporais. Como discutido anteriormente, registros temporais podem ser provenientes de várias fontes distintas como computadores, *smartphones*, dispositivos de rede, entre outros.

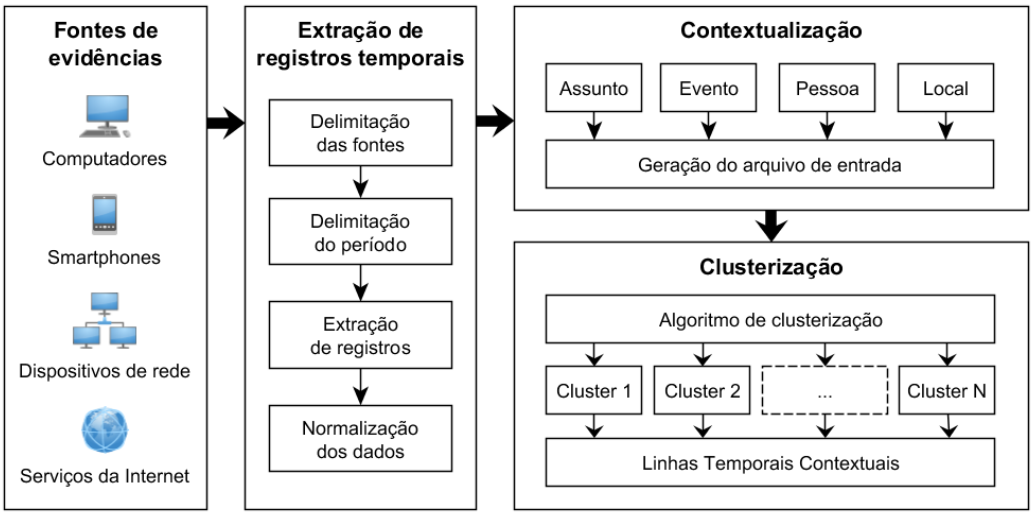


Figura 1 – Arquitetura da proposta

Em seguida, é realizada a extração dos registros temporais. Antes da extração propriamente dita, pode-se delimitar as fontes e o período de interesse, dependendo do incidente investigado. Muitos registros temporais podem não ser interessantes para determinado incidente como, por exemplo, datas de criação, acesso e modificação de

arquivos do sistema operacional. Nessa etapa, assume-se a utilização de ferramentas próprias para extrair os dados, como log2timeline (Guðjónsson, 2010). Por fim, os dados devem ser normalizados, com a uniformização dos formatos de dados gerados, em especial da forma de representação dos rótulos temporais, desvios nos relógios e zonas de tempo diferentes. Após a extração dos registros temporais de interesse, é iniciada a etapa de contextualização. Essa etapa visa vincular um registro temporal a um ou mais contextos. Cada contexto é composto por quatro dimensões: local, evento, pessoa e assunto.

A dimensão do local associa o registro temporal a um lugar no espaço. Por exemplo, o local de ocorrência de um determinado crime ou locais frequentados pelos suspeitos. Para vincular um registro temporal a um determinado local, pode-se utilizar metadados de arquivos de imagens de câmera fotográficas e telefones celulares, coordenadas de GPS extraídas de aplicativos, histórico de navegação de sites de mapas, entre outros. A dimensão evento diz respeito a eventos relevantes para a investigação, ocorridos digitalmente (ex.: troca de uma mensagem) ou no mundo real (ex.: homicídio). Um evento pode ou não ter um momento bem definido no tempo. Nesse caso, os registros temporais podem ser associados ao evento pela proximidade (ex.: logs obtidos após a detecção de uma invasão ao sistema). Registros temporais (eventos de baixo nível) também podem ser vinculados automaticamente a um evento (de alto nível), como sugerido por Hargreaves e Patterson (2012). Por exemplo, a detecção de alteração em vários arquivos do sistema, pode ser vinculado à instalação de um programa malicioso.

Em uma investigação digital, várias pessoas podem estar envolvidas. Logo, registros temporais provenientes de fontes diversas estarão associados a pessoas distintas. Um registro temporal pode ser associado a uma pessoa não só por ser o usuário principal de um determinado dispositivo, mas também pela interação por meio de registros de chamadas telefônicas, contas de e-mail e outros identificadores associados às pessoas de interesse. A dimensão assunto diz respeito a temas ou palavras-chave relacionadas à investigação. Para isso, devem ser definidos os assuntos segundo o crime investigado, onde cada assunto é composto por um conjunto de palavras-chave. Os registros temporais são então associados a um assunto, se o conteúdo do arquivo possuir uma ou mais dessas palavras. A informação sobre os termos que compõe o assunto pode vir de informações da equipe de investigação, por meio de declarações de vítimas, criminosos em busca do abrandamento da pena (delação premiada) ou por técnicas de processamento de linguagem natural para identificar o assunto a partir do conteúdo dos textos extraídos das fontes de evidências.

Após a definição dos contextos, é realizada a geração do arquivo de entrada que será submetido ao algoritmo de clusterização. Na aprendizagem de máquina, um cluster é o agrupamento de dados de certo conjunto de dados de entrada. Esses agrupamentos são realizados por meio de similaridades (dentro de um mesmo cluster) ou pelas diferenças (entidades de um cluster são diferentes de entidades de outro cluster). É desejável que os pontos em um cluster tenham uma distância pequena um do outro e que pontos em clusters diferentes sejam mais distantes. Como resultado do algoritmo, cada registro é associado a um cluster. Por fim, para cada cluster gerado, ordenam-se os registros por data e hora, obtendo-se assim uma linha temporal contextualizada. Logo, um número N de clusters gera N linhas temporais. De posse das várias linhas temporais, o perito é capaz de analisar atividades correlacionada pelos contextos definidos pelas dimensões de nome, evento, local ou assunto.

3. Aplicação experimental

O objetivo desta seção é exemplificar a aplicação da proposta em um cenário baseado em um caso real de posse e o compartilhamento de material pornográfico contendo crianças ou adolescentes. Nesse caso, o suspeito da conduta descrita, também dava aulas sempre às segundas-feiras em uma escola, não sendo descartada a hipótese de abuso sexual de alunos. Após denúncia e monitoramento do suspeito, foram descobertas outras pessoas associadas às condutas. Nesse cenário, os peritos foram acionados para a busca de evidências digitais no material apreendido na residência dos suspeitos e na escola: computadores, celulares e câmeras digitais. Para representar essas fontes de evidências, foi utilizado um conjunto de arquivos fictícios, simulando 11.500 registros temporais entre as datas de 01/10/2014 à 25/11/2014. Os formatos de data foram normalizados para um formato único (apresentado na Figura 2).

3.1. Contextualização

Para a definição do local, foram utilizadas informações colhidas de metadados EXIF de imagens de câmera fotográfica e telefone celular, vinculação manual por meio de reconhecimento visual (ex.: fotos da casa do suspeito ou da fachada da escola) e informações de localização coletadas por aplicativos. Na ausência de um local mais específico, o registro foi associado ao próprio dispositivo originário (ex.: celular) ou ao local no qual a fonte de evidências se localizava ou ser indeterminada (ex.: uma foto de abuso sem identificação clara do local).

Para a dimensão pessoa, os nomes foram reunidos baseado na extração de informações das trocas de mensagens de aplicativos de chat, nomes repassados pela investigação que seriam suspeitos dos crimes, nomes contidos nos metadados de arquivos (ex.: propriedade autor de um documento de texto).

Na definição do evento, primeiramente, foram definidos eventos relevantes no crime investigado. Por exemplo, a troca de mensagens entre suspeito e vítima. No caso em questão, foi observada uma grande quantidade de fotografias do abuso, downloads de arquivos contendo pornografia infantil e troca de mensagens entre os suspeitos. Por fim, na definição do assunto, foram utilizadas palavras-chave encontradas nos itens de pesquisa para *download* de arquivos de pornografia infantil.

3.2. Clusterização

Para a etapa de clusterização, os dados contextualizados foram inseridos na ferramenta WEKA (Bouckaert et al., 2016). Uma amostra do arquivo de entrada, no formato ARFF, é apresentado na Figura 2. Os atributos são definidos no início do arquivo. Entre eles, a fonte que originou o registro, o rótulo temporal (data) e os dimensões que definem o contexto (assunto, local, evento e nome). Os dados das dimensões são categóricos e os valores válidos são apresentados entre chaves na Figura 2. Os dados foram então submetidos ao algoritmo *Simple K-Means* utilizando a distância euclidiana para formação dos clusters. Registros não contextualizados (cujo valor faltante é representado por um ponto de interrogação) são ignorados. O parâmetro *k,* do número de clusters a serem gerados, depende da quantidade de valores em cada atributo (nome, assunto, local, evento) e da variância desses dados, o que depende do caso em concreto. Logo, o valor de *k* foi definido experimentalmente, variando seu valor até obter um conjunto de clusters satisfatório com k=5.


```
@RELATION caso_pi

@ATTRIBUTE fonte_dos_dados    string
@ATTRIBUTE data     DATE "yyyy/MM/dd HH:mm:ss"
@ATTRIBUTE assunto  {PTHC, PEDO, SEXO}
@ATTRIBUTE local    {escola, notebook1, notebook2, celular, escritorio}
@ATTRIBUTE evento   {Tirar_foto,chat_whatsapp,chat_messenger_facebook,download_arquivos}
@ATTRIBUTE nome     {Rodrigo, Carlos, Lucas, Frederico, Sade}

@DATA
Fonte_10,"2014/10/05 22:27:26",SEXO,celular,chat_messenger_facebook,Rodrigo
Fonte_3,"2014/10/10 09:19:55",?,notebook2,chat_messenger_facebook,Rodrigo
Fonte_4,"2014/10/03 10:13:47",PEDO,celular,chat_messenger_facebook,Carlos
Fonte_8,"2014/10/17 18:39:33",?,notebook1,download_arquivos,Carlos
Fonte_7,"2014/10/15 12:39:02",?,notebook2,?,?
Fonte_6,"2014/10/21 20:37:53",PEDO,notebook2,download_arquivos,Rodrigo
Fonte_10,"2014/10/23 10:31:54",PTHC,escritorio,download_arquivos,Carlos
Fonte_5,"2014/10/11 17:15:31",PTHC,escritorio,download_arquivos,Rodrigo
```

Figura 2 – Arquivo do tipo ARFF utilizado no experimento

Os clusters obtidos são apresentados na Figura 3, com a quantidade de registros em cada cluster e o contexto associado. O cluster 0, por exemplo, tem 5765 registros e o seu contexto é definido como [assunto=sexo, local=escola, evento=Tirar_foto, nome=Rodrigo].

Cluster#				
0	1	2	3	4
(5765.0)	(2163.0)	(1297.0)	(1405.0)	(869.0)
SEXO	PTHC	PEDO	PTHC	SEXO
escola	notebook1	escritorio	notebook2	celular
Tirar_foto	chat_whatsapp	chat_whatsapp	chat_messenger_facebook	chat_whatsapp
Rodrigo	Carlos	Sade	Carlos	Carlos

Figura 3 – Clusters formados após a execução do algoritmo SimpleKMeans

A Figura 4 apresenta as linhas temporais contextuais construídas a partir do clusters gerados. O eixo X apresenta a data em milissegundos (com a data correspondente inserida na figura) e cada linha do eixo Y representa um dos clusters. Cada ponto representa um registro temporal. É possível verificar alguns pontos de maior atividade como nos dias 13/10, 20/10, 27/10, 25/11. No cluster 0, esses pontos foram destacados com um círculo. Observa-se também um tempo de inatividade entre 28/10 e 24/11.

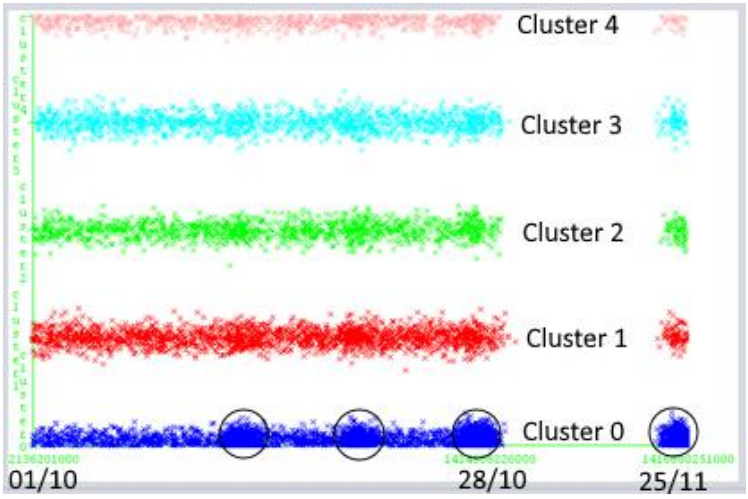


Figura 4 – Linha temporal contextual

Com a análise das linhas temporais contextuais, verificou-se grande atividade entre os interlocutores Rodrigo e Carlos, em especial associada aos eventos

"chat_whatsapp" e "chat_messenger_facebook". O suspeito Rodrigo também teve maior atividade nos dias citados anteriormente. Assim, a linha temporal proveniente do primeiro cluster (Cluster 0) facilita a visualização de evidências associadas ao abuso da vítima na escola exatamente nos dias mencionados, associados às fotos tiradas. Observou-se que as demais linhas permitem uma visualização melhor de mensagens utilizando WhatsApp e Facebook Messenger, usadas no compartilhamento de material pornográfico infantil.

4. Conclusão

A análise de linhas temporais é uma técnica muito empregada em investigações digitais. A grande quantidade de fontes de dados temporais torna essa análise mais complexa, dificultando a adequada visualização das relações entre eventos e da determinação de pontos no tempo que são de interesse da investigação. O presente trabalho demonstrou que essa análise pode ser simplificada pela contextualização e posterior agrupamento (clusterização) dos registros temporais. Como resultado do modelo proposto, obtém-se linhas temporais cujos registros apresentam maior similaridade contextual entre si, reduzindo a interferência de outros registros não relacionados. No experimento proposto, pode-se identificar com mais facilidade os suspeitos com maior interação e os momentos de maior atividade relacionados às condutas investigadas.

Em trabalhos futuros, pretende-se aprimorar o processo de agrupamento com a aplicação de outros algoritmos, a fim de comparar os resultados obtidos para cada um deles. Além disso, pretende-se explorar formas de contextualização baseadas em processamento de linguagem natural, especialmente na dimensão do assunto.

5. Referências

Bouckaert, R. R., Frank, E., Hall, M., Kirkby, R., Reutemann, P., Seewald, A., & Scuse, D. (2016). *WEKA Manual for Version 3-8-0*.

Buchholz, Florian. Falk, Courtney. (2005). *Design and Implementation of Zeitline: a Forensic Timeline Editor*. In: Digital forensic research workshop. https://users.cs.jmu.edu/buchhofp/publications/zeitline_dfrws.ps. Acessado em 10/10/2015.

Charbot, Yoan. Bertaux, Aurélie. Nicolle, Christophe. Kechadi, M-Tahar. (2014). *A complete formalized knowledge representation model for advanced digital forensics timeline analysis.* Digital Investigation 11 (2014) 95-105.

Gudhjonsson K. (2010). *Mastering the super timeline with log2timeline*. SANS Reading Room. http://www.sans.org/reading-room/whitepapers/logging/mastering-super-timeline-log2timeline-33438. Acessado em 20/10/2015.

Hargreaves, Christopher. Patterson, Jonathan. (2012). *An automated timeline reconstruction approach for digital forensic investigations*. Digital Investigation (2012) vol. 9, p. 69–79.

Jin, Yu. (2013). *Timeline analysis for Android-based systems*. Kongens Lyngby 2013 IMM-M.Sc.-2013-42.

Olsson J, Boldt M. (2009). *Computer forensic timeline visualization tool*. Digital Investigation (2009). vol. 6, p.78-87.

XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais

Niterói, RJ, novembro de 2016

WRAC+ – II Workshop de Regulação, Avaliação da Conformidade e Certificação de Segurança

Proposta de Metodologia de Homologação e Certificação de Produto de Defesa Cibernética: o caso dos equipamentos de videoconferência

Ariane C. B. Florentino, Sanderson C. M. Barbalho

Departamento de Engenharia Mecânica – Universidade de Brasília (UnB)
Campus Universitário Darcy Ribeiro – Brasília – DF – Brasil

arianecristin@hotmail.com, sandersoncesar@unb.br

Abstract. *The information security management brings procedures and processes based on international standards, such as ISO (International Standard Organization) standards, aimed at providing safety of the technology assets of an organization.This research will show the cyber defense and the videoconference system, its definition and a proposal for a homologation system and cyber defense product certification with the help of the Common Criteria.*

Resumo. *A gestão da segurança da informação traz procedimentos e processos baseados em normas internacionais, tais como a International Standard Organization ISO, que visa proporcionar segurança dos ativos de tecnologia de uma organização. Esta pesquisa irá expor a ciberdefesa e o sistema de videoconferência, a sua definição e uma proposta de sistema de homologação e certificação de produtos de defesa cibernética com o apoio do Common Criteria.*

1. Introdução

Segundo [Junior 2011], a utilização dos sistemas de informação pelas organizações tornou-se questão de sobrevivência em meados de 1990. Sejam elas públicas ou privadas, a informação é extremamente valiosa para essas entidades, considerando que é a base de suas tomadas de decisões.

A Tecnologia da Informação (TI) pode ser considerada a base para o modelo operacional de diversas entidades, além do que "A dependência frente à infraestrutura exige cada vez mais a participação dos gestores de TI no planejamento organizacional, bem como dos gestores de telecomunicações na formulação de políticas públicas" [Junior 2011].

Diversos dados sensíveis trafegam pelos sistemas de comunicações unificadas, sejam em reuniões virtuais, videoconferências ou web chats realizados entre as diferentes entidades da administração pública federal (APF). Qualquer falha de gestão de segurança pode impactar negativamente a organização, alterando sua disponibilidade de serviços, trazendo prejuízos financeiros e de imagem, e até mesmo multas e sanções administrativas aos envolvidos.

A pesquisa em tela analisa o aspecto da defesa cibernética nas homologações e certificações do produto mecatrônico "equipamento de videoconferência". Esse tipo de equipamento é muito difundido nas esfera pública e privada e objeto pelo qual transitam informações sensíveis das mais diversas categorias. O estudo parte de um modelo teórico para a homologação e certificação cibernética para esse tipo de produto e visa chegar à

realização de testes em um equipamento específico de uma empresa de capital misto de grande porte.

As próximas seções deste artigo estão dispostas da seguinte forma: a 2ª seção aborda a revisão teórica, com conceitos de produto mecatrônico e videoconferência; enquanto que a seção 3 apresenta a metodologia usada. Na 4ª seção são tratados os resultados e as discussões, e na 5ª seção seguem as considerações finais.

2. Revisão Teórica

A cibernética compreende a utilização de distintos meios tecnológicos, envolvendo sistemas de informação e comunicações. O setor cibernético é considerado pelo governo brasileiro como de importância estratégica – ao lado dos setores nuclear e espacial.

Após o governo brasileiro estabelecer, em dezembro de 2008, o setor cibernético como de importância estratégica – ao lado dos setores nuclear e espacial - o Exército Brasileiro instituiu no ano seguinte o Setor Cibernético no âmbito da Força Terrestre sob determinação do Ministério da Defesa. Desde então, o Projeto Estratégico de Defesa Cibernética vem consolidando-se devido à necessidade da existência de uma entidade para liderar os intervenientes nessa empreitada [Brasil 2015a]. Com o objetivo de desenvolver medidas de proteção e mitigar ataques no campo cibernético, o Projeto Estratégico de Defesa Cibernética, estabelecido em 2009 pelo Exército Brasileiro, é composto por várias ações em nível operacional e estratégico [Brasil 2015a] dentre elas a criação do Comando de Defesa Cibernética (ComDCiber).

Considerando o risco à soberania do país, o ComDCiber possui em sua estrutura dois elementos importantes: a Escola Nacional de Defesa Cibernética (ENaDCiber) e o Sistema de Homologação e Certificação de Produtos e Serviços de Defesa Cibernética (SHCDCiber). A primeira visa capacitar recursos humanos, permitindo dominar áreas multidisciplinares e implantar pesquisa científica voltada ao tema, dialogando com entidades civis corporativas e acadêmicas [Brasil 2015a]. Já o SHCDCiber objetiva se "constituir em um sistema nacional de homologação e certificação de produtos e serviços de defesa cibernética, que seja economicamente sustentável junto à Base Industrial de Defesa nacional, e em conformidade a padrões nacionais de segurança de equipamentos e serviços necessários à defesa do espaço cibernético" [Brasil 2015b].

A Fundação Universidade de Brasília (FUB) executou recentemente o projeto "Apoio ao Programa Defesa Cibernética na Defesa Nacional: Viabilidade e Concepção da Escola Nacional de Defesa Cibernética e do Sistema de Homologação e Certificação de Produtos e Serviços de Defesa Cibernética - ENaDCiber-SHCDCIBER@DCDN", atestando a viabilidade de implantação da ENaDCiber e do SHCDCiber. Diversos dos aspectos metodológicos aqui referenciados são oriundos desse estudo.

2.1 Produto Mecatrônico e Videoconferência

Produto mecatrônico é um sistema com múltiplos componentes e domínios, envolvendo elementos mecânicos, eletrônicos, de controles e engenharia da computação [Behbahani e De Silva 2012].

A multidisciplinaridade possibilita integrar as características de cada um de seus componentes, otimizando-o sinergicamente."O entendimento da mecatrônica como aplicação da eletrônica na engenharia mecânica explica um número considerável de

aplicações da mecatrônica" [Barbalho 2006]. Diversos produtos, como robôs, sistemas de manufatura automatizados, automação de produtos de consumo como automóveis e dispositivos iOT (*internet of things*) exemplificam tal definição. Outros produtos mecatrônicos têm origem em equipamentos eminentemente eletrônicos. Um exemplo dessas aplicações são os terminais de videoconferência.

Terminais de videoconferência são compostos basicamente por: um codec (codificador/decodificador de áudio e vídeo); microfones; uma (ou mais) câmeras de vídeo (fixas ou com PTZ - pan/tilt/zoom); um monitor de vídeo (ao mínimo, ou um projetor de grandes dimensões); câmara de documentos (opcional).

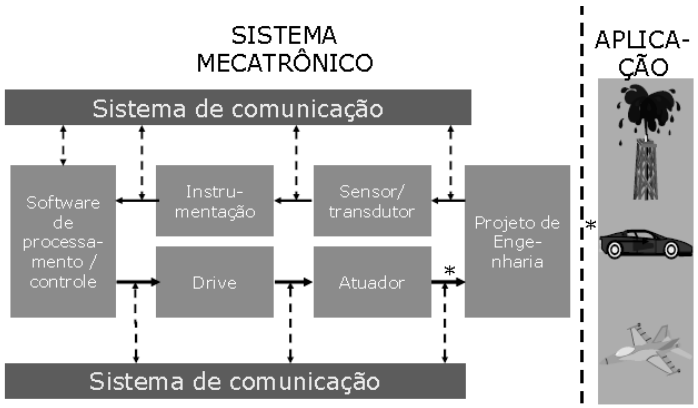


Figura 1 – Elementos de um Sistema Mecatrônico [Barbalho 2006]

A solução tecnológica utiliza redes convergentes, que encaminham e tratam voz, dados e vídeo em uma infraestrutura única de rede, através de uma única modalidade de equipamento. Com a videoconferência é possível a comunicação com voz e vídeo entre pessoas dispersas geograficamente, inclusive com compartilhamento de conteúdo. Os participantes podem acessar a conferência de qualquer lugar com acesso à Internet, usando seu próprio computador, com um microfone e uma webcam, ou seu dispositivo móvel.

O *Common Criteria for Information Technology Security Evaluation* (abreviado como *Common Criteria* ou CC) é um *framework* de Critérios Comuns de avaliação padronizada de Segurança da Informação voltada à segurança de computadores, padronizado na ISO/IEC 15048.O CC trata da proteção dos ativos com relação a três tipos de falhas de segurança: divulgação não-autorizada, modificação ou perda de uso. As categorias de proteção utilizadas são conhecidas por confidencialidade, integridade e disponibilidade, respectivamente.

Conforme suas necessidades, os usuários de sistemas computacionais especificam seus requisitos funcionais de garantia e de segurança [Common Criteria 2006]. Os fabricantes, ao seguirem esses requisitos na produção dos equipamentos de segurança computacional podem garantir que o fluxo de especificação, implementação e avaliação ocorreu conforme critérios determinados rigorosa e padronizadamente.

3. Metodologia

O estudo buscará identificar principalmente quais os riscos de segurança da informação envolvidos nos equipamentos de videoconferência e quais devem ser os parâmetros

mínimos a serem exigidos nesses sistemas, quando de sua fabricação, aquisição e utilização.

A partir do levantamento inicial serão verificados os requisitos funcionais de garantia e de segurança, conforme especificados no CC, proporcionando o desenvolvimento de um *Protection Profile* (PP). Para auxiliar nessa tarefa o *Target Of Evaluation* (TOE) a ser considerado será o equipamento de videoconferência, sendo o objetivo da pesquisa o desenvolvimento de um PP para o TOE. A metodologia empregada neste trabalho está apresentada na figura 2.

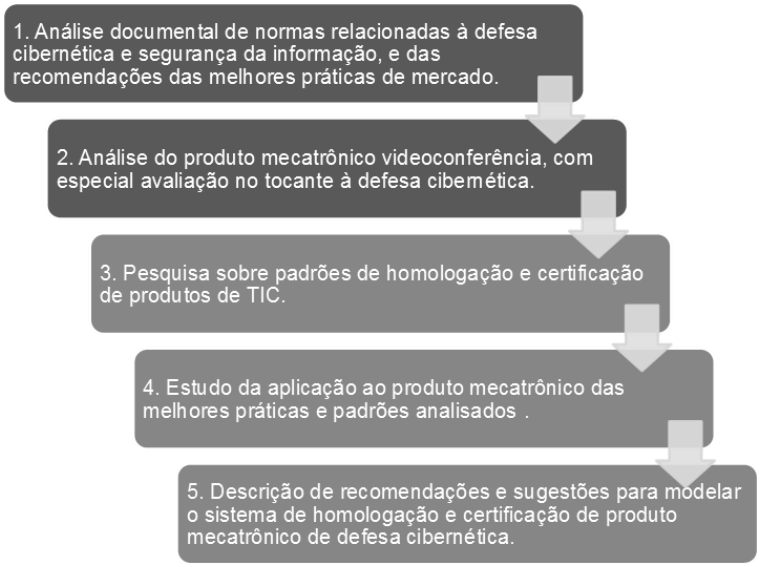


Figura 2 – Metodologia da pesquisa

Para fundamentação da pesquisa os seguintes conceitos serão analisados inicialmente: segurança cibernética; produto mecatrônico; sistema de videoconferência; segurança da informação e comunicações. Após o embasamento desses conceitos segue-se a análise do produto mecatrônico sob o ponto de vista da segurança, com vistas à elaboração de *protection profile* baseado no *Common Criteria*.

O trabalho acadêmico conclui com a análise do equipamento de videconferência utilizando o *protection profile* definido anteriormente. Tal análise permitirá descrever recomendações, propondo uma metodologia de homologação e certificação do produto de defesa cibernética.

4. Resultados e Discussões

Após a revisão bibliográfica dos conceitos fundamentais da pesquisa foram realizadas as análises estrutural e funcional do equipamento de videoconferência. Estruturalmente, o equipamento de videoconferência divide-se em: componentes ópticos, elétricos, eletroeletrônicos e de controle, conforme o esquema da figura 3.

Como componente óptico tem-se a câmera de vídeo. Em componentes elétricos agrupam-se: microfone, cabos de alimentação e de rede. O controle remoto e o decodificador de vídeo são considerados componentes eletroeletrônicos, enquanto que o software de controle é o componente de controle do produto mecatrônico.

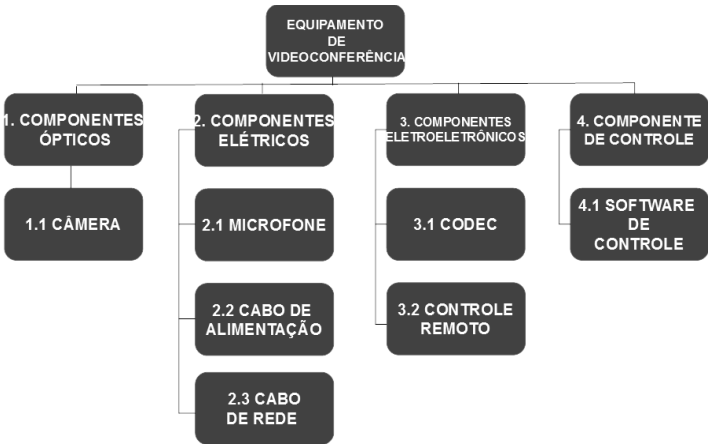


Figura 3 – Estrutura do Produto Mecatrônico Equipamento de Videoconferência

Conforme mostrado na figura 4, o produto mecatrônico apresenta dois sistemas principais: o sistema de alimentação e o sistema de controle.

O fluxo de energia entre esses sistemas e os componentes do equipamento estão representados pela linha tracejada. Dessa forma, estão diretamente associados ao sistema de controle o codec (codificador/decodificador) e o controle remoto. Ao sistema de alimentação relacionam-se diretamente: os cabos de alimentação e de rede, o microfone, a câmera e o monitor.

As setas de linha contínua representam o funcionamento essencial do sistema mecatrônico: os cabos de alimentação e de rede ligam-se diretamente ao microfone e câmera; o controle remoto direciona o software de controle do equipamento, que por sua vez controla o microfone, a câmera e o codec. Este último recebe instruções do software de controle, processando sinais de áudio e vídeo recebidos do microfone e da câmera e liberando-os ao monitor.

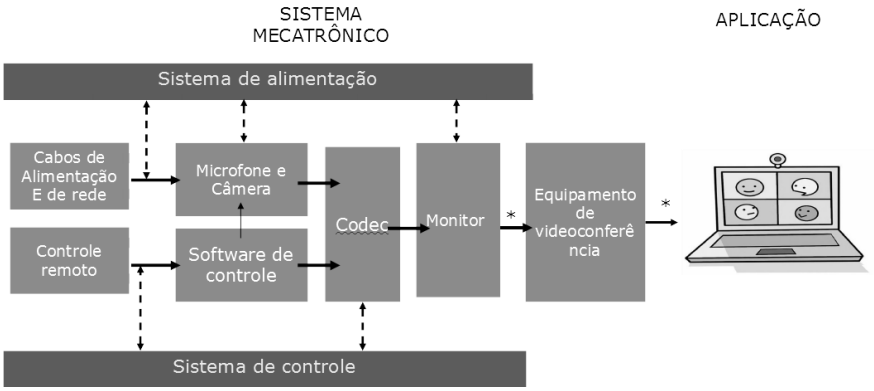


Figura 4 – Esquema Funcional do Equipamento de Videoconferência

Em sequência à análise do produto mecatrônico e de seu esquema funcional teve início o desenvolvimento do *protection profile,* com o objetivo de aplicar as melhores práticas de segurança da informação aos equipamentos de videoconferência, norteado pelo padrão *Common Criteria* de homologação e certificação de produtos de Tecnologia da Informação e Comunicações (TIC). Este é o estágio atual desta pesquisa acadêmica.

5. Considerações Finais

O uso da videoconferência pelas organizações permite agilidade, qualidade e segurança nas tomadas de suas decisões colegiadas, garantindo maior dinamismo na realização das transações comerciais entre seus clientes internos (diversas instâncias administrativas) e externos (clientes finais), já que dispensa deslocamento físico entre os funcionários para participar de reuniões com seus parceiros comerciais, dispersos pelo mundo.

Alinhada ao conceito de defesa cibernética e soberania do país, é vital a realização de um sistema de certificação e homologação de produto de defesa cibernética.

É extremamente válida a utilização de *protection profiles (PP)* para avaliar os equipamentos de videoconferência, considerando que o *Common Criteria* é um padrão internacionalmente reconhecido. Os requisitos de segurança a serem levantados para a formação do PP dos equipamentos devem possuir relação direta com as funcionalidades conforme a necessidade de utilização dos usuários. Para auxiliar nessa tarefa o TOE a ser considerado é o equipamento de videoconferência, sendo que o desenvolvimento de um PP para o TOE é o objetivo da pesquisa. Para o desenvolvimento do PP, os requisitos de segurança serão relacionados considerando ainda os seguintes aspectos: disponibilidade, integridade e confidencialidade dos dados.

O fato de um produto ter sido avaliado significa somente que no contexto de propriedades de segurança, determinadas características foram avaliadas através de métodos específicos, não que o produto seja completamente seguro, do ponto de vista computacional. As autoridades certificadoras devem verificar os produtos, propriedades e métodos para determinar que uma avaliação irá fornecer resultados significativos. Já os consumidores dos produtos avaliados devem considerar esse contexto para determinar se aquele produto é útil e aplicável a situações específicas e segundo suas necessidades.

Referências

Barbalho, S. C. M. (2006) “Reference Model for Mechatronic Product Development: Proposal and Applications”, Universidade de São Paulo, Escola de Engenharia de São Carlos, São Carlos.

Behbahani S. e De Silva C. W. (2012) A design paradigm for mechatronic systems. In *Mechatronics Jounal, Elsevier*, Volume 23, Issue 8, Dezembro, páginas 960 – 966, http://dx.doi.org/10.1016/,Dezembro 2015.

Brasil. Ministério da Defesa. Exército Brasileiro. (2015a) http://www.epex.eb.mil.br/index.php/projetos/defesa-cibernetica.html>, Dezembro.

Brasil. Ministério da Defesa. Exército Brasileiro. (2015b) “Estudo de Viabilidade do SHCDCiber no Âmbito do Projeto EnaDCiber-SHCDCiber@DCDN”, Brasília: UnB.

Common Criteria (2006) Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model, Version 3.1, Revision 1.

Junior, C. C. (2011) Sistemas Integrados de Gestão – ERP: Uma abordagem gerencial, Curitiba, Ibpex, 4ª edição.

SBSeg16

XVI SIMPÓSIO BRASILEIRO
EM SEGURANÇA DA INFORMAÇÃO
E DE SISTEMAS COMPUTACIONAIS

7 A 10 DE NOVEMBRO | NITERÓI | RJ

sbseg2016.ic.uff.br

Realizado por:

Apoiado por:

Patrocinado por:

Agência Brasileira do ISBN
ISBN 978-85-7669-350-5
9 788576 693505