.11.

SBSeg

SIMPÓSIO BRASILEIRO EM SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS

BRASÍLIA-DF

6 A 11 DE NOVEMBRO DE 2011

MINICURSOS

Editora
Sociedade Brasileira de Computação - SBC

Organizadores
Antonio Candido Faleiros
Célia Ghedini Ralha

Realização

Promoção
SBC – Sociedade Brasileira de Computação

Minicursos do XI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais – SBSeg 2011

ISBN: 978-85-7669-259-1

Organizadores
Antonio Candido Faleiros
Célia Ghedini Ralha

Para cópias adicionais destes anais, enviar pedido para:
Sociedade Brasileira de Computação
Av. Bento Gonçalves, 9500 – Setor 4 – Prédio 43424 – sala 116
Bairro Agronomia – CEP 91509-900 – Porto Alegre – RS
Fone/fax: (51) 3316-6835/(51)3316-7142
http://sbc.org.br

S612a Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (5. 2011 : Brasília, DF).
Minicursos / SBSeg 2011, XI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais, 6 a 11 de novembro de 2011, Brasília; coordenadores dos Minicursos Antonio Candido Faleiros, Célia Ghedini Ralha; coordenadores Anderson Clayton Alves Nascimento e Rafael Timóteo de Sousa Júnior. - Brasília: UnB: Sociedade Brasileira de Computação, 2011.
xii, 280p.

ISBN: 978-85-7669-259-1
Inclui bibliografia.

1. Computação – Brasil - Congressos. 2. Informática – Brasil – Congressos. I. Faleiros, Antonio Candido. II. Ralha, Célia Ghedini. III. Nascimento, Anderson Clayton Alves. IV. Sousa Júnior, Rafael Timóteo de. VI. Universidade de Brasília. VII. Sociedade Brasileira de Computação. VIII. Título.

CDD – 004.060981

"Esta obra foi impressa a partir de originais entregues, já compostos pelos autores."

Apresentação

O Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg) é um evento científico promovido anualmente pela Sociedade Brasileira de Computação (SBC). Representa o principal fórum no país para a apresentação de pesquisas e atividades relevantes ligadas à segurança da informação e de sistemas, integrando a comunidade brasileira de pesquisadores e profissionais atuantes nessa área.

Para o SBSeg 2011 foram selecionados seis minicursos:

1. Análise de Malware: Investigação de Códigos Maliciosos Através de uma Abordagem Prática;
2. Aprendizagem de Máquina para Segurança de Computadores: Métodos e Aplicações;
3. Técnicas para Análise Dinâmica de Código Malicioso;
4. Introdução à Composibilidade Universal;
5. Gerência de Identidades Federadas em Nuvens: Enfoque na Utilização de Soluções Abertas;
6. Live Forensics em Ambientes Windows.

O minicurso *Análise de Malware: Investigação de Códigos Maliciosos Através de uma Abordagem Prática* é composto por duas etapas básicas: uma teórica e uma análise real. O minicurso destina-se à divulgação do conhecimento necessário para que os interessados em análise de *malware* desenvolvam habilidades intrínsecas a um grupo de resposta a incidente e forense computacional. O minicurso apresenta os principais conceitos e discussões sobre as novas tendências de desenvolvedores de códigos maliciosos e contramedidas. Alguns modelos de condução de incidentes e análise de malware serão exibidos, juntamente com ferramentas utilizadas no processo.

O minicurso *Aprendizagem de Máquina para Segurança de Computadores: Métodos e Aplicações* analisa diferentes sistemas de detecção de atividades fraudulentas em páginas Web, de proliferação de códigos maliciosos, de ataques de negação de serviço, entre outros. A detecção de anomalias avançou significativamente com o emprego de técnicas de aprendizagem de máquina e de mineração de dados. Com um enfoque teórico, o minicurso fornece informações sobre o uso de técnicas de aprendizagem de máquina para segurança de dados, identificando técnicas são apropriadas ao problema de detecção de intrusão.

O minicurso *Técnicas para Análise Dinâmica de Código Malicioso* apresenta as principais técnicas utilizadas para efetuar a análise dinâmica de malware, que podem operar no nível do sistema operacional ou da Web, verificando quais estão presentes nos principais sistemas de análise disponíveis publicamente. Serão ainda mencionadas ferramentas utilizadas na captura de informações sobre a execução dos

programas maliciosos. Os participantes irão construir um sistema simples de análise dinâmica de malware e acompanhar um estudo de caso completo da análise de um malware proveniente da Web até o comprometimento do sistema operacional.

O minicurso *Introdução à Composibilidade Universal* tem como proposta apresentar um ambiente geral e modular para representar protocolos criptográficos e analisar a respectiva segurança. O minicurso permite a análise de protocolos complexos a partir de blocos mais simples. Durante o minicurso serão apresentados os conceitos básicos da segurança da Composibilidade Universal e a sua aplicação na concepção e análise de um protocolo criptográfico.

O minicurso *Gerência de Identidades Federadas em Nuvens: Enfoque na Utilização de Soluções Abertas* apresenta a idéia de transferir a maior parte do processamento e armazenamento das aplicações dos usuários para uma nuvem remota de serviços. A questão da segurança desta abordagem ainda é um problema em aberto e de difícil solução. Neste minicurso exploramos esta oferta federada de serviços sob a ótica da Gerência de Identidades. Diversas soluções abertas empregadas em ambientes de nuvem federadas serão apresentadas, finalizando com um estudo de caso em que se utiliza uma ferramenta que realiza experimentos de robótica em rede.

A utilização crescente de criptografia do disco e serviços Web mudou o perfil dos exames periciais em ambiente Windows. O minicurso *Live Forensics em Ambientes Windows* visa introduzir procedimentos de *live forensics*. *Live forensics* se caracteriza pelo exame das máquinas ainda em operação, permitindo a coleta de vestígios importantes, os quais podem ser perdidos com o desligamento da máquina. Neste minicurso serão abordados os procedimentos de *live forensics*, tais como reconhecimento de processos em execução, portas e arquivos em utilização, coleta e preservação de vestígios voláteis, utilizando apenas ferramentas disponíveis gratuitamente.

Os autores disponibilizam seus trabalhos à comunidade de segurança da informação e de sistemas computacionais e os editores esperam que os leitores encontrem inspiração técnica na leitura do material dos minicursos.

Os editores agradecem aos membros da Comissão de Avaliação dos Minicursos do SBSeg 2011 por sua colaboração voluntária e dedicação, lembrando que o sucesso dos minicursos deve ser creditado ao trabalho conjunto desta equipe.

Brasília, novembro de 2011.

Antonio Cândido Faleiros (Universidade Federal do ABC-UFABC)
Célia Ghedini Ralha (Universidade de Brasília-UnB)
Coordenadores dos Minicursos do SBSeg 2011

Organização do SBSeg 2011

Coordenadores Gerais
Anderson Clayton Alves Nascimento, UnB
Rafael Timóteo de Sousa Júnior, UnB

Comitê de Organização Local
Anderson Clayton Alves Nascimento, UnB
Aletéia Patrícia Favacho de Araújo, UnB
Edna Dias Canedo, UnB
Flávio Elias Gomes de Deus, UnB
Maristela Terto de Holanda, UnB
Laerte Peotta de Melo, UnB
Priscila América Solis M. Barreto, UnB
Rafael Timóteo de Sousa Júnior, UnB
Ricardo Staciarini Puttini, UnB

Coordenadores do Comitê de Programa
Jeroen van de Graaf, UFMG
Luiz Fernando Rust da Costa Carmo, UFRJ

Coordenadores de Minicursos
Célia Ghedini Ralha, UnB
Antonio Cândido Faleiros, UFABC

Coordenadora do WTICG
Michelle Nogueira, UFPR

Coordenador do Fórum de Segurança Corporativa
Rafael Timóteo de Sousa Júnior, UnB

Comissão de Avaliação dos Minicursos

Antonio Candido Faleiros (UFABC), Coordenador
Célia Ghedini Ralha (UnB), Coordenadora
Jacir Luiz Bordim (UnB)
Julio Cesar López Hernández (UNICAMP)
Marcos Antonio Simplicio Jr (USP)
Ricardo Dahab (UNICAMP)

SBC - Sociedade Brasileira de Computação

Sumário

Minicursos

Capítulo

1

Análise de Malware: Investigação de Códigos Maliciosos Através de uma Abordagem Prática

Laerte Peotta de Melo[1], Dino Macedo Amaral[1], Flavio Sakakibara[2], André Resende de Almeida[2], Rafael Timóteo de Sousa Jr.[1], Anderson Nascimento[1].

[1]Departamento de Engenharia Elétrica – Universidade de Brasília (UnB).
[2]Departamento de Ciência da Computação – Universidade de Brasília (UnB).
Campus Universitário Darcy Ribeiro – Asa Norte – 70910-900 – Brasília, DF – Brasil.

Resumo

Este curso destina-se a divulgar o conhecimento necessário para que profissionais interessados em análise de malware desenvolvam habilidades esperadas em um grupo de resposta a incidentes e forense computacional. A abordagem utilizada é teórico-prática, de modo que o aluno terá acesso aos principais conceitos e discussões sobre as novas tendências de desenvolvimento de códigos maliciosos bem como suas contramedidas. Modelos de condução de incidentes e análise de malware serão apresentados, juntamente com ferramentas utilizadas no processo, apontando quais informações são primordiais e as preocupações necessárias para se conter um incidente. Após a introdução o aluno será desafiado a executar uma análise real, discutida e apresentada em dois cenários distintos.

1.1. Introdução

Nos últimos anos o número de códigos maliciosos vem crescendo, acompanhando o aumento do número de usuários que acessa a Internet [Finjan Research Center 2009]. As defesas frente a essa ameaça têm se mostrado ineficientes. Empresas de segurança que desenvolvem ferramentas de detecção utilizam a forma clássica para encontrar e neutralizar os códigos maliciosos, ou seja, a forma baseada em assinaturas [Dube et al. 2010]. Assim, torna-se notório para o entendimento dos processos de segurança e inovação da tecnologia que é necessário conhecer um artefato para que este seja considerado malicioso. Da mesma forma, é preciso que um vetor de infecção seja encontrado e analisado para que uma assinatura seja produzida.

Devido à grande quantidade desses artefatos, algumas organizações e empresas de segurança utilizam métodos de análise automatizada ou semi-automatizada. Conhecer o funcionamento de um código malicioso é a base para se produzir ferramentas de detecção e proteção eficientes, pois permite que se conheça o contexto que o *malware* pretende atingir, entendendo o público alvo da ameaça, as informações coletadas, o uso e o destino dos dados. Entretanto, o tempo para encontrar as defesas não tem se mostrado condizente com o cenário atual dos ataques, que é extremamente desfavorável para o usuário final. Com essa motivação, os pesquisadores têm se mostrado dispostos a enfrentar o problema e confiantes em propor soluções que tornem os processos, tanto de análise quanto de detecção, eficientes e confiáveis.

Vale notar que, em casos que envolvem fraudes financeiras e roubo de identidades, conhecer a atuação do *malware* é preponderante para erradicar o incidente. A simples descoberta de códigos maliciosos pode sugerir uma ação preventiva em relação, por exemplo, a impedir que um usuário tenha acesso a um sistema, pois não é possível garantir que esse usuário seja corretamente reconhecido, pois o comprometimento de suas informações de identificação fazem o risco de fraude tornar-se alto. Um grande problema enfrentado nessas situações é determinar qual o tipo de comprometimento, ou seja, encontrar qual a capacidade que o código malicioso tem em capturar informações. Esse fator é determinante do tempo de resposta que a organização e as empresas de segurança têm para produzir uma assinatura de reconhecimento do malware.

1.1.1. Justificativas

Equipes de resposta a incidentes de segurança (CSIRT)[Brownlee and Guttman 1998] têm enfrentado grandes desafios para manter seus sistemas seguros e controlar possíveis comprometimentos de sistemas que estão sob sua responsabilidade. Entre os desafios a serem enfrentados estão os *malwares*, que são códigos capazes de comprometer totalmente uma infraestrutura de tratamento da informação e que estão tornando cada vez mais complexas as defesas. Além da questão das inovações nas formas de ataques, o número crescente desses códigos maliciosos [Finjan Research Center 2009] representa uma preocupação, não somente para as empresas, mas também para entidades governamentais, que devem se preocupar com ameaças que podem causar danos sistemas críticos de um país[Chen and Abu-Nimeh 2011] e afetar diretamente a segurança física do cidadão desse país.

1.1.2. Objetivos

O curso tem como objetivo introduzir o assunto tanto à comunidade acadêmica quanto a profissionais interessados no tema, de modo a subsidiar o entendimento de atuação de *malwares*. O foco é colocado em *trojans* bancários, procurando fazer o aluno compreender todo o processo de detecção, análise e tratamento desse tipo de artefato, bem como demonstrar a obtenção de evidências de que o artefato teve sucesso na coleta de informações da máquina comprometida.

A proposta é que o aluno desenvolva habilidades de identificar uma ameaça, analisar e avaliar o grau de sofisticação. Para isso o curso é aplicado utilizando exposições teóricas e práticas em laboratório. O perfil esperado para os alunos é o de profissionais

que atuem na área de segurança da informação, centro de resposta e tratamento de incidentes de segurança, forense computacional, assim como de discentes/docentes de cursos de engenharia e ciência da computação, com noções básicas de sistemas operacionais Windows e Linux, e redes TCP/IP.

1.1.3. Organização do minicurso

Este minicurso está organizado da seguinte maneira: no capítulo 1.1 se apresenta o tema sobre o ponto de vista de sua importância e justificativas para uma formação com objetivo final esperado. A questão do tratamento de incidentes, os termos mais importantes, juntamente com um estudo de caso de análise forense para *phishing*, sãos os temas apresentados no capítulo 1.2. O capítulo 1.3 detalha metodologias aplicadas em forense computacional, suas estratégias de atuação e preservação de evidências. Técnicas e metodologias para análise e condução de incidentes que envolvem *malwares* são detalhadas no capítulo 1.4. A seguir, tratam-se os modelos de análise de metadados no capítulo 1.5, a identificação de códigos ofuscados no capítulo 1.6 e as funções de *hash* e *fuzzy hash* no capítulo 1.7. Finalmente, no capítulo 1.8 são apresentados estudos de casos para análise de *malwares* e, no capítulo 1.9, um modelo de relatório de análise que serve para documentar e gerir o processo de investigação de *malwares*.

1.2. Identificação e Tratamento de Incidentes de Segurança

Nesta seção apresentamos as definições de tipos de incidentes e ataques, incluindo os artefatos de vírus, *worm*, *trojan horse* e os principais ataques, como engenharia social, *spam*, *honeynet*, *phishing* [Steding-Jessen 2008], *botnets*, negação de serviço [Binsalleeh et al. 2009] e exploração de vulnerabilidades [Abbas et al. 2006]. São também abordados os principais fatores da resposta a incidentes, os grupos de resposta a incidentes de segurança (CSIRT) [Brownlee and Guttman 1998], suas etapas de trabalho e as metodologias de análise e condução de incidentes, compostas por atividades de coleta, extração, tratamento e resposta[Kent et al. 2006].

Um incidente de segurança é um evento indesejado ou inesperado, que pode ser único ou em série e que possui características de comprometimento do bom funcionamento de uma organização, resultante de uma ameaça à segurança da informação[ABNT 2005]. Especificamente, um incidente pode ser relacionado a eventos como invasões, acessos não autorizados, negação de serviço, contaminação de sistemas através de *malwares*, furto de informações ou qualquer atividade considerada ilegal ou ilegítima. Nesses casos, é necessário que os eventos sejam investigados por pessoas qualificadas a fim de resolverem o problema causado.

Programa malicioso, código malicioso, softwares malicioso, ou simplesmente *malware*, são expressões que se referem a um programa que é inserido em um sistema, normalmente de forma encoberta, com a intenção de comprometer a confidencialidade, integridade ou disponibilidade dos dados da vítima, aplicativos ou o próprio sistema operacional. A produção de *malwares* se tornou a ameaça mais significativa para a maioria dos sistemas, causando danos generalizados e perturbação, ocorrências que necessitam de esforços de recuperação extensa dentro da maioria das organizações.

As técnicas utilizadas pelos *malwares* são associados à violação de privacidade de

um usuário, valendo observar que o denominado *spyware* tornou-se uma grande preocupação para as organizações. Embora a violação da privacidade através desses programas não seja novidade, tornou-se muito mais difundida recentemente [Mell and karen Kent 2005]. A monitoração das atividades pessoais e a realização de fraudes financeiras são ocorrências cada vez mais comuns, assim como a ocorrência de sistemas invadidos. As organizações também enfrentam outras ameaças que são frequentemente associadas aos *malwares*. Uma das formas que tem se tornado comum é o *phishing*. Outras formas de artefatos, como vírus, *backdoors*, cavalos de tróia, *keyloggers*, *rootkits*, também se denominam pelo termo *malware* e serão abordadas a seguir neste capítulo.

1.2.1. Spam e Phishing

O termo *spam* é utilizado para referenciar o recebimento de uma mensagem não solicitada, que geralmente tem o caráter de fazer propaganda de algum produto ou assunto não desejado. O termo em inglês utilizado para esse tipo de mensagem é *UCE (Unsolicited Commercial Email)*. A palavra *spammer* é o termo utilizado para definir quem envia esse tipo de mensagem, em cuja elaboração geralmente são utilizados softwares especiais automatizados para coletar bases de emails através de listas de discussão, caixas postais de grupos, ou exploração, através de listas adquiridas, por empresas especializadas nesse tipo de *marketing*. Existem diversas soluções no mercado que buscam conter o recebimento de *spams*, ferramentas estas que basicamente utilizam filtros contendo *blacklists* de *proxys* e *relays* abertos, que são utilizados para o envio das mensagens.

Servidores de *proxies* são utilizados por organizações de maneira a tornar suas estruturas de comunicação mais seguras, pois permitem prover acesso de redes internas a redes externas, como a *Internet*, sendo capazes de impor controles de acesso a conteúdos definidos em uma política de segurança e gerar informações de auditoria. Os servidores *relay* são utilizados conjuntamente com o protocolo SMTP *(Simple Mail Transfer Protocol)* de transmissão e recebimento de mensagens, protocolo este que é considerado inseguro, pois, caso um atacante tenha acesso ao meio de comunicação e capture o tráfego gerado entre cliente e servidor, as credenciais de cada um não são protegidas por criptografia. Apesar de existir serviços SMTP que utilizam de criptografia, a utilização é baixa. Por tais razões, servidores SMTP que estejam com o serviço de *relay* ativado podem permitir que os serviços sejam utilizados por *spammers* e que as mensagens sejam enviadas. Diante de tal situação, recomenda-se que os servidores sejam configurados utilizando técnicas de *hardening*, de modo a permitir que somente pessoas autenticadas e autorizadas possam enviar mensagens, restringindo o uso por terceiros. A Figura 1.1 confirma que é crescente o envio de mensagens não solicitadas.

O termo *phishing*, *phishing scam* ou *phishing/scam* ("fishing"significa pescaria), refere-se ao método pelo qual *iscas* (e-mails) são usadas para *pescar* informações de usuários da Internet, constituindo-se um ataque que tem como objetivo efetuar algum ilícito através do envio de mensagem não solicitada. Essa técnica geralmente é associada à engenharia social, atividade em que o *spammer* explora algum tema de modo a induzir o usuário final a executar alguma atividade ou ação que não faria normalmente. Essas mensagens são projetadas para roubar informações como: dados pessoais, credenciais de clientes de instituições financeiras de usuários. Para compreender a análise da mensagem, deve-se entender o comportamento esperado de um servidor SMTP padrão:

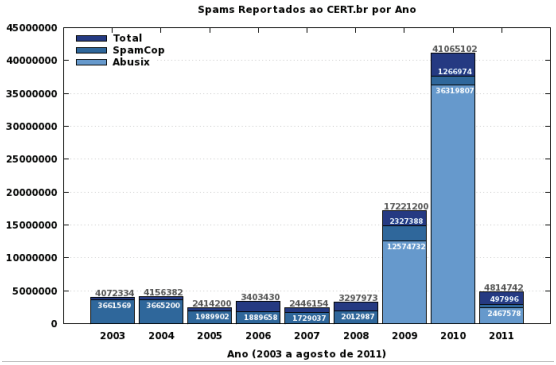

Figura 1.1. Valores acumulados: 2003 a agosto de 2011 - Fonte: CGI

1. Usuário escreve uma mensagem utilizando um software ou agente web (MUA - Mail User Agent Outlook, Eudora, Webmail);
2. Usuário envia a mensagem através do software utilizado para escrever a mensagem (MUA);
3. O software (MUA) conecta-se ao servidor SMTP do usuário (MTA - Mail Transfer Agente Sendmail), para isso usa o protocolo SMTP-AUTH, que envia o *username* e *password* do usuário;
4. O servidor SMTP recebe e armazena a mensagem na fila para envio;
5. O servidor SMTP localiza o domínio de destino da mensagem utilizando o protocolo DNS;
6. O servidor SMTP conecta-se conecta ao servidor SMTP do destino da mensagem;
7. Os servidores SMTP trocam identificação;
8. O servidor SMTP remoto verifica os cabeçalhos da mensagem;
9. O servidor SMTP remoto autoriza ou rejeita a mensagem, de acordo com regras de verificação de reverso e políticas anti-spam;
10. O servidor SMTP remoto recebe e entrega ao servidor local (LMTA - Local Mail Transfer Agent Procmail);
11. O servidor local (LMTA) processa a mensagem e entrega na Caixa Postal do usuário destino;
12. O usuário, utilizando o software para acesso (MUA) e suas credenciais *username* e *password* se autentica ao seu servidor de email MRA (Mail Reader Agent POP3, IMAP4) POP *(Post Office Protoco)* ou Imap *(Internet Message Access Protocol)* e

13. O servidor de email (MRA) entrega a nova mensagem para o software de acesso do cliente (MUA);

14. O Usuário pode ler a mensagem.

A Figura 1.2 apresenta um exemplo desse tipo de mensagem. Todo o processo de envio e recebimento é passível de auditoria, permitindo que se compreenda a taxonomia do evento que um *spammer* utiliza.

Em certas ocasiões, o *spammer* não se preocupa em ocultar sua origem, tornando o rastreamento rápido. Entretanto, na maioria dos casos a origem é ocultada ou alterada de maneira a indicar outro local de envio da mensagem.

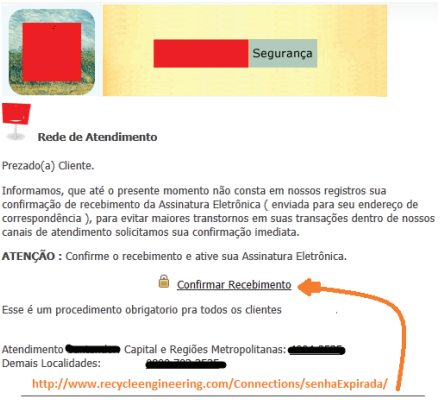

Figura 1.2. Exemplo de phishing

Analisando o exemplo da Figura 1.2 é possível coletar uma grande quantidade de informações que podem ser utilizadas para se determinar pontos de comprometimento. A informação coletada apontando o mouse para o link de "Confirmar Recebimento"apresenta o link para download de artefato malicioso, que neste caso indica ser um cavalo de tróia para roubo de credenciais de clientes bancários. Portanto qualquer e-mail enviado possui informações que não são apresentadas pelos programas ou interfaces de usuário. Essas informações podem ser obtidas analisando diretamente o código fonte da mensagem. O código da mensagem, referente a Figura 1.2, é apresentada pela Listagem 1.1.

Listagem 1.1. Listagem de cabeçalho e-mail phishing

```
1 Delivered-To: peotta@gmail.com
2 Received: by 10.142.166.12 with SMTP id o12cs148327wfe;
3         Tue, 20 Sep 2011 00:13:22 -0700 (PDT)
4 Received: by 10.236.179.97 with SMTP id g61mr2104588yhm
    .119.1316502409640;
5         Tue, 20 Sep 2011 00:06:49 -0700 (PDT)
6 Received-SPF: neutral (google.com: 200.175.8.217 is neither permitted
    nor denied by best guess record for domain of unknown) client-ip
    =200.175.8.217;
```

```
7  Received: by 10.200.27.12 with POP3 id a12mf92257gwa.33;
8          Tue, 20 Sep 2011 00:06:49 -0700 (PDT)
9  X-Gmail-Fetch-Info: lpi@pop.com.br 1 mail.pop.com.br 110 lpi
10 Received: (qmail 8396 invoked from network); 20 Sep 2011 06:52:38 -0000
11 Received: from unknown ([200.175.8.217])
12           by pop-37.pop.com.br (qmail-ldap-1.03) with QMQP; 20 Sep 2011
                 06:52:38 -0000
13 Delivered-To: CLUSTERHOST pop-8.pop.com.br lpi@pop.com.br
14 Received: (qmail 25792 invoked from network); 20 Sep 2011 06:52:37
       -0000
15 Received: from vvps-152833.dailyvps.co.uk ([91.223.16.239])
16           (envelope-sender <root@vvps-152833.dailyvps.co.uk>)
17           by pop-8.pop.com.br (qmail-ldap-1.03) with SMTP
18           for <lpi@pop.com.br>; 20 Sep 2011 06:52:37 -0000
19 X-C3Mail-ID: 1316501546274674
20 Received-SPF: none (pop-8.pop.com.br: domain at vvps-152833.dailyvps.co
       .uk does not designate permitted sender hosts)
21 Received: from vvps-152833.dailyvps.co.uk (localhost.localdomain
       [127.0.0.1])
22    by vvps-152833.dailyvps.co.uk (8.13.8/8.13.8) with ESMTP id p8K6qPYf
          023569
23    for <lpi@pop.com.br>; Tue, 20 Sep 2011 07:52:25 +0100
24 Received: (from root@localhost)
25    by vvps-152833.dailyvps.co.uk (8.13.8/8.13.8/Submit) id p8K6qPgZ
          023568
26    for lpi@pop.com.br; Tue, 20 Sep 2011 07:52:25 +0100
27 Date: Tue, 20 Sep 2011 07:52:25 +0100
28 From: Grupo Santander S/A <santander@santander.infoemail.com>
29 To: lpi@pop.com.br
30 Subject: Regularize sua Assinatura ôEletrnica!
31 Message-ID: <1316501545.@santander.infoemail.com>
32 Content-Type: text/html
33 X-Remote-IP: 91.223.16.239
34 X-Spam-Status: No, tests=filter, spamicity=0.500001
```

A investigação desse tipo de ocorrência é feita analisando o código fonte da mensagem sempre de baixo para cima, neste caso a linha 34 é uma informação de que esta mensagem especificamente foi testada por um filtro de spam e não foi considerada uma mensagem não solicitada. A linha 33 é importante para definir a origem do spam, o servidor SMTP insere essa informação para facilitar a identificação de quem envia, o campo pode variar de acordo com a versão, podendo ter nomenclatura como *X-Originating-IP* ou *X-IP*.

Para identificar a origem do *spammer* utilizaremos o campo *X-Remote-IP* que aponta para o IP 91.223.16.239. Utilizando o protocolo *whois*, é possível encontrar informações como: Contato Administrativo (Admin Contact), Contato Técnico (Technical Contact) e Contato de Cobrança (Registrant Contact). A saída resumida está disponível através da Listagem 1.2.

Listagem 1.2. Resultado para a consulta whois

```
1 IP Information for 91.223.16.239
2 IP Location:   United Kingdom United Kingdom Daily Internet Ltd
3 ASN:  AS31727
```

```
4  Resolve Host:   vvps-152833.dailyvps.co.uk
5  IP Address:     91.223.16.239
6  inetnum:        91.223.16.0 - 91.223.16.255
7  netname:        DAILY-UK-VPS
8  descr:          Daily Internet LTD
9  country:        GB
10 org-name:       Daily Internet Ltd
11 org-type:       OTHER
12 address:        67-69 George Street
13 address:        LONDON
14 address:        W1U 8LT
15 phone:          +44 115 9737260
16 fax-no:         +44 115 9725140
17 person:         Andrew Gilbert
18 address:        Node4 Ltd
19 address:        Millennium Way
20 address:        Pride Park
21 address:        Derby
22 address:        DE24 8HZ
23 phone:          +44 845 123 2222
```

A listagem 1.1, os campos apresentados pela linha 28 e 29, respectivamente informam origem e destino da mensagem. A linha 27 apresenta informação sobre a data, importante recurso para se conduzir identificação de quebra de sigilo judicial, pois a data é necessária para associar uma conexão a um usuário naquele exato momento. Como o servidor está localizado no Reino Unido (linha 2 da Listagem 1.2 é importante considerar o horário mundial, que neste caso é determinado pelo Greenwich Mean Time (GMT). A linha 11 revela que a mensagem foi entregue para o cliente do servidor *pop.com.br*, entretanto este ainda não é o destino final da mensagem, continuando a análise podemos perceber através da linha 9 que o servidor *Gmail* coletou a mensagem utilizando uma conexão através do protocolo Post Office Protocol (POP3) utilizando a porta 110. O destino final então é determinado pelo campo *Delivered-to*, apresentado na linha 1.

De acordo com a análise da mensagem recebida e apresentada pela Figura 1.2, foi possível analisar informações de headers da mensagem, conforme Listagem 1.1, identificando sua origem, conforme Listagem 1.2.

O método de roubo pode ser associado a um formulário web, onde as informações são solicitadas à vitima, outro método é conhecido como cavalo de tróia, em que a vítima instala o artefato de maneira desavisada. Os cavalos de tróia são tratados na seção 1.2.4. Alguns exemplos de situações que envolvem esse tipo de ataque:

- Formulários recebidos por e-emails que solicitam informações confidenciais;
- Links recebidos que direcionam para a instalação de cavalos de tróia;
- Alteração através de malwares que direcionam o tráfego DNS para um *proxy* controlado pelo atacante;
- Páginas falsificadas de empresas de comércio eletrônico, sites de leilões, instituições financeiras, órgãos do governo.

Praticamente os ataques que, utilizam técnicas de engenharia social, tem como vetor de comprometimento o *phishing*. *Spammers* precisam convencer o usuário a executar ações que tem como objetivo comprometer seu dispositivo, portanto esse tipo de ataque é fortemente dependente da colaboração da vítima.

Ganhar a confiança do usuário tem sido um desafio para os atacantes, que buscam camuflar suas investidas tornando-se parecidos com sites de serviços mais comumentes utilizados, como bancos, redes sociais, servidores de e-mail, comércio eletrônico. Como exemplo podemos observar a Figura 1.3, onde o atacante tenta reproduzir, de maneira fiel, o *site* de uma instituição financeira, entretanto, nesse tipo de ataque o ponto a ser explorado é a distração da vítima, que não percebe as modificações, muitas vezes grosseiras e que solicita de maneira direta o que o atacante precisa para se passar por um cliente legítimo da instituição.

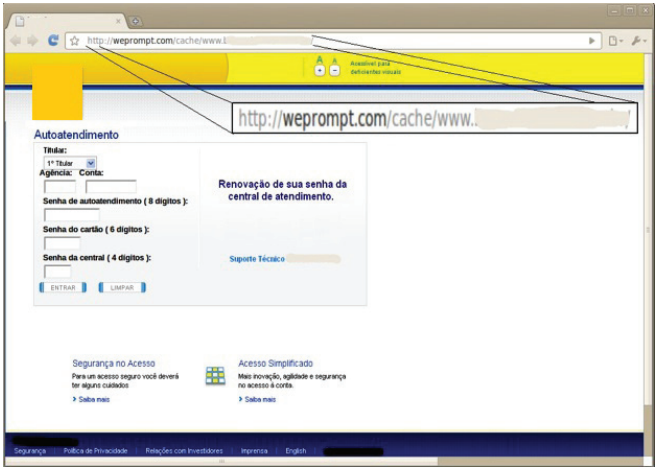

Figura 1.3. Exemplo de página falsa

Os programas maliciosos podem ser divididos primordialmente através de sua dependência ou não do hospedeiro. Os tipos dependentes são por exemplo: vírus, bombas lógicas e *backdoors*, os não-dependentes *worms* e zumbis. A replicação é outra maneira de se classificar, pois existem os programas que possuem a capacidade de se replicar, como vírus e *worms*, e os que não se replicam, como bombas lógicas, *backdoors*, zumbis, *rootkits* e *keylogger*. Entretanto é bastante comum os *malwares* que possuem capacidades híbridas, sendo capazes de se replicar e ao mesmo tempo possuírem outras características inerente às necessidades. Para que a classificação de cada tipo seja feita, é necessário que os termos utilizados sejam de conhecimento do analista que irá conduzir a investigação, portanto apresentaremos uma breve descrição de cada tipo mais comum de malware.

1.2.2. *Honeynets* e *Honeypots*

Honeypots são recursos e serviços computacionais em rede criados e monitorados com o objetivo de serem atacados e comprometidos, visando identificar novos ataques e ferramentas utilizadas. *Honeynet* são conjuntos de sensores monitorados de maneira a identificar tráfego suspeito e malicioso.

Um sistema de *honeypots* pode ser dividido em duas categorias[Steding-Jessen 2008]:

Alta Interatividade: Todos os sistemas e serviços disponibilizados são serviços reais sem inserções de vulnerabilidades propositadas. Nesse ambiente todo o tráfego é monitorado e controlado, de maneira que um comprometimento não seja utilizado para causar danos a outras redes.

Baixa Interatividade: Os sistemas são reais, entretanto os serviços disponibilizados são emulados através de ferramentas criadas exclusivamente para reproduzir o comportamento esperado desses serviços. Oferecem um grau menor de risco de serem utilizados como base a ataques a outras redes.

O objetivo dos *honeypots*, independentemente de sua categoria, é coletar informações de ataques, identificando origens e destinos, bem como serviços mais procurados. A monitoração é feita geralmente utilizando-se sistemas de detecção de intrusos (IDS) e armazenamento e correlacionamento de *logs*.

1.2.3. Vírus

Os vírus de computador foram introduzidos na década de 1980, com funções simples que ocasionalmente geravam inconvenientes ou apenas mostravam informações ao usuário. Atualmente esse tipo de código traz um risco significativo com potencial destrutivo e que demandam grande esforço das organizações para manterem seus sistemas a salvo. A designação de vírus é tratada de acordo com sua função. Para que um código malicioso seja considerado um vírus ele deve ter a capacidade de auto replicação, ou seja, fazer uma cópia de si mesmo e distribuir essa cópia para outros arquivos e programas do sistema infectado.

Cada vírus pode utilizar um mecanismo de infecção diferente, por exemplo: inserir seu código em programas executáveis sobrescrevendo parte do código de maneira a permitir que toda vez que o programa infectado seja executado o código viral também seja executado. Portanto o principal objetivo de um vírus é replicar-se e contaminar o maior número possível de programas, de maneira a comprometer outros sistemas. Podemos dividir os vírus em duas grandes categorias [Kent et al. 2006]: vírus compilados e vírus interpretados.

A) Vírus compilados

Esse tipo de código malicioso é composto de um programa fonte que é compilado com alguma linguagem e direcionado para um determinado sistema operacional, tipicamente esse tipo de vírus pode dividido em três categorias:

Infector de programas: Infectam programas executáveis, onde o programa infectado se

propaga para infectar outros programas no sistema, bem como outros sistemas que usam um programa infectado compartilhado.

Setor de Boot: Infectam o setor Master Boot Record (MBR) do disco rígido ou mídia removível inicializável. Esse tipo de vírus foi muito utilizado nas décadas 1980 e 1990, atualmente está em desuso.

Multipartite: Utiliza métodos de infecção múltipla, normalmente infectando arquivos e setores de boot. Assim, os vírus multipartite combinam as características dos vírus que infectam arquivos e setor de inicialização.

B) Vírus interpretados

Esse tipo de vírus é composto de código-fonte que pode ser executado apenas por uma determinada aplicação ou serviço, ao contrário dos vírus compilado, que pode ser executada por um sistema operacional. Tornaram-se mais comuns, pois considerados mais fáceis de escrever e modificar do que outros tipos de vírus. Não é necessário desenvolver capacidades técnicas elevadas, pois um atacante pode adquirir um vírus interpretado e modificar seu código fonte. Muitas vezes há dezenas de variantes de um único vírus interpretado, apenas com alterações triviais a partir do original. Os dois principais tipos de vírus interpretados são: vírus de macro e vírus de script.

Vírus de macro: Utilizam técnicas de propagação baseadas em anexo de documentos que executam macros, como planilhas eletrônicas e processadores de texto. Estes vírus tendem a se espalhar rapidamente, porque os usuários freqüentemente compartilham documentos com recursos de macro habilitados. Além disso, quando uma infecção ocorre, o vírus infecta o modelo que o programa usa para criar e abrir arquivos. Uma vez que um modelo é infectado, cada documento que é criado ou aberto com esse modelo também será infectado.

Vírus de script: Nesse tipo de vírus a principal diferença, em relação ao de macro, é a linguagem utilizada. O código é escrito para um serviço e sistema operacional específico, como por exemplo VBScript que executa em um servidor com sistema operacional Windows.

Técnicas de ofuscação podem ser utilizadas no código de vírus, particularmente para dificultar sua detecção. As técnicas mais comuns são:

Criptografia: O código possui funções para cifrar e decifrar o executável do vírus, gerando porções de códigos aleatórios para cada infecção.

Polimorfismo: Um modelo mais robusto que usa criptografia, com o código gerando um executável com tamanho e aparência diferentes para cada infecção, podem também habilitar funcionalidades diferentes na ação.

Metamorfismo: Assim como os polimórficos mudam o comportamento, podendo gerar códigos novos a cada iteração através de técnicas de códigos desnecessários, sendo recompilado e criando um novo executável.

1.2.4. Cavalos de Tróia

De acordo com a mitologia Grega, um cavalo de Tróia é um "presente"que esconde uma ação não esperada, são programas que não se replicam e que tem aparência inofensiva, escondendo sua proposta maliciosa. Esse tipo de malware pode tomar o lugar de um programa com intenções legítimas, executar todas as funções esperadas e então, conjuntamente, executar as ações para a qual foi programado, dentre essas ações pode-se citar:

- Coletar informações da máquina contaminada, como credenciais, fotos, arquivos;
- Esconder sua presença e ações de maneira a impedir ou dificultar sua detecção;
- Desabilitar softwares de segurança como *firewall*, anti-virus, *anti-malwares*.

Atualmente os cavalos de tróia utilizam diversas técnicas para que sua presença passe desapercebida ou que sua detecção seja difícil. Esse tipo de malware é o mais comum quando se trata do cenário da Internet brasileira, pois é bastante utilizado por fraudadores que aproveitam sua ação para coletar todo o tipo de informação necessária para fraudes financeiras em lojas de comércio eletrônico, bancos, financeiras, etc. No capítulo 1.6 trataremos das técnicas de ofuscação que são utilizadas pelos atacantes com o objetivo de dificultar sua identificação e análise.

1.2.5. Worms

Um programa que é capaz de se propagar automaticamente através de redes, enviando cópias de si mesmo de computador para computador, de sistema para sistema, essa é a definição de um worm. Diferentemente do vírus, o *worm* não tem a capacidade de infectar outros programas inserindo seu código em outros programas ou arquivos, também não depende de execução para se propagar em outros sistemas, pois é através da exploração de vulnerabilidades que esse tipo de malware contamina outros sistemas, aproveitando de falhas de configuração ou softwares vulneráveis.

A grande ameaça do *worm* reside em deteriorar a performance de um sistema, sobrecarregando serviços e gerando tráfego, que em alguns casos, pode provocar negação de serviço (*DoS - Deny of Service*). Por ser capaz de comprometer sistemas sem a necessidade de um usuário instalar ou executar algum programa, esse tipo de ataque tem se popularizado entre os criminosos, pois permite que em um curto espaço de tempo um grande número de sistemas seja comprometido, diferente dos vírus, que dependem do hospedeiro e usuário. Outra vantagem é que vulnerabilidades e configurações mal feitas são bastante comuns e diariamente surgem novas fraquezas em sistemas. Para efeito de classificação, existem dois tipos principais de worms, serviços em rede (*network service worms*) e envio em massa (*mass mailing worms*).

Os *worms* de serviços (*network service worms*) utilizam o processo de exploração de uma vulnerabilidade em um serviço de rede, geralmente associado a um determinado sistema operacional ou aplicativo. Depois que um sistema esteja infectado o código malicioso inicia a verificação de outros sistemas vulneráveis pela rede. Sua ação aumenta a medida que outras máquinas são comprometidas, o que justifica a queda de performance do sistema de comunicação. Como não há necessidade de interação humana, rapidamente uma rede vulnerável pode ser totalmente mapeada e comprometida.

1.2.6. Backdoors

O termo *backdoor* é geralmente associado a um programa malicioso que fica aguardando uma determinada ação para que permita uma conexão remota ou acesso não autorizado, e que não seja detectado pelos dispositivos de segurança. A instalação de um backdoor é inerente à invasão de um sistema, onde o atacante habilita um "serviço"que o permita ter acesso quando necessário ao sistema comprometido, geralmente possuem capacidades especiais:

Zombies: O termo zumbi é decorrente da utilização de máquinas comprometidas através de *bots*, possuindo a capacidade de propagação como um worm. Essas máquinas são controladas por atacantes que compõem um sistema maior. Botnets são redes compostas por milhares de computadores zumbis e que são utilizadas para ataques coordenados, como DDoS (*Distribution Deny of Service*), envio de *spams* e *phishing*.

Remote Administration Tools - RAT: Permite a um atacante ter acesso a um sistema quando desejar e em alguns casos o sistema controlado coleta e envia informações como imagens de tela, teclas digitadas, arquivos acessados sendo possível ativar remotamente dispositivos como impressoras, webcams, microfones e autofalantes. Exemplos de RAT's: Back Orifice, Netbus e SubSeven.

Um invasor, que tenha controle sobre uma botnet, poderá utilizá-la com o objetivo de aumentar o poder em seus ataques, inclusive a outras botnets, de modo a incorporar as máquinas zumbis de outras redes. Vários ataques tem sido reportados utilizando esse tipo de rede, como por exemplo: Envio de *phishing* ou *spam*, ataques de negação de serviço, fraudes bancárias entre outros.

1.2.7. Keyloggers

Keylogger ou *keystroke logger*, São programas utilizados para monitorar e armazenar atividades do teclado de uma determinada máquina. Qualquer tipo de informação inserida, como conteúdo de documentos, credenciais de bancos, número de cartão de crédito, senhas são capturadas e armazenadas localmente ou enviadas para servidores remotos utilizando protocolos como FTP *(File Transfer Protocol)*, HTTP/HTTPS *(HyperText Transfer Protocol Secure)* ou anexo de e-emails. Existem algumas variações desse tipo de malware, que monitora cliques do mouse (mouseloggers) ou efetuam cópias de imagens de tela (screenloggers). Apesar de ser classificado como um *malware*, esse tipo de software pode ter sua utilização legítima, quando uma organização, através de uma política de segurança que preveja seu uso, habilite a funcionalidade e monitore as atividades de uso de sistemas que compõem sua rede interna.

1.2.8. Rootkits

O termo *rootkit* provém do uso em sistemas Unix/Like, onde um atacante, após obter sucesso em comprometer um sistema, instala um conjunto de ferramentas, ou *toolkit* que tem como finalidade esconder a presença maliciosa dos artefatos. Sua utilização também ocorre em sistemas baseado em outras plataformas como o Windows. Apesar do termo

root ser proveniente de sistemas Unix, que neste caso seria mais indicado ter a nomenclatura modificada para *adminkits*, entretanto não é isso que acontece.

Os *rootkits* permitem que um atacante possa controlar remotamente um sistema comprometido, apagar todas os rastros de comprometimento, e esconder sua presença do administrador, de modo que alguns programas são modificados ou simplesmente substituídos por programas modificados que compõem o pacote de ferramentas de um *rootkit*. É tarefa de um *rootkit* capturar outras informações que possam ser utilizados pelo atacante, como senhas, credenciais, outros sistemas que possam ser comprometidos.

1.2.9. Kits de ferramentas

Um atacante possui um conjunto de ferramentas que o auxiliam a identificar sistemas que possam ser comprometidos, de modo a ter o controle de uma rede, total ou parcialmente. Dentre essas ferramentas podemos listar:

Sniffers: Utilizados para monitorar uma rede capturando todos os pacotes do tráfego gerado, tanto wireless quanto redes cabeadas. Alguns tipos de *sniffers* tem a capacidade de interagir com uma rede enviando pacotes, são conhecidos como *sniffers* ativos. Outros são projetados exclusivamente para capturar dados sensíveis como senhas e credenciais, utilizando protocolos mais comuns, como FTP, HTTP, POP3, IMAP *(Internet Message Access Protocol)*, entre outros.

Port Scanners: É um programa utilizado para varrer uma rede buscando serviços que esteja disponíveis através de portas TCP *(Transmission Control Protocol)* ou UDP *(User Datagram Protocol)*, potencialmente localizam um alvo e coletam informações como serviços e versões, permitindo que o atacante identifique serviços vulneráveis.

Scanners de vulnerabilidades: Esse tipo de programa pode ter sua utilização licita, onde uma organização o utiliza para varrer seus sistemas em busca de vulnerabilidades conhecidas, permitindo que seja identificada e corrigida, entretanto um atacante pode utilizar de maneira a encontrar e explorar as vulnerabilidades antes que sejam descobertas pela organização.

Password Crackers: Sistemas que utilizam-se de senhas para proteger sua aplicações são suscetível a ataques de força bruta, onde são testadas um conjunto de senhas aleatórias, ou ataques por dicionário, onde uma lista de palavras são testadas para encontrar qual é utilizada para proteger um segredo.

Um atacante sofisticado pode desenvolver suas próprias ferramentas e ter seu kit exclusivo, já um atacante que possua menos conhecimento, como *script kids*, utilizam-se de ferramentas prontas, de modo a agir com a intenção de invadir ou comprometer um sistema de maneira indiscriminada.

1.3. Método aplicado em forense computacional

A forense computacional é uma parte do processo de resposta a incidentes, geralmente o profissional que atua na área está associado a um centro de resposta a incidentes de segurança (CSIRT)[Mieres 2009]. O objetivo é, através de técnicas de investigação, confirmar

ou descartar uma ocorrência de incidente, permitindo que a organização estabeleça controles eficientes para recuperação e tratamento de evidências, protegendo a privacidade e as políticas da organização.

Toda a atividade é documentada de modo a minimizar interrupções das operações de um sistema, do mesmo modo a preocupação em se manter a integridade das evidências deve ser seguida, evitando-se que as informações sejam contaminadas em sua condução. Como resultado de uma análise, um relatório é produzido de maneira clara a explorar e responder as seguintes indagações: **onde, como, onde e porquê** um incidente ocorreu, de modo a relacionar o incidente a uma entidade, ou seja, **quem** gerou o incidente.

A condução de uma análise é algo complexo, que envolve diversas variáveis e que exigem uma abordagem clara e sem dúvidas quanto aos quesitos solicitados, portanto é importante ter claro os termos envolvidos[Prosise et al. 2003]:

- Preparação: É necessário que exista planos pré incidentes, pois quando o incidente ocorrer existirá um processo a ser seguido.
- Detecção: É necessário ter maneiras de se detectar a ocorrência de um incidente, seja ele através de monitoração ou análise dos eventos nos dispositivos de segurança.
- Resposta: Ao se identificar um incidente é necessário obter respostas sobre os fatos, em uma resposta inicial é necessário que as evidências sejam coletadas, não descartando entrevistas com as pessoas envolvidas.
- Estratégia: É necessário que exista estratégias de ação, que permita definir qual a melhor resposta utilizar mediante as evidências coletadas.
- Duplicação: Alguns casos a duplicação (Cópia pericial) pode ser descartada, entretanto para as análises quem envolvam acesso a imagens originais recomenda-se que a duplicação seja executada.
- Investigação: Deve-se o fato de esclarecer os acontecimentos, buscando responder os quesitos relacionados à análise, principalmente o autor do incidente e suas razões.
- Relatório: Toda análise deve ser concluída com a apresentação de um documento que descreva todos os fatos relevantes à investigação.

O método utilizado por um atacante representa a taxonomia do próprio ataque, sendo possível listar da seguinte maneira:

1. Identificação do alvo;
2. Coleta de informações;
3. Identificação de vulnerabilidades;
4. Comprometimento do sistema;

5. Controle do sistema;
6. Instalação de ferramentas;
7. Remoção de rastros;
8. Manutenção do sistema comprometido.

Quando um atacante identifica seu alvo ele coleta o maior número possível de informações que possam auxiliá-lo na identificação de vulnerabilidades. O comprometimento de um sistema é baseado nas vulnerabilidades encontradas, sendo que o controle é feito através da exploração dessas vulnerabilidades, é nesse ponto que o atacante controla e busca aumentar o nível de privilégio, que tem como objetivo ser o administrador do sistema. A instalação de ferramentas é o passo posterior ao comprometimento, nesta etapa o atacante instala o conjunto de ferramentas que geralmente é composto de um *rootkit* que irá auxiliá-lo a manter o sistema funcional e com suas atividades protegidas do administrador legítimo, onde todos os rastros serão removidos e o ultimo passo é a correção da vulnerabilidade explorada, o atacante deve proteger o sistema de modo que outro atacante não encontre e explore a mesma vulnerabilidade.

As fases que compõem a análise forense são apresentadas pela Figura 1.4.

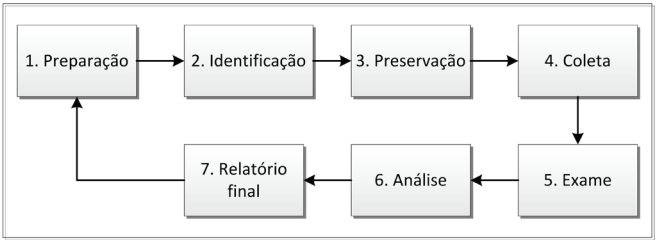

Figura 1.4. Metodologia Forense

A metodologia proposta não é algo fechado, mas que deve ser seguida, sendo a preparação algo inerente ao próprio processo. Um analista forense deve considerar a detecção como primeira passo no tratamento de um incidente, mesmo que essa função não esteja associada a atuação ela deve ser considerada no processo de análise, pois cabe ao analista forense obter a informação de como o incidente foi detectado. A preservação do local deve fazer parte da política de segurança da organização, pois o fato de haver interação com o sistema comprometido pode prejudicar a coleta da evidencias. O exame deve ser feito com objetivos diretos do caso, ou seja, o analista deve possuir os quesitos que deverão ser respondidos através do exame das evidências, e só então emitir o relatório final da análise, é nessa fase que o analista expõe o caso, baseado totalmente em evidências coletadas durante o processo.

O analista forense irá se deparar com dois tipos de cenários: Análise Post mortem e Máquinas vivas.

Post mortem: Este cenário contempla a situação sem atividade no disco rígido onde evidências voláteis estão perdidas e não há atividade do invasor, sendo que não existe a necessidade de contenção do ataque e é provável que algumas evidências foram modificadas ou perdidas.

Máquinas vivas: Neste cenário o equipamento encontra-se em funcionamento e há atividades tanto do sistema quanto de rede, há chances do invasor estar presente e ativo, sendo necessário procedimentos de contenção do ataque e preocupação quanto a modificações e contaminação das evidências.

Para os cenários em que o equipamento encontra-se inativo a fase de coleta é simplificada, entretanto a fase de exame e análise pode ser dificultada, pois uma séria de informações podem ter sido perdidas. Nos cenários em que o equipamento encontra-se ativo a preocupação recai sobre quais processos coletar, portanto a volatilidade é uma importante variável a ser utilizada, na seção 1.3.1 será discutido o tema. Após a definição de ordem de coleta tem-se a decisão a tomar: Desligar o equipamento de maneira elegante (*shutdown*) ou simplesmente cortar a energia (*power-off*).

Para executar o shutdown deve-se considerar o ganho em se manter a integridade do sistema em funcionamento, isso é bastante relevante quando se trata de um banco de dados em funcionamento, entretanto existe a contrapartida, que é alterações em arquivos e dependendo do controle efetuado o atacante pode executar ações na máquina que detectem o desligamento e iniciam outras ações como criptografia de disco ou destruição de informações através de remoção segura de arquivos, conhecido como técnicas de *wipe*[1], que além de apagar um arquivo, sobrescreve a área ocupado com dados aleatórios, dificultando assim a análise.

O corte de energia (*power-off*) não modifica os arquivos, entretanto impacta diretamente no sistema de arquivos do sistema operacional e seus serviços, podendo danificar a estrutura da informação e tornar o sistema inoperante em uma eventual decisão de retornar os sistemas.

Nos dois casos apresentados o analista forense ainda deve se preocupar com sistemas criptografados, atualmente bastante comuns, neste caso a falta de observância deste fator pode inviabilizar totalmente a análise, pois sistemas criptografados enquanto estão ativos ficam disponíveis, assim que desligados tornam-se verdadeiras caixas-pretas. Como exemplo pode ser citado o projeto Truecrypt[2] de criptografia *on-the-fly* de disco, *flashcard* (pendrive) e volumes, sendo totalmente transparente para o usuário, possuindo outras características como esteganografia associada a proteção de volumes criptográficos, conhecido como *Deniability*, protege o usuário caso seja de alguma forma obrigado a informar sua senha de acesso, pois não há como o atacante saber se existe uma ou duas senhas de proteção.

1.3.1. Volatilidade de informações: Coleta e Exame

Apresentar métodos de preservação de dados voláteis, bem como a ordem de volatilidade para a extração e análise dessas informações [Brezinski and Killalea 2002].

[1] `http://wipefreespace.sourceforge.net/`
[2] `http://www.truecrypt.org/`

Incidentes de segurança geram algum tipo de violação a política da organização, portanto devem ser tratados de maneira a minimizar as perdas e maximizar a resposta com o objetivo de fornecer aos administradores do sistema a melhor solução. As provas decorrente do incidentes são eventos muitas vezes voláteis e deve existir orientações sobre a coleta e o arquivamento de provas relevantes. Uma boa prática é que os analistas conheçam e saibam como coletar as evidências consideradas voláteis, portanto protegendo a informação quando a coleta e armazenamento relacionados ao incidente em questão.

Quando um sistema é identificado como comprometido, existem diversas maneiras para acelerar o re-instalação do Sistema Operacional e facilitar a reversão para um estado conhecido, deixando dessa maneira as coisas mais fáceis para os administradores. Entretanto a obtenção de provas são mais difíceis se comparada às facilidades em se recuperar um sistema, pois ainda não existe um modelo que contemple todas as necessidades de uma investigação, e isso pode ser considerado um agravante quando os atacantes conseguem ser mais rápidos que o analista forense.

De acordo com a RFC-3227 [Brezinski and Killalea 2002] que trata exclusivamente da ordem de volatilidade na coleta de evidências, o importante é capturar as informações que não estejam disponíveis caso o equipamento analisado fosse desligado, a Tabela 1.1 apresenta a ordem de acordo com a prioridade de coleta de cada evidência.

Tabela 1.1. Ordem de prioridade para coleta de evidências

Ordem	Tipo	Descrição
1	Registros	Registros do SO
2	Tabelas de roteamento	cache arp, tabela de processos, estatísticas do SO (kernel)
3	Memória	Dump memória RAM (Random Access Memory)
4	Arquivos temporários do SO	Arquivos criados e mantidos temporariamente pelo SO
5	Disco rígido	Cópia *"bit-a-bit"*
6	Logs de monitoração	Registros de dispositivos de rede como proxy, servidores de acesso
7	Configurações físicas	Informações sobre o local e topologia de acesso
8	Dispositivos de armazenamento	cartões de memória, disquetes, pendrives, cameras

1.4. Metodologia de Análise de Malware

Apresentar metodologia de condução de incidente [Mell and karen Kent 2005] que envolve artefatos maliciosos:

- Preparação: Equipe de resposta habilitada a lidar com esse tipo de incidente, com políticas internas que contemplem a atividade;
- Detecção e análise: Detectar, analisar e validar rapidamente um incidente envolvendo malwares;

- Contenção: Interromper e evitar a propagação do malware e prevenir maiores danos aos sistemas;
- Erradicação: Remover o malware dos sistemas infectados;
- Recuperação: Retornar a funcionalidade e os dados dos sistemas infectados, implantando medidas de contenção temporária;
- Pós incidente: Reduzir custos no tratamento dos incidentes utilizando a base de informações adquiridas e sugerindo contramedidas.

A Figura 1.5 apresenta o fluxo de tratamento de um incidente que envolve códigos maliciosos. A detecção é prioritária na condução da análise do incidente, onde a organização deve possuir um modelo que permita a identificação de um evento de segurança de maneira pró ativa. Os passos da metodologia abstrai processos técnicos, mostrando as etapas de forma a permitir a adequação onde esta será utilizada.

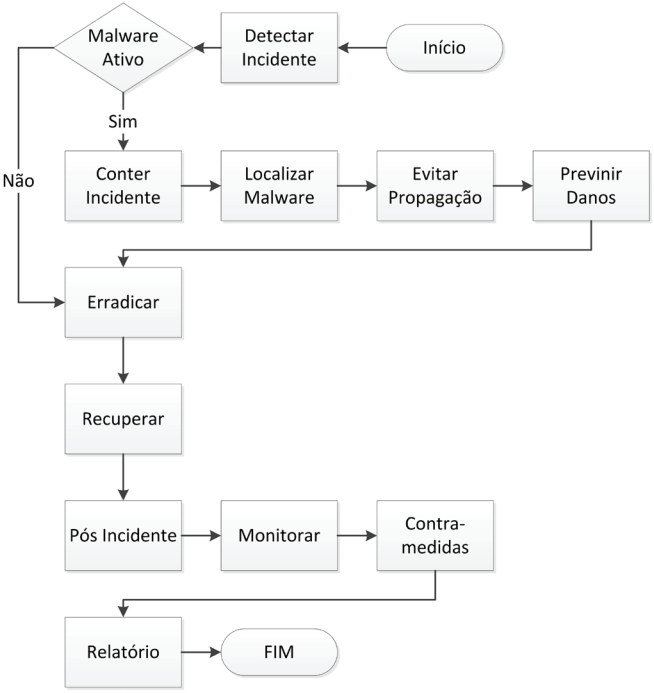

Figura 1.5. Metodologia Análise de Malware

A fase de preparação envolve ações que devem ser seguidas quanto a identificação de um incidente que envolve *malwares*, nesta fase é importante que a organização possua

uma política de segurança que preveja esse tipo de incidente e as atividades de tratamento e mitigação, permitindo a redução do risco de segurança e consequentemente uma menor atividade que envolva a exploração desse tipo de vulnerabilidade. A rapidez em se detectar as atividades de um *malware* tornam a resposta mais eficientes, pois possibilita a erradicação e recuperação com um escopo menor do que seria em um sistema totalmente comprometido, permitindo assim, diminuir o número de sistemas comprometidos.

Organizações que possuem um grupo de resposta a incidentes de segurança, convém que sejam adotados métodos para se detectar infecção de *malwares*, nesses métodos é importante que os profissionais que serão responsáveis por conduzir a análise e tratamento do incidente possuam uma sólida formação em identificação e classificação que envolve a segurança dos ativos da empresa.

A forma clássica para se detectar um *malware* é a utilização de softwares como antivírus, sistemas de detecção de intrusão baseados em análise de redes ou análise de hosts monitorados (Intrusion Detection System - IDS) e (Host Intrusion Detection System - HIDS), bem como softwares que detectam e removem *spywares* ou qualquer outro tipo de ferramenta que possa ser utilizada pela organização. Convém que um profissional tenha competências listadas no capítulo 1.3 para em casos em que um malware possua características mais complexas e que requeiram uma análise mais detalhada.

A metodologia adotada deve prever treinamentos a usuários do sistema da organização, de modo a que todos tenham conhecimento dos riscos, fazendo-se as recomendações de segurança que devem constar na política de segurança da organização.

1.5. Análise de Metatados

Informações disponíveis no código compilado: Bibliotecas utilizadas, módulos estáticos e dinâmicos, compiladores, sistemas operacionais 32 ou 64 bits, versões e outras informações importantes.

O termo *metadado* refere-se a informações sobre o próprio dado, possui diversos contextos como informações de programas compilados que estão armazenadas internamente ao código compilado e apenas é revelado quando solicitado ou extraído pelo analista forense. Cada tipo de arquivo possui informações relacionadas ao conteúdo, como tipo de compilador utilizado, bibliotecas compiladas junto ao código, identificadores únicos, tipos de sistemas operacionais compatíveis, versões 32 ou 64 bits de execução e outras variáveis.

Extrair as informações que estão ocultas é apenas uma questão de saber o que está procurando. Existem diversas ferramentas que se propõem em resolver essa tarefa, entretanto em alguns casos pode ser que o analista não tenha certeza de qual informação está buscando, esses casos geralmente são associados quando não existe ainda um incidente relatado, portanto o especialista está em busca de evidências que podem sugerir um incidente. Diversos formatos de arquivos permitem que informações sejam "escondidas"ou alteradas, particularmente um arquivo possui um formato de codificação em que este deverá ser armazenado, cabendo ao analista saber diferenciar esses tipos de arquivos.

Para cada sistema operacional existe um método que é utilizado para determinar o formato de um arquivo específico, além dos metadados é possível utilizar informa-

ções encontradas no cabeçalho (file header) desses arquivos. Nesta abordagem é possível identificar o tipo do arquivo analisado independente de sua extensão, permitindo que seja utilizada a melhor estratégia para se recuperar os metadados de acordo com o formato identificado.

Na Listagem 1.3 apresenta-se a extração da informação do cabeçalho do arquivo, neste caso temos dois arquivos hexdump.exe e winhex.exe de tamanhos diferentes e mesma extensão. Para este exemplo utilizou-se a ferramenta *hexdump* [3].

Listagem 1.3. Dump de arquivos EXE

```
1  hexdump.exe   71.168 bytes    13/06/2005   12:35:32
2
3   -0 -1 -2 -3   -4 -5 -6 -7   -8 -9 -A -B   -C -D -E -F
4
5   4D 5A 90 00   03 00 00 00   04 00 00 00   FF FF 00 00
6   B8 00 00 00   00 00 00 00   40 00 00 00   00 00 00 00
7   00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
8   00 00 00 00   00 00 00 00   00 00 00 00   80 00 00 00
9   0E 1F BA 0E   00 B4 09 CD   21 B8 01 4C   CD 21 54 68
10
11 winhex.exe   2.034.688 bytes    18/05/2011   16:00:00
12
13  -0 -1 -2 -3   -4 -5 -6 -7   -8 -9 -A -B   -C -D -E -F
14
15  4D 5A 50 00   02 00 00 00   04 00 0F 00   FF FF 00 00
16  B8 00 00 00   00 00 00 00   40 00 1A 00   00 00 00 00
17  00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00
18  00 00 00 00   00 00 00 00   00 00 00 00   00 01 00 00
19  BA 10 00 0E   1F B4 09 CD   21 B8 01 4C   CD 21 90 90
```

Comparando os dois arquivos é possível determinar sua semelhança completa da linha 5 e 15 onde cada byte, representado pela informação em hexadecimal, permanece a mesma para os tipos de arquivos EXE nos sistemas operacionais windows, pois não necessariamente o que defini um executável desse tipo seja sua extensão. É importante considerar que um atacante pode alterar a extensão de um arquivo para outra menos crítica na tentativa de enganar sua vitima ou mesmo os sistemas de anti virus que analisam arquivos de acordo com sua extensão, muito utilizado em sistemas web mail de empresas.

Os dois primeiros bytes (4*D*5*A*) representam um arquivo executável *Windows/-DOS*, são utilizados para identificar as seguintes extensões: COM, DLL, DRV, EXE, PIF, QTS, QTX e SYS.

Esta técnica de identificação é conhecida como *magic numbers*, que é uma maneira de incorporar metadados muitas vezes associada com o Unix e seus derivados, inserindo um "número mágico"dentro do próprio arquivo. Originalmente, esta técnica foi usado para um conjunto específico de 2 bytes identificadores, inseridos sempre no começo de um arquivo, entretanto qualquer sequência pode ser considerada como um número válido, possibilitando que um formato de arquivo seja identificado de maneira exclusiva. Portanto possibilitar a detecção dessas variáveis é uma maneira simples e eficaz de distinguir entre diversos tipos de formatos de arquivos, facilitando ao sistema operacional, a

[3] http://www.richpasco.org/

obtenção de informações importantes. A Tabela 1.2 apresenta algumas assinaturas mais comuns, a lista completa e atualizada está disponível[4].

Tabela 1.2. Lista parcial de assinaturas de arquivos (*magic numbers*)

Assinatura	Tipo	Descrição
50 4B 03 04	ZIP	Compactador PKZIP
37 7A BC AF 27 1C	7Z	Compactador 7-Zip
50 4B 03 04 14 00 06 00	DOCX, PPTX, XLSX	Microsoft Office
50 4B 03 04 14 00 08 00 08 00	JAR	Arquivo Java
7B 5C 72 74 66 31	RTF	Texto (Rich text format)
49 44 33	MP3	Arquivo de audio (MPEG-1 Audio Layer 3)
7F 45 4C 46	ELF	Arquivo executável (Linux/Unix)
89 50 4E 47 0D 0A 1A 0A	PNG	Imagem (Portable Network Graphics)
00 00 00 18 66 74 79 70 33 67 70 35	MP4	Arquivo video (MPEG-4 video)
25 50 44 46	PDF, FDF	Adobe (Portable Document Format)

Existem diversas ferramentas que permitem identificar os tipos de arquivos baseado em assinaturas, é conveniente que o analista conheça como é feito a identificação e só então utilize a aplicação que melhor atenda suas necessidades. Neste curso recomendamos a utilização de ferramentas livres com códigos abertos, como *file*, que é um comando padrão dos sistemas Linux. Existem diversos projetos de portar as ferramentas de linha de comando do Linux para Windows, como é o caso do projeto *gnu utilities for windows*[5].

Por ser um padrão utilizado para definir os tipos de arquivos, a identificação simplesmente baseada em uma sequência binários pode ser aproveitadas pelos adversários para explorar vulnerabilidades, inserindo características de outros tipos de arquivos em arquivos aparentemente inofensivo, como é o caso de arquivos de formato de documentos como *PDF*, que podem ser alterados de maneira a explorar falhas em softwares. Nesse entendimento é necessário conhecer técnicas que não sejam somente as clássicas.

1.6. Códigos Ofuscados

Métodos utilizados pelos desenvolvedores para dificultar a análise e detecção pelos antivírus. Será apresentado métodos de ofuscação baseado em packers (compactadores de executável), criptografia, *blinders*, *joiners* e *wrappers*.

[4] http://www.garykessler.net/library/file_sigs.html
[5] http://unxutils.sourceforge.net/

1.6.1. *Packers*

Packers ou *compressors* são técnicas utilizadas para ofuscação, *packing*, de código. Basicamente atuam comprimindo ou cifrando executáveis originais, buscando dificultar a identificação pelos antivirus e análise estática dos códigos por parte do analista. Porém, enquanto compactados, os programas não desempenham as suas funcionalidades primárias. Assim, uma rotina de descompressão, normalmente localizada ao final do arquivo, é necessária para descompressão antes de serem carregados à memória. A Figura 1.6 apresenta as etapas de codificação e decodificação de um arquivos executável.

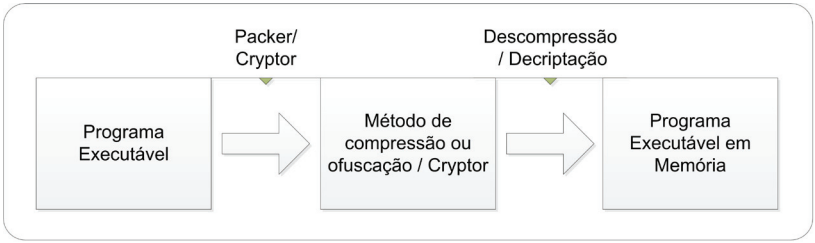

Figura 1.6. Criação e execução de arquivos compactados ou encriptados

Essa descompressão ocorre, normalmente, por meio de rotinas específicas para determinados *packers*, Tabela 1.3, salvos raros casos em que os programas responsáveis por comprimir também possuem funções nativas para descompressão. Nesse ponto, o analista deve tomar cuidado, uma vez que o programa por ele utilizado para tentar descomprimir determinado *malware* pode ser desde um programa falso, assim sendo ele nunca alcançará êxito na descompressão, até *malwares* em si, cujo o objetivo é comprometer as máquinas de possíveis analistas.

Tabela 1.3. *Packers* e seus respectivos *unpackers*

Packers	**Unpackers**
ASPack	ASPackDie
MEW	UnMew
FSG	UnFSG
PECompact	UnPECompact

Em alguns casos, o código de algum malware pode ser ofuscado utilizando packers diferentes, gerando polimorfismo [Martins et al. 2010], pois para cada código uma assinatura diferente é necessária para sua presença seja detectada.

Existem algumas técnicas para se detectar a presença de packers em códigos executáveis, as mais conhecidas são as que analisam a estrutura de arquivos que utilizam a estrutura *portable executable*, que foi projetado para suportar versões dos sistemas operacionais da Microsoft.

O projeto PEiD[6], que atualmente encontra-se suspenso, tem a capacidade de detectar mais de 600 tipos diferentes de ofuscadores packers, cryptors e compiladores em arquivos executáveis portáveis. Para detecções de arquivos modificados e desconhecidos é utilizado técnicas heurísticas, que aumentam a possibilidade de falsos positivos.

TrID[7] é um utilitário desenvolvido para identificar os tipos de arquivo a partir de suas assinaturas binárias, atualmente identifica cerca de 4410 arquivos diferentes. Embora haja utilitários similares a ferramenta não tem regras fixas, em vez disso, pode reconhecer novos formatos de arquivos de forma automática. Para isso possui uma base de assinaturas e utiliza a probabilidade para calcular a proximidade de um determinado arquivo a um tipo específico.

Para análises estáticas o analista deve ter a preocupação em identificar se existe ofuscador ou não, entretanto, nos casos em que a análise seja executado utilizando o método dinâmico é possível realizar uma cópia *dump* do processo em memória, uma vez que quando é carregado, todas as suas funcionalidades estão ativas. Os processos de análise serão discutidos no capítulo 1.8.

A identificação do tipo de ofuscador utilizado auxiliará a análise em busca de strings e padrões entre os tipos de *malwares*, a Tabela 1.3 apresenta alguns tipos de ferramentas para esse tipo de atividade, entretanto, mesmo que não seja possível executar o processo inverso do ofuscador, o analista pode seguir com o trabalho, neste caso em um ambiente controlado o código malicioso será executado e as atividades monitoradas armazenadas, a análise dinâmica será detalhada no Cenário 1.8.3.

1.6.2. *Cryptors*

Criptografar um executável é o objetivo de um *Cryptor*, também conhecido como *encryptors* ou *protectors*. Esta técnica possuiu características similares aos *packers*, discutido na sessão 1.6.1, porém alcançado de uma maneira diferente. Ao invés de compactar o arquivo executável, um *Cryptor* simplesmente utiliza de algoritmos criptográficos para ofuscar um arquivo, resultando em dificultar a análise estática e sua identificação por softwares antivírus ou IDSs (*Intrusion Detection System*). Esse tipo de técnica torna difícil a análise através de engenharia reversa. Assim como os packers, criptografar um executável torna necessário acoplar ao código uma função inversa de decifração, que funciona em tempo de execução ao programa original, onde todo o processo é feito em memória.

Uma lista com os packers, unpackers e cryptors mais comuns pode ser encontrada no Anexo 1.11 Tabela 1.7.

1.6.3. *Binders, Joiners e Wrappers*

Binders são utilizados para ofuscar códigos com propriedades similares aos packers e cryptors, com o objetivo de dificultar sua identificação e análise. Joiners são utilizados para combinar dois arquivos em um, mantendo a propriedade de execução simultânea, isso é ideal quando um atacante envia um programa licito e junto insere um malware (ex. Keylogger) que pode ser instalado pela vitima sem qualquer tipo de interação, esse

[6] http://www.peid.info/
[7] http://mark0.net/

método é conhecido como instalação silenciosa.

1.7. Funções de *Hashing*

Neste capítulo será apresentado os modelos de funções de hash utilizado na comparação de novos malwares, e sistema de indexação por similaridade de arquivos, que utiliza um processo conhecido como *context triggered piecewise hashes* (CTPH) ou *fuzzy hashing* [Kornblum 2006], auxiliando assim a análise por similaridade.

As funções de *hash* são algoritmos desenvolvidos para verificação de integridade de um arquivo, de modo que dois arquivos diferentes tenha saídas diferentes. O algoritmo é conhecido como uma função unidirecional, onde a entrada variável gera uma saída fixa de tamanho suportado pelo algoritmo, ou seja, com base no resultado de um cálculo não é possível obter o texto de entrada.

Para efeito da pericia forense e análise de malware, as funções de hash são indispensáveis, pois é uma maneira rápida e barata para se comparar arquivos. Não é intenção deste minicurso apresentar o algoritmo em si, mas seu resultado como um todo, pois sem ele não seria possível garantir que a informação está integra.

As funções SHA *(Secure Hash Algorithm)*[Jones 2001] e MD5 *(message-digest algorithm)*[Rivest 1992] são funções de hash que atendem os requisitos de segurança provendo uma baixa probabilidade de colisões, que são resultados de dois arquivos diferentes com a mesma saída. Para que exista uma maior conformidade de segurança é recomendado que o analista utilize mais de uma função simultaneamente.

A ferramenta hashdeep[8] possui a capacidade de computar, comparar, ou verificar vários arquivos, como resultado tem-se o exemplo apresentado na Tabela 1.4.

Tabela 1.4. Funções *hashing*

Função	Saída	Tamanho
MD5	79970AF07E4FDD8AAA6AFC3EFFF1B5D5	128 bits
SHA1	3A7FE56DAE27BBF7EDCB4C7C839D5EDBA46A05A8	160 bits
SHA256	9949EB0D844B2AB6A1623C1EA69CBE1CEDC19BABBFAE6BC24865DE5B4E18A681	256 bits

As funções de *hashing* facilitam a busca por arquivos, pois se algum artefato é encontrado e seu hash calculado, é possível efetuar uma busca por um mesmo resultado independente de nome ou extensão.

Em casos em que um arquivo seja alterado o resultado será diferente para cada alteração, entretanto é uma técnica conhecida pelos atacantes, que podem efetuar pequenas alterações com o objetivo de que não sejam reconhecidos. Portanto é necessário ir além dessa técnica.

Context Triggered Piecewise Hash(CTPH)[Kornblum 2006] ou simplesmente *fuzzy hashing* foi desenvolvido para identificar essas pequenas alterações. O técnica apresenta o conceito de comparar arquivos pela semelhança do contexto, independente do tipo de função de hash utilizado.

Para os casos de análise de malware, é possível determinar o grau de semelhança

[8] `http://md5deep.sourceforge.net/`

analisando *dumps* de memória dos processos gerados pelos artefatos maliciosos. Para efeitos práticos a ferramenta *ssdeep* [9] implementa as funções esperadas de *fuzzy hashing*.

Listagem 1.4. Utilização ferramenta ssdeep

```
1 ssdeep.exe -b teste.c >ssdeep.hash
2 ssdeep,1.1--blocksize:hash:hash,filename
3 48:a/mssIVGFPE3Z/iPYmYDFjf7KkItzWWsxCAw7fwTwpCS0sbhxdL:omsVVeMQAZDIAPw7
    oTwp50slD,"teste.c"
4 ssdeep.exe -bm ssdeep.hash teste1.c
5 teste1.c matches ssdeep.hash:teste.c (93)
```

A Listagem 1.4 apresenta a ferramenta *ssdeep*. Na linha 1 é criado a base de informações armazenando o resultado no arquivo *ssdeep.hash*, o conteúdo deste arquivo é apresentado pelas linhas 2 e 3, uma alteração é feita propositadamente no arquivo analisado *teste.c* e salvo em *teste1.c*. A linha 4 apresenta o comando para comparação e finaliza na linha 5 mostrando que o arquivo *teste1.c* é 93% similar ao arquivo *teste.c*.

1.8. Preparando uma imagem para análise

Ao cursista será entregue uma imagem composta por sistema operacional funcional, preparado em ambiente virtualizado[10], com todas as ferramentas necessárias para iniciar os trabalhos de análise. Neste capítulo serão apresentados dois estudos de casos reais.

1.8.1. Estudo de caso: Analisando códigos maliciosos

A etapa prática busca aplicar os conhecimentos iniciais adquiridos no minicurso em análises de casos reais. Neste intuito serão apresentados dois cenários distintos e mais comuns.

Cenário 1. *Análise forense de máquina comprometida por malware.*

Neste cenário, a máquina a ser analisada, encontra-se comprometida, onde o programa malicioso está totalmente funcional e em memória. Através de técnicas forenses [Aquilina et al. 2008], o analista deve ser capaz de identificar quais procedimentos a seguir na captura e análise, tendo como premissa inicial, encontrar o vetor de comprometimento e atividades que possam ser consideras suspeitas.

Cenário 2. *Análise de código malicioso capturado.*

Neste cenário o analista terá acesso a um código malicioso disponibilizado para os trabalhos. Utilizando técnicas de análise de um *phishing* discutido no capítulo 1.2.1 através da análise da Listagem 1.1, preparando o analista para responder a um incidente de maneira eficiente, analisando o código e as interações com a máquina alvo, no intuito de investigar quais informações são coletadas pelo *malware* e qual o destino destas informações.

[9] http://ssdeep.sourceforge.net/
[10] http://www.virtualbox.org/

1.8.2. Cenário 1: Análise forense de máquina comprometida por malware

O objetivo do analista nesse cenário será investigar se a máquina em questão está infectada por um *malware* ou não.

Identificando e encontrando a ameaça

Como o analista não sabe exatamente a que tipo de código malicioso a máquina em questão foi exposta, cabe a ele buscar por indícios que corroborem a identificação tanto do tipo da ameaça em questão quanto do local onde o código está armazenado.

Alguns fatores que influenciam a análise, e que devem ser seguidas na localização e identificação dos códigos maliciosos são[Aquilina et al. 2008]:

- Processo de origem desconhecida;
- Processos que não são familiares ao analista;
- Processos familiares, porém localizados em locais não usuais do sistema;
- Processos que parecem ser familiares, porém com pequenos erros de grafia;
- Processos que já foram identificados como suspeitos, por originar comportamentos anômalos, em outras etapas de análise, como na captura do tráfego de rede.

É bastante comum os atacantes tentarem esconder a presença do artefato de sua vitima, algumas técnicas geralmente associam processos conhecidos do sistema operacional com outros nomes similares ou em locais diferente do normal. Existem tentativas de se identificar quais processos através do nome em execução, como é o caso do projeto *Process Library*[11].

Uma característica dos malwares para a plataforma de sistema operacional *Windows*, é que a maioria necessita ser capaz de se manter em execução quando o sistema é desligado ou reiniciado, dessa maneira a máquina estará sempre sob o controle do atacante. Portanto é muito comum, que ao infectar um equipamento, o malware deve inserir chaves no registro do sistema, permitindo que em qualquer situação este seja executado. O analista deve conhecer registros mais utilizados e ferramentas que o auxiliem a obter essa informação.

A tática utilizada por diversos *malwares* é armazenar seu código em diretórios, chaves de registro ou arquivos específicos no qual o Windows busca por aplicativos que são inicializados a cada sessão, os chamados *autoruns*. Essa é uma maneira eficaz de um processo se manter em execução sem que o usuário tenha conhecimento. Algumas ferramentas que auxiliam na detecção de programas *autorun* são [Aquilina et al. 2008]:

Autoruns [12]: Ferramenta com interface gráfica (GUI) ou de linha de comando (CUI) que mostra quais programas estão configurados para serem executados durante a inicialização do sistema ou login;

[11] http://www.processlibrary.com/
[12] http://www.sysinternals.com/

WhatInStartup [13]: Utilitário exibe a lista de todos os aplicativos que são carregados automaticamente.

O analista deve ser capaz de diferenciar quais programas representam, de fato uma ameaça, uma vez que *autoruns* são extremamente comuns. É necessário uma busca por indícios que apontem para uma atividade suspeita de qualquer processo em execução no momento da análise. Uma recomendação é que a busca inicie pela análise dos processos executados a partir de diretórios diferentes dos esperados para cada processo, como por exemplo, Internet Explorer sendo executado a partir da pasta pessoal do usuário, ou que o nome do processo seja diferente do esperado.

Segundo a documentação da Microsoft, existem quatro chaves *Run* no registro que faz com que determinados programas sejam executados automaticamente, neste caso a análise se baseia nos registros do sistema operacional Windows XP.

Tabela 1.5. Chaves de registro - *autoruns*

Registro	Descrição
`HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`	Os comandos explicitados nessa chave são invocados a cada *login* do usuário.
`HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run`	Os comandos explicitados nessa chave são invocados a cada *login* do usuário.
`HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce`	Após a execução dos comandos presentes nessas chaves, elas são apagadas.
`HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce`	Após a execução dos comandos presentes nessas chaves, elas são apagadas.

Cada chave da Tabela 1.5 contém um conjunto de valores e cada valor é uma linha de comando. Dessa forma, caso o *malware* altere o valor original de uma chave e adicione sua localização através da variável *path*, ele será executado sem que haja necessidade de interação com o usuário, que neste caso passa a ser sua vitima.

Chaves *RunOnce* possuem algumas características que podem ser utilizadas por *malwares*:

- As chaves desse tipo são executadas de maneira sequencial, fazendo com que o Windows Explorer aguarde o término das mesmas, antes de seguir com o *logon* habitual, isso permite que o *malware* execute rotinas antes mesmo de o usuário realizar o processo de *logon*. Dessa maneira, após a autenticação do usuário, o ambiente pode já estar comprometido.
- Chaves *Run* e *RunOnce* são ignoradas em modo de segurança, porém, caso um asterisco $*$ seja acrescentado ao início do valor, chaves *RunOnce* podem ser executadas

[13] `http://www.nirsoft.net/`

mesmo em tal modo: dessa maneira o *malware* tem mais uma garantia de que será executado.

A utilização do *Autoruns*, Figura 1.7, permite ainda uma outra maneira de se identificar processos suspeitos. A Ferramenta disponibiliza a função *jump*, que direciona o analista ao diretório onde se encontra o arquivo executável referente ao processo. Por meio desse diretório, o arquivo suspeito pode ser capturado e enviado para testes em serviços de antivirus online, como o virustotal[14], que possui uma base atualizada de mais de 40 tipos diferentes de antivirus, sendo possível efetuar pesquisas da data que o artefato foi enviado e caso tenha sido considerado malicioso, apresenta a quantidade de antivirus que o detectam.

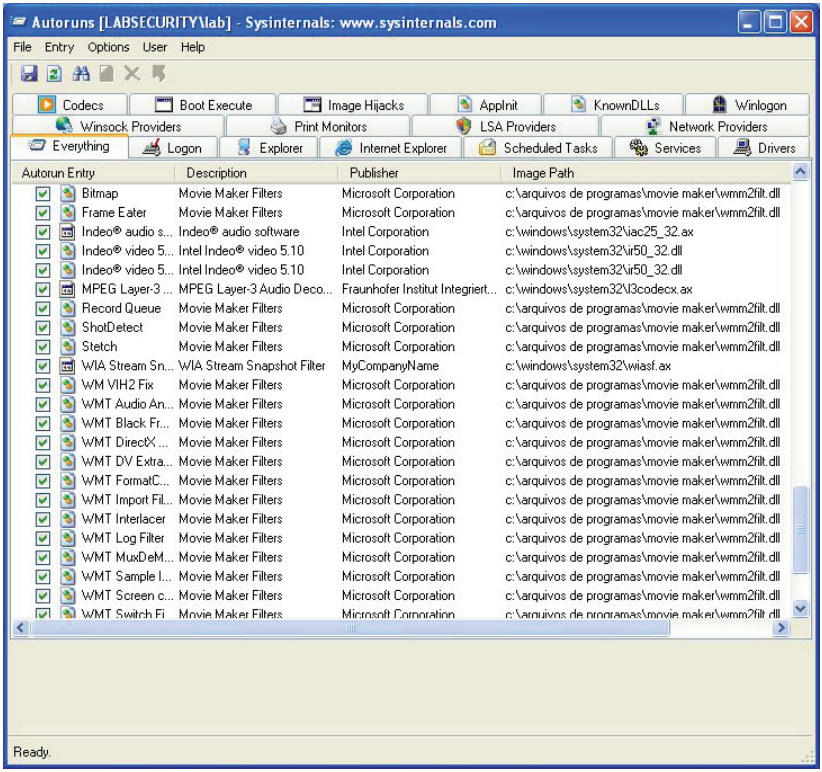

Figura 1.7. Execução do programa *Autoruns*

É possível que o atacante tente enganar sua vitima fazendo pequenas alterações no nome do processo, mas isso dificilmente enganará um expert que deve conhecer os

[14] http://www.virustotal.com

processos mais comuns do sistema operacional em que trabalha. A ferramenta também permite que seja listada as bibliotecas (DLL's) que cada processo utiliza, localizando diferenças e identificando ataques conhecidos como *dll injection* que é uma técnica usada para executar código no espaço de endereço de memória de outro processo, forçando a carregar uma biblioteca dinâmica. Uma lista das bibliotecas do sistema pode ser obtida através da ferramenta *ListDLLs*[15], permitindo que se identifique arquivos e suas respectivas bibliotecas associadas que estejam fora do padrão esperado.

Para que o analista tenha uma capacidade maior de identificação de processos, existe a ferramenta *HijackThis*, que permite ao analista rapidamente coletar as chaves de registros que aproveitam a execução automática de programas. Essa ferramenta analisa o sistema e gera como saída um arquivo de *log* de todas as chaves de registro que podem ser exploradas pelos malwares a fim de se manterem em execução, a Listagem 1.5 apresenta o resultado de uma coleta.

Listagem 1.5. Log gerado pelo programa HijackThis

```
1  Logfile of Trend Micro HijackThis v2.0.4
2  Scan saved at 16:17:42, on 23/9/2011
3  Platform: Windows XP SP2 (WinNT 5.01.2600)
4  MSIE: Internet Explorer v6.00 SP2 (6.00.2900.2180)
5  Boot mode: Normal
6  Running processes:
7  C:\WINDOWS\System32\smss.exe
8  C:\WINDOWS\system32\winlogon.exe
9  C:\WINDOWS\system32\services.exe
10 C:\WINDOWS\system32\lsass.exe
11 C:\WINDOWS\system32\VBoxService.exe
12 C:\WINDOWS\system32\svchost.exe
13 C:\WINDOWS\System32\svchost.exe
14 C:\WINDOWS\system32\spoolsv.exe
15 C:\WINDOWS\Explorer.EXE
16 C:\WINDOWS\system32\VBoxTray.exe
17 C:\Documents and Settings\lab\Desktop\Nova pasta\voegol.com.exe
18 C:\Arquivos de programas\Internet Explorer\iexplore.exe
19 C:\Documents and Settings\lab\Desktop\HijackThis.exe
20 R1 - HKCU\Software\Microsoft\Internet Explorer\Main,Search Page = &http
       ://home.microsoft.com/intl/br/access/allinone.asp
21 O2 - BHO: AcroIEHelperStub - {18DF081C-E8AD-4283-A596-FA578C2EBDC3} - C
       :\Arquivos de programas\Arquivos comuns\Adobe\Acrobat\ActiveX\
       AcroIEHelperShim.dll
22 O4 - HKLM\..\Run: [] C:\WINDOWS\system\sistem.exe
23 O4 - HKLM\..\Run: [Adobe Reader Speed Launcher] "C:\Arquivos de
       programas\Adobe\Reader 9.0\Reader\Reader_sl.exe"
24 O4 - HKCU\..\Run: [Google Update] "C:\Documents and Settings\lab\
       çõConfiguraes locais\Dados de aplicativos\Google\Update\GoogleUpdate
       .exe" /c
25 O4 - HKUS\S-1-5-19\..\Run: [CTFMON.EXE] C:\WINDOWS\system32\CTFMON.EXE
       (User 'LOCAL SERVICE')
26 O4 - HKUS\S-1-5-20\..\Run: [CTFMON.EXE] C:\WINDOWS\system32\CTFMON.EXE
       (User 'NETWORK SERVICE')
27 O4 - HKUS\S-1-5-18\..\Run: [CTFMON.EXE] C:\WINDOWS\system32\CTFMON.EXE
```

[15] www.sysinternals.com

```
    (User 'SYSTEM')
28  O4 - HKUS\.DEFAULT\..\Run: [CTFMON.EXE] C:\WINDOWS\system32\CTFMON.EXE
    (User 'Default user')
29  O9 - Extra button: Messenger - {FB5F1910-F110-11d2-BB9E-00C04F795683} -
     C:\Arquivos de programas\Messenger\msmsgs.exe
30  O9 - Extra 'Tools' menuitem: Windows Messenger - {FB5F1910-F110-11d2-BB
    9E-00C04F795683} - C:\Arquivos de programas\Messenger\msmsgs.exe
31  O14 - IERESET.INF: SEARCH_PAGE_URL=&http://home.microsoft.com/intl/br/
    access/allinone.asp
32  O16 - DPF: {E2883E8F-472F-4FB0-9522-AC9BF37916A7} - http://platformdl.
    adobe.com/NOS/getPlusPlus/1.6/gp.cab
33  O22 - SharedTaskScheduler: éPr-carregador Browseui - {438755C2-A8BA-11D
    1-B96B-00A0C90312E1} - C:\WINDOWS\system32\browseui.dll
34  O22 - SharedTaskScheduler: Daemon de cache de categorias de componente
    - {8C7461EF-2B13-11d2-BE35-3078302C2030} - C:\WINDOWS\system32\
    browseui.dll
35  O23 - Service: Remote Packet Capture Protocol v.0 (experimental) (
    rpcapd) - CACE Technologies, Inc. - C:\Arquivos de programas\
    WinPcap\rpcapd.exe
36  O23 - Service: VirtualBox Guest Additions Service (VBoxService) - Sun
    Microsystems, Inc. - C:\WINDOWS\system32\VBoxService.exe
37  --
38  End of file - 2727 bytes
```

As linhas 7 a 19 apresentam informações dos processos em memória, permitindo que se identifique qualquer atividade fora do padrão esperado, por isso é importante que o analista conheça os principais processos do sistema analisado. Entretanto essa informação ainda não é definitiva, pois um código malicioso pode facilmente substituir um processo válido por outro processo que irá desempenhar uma função dupla. Pode-se perceber que a linha 17 apresenta um processo suspeito, pois é um arquivo executável em uma pasta não convencional. Seguindo a análise, na linha 22 outro arquivo suspeito *sistem.exe*, grafias erradas de arquivos de comandos lícitos é uma tática comum que os atacante utilizam para nomear os malwares, pois um usuário muitas vezes desatento não irá perceber que o comando está errado, simplesmente passa desapercebido.

Para que as coisas sejam facilitadas é possível submeter o arquivo de *log* para análise remota, através do site do desenvolvedor `http://www.hijackthis.de` o *log* obtido é analisado de acordo com o padrão esperado de um arquivo desse tipo, a Figura 1.8 apresenta o resultado. O analista ao examinar o resultado deve estar atento para os processos desconhecidos *unknown process* que geralmente é marcado com um caracter "?"e para os arquivos sujos *nasty*, geralmente marcado em vermelho com o caracter "*X*".

Com base em um *pool* de ameaças e programas já conhecidos, os programas que estavam em execução na máquina alvo são identificados e classificados de acordo com sua periculosidade. Processos marcados como desconhecidos ou perigosos já são um bom ponto de partida para a identificação do *malware*.

Captura do código malicioso: Uma vez que o código malicioso tenha sido identificado, não é indicado que o analista realize qualquer tipo de interação com o mesmo, uma vez que se tem pouco ou nenhum controle sobre o estado da máquina e sobre a execução do programa.

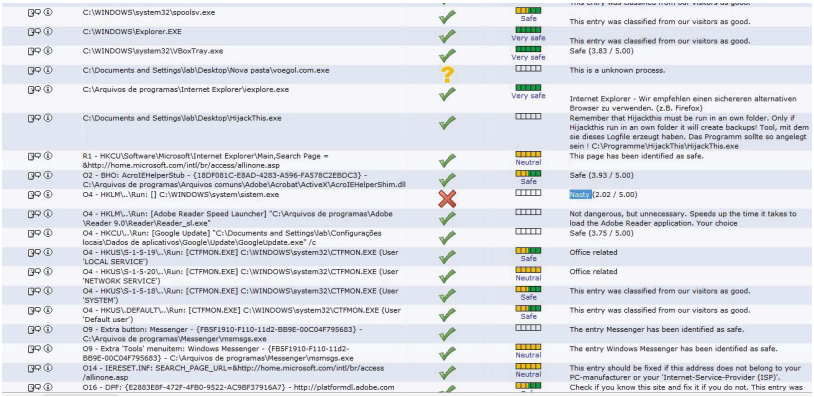

Figura 1.8. Análise do log gerado pelo HijackThis

O analista pode optar por extraí-lo da máquina infectada e transportá-lo a uma máquina específica na qual será possível executar uma análise mais profunda e controlada. Isso ocorre porque, em um ambiente mais controlado, o analista poderá utilizar, por exemplo, *snapshots* do sistema, comparando o estado anterior e posterior da execução do *malware*. Uma vez que o *malware* seja extraído, pode-se seguir os passos descritos no Cenário 2.

1.8.3. Cenário 2: Análise de código malicioso capturado

O objetivo desse cenário é preparar o analista para a investigação de um código malicioso.

Como um *malware* é um código potencialmente perigoso, devem ser tomadas medidas de segurança para que a análise desse código não comprometa ainda mais a máquina alvo e, tão pouco, infecte outros *hosts*. Por esse motivo, recomenda-se que, sempre que possível, a análise de um código malicioso seja executada em um ambiente isolado, como uma máquina virtual, ou em *Online Sandboxes* como é o caso do projeto *Anubis* [16] e outros projetos como *threatexpert* [17] e *joesecurity* [18].

Neste cenário a análise será dividida em duas etapas distintas:

- **Análise estática**: Análise do código sem execução.
- **Análise dinâmica**: Análise do código através de sua execução e monitoração das interações.

[16] http://anubis.iseclab.org/
[17] http://www.threatexpert.com/
[18] http://www.joesecurity.org/

1.8.4. Análise estática

Análise estática é geralmente mais eficiente na detecção de malware, porque usa informações de alto nível. Nessa etapa da análise, tentaremos extrair o maior número de informações úteis - IP's, *paths* de arquivos, *logins* e senhas, entre outros - do código em si do objeto.

Poderão ser encontradas duas situações distintas:

- Quando o código não encontra-se ofuscado, logo seu código encontra-se em linguagem natural e as informações podem ser coletadas de maneira simples, procurando-as diretamente no código.

- Quando o código foi ofuscado, ou seja, foi utilizada uma técnica de ofuscamento e o código foi "compactado". Nesse caso pouca ou nenhuma informação pode ser obtida sem que o código seja descompactado.

Independentemente de o código ter sido ofuscado ou não, deve-se ser capaz de garantir a consistência do arquivo analisado, bem como sua unicidade. Isso é importante para determinar se o *malware* alterou seu próprio código durante a execução, ou pode ser necessário provar que o código não foi alterado durante a análise.

Para isso, utilizaremos *hashing functions*. Uma função *hash* é uma função unidirecional, que gera uma string única (salvo alguns casos de colisões), para cada entrada, e de tamanho fixo. Aqui utilizaremos o MD5 (*Message Digest 5*). O programa responsável por realizar essa conversão será o *md5sum*. Esse programa recebe como entrada determinado arquivo e gera, como saída, uma string de 128 bits em hexadecimal, como mostrado na Listagem 1.6. Na primeira 1, executa-se o comando *md5sum* e redirecionamos o seu resultado a um arquivo texto e a linha 2 apresenta o conteúdo do arquivo texto *voegol.txt*

Listagem 1.6. Utilização do programa md5sum

```
1 C:\Trojans> md5sum voegol.com.exe > voegol.txt
2 02e44077c7b43040683ce2434f81d563 *voegol.com.exe
```

Uma vez que se tem um identificador para o *malware*, é verificado se existe o mesmo *hash* na base de dados do analista, ou se existe similaridade com outro malware analisado anteriormente, para o caso de similaridade o capítulo 1.7 apresenta a discussão do assunto. Caso não exista padrões encontrados a análise do código é iniciada.

O primeiro passo é obter uma lista de *strings* do código. A ferramenta *strings* da *sysinternals* recebe como entrada um arquivo e produz uma lista. Caso poucas ou nenhuma string tenham sido encontradas, é provável que o malware esteja utilizando alguma técnica para ofuscamento de código, o que pode ser identificado utilizando detectores como os apresentados no capítulo 1.6.1.

A utilização do programa *strings* é exemplificado na Listagem 1.7. Aplicado o comando sobre o *malware* presente no arquivo *voegol.com.exe* onde a saída é direcionada para o arquivo *strings.txt*. Feito isso, é possível executar buscas diretamente no arquivo

strings.txt, por meio de uma combinação dos comandos *grep* e *cat*[19], por palavras chaves relacionadas ao caso investigado, se for um *trojan* bancário, uma busca por nomes dos bancos é recomendado. Para fins didáticos a busca se dará pelas palavras *ftp* e *http*.

Listagem 1.7. Utilização do programa strings sobre o malware

```
1 C:\Trojans> strings strings.com.exe > voegol.txt
2 C:\Trojans> cat voegol.txt | grep -i ftp
3 C:\Trojans> cat voegol.txt | grep -i http
```

1.8.5. Análise dinâmica

Análise dinâmica é uma abordagem baseada em comportamento e requer observação em tempo de execução de um artefato malicioso, utilizando um ambiente protegido e preparado para coletar informações das atividades do malware. Nessa etapa da análise, serão observadas as interações que o *malware* tem com o computador alvo. Serão analisados registros, processos, sistema de arquivos e tráfego de rede.

Com relação à execução de um código malicioso, podemos realizá-la de três maneiras distintas:

1. Execução manual simples;
2. Execução monitorada por meio de um *Instalation Monitor*;
3. Execução monitorada por meio de um *API Monitor*.

Para fins didáticos será conduzida uma análise através de execução manual simples e a qual pode ser divida em 4 etapas:

1. Preparação do ambiente;
2. Análise dos processos em execução;
3. Análise do tráfego de rede;
4. Análise do sistema de arquivos e registros.

1. Preparação do Ambiente: Analise de registro é uma técnica indispensável, pois permite que o analista encontre rapidamente informações sobre quais registros foram alterados durante a execução do código malicioso. Nesta etapa será utilizada a ferramenta *RegShot*[20], que possui todas as características esperadas com o objetivo de capturar dois *snapshots* do sistema, um antes e outro depois da execução do *malware*. O *RegShot* permite realizar a comparação entre os dois *snapshots* capturados. As diferenças são registradas em um arquivo texto, facilitando o tratamento e coleta da informação.

É recomendado que todos os programas necessários durante a análise já estejam em execução antes da captura do primeiro *snapshot*. Caso contrário, esses programas

[19] http://unxutils.sourceforge.net/
[20] http://sourceforge.net/projects/regshot/

podem gerar alterações nos registros que eventualmente podem ser confundidas com as próprias interações do *malware* analisado.

2. Análise dos processos em execução: Uma etapa importante é a identificação do modo de atuação do *malware* através da análise dos processos que serão criados durante a execução do código malicioso. A ferramenta que recomendada nesta etapa será o *ProcExp*.

O *ProcExp* é uma ferramenta poderosa, cuja função básica é verificar quais processos estão sendo executados, facilitando a identificação dos processos pais e filhos, informando, também, a localização (*path*) do arquivo que originou o processo, bem como a quantidade de memória utilizada e identificador PID (*Process Identification*). Além disso, pode-se verificar quais são as *DLLs* (*Dynamic Link Library*) que estão sendo utilizadas pelo processo, bem como *handles* abertos sendo possível realizar *dump* da memória do processo e analisar as strings encontradas, pois a ferramenta possui sua própria versão do comando *strings*.

Em algumas situações, o analista pode ter modelos semi automatizados, portanto ferramentas gráficas podem não trazer a facilidade em capturar as informações dos processos que já se conhece, dessa maneira recomenda-se a utilização da ferramenta *procdump* da *sysinternals*. A Tabela 1.6 apresenta alguns exemplos de utilização da ferramenta.

Tabela 1.6. Principais comandos - *Procdump*

Comando	Descrição
procdump -h voe-gol.exe malware.dmp	dump do processo chamado *voe-gol.exe* quando uma de suas janelas não responde por mais de 5 segundos
procdump iexplore	dump do processo chamado *iexplore* e salva para arquivo iexplore.bmp (default)
procdump -c 20 -s 5 -n 3 -o teste `c:\dump\teste`	Grava 3 Dumps do processo chamado *teste* quando exceder uso da CPU em 20% por cinco segundos

Caso o analista deseje analisar o processo por meio de outro programa, o *ProcExp* permite a criação de um *dump* da região de memória na qual o processo é executado. Basta clicar com o botão direito do mouse sobre o processo desejado, escolher a opção *"create dump"* e escolher a versão que deseja realizar, Figura 1.9.

Recomenda-se atenção quanto à localização (*path*) do arquivo que originou o processo em memória, uma vez que não é necessariamente o mesmo arquivo que originou a infecção na máquina. Isso ocorre, por exemplo, quando o *malware* utiliza-se de ferramentas para instalar o artefato remotamente, conhecida como *downloader*, direcionando o sistema para o código malicioso que efetivamente interagirá com a máquina alvo. Essa tática é utilizada para diminuir o tamanho dos arquivos utilizados para a disseminação e restringe a detecção por parte de alguns antivirus, pois um *downloader* não é propriamente um artefato malicioso, mas sim um forte indicio de evidência.

Caso o arquivo que esteja sendo executado não seja o que executamos no início da análise, isso é um forte indício de que houve um conteúdo baixado da internet para a má-

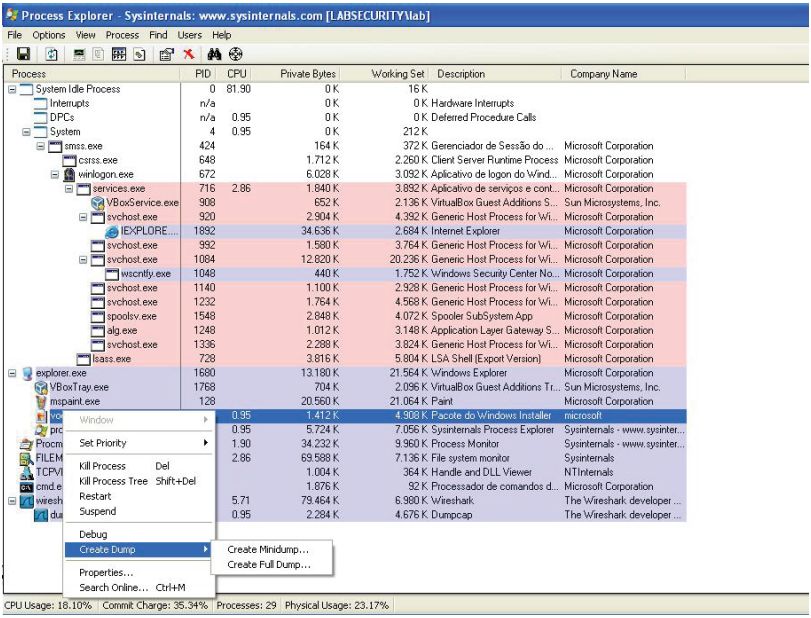

Figura 1.9. Dump de processo em memória com ProcExp

quina infectada, que pode ser confirmado através da análise de rede capturando os pacotes transmitidos e recebidos. Recomenda-se muita a atenção quanto ao nome verdadeiro do processo, que não é necessariamente o mesmo do arquivo executado. Esse nome, bem como o PID, serão importantes como filtros para as demais aplicações que utilizaremos.

O conjunto de ferramentas que será utilizada deve estar preparada junto com o analisador de processos *ProcExp* que em seguida, permitirá verificar quais processos são criados quando o *malware* for executado. Uma vez localizado o processo originado da execução do código malicioso, a análise propriamente dita é iniciada.

3. Análise do tráfego de rede: Ao se analisar o tráfego de rede originado pelas interações do *malware* procuramos por informações que auxiliem a identificação de aspectos chaves como:

- *Host* de destino dos dados capturados na máquina infectada;
- Tipos de dados capturados pelo *malware*;
- Possível *download* de atualizações ou de outros artefatos.

Nessa etapa são apresentadas técnicas para se capturar e analisar o tipo de tráfego gerado após a execução do *malware*, sendo necessário um analisador de rede. Um

dos mais populares é *Wireshark*[21], que atua como um sniffer de rede e permite coletar e analisar todo o tráfego gerado pelo artefato, de maneira a criar filtros rapidamente, identificando os protocolos e comunicações geradas, é multiplataforma e permite analisar o tráfego em tempo real ou *off-line*, possuindo suporte a decifração de diversos protocolos como *IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP* e *WPA/WPA2*. A Figura 1.10 apresenta a imagem dessa captura.

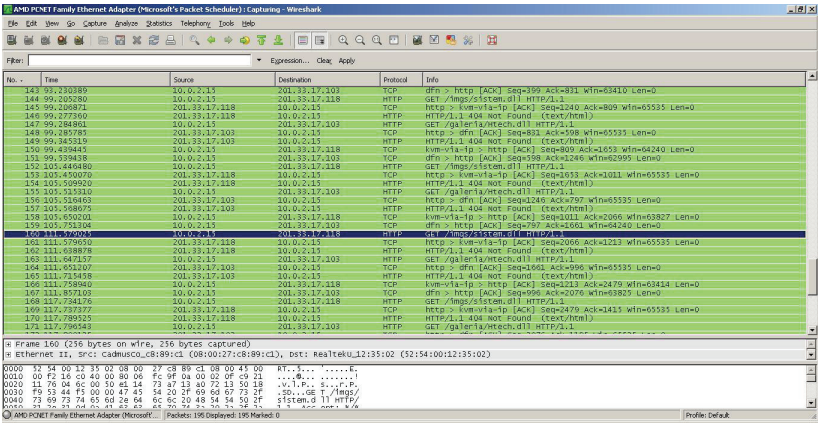

Figura 1.10. Análise de tráfego de rede com Wireshark

Tendo por base a captura realizada por esse programa, o analista deve procurar por pacotes que indiquem atividades do tipo:

- *http-post*: é utilizado para enviar dados a um servidor sem que haja necessidade de autenticação;
- *http-get*: é utilizado para *downloads* desde bibliotecas que serão utilizadas pelo *malware* até outros códigos maliciosos;
- *ftp*: usado tanto para receber quanto para enviar arquivos a um servidor.

Para facilitar a localização dos pacotes relevantes, é de fundamental importância a utilização dos filtros de captura. Neles podem ser especificados desde os IPs que se deseja capturar bem como os protocolos a serem exibidos ou uma combinação de ambos.

Exemplos de filtros:

- *ip.addr* == 192.168.0.102: filtra pacotes pelo endereço de IP informado;
- *icmp*: filtra os pacotes pelo protocolo desejado;

[21] `http://www.wireshark.org/`

- $ip.addr == 192.168.0.102||ftp||dns||http$: combinação de filtros pode ser realizada por meio dos operadores lógicos AND (&&) OR (||) ou comparadores igual (==) e diferente (!=).

Esse tipo de tráfego é importante pois permite determinar para onde as informações estão sendo enviadas, de maneira a facilitar a identificação pelo endereço IP de origem, caso os pacotes estejam sendo recebidos pela máquina infectada, ou de destino, caso eles estejam sendo enviados, de qualquer modo deve-se considerar que todo o tráfego é decorrente da interação das atividades do código malicioso.

É importante considerar que os dados coletados na análise estática, através das ferramentas como *strings*, bem como os obtidos pelos *dumps*, oriundos do *ProcExp*, Figura 1.9. Ao informar IPs, *URLs* (*Uniform Resource Locators*) e *PATHs*, eles podem auxiliar a nortear a busca por pacotes relevantes na captura.

Como a comunicação do malware aproveita a conexão de sua vitima, pode analisar os processos de rede que estejam ativos, neste caso recomenda-se a ferramenta *TCPView* ou *netstat*, que lista as conexões *TCP* (*Transmission Control Protocol*), *UDP* (*User Datagram Protocol*) e as portas estabelecidas com a máquina hospedeira. Saber quais conexões estão abertas ou mesmo aguardando pode ser útil a fim de se identificar quais *hosts* estão interagindo com a máquina infectada. A Listagem 1.8 mostra o resultado da execução *TCPView*.

Listagem 1.8. Resultado do programa *TCPView*

```
1  lsass.exe    736   UDP   labsecurity isakmp    *  *
2  lsass.exe    736   UDP   labsecurity 4500  *  *
3  svchost.exe 984   TCP   labsecurity epmap labsecurity 0  LISTENING
4  svchost.exe 1076  UDP   labsecurity ntp   *  *
5  svchost.exe 1204  UDP   labsecurity 1900  *  *
6  svchost.exe 1076  UDP   labsecurity ntp   *  *
7  svchost.exe 1136  UDP   labsecurity 1026  *  *
8  svchost.exe 1204  UDP   labsecurity 1900  *  *
9  System    4       TCP   labsecurity microsoft-ds   labsecurity 0
       LISTENING
10 System    4       TCP   labsecurity netbios-ssn labsecurity 0  LISTENING
11 System    4       UDP   labsecurity netbios-ns  *  *
12 System    4       UDP   labsecurity netbios-dgm *  *
13 System    4       UDP   labsecurity microsoft-ds   *  *
```

A primeira coluna diz respeito ao processo que está ativo seguido da porta de comunicação e o protocolo utilizado, as seguintes mostram os endereços resolvidos tanto de origem como destino conjuntamente com a porta de comunicação. Os serviços que se encontram em *LISTENING*, estão aguardando uma solicitação para que a conexão seja fechada, devem ser acompanhados com cuidado, pois alguns *malwares* instalam processos que são interpretados como serviços do sistema operacional.

Após a identificação do comprometimento, é de responsabilidade da equipe de resposta a incidentes (CSIRT), contatar o servidor remoto para onde as informações estão sendo enviadas, de modo a conter o incidente e neutralizar outras fontes de infecção que não serão tratadas, pois a partir do momento em que o incidente é neutralizado e sua

fonte de informação e conexão é suspensa, todo o sistema deixa de estar ativo, portanto o incidente passa a ser considerado erradicado.

4. Análise do sistema de arquivos e registros: Nessa etapa é dada uma atenção especial a análise das interações do *malware* com o sistema de arquivos e registros da máquina infectada. Para isso, a ferramenta utilizada será o *ProcMon*, Figura 1.11. A utilização dessa ferramenta será complementar a analise obtida através do *RegShot*. O *ProcMon* permite visualizar, em tempo real, quais arquivos e registros estão sendo acessados ou modificados e qual operação está sendo realizada sobre eles. O resultado obtido é ordenado por um número sequencial e cada entrada possui, também, um *timestamp*. Para facilitar a sua utilização, a ferramenta provê o recurso de filtragem pelo nome do processo ou seu PID. Dessa forma, basta adicionar ao filtro o nome do processo sendo analisado, assim todas as alterações de arquivo ou registro que serão capturadas dirão respeito a tal processo.

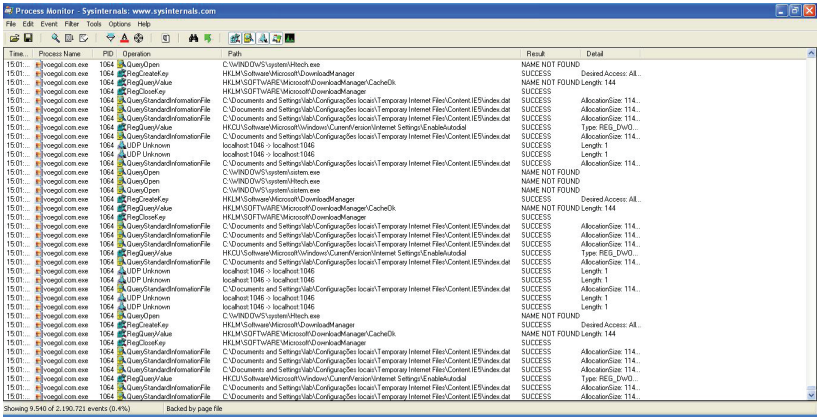

Figura 1.11. Análise sistema de arquivos e registros com ProcMon

Como apresentado na seção 1.8.4, calcular o *hash* do código malicioso a ser analisado é de extrema importância, entre outros fatores, para verificar se o *malware* já foi analisado ou se alterou seu próprio código durante a execução. Uma técnica utilizada para realizar tal verificação é a do *rehashing*. Essa técnica consiste em, cada vez que o artefato analisado for alterado, como quando o arquivo que originou o processo em memória não é o mesmo que executamos no início da análise, calcular o *hash* do novo artefato e comparar com o primeiro [Aquilina et al. 2008].

Alterar seu próprio código fonte é uma técnica que pode ser útil burlar os mecanismos de detecção dos antivírus.

1.9. Modelo de relatório de análise

Nesta etapa será apresentado um modelo de relatório de análise, que servirá para documentar e registrar todos os detalhes na condução da investigação e propostas de mitigação

de impacto, visando a melhoria dos processos de segurança.

Relatório 1.9. Exemplo de relatório de uma análise de malware

```
[1] Data do Report:
13/08/2009 11:48:23

[2] Tipo de Ataque:
Pop-up

[3] Incidente:
2009-08-12_BO3__[http_www.groupautounion.si_images_images_Voegol.php]

[4] URL:
http://www.groupautounion.si/images/images/Voegol.php

[5] Analisado por:
John Doe

[6] Hash (MD5):
02e44077c7b43040683ce2434f81d563 voegol.com.exe

[7] Arquivo criado/acessado:
c:\Documents and Settings\kurumin\Configuracoes locais\Historico\
    History.IE5\MSHist012009081320090814
c:\Documents and Settings\kurumin\Configuracoes locais\Historico\
    History.IE5\MSHist012009081320090814\index.dat
c:\Documents and Settings\kurumin\Configuracoes locais\Temp\~DFD33A.tmp
c:\WINDOWS\control.ctr
c:\WINDOWS\system\Htech.exe
c:\WINDOWS\system\sistem.exe

[8] Gerenciador de Tarefas:
c:\windows\system32\userinit.exe
c:\windows\explorer.exe
c:\windows\system32\vboxservice.exe
c:\windows\system\sistem.exe

[9] Registry:

   C:\WINDOWS\system32\userinit.exe

   Aplicativo de logon Userinit

   Microsoft Corporation

   c:\windows\system32\userinit.exe
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell
   Explorer.exe
   Windows Explorer
   Microsoft Corporation
   c:\windows\explorer.exe
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
   VBoxService
   VirtualBox Additions Service
   innotek GmbH
```

```
51      c:\windows\system32\vboxservice.exe
52      msne
53      c:\windows\system\sistem.exe
54
55  [10] Bancos Alvo:
56  BRADESCO
57
58  [11] Acao via Rede:
59  FTP
60
61  [12] Dados de rede (Destino):
62  ciroinfect@201.7.184.42
63  FTP= 201.7.184.42:21 LOGIN= ciroinfect SENHA= flor123.
64  http://200.106.147.115/recadastramento/stats.php
65  http://200.106.147.115/cartao2/finaliza.php
66
67  [13] Dados de rede (Origem):
68  Nao identificado.
69
70  [14] Observacoes:
71
72  Protecoes atacadas:
73  BRADESCO
74
75  URLs acessadas pelo trojan:
76  kaos.local - - [13/Aug/2009:11:39:43 -0300] "GET http://voegol.com.br/
        HTTP/1.1" - - "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
        5.1; SV1)"
77  kaos.local - - [13/Aug/2009:11:39:43 -0300] "GET http://
        oficinadacostura.com.br/imgs/sistem.dll HTTP/1.1" - - "-" "Mozilla
        /4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)"
78  kaos.local - - [13/Aug/2009:11:39:44 -0300] "GET http://www.voegol.com.
        br/Paginas/home.aspx HTTP/1.1" - - "-" "Mozilla/4.0 (compatible;
        MSIE 6.0; Windows NT 5.1; SV1)"
```

1.10. Considerações Finais

Empresas têm demonstrado grande preocupação frente aos desafios em manter seus sistemas seguros. Entender a taxonomia de um ataque, sistematizando os processos de responsabilidade de um grupo de resposta a incidentes e proteger os ativos da organização, é uma necessidade crescente e que exige preparo constante do profissional responsável pela segurança da informação.

Atividades que envolvem códigos maliciosos podem ser exploradas para criar verdadeiros exércitos de soldados do crime. Para enfrentar esse desafio, é preciso entender não só as ferramentas utilizadas para criar e distribuir esses códigos, mas também é preciso interpretar o pensamento do próprio atacante, buscando estar à frente dele para que seja possível a detecção e neutralização desses eventos.

Quando um incidente de segurança é reportado, a atuação da equipe de resposta deve ser precisa. A tentativa de tratar cada código malicioso isoladamente é uma estratégia fadada ao insucesso. Por outro lado, é preciso ter a capacidade de distinguir e identificar cada artefato malicioso, e só então colocar em prática a estratégia de segurança,

integrando contra-medidas e medidas de proteção relativas ao conjunto dos *malwares* que se apresentam.

Neste curso, foram apresentados os conceitos e os métodos para que o tratamento de incidentes que envolvem *malwares* seja realizado com fortes bases técnicas e que instrumentalize o profissional com as ferramentas que permitam o desenvolvimento do trabalho de análise. Entender o comportamento de um *malware* é o primeiro passo para identificar seu autor e então neutralizar suas ações. Isoladamente, o ato de detectar o ataque constitui-se como um processo de caráter apenas reativo. Entretanto, é necessário ter a capacidade de agir preventivamente, os seja, de criar e manter sistemas seguros que contemplem as proteções contra os ataques já conhecidos, o que requer a contínua análise da estrutura e funcionamento dos *malwares*.

Referências

[Abbas et al. 2006] Abbas, A., Saddik, A., and Miri, A. (2006). A Comprehensive Approach to Designing Internet Security Taxonomy. *2006 Canadian Conference on Electrical and Computer Engineering*, pages 1316–1319.

[ABNT 2005] ABNT (2005). ISO IEC 27001 Tecnologia da informacao Tecnicas de seguranca Sistemas de gestao de segurança da informacao Requisitos.

[Aquilina et al. 2008] Aquilina, J. M., Casey, E., and Malin, C. H. (2008). *Malware Forensics: Investigating and Analyzing Malicious Code*. Syngress Publishing, 1 edition.

[Binsalleeh et al. 2009] Binsalleeh, H., Ormerod, T., Boukhtouta, A., Sinha, P., and A (2009). On the Analysis of the Zeus Botnet Crimeware Toolkit. In *Proceedings of the Eighth Annual Conference on Privacy, Security and Trust (PST'2010)*, Ottawa, ON, Canada. IEEE Press.

[Brezinski and Killalea 2002] Brezinski, D. and Killalea, T. (2002). Guidelines for Evidence Collection and Archiving. RFC 3227, Internet Engineering Task Force - IETF.

[Brownlee and Guttman 1998] Brownlee, N. and Guttman, E. (1998). Expectations for Computer Security Incident Response. RFC 2350.

[Chen and Abu-Nimeh 2011] Chen, T. M. and Abu-Nimeh, S. (2011). Lessons from stuxnet. *IEEE Computer Society*, 44:91–93.

[Dube et al. 2010] Dube, T., Raines, R., Peterson, G., Bauer, K., Grimaila, M., and Rogers, S. (2010). Malware Type Recognition and Cyber Situational Awareness. *2010 IEEE Second International Conference on Social Computing*, pages 938–943.

[Finjan Research Center 2009] Finjan Research Center (2009). Cybercrime Intelligence: Cybercriminals use Trojans & Money Nules to Rob Online Banking Accounts. Number 3 in 1, pages 1–9. Finjan Malicious Code Research Center, Finjan Malicious Code Research Center.

[Jones 2001] Jones, P. (2001). US secure hash algorithm 1 (SHA1) RFC 3174.

[Kent et al. 2006] Kent, K., Chevalier, S., Grance, T., and Dang, H. (2006). *Guide to integrating forensic techniques into incident response*. National Institute of Standards and Technology, 800-86 edition.

[Kornblum 2006] Kornblum, J. D. (2006). Identifying almost identical files using context triggered piecewise hashing. In *Proceedings of the Digital Forensic Workshop*, pages 91–97.

[Martins et al. 2010] Martins, V., Grégio, A., Afonso, V., and Fernandes, D. (2010). xFile: Uma Ferramenta Modular para Identificação de Packers em Executáveis do Microsoft Windows. In SBC, editor, *SBSeg 2010*, pages 31–40, Fortaleza - CE.

[Mell and karen Kent 2005] Mell, P. and karen Kent, N. J. (2005). *Guide to Malware Incident Prevention and Handling Recommendations of the National Institute of Standards and Technology*, volume 800-83. Department of Homeland Security, Gaithersburg, 800-83 edition.

[Mieres 2009] Mieres, J. (2009). Analysis of an attack of malware web-based. Technical report, Malware Intelligence.

[Prosise et al. 2003] Prosise, C., Mandia, K., and Pepe, M. (2003). *Incident Response & Computer Forensics, 2nd Ed.* McGraw-Hill, Inc., New York, NY, USA, 2 edition.

[Rivest 1992] Rivest, R. (1992). The MD5 message-digest algorithm (RFC1321).

[Steding-Jessen 2008] Steding-Jessen, K. (2008). *Uso de Honeypots para o estudo de Spam e Phishing (Doutorado)*. Tese, INPE - Instituto Nacional de Pesquisas Espaciais.

1.11. Anexo I: Lista de Packers, Unpackers, Cryptors

Tabela 1.7. Ferramentas: Packers, Unpackers, Cryptors[Aquilina et al. 2008]

Tipos	URL's
Armadillo	www.siliconrealms.com/armadillo_engine.shtml
ASPack	www.aspack.com
BeRoEXEPacker	bero.0ok.de/blog/projects/beroexepacker/
CExe	www.scottlu.com/Content/CExe.html
Exe32pack	www.steelbytes.com
EXECryptor	www.strongbit.com/execryptor.asp
eXPressor	www.expressor-software.com/
FSG	www.exetools.com/protectors.htm
Krypton	programmerstools.org/taxonomy/term/17?from=20
MEW	www.exetools.com/protectors.htm
Molebox	www.molebox.com/
Morphine	www.exetools.com/protectors.htm
NeoLite	www.exetools.com/protectors.htm
Obsidium	www.obsidium.de/show.php?download
PEBundle	www.bitsum.com/pebundle.asp
PECompact	www.bitsum.com/
PE Crypt 32	www.opensc.ws/asm/1071-pecrypt.html
PELock	http//pelock.com/page.php?p=pelock#download
PEPack	www.dirfile.com/freeware/pepack.htm
PESpin	pespin.w.interia.pl/
Petite	www.exetools.com/protectors.htm
PKLite32	pklite32.qarchive.org/
PolyCryptPE	www.cnet.com.au/downloads/0,239030384,10420366s,00.htm
RLPack	rlpack.jezgra.net
SFX	www.exetools.com/protectors.htm
Shrinker32	www.exetools.com/protectors.htm
Themida	www.oreans.com/downloads.php
UPX	upx.sourceforge.net/
yoda	yodap.cjb.net/

Capítulo

2

Aprendizagem de Máquina para Segurança em Redes de Computadores: Métodos e Aplicações

Márcia Henke, Clayton Santos, Eduardo Nunan, Eduardo Feitosa, Eulanda dos Santos, Eduardo Souto.

Instituto de Computação (IComp)
Universidade Federal do Amazonas
69077000 - Manaus - AM - Brasil

{henke, clayton.maia, nunan, efeitosa, emsantos, esouto}@dcc.ufam.edu.br

Abstract

The growth of malicious activities owing to fraud such as phishing, malicious code proliferation, Distributed Denial of Service attack, and others, may be clearly observed through Internet traffic analysis. This chapter presents Machine Learning tools which can be used for the design of effective automatic malicious activity detection systems. Moreover, it points out that Machine Learning has already been successfully applied to improve malicious activity detection rates in problems dealing with phishing, cross-site scripting, malware, denial of service and intrusion detection. Finally,it also highlights a list of research topics and open problems, as well as potential application of Machine Learning on network security domain.

Resumo

Uma breve análise do tráfego Internet comprova o crescente aumento de atividades maliciosas originadas por ações fraudulentas como phishing, proliferação de códigos maliciosos, ataques de negação de serviço, entre outras. Este capítulo apresenta ferramentas de aprendizagem de máquina (AM) que podem ser usadas para o desenvolvimento de sistemas efetivos de detecção de atividades maliciosas. Além disso, destaca que significativos avanços nessa área de aplicação já estão sendo obtidos graças ao emprego de técnicas de AM. São apresentados exemplos do uso de AM em diversos problemas de detecção, tais como: phishing, cross-site scripting, malware, negação de serviço e detecção de intrusos. Também destaca pesquisas em aberto e potenciais aplicações de AM em problemas de segurança de redes de computadores.

2.1. Introdução

Com uma rápida consulta às estatísticas de tráfego na Internet, percebe-se que o tráfego conhecido, solicitado (ou em outras palavras, legítimo) foi dominado pelo tráfego de mensagens eletrônicas não solicitadas (*spam*), atividades fraudulentas como *phishing*[1] e *pharming*[2], ataques de negação de serviço, proliferação de vírus, *worms* e outros código maliciosos (ou seja, tráfego ilegítimo, improdutivo, não desejado e não solicitado).

O instituto de segurança americano CSI (*Computer Security Institute*) [Richardson, 2010], depois de entrevistar 149 empresas nos Estados Unidos, constatou o crescimento no número de infecções por *malware*, *botnets* e *phishing*. Considerando dados de 2007 e 2010, o número de *Malware* subiu de 52% para 67% em 2010; *botnets* contabilizava 21% em 2007, e em 2010 registrou 29%; e o número de páginas *phishing*, subiu de 26% para 39% nesse mesmo período. No Brasil, o CERT.br (Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil) também contabilizou aumentos expressivos de atividades maliciosas em 2010: 17.628 incidentes relativos a *worms*, 198 ataques de negação de serviço e 31.008 tentativas de fraude (*phishing*) [CERT.br, 2011].

O aumento das atividades maliciosas se deve, entre outras coisas, a ineficiência das atuais soluções em identificar, reduzir e interromper esse tipo de tráfego. Tipicamente, a efetividade fornecida pelas atuais soluções é comprometida pelas altas taxas de falsos alarmes e pela falta de cooperação com outras soluções ou mesmo com a infraestrutura de rede. Além disso, muitas das soluções dependem de ativação ou interferência humana (configuração, adequação e ajuste) para funcionar.

Na busca por soluções mais efetivas e corretas, especialmente aquelas relacionadas ao tráfego Internet malicioso, muitos pesquisadores e a indústria de software de segurança vêm investindo tempo e recursos para tornar a aprendizagem de máquina uma ferramenta poderosa e eficaz na segurança de redes de computadores. A idéia é bastante simples: se as máquinas podem aprender quando um sistema está funcionando normalmente e quando ele está sob ataque, então é possível construir mecanismos capazes de, automaticamente, responder a ataques e anomalias normalmente encontradas em uma rede.

Entretanto, em termos operacionais, observa-se um desequilíbrio no uso de aprendizagem de máquina. Enquanto em alguns em ambientes operacionais existem implementações comerciais bem sucedidas como, por exemplo, os sistemas de recomendação de produtos usados pela Amazon [Lindenet al., 2003] e Netflix [Bennett et al., 2007] e os sistemas de reconhecimento de caracteres (OCR) [Vincent, 2007] e [Smith, 2007], na área de segurança parece que o uso de aprendizagem de máquina ainda está em um estágio inicial. A explicação para essa diferença será discutida nesse minicurso.

[1] *Phishing* é um tipo de fraude eletrônica caracterizada pela tentativa de obter informações pessoais privilegiadas através de sites falsos ou mensagens eletrônicas forjadas.

[2] *Pharming* refere-se ao ataque de envenenamento de *cache* DNS cujo objetivo é preparar terreno para atividades de *phishing.*

2.1.1. Contexto

Detecção de anomalias (também chamada de detecção de *outliers*) se refere ao problema de encontrar padrões em dados que não estão de acordo com o comportamento esperado. Sistemas de detecção de anomalias objetivam encontrar desvios inesperados de comportamento. Com base em uma "noção de atividade normal", eles relatam desvios desse perfil como alertas. A importância da detecção de anomalias se deve ao fato de que as anormalidades nos dados podem traduzir informações úteis, significantes e muitas vezes críticas sobre o domínio de aplicação em questão. Chandola et al. [Chandola et al., 2009] fornecem um vasto documento sobre detecção de anomalias, incluindo várias áreas de aplicação. Na área de segurança de redes de computadores, por exemplo, um padrão anômalo de tráfego em um computador pode significar que o mesmo foi invadido e pode, possivelmente, estar enviando dados pessoais a um destino não autorizado. Em outro exemplo, anomalias nas transações de cartão de crédito podem indicar o roubo de cartões ou de identidades.

De modo geral, a detecção de anomalia vem sendo estudada desde o século 19, o que resultou no surgimento de uma variedade de técnicas e soluções. No contexto de segurança de redes, o conceito básico de qualquer sistema de detecção de anomalias foi introduzido pela primeira vez por Denning [Denning, 1987] em seu trabalho seminal em 1987. Desde então, muitas abordagens foram desenvolvidas. Muitas delas emprestando esquemas, técnicas e metodologias da área de aprendizagem de máquina.

Contudo, o uso da aprendizagem de máquina na detecção de anomalias é diferente de outros domínios de aplicação. A diferença está nas informações. Abordagens de detecção de anomalias precisam lidar com um conjunto de problemas bem conhecidos [Gates e Taylor, 2007]: os detectores tendem a gerar grandes quantidades de falsos positivos; existe uma grande dificuldade em encontrar dados livre de ataque para realizar o treinamento das soluções; e os atacantes podem escapar da detecção ensinando gradualmente ao sistema a aceitar atividades maliciosas como benignas.

Assim, quando comparado com ao resultado operacional de um grande número de pesquisas, a detecção de anomalias em redes não parece ser tão atraente no "mundo real". Os sistemas existentes são geralmente baseados em perfis estatísticos que usam tráfego agregado, como o *Arbor Peakflow*[3] e o *LANScopeStealthWatch*[4], embora operem com um foco muito mais específico quando comparado as generalizações feitas pelas pesquisas acadêmicas.

Desta forma, este minicurso tem por objetivo discutir os diferentes pontos que envolvem o uso da aprendizagem de máquina na detecção de anomalias de redes, incluindo as dificuldades de detecção e uso incorreto da aprendizagem.

2.1.2. Ameaças à Segurança na Internet

Para melhorar a compreensão dos leitores sobre assunto, esta seção apresenta algumas das principais ameaças (ataques e anomalias) à segurança da Internet.

[3] http://www.arbornetworks.com/en/peakflow-sp.html
[4] http://www.lancope.com/products/

2.1.2.1. *Cross-Site Scripting* (XSS)

A popularização de serviços pautados no uso de tecnologias *Web* e tendo como principal ferramenta, o *browser*, passou a ser o alvo frequente de diversos agentes que tentam explorar recursos computacionais de forma não autorizada como, capturar *cookies* e sessões de usuários, dados pessoais, número de cartões de crédito, senhas, *logins*, além de outras informações críticas. A demanda cada vez maior por novos recursos para essas aplicações, a fim de prover um número maior de serviços e funcionalidades para seus usuários promoveu um crescimento desordenado e o desalinhamento das políticas de segurança entre servidores e clientes, aumentando assim os riscos de segurança. Dentre eles, destacam-se os ataques de injeção de códigos maliciosos ou *Cross-Site Scripting* (XSS), viabilizados por *scripts* maliciosos injetados em páginas *Web*, no lado cliente ou servidor, de forma que os dados de entrada do usuário não são devidamente validados, permitindo o roubo de informações confidenciais e sessões do usuário, além de comprometer a integridade do sistema em execução. Os códigos *script* são tipicamente desenvolvidos nas linguagens *JavaScript*, *Java* ou *VBSript* e inseridos na estrutura HTML. Tecnologias *Active X*, *Flash* ou quaisquer outros suportados pelo navegador também são usados como vetores de ataques [Grossman et al., 2007].

Notadamente, o vetor mais empregado em ataques de XSS é a linguagem *JavaScript*, que é interpretada pelo *browser*, que aguarda por eventos ou ações, tais como o término de leitura de uma página, ou por ações do usuário, como o movimento do *mouse* sobre um *link* ou o *click* em um objeto, como um botão. Conforme destaca Yue e Wang [Yue e Wang, 2009], 96,9% das páginas *Web* mais populares do mundo fazem uso de *scripts* com base nessa linguagem, denotando a sua popularidade e o seu emprego potencial em ataques do tipo XSS. Isso se deve, principalmente, a inserção e geração insegura de elementos da linguagem *JavaScript* em esquemas potencialmente perigosos, empregando *tags* HTML, atributos e funções *script*. Pode-se citar como exemplo, a geração de código de outros domínios por meio do uso de funções como *document.write* ( ) ou *eval* (), que executa código passado por parâmetro, configurando-se em uma ação potencialmente insegura.

Os ataques são categorizados em XSS Reflexivo ou Não Persistente, Persistente e *DOM-Based*. Nos ataques do tipo Persistente e Reflexivo, o código malicioso está incorporado na página que é retornada para o *browser* cliente. Já no tipo *DOM-Based* o código ao manipular a estrutura DOM (*Document Object Model*), pode eliminar ou criar novos nós dessa estrutura, o que requer a execução ou simulação do preenchimento dos objetos DOM [Saha, 2009] para que esse tipo de ataque possa ser identificado. A figura 2.1 apresenta uma visão geral de um ataque de XSS.

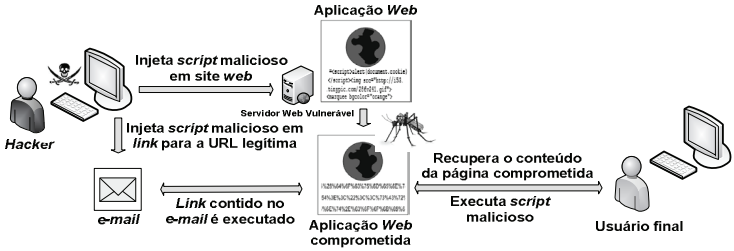

Figura 2.1 apresenta uma visão geral de um ataque de XSS.

Cross-Site Scripting Reflexivo ou Não Persistente

Em ataques não-persistentes, o código malicioso é executado de forma reflexiva no *browser* do cliente. O código não necessita estar armazenado no servidor. O ataque é viabilizado por meio da ação do usuário, que ao receber e executar um *link* contendo código malicioso é direcionado para a página confiável, porém vulnerável a XSS. Ao carregar a página o código é interpretado pelo *browser* e a ação maliciosa é efetivada. Normalmente, os *links* são enviados por *spam* ou por qualquer outro recurso que induza o usuário a visitar uma página por meio do *link* (URL) enviado. Este é o tipo de ataque XSS mais frequente [OWASP, 2010].

Cross-Site Scripting Persistente

Neste tipo de ataque, o código malicioso é permanentemente armazenado em recursos do servidor (banco de dados, sistema de arquivos, entre outros). Aplicações *Web* que permitem ao usuário armazenar dados, tais como fóruns de mensagens, livros de visita, são as mais vulneráveis. Ao acessar uma página contaminada, o usuário poderá executar o código malicioso por meio de uma ação sobre um campo de busca, por exemplo. Este é o tipo mais perigoso de XSS [OWASP, 2010].

Cross-Site Scripting baseado em DOM

Ataques *DOM-Based* são realizados por modificações no ambiente DOM (*Document Object Model*), que é um formato estrutural que pode ser usado para representar documentos no *browser*. O DOM habilita os *scripts* dinâmicos para referenciar componentes do documento, como um campo de texto ou uma sessão de *cookie*, por exemplo [OWASP 2010]. Em ataques deste tipo, funções dinâmicas *JavaScript* são modificadas por códigos maliciosos que realizam mudanças nesse ambiente [Saha, 2009]. Neste caso, o servidor não tem qualquer ingerência, pois o ataque modifica apenas o lado cliente e não envia qualquer código malicioso para o servidor.

Geralmente, para detecção de ataques XSS é necessário aderir às políticas da mesma origem, que são estabelecidas por meio de mecanismos que permitem aos *scripts* acessarem páginas apenas a partir do mesmo domínio, ou ainda, de alterações no código fonte de navegadores atuais. Logo, pesquisadores têm adotado diversos conceitos para identificação e classificação deste tipo de ataque como modelos estatísticos, mineração de dados, teoria da informação e aprendizagem de máquina.

2.1.2.2. *Phishing*

Segundo o *Anti-Phishing Working Group* [APWG, 2010], o *phishing* é uma prática fraudulenta em que criminosos (conhecidos como *phishers*) utilizam tanto engenharia social quanto artifícios técnicos para tentar roubar informações confidenciais das vítimas. Em ataques usando engenharia social, o *phisher* envia mensagens eletrônicas, normalmente e-mails ou mensagens instantâneas, alegando ser uma entidade legítima (bancos, SPC/SERASA, receita federal, entre outros) ou uma pessoa conhecida. O conteúdo dessas mensagens apresenta formulários para o preenchimento e o envio de dados pessoais e financeiros, links para baixar e abrir/executar programas ou links para uma página *Web* construída especificamente para que o destinatário da mensagem forneça dados pessoais e financeiros. Em ataques que utilizam artifícios técnicos, o criminoso instala no computador da vítima software maliciosos, como *trojan*s, *keylogger* e *spyware*s, para roubar dados sigilosos.

A seguir são apresentadas algumas situações envolvendo *phishing*, que vem sendo utilizadas por fraudadores na Internet. Cabe ao leitor deste minicurso observar que há diversas variantes para as situações apresentadas e que, constantemente, surgem novas formas de *phishing*.

- **Mensagens contendo *links* para programas maliciosos**: São mensagens enviadas por *e-mail,* na qual o texto procura atrair a atenção do usuário seja ela por curiosidade, ingenuidade ou na busca de obter vantagem financeira. Geralmente, o texto busca induzir o usuário a clicar em um *link* para baixar ou executar um arquivo, sob a pena de sofrer alguns danos, como por exemplo, a inclusão do seu nome no SPC/SERASA, o cancelamento de um cadastro, da sua conta bancária ou do seu cartão de crédito, etc.
- **Páginas de comércio eletrônico ou *Internet Banking* falsificadas:** Neste tipo de fraude o usuário recebe uma mensagem por e-mail, em nome de um *site* de comércio eletrônico ou de uma instituição financeira. O objetivo é tentar persuadir o usuário a clicar em um *link* contido no texto ou em uma imagem. Geralmente, os textos contidos nestas mensagens envolvem o recadastramento ou confirmação dos dados do usuário, ou a participação em uma nova promoção. Vale ressaltar que, o link direciona o usuário para uma página falsa, semelhante à página legítima, onde serão solicitados dados pessoais e financeiros.
- **Comprometimento do DNS**: Comumente conhecida como *pharming*, neste tipo de fraude, o usuário ao tentar acessar um *site* de comércio eletrônico ou *Internet Banking*, mesmo digitando o endereço corretamente e diretamente no seu *browser*, será redirecionado para uma página falsificada, semelhante ao *site* verdadeiro. Geralmente, a causa deste ataque é decorrente do comprometimento do DNS pelo atacante, de modo que todos os acessos passaram a ser redirecionados para páginas falsificadas. O atacante pode ter induzido o usuário a instalar, em algum momento, um *malware* específico para alterar o DNS.
- **Utilização de computadores de terceiros**: É um dos principais vetores para disseminação de fraudes do tipo *phishing*, pois geralmente o usuário utiliza uma *LAN house* ou *cybercafe*, para acessar um site de comércio eletrônico ou *Internet Banking*. O risco se deve à utilização destes computadores por diversas pessoas,

que em algum momento, foram induzidas a instalar um código malicioso, tendo, portanto todas as ações monitoradas. A partir daí, qualquer usuário que necessitar utilizar estes computadores para realizar transações eletrônicas poderão ter seus dados comprometidos.

2.1.2.3. Negação de Serviço

Diferente de ataques como XSS e *phishing*, um ataque de negação de serviço (do Inglês *Denial-of-Service* ou DoS) consiste na tentativa de impedir usuários legítimos de utilizarem um determinado serviço de um computador ou rede, ou seja, tornar indisponíveis recursos ou serviços oferecidos por um servidor, sistema ou rede [Kumar e Selvakumar, 2011].

Atualmente, ataques de negação de serviço são executados de forma distribuída (DDoS – *Distributed Denial of Service*), onde o atacante busca potencializar e dimensionar o ataque. Desta forma, os ataques DDoS não são baseados no uso de um único computador, mas sim de centenas ou até milhares de computadores previamente comprometidos e ligados na Internet para lançar um ataque coordenado a determinada vítimas ou vítimas. Um ataque DDoS utiliza uma espécie de arquitetura (classe) social composta por: (i) **vítima**, alvo do ataque; (ii) **zumbi**, máquina que efetivamente concretiza o ataque DoS contra uma ou mais vítimas; (iii) **mestre**, computador intermediário localizado entre o atacante e os zumbis, que recebe os parâmetros (ordens) para o ataque. Cada mestre controla certo número (centenas ou milhares) de zumbis; (iv) **atacante**, responsável por coordenar o ataque.

A Figura 2.2 exemplifica a estrutura de um ataque de negação de serviço distribuído (DDoS).

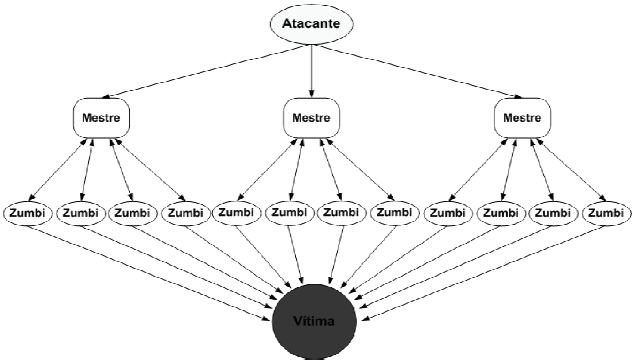

Figura 2.2. Estrutura de um ataque de negação de serviço distribuído.

Em [Feitosa et al., 2008] é encontrada uma descrição mais detalhada dos principais tipos de ataque de negação de serviço, incluindo inundação (*flooding*), ataques ao protocolo TCP, ataques por refletores, *backscatter*, entre outros.

2.1.2.4 Códigos Maliciosos (*Malware*)

A disseminação de código malicioso através da Internet tem sido a maior ameaça atual à segurança de sistemas de informação. Código Malicioso ou *Malware* é um termo

genérico que abrange todos os tipos de programas desenvolvidos especificamente para executar ações maliciosas em um computador. Esta denominação é comumente empregada a um conjunto de aplicações também denominadas individualmente como vírus, *worms*, *keyloggers*, *trojans*, *backdoors* e outros tipos de programas com a intenção de comprometer um sistema.

De acordo com [Feitosa et al., 2008], os principais exemplares *malware* podem ser descritos conforme a tabela 2.1.

Tabela 2.1. Principais exemplares de *malware*.

Tipo	Descrição
Vírus	São programas que modificam a operação normal de um computador, sem permissão e conhecimento do usuário. Assim como um vírus biológico, um vírus de computador se replica e se espalha introduzindo cópias suas em outros códigos ou programas executáveis.
Worms	São programas auto-replicantes capazes de se auto-propagar através da rede explorando principalmente falhas de segurança em serviços. A taxa de propagação dos *worms* é muito rápida e pode ameaçar a infraestrutura da Internet uma vez que cada máquina infectada torna-se um potencial atacante. De modo geral, prejudicam a rede consumindo largura de banda. A ativação de um *worm* pode ser tão rápida quanto sua velocidade de propagação.
Cavalo de Tróia (*Trojan Horse*)	São programas que, uma vez ativados, executam funções escondidas e não desejadas como, por exemplo, varreduras de endereços IP e porta, envio de grande volume de *spam*, ataques DDoS ou até mesmo adicionar o computador em uma *botnet*. Diferente dos *worms*, um cavalo de tróia não se auto-propaga, depende da interferência e curiosidade humana para se propagar.
Spyware	São programas espiões que automaticamente recolhem informações sobre o usuário e os transmite para seus "instaladores". Geralmente, os dados monitorados referem-se aos habítos de compras na Internet ou a informações confidenciais como contas bancárias e senhas pessoais. Além dos próprios *spywares*, na categoria de programas espiões também estão o *adware* e o *keylogger*.

Percebe-se que a crescente interação de dispositivos diversos, juntamente com a popularização da Internet, aliada à vasta provisão de serviços e falta de conhecimento adequado por parte dos usuários contribui para o cenário atual de ataques por meio de *malware*. A venda de informações sensíveis, tais como contas e senhas bancárias e números de cartões de créditos são grandes motivadores para ocorrência desses ataques.

Os sistemas computacionais vulneráveis são facilmente explorados por atacantes que utilizam códigos maliciosos. Há, portanto, diversas técnicas para que um código malicioso seja executado nos sistemas, tendo as mais comuns baseadas em engenharia social, exploração remota de serviços e injeção de código [Ceron et al, 2009]. Quando o sistema é comprometido, o mesmo fica sob o domínio do atacante, podendo realizar as mais diversas ações fraudulentas, tais como roubo de dados sigilosos, envio de *spam*, e *emails phishing*.

Uma das maneiras mais utilizadas para o comprometimento de sistemas é através da propagação autonômica de *malwares*. A busca por máquinas vulneráveis com o intuito de explorar vulnerabilidades e comprometer o sistema é característica de *malwares* como *worms* e *bots*. Uma forma de entender as características dos diferentes tipos de *malware* é realizar a análise de tráfego malicioso.

O principal meio de defesa contra códigos maliciosos é através de softwares antivírus, pois se baseiam em um banco de dados contendo diversas assinaturas para caracterizar um *malware* conhecido. Porém, quando um *malware* não possui uma assinatura é necessário analisá-lo e entender como o seu comportamento afeta o sistema, para então desenvolver sua assinatura, e posteriormente, conhecer suas funcionalidades para que seja removido do sistema.

2.1.2.5. *Spam*

O problema de e-mail não solicitado ou *spam* é bem conhecido pela maioria dos usuários Internet. O *spam* causa a perda de tempo dos usuários e a sobrecarga dos recursos computacionais, ocasionando assim grandes perdas financeiras.

De acordo com [Feitosa et al., 2008], o *spam* pode ser classificado conforme a Tabela 2.2.

Tabela 2.2. Classificações de *Spam*.

Tipo	Descrição
Boatos (*hoaxes*)	São mensagens que tentam impressionar os usuários através de estórias falsas e assim garantir sua divulgação. Alguns exemplos deste tipo de mensagens incluem o roubo de rins, desaparecimento de crianças, difamação de empresas, etc.
Correntes (*chainletters*)	São mensagens que prometem algum de tipo de lucro financeiro ao leitor, caso seja feito o repasse destas mensagens a um determinado número de usuários.
Propagandas ("*pumpanddump*")	São as mensagens mais comuns e mais divulgadas na Internet, sendo representadas por mensagens eletrônicas não solicitadas com oferta de produtos ou artigos baratos com links para companhias pequenas ou inexistentes.
Golpes (*scam*)	São mensagens enganosas que afirmam que o leitor foi "agraciado" com algum produto. Exemplos corriqueiros incluem sorteios, oferta de empregos, abertura do próprio negócio, etc.
Estelionato (*phishing*)	São mensagens usadas para obter dados pessoais (tais como números de contas bancárias, cartão de crédito, senhas, entre outros) de destinatários. Normalmente, induzem o leitor a acessar a URL indicada na mensagem, clicar na imagem de uma empresa ou preencher algum tipo de formulário.
Malware	São mensagens enviadas ao leitor contendo códigos maliciosos, com a finalidade de enganar o leitor, levando-o a executar um determinado programa enviado junto a mensagem. O resultado é a instalação de vírus, *worms* e cavalos de tróia, sempre visando alguma tentativa de fraude ou ataques de negação de serviço.

2.1.3. Soluções Existentes

A Internet é uma das poucas plataformas operacionais existentes que não possui centros de controle. Tal característica serve tanto como fator de sucesso quanto serve como ponto o de falhas, o que explica o aumento do tráfego malicioso. Na tentativa de lidar com esse lado negativo da Internet, várias soluções têm sido desenvolvidos ao longo dos anos. Esta seção apresenta algumas dessas soluções que visam tornar o ambiente de redes mundiais de computadores um ambiente mais seguro e menos hostil em relação às ameaças a segurança da Internet apresentadas na seção anterior.

2.1.3.1. Filtragem

Entre as soluções mais usadas atualmente na segurança da Internet, sem dúvida nenhuma os esquemas de filtragem ou *firewall* são principais. Basicamente, *firewalls* utilizam regras (filtros) que definem o que deve ser feito. Desta forma, todo o tráfego

que entra ou sai da rede ou máquina é comparado com as regras e o resultado é uma ação, geralmente permitir ou negar o tráfego.

Embora não exista um consenso sobre a classificação dos tipos de *firewall*, a mais usual considera a existência de três: filtro de pacotes, filtro de estados e filtros de aplicação (*gateways*). O primeiro tipo, filtro de pacote, compara as informações do cabeçalho de cada pacote (endereços IP, portas e protocolo) com as regras definidas para decidir qual ação tomar. A segunda, filtros de estado, mantêm registros do estado das conexões de rede (TCP e UDP) que estão ativas. Assim, a filtragem é feita baseada na tabela de estados de conexões estabelecidas e não apenas no cabeçalho. O último tipo, *firewall* de aplicação, opera na camada de aplicação, vasculhando o conteúdo dos pacotes a procura de indícios de anomalias como, por exemplo, sequências de caracteres específicos (palavras ou frases) que indicam a presença de ataques, código maliciosos e até mesmo de determinadas aplicações como SMTP (*Simple Mail Transfer Protocol*), FTP (*File Transfer Protocolo*), HTTP (*Hyper Text Transfer Protocol*), P2P, entre outros.

Entretanto, a efetividade das soluções de filtragem não tem sido suficiente para limitar o tráfego malicioso. Em primeiro lugar, embora prática, a filtragem é imperfeita, porque depende de heurísticas e configurações manuais, o que por muitas vezes acaba filtrando informações legítimas. Além disso, a filtragem mais eficiente é destinada a aplicações específicas. Mecanismos de filtragem como *proxies* e *gateways* de aplicação são desenvolvidos para avaliar tráfego específico de determinadas aplicações. Desta forma, para cada novo serviço ou aplicação que se desejar filtrar, uma solução específica será necessária.

2.1.3.2. Sistemas de Detecção de Intrusão

A detecção de intrusão refere-se à descoberta de atividades maliciosas (intrusões, penetrações e outras formas de abuso de computadores) dentro de um sistema de computadores relacionados [Phoha, 2002]. A detecção de intrusão pode ser classificada em duas categorias [Anderson, 1995] [Rhodes et al., 2000]: detecção de abuso (*misuse detection*) e detecção de anomalias (*anomaly detection*). Na primeira categoria, as soluções utilizam bases de assinaturas, assim os ataques conhecidos são detectados com bastante rapidez e com baixa taxa erro. Por outro lado, a principal limitação dessas ferramentas é que elas não podem detectar novas formas de códigos maliciosos que não sejam compatíveis com as assinaturas existentes. A segunda categoria, detecção de anomalias, é baseada na construção de perfis de comportamento para padrões considerados como atividade normal. Desvios da normalidade são então tratados como ameaças. Entretanto, é difícil saber o que procurar quando atividades não autorizadas sob um sistema assumem diferentes formas ou mesmo imitam atividades legítimas. Na tentativa de evitar que atividades com potencial malicioso sejam autorizadas, muitos sistemas emitem uma taxa elevada de falsos alarmes, reduzindo substancialmente sua efetividade.

A Figura 2.3 ilustra uma visão genérica da organização de um sistema de detecção de intrusão. O administrador recebe relatórios do servidor de alarmes que também envia relatórios para o sistema monitorado. O administrador conta com um servidor de modelos de intrusão que auxilia com o reconhecimento de uma intrusão.

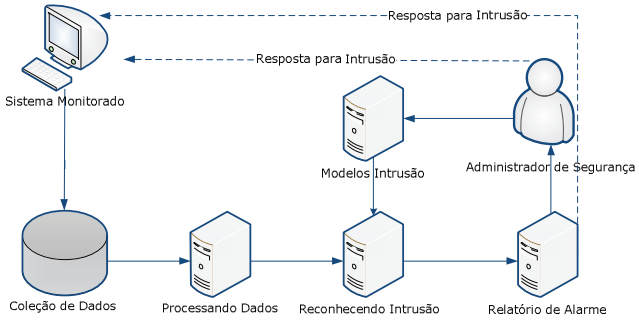

Figura 2.3. Organização genérica de um Sistema de Detecção de Intrusão

O principal problema dos IDS é a precisão. Como existe uma constante mutação no tráfego de rede e nos padrões de assinatura, a detecção de intrusão se torna cada vez mais difícil e sujeita a erros.

2.1.3.3. Software anti-*

A terceira e última categoria de soluções são os software anti-*, programas desenvolvidos para detectar e eliminar potenciais ameaças aos computadores e redes como, por exemplo, antivírus, anti-*spyware*, anti-*phishing* e anti-spam. Antivírus são programas que detectam e removem vírus de computador. Uma vez que novos vírus e variantes de vírus conhecidos são "lançados" quase que diariamente, um *software* antivírus deve manter-se automaticamente ou ser mantido sempre atualizado. Já os anti-*spywares* são usados no combate a programas e códigos espiões como *spyware*, *adware*, *keyloggers*. Softwares anti-*phishing* visam bloquear possíveis tentativas de fraude através de sites *Web* ou mensagens de correio eletrônico. Normalmente, este tipo de solução é apresentado na forma de barras de tarefas (*toolbar*) integradas com navegadores *Web* ou clientes de correio eletrônico.

As soluções anti-*spam* também se enquadram nesta categoria. Basicamente, a detecção de *spam* é baseada na filtragem de mensagens não solicitadas através dos campos do cabeçalho ou do conteúdo da mensagem. A filtragem de cabeçalho verifica o endereço de origem, nome do remetente e assunto de uma mensagem para validá-la ou não. Este tipo de solução anti-*spam* é mais simples e sujeita a erros de configuração, uma vez que é preciso definir regras do que se quer ou não receber, ou seja, quais endereços, remetentes e assuntos são indesejados. Listas negras (*blacklist*), listas brancas (*whitelist*) e listas cinza (*greylist*) são exemplos de filtragem de cabeçalho. A filtragem baseada no conteúdo da mensagem é a mais utilizada. Normalmente esta técnica realiza buscas por palavras chaves tais como "Viagra" no conteúdo das mensagens. Quando configurados corretamente, a filtragem baseada em conteúdo é bem eficiente, mas também podem cometer erros (barrar "especialista" porque contém "cialis", por exemplo). Além disso, a inspeção de conteúdo não verifica a origem da mensagem. Atualmente, o uso de filtros com mecanismo de aprendizagem de máquina tem sido muito utilizado.

2.1.4. Organização do Capítulo

Diante do exposto, torna-se claro a necessidade de desenvolver novas soluções que contribuam para a diminuição das atividades maliciosas na Internet. Este minicurso tenta fornecer uma visão estruturada e abrangente da pesquisa sobre detecção de anomalias que utiliza técnicas e algoritmos de aprendizagem de máquina, com o objetivo de facilitar a compreensão das diferentes direções em que a pesquisa nesta área tem evoluído e como as técnicas desenvolvidas podem ser aplicadas na prevenção e detecção de atividades maliciosas.

O restante deste capítulo está organizado da seguinte forma. A Seção 2.2 apresenta de forma clara e didática as técnicas de aprendizagem de máquina. Para tanto, os conceitos relacionados à aprendizagem de máquina são explicados, objetivando definir formalmente aprendizagem de máquina e os contextos dos problemas em que pode ser utilizada com sucesso. Também serão apresentadas definições e terminologias importantes como classes, características, parâmetros, entre outras. Além disso, as três principais categorias dos métodos de aprendizagem (supervisionada, não supervisionada e por reforço) são apresentadas. Por fim, são descritas as principais técnicas de aprendizagem de máquina tais como SVM (*Support Vector Machines*), *k*-NN (*k-Nearest Neighbors*), RNAs (Redes Neurais Artificiais), *k-Means*, *NaiveBayes*, conjuntos de classificadores, entre outras, aplicáveis na área de detecção de anomalias na Internet.

A Seção 2.3 discute o emprego de técnicas de aprendizagem de máquina em detecção de anomalias. A primeira seção deste tópico descreve uma série de desafios de uso e, orientações indicando quando e como usar aprendizagem de máquina na detecção de anomalias de forma apropriada. Em seguida, as técnicas de aprendizagem de máquina descritas na seção anterior são analisadas no contexto de aplicações de segurança, onde exemplos de aplicações práticas bem sucedidas de cada técnica são apresentados. Os principais problemas abordados são: *Cross-Site Scripting* (XSS), *phishing*, *malware*, negação de serviço e *spam*. Por fim, é apresentada uma síntese dessas aplicações, sumarizando questões como aplicabilidade, vantagens e desvantagens.

Por último, a Seção 2.4 apresenta objetos de pesquisa que devem crescer em relevância nos próximos anos, destacando o emprego de diferentes técnicas de aprendizagem de máquina na detecção de anomalias na Internet. São mencionadas as questões de pesquisa em aberto com o propósito de indicar novas oportunidades. Por fim, são apresentados os comentários finais sobre o tema.

2.2. Aprendizagem de Máquina

Entidades inteligentes destacam-se pela capacidade de adequar-se a novos ambientes e de resolver novos problemas. Um computador pode ser orientado a interpretar as informações recebidas de uma forma que melhore gradualmente seu desempenho [Rich e Knight, 1991]. Essa é a base na qual a área de pesquisa em aprendizagem de máquina está fundamentada.

2.2.1. Definição

Existem inúmeras atividades associadas à noção de aprendizagem, dificultando a definição exata do termo e tornando-o dependente de contexto. Porém, no contexto de

aplicações computacionais, uma definição muito precisa de aprendizagem de máquina pode ser encontrada em [Alpaydin, 2010]:

> *"... são programas de computador utilizados para otimizar um critério de desempenho, usando dados de exemplo ou experiência do passado".*

Independente da definição exata de aprendizagem é amplamente aceito na literatura que sistemas de aprendizagem confiáveis são de importância estratégica em diversas áreas de aplicação, pois há muitas tarefas que não podem ser solucionadas através do uso de técnicas de programação clássicas. Podemos citar como exemplo, a classificação de e-mails em duas classes: legítimos e s*pam*. Nesse tipo de problema a entrada é conhecida: um documento e-mail que em seu caso mais simples é um arquivo texto; e a saída também é conhecida: *spam* ou não *spam*. Porém, a forma como a entrada deverá ser convertida na saída é desconhecida.

Os algoritmos de aprendizagem de máquina, portanto, podem ser a chave para solucionar problemas dessa natureza, pois são algoritmos que podem "aprender" a definir padrões das classes envolvidas no problema, a partir de exemplos reais obtidos do ambiente. A Figura 2.4 mostra um modelo genérico de aprendizagem de máquina. O ambiente fornece informação para um elemento de aprendizagem que usa essa informação para fazer melhoramentos em uma base de conhecimento e então, um elemento de desempenho usa essa base para executar sua tarefa.

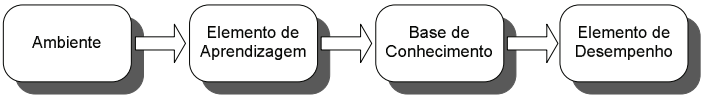

Figura 2.4. Modelo genérico de aprendizagem de máquina.

2.2.2. Categorização

Aprendizagem de máquina pode ser dividida em dois paradigmas fundamentais: **aprendizagem com professor** e **aprendizagem sem professor** [Haykin, 2008]. No primeiro, normalmente chamado **aprendizagem supervisionada**, o sistema precisa conhecer o ambiente. Esse conhecimento é representado por um conjunto de exemplos de pares de entrada-saída que são transmitidos em uma sequencia de instruções que o computador seguirá para alcançar o efeito desejado. Na aprendizagem sem professor, não há exemplos rotulados da função a ser aprendida. Nesse paradigma são identificadas duas subdivisões:

- **Aprendizagem de reforço**: aprendizagem a partir de um mapeamento de entrada-saída alcançada através de interação continuada com o ambiente para minimizar um índice escalar de desempenho [Haykin, 2008].
- **Aprendizagem não-supervisionada**: não há valores de saída desejada. A tarefa de aprendizagem envolve a obtenção de alguma compreensão do processo que gerou os dados e desenvolver habilidades para formar representações internas capazes de codificar características da entrada e, portanto, criar novas classes automaticamente [Haykin, 2008]. Esse tipo de aprendizagem inclui estimação de densidade, formação de agrupamentos, dentre outros.

2.2.3. Taxonomia

Para uma compreensão mais abrangente dos algoritmos de aprendizagem de máquina, é necessário que alguns dos termos e conceitos relevantes sejam definidos. As definições apresentadas a seguir, foram extraídas de [Kuncheva, 2004]:

1. Classes

Uma classe possui objetos similares, enquanto objetos de classes diferentes são dissimilares. Por exemplo, sites *phishing* podem ser categorizados como objetos da classe *phishing*, enquanto sites não *phishing* são objetos da classe *sites* legítimos. Algumas classes são facilmente definidas, por exemplo, um e-mail deverá obrigatoriamente ser *spam* ou não *spam.* Outras classes, porém, são de difícil definição, como por exemplo, a classe das pessoas destras e das pessoas não destras.

Resumidamente, pode-se definir que um problema de classificação de dados possui *c* classes, rotuladas de ω_1 a ω_c, organizadas em um conjunto de rótulos $\Omega = \{\omega_1,..,\omega_c\}$e que cada objeto pertence a apenas uma classe.

2. Características ou atributos

Os objetos que compõem as classes são descritos através de características ou atributos. As características podem ser nominais, tais como endereço e profissão, e podem ser numéricas, como tamanho da URL e quantidade de palavras-chaves. Os valores de características de um objeto *x* são organizados em um vetor *n*-dimensional $x = [x_1,...,x_n] \in \Re^n$. O espaço $\Re^n$ é chamado espaço de características, sendo que cada eixo corresponde a uma característica do objeto.

3. Base de Dados

A informação fundamental para algoritmos de aprendizagem de máquina, independentemente do tipo de aprendizagem, é proveniente dos dados disponíveis do problema. Essa informação normalmente está organizada na forma de um conjunto de dados $Z = \{z_1,...,z_N\}, z_j \in \Re^n$. Em problemas de aprendizagem supervisionada, o rótulo da classe de z_j é definido por $l(z_j) \in \Omega, j = 1,...,N$. Para construir uma base de dados, as instâncias do problema devem ser transformadas em vetores de características. Nem sempre essa tarefa é simples. Por exemplo, dadas diversas instâncias de sites com código *script*, não é uma tarefa trivial representar as páginas através de vetores de características como: tamanho de URL, existência de código ofuscado, etc. Entretanto, o uso de características relevantes para representar as instâncias do problema a ser tratado é fundamental para o sucesso do processo de classificação automática.

2.2.4. Técnicas de Aprendizagem de Máquina

São descritos nesta seção cinco (5) métodos de aprendizagem supervisionada: *k*NN, SVM, *Naive Bayes*, Redes Neurais Artificiais e Árvores de Decisão; um (1) método de aprendizagem não supervisionada: *k-means*; e métodos que utilizam a combinação de classificadores.

2.2.4.1. KNN (*K-Nearest Neighbor*)

KNN é uma técnica de classificação baseada em instâncias, isto é, consiste em atribuir a classe de cada elemento desconhecido (amostras de teste) a partir da classe majoritária obtida entre os seus vizinhos mais próximos identificados no conjunto de treinamento. A definição de vizinhança é feita segundo uma medida de similaridade que normalmente é uma medida de distância calculada no espaço de características. A distância Euclidiana é uma das medidas de similaridade mais utilizada na literatura [Theodoridis e Koutroumbas, 2006]. Essa medida é calculada pela seguinte fórmula,

$$d(x,y)=\sqrt{\sum_{i}^{n}(x_i-y_i)^2} \quad (1)$$

Onde *x* é um exemplo de treino, representado por $x=[x_1,...,x_n]\in\Re^n$ sendo *n* a dimensão do vetor de características, enquanto *y* representa a amostra de teste, sendo $y=[y_1,...,y_n]\in\Re^n$.

O algoritmo KNN é um dos mais simples dentre as técnicas de aprendizagem de máquina. O funcionamento desse algoritmo é descrito abaixo, de acordo com [Theodoridis e Koutroumbas, 2006]:

1. Defina um valor para *k*, ou seja, a quantidade de vizinhos mais próximos;
2. Calcule a distância da nova amostra a ser classificada a todas as amostras de treinamento;
3. Identifique os *k* vizinhos mais próximos, independentemente do rótulo das classes;
4. Conte o número de vizinhos mais próximos que pertencem a cada classe do problema;
5. Classifique a nova amostra atribuindo-lhe a classe mais frequente na vizinhança.

Este processo de classificação pode ser computacionalmente exaustivo quando o conjunto de treinamento possui muitos dados. Por isso, a grande desvantagem de KNN é a complexidade computacional envolvida na obtenção dos *k* vizinhos mais próximos. Por outro lado, KNN tem como uma de suas vantagens a independência quanto à distribuição de dados no espaço de características.

2.2.4.2. SVM (*Support Vector Machines*)

É uma técnica de classificação amplamente aplicada em problemas de segurança de redes tais como detecção de *phishing* [Miyamoto et al. 2009] e detecção de intrusos [Xiao *et al.,* 2007]. Basicamente, o funcionamento de SVM pode ser descrito da seguinte forma: dadas duas classes e um conjunto de instâncias de treinamento cujas amostras pertencem a essas classes, SVM constrói um hiperplano que divide o espaço de características em duas regiões, maximizando a margem de separação entre as mesmas. Esse hiperplano é conhecido como hiperplano de separação ótima. As amostras desconhecidas (exemplos de teste) são então mapeadas para esse mesmo espaço, e atribuídas a uma das classes [Alpaydim, 2010].

A Figura 2.5 mostra o hiperplano de separação ótima (reta separadora) para um problema bidimensional típico e linearmente separável. As retas pontilhadas H_1 e H_2, paralelas ao hiperplano, constituem o par de hiperplanos que geram a margem máxima pela minimização do vetor peso *w*. Além disso, $|b|/||w||$ é a distância perpendicular do hiperplano à origem e $||w||$ é a norma Euclidiana de *w*. Os pontos que estão em um dos hiperplanos H_1 e H_2 são chamados vetores de suporte. Esses pontos, indicados na Figura 2.5 por círculos extras, alteram a solução encontrada caso sejam removidos. Além disso, em fase de uso de SVM, apenas os vetores de suporte são necessários para que dados desconhecidos sejam classificados.

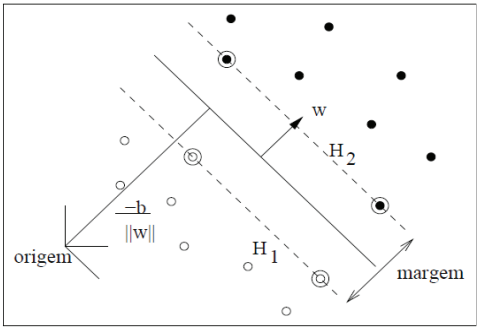

Figura 2.5. Hiperplano de separação ótima para um problema com duas classes.

O algoritmo original de SVM não encontra a solução desejada quando aplicado a dados não linearmente separáveis, característica presente na maioria dos problemas reais [Alpaydim, 2010]. Entretanto, tais problemas podem ser solucionados por SVM através da utilização de funções de separação de dados mais complexas do que funções lineares. Sendo assim, o uso de diferentes funções *kernel* possibilita a construção de SVM com diferentes tipos de superfícies de decisão não-linear no espaço de entrada.

Com o uso de funções *kernel,* as instâncias são inicialmente mapeadas para um espaço de características de maior dimensão que o espaço de características original. Permitindo dessa forma, a classificação em espaços não linearmente separáveis. A Figura 2.6 mostra o processo de transformação de um domínio não linearmente separável, em um problema linearmente separável através do aumento da dimensão, consequência do mapeamento feito por uma função *kernel* $F(x)$.

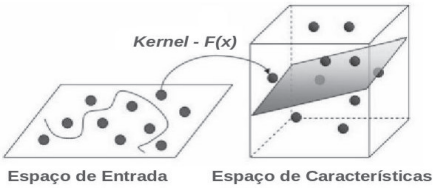

Figura 2.6. Mapeamento do espaço de entrada via função *kernel*.

Dentre as funções *kernel* mais usadas destacam-se: Polinômios, Funções de Base Radial (RBF) ou Gaussiana e Sigmóide, definindo diferentes máquinas de

aprendizagem conforme Tabela 2.3. Nessa tabela, *x* representa os vetores de suporte, enquanto *y* representa os dados de teste.

Tabela 2.3. Funções de *Kernel* mais utilizadas com SVM

Tipo de *Kernel*	Função $K(x,x_i)$	Tipo de Classificador
Polinomial	$[(x*y)+1]^d$	Máquina de Aprendizagem Polinomial
Gaussiano ou (RBF)	$\exp\left(-\frac{(x-y)^2}{2\sigma^2}\right)$	Rede RBF
Sigmoidal	$\tanh[\beta_0(x*y)]+\beta_1$	Perceptron de duas camadas

A principal vantagem de SVM é a baixa probabilidade de erros de generalização [Vapnik, 1995]. Quanto à utilização de SVM para detecção de intrusão, por exemplo, há duas vantagens principais. A primeira está relacionada à rapidez do algoritmo em fase de uso, uma vez que o desempenho em tempo real é de primordial importância para esse tipo de aplicação. A segunda razão é a escalabilidade, pois SVM é relativamente insensível ao número de pontos de dados. Dessa forma, a taxa de precisão na classificação não depende diretamente da dimensão do espaço de características [Mukammala *et al*., 2002]. Entretanto, SVM tem como desvantagem a demanda por elevada complexidade computacional na fase de treinamento, fato que pode inviabilizar o uso de SVM em problemas que necessitem de treinamento online.

2.2.4.3. *Naive Bayes*

Esse método de aprendizagem de máquina baseia-se em fundamentações simples, mas frequentemente produz elevada taxa de classificação em problemas reais. O modelo busca uma resposta para questionamentos como: "Qual a probabilidade do ataque ser de um determinado tipo, dado alguns eventos do sistema observado?" [Tsai et al., 2009].

Considerando uma instância $x = \{x_1, x_2, \dots, x_n\} \in \Re^n$, o problema de classificação tratado por *Naive Bayes* consiste em atribuir à amostra *x* uma das *c* classes envolvidas no problema, em que $\Omega = \{\omega_1, .., \omega_c\}$ é o conjunto de classes. Um erro mínimo de classificação pode ser obtido quando a classe com maior probabilidade *a posteriori* $P(\omega_i|x)$ é atribuída à *x*. Essa probabilidade é calculada através da fórmula de *Bayes* definida na Equação 2.

$$P(\omega_i|x) = \frac{P(\omega_i)p(x|\omega_i)}{\sum_{j=1}^{c} P(\omega_j)\, p(x|\omega_j)}, i = 1, \dots, c \qquad (2)$$

em que $P(\omega_i)$ é a probabilidade *a priori* de ocorrência da classe ω_i e $p(x|\omega_i)$ é a função de densidade de probabilidade condicional de cada classe (PDF– *Probability Distribution Function*).

A obtenção de uma estimação precisa das PDFs conjuntas é difícil, especialmente se a dimensão do espaço de características $\Re^n$ é elevada. Para superar essa dificuldade, *Naive Bayes* assume que as características são condicionalmente independentes. Neste caso, a PDF conjunta para uma classe dada é obtida da seguinte forma:

$$p(x|\omega_i) = \prod_{j=1}^{n} p(x_j|\omega_i), \qquad i = 1,..,c \tag{3}$$

Precisamente, a PDF conjunta é obtida através da verificação da frequência decorrência de cada valor de atributo para cada classe do problema nas amostras de treinamento. Além disso, a probabilidade *a priori* também pode ser obtida através da frequência de ocorrência de cada classe na base de treinamento. A hipótese de independência entre características pode parecer restritiva, mas resultados práticos encontrados na literatura em diversas áreas de aplicação mostram que *Naive Bayes* produz elevada taxa de classificação mesmo quando as características são claramente dependentes [Kuncheva e Hoare, 2008].

Baixa complexidade na fase de treinamento é uma das principais vantagens de *Naive Bayes*, uma vez que essa fase envolve apenas o cálculo de frequências para que as probabilidades sejam obtidas. Essa característica torna *Naive Bayes* indicado para aplicações online, tais como problemas envolvendo segurança de redes cujo treinamento precisa ocorrer de forma online e com frequência regular. Outra característica que torna *Naive Bayes* atrativo para a área de segurança de redes é o fato de possibilitar a manipulação de atributos nominais e numéricos. Atributos nominais são frequentes em detecção de *spam*, *cross-site scripiting*, páginas *phishing*, dentre outros problemas de classificação de documentos textuais.

Naive Bayes tem sido usado para detecção de anomalia em definições de múltiplas classes [Chandola et al., 2009]. Diversas variantes da técnica básica foram propostas para detecção de intrusão em redes de computadores através de detecção em vídeo de vigilância, detecção de anomalia em dados de texto e para detecção de surto de doenças [Chandola et al., 2009].

2.2.4.4. Redes Neurais Artificiais

Redes Neurais Artificiais ou simplesmente redes neurais (ou neuronais) são inspiradas no sistema nervoso biológico. Tal inspiração deve-se à seguinte razão: o cérebro é um dispositivo de processamento de informação com inúmeras habilidades, como por exemplo, visão, reconhecimento de voz, etc. [Haykin, 2008]. Com as redes neurais busca-se, portanto, definir soluções algorítmicas para os mesmos problemas solucionados pelo cérebro.

Inicialmente, rede neural foi proposta com a expectativa de ser um método computacional massivamente paralelo, tal qual o cérebro humano. Porém, com a evolução das pesquisas, esse objetivo tornou-se remoto. Por outro lado, a técnica baseada em rede neural é atualmente uma ferramenta de aprendizado de máquina importante, na teoria e na prática, em muitas áreas de aplicação. Pode-se dizer que as redes neurais artificiais assemelham-se ao cérebro humano em dois (2) aspectos:

1. O conhecimento é adquirido pela rede através de um processo de aprendizado;
2. Interconexões entre os neurônios são usadas para armazenar o conhecimento.

Diversos protocolos e algoritmos de treinamento foram e ainda são desenvolvidos, fator que é a chave do sucesso das redes neurais [Kuncheva, 2004].

Alguns dos principais tipos de redes são: Função de Base Radial (RBF); redes de Hoppfield; *Adaptive Resonant Theory* (ART); Redes de Kohonen (*Self Organizing Map* - SOM); *Perceptron* de uma única camada e Perceptron Multi-Camadas (MLP). Nesta seção, a rede MLP é descrita sucintamente. Trata-se, provavelmente, do tipo de rede neural mais utilizado [Kuncheva, 2004]. Antes, porém, é necessário descrevermos o componente fundamental da rede, neurônio artificial.

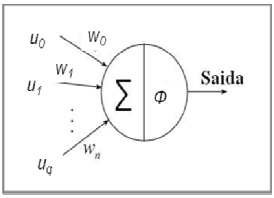

Figura 2.7.Esquema básico do processamento de um neurônio

Os neurônios são as unidades de processamento no cérebro humano. Da mesma forma, os neurônios artificiais, também chamados de nós, são as unidades de processamento de uma rede neural. Considerando $u = [u_0, \dots, u_q]^T \in \Re^{q+1}$ o vetor de entrada do neurônio (os vetores de entrada iniciais são as amostras do conjunto de treinamento) e $v \in \Re$ sua saída, o vetor $w = [w_0, \dots, w_q]^T \in \Re^{q+1}$ é chamado vetor de pesos sinápticos. Conforme pode ser observado na Figura 2.7, o neurônio executa um processamento que ocorre através do cálculo da seguinte função:

$$v = \phi(\xi); \qquad \xi = \sum_{i=0}^{q} w_i u_i \qquad (4)$$

em que $\phi: \Re \to \Re$ é a função de ativação e ξ é o somatório da rede. As escolhas mais comuns de ξ são:

Função Limiar: $\phi(\xi) = \begin{cases} 1, & se\ \xi \geq 0, \\ 0, & caso\ contrário \end{cases}$

Função Sigmóide: $\phi(\xi) = \frac{1}{1+\exp(-\xi)}$

Função identidade: $\phi(\xi) = \xi$

O peso w_0 é usado como *bias* (tendência), e sua entrada correspondente u_ié normalmente definido como 1. Com essa nova informação, a equação 4 pode ser reescrita como:

$$v = \phi\left[\sum_{i=1}^{q} w_i u_i - (-w_0)\right] \qquad (5)$$

Geometricamente, a equação 6 define um hiperplano em $\Re^q$.

$$\sum_{i=1}^{q} w_i u_i - (-w_0) = 0 \qquad (6)$$

Considerando um problema de classificação com apenas duas classes envolvidas, um neurônio com função de ativação limiar, por exemplo, atribui o valor +1 para as amostras que se localizam de um dos lados do hiperplano, enquanto as demais amostras recebem valor 0.

Rede Perceptron Multicamadas (MLP)

É uma rede composta por três grupos de camadas: (1) camada de entrada - recebem as amostras de dados na forma de um vetor de características; (2) camadas escondidas – processam os dados de entrada e podem compor várias camadas; e (3) camada de saída – processa a saída da rede. A figura 2.8 mostra um modelo genérico de uma rede MLP. Nesse exemplo, a arquitetura da rede é composta por apenas uma camada escondida. Além disso, a estrutura das redes MLP é progressiva, ou seja, as saídas de cada camada são enviadas progressivamente às camadas subsequentes. O número de camadas escondidas e o número de neurônios em cada camada escondida não são limitados. As características mais comuns são [Kuncheva, 2004]:

- A função de ativação ϕ na camada de entrada é a função identidade.
- Não há conexões laterais entre os nós da mesma camada.
- Camadas não adjacentes não são conectadas diretamente.
- Todos os nós de todas as camadas escondidas têm a mesma função de ativação ϕ.

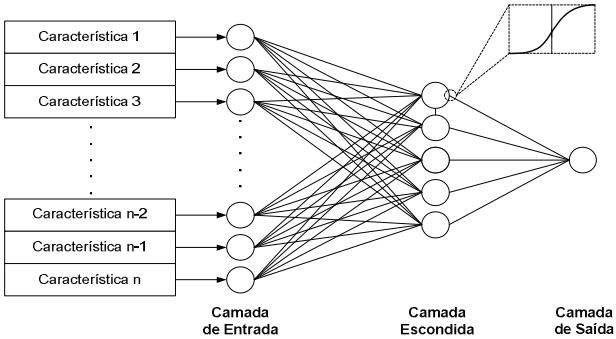

Figura 2.8. Modelo genérico de um classificador do tipo MLP.

O algoritmo usado para o treinamento do classificador MLP é o algoritmo de retro-propagação (*backpropagation)* descrito abaixo, de acordo com [Kuncheva, 2004]:

1. Defina uma arquitetura para a rede: número de camadas escondidas, número de neurônios em cada camada e funções de ativação;
2. Inicialize os pesos aleatoriamente (incluindo *bias*). Defina um número máximo T de iterações e um valor de erro de treinamento máximo $\epsilon > 0$;
3. Enquanto (erro de treino $>\epsilon$ e iteração $\leq$ T)
   a) Submeta a próxima amostra de treinamento z_j;

b) Calcule a saída de cada neurônio da camada de saída da rede usando os valores dos pesos atuais;

c) Calcule o erro de cada neurônio de saída.

d) Atualize os pesos de cada neurônio da camada de saída e das camadas escondidas em função do erro.

e) Calcule o erro de treino.

f) Quando toda a base de treino for submetida à rede, incremente o número de iterações.

4. Fim

Redes neurais têm sido aplicadas em detecção de intrusão [Chandola et al., 009], detecção de DDoS [Gavrilis e Dermatas, 2005] e em outros problemas de segurança de redes. Uma das principais vantagens do uso de redes neurais é a elevada taxa de detecção normalmente alcançada pela rede (especialmente em situações com muitas amostras de treinamento disponíveis). Além disso, apresenta baixa complexidade na fase de uso, uma vez que a classificação de uma amostra desconhecida é feita através de um simples produto entre os atributos da amostra e os vetores de peso da rede obtidos durante o treinamento. Porém, o classificador baseado em rede neural é pouco indicado para aplicações cujo treinamento ocorre de forma online devido à elevada complexidade computacional envolvida no treinamento da rede. Outro fator negativo, especialmente em MLP, é o elevado número de parâmetros que devem ser definidos experimentalmente pelo usuário, tais como: número de camadas escondidas, número de neurônios em cada camada escondida, valores inicias dos pesos, número de iterações, dentre outros.

2.2.4.5. Árvores de Decisão

Árvore de decisão é uma das principais técnicas de aprendizagem de máquina, especialmente quando problemas de classificação envolvem atributos nominais. Uma DT (do Inglês *Decision Tree*) é composta por três elementos básicos:

- Nó raiz: corresponde ao nó de decisão inicial que normalmente é gerado utilizando-se o atributo mais discriminante entre as classes envolvidas no problema;
- Arestas: correspondem aos diferentes valores possíveis das características;
- Nó folha: corresponde a um nó de resposta, contendo a classe a qual pertence o objeto a ser classificado.

Em árvores de decisão, duas grandes fases devem ser asseguradas:

1. Construção da árvore: uma árvore de decisão é construída com base no conjunto de dados de treinamento, sendo dependente da complexidade dos dados. Uma vez construída, regras podem ser extraídas através dos diversos caminhos providos pela árvore para que sejam geradas informações sobre o processo de aprendizagem.
2. Classificação. Para classificar uma nova instância, os atributos da amostra são testados pelo nó raiz e pelos nós subsequentes, caso seja necessário. O

resultado deste teste permite que os valores dos atributos da instância dada sejam propagados do nó raiz até um dos nós folhas. Ou seja, até que uma classe seja atribuída à amostra.

Para melhor ilustrar o processo de classificação de uma DT, a Figura 2.9 apresenta um modelo utilizado no treinamento de dados para a tarefa de detecção de intrusão. Inicialmente, uma base de dados supervisionada é submetida ao algoritmo, que por sua vez, gera uma árvore construída para separar amostras da classe "Benigna" de amostras da classe "Maligna". Após a construção da árvore, dados desconhecidos podem ser submetidos ao classificador, o qual atribuirá um das duas classes à amostra testada.

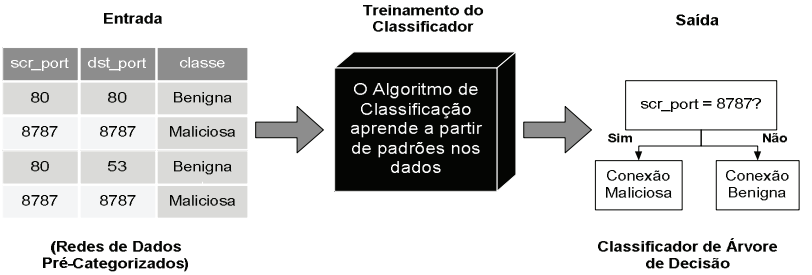

Figura 2.9: Treinamento dos dados em uma Árvore de Decisão.

Vários algoritmos foram desenvolvidos a fim de assegurar a construção de árvores de decisão e seu uso para a tarefa de classificação. O ID3 e C4.5 algoritmos desenvolvidos por [Quinlan, 1993] são provavelmente os mais populares. Vale também mencionar o algoritmo CART de Breiman [Breiman et al., 1984]. A maioria desses algoritmos utiliza uma estratégia descendente, ou seja, desde a raiz até as folhas. A seguir são exibidos detalhes desses algoritmos, de acordo com [Duda et al., 2001].

- ID3 (*Iterative Dichotomiser 3*) é um modelo de DT para dados nominais (não ordenados), embora seja possível a utilização do ID3 com dados contínuos, desde que sejam convertidos em valores discretos. É utilizada uma medida que calcula o ganho de informação para dividir os atributos ao longo da árvore. O algoritmo padrão não possibilita que a árvore gerada seja podada, ação necessária para evitar aprendizagem viciada e para reduzir a complexidade da árvore gerada. Porém, essa função pode ser incorporada ao algoritmo.

- C4.5 é o sucessor e um refinamento do algoritmo ID3, desenvolvido por Quinlan [Quinlan 1993]. O algoritmo usa heurísticas para podar as árvores geradas. Outra vantagem deste algoritmo é o fato de aceitar atributos contínuos e categorizados na construção da DT. C4.5 usa o método do ganho da relação de impurezas para avaliar a divisão dos atributos.

- CARTs (*Classification on Regression Trees*) foi proposto em [Breiman et al., 1984]. Uma das suas características principais é a grande capacidade de pesquisa das relações entre os dados, mesmo quando elas não são evidentes, bem como a produção de resultados sob a forma de árvores de decisão de grande simplicidade e legibilidade. Tal como o seu nome indica, CART pode

ser usado em problemas de classificação (classes discreta) ou regressão (valores contínuos) usando uma mesma tecnologia. O resultado deste algoritmo é sempre uma árvore binária que pode ser percorrida da sua raiz até as folhas respondendo apenas a questões simples do tipo sim/não. A análise é efetuada de forma completamente automática requerendo uma intervenção humana mínima.

Uma das vantagens do uso de Árvores de Decisão em aplicações envolvendo problemas de segurança em redes de computadores é o fato de ser um método não paramétrico, ou seja, não há necessidade de definição de valores de parâmetros. Outra propriedade interessante é que regras podem ser extraídas a partir das árvores. Essas regras explicam claramente o processo de aprendizagem, podendo ser usadas para uma compreensão mais completa dos dados e dos atributos mais relevantes para o problema de classificação. Entretanto, é difícil obter resultados exatos quando há ruídos, devido aos inúmeros detalhes dos dados definidos durante o treinamento. Vale também ressaltar que esta técnica permite a obtenção de resultados, que em geral, são superados apenas por algoritmos de complexidade muito superior.

Além do uso de DTs para extração de regras que explicam o aprendizado do dados, métodos unicamente baseados em indução de regras, como o método RIPPER, também podem ser aplicados em problemas de detecção de anomalias, como em [Likarish et al., 2009].

2.2.4.6. *K-means*

Algoritmos de agrupamento são geralmente usados de forma não supervisionada. Ou seja, é apresentado um conjunto de instâncias de dados que devem ser agrupados de acordo com alguma similaridade. O algoritmo tem como informação de entrada somente o conjunto de características que descrevem cada instância dada.

K-means [MacQueen, 1967] é um popular algoritmo de agrupamento comumente usado para particionar automaticamente um conjunto de dados em *k* grupos. Dado o conjunto de amostras $Z = \{z_1, ..., z_N\}, z_j \in \Re^n$. O usuário deverá definir a quantidade de grupos *c* que *k-means* criará, sendo que para cada grupo haverá um centróide $\mu_1, \mu_2, \ldots, \mu_c$. São selecionados inicialmente *k* centróides, os quais são iterativamente refinados pelo algoritmo da seguinte forma:

1. Inicialize os centróides $\mu_1, \mu_2, \ldots, \mu_c$.
2. Agrupe cada uma das *N* amostras ao centróide mais próximo.
3. Calcule novamente o valor de cada centróide μ_i.
4. Repita os passos 2 e 3 até que não ocorra mais mudanças em μ_i.

As *k* primeiras amostras da base são normalmente escolhidas como os *k* centróides iniciais, enquanto que medidas de distância, como a distância Euclidiana, são usadas para o cálculo da similaridade entre as demais amostras e os centróides de cada grupo. O cálculo das distâncias é, portanto, a etapa que mais exige processamento. Se existem *N* pontos e *k* centróides calcula-se *N* x *k* distâncias neste passo. O centróide mais próximo de cada amostra vai 'incorporá-la', ou seja, a amostra vai pertencer ao grupo representado pelo mesmo. Após, os valores das coordenadas dos centróides são refinados, pois para cada grupo que possui mais de uma amostra, o novo valor do

centróide é calculado através da média dos atributos de todas as amostras do grupo. Esse passo é repetido até a convergência, isto é, o algoritmo calcula a distância entre as amostras da base e os centroides iterativamente, até que mais nenhuma amostra mude de classe.

Chandola et al. [Chandola et al, 2009] destaca que a utilização de técnicas baseadas em agrupamento proporciona algumas vantagens como a operacionalização de maneira não supervisionada; sendo frequentemente adaptada a tipos complexos de dados e; rapidez na fase de teste, desde que o número de grupos com que cada instância precisa ser comparada seja uma constante pequena. Porém, ainda segundo os autores, o uso destas técnicas acarreta desvantagens, como por exemplo, em relação ao desempenho, onde há uma enorme dependência da eficácia dos algoritmos em capturar a estrutura central das instâncias. Além disso, é difícil de ser calculada por não existir conhecimento sobre a saída desejada. Outra desvantagem se deve a complexidade computacional para agrupamento dos dados, sendo frequentemente um gargalo, especialmente para aplicações online como os problemas de segurança em redes de computadores.

2.2.4.7. Combinação de Classificadores

A combinação de classificadores tenta superar a difícil tarefa de construir um único classificador para resolver problemas específicos, através da combinação da decisão de classificadores menos específicos, ou menos definidos para o problema a ser solucionado. Existem diversas razões para a combinação de múltiplos classificadores, algumas dessas razões são encontradas em [Jain et al., 2000]:

1. Um desenvolvedor tem acesso a diferentes classificadores, treinado sem um contexto diferente e para uma representação ou descrição inteiramente diferente do mesmo problema;
2. Muitas vezes, mais de um conjunto de treino é disponível, coletados em um tempo diferente ou em um ambiente diferente. Esses conjuntos de treinamento podem ainda usar características diferentes;
3. Diferentes classificadores treinados sobre o mesmo dado não diferem somente em seu desempenho global. Possivelmente, os classificadores apresentam fortes diferenças locais. Cada classificador tem sua própria região no espaço de características na qual atua com um desempenho melhor;
4. Alguns classificadores, tais como redes neurais, mostram diferentes resultados com diferentes inicializações dos parâmetros devido à aleatoriedade inerente ao procedimento de treino. Nesse tipo de situação, normalmente é feita uma escolha da melhor rede e descarte das demais. Porém, é possível combinar varias redes, na tentativa de fortalecer a aprendizagem dos dados.

Portanto, diferentes conjuntos de características, diferentes conjuntos de treinamento, diferentes métodos de classificação, ou diferentes sessões de treino, podem resultar em um conjunto de classificadores cuja saída é combinada buscando-se melhorar a precisão da classificação global.

Classificadores híbridos e conjuntos de classificadores podem ser considerados dois tipos de combinação de classificadores. A idéia básica de classificadores híbridos [Tsai et al., 2009] é combinar diversas técnicas de aprendizagem de máquina no intuito de melhorar o desempenho geral do sistema. Mais especificamente, uma abordagem hibrida é composta por dois componentes funcionais. O primeiro usa os dados de entrada e gera um resultado intermediário. O segundo atua nos resultados intermediários para produzir o resultado final.

Em [Peddabachigare et al., 2007] tem-se uma abordagem de combinação de classificadores híbridos para detecção de intrusão. É utilizado DT na primeira fase da classificação, resultando em uma saída intermediária. Na segunda fase, SVM é aplicado para que uma saída final seja obtida. A Figura 2.10 mostra a arquitetura desta combinação de classificadores híbridos. DT gera um nó informação com o conjunto de características inicial. Esse nó informação é determinado por regras geradas pela DT. Todo conjunto de dados é assinalado a um nó terminal o qual representa uma classe particular ou um subconjunto. A informação do nó terminal é adicionada ao conjunto de atributos original. Esse novo vetor de características é então manipulado por SVM para a classificação final.

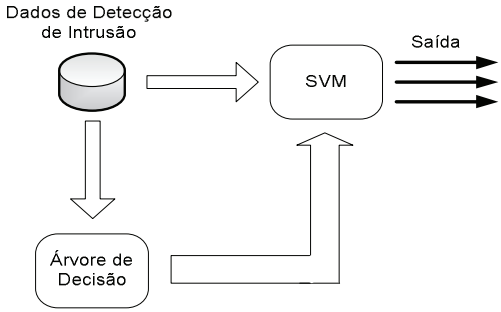

Figura 2.10. Uma abordagem para combinação de classificadores híbridos. [Peddabachigare et al., 2007]

A estratégia de conjuntos de classificadores refere-se à combinação de algoritmos de aprendizagem fraca. Os classificadores são chamados fracos porque são treinados sem definições ideais de parâmetros, com diferentes conjuntos de treino e ou de características, dentre outras estratégias. O objetivo é que o desempenho global seja efetivamente melhorado, quando comparado ao desempenho individual dos membros do conjunto [Tsai et al., 2009].

Existem três estratégias clássicas para a geração de conjuntos de classificadores: (1) *Adaboost;* (2) *Bagging;* e (3) Subespaços Aleatórios [Kuncheva, 2004]. A primeira estratégia é um método iterativo, que cria um conjunto de classificadores da seguinte forma sequencial: cada amostra recebe uma probabilidade (peso) de ser selecionada para compor a base de treinamento usada pelos membros do conjunto. A cada nova iteração, os pesos das amostras incorretamente classificadas pelos membros anteriores são ajustados, para que essas amostras tenham maior probabilidade de serem selecionados para compor a base de treinamento dos classificadores posteriores. *Bagging* também atua fazendo uma seleção de amostras de treinamento. Porém, o

processo de seleção das amostras é totalmente aleatório, ou seja, para cada classificador, é feita uma seleção aleatória das amostras presentes na base de treinamento. Por fim, o método Subespaços aleatórios altera os vetores de características usados por cada membro do conjunto. Cada classificador é treinado com um subconjunto de características escolhidas aleatoriamente.

A técnica conhecida como Florestas Aleatórias, tem sido aplicada em vários problemas de detecção de anomalias [Abu-Nimeh et al., 2007] [Debarr e Wechsler, 2009]. Essa técnica combina *Bagging* e Subespaços Aleatórios para gerar diferentes conjuntos de treinamento, usados para treinar um conjunto de DT. Outro exemplo de método aplicado ao problema de detecção de anomalias encontra-se em [Peddabachigare et al., 2007]. A Figura 2.11 mostra a arquitetura proposta pelos autores utilizando DT, SVM e a abordagem híbrida (DT – SVM). Esta abordagem de combinação de conjunto de classificadores procura usar a pontuação mais alta como saída final entre os classificadores (DT, SVM e DT-SVM).

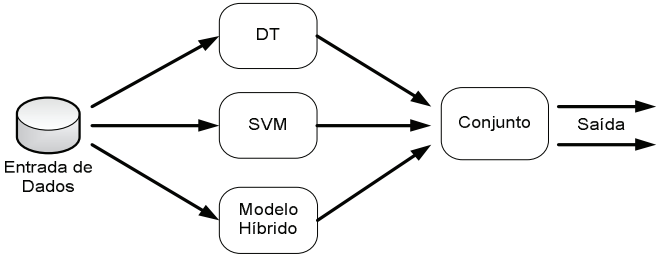

Figura 2.11. Uma abordagem para combinação de conjuntos de classificadores [Peddabachigari et al., 2007]

A combinação de classificadores, seja através de conjuntos ou de classificadores híbridos, tem apresentado resultados muito promissores em trabalhos que investigam segurança em redes de computadores. A principal vantagem desse método está no fato de que os membros do conjunto normalmente não precisam ter seus parâmetros internos ajustados para o problema. Segundo, sistemas de segurança de redes podem ser desenvolvidos utilizando-se conjuntos de classificadores para que dados provenientes de diferentes fontes sejam considerados em conjunto. Por fim, a literatura indica que conjuntos de classificadores normalmente superam as taxas de classificação obtidas por classificadores individuais. A complexidade computacional envolvida na geração e uso de um conjunto de classificadores pode ser considerada a principal desvantagem do método.

As técnicas de aprendizagem de máquina descritas nesta seção estão sendo utilizadas no desenvolvimento de diversos sistemas aplicados a problemas relacionados à segurança de redes de computadores. É importante destacar que muitas outras técnicas também podem ser aplicadas, tais como Regressão Logística, BART (*Bayesian Addictive Regression Trees*), HMM (*Hidden Markov Models*), métodos baseados em aprendizagem por reforço e outros métodos de aprendizagem não supervisionada. Alguns exemplos do uso de aprendizagem de máquina em problemas de segurança de redes são apresentados na próxima seção.

2.3. Aprendizagem de Máquina na área de Segurança na Internet

Nesta seção será examinado o uso de aprendizagem de máquina no domínio de prevenção e detecção de anomalias (também chamado de detecção de intrusão).

As subseções subsequentes discutem os desafios e problemas, bem como as recomendações de uso da aprendizagem de máquina na detecção de anomalias.

2.3.1. Os desafios do uso de aprendizagem máquina na detecção de anomalias

Quando comparada a outras áreas ou domínios de conhecimento, a aprendizagem de máquina aplicada na detecção de anomalias em redes de computadores parece ser mal sucedida. Isso porque, embora existam centenas de pesquisas acadêmicas, o número de ferramentas e soluções resultantes e efetivamente operacionais é muito reduzido. Quais seriam as razões que explicam tal fato? Existem limitadores ou até mesmo culpados?

Respostas a essas perguntas são elucidadas em grande parte nos trabalhos de [Sommer e Paxson, 2010], [Nelson, 2010] e [Tygar, 2011], que identificam algumas características especiais que tornam o emprego da aprendizagem de máquina mais difícil na detecção de anomalias do que em outros domínios. Essas características são:

1. **Detecção de *Outliers***

Analisando o objetivo da detecção de anomalias (identificar atividades que não pertencem a determinado contexto) é possível estabelecer uma relação com o conceito de *outlier*, uma observação que parece ser inconsistente com o restante de um conjunto de dados [Barnett e Lewis, 1978]. Assim, é possível dizer que a detecção de anomalias é a descoberta de *outliers* significativos que representam um comportamento anormal [Sommer e Paxson, 2010]. Nesse contexto, a detecção de *spam* é um exemplo extremamente adequado, visto que o objetivo é classificar e-mails não válidos (*spam*), ou seja, fora da normalidade.

Entretanto, usar técnicas de aprendizagem de máquina na detecção de anomalias para encontrar *outliers* parece contraditório, pois uma regra básica da aprendizagem de máquina é a necessidade de treinar o sistema com um conjunto grande de dados que englobe todos os tipos (espécimes) de todas as classes [Duda et al., 2001]. No entanto, como na detecção de anomalias o objetivo é encontrar novos ataques, não se pode "treinar" os ataques de interesse, uma vez que somente o tráfego normal (sem ataque) é conhecido. Em outras palavras, o treinamento de um sistema de detecção de anomalias acaba com um resultado oposto do esperado.

2. **Alto Custo dos Erros**

Em geral, um sistema de detecção de anomalias é regido por limites rigorosos sobre o número de erros que pode tolerar. Um falso positivo requer gastos (principalmente tempo e dinheiro) para examinar o incidente relatado, o que por muitas vezes acaba por não revelar nada. Por outro lado, falsos negativos têm o potencial de causar sérios danos, já que um único sistema comprometido pode prejudicar gravemente a integridade da infraestrutura de TI (Tecnologia da Informação) de toda uma empresa. Desta forma, o uso da aprendizagem de máquina na detecção de anomalias acrescenta um custo elevado no caso de erros de classificação, principalmente quando comparado a outros domínios de aplicação como, por exemplo, sistemas de recomendação. Uma exceção é a detecção de *spam*, cujo modelo de custo é altamente desequilibrado: falsos

positivos (*e-mails* válidos declarados como *spam*) podem custar caro, como a perda de um negócio, enquanto falsos negativos (*spam* identificados como *e-mails* válidos) não têm impacto significativo.

3. Diferença semântica

Sistemas de detecção de anomalias enfrentam um problema de diferença semântica, ou seja, a dificuldade em transformar seus resultados em ações para os operadores de rede. Em muitos estudos, observa-se uma tendência em limitar a avaliação do sistema apenas à capacidade de identificar, confiavelmente, os desvios de um perfil normal. Ao fazer isso, o resultado é que o operador é mais exigido e, muitas vezes é obrigado a se perguntar: o que isso significa?

Mesmo que, por definição, a detecção de anomalias seja focada em identificar comportamento malicioso (se um evento não visto antes é benigno ou não), não se pode parar nesse ponto, afinal, em ambientes operacionais, sistemas de detecção de anomalias produzem muitos falsos positivos. Vale lembrar que um algoritmo de aprendizado de máquina não comete erros dentro de seu modelo de normalidade e que no fim o que importa para o operador de rede é a interpretação dos resultados.

Por fim, é preciso considerar que existe uma diferença semântica entre o ambiente acadêmico e redes operacionais. Tipicamente, redes operacionais possuem políticas de segurança local (restrições de segurança). Assim, é muito provável que a atividade de um sistema de detecção de anomalias operado em um ambiente acadêmico seja proibida em uma rede corporativa.

4. Diversidade do Tráfego de Rede

Ao contrário do que muitos usuários e administradores de redes imaginam, o tráfego de uma rede apresenta grande diversidade, devido, por exemplo, a fatores como a correlação entre determinado tipo tráfego e a transferência de grandes volumes (*flash crowds*, por exemplo). Essa "inesperada" diversidade de tráfego é muitas vezes regular e comum, e na maioria dos casos não apresenta malefícios às redes. Entretanto, para um sistema de detecção de anomalias, essa variabilidade pode ser manipulada com dificuldade, pois torna a tarefa de encontrar uma noção estável de "normalidade" mais complexa.

A solução mais comum para que um sistema que usa aprendizagem de máquina reconheça a variabilidade como algo corriqueiro é através da agregação do tráfego. A idéia é estabilizar as propriedades do tráfego observando-as em períodos mais longos de tempo (horas e dias, talvez semanas) ao invés de intervalos pequenos (segundos e minutos). Observações feitas durante "horas do dia" e "dias da semana" exibem padrões mais confiáveis. Por exemplo, se durante o intervalo do almoço de um dia, o volume de tráfego é duas vezes maior do que durante os intervalos de tempo correspondentes a última semana, isso provavelmente reflete algo incomum acontecendo.

5. Dificuldades com a Avaliação

A avaliação de um sistema de detecção de anomalias é essencial para comprovar sua eficácia e eficiência. Muitas abordagens promissoras acabam, quando postas em prática, abaixo das expectativas. Por outro lado, a correta elaboração de um esquema de avaliação não é fácil e, muitas vezes, acaba sendo mais difícil do que a construção da própria solução. Devido à opacidade do processo de detecção, os resultados de um

sistema de detecção de anomalias são mais difíceis de prever. Basicamente, três (03) fatores contribuem para isso:

- **Dificuldades de Dados**: O maior desafio para avaliação de sistemas de detecção de anomalias é a falta de conjuntos de dados públicos adequados para testes e validação. A principal causa dessa falta é a "sensibilidade" dos dados. Tipicamente, a inspeção do tráfego de rede pode revelar informações sensíveis como dados pessoais, comunicações confidenciais e segredos de negócios. Diante desse risco, os "donos" do tráfego de rede se sentem ameaçados e criam barreiras organizacionais e legais que impedem sua utilização. Para burlar a falta de dados, pesquisadores têm adotado três estratégias [Sommer e Paxson, 2010]: *i) Simulação* - livre de preocupações quanto à sensibilidade dos dados, mas dificilmente é realista; *ii) Sanitização* (*sanitization*) - informações potencialmente sensíveis são removidas ou "anonimizadas" de dados reais, entretanto, ainda persiste o temor, por parte dos donos, que alguma informação confidencial possa tornar-se pública; e *iii) criação de conjuntos de dados próprios* - dados são gerados em ambientes controlados e o tráfego acaba se tornando muito diferente do real.
- **Diferença semântica**: Além de identificar corretamente os ataques, um sistema de detecção de anomalias precisa também dar suporte ao operador na compreensão da anomalia, permitindo assim uma rápida avaliação do seu impacto. Sempre que o sistema encontra, corretamente, um *exploit* para servidor *Web* desconhecido, mas apenas repassa ao operador a mensagem "o tráfego HTTP do *host* não corresponde ao perfil normal", um esforço adicional é necessário para entender o que aconteceu, mesmo que haja total confiança nos resultados do sistema. Por outro lado, usando novamente o exemplo do *spam*, se o detector informa que um e-mail é *spam*, não há muito espaço para interpretações equivocadas.
- **Configuração contraditória**: Sistemas de detecção de anomalias operam em ambientes contraditórios, em uma verdadeira briga entre gato e rato (atacantes e defensores),onde cada grupo tenta aperfeiçoar suas ferramentas em resposta à elaboração de novas técnicas providas pelo grupo rival. Uma preocupação em particular são as técnicas de evasão, nas quais os atacantes tentam ajustar suas atividades para evitar a detecção. Embora autores como [Sommer e Paxson, 2010], [Nelson, 2010] argumentem que explorar a especificidades de um sistema de detecção de anomalias que usa aprendizagem de máquina requer significativo esforço, tempo e *expertise* do atacante, Fogla e Lee [Fogla e Lee, 2006] provaram, através de uma abordagem automatizada, que é possível gerar ataques mutantes que combinam exatamente com o perfil normal de um sistema.

2.3.2. Quando e Como usar Aprendizagem de Máquina na Detecção de Anomalias

Após ler a subseção anterior e se deparar com vários problemas e dificuldades, o leitor deste minicurso deve estar se perguntando: será que vale a pena usar aprendizagem de máquina para encontrar anomalias em redes de computadores? A resposta é sim.

Para isso, a primeira coisa a ser feita é entender o que o sistema faz ou irá fazer Tipicamente, quando uma solução consegue resultados melhores que outros trabalhos, usando os mesmos dados, é óbvio aceitá-la como uma contribuição. Entretanto, no domínio de detecção de anomalias, é sempre possível achar uma variação de solução que funcione um pouco melhor do que outra em uma determinada configuração. Assim, o ponto chave para o uso da aprendizagem de máquina na detecção de anomalias é a percepção (visão) do comportamento do sistema. Sommer e Paxson [Sommer e Paxson, 2010] enumeram algumas recomendações para o uso de aprendizagem de máquina na detecção de anomalias.

1. Compreender o Modelo da Ameaça

Ao iniciar o desenvolvimento de um detector de anomalia é preciso considerar as ameaças às quais a solução será submetida. Para tanto, algumas perguntas precisarão ser respondidas:

- **Que tipo de ambiente é o alvo do sistema?** A operação em uma rede pequena enfrenta desafios bem diferentes do que para uma grande empresa ou *backbone*. Ambientes acadêmicos são diferentes de empresas comerciais.
- **Quais habilidades e recursos terão os atacantes?** Se um sítio *Web* é considerado de alta importância e, consequentemente, de alto risco a ataques, é preciso se antecipar aos ataques mais sofisticados.
- **Qual o grau de preocupação com evasão?** O grau com o qual os atacantes podem analisar as técnicas de defesa e tentar contorná-las determina os requisitos de robustez para qualquer detector de anomalias.

Em resumo, embora não existam detectores perfeitos, melhores decisões podem ser tomadas quando existe um modelo de ameaças ao sistema.

2. Manter o âmbito restrito

Imaginar que a aprendizagem de máquina é a "lâmpada mágica" e que seu emprego em uma solução de detecção de anomalias é garantia de sucesso é um equívoco comum. Na detecção de anomalias é essencial ter a visão clara dos objetivos da solução proposta, isto é, quais ataques deverão ser detectados. A forma mais simples e direta de se alcançar essa visão é restringindo atividade alvo. Assim, pode-se adaptar o detector as especificidades e reduzir o potencial de erros de classificação. Uma dica para escolher qual algoritmo ou técnica usar é sempre ser capaz de responder o porquê da escolha.

3. Reduzir custos

O principal argumento para o uso de sistemas de detecção de anomalias é a redução dos custos associados com segurança. Contudo, de forma curiosa, a principal queixa sobre sistemas de detecção de anomalias é o número excessivo de falsos positivos, o que aumenta os custos. Uma solução para diminuir custos é reduzir o escopo do sistema, uma vez que sem um objetivo claro a configuração do problema com aprendizagem de máquina terá impacto direto no número de falsos positivos. Outra solução é o uso de estratégias para lidar com a diversidade do tráfego. O uso de esquemas de agregação, que usam intervalos de tempo maiores, e o exame cuidadoso das propriedades particulares do tráfego têm se mostrado bastante úteis. Por fim,

esquemas de pós-processamento com suporte a informações adicionais também têm ajudado a reduzir os falsos positivos [Gu et al., 2007] [Anagnostakis et al., 2005].

4. Avaliação

A avaliação de um sistema de detecção de anomalias deve responder as seguintes perguntas: O que ele pode detectar e por quê? O que não é possível detectar e por que não? Quão confiável ele funciona? Onde pode falhar?

De acordo com a experiência de [Sommer e Paxson, 2010] como revisores, a razão número 1 de rejeições em submissões a conferências de detecção de anomalias reside em falhas ao se explorar adequadamente estas questões em termos de como trabalhar com dados e interpretação de resultados.

- **Como trabalhar com os dados**: Como previamente discutido, o passo mais importante para avaliação é a obtenção de dados para trabalhar. O "nirvana" na detecção de anomalias é obter acesso a um conjunto de dados com tráfego de rede de um grande ambiente real, de vários pontos diferentes da rede. Trabalhar com tráfego real fortalece muito a avaliação, pois pode demonstrar que o sistema pode funcionar na prática. É importante sempre lembrar que para avaliar um sistema de detecção de anomalias é preciso vários conjuntos de dados, isso porque é preciso treinar o sistema com dados diferentes daqueles utilizados para a avaliação final. Uma abordagem padrão para a realização de treinamento e detecção em tráfegos diferentes é a técnica de subdivisão, onde subconjuntos dos dados disponíveis são selecionados através de uma amostragem aleatória.
- **Interpretação dos resultados**: Uma avaliação confiável frequentemente requer o relacionamento das entradas e saídas em baixo nível, ou seja, os pesquisadores precisam examinar manualmente os falsos positivos e negativos. Sempre que uma investigação nesse nível é realizada e não se é capaz de identificar a razão pela qual o sistema incorretamente relatou um caso particular, isso indica uma falta de visão sobre a operação do sistema de detecção de anomalias. Um ponto importante, e muitas vezes esquecido, é a inspeção dos verdadeiros positivos e verdadeiros negativos. Isto porque com a aprendizagem de máquina, muitas vezes não é aparente que o sistema aprendeu, mesmo quando produz resultados corretos. Um exemplo clássico vem de uma rede neural treinada para detectar tanques em fotos, cuja avaliação inicial era realmente capaz de fazê-lo. Contudo, descobriu-se depois que conjuntos de dados usados para treinamento e avaliação compartilhavam uma propriedade sutil: as fotos dos tanques foram tiradas em um dia nublado, enquanto todos os outros elementos não tanques tinham sido tiradas com céu azul. Assim, a rede neural simplesmente aprendeu a detectar a cor do céu.

2.3.3. Exemplos de aplicações da Aprendizagem de Máquina na Detecção de Anomalias

Para melhorar a compreensão dos leitores sobre o assunto, esta seção apresenta alguns dos principais trabalhos, ferramentas e soluções que empregam aprendizagem de máquina em problemas de detecção de anomalias.

2.3.3.1. *Cross-Site Scripting* (XSS)

Ainda pouco explorado neste contexto, o uso de aprendizagem de máquina tem surgindo como um meio eficiente de tratamento de ataques XSS. As propostas mais recentes são descritas abaixo.

[Likarish et al., 2009] apresentam um algoritmo que realiza a busca automática de *scripts* maliciosos na Internet com o apoio de um coletor *Web* (*Crawler*) denominado *Heritrix*. Nos experimentos foi utilizada uma base contendo cerca de 63 milhões de *scripts* benignos (*Alexa*[5]), sendo que destes somente 50.000 foram utilizados. O *Webcrawler* coletou ainda 62 *scripts* maliciosos para a composição da base. Logo, a partir destes dados, o vetor de características foi composto e repassado como parâmetro para diversos classificadores como SVM, DT, *Naive Bayes* e RIPPER para detectar *scripts* maliciosos e avaliar seus respectivos desempenhos, a partir de características focadas na detecção de ofuscação, uma técnica comum para contornar detectores de *malwares* tradicionais. Resultados experimentais demonstraram que SVM obteve resultados superiores aos demais algoritmos de aprendizagem, podendo alcançar até 92% de detecção.

[Riech et al., 2010] apresentam um sistema para detecção automática e prevenção de ataques XSS do tipo *drive-by-download* (*downloads* que ocorrem sem o consentimento do usuário) denominado CUJO (*Classification of Unknown JavaScript Code)*. O sistema foi incorporado em um *Proxy Web* para inspecionar páginas e partes de blocos de códigos *JavaScript* maliciosos. Recursos de códigos estáticos e dinâmicos foram extraídos e analisados a partir de padrões utilizando técnica de aprendizagem de máquina como SVM. O sistema realizou uma avaliação empírica com 200.000 páginas *Web* e 600 ataques do tipo *drive-by-download* para demonstrar a eficácia da abordagem. CUJO foi capaz detectar 94% dos ataques com uma taxa de falso positivo de 0,002%, o que corresponde a 2 alarmes falsos em100 mil sites visitados. Em termos de tempo de execução, CUJO forneceu uma média de 500 *milisegundos* por página *Web*, incluindo o *download* do conteúdo da página *Web* e a análise completa de código *JavaScript*. Na prática, CUJO foi o primeiro sistema capaz de bloquear com eficiência ataques do tipo *drive-by-downloads*.

2.3.3.2. *Phishing*

De acordo com [Sheng et al., 2009], a detecção de *phishing* é basicamente feita através da filtragem de e-mails e páginas *Web*. Com foco na detecção de e-mails *phishing*, [Fette et al., 2007] apresentam um algoritmo capaz de detectar *phishing*, chamado PILFER (do Inglês *Phishing Identification by Learning on Features of Email Received*). A idéia é extrair um conjunto de características (como a presença de *links*, presença de *JavaScript*, entre outras) para treinar e testar classificadores. PILFER usa Floresta Aleatória como classificador, mas outros classificadores como SVM, Árvore Decisão e *Naïve Bayes* também foram testados e podem ser utilizados.

Em termos de resultado, PILFER reduziu significativamente a quantidade de e-mails *phishing* comum custo mínimo em termos de falsos positivo. O método foi avaliado em dois conjuntos de dados publicamente disponíveis: *corpora ham* (projeto *SpamAssassin*), com 6.950 e-mails não-*phishing*, e *phishingcorpus*, com 860 e-mails

[5] *Alexa Top Sites*, http://www.alexa.com/topsites

phishing. Foi utilizada a estratégia de validação cruzada em 10 partições nas duas bases, sendo que a ferramenta atingiu 99,5% de precisão, com uma taxa de falso positivo de aproximadamente 0,0013 e uma taxa de falso negativo de 0,035.

Em [Abu-Nimeh et al., 2007], os autores compararam a precisão preventiva de diversos métodos de aprendizagem de máquina na detecção de e-mails *phishing*, incluindo: Regressão Logística, CART, Floresta Aleatória, *Naive Bayes,* SVM, e BART (*Bayesian Addictive Regression Trees*) [Chipman et al, 2006]. Foram analisados 2.889 e-mails, dos quais 59,8% eram legítimos com 43 características. Ficou demonstrado que, limitando-se ao uso da abordagem *bag-of-words* [Csurka et al., 2004], os classificadores estudados poderiam predizer com sucesso mais de 92% dos e-mails *phishing*. As taxas de falso positivo e falso negativo foram, respectivamente, melhores com Regressão Logística (4,89%) e Floresta Aleatória (11,12%).

Como salientado por [Chandrasekaran et al., 2006] a maioria dos ataques de *phishing* atuais empregam e-mails como principal meio de acesso às vítimas inocentes, levando-as a visitar os *sites* mascarados. Enquanto os mecanismos de defesa recente focam na detecção por validar a autenticidade do site, poucas abordagens têm sido propostas que se concentram na detecção de e-mails baseados em ataques de *phishing* com base nas propriedades estruturais inerentemente presentes nesses emails. Com uma objetivo de atacar o problema de *phishing* através de e-mails, os autores propõem um estratégia de classificação baseada na propriedade estrutural dos e-mails *phishing* (25 características), aplicando *one-class* SVM. Os experimentos foram realizados com uma base de 400 e-mails, sendo 200 do tipo *phishing* e 200 legítimos. A aplicação proposta pelos autores atingiu uma taxa de detecção de 95% de e-mails *phishing* com uma baixa taxa de falso positivo.

No âmbito de detecção de *phishing* em páginas *Web*, [Sanglerdsinlapachai e Rungsawang, 2010] propuseram o uso de um conjunto de classificadores. Os autores utilizaram voto majoritário para definir a classe atribuída à instância pelos seis (6) classificadores (*classifier ensemble*) usados no conjunto (*AdaBoost*, árvore de decisão J48, *Naive Bayes*, Redes Neurais, Floresta Aleatória e SVM) e Média Bayesiana Simples (*Simple Bayes Average* - SBA) para calcular a probabilidade média de cada classe. Nos experimentos foram testadas todas as combinações possíveis entre os classificadores para compor conjuntos de classificadores. Para o treinamento foram usadas 500 páginas *phishing* e 500 páginas *não-phishing*, e para classificação foram usadas 1.500 páginas *phishing* e 1.500 *não-phishing*. Como resultado, os autores apontam uma melhora na precisão de detecção em aproximadamente 30% em relação a métodos heurísticos, com a vantagem de serem métodos de fácil de implementação.

[Miyamoto et al., 2009] utilizou nove (9) técnicas de aprendizagem de máquina (*AdaBoost, Bagging,* SVM, CART, Regressão Logística, Floresta Aleatória, Redes Neurais, *Naive Bayes* e BART) para detecção de páginas *Web* de *phishing*. Na proposta dos autores, todas as técnicas foram usadas para classificar se um site era ou não *phishing*. Para avaliação das técnicas, foram utilizadas 3.000 amostras, sendo 50% para sites *phishing* e 50% para sites legítimos. Os autores empregaram validação cruzada com quatro (4) partições para melhor obter a média do resultado. A menor taxa de falso positivo foi 13,64%, usando *Naive Bayes*, enquanto a menor taxa de falso negativo foi 13,54%, usando Redes neurais.

A Tabela 2.4 sumariza as soluções apresentadas nesta seção para detecção de *phishing* que utilizam técnicas de aprendizagem de máquina.

Tabela 2.4. Sumarização das soluções para detecção de *phishing*

Publicações	Técnica de AM	Bases	Resultados
[Fette et al., 2007] PILFER	Floresta aleatória	*SpamAssassin* (6.950 e-mails) e *Phishingcorpus* (860 *phishing*)	Precisão de 99,5%, com FP de 0,0013% e FN de 0,035%
[Abu-Nimeh et al., 2007]	Regressão logística, CART, SVM, NN, BART e Floresta aleatória	1.718 e-mails legítimos e 1.171 *spams*	FP de 4,89% (Regressão logística) e FN de 11,12% (Floresta Aleatória)
[Chandrasekaran et al., 2006]	*one-class* SVM	400 *e-mails*, sendo 200 *phishing* e 200 legítimos	Precisão de 95%
[Sanglerdsinlapachai e Rungsawang, 2010]	*AdaBoost*, Árvore de decisão J48, *Naïve Bayes*, NN, Floresta aleatória e SVM	*CleanMX* e *PhishTank* (*phishing*) e *anti-phishing toolbar*, *Google* e *Yahoo!*, (não-*phishing*).	*F-measure (Medida Harmônica de Precisão)* de 0,8 e uma taxa de erro de 20%
[Miyamoto et al., 2009]	*AdaBoost*, *Bagging*, SVM, CART, Regressão logística, Floresta aleatória, NN, *Naive Bayes* e BART	*PhishTank*, *3Sharp*, *Alexa Web Search* e *Yahoo!*	Técnica / FP / FN *AdaBoost* 14.49% 13.83% *Bagging* 14,27% 15,36% SVM 15,02% 13,57% CART 14,58% 18,16% Regr. logística 14,12% 15,10% Flor. aleatória 14,54% 14,57% NN 14,88% 13,54% Naive Bayes 13,64% 15,74% BART 14,50% 14,39%

Legenda: FP (Falso Positivo), FN (Falso Negativo) e NN (*Neural Networks* – Redes Neurais)

2.3.3.3. Negação de Serviço

No âmbito de combater ataques de negação de serviço, pesquisadores também têm proposto o uso de técnicas de aprendizagem de máquina. Algumas aplicações de detecção de ataques de negação de serviço usando aprendizagem de máquina são apresentadas a seguir.

Em [Gavrilis e Dermatas, 2005], os autores apresentam e avaliam um sistema de detecção de negação de serviço em redes públicas baseado em Redes Neurais de Base Radial (RBF-NN) e características estatísticas. Um pequeno número de descritores estatísticos foram utilizados para descrever o comportamento de ataques DDoS, sendo que uma elevada taxa classificação foi alcançada por RBF-NN. O método proposto foi simulado e avaliado em uma rede pública, obtendo taxa de detecção superior a 98% dos ataques DDoS, utilizando apenas três características estatísticas.

[Nguyen e Choin, 2010] apresentam um método de detecção proativa de ataques DDoS, contendo duas fases sequenciais: Detecção e Prevenção. Primeiramente,

analisam a arquitetura DDoS para descrição e obtenção das fases iniciais como controle e ataque. A partir deste ponto, investigam os procedimentos de ataques DDoS para selecionar variáveis relevantes no reconhecimentos de ataques DDoS, uma vez que o comportamento anormal é inconstante e altera sempre que o ataque ocorre. Por fim, KNN é aplicado para classificar o estado das redes para cada fase do ataque DDoS. O classificador obteve taxa de precisão equivalente a 91,8%.

[Xu et al.,2007] propuseram um método de detecção de DDoS baseado em Modelos Ocultos de Markov (HMMs) [Rabiner, 1989] e aprendizado por reforço. O método utiliza um esquema de modelo colaborativo para detecção baseado no monitoramento do IP de origem. Para realizar a detecção automática de ataques DDoS, os detectores são distribuídos nos nós intermediários da rede ou perto das fontes de ataques DDoS. HMMs são utilizados para estabelecer um perfil para o tráfego normal baseados nas frequências dos novos endereços IP. O algoritmo de aprendizagem por reforço calcula as estratégias para as trocas de informações entre os múltiplos detectores distribuídos de forma que a precisão na detecção possa ser melhorada sem exigência de alta carga de informações na comunicação entre os detectores. O método mostrou que a utilização de HMMs e algoritmos colaborativos pode ser aplicado para equilibrar a precisão na detecção de ataques DDoS e a carga de comunicação. A precisão obtida pelo sistema foi 98,8%.

No trabalho de [Kumar e Selvakumar, 2011] é apresentada a proposta do algoritmo denominado RBPBoost, com o objetivo de aumentar o desempenho do classificador RBP (do Inglês *Resilient Back Propagation*). Para tanto, RBPBoost combina o conjunto de saídas de RBP e uma estratégia de minimização dos custos, para a decisão final do classificador. Um conjunto de dados disponíveis publicamente foi utilizado nos experimentos, tais como KDD Cup[6], DARPA 1999[7], DARPA 2000[8] e CONFICKER[9]. RBPBoost foi treinado e testado com DARPA,CONFICKER, e um conjunto de dados coletados através de *webcrawlers*. Como forma de avaliar o desempenho do algoritmo de classificação, foram utilizadas duas métricas: a precisão na detecção e o custo por amostra. Os resultados de simulação evidenciaram que o algoritmo RBPBoost alcançou uma precisão de 99,4%, com uma redução na quantidade de falsos alarmes, superando os demais algoritmos de Redes Neurais, comumente utilizados na literatura.

[Wu et al, 2011] propõem um sistema de detecção de ataques DDoS que se baseia em decidir o padrão de fluxo de tráfego analisando ocorrências em diversas situações de ataque. A detecção de ataques em situação normal foi considerada como um problema de classificação. Nesse contexto, os autores utilizaram 15 atributos, que não apenas monitoram a entrada/saída da taxa de pacotes/*bytes*, mas também compilam o TCP SYN e taxa de *flags* ACK, para descrever o padrão de fluxo de tráfego. Para detectar o fluxo de tráfego anormal, os atributos de testes foram aplicados utilizando a técnica de árvore de decisão C4.5. Este novo padrão de tráfego foi usado combinando procedimentos para identificar fluxo de tráfego normal, que segundo os autores é semelhante ao fluxo de ataque, para rastrear a origem de um ataque. O sistema foi capaz

6 http://www.ll.mit.edu/IST/ideval/data/1998/1998_data_index.html
7 http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.htm
8 http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/2000data.html
9 http://www.caida.org/data/passive/telescope-3days-conficker_dataset.xml

de detectar ataques DDoS com uma taxa de falso positivo variando entre 1,2% e 2,4%, e a razão falso negativo sobre 10% e 20%. O sistema também foi capaz de mensurar os fluxos de tráfegos anormais, com taxas de falso negativo variando de 8% a 12% e taxa de falsos positivos de 12% a 14%.

A Tabela 2.5 sumariza as soluções apresentadas nesta seção para detecção de ataques de negação de serviço que utilizam técnicas de aprendizagem de máquina.

Tabela 2.5. Sumarização das soluções para detecção de ataques de negação de serviço

Publicação	Técnica de AM	Base	Resultados (%)
[Gavrilis e Dermatas, 2005]	RBF-NN	Tempo Real	Precisão de 98%
[Nguyen e Choin, 2010]	*k*-NN	[Darpa 2000]	Precisão de 91,8%
[Xu et al.,2007]	HMMs	[TFN2K[10]]	Precisão de 98,8%
[kumar e Selvakumar, 2011]	NN	[Darpa 1999]	Precisão de 99,4%
[Wu et al., 2011]	DT	[Benzel et al., 2006]	Precisão de 98%

2.3.3.4 Códigos Maliciosos (*Malware*)

Considerando-se a ameaça de *malware* e sua prevalência, não é surpreendente que uma quantidade significativa de pesquisadores tenha concentrado esforços no desenvolvimento de técnicas para coletar, estudar e mitigar código malicioso. Por exemplo, há estudos que medem o tamanho de *botnets*, a prevalência de *sites* mal-intencionados e a infestação de executáveis com *spyware*. Além disso, existem ferramentas que podem executar amostras desconhecidas e monitorar o seu comportamento. Algumas ferramentas fornecem relatórios que resumem as atividades de programas desconhecidos em nível da API (*Application Programming Interface*) do *Windows* ou chamadas de sistema. Esses relatórios podem ser avaliados para encontrar conjuntos de amostras que se comportam da mesma forma, ou para classificar o tipo de atividade maliciosa observada. Outras ferramentas incorporam fluxo de dados em análise, o que resulta numa visão mais abrangente da atividade de um programa na forma de grafos.

Abordagens baseadas em aprendizagem de máquina para detecção de códigos maliciosos utilizam algoritmos para treinamento dos dados no intuito de detectar *malwares*. No entanto, uma vez que classificadores baseados em aprendizagem supervisionada requerem um elevado número de dados rotulados para cada uma das classes (isto é, *malware* e benigno), a indústria anti-*malware* vem auxiliando e investindo pesado em pesquisas nesta área. Alguns exemplos de aplicações para detecção de *malwares* utilizando aprendizado de máquinas são apresentados a seguir.

[Schultz et al., 2001] foram os primeiros a introduzir o conceito de aplicação de algoritmos de aprendizado automático para a detecção de *malware*, com base em seus respectivos códigos binários, aplicando diferentes classificadores como RIPPER, *Naive Bayes* e a combinação de classificadores e considerando três tipos de conjuntos de características: (i) cabeçalhos de programa, (ii) cadeias e (iii) sequencias de *bytes*. Posteriormente, [Kolter e Maloof, 2004] melhoraram os resultados de Schultz, aplicando *n-grams* (ou seja, sobrepondo sequências de *bytes*). Os resultados

[10] http://www.cert.org/incident_notes/IN-99-07.html

experimentais obtidos com Árvore de Decisão foram superiores aos demais algoritmos de aprendizagem de máquinas investigados nos experimentos.

[Rieck et al., 2008] apresentaram a proposta de uma abordagem de classificação automática de comportamento *malware*, tendo como base os seguintes itens: *i)* a incorporação de rótulos atribuídos pelo software antivírus para definir classes para a construção de modelos supervisionados; *(ii)* o uso de recursos específicos, como cadeia de caracteres, que descrevem padrões de comportamento de *malware*; *iii)* a construção automática de modelos discriminativos utilizando algoritmos de aprendizagem como SVM e; *iv)* identificação das características dos modelos explicativos que mais influenciaram o aprendizado, produzindo um *ranking* de padrões de comportamento de acordo com seus pesos. Na prática o método funciona da seguinte forma: coleta-se um número de amostras de *malware* e analisa-se seu comportamento usando um ambiente *Sandbox*, identificam-se *malware* típicos a serem classificados, executando um software antivírus padrão, e constrói-se um classificador baseado no comportamento de *malware* aprendendo as classes simples dos modelos. A precisão obtida pelo método foi equivalente a 70%.

[Wang et al., 2003] propõem um método de detecção de vírus até então desconhecidos usando técnicas de aprendizagem de máquina como DT e *Naive Bayes*. O método utiliza uma base de dados contendo 3.265 códigos maliciosos e 1.001 benignos. Todo o processo de extração de características foi realizado automaticamente, a partir de uma abordagem dinâmica. Por fim, os vetores foram utilizados para treinar os classificadores e detectar os possíveis vírus. Resultados experimentais mostraram que a precisão do modelo utilizando DT foi superior a *Naive Bayes*, obtendo precisão de 91,4% e 77,1%, respectivamente.

[Kolter e Mallof, 2006] desenvolveram MECS (*Malicious Executable Classification System*) para detectar automaticamente executáveis maliciosos, sem pré-processamento ou remoção de qualquer ofuscação. O sistema utiliza um conjunto de 1.971 executáveis benignos e 1.651 executáveis maliciosos, com uma variedade de mecanismos de transporte e carga úteis (por exemplo, *key-loggers* e *backdoors*). Embora todos os códigos sejam executados no *MSWindows*, vale ressaltar que o métodos não se restringe unicamente a este sistema operacional. As sequências de *bytes* dos executáveis foram convertidas em *n-grams* e, então, indexadas no vetor de características, que por sua vez foi repassado como parâmetro de entrada para treinamento e teste de diversos classificadores como: KNN, *Naive Bayes*, SVM e DT. Neste domínio, os autores atentaram para o problema de falsos alarmes, o que ocasiona custos desiguais de classificação. Desta forma, os métodos foram avaliados usando análise ROC (*Receiver Operating Characteristic*) como métrica de desempenho. DT superou os diversos outros métodos, com área ROC de 0,996.

Por fim, [Ye et al., 2009] apresentam o IMDS (*Intelligent Malware Detection System*), ou seja, um sistema inteligente para detecção de *malwares*, mais precisamente, vírus polimórficos. O IMDS é aplicado sobre o OOA (*Objective-Oriented Association*) baseado em técnicas de mineração de dados. O sistema realiza análise de sequências de execução de diversas APIs do *Windows*, que refletem o comportamento dos códigos maliciosos. A captura da semântica da associação de APIs é essencial para a detecção de *malware*. Em seguida, a detecção de *malware* é realizada diretamente no PE (*Windows Portable Executable*) em três etapas: *i)* Construção das sequencias de

execução de API através do desenvolvimento de um analisador PE; *ii)* Extração de regras usando OAA e técnicas de mineração de dados e; *iii)* Classificação baseada nas regras de associação geradas na segunda etapa. O sistema utiliza uma base de dados contendo 29.580 executáveis, sendo 12.214 códigos benignos e 17.366 maliciosos. O sistema IMDS foi comparado com diversos antivírus populares como *Norton AntiVirus* e *McAfee VirusScan*[11], sendo que os resultados experimentais foram superiores a eficiência dos demais antivírus. Para isso, foram utilizadas técnicas de aprendizagem de máquina como *Naive Bayes*, SVM e DT. Este é um exemplo clássico que mostra o resultado de aplicações de técnicas de aprendizagem de máquina em segurança de redes, pois a abordagem introduzida pelo IMDS resultou na incorporação do sistema à ferramenta de verificação do *Kingsoft Antivirus*.

A Tabela 2.6 sumariza as soluções apresentadas nesta seção para detecção de *malwares* que utilizam técnicas de aprendizagem de máquina.

Tabela 2.6: Sumarização das soluções para detecção de *malwares*.

Publicação	Técnica de AM	Base	Resultados (%)
[Schultz et al., 2001]	RIPPER, *Naive Bayes*	*MacAfee*	Precisão de 71,05% (RIPPER), 97,76% (*Naive Bayes*)
[Rieck et al., 2008]	SVM	8.082 amostras, sendo que 3.139 *malwares*	Precisão de 70%
[Ye et al., 2007]	SVM, *Naive Bayes*, DT	29.580 amostras, sendo 17.366 maliciosos	Precisão de 90,54% (SVM), 83,86% (*Naive Bayes*) e 91,49% (DT)
[Wang et al., 2003]	DT, *Naive Bayes*	3.265 executáveis maliciosos e 1.001 benignos	Precisão de 91,4% (DT) e 77,1% (*Naive Bayes*)
[Kolter e Mallof, 2006]	*k*-NN, *Naive Bayes*, SVM, DT	1.971 benignos e 1.651 malignos	Precisão de 98,99% (*k*-NN), 98,87% (*Naïve Bayes*), 99,03% (SVM) e 99,58% (DT)

2.3.3.5. *Spam*

Em relação às soluções para detecção de *spam* baseadas em aprendizagem de máquina, o foco principal inclui a proteção para páginas *Web* através da coleta de e-mails, filtros de lista negra (*blacklist*), filtros de listas brancas (*whitelist*), filtros de listas cinza (*greylist*), análise de *spam* baseado em texto e combate a *spam* baseado em imagem.

A solução proposta por [Debarr e Wechsler, 2009] usa uma classificação híbrida com foco na construção eficiente de modelos para detecção de *spam*. A idéia é fazer o agrupamento de mensagens, permitindo assim, a rotulagem eficiente de um exemplo representativo de mensagens para um aprendizado de modelo de detecção usando Florestas Aleatórias para classificação e aprendizado ativo (seleção efetuada por um algoritmo do computador) para refinar o modelo de classificação. Os autores usam *Area Under the receirver operating characterisitc Curve* (AUC) [Bradley, 1997], como forma de medir o desempenho dos classificadores. Com essa medida Florestas Aleatórias atingiram um desempenho de 95%.

[11] *MacAfee*, http://www.mcafee.com.

O trabalho de [Blanzieri e Bryl, 2007] propõe a avaliação de desempenho de SVM vizinho mais próximo. Em linhas gerais, o classificador proposto funciona da seguinte maneira: a fim de classificar uma amostra *x*, o SVM-NN seleciona as *k* amostras treinadas mais próximas da amostra *x* e então, usa essas *k* amostras para treinar um modelo SVM o qual é posteriormente usado para tomar a decisão. Para avaliar o classificador foram utilizadas 4.150 mensagens legítimas e 1.897 mensagens *spam* do *SpamAssassin Corpus*[12]como conjunto de dados para os experimentos, conseguindo atingir uma taxa de verdadeiro positivo de 99%.

A aplicação de [Castillo et al., 2007] está fundamentada em um sistema de detecção de *spam* usando Árvore de Decisão, *Bagging* e agrupamento. Os autores trabalham em três níveis para incorporar a topologia gráfica da *Web*: *1)* agrupando o gráfico host; *2)* rotulando todos os hosts no grupo por voto majoritário; e *3)* propagando os rótulos esperados para os hosts vizinhos. Os rótulos esperados dos hosts vizinhos são usados como novas características e o classificador é re-treinado. O conjunto de dados usado para os experimentos foi o *WEBSPAM-UK2006*[13], uma coleção de *spam Web* disponível publicamente. Os resultados obtidos foram de 78,7% de verdadeiro positivo e 5,7% de falso negativo com *Bagging*.

[Markines et al., 2009], focando a detecção de *spam* social, utilizaram a ferramenta Weka[14] para conduzir experimentos usando trinta (30) algoritmos de aprendizagem de máquina. O conjunto de dados utilizado para demonstrar os experimentos é uma coleção privada, disponível como parte do ECML/PKDD 2008[15] (*Discovery Challenge on Spam Detection*), com um total de 27.000 usuários, dos quais 25.000 são usuários *spammers* e 2.000 são usuários legítimos. Os resultados obtidos foram de 98% de precisão para detecção de spam social e 2% de falso positivo, para o classificador de Regressão Logística. Os demais classificadores também tiveram um desempenho com precisão em torno de 96% e falsos positivos menores que 5%. Os autores se utilizaram da estratégia de validação cruzada com 10 partes para conduzir o treinamento e teste.

[Wu, 2009] apresenta um método híbrido baseado em regras de processamento e redes neurais *back-propagation* para a filtragem de spam. Em vez de usar palavras chaves, usa o comportamento *spamming* como característica para descrever e-mails. Um processo baseado em regras é usado primeiro para identificar e digitalizar o comportamento *spamming* observado de cabeçalhos e *logs* de e-mails. A coleção de dados foi obtida através do MTA (do Inglês *Mail Transfer Agent*) com um total de 120.207 amostras das quais 50.651 são e-mails legítimos e 69.556 é *spam*. Os autores atingiram uma precisão de 99,6% com seu método.

A Tabela 2.7 sumariza as soluções apresentadas nesta seção para detecção de *spam* que utilizam técnicas de aprendizagem de máquina.

12 http://wiki.apache.org/spamassassin/CorpusCleaning
13 http://barcelona.research.yahoo.net/webspam/datasets/uk2006/
14 http://www.cs.waikato.ac.nz/ml/weka/
15 http://www.kde.cs.uni-kassel.de/ws/rsdc08

Tabela 2.7. Sumarização das soluções para detecção de spam

Publicações	Técnicas de AM	Base	Resultados (%)
[Blanzieri e Bryl, 2007]	Combinação de SVM e k-NN	*SpamAssassin Corpus* - 4.150 mensagens legitimas e 1.897 *spam*	99% de TP
[DeBarr e Wechsler, 2009]	Agrupamento, Floresta Aleatória	TREC *Spam EmailCorpus*[16]com 75.419 amostras	95,2% AUC
[Markineset al., 2009]	30 algoritmos de aprendizagem	ECML/PKDD 2008 com um total de 27.000 usuários	98% precisão e 2% FP com Regressão Logística
[Castillo et. al., 2007]	DT, *Bagging* e agrupamento	WEBSPAM-UK2006	78,7% de TP e 5,7% de FN com *Bagging*
[Wu, 2009]	MLP	Coleção privada usando MTA, 120.207 e-mails	99,60 de precisão

Legenda: FP (Falso Positivo) , FN (Falso Negativo) e TP (Verdadeiro Positivo)

2.3.3.6. Detecção de Intrusão

Na literatura, a detecção de intrusão tem sido tratada através da proposta de diversos sistemas que utilizam técnicas de aprendizagem de máquina, tais como: redes neurais artificiais [Souza e Monteiro, 2009] [Xia et al., 2010], *k*-means [Tian e Jianwen, 2009], *k*-NN [Tsai e Lin, 2010] e SVM [Xiao et al., 2007], entre outras. Tais técnicas têm sido usadas como: classificadores individuais, classificadores híbridos e conjunto de classificadores.

Um exemplo de uma aplicação recente encontra-se em [Horng et al., 2011]. Os autores propuseram um sistema de detecção de intrusão baseado em agrupamento hierárquico, seleção de características e SVM. O método foi avaliado na base KDD Cup 1999, composta por 41 características e 4.898.431 amostras para treino e 311.029 amostras para teste. Os resultados atingidos foram de 95% de precisão e 0,7% de falso positivo.

Semelhante a aplicação anterior, [Khan et al., 2007] usaram análise de agrupamento para aproximar vetores de suporte a fim de acelerar o processo de treino de SVM. Desta forma, os autores propuseram o agrupamento de árvores baseado em SVM para melhorar a precisão do classificador. Os experimentos foram conduzidos sobre a base de dados DARPA 1998 com 41 características e cinco classes (Normal, DoS, *Probe*, U2R[17], R2L[18]). A avaliação desses experimentos sobre as cinco classes obteve os seguintes resultados em relação à precisão: 95% (normal), 97% (DoS), 23% (U2R), 43% (R2L) e 91% (*Probe*).

A solução de [Tsai e Lin, 2010] é conhecida como TANN (*Triangle Area based Nearest Neighbor*). Esse método transforma o espaço de características original em um novo espaço de características. Inicialmente, *k-means* é utilizado para agrupar as amostras da base de dados. Com base nos grupos encontrados, são calculadas áreas de triângulos formados entre todas as amostras e os centros de cada grupo. Na segunda

16 http://trec.nist.gov/data/spam.html
17 U2R – User to Root
18 R2L – Remote to Local

fase, um novo vetor de características formado pelas áreas triangulares é usado como entrada para o classificador *k*-NN que definirá a classe de cada amostra. A base de dados investigada foi a KDD Cup 1999, atingindo uma precisão de 99,01%, uma taxa de detecção de 99,27% e uma taxa de falso alarme de 2,99%.

Fazendo uso de Redes Neurais de Kohonen e SVM, [Mafra et al. 2008] propuseram um sistema multicamadas, chamado POLVO–IIDS (*Intelligent Intrusion Detection System*). A primeira camada analisa o tráfego da rede e o classifica em quatro categorias: *DoS*, *Worm*, *Scan*, *R2L*. A segunda camada é responsável pela detecção de intrusão propriamente dita. Os experimentos foram realizados com a base de dados KDD Cup 99, sendo que os seguintes resultados foram obtidos: taxa de acerto de 96,55% e um desvio máximo de 9,53%.

[Xiao et al., 2007] propuseram uma aplicação que aborda a técnica de conjunto de classificadores, utilizando três (3) SVMs. O primeiro classificador é treinado com dados originais, sem alterações, como normalmente é feito com classificadores individuais. O segundo classificador é treinado com dados originais submetidos a um processo de seleção de características. Por fim, o último SVM é treinado apenas com uma parte dos dados de entrada, isto é, é feita uma seleção de amostras. A combinação da decisão do conjunto é obtida através de uma função de voto ponderado. A base de dados utilizada nos experimentos foi à base KDD Cup 99. Os autores alcançaram os seguintes resultados: 99% para taxa de detecção e 0,0018% para falso positivo.

A Tabela 2.8 sumariza as soluções apresentadas nesta seção para detecção de intrusão que utilizam técnicas de aprendizagem de máquina.

Tabela 2.8. Sumarização das soluções para detecção de intrusão

Publicações	Técnicas de AM	Base	Resultados (%)
[Horng et al., 2011]	Agrupamento e SVM	KDD Cup 1999	95,71% de Precisão e 0,7% FP
[Khan et al., 2007]	Agrupamento de Árvores e SVM	DARPA 1998	Precisão para cada classe: 95% (normal), 97% (DoS), 23% (U2R), 43% (R2L) e 91% (Probe)
[Mafra et al, 2008]	Redes Neurais de *Kohonen* e SVM	KDD Cup 1999	Taxa de acerto 96,55% e desvio máximo de 9,54%
[Tsai e Lin, 2010]	*k-means* e *k*-NN	KDD Cup 1999	Precisão de 99,01%, taxa de detecção 99,27% e falso alarme de 2,99%
[Xiao et al., 2007]	Conjunto de Classificadores (SVM)	KDD Cup 1999	Taxa de Detecção 99% e falso positivo 0,0018%

2.3.4. Síntese

Supondo que neste ponto, o leitor se sinta confortável e familiarizado com as diferentes técnicas de aprendizagem de máquina para detecção de anomalias, os autores deste minicurso têm algumas considerações a fazer em relação às diversas aplicações apresentadas na seção anterior. Essas considerações não têm como objetivo abordar ou esgotar todos os aspectos positivos, negativos e de complexidade computacional relacionados à aprendizagem de máquina para detecção de anomalias.

Um aspecto que se pode observar nas varias aplicações apresentadas anteriormente é a falta de dados rotulados para treino/validação, o que também é apontado por [Chandola et al., 2009] como uma desvantagem em detecção de intrusão utilizando técnicas de aprendizagem de máquina supervisionada.

Outro ponto a considerar são as várias técnicas de aprendizagem de máquina aplicadas nas soluções exibidas confirmando certa tendência de classificadores específicos para problemas peculiares, conforme mostra a Tabela 2.9. Por exemplo, o classificador SVM, que é considerado na literatura como um classificador estável e com bom desempenho individual [Tsai et al., 2009] para detecção de anomalias, continua se mostrando da mesma forma em combinações com outros classificadores seja em classificadores híbridos ou em conjuntos de classificadores.

A técnica de classificação baseada em Árvores de Decisão também tem sido utilizada em muitos problemas relacionados à detecção de anomalia. Observa-se também que as técnicas de aprendizagem supervisionada são utilizadas como muito mais frequência quando comparadas aos métodos não supervisionados. Por fim, a combinação de classificadores em conjuntos e em estratégias híbridas parece estar tornando-se uma estratégia padrão na área de segurança de redes, assim como ocorre em diversas outras áreas de aplicação.

Com um enfoque diferenciado das tabelas apresentadas anteriormente que sumarizam aplicações para determinadas ameaças, a Tabela 2.9 sintetiza as aplicações para detecção de anomalia correlacionada com as técnicas de aprendizagem de máquinas. Essa síntese proporciona visualizar quais técnicas de aprendizagem de máquina tem uma melhor eficácia e eficiência em problemas de detecção de anomalia em redes de computadores para determinadas abordagens de problemas (*spam*, *phishing*, XSS, *malware, DDoS*).

2.4. Comentários Finais e Direções Futuras

Este capítulo procurou introduzir o leitor no universo da aprendizagem de máquina aplicada à prevenção e detecção de anomalias. Primeiramente, a Seção 2.1 introduziu o problema envolvendo o tráfego malicioso, onde foram descritas algumas das principais ameaças, bem como algumas das soluções existentes. Uma contextualização sobre o uso de aprendizagem de máquina na detecção de anomalias também foi discutida.

Em seguida, a Seção 2.2 forneceu uma visão geral da área de aprendizagem de máquina. Foram apresentados conceitos básicos essenciais e a terminologia utilizada, além da categorização dos métodos de aprendizagem. Por fim, foram descritas as principais técnicas de aprendizagem de máquina. Com as informações apresentadas na seção 2.2, os autores esperam que os leitores possam ser capazes de entender de maneira geral cada técnica ou sintam-se aptos a consultar alguns dos livros e artigos citados. Para os interessados, o livro de Witten e Frank [Witten e Frank, 2000] é um excelente próximo passo, principalmente porque os leitores podem baixar o Weka, um software livre, de código aberto, com implementações em Java de um grande número de algoritmos para construção de modelos, seleção de atributos e outras operações. Experimentar é uma ótima maneira de compreender melhor os aspectos práticos da aprendizagem de máquina. Existe ainda um livro complementar ao primeiro, Mena [Mena, 2003] que trata de várias abordagens de mineração de dados aplicadas a segurança de computadores, onde técnicas de aprendizagem de máquina e mineração de

dados são aplicadas em várias áreas do conhecimento, incluindo a detecção de intrusão, detecção de fraude e de perfis criminais.

A Seção 2.3 apresentou o emprego da aprendizagem de máquina no contexto da segurança de redes de computadores. Primeiramente foram discutidos os desafios do uso e implementação, onde se pode observar claramente que, por se tratar de um domínio específico e cheio de nuances, diversas questões, que não interferem em outras áreas do conhecimento, são problemáticas para detecção de anomalias, incluindo: *i)* a necessidade de detecção de *outlier*, embora a aprendizagem de máquina apresente melhor desempenho na atividade de encontrar semelhanças; *ii)* os elevados custos dos erros de classificação, que fazem com que, quando comparados a outros domínios, as taxas de erro cheguem a ser irrealistas; *iii)* uma lacuna semântica entre os resultados de detecção e sua interpretação operacional; *iv)* a enorme variabilidade do tráfego benigno, o que torna difícil encontrar noções estáveis de normalidade; e *v)* desafios significativos com a realização de avaliação e testes das soluções.

Em seguida, de forma a superar estes desafios, foram fornecidas algumas diretrizes para a aplicação de aprendizagem de máquina para detecção de anomalias de rede, incluindo: *i)* a importância de se obter uma visão completa, em termos de capacidades, limitações e operação, do sistema de detecção de anomalias a ser desenvolvido; e *ii)* o reconhecimento do papel da avaliação para comprovação da efetividade e eficiência do sistema de detecção.

2.4.1. Oportunidades de Pesquisa

É evidente a partir das discussões apresentadas neste minicurso que a aprendizagem de máquina é naturalmente uma valiosa ferramenta para a detecção e prevenção de atividades maliciosas. Algumas sugestões que poderiam ser passíveis de abordagens de aprendizagem de máquina são destacadas a seguir:

- Embora a maioria das pessoas que utiliza redes sociais não represente uma ameaça, pessoas mal-intencionadas estão sendo atraídas para estas redes por causa da acessibilidade e da quantidade de informação pessoal disponível. Tais informações podem ser usadas para conduzir ataques de engenharia social [Lam e Yeung, 2007] e distribuição de códigos maliciosos. O emprego de técnicas de aprendizagem de máquina nesta área pode ser bastante útil para a definição de perfis de usuários que não tenham conhecimento de boas práticas de segurança;
- Agentes automatizados tais como, *worms*, vírus ou *trojans*, podem ser detectados com base em padrões de tráfego de rede de saída. As técnicas de aprendizagem de máquina são conhecidas por apresentarem bons desempenhos no reconhecimento de padrões;

Tabela 2.9. Síntese das técnicas de AM empregadas em detecção de anomalias.

Aplicação	Publicação	Técnica de AM														
		SVM	***k*-means**	**Redes Neurais**	***k*-NN**	**Árvores de Decisão**	***Naive Bayes***	**RIPPER**	**Floresta Aleatória**	**HMM**	***AdaBoost***	**Regressão Logística**	***Bagging***	**Conj. de Classif.**	**BART**	***Clustering***
Cross Site Scripting	[Likarish et al., 2009]	X				X	X	X								
	[Rieck et al., 2010]	X														
Phishing	[Fette et al., 2007]								X							
	[Abu-Nimeh et al., 2007]	X		X		X			X			X			X	
	[Chandrasekaran et al., 2006]	X														
	[Sanglerdsinlapachai., 2010]	X		X		X	X		X		X					
	[Miyamoto et al., 2009]	X		X		X	X		X		X				X	
Negação de Serviço	[Gavrilis, 2005]			X												
	[Nguyen e Choin, 2010]									X						
	[Xu et al.,2007]					X										
	[kumar, 2011]			X												
	[Wu et al., 2011]					X										
	[Schultz et al., 2001]						X	X								
Códigos Maliciosos *(Malware)*	[Rieck et al., 2008]	X					X									
	[Ye et al, 2007]	X				X	X									
	[Wang et al, 2003]					X	X									
	[Kolter e Mallof, 2006]	X			X	X	X									
	[Blanzieri e Bryl, 2007]	X			X									X		
Spam	[DeBarr, 2009]								X							
	[Markines et al., 2009]													X		
	[Castillo et. al., 2007]					X							X			X
	[Wu, 2009]			X												
	[Horng et al., 2011]	X														X
Detecção de Intrusão	[Khan et al., 2007]	X				X										X
	[Mafra et al, 2008]	X		X												
	[Tsai e Lin, 2010]		X		X											
	[Xiao et al., 2007]	X												X		

- Pode ser difícil classificar falhas na rede como intencional ou não. Por exemplo, um aumento repentino no consumo de largura de banda da rede pode indicar um ataque de negação de serviços, ou simplesmente um evento onde muitos usuários simultaneamente acessam um servidor devido a algum evento popular (por exemplo, um *flash crowd*). Pode ser possível diferenciá-los através do emprego de classificadores automatizados;
- Outro desafio mostrado por técnicas de detecção de anomalia em um domínio de grande tráfego de dados como a Internet é a natureza das anomalias que muda ao longo do tempo, como forma de iludir as soluções de detecção de intrusão, conforme [Chandola et al., 2009]. Nessa situação, os dados têm um aspecto temporal associado a ele e relevante para aplicação de detecção de padrão com aprendizagem de máquina.
- É importante salientar ao leitor, neste ponto do minicurso, que a inclusão de aprendizagem de máquina em um sistema deve ser feito com cuidado para evitar que o componente de aprendizagem em si ceda ao ataque. A atual literatura mostra que os atacantes podem atacar com sucesso máquina de aprendizagem dos sistemas, tanto no geral [Barreno et al., 2006] como em domínios de aplicação específicos como geração de uma assinatura automática [Chung e Mok, 2006] [Chung e Mok, 2007], sistemas de detecção de intrusão [Fogla e Lee, 2006] e e-mail com filtros *spam*. É necessario assegurar que a aprendizagem seja bem sucedida, apesar de tais ataques, em outras palavras, atingir um aprendizado seguro [Barreno et al., 2008].

A maioria dos trabalhos de aprendizagem de máquina na área de segurança tem se concentrado no processo de desenvolvimento de técnicas para detecção de ataques. Esta é apenas uma pequena parte do escopo que envolve aspectos de segurança. As idéias acima salientam alguns problemas de segurança que parecem demandar mais investigação científica. Há certamente, porém, muitos outros aspectos em que as abordagens de aprendizagem de máquina podem trazer novas melhorias na segurança de computadores.

Referências

[Abu-Nimeh et al., 2007] Abu-Nimeh, S., Nappa, D., Wang, X., Nair, S. (2007) "A Comparison of Machine Learning Techniques for Phishing detection", Em *Proceedings of the Anti-phishing Working Groups 2nd annual eCrime Researchers Summit* (*eCrime '07*), pp. 60-69.

[Abu-Nimeh et al., 2009] Abu-Nimeh S., Nappa Dario, Wang X., Nair S., (2009) "Distributed Phishing Detection by Applying Variable Selection using Bayesian Additive Regression Trees", Em *IEEE International Conference on Communications (ICC 2009)*. pp. 1-5.

[Alpaydim, 2010] Alpaydim, E. (2010) "Introduction to Machine Learning", *The MIT Press*, Cambridge, Massachusetts, EUA, 537 páginas.

[Anderson, 1995] Anderson, J. (1995). "An Introduction to Neural Networks", Cambridge: *MIT Press*, 650 páginas.

[Anagnostakis et al., 2005] Anagnostakis, K. G., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E., e Keromytis, A. D. (2005). "Detecting Targeted Attacks Using Shadow Honeypots", Em *Proceedins of 14thUSENIX Security Symposium*, pp. 9-9.

[APWG, 2010] Anti-Phishing Working Group. (2010) "Phishing Activity Trends ReportQ1 2010", Disponível em: http://www.antiphishing.org/reports/apwg_report_Q1_2010.pdf.

[Barnett e Lewis, 1978] Barnett, V. e Lewis, T. (1978) "Outliers in Statistical Data*".Wiley Series in Probability & Statistics*, *John Wiley and Sons*, 584 páginas.

[Barreno et al., 2006] Barreno, M., Nelson, B., Sears, R., Joseph, A. D., Tygar, J. D. (2006). "Can machine learning be secure?" Em *Proceedings of the ACM Symposium on InformAtion, Computer, and Communications Security (ASIACCS'06).*

[Barreno et al., 2008] Barreno, M., Nelson, B., Bartlett, P. L., Chi, F. J., Rubinstein, B. I. P., Saini, U., Joseph, A. D., Tygar, J.D. (2008). "Open Problems in the Security of Leaning", Em *Proceedings of the 1st ACM Workshop on AISec - AISec '08*

[Bennett et al., 2007] Bennett, J., Lanning, S. e Netflix, N. (2007) "The Netflix Prize". Em *Proceeding of KDD Cup and Workshop*.

[Benzel et al., 2006] Benzel, T., Braden, R., Kim, D., Neuman, C., Joseph, A. and Sklower, K. (2006) "Experience with DETER: a Testbed for Security Research", Em *Proceedings of the 2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM'06),* pp. 379-388.

[Blanzieri e Bryl, 2007] Blanzieri, E., Bryl, A. (2007). "Evaluation of the Highest Probability SVM Nearest Neighbor Classifier with Variable Relative Error Cost", Em *Proceedings of Fourth Conference on email and Anti-Spam (CEAS'2007).*

[Bradley, 1997] Brandley P. A., (1997) "The Use of the Under the ROC Curve in the Evaluation of Machine Learning Algorithms". *Pattern Recognition*, Vol. 30 (7), pp. 1145-1159

[Breiman et al., 1984] Breiman, L., Friedman, J. H., Olshen, R. A., Stone, C. J. (1984). "Classification and Regression Trees", Monterey, *CA, Wadsworth & Brooks*.

[Castillo et al., 2007] Castillo, C., Donato, D., Gionis, A., Murdock, V., Silvestri, F. (2007), "Know Your Neighbors: Web Spam Detection Using the Web Topology". Em *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* pp. 423-430.

[Ceron et al., 2009] Ceron, J. M., Granville, L., Tarouco, L. "Taxonomia de Malwares: Uma Avaliac¸ao dos Malwares Automaticamente Propagados na Rede" *Anais do IX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais.*

[CERT.br, 2011] CERT.br – Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil. (2011) *Estatísticas do CERT.br*. http://www.cert.br/stats/incidentes/

[Chandola et al., 2009] Chandola, V., Banerjee, A., Kumar, V. (2009), "Anomaly Detection : A Survey", *ACM Computing Surveys*, pp. 1-72.

[Chandrasekaran et al., 2006] Chandrasekaran, M., Narayanan, K., e Upadhyaya, S. (2006), "Phishing Email Detection Based on Structural Properties", Em *NYS Cyber Security Conference*.

[Chipman et al, 2006] Chipman, H. A., George, E. I., McCulloch, R. E. (2006) "BART: Bayesian Additive Regression Trees", *Journal of the Royal Statistical Society*, Vol. 4 (1), pp. 266-298.

[Chung e Mok, 2006] Chung, S., Mok, A. K. (2006), "Allergy Attack Against Automatic Signature Generation", Em *Recent Advances in Intrusion Detection*, pp. 61–80.

[Chung e Mok, 2007] Chung, S., Mok, A. K. (2007). "Advanced Allergy Attacks: Does a Corpus Really Help?", Em *Recent Advances in Intrusion Detection*, pp. 236–255.

[Csurka et al., 2004] Csurka, G., Dance, C., Fan, L., Williamowski, J., Bray, C. (2004) "Visual Categorization with Bags of Keypoints," Em *ECCV04 Workshop on Statistical Learning in Computer Vision*, pp. 59–74

[Debarr e Wechsler, 2009] Debarr, D., Wechsler, H. (2009) "Spam Detection Using Clustering, Random Forests, and Active Learning", Em *Sixth Conference on Email and Anti- Spam (CEAS 2009)*.

[Denning, 1987] Denning, D. E. (1987) "An Intrusion-Detection Model".*IEEE Transactions on Software Engineering*, Vol. 13 (2), pp. 222–232.

[Duda et al. 2001] Duda, R. O., Hart, P. E. e Stork, D. G. (2001) *Pattern Classification*.2ª. Edição. *Wiley Interscience*, 680 páginas.

[Feitosa et al., 2008] Feitosa, E. L., Souto, E. J. P., Sadok, D. (2008) "Tráfego Internet não Desejado: Conceitos, Caracterização e Soluções*",Livro-Texto dos Minicursos do VIII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, páginas 91-137.

[Fette et al, 2007] Fette, N., Sadeh, Tomasic, A. (2007) "Learning to Detect Phishing E-mails", Em *Proceedings of the 16th International Conference on World Wide Web (WWW 07)*, páginas 649–656.

[Fogla e Lee, 2006] Fogla, P. e Lee, W. (2006) "Evading Network Anomaly Detection Systems: Formal Reasoning and Practical Techniques", Em *Proceeding of ACM Conference on Computer and Communications Security*.

[Fraser, 1998] Fraser, N. (1998) "*Neural Network Follies"*. Disponível em: http://neil.fraser.name/writing/tank.

[Gavrilis e Dematas, 2004] Gavrilis, D., Dermatas, E. (2004) "Real-time Detection of Distributed Denial-of-Service Attacks Using RBF Networks and Statistical Features", *Computer Networks.* Vol. 48, (2), páginas 235-245

[Gates e Taylor, 2007] Gates, C., Taylor, C. (2007) "Challenging the Anomaly Detection Paradigm: A Provocative Discussion," Em *Proc. Workshop on New Security Paradigms.*

[Grossman et al., 2007] Grossman, J., Hansen, R., Petkov, D.P., Rager, A., Fogie, S. (2007) "Cross Site Scripting Attacks: XSS Exploits and Defense". Burlington, MA, EUA, *Syngress Publishing Inc*, 482 páginas.

[Gu et al., 2007] Gu, G., Porras, P., Yegneswaran, V., Fong, M. e Lee, W. (2007) "Bot Hunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation", Em *Proceedings of 16thUSENIX Security Symposium,* 12 páginas.

[Haykin, 2008] Haykin, S. (2008) "Neural Networks and Learning Machines" 3rd Edition. *Prentice Hall.*

[Horng et al., 2011] Horng, S., Su, M., Chen, Y., Kao, T, Chen, R., Lai, J., Perkasa, C. (2011) "A Novel Intrusion Detection System Based on Hierarchical Clustering and

Support Vector Machines", *International Journal on Expert System with Applications*, Vol. 38 (1), pp. 306-313.

[Jain et al., 2000] Jain, A. K., Duin, R. P. W., Jianchang, M. (2000), "Statistical Pattern Recognition: A Review", *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 22 (1), pp. 4-37.

[Khan et al., 2007] Khan, L., Awad, M., Thuraisingham, B. (2007), "A New Intrusion Detection System Using Support Vector Machines and Hierarchical Clustering", Em *International Journal on Very Large Data Bases*. Vol. 16(4), pp. 507-521.

[Kolter e Maloof, 2004] Kolter, J., Maloof, M. (2004)."Learning to Detect Malicious Executables in the Wild". Em *Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD)*, pp. 470–478.

[Kolter e Maloof, 2006] Kolter, J. Z., Maloof, M. (2006), "Learning to Detect and Classify Malicious Executables in the Wild" Em *Journal of the Machine Learning*. Vol. 7, pp. 2721-2744.

[Kumar e Selvakumar, 2011] Kumar, P. A. R., Selvakumar, S. (2011). "Distributed Denial of Service Attack Detection Using an Ensemble of Neural Classifier", *Computer Communication*, Vol. 34, pp. 1328-1341.

[Kuncheva, 2004] Kuncheva, L.K. (2004), "Combining Pattern Classifiers – Methods and Algorithms".*Wiley-Interscience.*

[Kuncheva e Hoare, 2008] Kuncheva, L.I., Hoare, Z.S.J (2008), "Error-Dependency Relationships for the Naive Bayes Classifier with Binary Features", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 30(4), pp. 735-740.

[Lam e Yeung, 2007] Lam, H., Yeung, D. (2007). "A Learning Approach to Spam Detection based on Social Networks". Em *4th Conference on Email and AntiSpam.*

[Likarish et al., 2009] Likarish, P., Jung, E., Jo, I. (2009), "Obfuscated Malicious Javascript Detection using Classification Techniques", Em *4th IEEE International Conference on Malicious and Unwanted Software (MALWARE).*

[Linden et al., 2003] Linden, G., Smith, B. e York, J. (2003) "Amazon.com Recommendations: Item-to-Item Collaborative Filtering". *IEEE Internet Computing*, Vol. 7 (1), pp 76–80.

[MacQueen, 1967] MacQueen, J. B. (1967). "Some Methods for Classification and Analysis of Multivariate Observations", Em *Proceedings of the Fifth Symposium on Math, Statistics, and Probability,* pp. 281–297.

[Mafra et al., 2008] Mafra, M. P., Fraga, S. J., Moll, V., Santin, O. A. (2008), "POLVO-IIDS: Um Sistema de Detecção de Intrusão Inteligente Baseado em Anomalias". Em *VIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*.

[Markines et al., 2009] Markines, B., Catutto, C., e Menczer, F., (2009), "Social Spam Detection". Em *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web*, pp. 41-48.

[Mena, 2003] Mena, J. (2003) "Investigative Data Mining for Security and Criminal Detection". *Butterworth Heinemann*, New York, NY.

[Miyamoto et al., 2009] Miyamoto, D., Hazeyama, H., Kadobayashi, Y. (2009), "An Evaluation of Machine Learning-Based Methods for Detection of Phishing Sites"

Em *Proceedings of the 15th International Conference on Advances in Neuro-Information Processing*, pp. 539-546.

[Mukkamala et al., 2002] Mukkamala, S., Janowski, G., Sung, A. H., (2002) "Intrusion Detection Using Neural Networks and Support Vector Machines", *Proceedings of the International Joint Conference on Neural Networks*. pp. 1702-1707.

[Nelson, 2010] Nelson, B. A. (2010) "Behavior of Machine Learning Algorithms in Adversarial Environments". *PhD. Thesis*, University of California, Berkeley, 244 páginas.

[Nguyen and Armitage, 2008] Nguyen, T. T. T., Armitage, G. (2008). "A Survey of Techniques for Internet Traffic Classification using Machine Learning". *IEEE Communication Surveys and Tutorials.*Vol. 10 (4), pp. 56-76.

[Nguyen e Choi, 2010] Nguyen, H-V., 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 (4), pp. 247-252.

[Owasp, 2010] OWASP, The Open Web Security Project (2010) "Cross-site Scripting (XSS)", Disponível em https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29.

[Peddabachigari et al., 2007] Peddabachigari, S., Abraham, A., Grosan, C., Thomas, J. (2007), "Modeling Intrusion Detection System Using Hybrid Intelligent Systems", *Journal of Network and Computer Applications,* vol. 30, pp. 114-132.

[Phoha, 2002] Phoha, V. V. (2002). "Springer Internet Security Dictionary", *Springer-Verlag*, 320 páginas.

[Quinlan, 1993] Quinlan, J. R. (1993) "C4.5, Programs for machine learning". *Morgan Kaufmann,* San Mateo, Ca.

[Rabiner, 1989] Rabiner, L.R. (1989) "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", *Proceedings of the IEEE* , Vol. 77 (2), pp. 257-286.

[Rich e Knight, 1991] Rich, E. Knight, K. (1991). "Artificial Intelligence", *McGraw-Hill.*

[Richardson, 2010] Richardson, R. (2010) CSI/FBI Computer Crime Survey. Em *15th Annual 2010/2011 Computer Crime and Security*, 44 páginas.

[Riech et al., 2010] Rieck, K., Krueger, T., Dewald, A. (2010), "Cujo: Effcient Detection and Prevention of Drive-by-Download Attacks", Em *26thAnnual Computer Security Applications Conference 2010 (ACSAC 2010)*, pp. 31-39.

[Rieck et al., 2008] Rieck, K., Holz, T., Willems, C., Dussel, P., Laskov, P. (2008) "Learning and Classification of Malware Behavior", Em *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA '08)*, pp. 108-125.

[Rhodes et al, 2000] Rhodes, B., Mahaffey, J., Cannady, J. (2000). "Multiple Self-Organizing Maps for Intrusion Detection", Em *Proceedings of the 23rd National Information Systems Security Conference*.

[Saha, 2009] Saha, S. (2009). "Consideration Points: Detecting Cross-Site Scripting", *International Journal of Computer Science and Information Security (IJCSIS)*, Vol. 4, No. 1 & 2.

[Sanglerdsinlapachai e Rungsawang, 2010] Sanglerdsinlapachai, N., Rungsawang, A. (2010) "Web Phishing Detection Using Classifier Ensemble", Em *Proceedings of the 12th ACM International Conference on Information Integration and Web-based Applications & Services (iiWAS2010)*, pp. 210-215.

[Schultz et al., 2001] Schultz, M., Eskin, E., Zadok, F., Stolfo, S. (2001)."Data Mining Methods for Detection of New Malicious Executables", Em *Proceedings of the 22nd IEEE Symposium on Security and Privacy*, pp. 38–49.

[Sheng et al, 2009] Sheng, S., Wardman, B., Warner, G., Cranor, L. F., Hong, J., Zhang, C. (2009) "An Empirical Analysis of Phishing Blacklists", Em *Proceedings on Conference on Email and Anti-Spam (CEAS 09).*

[Smith, 2007] Smith, R. (2007) "An Overview of the Tesseract OCR Engine".Em *Proceedings of International Conference on Document Analysis and Recognition*.

[Sommer e Paxson, 2010] Sommer, R. e Paxson, V. (2010) "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection". Em *Proceedings of the IEEE Symposium on Security and Privacy 2010,* pp. 305-316.

[Souza e Monteiro, 2009] Souza, E. P., Monteiro, J. A. S (2009), "Estudo Sobre Sistema de Detecção de Intrusão por Anomalias, uma Abordagem Utilizando Redes Neurais". Em *XIV Workshop de Gerência e Operação de Redes e Serviços - WGRS. Sociedade Brasileira de Redes de Computadores – SBRC.*

[Theodoridis e Koutroumbas, 2006] Theodoridis, S., Koutroumbas, K. (2006). "Pattern Recognition", 3rd Edition. *Academic Press*.837 páginas.

[Tian e Jianwen, 2009] Tian, L., Jianwen, W. (2009), "Research on Network Intrusion Detection System Based on Improved K-means Clustering Algorithm".Em *Internacional Forum on Computer Science – Technology and Applications (IFCSTA 2009). IEEE Computer Science,* Vol. 1, pp. 76-79.

[Tygar, 2011] Tygar, J.D. (2011) "Adversarial Machine Learning". *IEEE Internet Computing*, Vol. 15 (5), pp. 4-6.

[Tsai e Lin, 2010] Tsai, C., Lin, C. (2010). "A Triangle Area Based Nearest Neighbors Approach to Intrusion Detection".*Pattern Recognition,* Vol. 43, pp. 222-229.

[Tsai et al., 2009] Tsai, C., Hsu, Y., Lin, C., Lin, W. (2009), "Intrusion Detection by Machine Learning" *Expert Systems with Applications*, vol. 36, pp. 11994-12000.

[Vapnik, 1995] Vapnik, V. N. (1995) The Nature of Statistical Learning Theory. *Springer*, Berlin Heidelberg New York.

[Vincent, 2007] Vincent, L. (2007) "*Google Book Search: Document Understandingon a Massive Scale"*. Em *International Conference on Document Analysis and Recognition (ICAR 2007)*, pp. 819-823

[Xiang e Zhou, 2005] Xiang, Y., Zhou, W. (2005), "A Defense System against DDoS Attacks by Large-Scale IP Traceback", *Third International Conference on Information Technology and Applications (ICITA'05)*, pp. 431-436.

[Xia et al., 2010] Xia, D. X., Yang, S. H. e Li, C. G., (2010). "Intrusion Detection System Based on Principal Component Analysis and Grey Neural Networks". Em *2nd International Conference on Networks Security Wireless Communications and Trusted Computing*, pp. 142-145.

[Xiao et al., 2007] Xiao, H., Hong, F., Zhang, Z., Liao, J. (2007). "Intrusion Detection Using Ensemble of SVM Classifier". *Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FKSD 2007)*, pp. 45-49.

[Xu et al., 2007] Xu, X., Sun, Y., Huang, Z. (2007) "Defending DDoS Attacks Using Hidden Markov Models and Cooperative Reinforcement Learning", Em *(PAISI) Pacific Asia Workshop on Intelligence and Security Informatics*, pp. 196–207.

[Ye et al, 2007] Ye, Y., Wang, D., Li, T., Ye, D. (2007), "IMDS: Intelligent Malware Detection System". Em *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD '07), pp. 1043-1047.

[Yue e Wang, 2009] Yue, C., Wang, H. (2009) "Charatering Insecure JavaScript Practice on the Web. Em *18th International Conference on the World Wide Web*.

[Wang et al, 2003] Wang, J-H., Deng, P.S., Fan, Y-S., Jaw, L-J., Liu, Y-C. (2003), "Virus Detection Using Data Mining Techniques", Em *Proceedings of the 37th International Conference on Security Technology*, pp. 71-76.

[Witten e Frank, 2000] Witten, I.H., Frank, E. (2000) "Data Mining: Practical Machine Learning tools and Techniques with Java Implementations*". Morgan Kaufmann*.

[Wu, 2009] Wu, S. (2009) "Behavior-based Spam Detection Using a Hybrid Method of Rule-based Techniques and Neural Networks", *Expert Systems with Applications*, Vol. 36 (3), pp. 4321-4330.

[Wu et al., 2011] Wu, Y-C., Tseng, H-R., Yang, W., Jan, R.H. (2011) "DDoS detection and traceback with decision tree and grey relational analysis*", International Journal Ad Hoc and Ubiquitous Computing*, Vol. 7 (2), pp.121–136.

[Zhang et. al., 1996] Zhang, T., Ramakrishnan, R., & Livny, M., (1996) "BIRCH: An Efficient Data Clustering Method for Very Large Databases", Em *Proceedings of the ACM SIGMOD.*

[Zhong e Yue, 2010] Zhong, R., Yue, G. (2010) "DDoS Detection System Based on Data Mining", *Em Proceedings of the Second International Symposium on Networking and Network Security (ISNNS '10)*, pp. 062-065.

Capítulo

3

Técnicas para Análise Dinâmica de *Malware*

Dario Simões Fernandes Filho[1], Vitor Monte Afonso[1], Victor Furuse Martins[1], André Ricardo Abed Grégio[1,2], Paulo Lício de Geus[1], Mario Jino[1], Rafael Duarte Coelho dos Santos[3]
[1]Universidade Estadual de Campinas (Unicamp)
[2]Centro de Tecnologia da Informação Renato Archer (CTI/MCT)
[3]Instituto Nacional de Pesquisas Espaciais (INPE/MCT)

Abstract

The threat posed by malware to systems security led to the development of analysis mechanisms that operate in a dynamic and controlled manner. These mechanisms (dynamic analysis systems) use a variety of techniques to obtain the behavior from malware samples' execution. The complexity of those techniques varies from monitoring events on user-level interfaces, to Web malicious code desobfuscation, to hooking operating system (OS) kernel structures. In this book chapter, we present the main techniques that are used to perform malware dynamic analysis, either at the operating system level or at the Web applications level. These techniques are used in the main publicly available analysis systems. Also, we show some tools used to capture information about the execution of malware samples and how to build a simple system to analyze malware using open-source and free tools. Finally, we describe in details a case study about the analysis of a malware sample that starts its attack from the browser and then compromises the OS.

Resumo

A ameaça dos códigos e programas maliciosos à segurança dos sistemas computacionais fez com que surgissem muitos sistemas cujo propósito é analisar, de maneira dinâmica e controlada, tais programas. Estes sistemas se utilizam de diversas técnicas para obter o comportamento apresentado por amostras de malware durante sua execução. A complexidade destas técnicas varia desde a monitoração de eventos através de interfaces no nível de privilégio dos usuários, passando pela desofuscação de programas maliciosos em linguagens típicas da Web, até a inserção de código em estruturas do kernel do

sistema operacional. Neste capítulo, visa-se apresentar as principais técnicas utilizadas para efetuar a análise dinâmica de malware - sejam estes do nível do sistema operacional ou da Web - e as quais estão presentes nos principais sistemas de análise disponíveis publicamente. Além disso, são mencionadas algumas ferramentas utilizadas na captura de informações da execução dos programas maliciosos. Mostra-se, também, como construir um sistema simples de análise dinâmica de malware utilizando ferramentas gratuitas ou de código aberto. Para finalizar o capítulo, os leitores terão a oportunidade de acompanhar um estudo de caso completo da análise de um exemplar de malware, do ataque a partir da Web até o comprometimento do sistema operacional.

3.1. Código Malicioso

Ataques realizados por meio de *malware* tomaram uma dimensão tão grande que atividades simples como a navegação na Web [Chen et. al. 2011], [Cova et. al. 2010], a participação em redes sociais digitais [Stringhini et. al. 2010], [Yang et. al. 2011] e o uso de celulares[Becher et. al. 2011], [Egele et. al. 2011] tornaram-se perigosas. Para se ter uma idéia do cenário nacional, quase a totalidade dos 142.844 incidentes reportados ao CERT.br[1] entre janeiro e dezembro de 2010 referem-se a ataques envolvendo código malicioso. Isto pode ser observado na Figura 3.1.

Figura 3.1. Incidentes reportados ao CERT.br em 2010. Fonte: http://www.cert.br/stats/.

Um estudo recente publicado na *IBM Systems Magazine* [IBM 2011] mostra que um dos maiores culpados pelo alto custo financeiro causado por um ataque é relacionado à atividade de *malware*. Na Figura 3.2 é mostrado o resultado do levantamento do custo de invasões em diversos países no ano de 2009.

No decorrer do tempo, em intervalos determinados, pode-se notar maior incidência de grupos específicos de *malware*. Por exemplo, nos últimos cinco anos tem havido uma proliferação grande de *botnets*[2], gerando tentativas (muitas vezes frustradas) do fechamento destas por parte de autoridades e pesquisadores de segurança [Stone-Gross et. al. 2011], [Zhang et. al. 2011], [Shin et. al. 2011], [Cho et. al. 2010].

[1]Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil. (Link: `http://www.cert.br`).

[2]redes de máquinas comprometidas controladas remotamente por comandos dados por um atacante.

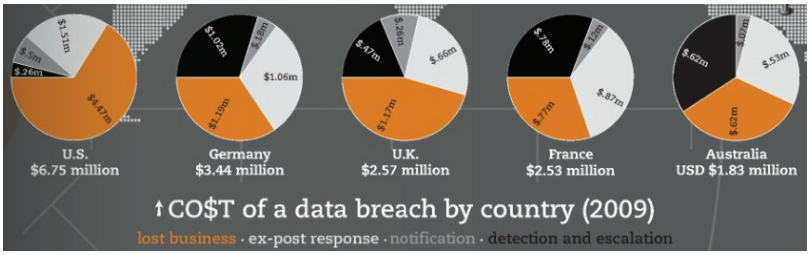

Figura 3.2. Custo de uma invasão por país em 2009. Fonte: [IBM 2011].

A incidência pontual de certas classes de *malware* faz com que estes sejam frequentemente responsáveis por sérios incidentes de proporções globais, tais como os recentes resultados do alastramento dos *malware* StuxNet [Falliere et. al. 2010], Zeus [Binsalleeh et. al. 2010] e Conficker [Shin and Gu 2010], ou casos que ganharam notoriedade no passado como, por exemplo, ILoveYou (2000), Nimda (2001), Code Red (2001), Slammer (2003), Blaster/Sasser (2004).

Embora tais grupos, ou famílias de *malware*, tendam a explorar as vulnerabilidades do momento e do contexto no qual estão inseridos, é comum a revisitação de antigos ataques, o compartilhamento de funções com finalidade maliciosa—varreduras, conexões com servidores, métodos de exploração—ou mesmo a mutação de pequenos trechos do código, a qual pode levar à redução da detectabilidade das famílias cujas vacinas já são conhecidas e, em paralelo, manter as funcionalidades e o comportamento malicioso apresentados nos ancestrais.

No âmbito da defesa, a mera identificação de um arquivo executável como sendo um *malware* conhecido (já coletado, analisado e talvez combatido) permite a tomada de contra-medidas de maneira rápida e eficiente. Isto facilita a contenção de danos, minimiza prejuízos e reduz a possibilidade de infecção em redes e sistemas ainda intactos por meio de regras de bloqueio ou aplicação de *patches* de segurança.

Com isso, é necessário haver meios de se identificar *malware* para que ações defensivas possam ser coordenadas, ao mesmo tempo em que sejam obtidos conhecimentos sobre o comportamento de um certo *malware* e sobre a possível extensão dos danos aos sistemas comprometidos por este código específico. A solução mais tradicional para identificação de *malware* ainda é o uso de antivírus. Porém, é cada vez mais comum a utilização de sistemas de análise dinâmica para traçar a execução de um programa e verificar por ações maliciosas. Adiante, será apresentada uma taxonomia sucinta de algumas classes importante de *malware*. Para motivar o leitor na identificação de comportamentos maliciosos, serão discutidos brevemente problemas relacionados ao emprego de mecanismos antivírus, da análise estática de *malware* e, por fim, da análise dinâmica, que é o tema deste texto.

3.1.1. Classes de código malicioso

Códigos maliciosos podem ser classificados de uma maneira geral de acordo com alguma faceta específica de seu comportamento. Embora atualmente seja difícil "enquadrar" um exemplar de *malware* em uma única classe, devido à evolução destes códigos e à facilidade de se adicionar novas funcionalidades, a taxonomia apresentada a seguir ainda é utilizada para se referir a certos tipos de *malware* e também nos identificadores atribuídos por mecanismos antivírus.

As classes e descrições abordadas a seguir baseiam-se nas definições de Peter Szor [Szor 2005].

Vírus. Segundo Fred Cohen, considerado o pai dos vírus de computador, "*um vírus é um programa que é capaz de infectar outros programas pela modificação destes de forma a incluir uma cópia possivelmente evoluída de si próprio*." Os vírus costumam infectar seções de um arquivo hospedeiro, propagando-se através da distribuição deste arquivo. Comumente, os vírus necessitam ser acionados e propagados por uma entidade externa (e.g., usuário).

Worm. Os *worms*, por sua vez, propagam-se pela rede e em geral não necessitam de ativação por parte do usuário. Uma outra característica é a independência de outro programa para disseminação e ataque. Alguns *worms* podem carregar dentro de si um outro tipo de *malware*, que é descarregado na vítima após o sucesso da propagação e do ataque.

Trojan. Cavalos-de-tróia são tipos comuns de *malware* cujo modo de infecção envolve despertar a curiosidade do usuário para que este o execute e comprometa o sistema. Este tipo de código malicioso também pode ser encontrado em versões modificadas de aplicações do sistema operacional, substituídas por indivíduos maliciosos. Estas versões apresentam as mesmas funcionalidades da aplicação íntegra, porém também contêm funcionalidades adicionais com a finalidade de ocultar as ações malignas.

Backdoor. Tradicionalmente, as *backdoors* permitem a um atacante a manutenção de uma máquina comprometida por abrirem portas que permitem a conexão remota. Um outro tipo de *backdoor* envolve erros na implementação de uma aplicação ou sistema que o tornam vulnerável e podem levar à execução de código arbitrário na máquina da vítima

Downloader. Um programa malicioso que conecta-se à rede para obter e instalar um conjunto de outros programas maliciosos ou ferramentas que levem ao domínio da máquina comprometida. Para evitar dispositivos de segurança instalados na vítima, é comum que *downloaders* venham anexos à mensagens de correio eletrônico e, a partir de sua execução, obtenham conteúdo malicioso de uma fonte externa. (e.g., Web site).

Dropper. Possui características similares as dos *downloaders*, com a diferença que um *dropper* é considerado um "instalador", uma vez que contém o código malicioso compilado dentro de si.

Rootkit. É um tipo especial de *malware*, pois consiste de um conjunto de ferramentas para possibilitar a operação em nível mais privilegiado. Seu objetivo é permanecer residindo no sistema comprometido sem ser detectado e pode conter *exploits*, *backdoors* e versões *trojan* de aplicações do sistema. Os *rootkits* modernos atacam o *kernel* do sistema

operacional, modificando-o para que executem as ações maliciosas de modo camuflado. Este tipo de *rootkit* pode inclusive interferir no funcionamento de mecanismos de segurança.

3.1.2. O problema dos antivírus

Um dos mecanismos de defesa contra *malware* mais populares ainda é o antivírus, que pode ser explicado basicamente como um programa que varre arquivos ou monitora ações pré-definidas em busca de indícios de atividades maliciosas. Em geral, os antivírus operam de duas formas para identificar código malicioso: correspondência de padrões em bancos de dados de assinaturas ou heurísticas comportamentais. Na detecção por assinatura, um arquivo executável é dividido em pequenas porções (*chunks*) de código, as quais são comparadas com a base de assinaturas do antivírus. Assim, se um ou mais *chunks* do arquivo analisado estão presentes na base de assinaturas, a identificação relacionada é atribuída ao referido arquivo. Já na detecção por heurística, um arquivo sob análise é executado virtualmente em um emulador minimalista e os indícios de comportamento suspeito são avaliados a fim de se verificar se a atividade realizada pelo programa pode ser considerada normal ou se um alerta deve ser emitido.

O grande problema dos antivírus é o surgimento frequente e crescente de variantes de *malware* previamente identificados, cujas ações modificadas visam evadir a detecção. Essas variantes precisam ser tratadas muitas vezes individualmente (e manualmente), no caso da confecção de uma assinatura para um antivírus. Além do mais, pode ser preciso modificar a heurística de detecção de toda uma classe para que esta seja capaz de identificar a variante, sempre levando-se em conta se a modificação não vai gerar mais falsos-positivos. Esses, por sua vez, são uma outra dificuldade no processo de criação de assinaturas/heurísticas de *malware*—identificar erroneamente uma aplicação inofensiva do usuário como sendo maliciosa e, consequentemente, removê-la, colocá-la em quarentena (bloqueio) ou restringir de algum modo a usabilidade de um sistema computacional não é um requisito desejável para os fabricantes de antivírus.

Deve-se considerar também que muitos exemplares de *malware* possuem mecanismos próprios de defesa cujas ações variam entre desabilitar as proteções existentes no sistema operacional alvo (*firewall*, antivírus, *plugins* de segurança), verificar se o exemplar está sob análise (o que pode causar uma modificação em seu comportamento em razão disto), e disfarçar-se de programas do sistema, inclusive de falsos antivírus. Portanto, evitar que o mecanismo antivírus seja desabilitado, identificar programas maliciosos rapidamente com uma taxa mínima de falsos-positivos sem interferir na usabilidade do sistema e, ainda, expandir a capacidade de detecção sem causar sobrecarga têm se tornado tarefas cada vez mais difíceis. Contribui a isso o cenário atual das variantes, popularização das redes sociais, proliferação de celulares e *tablets* com o mesmo poder e funcionalidade de computadores, e o estabelecimento da *cloud computing*.

Não bastassem os problemas supracitados, a comunidade de desenvolvedores de antivírus ainda não possui padronização para a classificação dos *malware* identificados por suas ferramentas. Isso faz com que o processo de resposta a incidentes de segurança envolvendo código malicioso seja prejudicado, tornando-o ineficiente em determinados casos. De modo ilustrativo, pode-se tomar como exemplo alguns sistemas que identificam

um programa como malicioso baseando-se apenas no *packer*[3] com o qual o arquivo foi cifrado, ou ainda, que fazem a identificação baseada em apenas uma faceta do binário, a qual pode não condizer com as atividades efetuadas na máquina[4]. Divergências de nomenclatura terminam por atrapalhar na classificação, pois a falta de um padrão faz com que cada fabricante de antivírus atribua o identificador que desejar. Assim, um dado *malware* pode ser identificado como 'Família_X.A' por um fabricante, 'Família_X.123' por outro e 'Família_Z.y' por um terceiro. Para ilustrar este problema de forma mais clara, na Tabela 3.1 estão os resultados obtidos pela indentificação de um exemplar de *malware* submetido ao VirusTotal[5].

Tabela 3.1. Resultado do VirusTotal para um exemplar de código malicioso.

Antivirus	*Identificação*
McAfee	**PWS-OnlineGames.bu**
K7AntiVirus	**Trojan**
NOD32	**a variant of Win32/PSW.OnLineGames.OAF**
F-Prot	**W32/Agent.L.gen!Eldorado**
Symantec	**Infostealer.Gampass**
Norman	**W32/Packed_Upack.H**
TrendMicro-HouseCall	**TSPY_ONLING.AG**
Avast	**Win32:OnLineGames-ECV [Trj]**
eSafe	**Win32.Looked.gen**
Kaspersky	**Trojan-GameThief.Win32.OnLineGames.akyb**
Sophos	**Mal/Behav-214**
Comodo	**TrojWare.Win32.Trojan.Inject.ÎI**
DrWeb	**Trojan.DownLoader.62898**
VIPRE	**BehavesLike.Win32.Malware.bsu (vs)**
AntiVir	**TR/Spy.Gen**
TrendMicro	**TSPY_ONLING.AG**
McAfee-GW-Edition	**Heuristic.BehavesLike.Win32.Packed.A**
eTrust-Vet	**Win32/Lemir!generic**
AhnLab-V3	**Win-Trojan/Xema.variant**
VBA32	**BScope.Trojan-PSW.SataGames.3**
PCTools	**Rootkit.Order**
Rising	**Trojan.PSW.Win32.GameOL.odt**
Ikarus	**Virus.Win32.Virut**
Panda	**Trj/Lineage.ISP**

Para a resposta ser efetiva, isto é, permitir a remoção do código malicioso e as atividades derivadas da infecção do sistema comprometido, é necessário, em muitos ca-

[3]No contexto deste texto, um *packer* é um programa utilizado para cifrar ou comprimir o conteúdo de um arquivo/programa malicioso para disfarçá-lo.

[4]Exemplares de *malware* infectados por outros *malware* podem fazer com que o antivírus emita um alerta devido a assinatura do infectante, e não do *malware* original infectado.

[5]VirusTotal é um serviço gratuito disponibilizado publicam ente na Internet em `http://www.virustotal.com`, para o qual podem ser submetidos programas que são analisados por grande parte das ferramentas antivírus atualmente disponíveis no mercado.

sos, uma intervenção manual, dado que nem todos os antivírus possuem o procedimento de desfazimento das ações efetuadas (nem há tal procedimento para todas as amostras de *malware* em atividade, devido ao já citado problema da existência massiva de variantes). Alguns casos mais epidêmicos acabam por ter aplicações ou rotinas automatizadas para remoção, as quais podem ser inócuas na presença de variantes. Entretanto, o sucesso na obtenção do procedimento de remoção (automático ou manual) correto é completamente dependente da identificação provida pelo mecanismo antivírus que, como visto anteriormente, pode ter sido feita de maneira errônea.

3.1.3. Análise estática e suas limitações

A análise de código malicioso visa o entendimento profundo do funcionamento de um malware—como atua no sistema operacional, que tipo de técnicas de ofuscação são utilizadas, quais fluxos de execução levam ao comportamento principal planejado, se há operações de rede, download de outros arquivos, captura de informações do usuário ou do sistema, entre outras atividades. Divide-se a análise de malware em análise estática e dinâmica, sendo que no primeiro caso tenta-se derivar o comportamento do malware extraindo características de seu código sem executá-lo, através de análise de strings, disassembling e engenharia reversa, por exemplo. Já na análise dinâmica, o malware é monitorado durante sua execução, por meio de emuladores, debuggers, ferramentas para monitoração de processos, registros e arquivos e tracers de chamadas de sistema. Dependendo da técnica ou ferramenta que se utiliza para fazer cada análise, a velocidade pode variar, mas, em geral, análises estáticas simples são mais rápidas do que as dinâmicas. Entretanto, se há a necessidade de engenharia reversa, se o malware possui muitos fluxos de execução ou se está comprimido com um packer de difícil descompressão, a análise dinâmica tradicional é muito mais rápida e eficaz na provisão de resultados acerca do comportamento do exemplar analisado. Em [Moser et al. 2007] é apresentada uma ferramenta que transforma um programa de forma a ofuscar seu fluxo de execução, disfarçar o acesso a variáveis e dificultar o controle dos valores guardados pelos registradores, mostrando que a análise estática de um malware ofuscado por essa ferramenta é um problema NP-difícil. Além disso, durante a análise estática não se sabe como o sistema vai reagir em resposta às operações do programa.

A análise estática pode ser utilizada para obter informações gerais sobre o programa e para identificar a existência de código malicioso. Dentre as técnicas utilizadas para a obtenção de informações gerais estão a geração de hashes criptográficos que identificam o arquivo de forma única, a identificação das funções importadas e exportadas, a identificação de código ofuscado e a obtenção de cadeias de caracteres que possam ser lidas por uma pessoa, como mensagens de erro, URLs e endereços de correio eletrônico. Para identificar código malicioso são usadas, de forma geral, duas abordagens: a verificação de padrões no arquivo binário e a análise do código assembly gerado a partir do código de máquina do malware. No caso da verificação de padrões, são geradas seqüências de bytes, chamadas de assinaturas, que identificam um trecho de código freqüentemente encontrado em programas maliciosos e verifica-se se o programa possui esta seqüência. Já no caso da investigação do código assembly, são empregadas técnicas de análise mais profundas que buscam padrões de comportamento malicioso. Em [Song et al. 2008] os autores transformam o código assembly em uma linguagem intermediária e, a partir

desta, extraem informações a respeito do fluxo de dados e fluxo de controle do programa. A maior dificuldade encontrada pela análise estática é o uso dos packers. Para combater a evolução destes, foram desenvolvidos diversos mecanismos [Yegneswaran et al. 2008] [Kang et al. 2007] [Martignoni et al. 2007] que visam obter o código não ofuscado do malware, permitindo que a análise estática seja efetuada.

3.1.4. Analisadores dinâmicos

Embora o *modus operandi* dos antivírus ainda seja majoritariamente a identificação com base em assinaturas, tem crescido o interesse por heurísticas, pois uma heurística bem construída pode resultar na substituição de dezenas de assinaturas. Para gerar uma heurística que identifique um exemplar de *malware* (ou uma classe), é necessário conhecer primeiro o seu comportamento, isto é, quais são as ações realizadas no sistema operacional alvo que denotam uma atividade anormal ou suspeita. Como opção aos emuladores limitados embutidos nos antivírus com o objetivo de realizar a identificação por heurísticas, os sistemas de análise dinâmica de *malware* foram sendo aprimorados e popularizados nos últimos anos. Tais sistemas lançam mão de uma variedade de técnicas para monitorar a execução de um *malware* de maneira controlada, utilizando desde a instrumentação de emuladores complexos até a interceptação de chamadas ao *kernel* do sistema operacional monitorado.

É comum as empresas fabricantes de antivírus possuirem seus próprios sistemas de análise dinâmica de *malware*, também chamados de *sandboxes*, mas há soluções disponíveis gratuita e publicamente via Internet, como *Anubis*[6], *ThreatExpert*[7] e *CWSandbox*[8]. No contexto deste trabalho, uma *sandbox* é um ambiente restrito e controlado que permite a execução de um código malicioso de forma a causar danos mínimos à sistemas externos por meio da combinação de filtragem e bloqueio de tráfego de rede e da execução temporizada do *malware*. Em geral, o *malware* é executado por quatro ou cinco minutos e, durante este tempo, são monitoradas as ações pertinentes tanto ao *malware* quanto aos processos derivados dele. Após o período de monitoração, um relatório de atividades é gerado para análise.

Entretanto, um dos grandes problemas da análise dinâmica é que a interpretação dos relatórios é deixada a cargo do usuário que submeteu o exemplar, ou seja, não se pode realmente dizer que o sistema fez uma análise, mas sim, uma monitoração da execução com registro das atividades efetuadas no período. Além disso, limitações das técnicas comumente utilizadas para interceptar as chamadas de sistema ou instrumentar o ambiente da *sandbox* podem levar à evasão da análise por exemplares de *malware* modernos. Isso, agravado pelo fato de alguns exemplares de *malware* terem por característica a mutação de seus comportamentos durante execuções distintas, faz com que o relatório gerado por um sistema possa ser diferente do relatório gerado por outro, no que diz respeito às ações efetuadas pelo *malware* sob análise.

Ainda assim, a análise dinâmica é um importante instrumento para prover informações úteis a um usuário ou analista de segurança, permitindo tomadas de decisão com

[6] `http://anubis.iseclab.org`
[7] `http://www.threatexpert.com`
[8] `http://mwanalysis.org`

base no padrão de ações nocivas executadas pelo exemplar de *malware* analisado.

3.1.5. Organização do texto e considerações

Este texto visa cobrir as técnicas mais utilizadas em análise dinâmica de *malware*, sejam eles voltados aos ataques via Web ou no nível do sistema operacional e suas aplicações. Na Seção 3.2 são apresentadas algumas destas técnicas, enquanto que as ferramentas que as utilizam são mostradas na Seção 3.3. É esperado que o leitor compreenda os pontos fortes e as limitações de cada técnica, bem como o funcionamento destas de forma a poder escolher aquela que se adapta às suas necessidades de análise. Um conjunto de ferramentas para a extração de informações da execução de um código malicioso é mostrado na Seção 3.4. As etapas da análise serão ilustradas por meio de exemplos práticos, permitindo o seu acompanhamento detalhado em um estudo de caso apresentado na Seção 3.6.

3.2. Técnicas de Análise

A análise dinâmica de *malware*, normalmente realizada para identificar as ações desenvolvidas pelo *malware* no sistema, pode ser implementada através de diversas técnicas, cada uma com suas vantagens e desvantagens. Neste capítulo serão apresentadas as principais técnicas utilizadas por sistemas de análise importantes ou por ferramentas que auxiliam na análise dinâmica. Dentre elas, podemos citar *Virtual Machine Introspection* e *Hooking* como as principais. Cada subseção irá tratar cada uma delas de forma detalhada, de forma que ao final deste capitulo o leitor será capaz de compreender cada uma delas, podendo escolher dentre elas a que mais se adequar para a sua análise.

3.2.1. *Virtual Machine Introspection*

Virtual Machine Introspection (VMI) é uma técnica onde se cria uma camada entre o sistema de análise (*guest*) e o ambiente de processamento (*host*), de forma que todas as ações que ocorrem dentro do sistema *guest* não são propagadas para o *host*. Seu uso possibilita a captura das ações que estão sendo executadas dentro do ambiente de análise, sem que haja qualquer interferência dentro do ambiente onde está sendo executado, possibilitando assim que um *malware* seja análise sem qualquer tipo de modificação no sistema *guest*. Como não há modificação no sistema *guest*, a análise fica transparente para o *malware*, impossibilitando qualquer tentativa de identificação do componente de captura. A ilustração apresentada na figura 3.3 visa clarificar o funcionamento da técnica de *Virtual Machine Introspection*, que será tratada mais detalhadamente nas seções seguintes. É possível verificar a diferenciação entre os sistemas *guest*, que funcionam dentro do ambiente emulado e o sistema *host*, onde está executando a aplicação responsável pela implementação da camada entre o ambiente *guest* e o *host*.

Programas para emulação e virtualização, tais como Qemu [Bellard 2005], VMWare [VMware 2011] e VirtualBox [Virtualbox 2011] possibilitam a aplicação desta técnica, dado que estes implementam a camada intermediária de maneira nativa, fazendo assim uma distinção entre o ambiente real (ou sistema *host*) e o emulado/virtualizado (ou o *guest*). A seguir serão explicados os conceitos de emulação e como a técnica de VMI pode ser aplicado à ela.

Emulação é um termo utilizado na área de computação para descrever o modo de

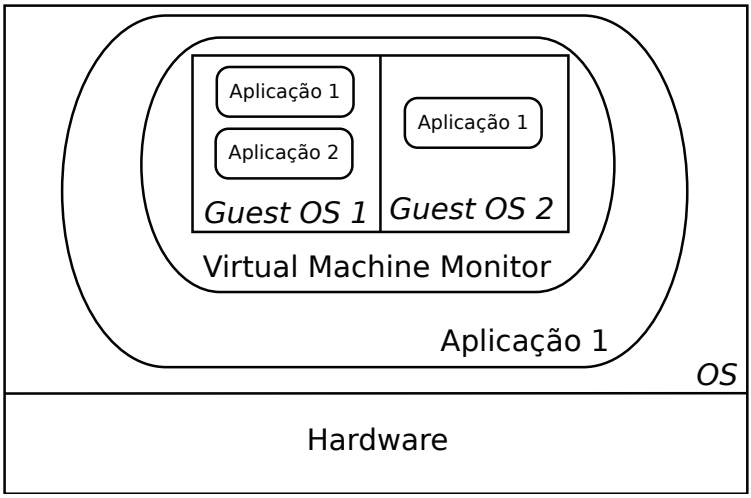

Figura 3.3. Ilustração do uso da técnica de *Virtual Machine Introspection*.

operação de um *software* desenvolvido para simular um determinado *hardware* [Martignoni et. al. 2009], como por exemplo um processador específico diferente do que está executando o emulador. Na análise de *malware*, este tipo de *software* é utilizado para simular uma máquina, para que seja possível a instalação do sistema operacional que será utilizado no processo de análise. Além disso, o emulador pode ser utilizado para separar o ambiente de análise, fazendo com que as ações efetuadas por um exemplar de *malware* durante sua execução não contaminem o ambiente real, dado que as modificações executadas pelo *malware* só irão ocorrer no ambiente de análise *host*. Um emulador muito utilizado para este fim é o Qemu [Bellard 2005]—uma ferramenta de código aberto e de fácil utilização com o qual é possível emular o processador, o disco rígido e demais dispositivos do sistema de forma que seja possível instalar vários tipos de sistemas operacionais.

Virtualização, de modo similar à emulação, é utilizada para simular uma máquina, tornando possível a instalação e execução de vários sistemas operacionais em paralelo com o mesmo *hardware*. Porém, na virtualização as instruções são executadas no *hardware* real da máquina, ao contrário do que ocorre na emulação, na qual as instruções são executadas em um processador emulado. Isto torna a virtualização mais rápida em relação à emulação, pois as instruções do *guest* ficam sob responsabilidade do *hardware* do *host*. A limitação da virtualização é que ela possibilita somente a instalação de sistemas cuja arquitetura seja a mesma do *host*. Quando se utiliza um virtualizador para análise de *malware*, as funcionalidades são parecidas com as do emulador: há o isolamento entre os ambientes dos sistemas *host* e *guest*. Nesse caso, o programa responsável pela virtualização, acrescenta uma camada adicional entre o ambiente real e o de análise chamada de *Virtual Machine Monitor*. Esta camada realiza a abstração do *hardware* real para as

máquinas virtuais, executando suas ações de forma que sejam percebidas somente dentro do ambiente virtual [Rosenblum 2004].

Tanto a virtualização quanto a emulação fazem uma distinção entre o ambiente onde o *malware* é executado e o real. Para simplificar a forma de mencionar tais ambientes, a partir de agora serão utilizados os termos *guest* e *host*, sendo que o primeiro identifica o sistema operacional virtualizado ou emulado utilizado na análise dinâmica e o último identifica o sistema base que executa o emulador ou virtualizador.

O programa de virtualização/emulação responde ao sistema *guest* da mesma forma que os dispositivos físicos (processador, disco rígido, placas de rede e vídeo etc) responderiam, sem entretanto comprometer o *host* no qual ele está sendo executado. Essa transparência faz com que o *host* tenha total controle sobre o *guest*, podendo inclusive observar em tempo real o estado dos diversos recursos da máquina onde o *malware* está sendo executado, como memória e CPU, por exemplo.

Desta forma, torna-se trivial a obtenção de informações a respeito da execução do *malware* de maneira externa ao *guest*, bastando que se modifique o *software* responsável por executar a emulação/virtualização para que este realize a captura dos dados. A modificação de um programa de virtualização ou emulação com o objetivo de se obter informações internas ao *guest* a partir do sistema *host* é chamada de VMI e, com a utilização desta técnica é possível alcançar um nível de privilégio adicional na camada de abstração intermediária entre o *host* e o *guest*. Uma das características mais interessantes da VMI é que esta torna possível a análise de *malware* cuja execução ocorre no nível do *kernel*, tais como os *rootkits*. As técnicas comumente utilizadas por *rootkits* para esconder ou alterar estruturas internas do sistema operacional atacados [Hoglund and Butler 2005] podem inviabilizar sua detecção e monitoração por mecanismos de segurança ou outros métodos de análise, como por exemplo, *hooking* (Seção 3.2.2).

Para ilustrar a técnica de VMI, um tipo de informação que pode ser capturada do sistema *guest* são as *syscalls* que o *malware* executou durante a análise. Um método muito utilizado para identificar a ocorrência de uma *syscall* baseia-se na leitura do valor contido no registrador `SYSENTER_EIP_MSR` do processador. Este registrador é utilizado quando ocorre uma instrução do tipo "`SYSENTER`", que indica que uma chamada de sistema deve ser feita. Quando a chamada é efetuada, o sistema realiza a troca de contexto entre o espaço de usuário e o espaço de *kernel*, permitindo finalmente a execução da *syscall*.

Uma forma de identificar qual *syscall* está sendo invocada é através da leitura do valor contido no registrador `EAX`. No momento em que a instrução `SYSTENTER` for executada, o registrador EAX armazena um valor utilizado para se encontrar o endereço da *syscall* que se quer realizar. Esse valor corresponde ao índice de uma tabela que contém os endereços de todas as *syscalls* possíveis no sistema operacional. Em sistemas *Windows*, esta tabela corresponde a uma estrutura que atende pelo nome de *System Service Dispatch Table*. Os parâmetros utilizados para compor a *syscall* podem ser obtidos através de verificações nos registradores do processador e na memória do sistema, no momento em que a chamada estiver sendo executada.

Um sistema de análise dinâmica bem conhecido e disponível publicamente para

utilização através da Internet é, *Anubis* [Anubis 2011], o qual utiliza a técnica de VMI para monitorar as ações de um exemplar de *malware* durante sua execução em um sistema operacional *Windows XP*. A fim de aplicar VMI, *Anubis* foi implementado sobre o emulador *Qemu*, modificado para efetuar a captura de informações de maneira externa ao *guest*. Mais detalhes da implementação de *Anubis* podem ser encontrados em [Bayer et. al. 2006].

A principal desvantagem da VMI é que um *malware* pode detectar que está sendo executado em um ambiente emulado/virtual, evitando a análise como um todo ou apresentando um comportamento alternativo ao malicioso. No caso dos emuladores, a detecção pode ser feita de forma muito simples, por exemplo, através da realização de uma instrução no processador que causa um comportamento específico. Um dos modos utilizados para realizar tal verificação é por meio de *bugs* conhecidos em processadores de determinadas arquiteturas que fazem com que certas instruções não se comportem como esperado. Se esta instrução for executada no emulador e este não estiver preparado para apresentar o mesmo comportamento de um processador real, o *malware* irá perceber essa diferença, podendo parar ou modificar a sua execução [Raffetseder et. al. 2007]. A detecção de ambiente virtualizado também é simples, com apenas uma instrução *assembly* que, mesmo executada em um nível de baixo privilégio, retorna informações internas sobre o sistema operacional presente no *guest*. Tais informações identificam o ambiente virtualizado com base nas diferenças entre estes e sistemas reais [Quist and Smith 2006].

Para contornar as técnicas de anti-análise, existem meios de detectar que um *malware* verifica se está em ambiente emulado, ou mesmo de modificar alguns valores presentes no ambiente virtual para tentar disfarçá-lo [Kang et. al. 2009],[Liston and Skoudis 2006]. Entretanto, o uso destas técnicas muitas vezes é insuficiente e o *malware* ainda pode detectar que está sendo executado em ambiente emulado/virtual.

Um outro sistema utilizado para traçar o comportamento de um *malware* e que se baseia em VMI é *Ether* [Dinaburg et. al. 2008]. Este sistema utiliza VMI para obter as ações realizadas no *guest*, porém, ao contrário de *Anubis*, *Ether* se utiliza de virtualização direta do *hardware*, o que o torna imune às técnicas de anti-análise que verificam se o *hardware* é real ou emulado. A implementação da VMI é feita em uma versão modificada do *Xen hypervisor*, um *software* de virtualização. Uma vantagem de *Ether* sobre *Anubis* diz respeito à análise de exemplares de *malware* com *packers* que apresentam mau funcionamento em emuladores. Diferentemente de *Anubis*, *Ether* consegue analisar este tipo de *malware* sem qualquer problema em sua execução, dado que o *Xen* utiliza o *hardware* nativo para executar as operações do processador. Entretanto, *Ether* apresenta problemas de desempenho para obter o traço composto pelas *syscalls* que o *malware* realizou. Isso ocorre porque cada chamada de sistema executada pelo programa gera uma *page fault*, a qual é tratada pelo componente de *Ether* responsável pela obtenção do referido traço. Além disso, apesar de ser dito em sua documentação que não é possível detectar sua presença, existem meios de verificá-la, como os descritos em [Pék et. al. 2011]. Nesta referência, são apontados alguns possíveis modos de detectar a presença de *Ether*, como por exemplo através de uma modificação feita pelo sistema de análise que desabilita o bit `TSC` (*Time-Stamp Counter*). Este bit é retornado quando se executa a instrução `CPUID` e serve para indicar quando a instrução *RDTSC* é suportada. Portanto, para detectar a execução em *Ether*, basta que se execute a instrução `CPUID` e se observe o valor retornado

no bit `TSC`.

Além dos problemas apresentados com o uso da VMI para captura de informações, há uma outra limitação que diz respeito ao desempenho. Como o componente que obtém as informações fica na camada da VMI, que faz o interfaceamento entre o *guest* e o *host*, os dados capturados são de nível mais baixo, isto é, valores encontrados em registradores da CPU ou endereços de memória. Porém, a análise do comportamento do *malware*, isto é, as modificações feitas no sistema da vítima, requer a obtenção de valores de mais alto nível, como nomes de arquivos criados, registros modificados e processos inicializados. Assim, para acessar tal conteúdo precisa-se interpretar, em tempo de execução, os dados contidos na memória e no processador durante a monitoração do *malware*, o que na maioria das vezes não é uma tarefa fácil e causa uma sobrecarga no processo de análise.

3.2.2. *Hooking*

A técnica de *hooking* pode ser definida como um meio de se alterar as requisições e respostas resultantes das interações realizadas em um sistema operacional ou por suas aplicações, através da interceptação das funções ou eventos utilizados [Holy Father 2004]. Pode-se categorizar *hooking* como sendo de modo de usuário (*userland hooking*) ou de modo de *kernel* (*kernel hooking*). O que difere estes dois tipos é a extensão da modificação que pode ser feita no sistema e, consequentemente, nas aplicações. *Malware* geralmente utilizam-se de técnicas de *hooking* para capturar ou modificar informações que estejam transitando em uma aplicação ou no sistema operacional. Através disto é possível a ocultação de suas atividades, dificultando assim a sua identificação. Alguns *rootkits* empregam *hooking* para tornar sua presença indetectável ao sistema [Hoglund and Butler 2005]. Na figura 3.4 é apresentada uma ilustração da aplicação da técnica de *SSDT hooking*, que será explicada com mais detalhes nas seções seguintes, em um sistema. Pode-se perceber como era o fluxo de execução antes do *hooking* e depois dele.

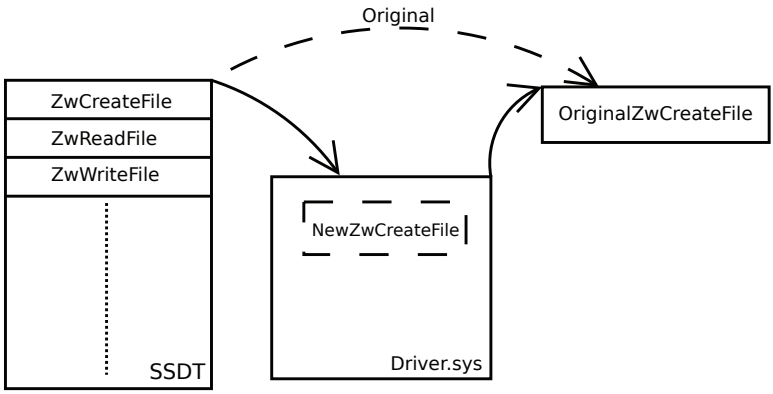

Figura 3.4. Ilustração da aplicação da técnica de *SSDT Hooking*.

Nas seções a seguir, serão detalhadas as diferenças existentes entre *hooking* de nível de usuário e de *kernel*. Serão citados também exemplos de cada uma das abordagens.

3.2.2.1. *Userland Hooking*

Userland hooking ou *hooking* de nível de usuário é uma técnica de interceptação que pode afetar somente programas que executam em nível de usuário, não podendo interferir em qualquer aplicação que opere em um nível mais privilegiado. Mesmo com esta limitação, tal técnica é bastante utilizada por *malware*, dado que sua implementação é mais simples. Embora sua utilização seja frequente, *userland hooking* pode ser facilmente detectado, o que pode levar o programa a desfazer o *hooking* ou não executar a função modificada. Como esse tipo de *hooking* é feito normalmente sobre APIs[9] disponibilizadas pelo sistema operacional, para um programa detectar se a interceptação está sendo feita ou não basta verificar se os endereços das APIs utilizadas por ele estão modificados ou se o endereço é uma instrução de pulo incondicional (JMP), que é uma prática comumente utilizada em *hooking*. Além deste tipo de detecção, existem outras formas que podem ser utilizadas para evitar um possível *hooking*. Uma delas, muito eficaz, é a utilização de funções nativas do sistema operacional, ao invés das APIs fornecidas por ele. Entretanto, empregar este processo requer um maior cuidado na implementação do programa, pois este deverá fornecer um número maior de informações quando for executar cada função, tarefa esta que antes ficava a cargo do sistema operacional. A fim de exemplificar técnicas de *userland hooking* para modificação de APIs, apresenta-se a seguir *IAT hooking*, *Detours* e *inline hooking*.

IAT Hooking

Import Address Table hooking, ou (*IAT hooking*), é uma técnica aplicada para interceptar as funções utilizadas por um determinado programa, antes que este esteja em execução. Para isso, a técnica deve ser empregada sem que o *malware* tenha comprometido o sistema. A *Import Address Table* é uma estrutura presente no cabeçalho de arquivos do tipo PE32 (arquivos executáveis do sistema *Windows*), e é responsável por indicar os endereços das rotinas externas utilizadas por um programa [Microsoft 2008]. Esses endereços são definidos no processo de carregamento do programa que antecede a sua execução, quando este está sendo inicializado pelo sistema operacional. Tais endereços são fornecidos por DLLs (*dynamic-link libraries*), pacotes binários que contêm funções e variáveis as quais podem ser utilizadas por outros programas [Russinovich and Solomon 2004]. Quando o programa está na fase de inicialização, o sistema operacional se encarrega de verificar quais DLLs e métodos externos são utilizadas por ele. De posse destas informações, o sistema operacional pode preencher a IAT com os endereços referentes aos métodos usados, de forma que durante a execução o programa carregado consiga invocar corretamente as funções [Holy Father 2004].

Para realizar este tipo de *hooking* é necessário modificar a tabela IAT do programa que se deseja monitorar, de forma que os endereços contidos nela sejam de funções que se tem controle. Portanto, para cada função modificada é necessária uma nova função, a qual poderá modificar ou simplesmente monitorar os dados passados pelo programa sob análise para a função original. Além disso, cada função deve invocar a função original para que as ações produzidas por um programa tenham seu efeito consumado no sistema, prosseguindo assim com a execução normal do programa monitorado. Um problema evi-

[9] Application Programming Interfaces

dente desta abordagem é a necessidade de modificações no programa sob análise, a fim de que seja possível instalar os *hookings* nas *APISs* monitoradas. A detecção deste tipo de ação pode ser feita com um simples teste de integridade no código do programa monitorado. Se isto ocorrer, um *malware* pode identificar que está sendo monitorado, o que pode fazê-lo tomar medidas que inviabilizem sua análise ou que os resultados retornados por esta não correspondam ao fluxo de execução malicioso pretendido originalmente. Outro problema com a abordagem que pode inutilizar a captura ocorre caso o *malware* carregue a DLL que irá utilizar durante a sua execução. Para isso, basta que ele use APIs disponibilizadas pelo *Windows*, as quais possibilitam que a DLL seja utilizada, mesmo sem ser previamente inicializada com o *malware*. Esta prática é bem simples de ser utilizada e desabilita por completo o *IAT hooking*, já que as APIs utilizadas assim pelo *malware* não serão afetadas.

Detours

Uma outra forma de interceptar *APIs* do *Windows* é através do uso de *Detours*, uma biblioteca provida pela própria *Microsoft* para interceptar funções do *Windows* em arquiteturas x86 [Hunt and Brubacher 1999]. Através de sua utilização, é possível realizar modificações no início da função que se deseja interceptar de maneira dinâmica, durante a execução do programa. Esta técnica é implementada através da inserção de uma instrução *assembly* de pulo incondicional (`JMP`) no início da função. Assim, quando tal função for invocada, o *JMP* será executado e irá direcionar o fluxo de execução para uma região sobre a qual se tem controle. Isto possibilita a captura dos dados que estão passando pela função, bem como os valores retornados com sua execução. Sua implementação pode ser feita de maneira simples, visto que o próprio sistema operacional provê suporte para isso, através de um programa que executa em nível de usuário.

Um sistema *open source* para análise dinâmica de *malware* disponibilizado recentemente, chamado *Cuckoobox* [Cukoo Sandbox 2011], aplica a técnica de *Detours* para obter as informações das APIs utilizadas pelo *malware* durante sua execução. Embora o sistema necessite de máquinas virtuais para realizar uma análise, o mecanismo de captura é inserido no *guest* e passa as informações obtidas via protocolo de comunicação para o *host*, diferentemente de *Anubis*, que requer uma modificação no *software* de emulação para capturar as informações de execução do *malware*. Isto torna a implementação da técnica mais simples, porém, seu custo é que pode-se facilmente verificar o uso de *Detours* através de uma checagem no início da função que se quer realizar: se a instrução inicial for um *JMP*, há a presença de um *Detour*.

Inline Hooking

Inline Hooking é uma outra técnica que pode ser utilizada para redirecionar o fluxo de execução normal de um programa para uma região que se tenha total controle. Em geral, esta técnica é utilizada em *malware* para alterar as APIs do sistema operacional, de forma que as respostas produzidas sejam capturadas ou modificadas. Cabe ressaltar que *inline hooking* pode ser implementada tanto no nível do usuário como no do *kernel*, entretanto, sua forma mais comum aparece em nível de usuário devido à simplicidade da implementação. Seu funcionamento é bem parecido com o do *Detours*, mas em vez de trocar somente a primeira instrução *assembly* da função que se deseja interceptar, é possível alterar uma porção maior de código. Como a parte que desvia a execução do

programa para a região de que se tem controle não fica no início do código, a detecção do *hooking* é mais complicada, visto que será necessário inspecionar uma área maior do código que se quer executar em busca de algum desvio de execução.

O sistema de análise de *malware CWSandBox* [CWSandbox 2011] utiliza a técnica de *inline hooking* para capturar as informações resultantes da execução de um *malware* em um sistema *Windows XP*. O *inline hooking* feito por ele é similar à técnica do *Windows Detour*, onde o começo da função que se deseja interceptar é substituído por um *JMP* [Willems et. al. 2007].

3.2.2.2. *Kernel Hooking*

O *kernel hooking*, ou *hooking* em nível de *kernel*, executa em um nível mais privilegiado, utilizando técnicas mais complexas que não são trivialmente detectadas por *malware*. Isto atribui uma vantagem sobre o *userland hooking*, pois torna a sua detecção mais difícil. Porém, na maioria das vezes a detecção de *kernel hooking* pode ser feita por programas que executam em nível privilegiado, como por exemplo os *rootkits* [Hoglund and Butler 2005]. No caso geral, a análise de *malware* cuja execução ocorre no nível de usuário é mais confiável caso se aplique a técnica de *kernel hooking*, pois o componente responsável pela captura das ações do *malware* está em um nível de privilégio mais elevado, cujo acesso direto não é permitido. Por outro lado, mesmo com a possibilidade de subversão do *hooking*, os *rootkits* (e programas de nível de *kernel* em geral) precisam ser inicializados por programas de nível de usuário. Portanto, todas as ações executadas durante o processo de inicialização, como por exemplo, o carregamento de um *driver*, são capturadas, podendo ao menos levantar suspeitas sobre um possível comportamento malicioso. Um exemplo de *kernel hooking* comumente utilizado por mecanismos de segurança, como os antivírus, e também por *rootkits* é a técnica de *SSDT hooking*, explicada a seguir. Outro exemplo, explicado adiante, é a técnica de *kernel callbacks*.

SSDT Hooking

O *hooking* da *System Service Dispatch Table*, comumente conhecido como *SSDT hooking*, consiste na modificação de uma estrutura interna presente em sistemas *Windows*, a qual é responsável por armazenar os endereços das *syscalls* do sistema. Esta estrutura é composta basicamente por um vetor de endereços, onde cada índice corresponde a uma das rotinas de chamadas de sistema disponibilizadas pelo sistema operacional, representando uma tabela. Tal tabela reside no *kernel* do sistema operacional e, portanto, programas no nível de usuário não têm acesso a ela. Esta tabela é utilizada pelo sistema operacional quando uma chamada de sistema é requisitada, retornando assim o endereço de memória da função apropriada [Blunden 2009]. Para realizar o *hooking* de SSDT é preciso utilizar um *driver* que opere em nível de *kernel*. Como esse *driver* executa em modo privilegiado, ele pode realizar alterações em outros programas e estruturas internas do sistema presentes no nível de *kernel*. Portanto, o *driver* tem permissão para alterar os endereços contidos na SSDT, trocando-os por valores que indiquem métodos de seu controle. Antes de realizar a troca, os endereços originais precisam ser armazenados para que possam ser utilizados posteriormente, completando assim a requisição feita originalmente. Os endereços alterados irão apontar para funções interceptadas, que serão

executadas ao invés das originais. Como se tem o controle destas funções, é possível monitorar as requisições feitas ao sistema e os valores retornados, ou modificar as respostas retornadas pelo sistema operacional ao programa sob análise. Além disso, fica a cargo das funções controladas realizar a chamada às *syscalls* originais, através dos endereços salvos antes das modificações na SSDT, possibilitando que o fluxo de execução original de um *malware* monitorado seja mantido.

Técnicas de *kernel hooking* são muito mais poderosas do que as de *userland* justamente por atuarem em um nível privilegiado, o que lhes permite maior controle sobre os demais programas que executam no sistema operacional. Outra vantagem é que o monitoramento fica transparente para as aplicações de nível de usuário, pois elas não têm acesso às aplicações que executam no nível do *kernel*. Uma desvantagem deste tipo de técnica diz respeito às informações que são extraídas das funções que se pode interceptar. Na abordagem de IAT *hooking*, é possível capturar funções contidas nas *DLLs* utilizadas por um programa, as quais correspondem muitas vezes ao modo utilizado pelo programador para executar a chamada de sistema. Já no caso das chamadas de sistema capturadas através do SSDT *hooking*, a informação não se apresenta de uma forma tão clara. Por exemplo, caso se queira obter o nome de um arquivo utilizado durante uma *syscall*, pode ser necessário realizar a verificação do conteúdo de outras estruturas utilizadas na invocação da *syscall*, dado que este nome muitas vezes não é passado de forma direta. Neste caso é necessário realizar alguns procedimentos para "traduzir" os dados passados como argumento para a *syscall* de forma a encontrar a informação desejada. Além deste problema, o SSDT *hooking* é dependente da versão do sistema onde ele está sendo aplicado, sendo que para cada versão o SSDT *hooking* deve ser feito de uma maneira diferente. Tal efeito acontece pois podem ocorrer modificações na SSDT entre as versões, fazendo com que endereços antes utilizados para identificar uma *syscall* não sejam os mesmos.

Um sistema de análise de *malware* que faz uso desta técnica é o *JoeBox* [Joebox 2011]. Além de interceptar as chamadas da SSDT ele também realiza um *hooking* de nível de usuário, o que possibilita obter um volume bem maior de informação. Infelizmente só é possível submeter um número limitado de *malware* a este sistema, dado que se trata de um sistema comercial.

3.2.3. *Kernel Callbacks* e *Filter Driver*

Callbacks são funções disponibilizadas pelo sistema operacional que notificam uma aplicação sobre determinadas modificações no sistema, como por exemplo, a criação de uma chave de registro ou de um novo arquivo [Seifert et. al. 2007]. Estas funções são bem documentadas e, portanto, sua implementação não apresenta incompatibilidades entre as diferentes versões do *Windows*, como ocorre no caso do *SSDT Hooking*, que realiza modificações em estruturas do *kernel* específico de cada versão do sistema operacional. Isto torna possível a monitoração de determinadas ações que ocorrem no sistema de uma forma mais simples e genérica, permitindo a identificação de comportamento possivelmente malicioso. Uma limitação presente nesta abordagem é que ela permite somente a captura das ações realizadas por funções disponibilizadas pelo sistema operacional, o que pode levar à obtenção de um comportamento de execução incompleto.

Uma tentativa de melhorar a captura de dados com o uso de *kernl callbacks* é atra-

vés do uso de *filter drivers*. Seu funcionamento é similar ao de um filtro, interceptando todas as requisições feitas a um determinado dispositivo do sistema. Eles se interpõem entre o *driver*, o qual estão sendo interceptadas as requisições, e o nível de usuário, tendo acesso assim a todas as chamadas feitas ao dispositivo interceptado. Seu uso em conjunto com *kernel callbacks* possibilitam que seja capturado um número maior de informações porém, elas ainda são restritas a um limitado conjunto, o qual pode resultar em uma análise incompleta ou inconclusiva. Uma ferramenta que implementa a técnica de *kernel callbacks* e *filter driver* é o CaptureBat [CaptureBat 2011]. Sua utilização requer o carregamento de *drivers* no sistema operacional a ser monitorado, o que possibilita sua aplicação tanto em máquinas virtuais como reais.

3.2.4. *Debugging*

Utilizada inicialmente para fins não maliciosos, a técnica de *debugging* consiste em para execução de um programa em um dado momento, sendo possível verificar o que será executado a seguir. Normalmente ela é utilizada por desenvolvedores de *software*, que desejam encontrar em qual ponto sua aplicação está apresentando problemas. Para isto, basta colocar um marcador em determinados pontos, chamados de *breakpoints*, os quais irão interromper o fluxo de execução do programa no aguardo de um comando sobre o que fazer a seguir. *Debuggers*, programas que executa o processo de *debugging* de uma aplicação, são separados em duas categorias, os que utilizam os recursos providos pelo processador, mais comuns, e os que emulam o processador, sendo estes os mais poderosos pois controlam toda a execução do programa. Se o emulador for bem feito, será muito difícil para um programa descobrir que está sofrendo *debugging*, dado que o *debugger* estará no controle da execução.

Os *breakpoints* podem ser divididos em dois grupos, os *software breakpoints*, que modificam o código da aplicação onde se deseja realizar o *debugging*, e os *hardware breakpoints*, que utilizam recursos do processador, sendo os mais difíceis de serem identificados. Como os *software breakpoints* modificam a aplicação, basta um simples teste de intergridade para identificar a presença de um *breakpoint* no código. Já os *hardware breakpoints* não sofrem deste problema, dado que as modificações feitas para realizar o *debugging* estão em estruturas disponibilizadas pelo processador. Mesmo que o programa tente identificar estas mudanças, o *debugger* pode interceptar estas requisições e retornar uma resposta falsa, não revelando assim que o programa está sofrendo um *debugging*.

Na análise dinâmica de *malware* a técnica de *debugging* pode ser utilizada para identificar características do *malware*. Com isto será possível desenvolver métodos que possam identificar novos *malware* com características semelhantes, facilitando a identificação destes exemplares e evitando que novas contaminações ocorram. Por obter dados de mais baixo nível, composto pelo código *assembly* da aplicação, é preciso realizar algumas inferências para obter informações de mais alto nível, como por exemplo um nome de arquivo. Algumas aplicações possibilitam que o *debugger* tenha acesso direto à essa informação, sendo possível identificar o momento em que a aplicação faz referência a ela.

Duas ferramentas de *debugger* bem conhecidas e utilizadas constantemente, uma paga e outra gratuita, são o *IDA Pro* e o *OllyDBG* respectivamente. A primeira dispõe de mais recursos que podem auxiliar durante o processo de *debugging*, como por exemplo

um gráfico que mostra todas as chamadas de funções identificadas no programa. Com ele é possível ver a relação existente entre as funções presentes no programa. Já a segunda, *OllyDBG*, mesmo sendo uma ferramenta gratuita dispõe de recursos indispensáveis durante o processo de *debugging*, como por exemplo uso de *plugins* que podem ajudar no processo de análise.

3.2.5. Engenharia Reversa

O processo de engenharia reversa consiste em extrair informações sobre um *software*, de forma que seja possível compreender seu funcionamento. Não existe uma ferramenta que realize de forma automática este processo, sendo somente possível realizá-lo de forma manual. Normalmente é aplicado em *software* onde não é possível se ter acesso ao código fonte, possibilitando assim um maior entendimento sobre as ações executadas pelo *software* durante sua execução, auxiliando inclusive na remontagem do código do programa.

Na análise de *malware*, esta técnica é utilizada quando se deseja descobrir, de forma detalhada, quais os passos desenvolvidos pelo *malware* no sistema. Isso possibilita, por exemplo, identificar quais técnicas ele utilizou para comprometer o sistema, como ele esconder suas atividades para ocultar sua execução, quais dados do sistema comprometido ele captura, dentre outras atividades normalmente executadas por *malware*. A partir dessas informações, é possível inclusive criar procedimentos que retirem da máquina comprometida todas as modificações feitas pelo *malware*, levando-a a um estado integro, anterior ao comprometimento. Outra aplicação para a engenharia reversa de *malware* é o reconhecimento de rotinas de cifração presentes nele. Elas normalmente são aplicadas ao *malware* para evitar que certos dados fiquem em claro no código binário, como por exemplo um endereço utilizado pelo *malware* onde está outro componente utilizado por ele, o qual será obtido durante sua execução, ou um endereço de *email* para onde serão enviados os dados capturados na máquina comprometida.

Para que uma engenharia reversa seja realizada, é preciso um bom nível de conhecimento sobre o ambiente onde a aplicação irá executar, para compreender as iterações realizadas entre o programa e o sistema operacional bem como as respostas retornadas em cada uma delas. Com a mescla de técnicas que combinam dados de baixo e alto nível, é possível obter informações necessárias para realizar uma engenharia reversa de forma fácil e rápida. Portanto, todas as técnicas de análise dinâmica apresentadas até aqui podem ser aplicadas durante o processo de engenharia reversa, evitando que seja desperdiçado tempo na busca de informações que são providas naturalmente por elas.

3.3. Sistemas de Análise Dinâmica

As técnicas apresentadas no capitulo anterior são utilizadas em um grande número de sistemas de análise de *malware* e *Web malware*. Normalmente, os sistemas apresentam os dados coletados de modo que seja fácil compreender as ações executadas pela amostra durante a análise. Alguns sistemas aceitam somente submissão através de seus *Web sites* enquanto outros só podem ser executados se instalados em uma máquina local. Neste capitulo serão discutidos em cada subseção sistemas de análise relevantes, frequentemente utilizados na análise de *malware* e *Web malware*.

3.3.1. Sistemas de análise de *malware*

3.3.1.1. *Anubis*

Iniciou suas atividades em 2007, baseado em um trabalho de mestrado, *TTAnalyze* [Bayer et. al. 2006]. Utiliza a técnica de *Virtual Machine Introspection* (VMI) aplicada ao *Qemu*, explicada detalhadamente no capitulo anterior, para capturar as ações executadas pelo *malware* no sistema de análise. O sistema operacional utilizado no ambiente de análise é um *Windows XP* SP3, com uma instalação básica. Ao final da análise, que pode demorar 8 minutos, é produzido um relatório em vários formatos, *html*, *xml* e txt, que sumariza as ações do *malware*. Além disto, caso seja gerado algum tráfego de rede, um arquivo no formato pcap também é fornecido. A submissão pode ser feita em lote, através de um *script python*, ou de forma singular, onde somente um *malware* é enviado. No caso desta submissão, é possível torna-la mais rápida, passando-a à frente das submissões em lote. Para isto, basta que um *captcha* seja fornecido no momento da submissão.

Apesar do uso de VMI para capturar as ações desenvolvidas pelo *malware*, o *Anubis* têm dois componentes no sistema de análise, um *driver* e um programa de nível de usuário, os quais irão capturar os valores contidos no registrador *CR3*, usados pelos processos do *malware*, e realizar a comunicação entre o ambiente de análise e a máquina *host* respectivamente. A tabela 3.2 mostra as vantagens e desvantagens presentes no sistema *Anubis*, levado em consideração todos os pontos levantados até aqui.

Tabela 3.2. Vantagens e desvantagens do sistema de análise *Anubis*.

Vantagens	Desvantagens
A captura é feita por um componente fora do ambiente de análise.	Existem componentes do sistema de análise dentro do ambiente de análise.
Não é necessário nenhum preparo local para executar a análise.	Caso o sistema esteja fora do ar, a análise não pode ser realizada.

3.3.1.2. *CWSandbox*

Teve início em 2007, utilizando a técnica de *userland hooking* para obter as informações geradas pelo *malware*, em máquinas virtuais. Os ambientes de análise têm como sistema operacional um *Windows XP* [Willems et. al. 2007]. A captura das informações é realizada por uma *DLL (Dynamic Link Library)*, que precisa ser injetada no processo do *malware*. Quando ela é carregada, as principais funções utilizadas para fazer a interface entre o programa e o sistema de análise, como por exemplo modificações em arquivos, têm seu início modificado, de forma que um desvio incondicional seja executado assim que a função é chamada. Finalizada a análise, é gerado um relatório nos formatos *html*, *xml* e txt, contendo as ações realizadas no ambiente pelo *malware*. O processo de submissão pode ser feito de forma singular, para um único arquivo, ou na forma de um arquivo comprimido, contendo múltiplos arquivos para análise. É necessário fornecer um endereço de *email* para onde será enviado o *link* para o resultado da análise, quando este estiver pronto.

Para iniciar o processo de análise, existe um componente dentro do ambiente de análise, o cwsandbox.exe, que irá começar o processo do *malware* em estado suspenso, injetar a *DLL* e retomar a execução do processo. Além disto, este componente será informado caso o *malware* inicialize ou modifique algum processo, para que a *DLL* seja injetada neles também. Existe um outro componente dentro do ambiente de anpalise, que é responsável pela proteção dos componentes de captura, escondendo evidências de sua existência. Na tabela 3.3 é possível ver as vantagens e desvantagens presentes no sistema *CWSandbox*.

Tabela 3.3. Vantagens e desvantagens do sistema de análise *CWSandbox*.

Vantagens	Desvantagens
Os dados capturados são de funções de mais alto nível.	Caso o sistema esteja fora do ar, a análise não pode ser realizada.
Não é necessário nenhum preparo local para executar a análise.	Existem componentes do sistema de análise dentro do ambiente de análise.
Existem componentes que dificultam a detecção do sistema	

3.3.1.3. *Cuckoobox*

Começou como um projeto do *Google Summer of Code* de 2010 ligado à *The Honeynet Project* sendo disponibilizado somente em 2011, tendo seu código totalmente disponível para *download*. Em 2011, foi novamente selecionado para o *Google Summer of Code*, onde foram feitas modificações na técnica de captura de informações e criado um novo componente que esconde os elementos do *Cuckoobox* que estão no ambiente de análise. Utiliza a técnica de *inline hooking* para interceptar as chamadas de sistema executadas pelo *malware*. Não dispõe de uma interface de submissão, sendo que para realizar uma análise é preciso preparar o ambiente antes, instalando localmente todos os requisitos necessários. Não apresenta restrições quanto a versão do sistema que pode ser utilizado no ambiente de análise, podendo ser qualquer versão do *Windows* a partir do XP. Depois de instalado o ambiente e todas as dependências do *Cuckoobox* é preciso inicializar um *script* em *python*, o qual irá carregar todas as configurações necessárias para realizar uma nova análise. Após isto é possível então invocar um outro *script* em *python* que irá enviar o *malware* para análise. Ao final dela, estarão disponíveis em uma pasta os traços de execução criados pelos processos gerados durante a análise, os *snapshots* de telas geradas, arquivos modificados/criados/deletados e o tráfego de rede.

Para implementar o *inline hooking* o *Cuckoobox* precisa carregar uma DLL no processo que deseja monitorar. O *hooking* é implementado de forma diferente em cada função interceptada diferentemente de um simples salto incondicional no início da função. Isto dificulta métodos triviais de detecção, evitando assim que a análise não seja bem sucedida. Atualmente, existem dois métodos implementados, os quais são escolhidos de forma aleatória no momento que o *inline hooking* é instalado. Esta *DLL* é carregada por um *script* em *python* que fica em execução no ambiente de análise durante todo o processo. Além desta tarefa, ele também é notificado caso o *malware* modifique ou crie

um novo processo durante a análise, indicando ao *script* que a DLL de monitoração deve ser carregada nestes outros processos. A tabela 3.4 explicita as vantagens e desvantagens presentes no sistema *Cuckoobox*.

Tabela 3.4. Vantagens e desvantagens do sistema de análise *Cuckoobox*.

Vantagens	Desvantagens
Os dados capturados são de funções de mais alto nível.	Existem componentes do sistema de análise dentro do ambiente de análise.
O código está disponível e é possível customizar o sistema de acordo com suas necessidades.	É necessário preparar o ambiente para executar a análise.
Existem componentes que dificultam a detecção do sistema	

3.3.1.4. *Ether*

Foi apresentado em 2008 e teve seu código disponibilizado em 2009. Utiliza *Virtual Machine Introspection* (VMI), aplicada ao *Xen*, um *hypervisor open source* bem conhecido, para capturar as informações geradas. Para realizar uma análise é preciso instalá-lo localmente, em uma máquina com processador com suporte *Intel VT*, um conjunto de instruções que facilita a virtualização de instruções x86. Terminada a instalação, é necessário preparar o ambiente de análise, instalando um *Windows XP* com SP2 sem nenhum programa adicional. O controle da análise é feito inteiramente por um componente fora do ambiente de análise, o que evita possíveis detecções do sistema. Sempre que for realizar uma nova análise é preciso utilizar este componente, o qual irá enviar o *malware* para o ambiente de análise e executá-lo. Infelizmente, a captura das ações realizadas se restringe a todos os processos do sistema ou somente ao processo do *malware*. Portanto, caso o *malware* crie ou modifique outros processos, o *Ether* não será capaz de restringir a captura somente à estas ações. A tabela 3.5 mostra as vantagens e desvantagens presentes no sistema *Ether*, baseados nos pontos levantados até aqui.

Tabela 3.5. Vantagens e desvantagens do sistema de análise *Ether*.

Vantagens	Desvantagens
A captura é feita por um componente fora do ambiente de análise.	Só faz a monitoração de um processo por vez.
O código está disponível e é possível customizar o sistema de acordo com suas necessidades.	É necessário preparar o ambiente para executar a análise.
Não têm componentes dentro do sistema de análise	É necessário indicar o nome/endereço do processo que se deseja monitorar.

3.3.1.5. *Joebox*

Iniciou suas atividades em 2007, disponibilizando de forma gratuita análises de *malware*. No ano de 2011, tornou-se um serviço pago, aceitando submissões somente mediante pagamento de uma taxa mensal. Utiliza técnicas de *hooking* para obter as ações executadas no ambiente de análise. Fornece relatórios em formato *html* e *xml* junto com os arquivos criados, capturas de tela, tráfego de rede e *dumps* de memória gerados durante a análise. Não há restrições quanto ao sistema de análise que deve ser utilizado, podendo ser qualquer versão *Windows*, acima do XP. Além disto, ele pode ser executado em diversas plataformas como máquinas virtuais, emuladas e reais.

Combinando várias técnicas de *hooking*, tanto de *userland* quanto de *kernel*, o *Joebox* é capaz de obter informações mais detalhadas, capturando dados tanto de baixo quanto de alto nível. Existe um componente interno no sistema de análise, joesandboxcontrol.exe, que contém várias funcionalidades, dentre elas a inicialização do *malware*. Na tabela 3.6 é possível verificar as vantagens e desvantagens presentes no sistema.

Tabela 3.6. Vantagens e desvantagens do sistema de análise *Joebox*.

Vantagens	Desvantagens
Pode ser executado em diversas arquiteturas.	Não está disponível gratuitamente.
Implementa várias técnicas de captura diferentes.	Existem componentes do sistema de análise dentro do ambiente de análise.

3.3.2. Sistemas de análise de *Web malware*

3.3.2.1. JSand

JSand é um sistema de análise de *Web malware*, de baixa interatividade, que foi apresentado em [Cova et. al. 2010] e pode ser usado através da interface pública de submissão *online*[10]. Sua principal função é analisar o código JavaScript presente na página, provendo informações a respeito de sua execução e informando se a página analisada é benigna, suspeita ou maliciosa. Essa identificação é feita por meio da detecção de anomalia no comportamento do código JavaScript. Para emular a página a ser analisada, o JSand utiliza uma versão modificada do HtmlUnit, uma plataforma em Java para testes de aplicações *Web*. Como os *Web malware* atualmente atacam vulnerabilidades de diversas aplicações, o sistema emula todo objeto ActiveX requisitado pelo código, de forma que uma verificação pela presença dele retornará positiva e o código continuará sua execução.

Para realizar a detecção por anomalia, o sistema JSand extrai dez atributos a partir da análise realizada. Esses atributos representam características de redirecionamento, ofuscação, preparação do ambiente para o abuso de vulnerabilidades e o processo de abuso. O treinamento e detecção são realizados com o uso da ferramenta libAnomaly[11], desenvolvida pelo mesmo grupo que desenvolveu JSand.

[10] http://wepawet.cs.ucsb.edu

[11] http://www.cs.ucsb.edu/~seclab/projects/libanomaly/

As informações disponibilizadas após a análise dizem respeito ao comportamento do código JavaScript presente na página e incluem trechos de código desofuscados, vulnerabilidades das quais o código tenta abusar, *shellcodes* utilizados, objetos ActiveX utilizados, *links* para o sistema Anubis com a análise de arquivos executáveis que o código tenha tentado obter e requisições HTTP. A tabela 3.7 apresenta as vantagens e desvantagens da abordagem usada por esse sistema.

Tabela 3.7. Vantagens e desvantagens do sistema de análise *JSand*.

Vantagens	Desvantagens
Por ser emulada, a análise é executada rapidamente.	O ambiente emulado falha ao analisar certos códigos.
Pode detectar ataques a vulnerabilidades desconhecidas.	Pode detectar apenas ataques que utilizam JavaScript.

3.3.2.2. PhoneyC

O sistema PhoneyC, apresentado em [Nazario 2009], é um *honeyclient* de baixa interatividade que também utiliza um emulador para processar as páginas analisadas e possui seu código disponível para *download*. Ele é capaz de analisar códigos JavaScript e Visual Basic Script. A emulação do ambiente de JavaScript é feita com o uso do interpretador da Mozilla Foundation, SpiderMonkey, que faz parte do navegador *Web* Firefox. Já o código Visual Basic Script é primeiramente transformado em um código equivalente em Python, pela ferramenta vb2py[12], e então processado pelo próprio interpretador da linguagem.

Para detectar tentativas de abuso de vulnerabilidades o sistema emula certos componentes vulneráveis. PhoneyC prove informações sobre objetos ActiveX utilizados, vulnerabilidades que o código tenta atacar e *shellcodes* utilizados. A tabela 3.8 apresenta as vantagens e desvantagens desse sistema.

Tabela 3.8. Vantagens e desvantagens do sistema de análise *PhoneyC*.

Vantagens	Desvantagens
Por ser emulada, a análise é executada rapidamente.	O ambiente emulado falha ao analisar certos códigos.
Além de JavaScript, detecta ataques que utilizam Visual Basic Script	Não pode detectar ataques a vulnerabilidades desconhecidas.

3.3.2.3. Capture-HPC

O sistema Capture-HPC, descrito com mais detalhes em [Seifert and Steenson 2006], é um *honeyclient* de alta interatividade que possui seu código disponível. As páginas a serem analisadas são processadas dentro de um ambiente virtualizado, utilizando um navegador *Web* completo e um *driver* de *kernel* que captura as chamadas de sistemas realizadas

[12]http://vb2py.sourceforge.net

pelo navegador. Caso essas chamadas de sistema sejam consideradas anômalas, a página analisada é classificada como maliciosa.

Após a análise o sistema informa as chamadas de sistema que foram identificadas como anômalas. As vantagens e desvantagens desse sistema podem ser vistas na tabela 3.9.

Tabela 3.9. Vantagens e desvantagens do sistema de análise *Capture-HPC*.

Vantagens	Desvantagens
Utiliza um navegador *Web* completo para processar as páginas	Devido ao uso de ambiente virtualizado, a análise é mais demorada.
Pode detectar ataques a vulnerabilidades desconhecidas.	Pode detectar apenas ataques bem sucedidos, que resultem em chamadas de sistema anômalas.
Pode detectar ataques independentemente da linguagem utilizada.	

3.4. Caixa de Ferramentas

3.4.1. Informações Gerais

Neste capítulo será apresentado um conjunto de ferramentas gratuitas e disponíveis na Internet que são capazes de criar um ambiente de análise dinâmica de *malware*. Além disso, serão apresentadas ferramentas para análises mais aprofundadas, *debugging* e forense computacional. O ambiente de análise a ser implementado neste capítulo será denominado de Protótipo Sandbox. Como base para sua construção, será usada uma máquina virtual para agilizar a restauração do ambiente. É importante deixar claro algumas limitações do sistema, visto que em sua concepção foi dado enfoque à praticidade na montagem e no uso. Dessa forma o sistema possuirá pontos vulneráveis, que serão destacados no decorrer do texto. As ressalvas são a necessidade de um analista para dar confiabilidade aos relatórios providos e a falta de proteção contra técnicas de detecção de ambientes virtuais.

3.4.2. Modelo proposto para o sistema de análise

É importante identificar e organizar a ordem com que as ferramentas deverão ser executadas. Dessa forma, pode-se tornar a análise mais eficaz. Na Figura 3.5 pode-se observar que o conhecimento sobre o funcionamento das ferramentas é fundamental para uma boa análise, pois a ordem em que os comandos foram executados altera diretamente a qualidade das informações obtidas. No trecho 1, caso o *malware* contenha algum *packer* que ofusque o código, a extração das *strings* dificilmente trará informações úteis, enquanto no trecho 2, verifica-se se foi utilizado o *packer* UPX. Caso este tenha sido utilizado, é removido do *malware*[13], assim a ferramenta *strings* conseguirá capturar dados relevantes que estiverem no código.

O Protótipo Sandbox possui dois ambientes,o primeiro é o hospedeiro, no qual haverá um sistema operacional base, para processar as informações obtidas da análise dinâmica. Este ambiente é a área de trabalho principal do analista e não deve ser conta-

[13]A mesma ferramenta que empacota um binário com o *packer* UPX pode ser usada para removê-lo.

```
/* Trecho 1: ineficiente para extração de dados */
relatorio.escrever(strings(malware));
relatorio.escrever(packer(malware));

/* Trecho 2: eficiente para extração de dados */
nomePacker = packer(malware);
if(nomePacker == "upx")
     malware = unpacking(malware, nomePacker);
relatorio.escrever(strings(malware));
relatorio.escrever(nomePacker);
```

Figura 3.5. Pseudo-código que demonstra a importância do uso das ferramentas na ordem correta

minada pelo *malware*, ficando responsável apenas pela análise estática e por hospedar as máquinas virtuais.

O segundo ambiente é o virtualizado. Nele os exemplares de *malware* serão executados para que seu comportamento seja capturado. Portanto, esses ambientes deverão ser preparados para estimular e monitorar as atividades maliciosas e, posteriormente, enviar as informações coletadas para a máquina hospedeira. É importante que a máquina virtual não esteja na mesma rede que outros sistemas, pois pode haver contaminação destes. A arquitetura projetada para o Protótipo Sandbox é ilustrada na Figura 3.6.

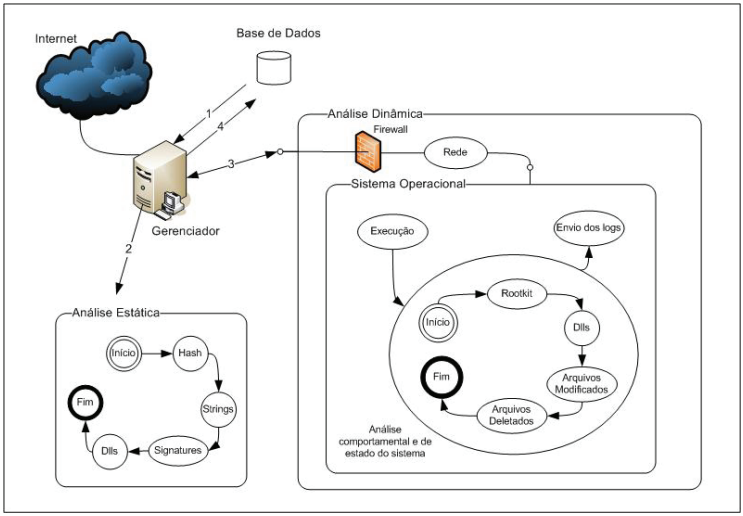

Figura 3.6. Arquitetura do Protótipo Sandbox

Na Figura 3.6, observa-se que o ambiente base (Gerenciador) opera como uma camada intermediária entre a Internet e o ambiente de análise (Análise Dinâmica) virtu-

alizado. Isto torna possível restringir o acesso da máquina virtual através de um *firewall* entre os ambientes, além de bloquear tentativas de interação com a máquina real. Outra responsabilidade importante do Gerenciador é o armazenamento dos relatórios em uma base de dados, podendo ser feito de forma simples, como um diretório, ou algo mais complexo como um banco de dados.

3.5. Máquina base

Para os testes com o Protótipo Sandbox foi utilizado um computador com processador Intel Core 2 Duo de 1.6 GHz, 2GB de memória RAM e disco rígido de 160 GB. Outras configurações que sejam equivalentes ou superiores também são válidas. A sugestão é apenas uma estimativa do mínimo que o computador precisa ter para realizar análises sequenciais sem *overheads* prejudiciais. Esta máquina é a que será usada para armazenar os resultados dos relatórios, efetuar a análise estática, hospedar a máquina virtual e gerenciar a análise dinâmica. O sistema operacional escolhido para a máquina base foi um Ubuntu linux com sua instalação padrão.

3.5.1. Base de resultados

A base de resultados do Protótipo Sandbox terá como finalidade armazenar *malware* e todos os arquivos produzidos durante as análises, inclusive o relatório final. Para isto, será construída em um diretório normal do sistema, terá como chaves primárias o hash sha256 do *malware* e será gerenciada através de shell script.

A base de resultados contém diversos diretórios, cada um com a coletânea de arquivos relacionados ao *malware*, tais como o arquivo executável, o relatório e os arquivos auxiliares produzidos nas análises. A organização da base de resultados pode ser melhor visualizada na Figura 3.7.

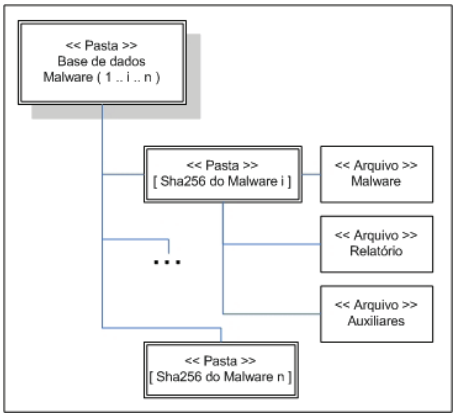

Figura 3.7. Estrutura de pastas para armazenamento dos arquivos produzidos pelas análises

Existem algumas operações que são fundamentais para gerenciar a base de dados a

ser construída, como a busca, a exclusão e a verificação se existe uma análise. Estas ações podem ser implementadas com comandos de *shell script*, como o exemplo mostrado na Figura 3.8.

```
# Configuração dos parâmetros
sha256=`sha256sum ${malware} | cut -d "" -f 1`

# Verificando se análise existe
if [ -d ${baseDados}/${sha256} ]
then
     echo "Malware já foi analisado!"
     nautilus ${baseDados}/${sha256} &
else
     echo "Malware ainda não analisado!";
     rodarAnalise
fi
```

Figura 3.8. Comandos de shell script usados para verificar se um *malware* já foi analisado

3.5.2. Análise estática simplificada

A análise estática tem como objetivo coletar informações presentes no *malware* sem ter que executá-lo. No Protótipo Sandbox as informações foram divididas em quatro grupos que podem ser visualizados na Figura 3.9.

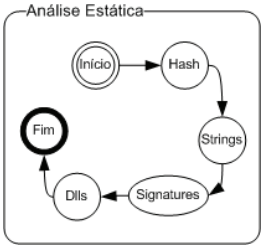

Figura 3.9. Informações obtidas na análise estática

Após o início da análise estática, a primeira informação buscada é uma chave que identifique univocamente o *malware*, a opção escolhida foi a função de *hash* criptográfico `sha256`, todavia poderia ser qualquer outra, como `md5` (mais comum) ou *sha512*.

Posteriormente, é utilizado o programa `strings` para buscar cadeias de caracteres relevantes, com pelo menos 3 *bytes*, no código. Assim, recomenda-se o uso de filtros com expressões regulares e a remoção prévia do *packer*, como mostrado na Seção 3.4.2. Mesmo que o *packer* não seja removido, informações sobre as bibliotecas utilizadas estarão presentes e serão úteis para indicar as ações do *malware* e caminhos para um processo de engenharia reversa, como pode ser visto na Figura 3.10.

Algumas identificações do *malware* podem ser coletadas com o uso de programas antivírus, identificadores de *packer* e identificadores de bibliotecas. No Protótipo

Exemplo de strings de malware com Armadillo v1.71
RegSetValueExA, RegCloseKey, RegOpenKeyExA, CryptGetHashParam,
CryptDestroyHash, CryptReleaseContext, CryptHashData,
CryptCreateHash, CryptAcquireContextA, AdjustTokenPrivileges,
LookupPrivilegeValueA, OpenProcessToken, RegCreateKeyA,
RegQueryValueExA, GetStartupInfoA, system32, SeDebugPrivilege, ...

Figura 3.10. Exemplo de *strings* encontradas em um *malware* com o *packer Armadillo*

Sandbox foram utilizados apenas identificadores de *packer* e de bibliotecas, através dos programas `pefile` e `sigcheck`. É interessante saber o *packer* utilizado para tentar removê-lo, e também saber se a análise dinâmica poderá ser feita, visto que certos *packers*, como o tElock e o Themida, são capazes de identificar ambientes emulados. O programa *sigcheck* é importante para verificar se as bibliotecas nativas utilizadas são legítimas.

Caso alguma biblioteca seja falsa, pode-se extrair informações usando os programas *objdump* e *Dependency Walker*, que revelaram dados do cabeçalho, seções e funções da biblioteca. Para a análise estática automatizada é preferível o programa `objdump`, por permitir seu uso através de linha de comando.

3.5.3. Máquina virtual

A máquina virtual tem como finalidade prover um ambiente de análise controlado para o *malware* ser executado, isolando a máquina base, além de possibilitar a rápida restauração do ambiente após o comprometimento. Para o Protótipo Sandbox foi escolhido o programa Virtualbox como tecnologia de virtualização.

3.5.4. Análise dinâmica

Na análise dinâmica é preparado um ambiente para monitorar todas as atividades do *malware* que será executado, obtendo assim seu comportamento e a interação dele com o sistema. No caso do Protótipo Sandbox as ações do *malware* foram divididas em rede, arquivos modificados e excluídos, registros modificados e excluídos, *rootkits*, processos criados e finalizados e DLLs. O processo de análise dinâmica pode ser visto na Figura 3.11.

3.5.4.1. Preparação do ambiente virtual

O ambiente virtualizado foi configurado com um disco de expansão dinâmica e 256 MB de memória RAM, e foi instalado com a configuração padrão do Windows XP Service Pack 3.

Para iniciar a máquina virtual é utilizado um conjunto de *scripts*, tanto na máquina virtual como na máquina base. A máquina base fica encarregada por iniciar a máquina virtual e depois desligar forçadamente a mesma, após o tempo limite de análise. A Figura 3.12 mostra os comandos utilizados para iniciar e finalizar a máquina virtual.

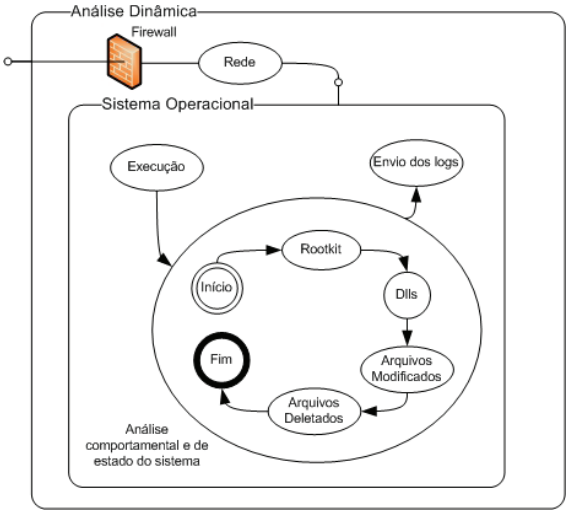

Figura 3.11. Processo de análise dinâmica

```
# Iniciar a máquina virtual
VBoxManage -q startvm ${MACHINE}

# Desligar a máquina virtual
VBoxManage -q controlvm ${MACHINE} poweroff

# Desligar forçadamente a máquina virtual
killall VirtualBox
killall VBoxXPCOMIPCD
```

Figura 3.12. Comandos para inicialização e término de máquinas virtuais

A máquina virtual é iniciada a partir de um *snapshot*[14], no qual o sistema está completamente inicializado e com um *script* do tipo WSH (Windows Script Host) sendo executado. Esse *script* tem a função de obter o *malware* através de FTP, iniciar/executar as ferramentas de análise e executar o *malware*, como observado na figura 3.13.

3.5.4.2. Monitoração do sistema

Para a monitoração do sistema é utilizada a ferramenta `Capture-BAT` [CaptureBat 2011], que é capaz de monitorar certas operações feitas pelo *malware*. Essas operações

[14] *Snapshots* são arquivos que guardam o estado do sistema. Com ele é possível iniciar o sistema virtualizado a partir de um estado previamente salvo

```
' Jogando os arquivos da Real para a Virtual
WshShell.Run "ftp -s:C:\CaixaCBT\ftp_m.txt <IP>",1,true
WScript.Sleep 2000

' Ligar CaptureBAT
WshShell.Run "C:\CaixaCBT\Capture\CaptureBAT.exe -cn -l
log_CBT.txt",2,false
WScript.Sleep 2000

' Ligar Malware
WshShell.Run "C:\CaixaCBT\Malware\malware.exe",2,false
WScript.Sleep 2000

' Deixar o Malware Rodando
For IntLoop = 1 To 1
    WScript.Sleep 60000
Next
```

Figura 3.13. *Script WSH* para inicialização da ferramenta de análise e execução do *malware* no ambiente virtualizado

incluem criação, modificação e remoção de arquivos e registros, além da inicialização e término de processos. São usados filtros para monitorar apenas as ações do *malware* e de processos iniciados por ele.

3.5.4.3. Tráfego de rede

O tráfego de rede pode ser monitorado de duas formas, internamente à máquina virtual, através do programa `Capture-BAT`, ou na máquina base, com o programa `tcpdump`, utilizando filtros. Por meio do tráfego de rede é possível descobrir servidores de distribuição de *malware*, contas de email utilizadas pelo criador do artefato, tentativas de ataque a outros sistemas e sistemas de comando e controle utilizados para passar comandos ao *malware* caso seja do tipo *bot*.

3.5.4.4. Rootkit

Alguns exemplares de *malware* mais sofisticados utilizam *rootkits* para que as ações maliciosas sejam executadas em nível de *kernel* e, portanto, não possa ser monitorada por ferramentas que capturam as ações dentro do ambiente virtualizado, como o `Capture-BAT`. Dessa forma é preciso, pelo menos, identificar quando os *rootkits* estão sendo usados. A ferramenta `Gmer` é utilizada para essa tarefa

3.5.4.5. DLLs

As bibliotecas encontradas na análise estática não são necessariamente as mesmas utilizadas durante a execução do *malware*, assim é preciso checar quais foram carregadas em tempo de execução. Para essa tarefa utiliza-se a ferramenta `ListDlls`.

3.5.4.6. Término da análise dinâmica

Ao fim da análise, o *script WSH* de controle deverá terminar os processos do *malware* e das ferramentas utilizadas, reunir todos os arquivos de registro e enviá-los por FTP para a máquina base. Esta, por sua vez, deverá conferir se os arquivos de registro foram enviados corretamente e deve desligar a máquina virtual forçadamente, caso precise, como mostrado na Seção 3.5.4.1. Por fim, a máquina utilizada será restaurada para o último *snapshot*. Os diversos arquivos de resultados serão processados pela máquina base e reunidos em um único relatório, para então ser armazenado na base de dados.

3.5.5. Vulnerabilidades do sistema

Como enunciado na Seção 3.4.1, o Protótipo Sandbox possui diversas vulnerabilidades, como por exemplo, não ter proteção contra rotinas que identifiquem ambientes virtualizados. Outra fragilidade é a identificação de uma das ferramentas utilizadas na análise dinâmica. Assim, caso o relatório seja suspeito é preciso realizar uma análise manual e aprofundada no *malware*.

3.5.6. Análise aprofundada

Para análises mais aprofundadas é preciso compreender em baixo nível o que o *malware* está fazendo no sistema, para isto, é possível fazer um *dump* da memória RAM em um momento crítico para analisá-lo ou fazer *debugging* do código binário. Através de um *dump* é possível identificar interceptações, *rootkits* e mutexes (objetos de sincronização) presentes no sistema e que podem estar bloqueando a monitoração de alguma atividade maliciosa, além de ser possível ver *strings*, bibliotecas, conteúdo da pilha (*stack*) e conteúdo da *heap* dos processos que estavam em execução. Uma opção de programa para essa finalidade é o `Memoryze`. Para realizar a engenharia reversa e o compreendimento profundo do funcionamento do *malware*, é preciso utilizar um *debugger*, como o *OllyDbg*. Como a análise manual do código é uma tarefa exaustiva, recomenda-se utilizá-la apenas para conferir os trechos mais suspeitos.

3.6. Estudo de Caso

Nesta parte será apresentado um estudo de caso do processo completo de comprometimento de uma máquina, que ocorreu a partir do acesso a uma *URL* maliciosa. Este endereço foi retirado do site *www .malwaredomainlist.com* que disponibiliza *URL*s maliciosas diariamente. A análise foi feita utilizando um sistema de análise de *URL*s maliciosas, desenvolvido por um dos autores, onde todas as ações desenvolvidas pela *URL* são inspecionadas, juntamente com as *system calls* realizadas pelo processo do *browser* e o tráfego de rede gerado. Serão apresentados alguns trechos dos relatórios produzidos pelo sistema, bem como uma explicação sobre as ações que estão sendo observadas. O exemplar analisado utiliza um *applet java* malicioso, o qual explora uma vulnerabilidade e possibilita o *download* de um *malware*, que é executado no sistema logo após a exploração.

A arquitetura do sistema onde foi realizado o estudo é composto de um *Desktop* com uma máquina virtual do sistema *VirtualBox*, dispondo de acesso controlado à internet. O sistema utilizado no ambiente *guest*, ou seja o ambiente virtualizado, é um

Windows XP SP2 com instalação básica. Para permitir a execução de certos tipos de código que não dispõem de suporte nativo em sistemas *Windows* foi necessário adicionar alguns aplicativos extras, como por exemplo *Java* e *Flash*. A *URL* maliciosa foi acessada em um *browser Internet Explorer 8*, normalmente utilizado pela grande maioria de usuários de sistema *Windows*.

Primeiramente, o *Internet Explorer* é iniciado juntamente com o componente responsável pela captura das ações desenvolvidas pela *URL* e pelo que faz a captura das chamadas de sistema executadas pelo processo do*browser* e seus filhos. Após esta etapa, o *browser* fica a espera de uma *URL*, para que então possa ser realizada uma análise. Diferentemente do componente que obtém as informações de execução da *URL*, o componente de captura de chamadas de sistema fica capturando todas as ações do *Internet Explorer*, mesmo que a *URL* não tenha sido carregada ainda. As ações desnecessárias serão filtradas posteriormente, deixando somente as ações relevantes. Terminada a etapa de instanciação do sistema, a *URL* maliciosa é passada para o *browser*, o qual irá acessá-la, tendo assim todas as atividades geradas capturadas.

Quando a página utilizada no estudo de caso é carregada pelo *browser*, o componente de captura obtém várias ações inofensivas, realizadas normalmente por qualquer página *web*. No meio destas ações pode haver código de exploração, caso algum *script* malicioso em JavaScript seja executado. No trecho de código apresentado a seguir 3.1, uma página HTML é utilizada para chamar um objeto Java que recebe um parâmetro (linha 4) cujo valor, representado por uma sequência de caracteres "A", é usado para explorar uma vulnerabilidade na máquina virtual Java que executa no *browser* do lado do cliente.

Trecho de Código 3.1. Código HTML utilizado na exploração do *browser*

```
1 <html>
2     <object id="java_obj" classid="clsid:CAFEEFAC-DEC7-0000-0000-
          ABCDEFFEDCBA" width="0" height="0">
3                 <PARAM name="launchjnlp" value="1"/>
4                 <PARAM name="docbase" value="AAAAAAAAAAAAAAAAAAAA..."/
                      >
5     </object>
6 </html>
```

Observando a saída gerada pela ferramenta que captura as ações executadas durante o carregamento da página, mostrada no trecho de código 3.2, podemos observar que um dos atributos presentes na linha 2, e os das linhas 3 e 4 do trecho de código 3.1 estão presentes nas ações capturadas pela ferramenta de análise. É possível identificar isto nas linhas 1, 2 e 3 do trecho de código 3.2, que correspondem a uma parte do código obtido durante a execução da pagina maliciosa no *browser*.

Trecho de Código 3.2. Ações capturadas durante o carregamento da *URL* maliciosa.

```
1 10:27:45.010 <URL MALICOSA> SET PROPERTY clsid=CAFEEFAC-DEC7-0000-0000-
      ABCDEFFEDCBA
2 10:27:45.070 <URL MALICOSA> SET PROPERTY launchjnlp=1
3 10:27:45.230 <URL MALICOSA> SET PROPERTY docbase=<EXPLOIT>
```

Já nas ações desenvolvidas pelo processo do *browser* e seus filhos, foi possível notar a ação de um *applet java* e de vários outros processos. Cada ação executada é

colocada em uma linha dividida em três campos, sendo o primeiro o nome do processo executor da ação, o segundo o tipo de ação realizada e o terceiro o alvo da ação. Dentre as ações executadas pelo processo do *browser* foi possível identificar um conjunto de ações relevantes, que correspondem à criação de arquivos, de um *applet java* e de um novo processo, possivelmente um *malware*.

No trecho de código apresentado em 3.3 é possível ver um trecho das ações executadas no sistema pelo *applet java*.

Trecho de Código 3.3. Trecho de atividade realizada pelo processo *applet Java* carregado pelo *Internet Explorer*.

```
1  javaw.exe;CreateFile;C:\WINDOWS\system32\d3d9caps.tmp
2  javaw.exe;WriteFile;C:\WINDOWS\system32\d3d9caps.tmp
```

Já o trecho de código apresentado em 3.4, que corresponde a algumas ações executadas pelo processo do *browser*, é possível ver que ele cria um novo arquivo e executa-o. Este novo processo irá realizar várias ações, sendo que as mais relevantes estão apresentadas no trecho de código 3.5. Neste pedaço de código existem trechos de execução de quatro processos: o primeiro deles corresponde às ações que vão da linha 1 a 13 e é o arquivo criado e executado pelo *Internet Explorer*, como pode ser verificado na linha 3 do trecho de código 3.4. Os outros três processos, que estão entre as linhas 15 a 21, 23 a 25 e 27 a 28, foram todos criados pelo processo `9eg1.exe`, conforme mencionado anteriormente.

Trecho de Código 3.4. Trecho de atividade realizada pelo processo do *Internet Explorer*.

```
1  iexplore.exe;CreateFile;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\9eg1.exe
2  iexplore.exe;WriteFile;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\9eg1.exe
3  iexplore.exe;CreateProcess;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\9eg1.exe
```

Trecho de Código 3.5. Trecho de atividade realizada pelos processos criados pelo *Internet Explorer*.

```
1  9eg1.exe;ConnectNet;<IP>:8000
2  9eg1.exe;SendNet;TCP:<IP>:8000
3  9eg1.exe;ReceiveNet;TCP:<IP>:8000
4  9eg1.exe;CreateFile;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\_4.tmp
5  9eg1.exe;WriteFile;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\_4.tmp
6  9eg1.exe;CreateProcess;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\_4.tmp
7  9eg1.exe;CreateFile;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\_5.tmp
8  9eg1.exe;WriteFile;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\_5.tmp
9  9eg1.exe;CreateProcess;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\_5.tmp
10 9eg1.exe;CreateFile;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\_6.tmp
11 9eg1.exe;WriteFile;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\_6.tmp
12 9eg1.exe;CreateProcess;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\_6.tmp
13 9eg1.exe;TerminateProcess;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\9eg1.ex
14
15 _4.tmp;ConnectNet;<IP>:80
16 _4.tmp;SendNet;TCP:<IP>:80
17 _4.tmp;DisconnectNet;TCP:<IP>:80
18 _4.tmp;CreateFile;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\DAT7.tmp.exe
19 _4.tmp;WriteFile;C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\DAT7.tmp.exe
20 _4.tmp;CreateFile;C:\WINDOWS\system32\drivers\str.sys
21 _4.tmp;WriteFile;C:\WINDOWS\system32\drivers\str.sys
22
```

```
23 _5.tmp;CreateFile;C:\Documents and Settings\Administrator\Local
       Settings\Temporary Internet Files\Content.IE5\ZMD946CA\gr[1].htm
24 _5.tmp;DeleteFile;C:\Documents and Settings\Administrator\Local
       Settings\Temporary Internet Files\Content.IE5\ZMD946CA\gr[1].htm
25
26 _6.tmp;CreateFile;C:\WINDOWS\system32\dll.dll
27 _6.tmp;WriteFile;C:\WINDOWS\system32\dll.dll
```

Analisando todas as linhas que mostram o comportamento dos processos criados por `9eg1.exe` é possível verificar que existem ações de conexão de rede (linhas 15 a 17), criação de *drivers* (linha 20), *DLLs* (linha 27) e criação e remoção de arquivos (linhas 23 e 24).

Uma outra informação importante obtida durante a análise é o tráfego de rede gerado pela execução do código malicioso. A partir dele é possível observar que o *malware* faz conexões em redes *P2P* utilizando o protocolo *Gnutella*. No conjunto de ações presentes no trecho de código 3.6 é possível ver a ação de *download* de uma lista de *servents* (clientes) *Gnutella* presentes na rede (linha 1) e a tentativa de conexão a um destes *servents* (linhas 16 a 22).

Trecho de Código 3.6. Requisições à rede *Gnutella* realizadas pelo *malware*.

```
1  GET /skulls.php?net=gnutella2&get=1&client=RAZA2.5.0.0 HTTP/1.1
2  Host: gwc2.wodi.org
3  Content-Type: text/html
4  Accept-Language: en
5  User-Agent: Shareaza
6  Connection: close
7  HTTP/1.1 200 OK
8  Date: Wed, 30 Mar 2011 13:28:46 GMT
9  Server: Apache/2.2.15 (Linux/SUSE)
10 X-Powered-By: PHP/5.3.3
11 Connection: close
12 X-Remote-IP: <IP>
13 Content-Length: 1257
14 Content-Type: text/plain
15 <LISTA DE HOSTs>
16 GNUTELLA CONNECT/0.6
17 Listen-IP: 0.0.0.0:18509
18 Remote-IP: <IP>
19 User-Agent: Shareaza 2.5.0.0
20 Accept: application/x-gnutella2
21 X-Ultrapeer: False
22 X-Ultrapeer-Needed: True
```

Neste capítulo foi exibida uma análise de um comprometimento que ocorreu a partir do acesso a uma *URL* maliciosa, tendo início na exploração de uma vulnerabilidade na máquina virtual Java utilizada pelo *browser* e culminando em ações nocivas efetuadas diretamente no sistema operacional. O endereço da *URL* foi obtido de um domínio que disponibiliza diariamente *links* de *sites* maliciosos. Na análise foi apresentado todo o processo do ataque, desde as ações maliciosas realizadas pelo código *HTML* da página maliciosa, até as modificações realizadas no sistema por um *malware* que foi obtido por *download*.

Referências

[1] Thorsten Holz, Markus Engelberth, Felix Freiling. Learning More About the Underground Economy: A Case-Study of Keyloggers and Dropzones. Reihe informatik tr-2008-006, University of Mannheim, 2008.

[2] B. Stone-Gross, M. Cova, B. Gilbert, R. Kemmerer, C. Kruegel, and G. Vigna. Analysis of a Botnet Takeover. *IEEE Security and Privacy Magazine*, 9(1):64–72, January 2011.

[3] Chao Yang, Robert Harkreader, and Guofei Gu. Die free or live hard? empirical evaluation and new design for fighting evolving twitter spammers. In *Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection (RAID'11))*, September 2011.

[4] Kevin Zhijie Chen, Guofei Gu, Jose Nazario, Xinhui Han, and Jianwei Zhuge. WebPatrol: Automated collection and replay of web-based malware scenarios. In *Proceedings of the 2011 ACM Symposium on Information, Computer, and Communication Security (ASIACCS'11)*, March 2011.

[5] Michael Becher, Felix C. Freiling, Johannes Hoffmann, Thorsten Holz, Sebastian Uellenbeck, and Christopher Wolf. Mobile security catching up? revealing the nuts and bolts of the security of mobile devices. In *Proceedings of the 2011 IEEE Security and Privacy Symposium*, pages 96–111, May 2011.

[6] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS: Detecting Privacy Leaks in iOS Applications. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2011.

[7] G. Stringhini, C. Kruegel, and G. Vigna. Detecting spammers on social networks. In *Annual Computer Security Applications Conference*, 2010.

[8] Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, Newso James, Pongsin Poosankam, and Prateek Saxena. Bitblaze: A new approach to computer security via binary analysis. In *Proceedings of the 4th International Conference on Information Systems Security*, ICISS '08, pages 1–25, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] Chia Yuan Cho, Juan Caballero, Chris Grier, Vern Paxson, and Dawn Song. Insights from the inside: a view of botnet management from infiltration. In *Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, LEET'10, pages 2–2, Berkeley, CA, USA, 2010. USENIX Association.

[10] Junjie Zhang, Xiapu Luo, Roberto Perdisci, Guofei Gu, Wenke Lee, and Nick Feamster. Boosting the scalability of botnet detection using adaptive traffic sampling. In *Proceedings of the 2011 ACM Symposium on Information, Computer, and Communication Security (ASIACCS'11)*, March 2011.

[11] Seungwon Shin, Raymond Lin, and Guofei Gu. Cross-analysis of botnet victims: New insights and implications. In *Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection (RAID'11))*, September 2011.

[12] Jason Franklin, Vern Paxson, Adrian Perrig, Stefan Savage. An inquiry into the nature and causes of the wealth of internet miscreants. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 375–388. ACM, 2007.

[13] Hamad Binsalleeh, Thomas Ormerod, Amine Boukhtouta, Prosenjit Sinha, Amr Youssef, Mourad Debbabi, Lingyu Wang. On the Analysis of the Zeus Botnet Crimeware Toolkit. In *Proceedings of the Eighth Annual Conference on Privacy, Security and Trust*, PST '2010. IEEE Press, August 2010.

[14] Seungwon Shin, Guofei Gu. Conficker and beyond: a large-scale empirical study. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 151–160. ACM, 2010.

[15] Nicholas Falliere, Liam O. Murchu, Eric Chien. Symantec Stuxnet Report: W32.Stuxnet Dossier. Report, Symantec, October 2010.

[16] Julio Canto, Marc Dacier, Engin Kirda, Corrado Leita. Large scale malware collection: lessons learned. In *27th International Symposium on Reliable Distributed Systems*, SRDS 2008. IEEE, October 2008.

[17] André R. A. Grégio, Isabela L. Oliveira, Rafael D. C. dos Santos, Adriano M. Cansian, Paulo L. de Geus. Malware distributed collection and pre-classification system using honeypot technology. In *Data Mining, Intrusion Detection, Information Security and Assurance, and Data Networks Security 2009*, volume 7344. SPIE.

[18] André R. A. Grégio, Dario S. Fernandes Filho, Vitor M. Afonso, Rafael D. C. dos Santos, Mario Jino and Paulo L. de Geus. Behavioral analysis of malicious code through network traffic and system call monitoring. In *Defense, Security and Sensing 2011*, volume 8059. SPIE.

[19] Paul Baecher, Thorsten Holz, Markus KÃ¶tter, Georg Wicherski. The Malware Collection Tool (mwcollect). PÃ¡gina na internet, 2011. `http://www.mwcollect.org/`.

[20] Joanna Rutkowska. Introducing Stealth Malware Taxonomy. White paper, 2006. `http://invisiblethings.org/papers/malware-taxonomy.pdf`.

[21] Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert Cunningham. A taxonomy of computer worms. In *Proceedings of the 2003 ACM workshop on Rapid malcode*, WORM '03, pages 11–18, New York, NY, USA, 2003. ACM.

[22] David Dagon, Guofei Gu, Cliff Zou, Julian Grizzard, Sanjeev Dwivedi, Wenke Lee, and Richard Lipton. R.: A taxonomy of botnets. In *In: Proceedings of CAIDA DNS-OARC Workshop*, 2005.

[23] Jonathon Giffin, Somesh Jha, and Barton Miller. Automated discovery of mimicry attacks. In *Symposium on Recent Advances in Intrusion Detection (RAID)*, 2006.

[24] Gregoire Jacob, Eric Filiol, and Herve Debar. Functional polymorphic engines: formalisation, Implementation and use cases. *Journal in Computer Virology*, 5(3), 2008.

[25] Justin Seitz. *Gray Hat Python: Python Programming for Hackers and Reverse Engineers*. No Starch Press, San Francisco, CA, USA, 2009.

[26] Hispasec Sistemas. Virustotal. `http://www.virustotal.com/`, 2011.

[27] Nicolas Fallieri, Liam O. Murchu, and Eric Chien. W32.stuxnet dossier. `http://www.symantec.com/en/ca/content/en/us/enterprise/media/security\_response/whitepapers/w32\_stuxnet\_dossier.pdf`, 2011.

[28] Norman Sandbox. Norman sandbox whitepaper. `http://download.norman.no/whitepapers/whitepaper_Norman_SandBox.pdf`, 2003.

[29] Joris Kinable and Orestis Kostakis. Malware classification based on call graph clustering. *CoRR*, abs/1008.4365, 2010.

[30] Madhu Shankarapani, Subbu Ramamoorthy, Ram Movva, and Srinivas Mukkamala. Malware detection using assembly and api call sequences. *Journal in Computer Virology*, 7:107–119, 2011. 10.1007/s11416-010-0141-5.

[31] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. Learning and classification of malware behavior. In *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '08, pages 108–125, Berlin, Heidelberg, 2008. Springer-Verlag.

[32] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauscheck, Christopher Kruegel, and Engin Kirda. Scalable, Behavior-Based Malware Clustering. In *16th Symposium on Network and Distributed System Security (NDSS)*, 2009.

[33] Gregoire Jacob, Matthias Neugschwandtner, Paolo Milani Comparetti, Christopher Kruegel, and Giovanni Vigna. A static, packer-agnostic filter to detect similar malware samples. Technical Report 2010-26, UCSB, November 2010.

[34] Younghee Park, Douglas Reeves, Vikram Mulukutla, and Balaji Sundaravel. Fast malware classification by automated behavioral graph matching. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '10, pages 45:1–45:4, New York, NY, USA, 2010. ACM.

[35] Michael Bailey, Jon Oberheide, Jon Andersen, Zhuoqing Morley Mao, Farnam Jahanian, and Jose Nazario. Automated Classification and Analysis of Internet Malware. In *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID '07)*, pages 178–197, Gold Coast, Australia, September 2007.

[36] Qinghua Zhang and D.S. Reeves. Metaaware: Identifying metamorphic malware. In *Computer Security Applications Conference. ACSAC 2007.*, pages 411 –420, dec. 2007.

[37] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 421 –430, dec. 2007.

[38] Gnu diff. `http://en.wikipedia.org/wiki/Diff`, 2011.

[39] The jaccard index. `http://en.wikipedia.org/wiki/Jaccard{\textunderscore}index`, 2011.

[40] Clam antivirus. `http://www.clamav.net`, 2011.

[41] Peng Li, Limin Liu, Debin Gao, and Michael K. Reiter. On challenges in evaluating malware clustering. In *Proceedings of the 13th International Conference on Recent Advances in Intrusion Detection*, RAID'10, pages 238–255, Berlin, Heidelberg, 2010. Springer-Verlag.

[42] C. Seifert, R. Steenson, I. Welch, P. Komisarczuk, and B. Endicott-Popovsky. Capture-a behavioral analysis tool for applications and documents. *digital investigation*, 4:23–30, 2007.

[43] Virtualbox, Julho 2011. `http://www.virtualbox.org/`.

[44] Lorenzo Martignoni, Roberto Paleari, Giampaolo Fresi Roglia, and Danilo Bruschi. Testing CPU emulators. In *Proceedings of the eighteenth international symposium on Software testing and analysis*, ISSTA '09, pages 261–272, New York, NY, USA, 2009. ACM.

[45] Galen Hunt and Doug Brubacher. Detours: binary interception of Win32 functions. In *Proceedings of the 3rd conference on USENIX Windows NT Symposium - Volume 3*, pages 14–14, Berkeley, CA, USA, 1999. USENIX Association.

[46] Gábor Pék, Boldizsár Bencsáth, and Levente Buttyán. nEther: in-guest detection of out-of-the-guest malware analyzers. In *Proceedings of the Fourth European Workshop on System Security*, EUROSEC '11, pages 3:1–3:6, New York, NY, USA, 2011. ACM.

[47] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. 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, 2008. ACM.

[48] Tom Liston and Ed Skoudis. On the Cutting Edge: Thwarting Virtual Machine Detection., 2006. `http://handlers.sans.org/tliston/ThwartingVMDetection\_Liston\_Skoudis.pdf`.

[49] Danny Quist and Val Smith. Detecting the Presence of Virtual Machines Using the Local Data Table, 2006. http://www.offensivecomputing.net/files/active/0/vm.pdf.

[50] Mendel Rosenblum. The Reincarnation of Virtual Machines. *Queue*, 2:34–40, July 2004.

[51] Microsoft Portable Executable and Common Object File Format Specification. http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx.

[52] Mark E. Russinovich and David A. Solomon. *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server(TM) 2003, Windows XP, and Windows 2000 (Pro-Developer)*. Microsoft Press, Redmond, WA, USA, 2004.

[53] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security and Privacy*, 5(2):32–39, March 2007.

[54] CWSandbox :: Behavior-based Malware Analysis, Julho 2011. http://mwanalysis.org/.

[55] Cuckoo Sandbox - Automated Malware Analysis System, MarÃ§o 2011. http://www.cuckoobox.org/.

[56] JoeBox, Julho 2011. http://www.joesecurity.org/.

[57] Anubis - Analyzing Unknown Binaries, MarÃ§o 2011. http://anubis.iseclab.org/.

[58] CaptureBat, MarÃ§o 2011. http://www.honeynet.org/project/CaptureBAT.

[59] Greg Hoglund and Jamie Butler. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Professional, 2005.

[60] Thomas Raffetseder, Christopher Krugel, and Engin Kirda. Detecting System Emulators. In *ISC*, pages 1–18, 2007.

[61] Bill Blunden. *The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System*. Jones and Bartlett Publishers, Inc., USA, 2009.

[62] Holy Father. Hooking Windows API-Technics of Hooking API Functions on Windows, 2004.

[63] Fabrice Bellard. 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, 2005. USENIX Association.

[64] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. TTanalyze: A Tool for Analyzing Malware, 2006.

[65] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 281–290, New York, NY, USA, 2010. ACM.

[66] J. Nazario. Phoneyc: A virtual client honeypot. In *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, pages 6–6. USENIX Association, 2009.

[67] C. Seifert and R. Steenson. Capture - honeypot client (capture-hpc), 2006.

[68] Vmware, Julho 2011. `http://www.vmware.com/`.

[69] Min Gyung Kang, Heng Yin, Steve Hanna, Stephen McCamant, and Dawn Song. Emulating emulation-resistant malware. In *Proceedings of the 1st ACM workshop on Virtual machine security*, VMSec '09, pages 11–22, New York, NY, USA, 2009. ACM.

Capítulo

4

Introdução à Composibilidade Universal

João J. C. Gondim
Departamento de Ciência da Computação
Universidade de Brasília

Resumo

A composibilidade universal - universal composability (UC) é um framework geral que se propõe representar protocolos criptográficos e analisar sua segurança. Neste framework, a segurança dos protocolos é preservada sob uma operação geral de composição com outros protocolos, permitindo o design modular e a análise de protocolos complexos a partir de blocos mais simples de forma unificada e sistemática. Além disso, neste framework a segurança dos protocolos é mantida sob uma operação geral de composição. Esta operação é definida de tal forma que, satisfazendo as condições desta operação, é possível construir protocolos que não só são seguros mas também tem sua segurança preservada quando compostos entre si, mesmo na presença de um número qualquer de cópias concorrentes. A proposta é apresentar no minicurso os conceitos básicos da segurança UC e sua aplicação na concepção e análise de um protocolo criptográfico. Apesar de já fazer dez anos da apresentação do conceito, contando com vários resultados relevantes, ele ainda é pouco conhecido.

Inicialmente, é feita toda a modelagem de como vem a ser o modelo de execução dos protocolos. Apesar da motivação inicial ter sido protocolos relacionados a primitivas criptográficas, a abordagem se aplica a protocolos de computação segura multipartes, como indicado em trabalhos recentes. O modelo de computação será construído a partir da noção de sistemas de máquinas de Turing interativas. Na sequência é apresentado o framework, começando com a definição de dois modelos específicos que diferem no modelo de comunicação entre as partes e do adversário. Pode-se então apresentar o Teorema da Composição. Com o Teorema da Composição, duas aplicações são abordadas,

no contexto dos protocolos de comprometimento de bit. Primeiro é mostrado um exemplo de como o framework pode ser usado para analisar os requisitos para a construção segura de protocolos de comprometimento de bit. Especificamente, é demonstrado que não há implementação segura sem que se recorra a hipótese de setup. Na sequência, se constrói um protocolo de comprometimento de bit e se estabelece a segurança UC do mesmo.

4.1. Introdução

A demonstração rigorosa da segurança de um protocolo é um requisito indispensável no projeto de um protocolo criptográfico. Isto requer pelo menos dois passos de formalização. O primeiro envolve a definição do modelo matemático apropriado para representar o protocolo, enquanto o outro consiste em formular nesse modelo uma definição de segurança que capture os requisitos para a tarefa que o protocolo pretende realizar seguramente. Com a definição de segurança, pode-se então estabelecer que o protocolo é seguro demonstrando que sua representação matemática satisfaz a definição de segurança no modelo adotado.

Entretanto, a formulação de um modelo matemático apropriado para representação dos protocolos e a formulação de uma definição de segurança adequada no modelo escolhido são tarefas complexas. O modelo deve ser capaz de capaz de representar todos os comportamentos adversariais realistas e a definição deve garantir que a noção intuitiva de segurança foi capturada corretamente com respeito a qualquer comportamento adversarial que seja considerado.

Outro aspecto importante no contexto da segurança de um protocolo criptográfico é a sua robustez com relação ao ambiente de execução. O comportamento de um protocolo criptográfico sofre grande influência de seu ambiente de execução, em particular com relação aos outros protocolos que estejam executando no mesmo sistema ou rede. Consequentemente, um critério que precisa ser incorporado à definição de segurança é a garantia de robustez ao ambiente de execução, que pode ser colocada de forma mais geral como uma forma de modularidade. Isto é, a necessidade de se garantir a segurança quando o protocolo é usado como um componente dentro de um sistema maior.

Tradicionalmente, primitivas criptográficas são desenvolvidas inicialmente como tarefas isoladas, sem levar em consideração ambientes de execução mais complexos. esta abordagem tem vantagens como permitir definições mais concisas e intuitivas além de simplificar a análise dos protocolos. Entretanto, em muitos casos estas definições iniciais se mostram insuficientes quando as condições do ambiente são refinadas aumentando em complexidade onde os protocolos são empregados em situações mais gerais. Alguns exemplos desse tipo de situação que podem ser citados, entre outros, são:

- cifração: a noção básica de segurança semântica [Goldwasser and Micali 1984] foi depois refinada de várias formas de segurança ataques CCA (como [Naor and Yung 1990], [Dolev et al. 2000], [Rackoff and Simon 1992], [Bellare et al. 1998b]) e segurança adaptativa (como por exemplo [Beaver and Haber 1992], [Canetti et al. 1996]), visando satisfazer situações mais gerais;

- comprometimento: onde as noções iniciais foram estendidas com conceitos como não maleabilidade ([Dolev et al. 2000], [Di Crescenzo et al. 1998], [Fischlin and Fischlin 2000]) e equivocação ([Brassard et al. 1988], [Beaver 1996]);

- zero knowledge: onde se demonstrou que as noções iniciais ([Goldwasser et al. 1989], [Goldreich and Oren 1994]) não eram fechadas sob composição paralela e concorrente, levando a necessidade de novos conceitos e construção ([Goldreich and Krawczyk 1990], [Dwork et al. 1998], [Canetti et al. 1999], [Barak et al. 2001]);

- troca de chaves: onde o conceito original não era suficiente para garantir sessões seguras (como em [Bellare and Rogaway 1994], [Bellare et al. 1998a], [Shoup 1999], [Canetti and Krawczyk 2001]);

- oblivious transfer: como em [Rabin 1981], [Even et al. 1985], [Garay and MacKenzie 2000].

Uma forma de capturar as questões de segurança que surgem em ambientes de execução específico ou em uma dada aplicação é representar o ambiente ou a aplicação diretamente dentro de uma extensão da definição de segurança. Essa abordagem é a adotada, por exemplo, em casos de troca de chaves [Bellare and Rogaway 1994], [Canetti and Krawczyk 2001], comprometimento não maleável [Dolev et al. 2000], e em zero-knowledge concorrente [Dwork et al. 1998], onde a definição modela explicitamente várias instâncias do protocolo em questão que são coordenadas pelo adversário. Essa abordagem tem a desvantagem de resultar em definições cada vez mais complexas e é inerentemente limitado em escopo pois trata apenas de ambientes e questões específicas.

Uma abordagem alternativa, adotada no *framework* UC é tratar os protocolos como *stand alone* mas garantir que sua composição é segura. Isto é, as definições de segurança se aplicam na inspeção de uma instância isolada do protocolo. Em situações complexas, onde uma instância de um protocolo pode ser executada concorrentemente com outras instâncias, com entradas arbitrárias e possivelmente com controle do adversário, a segurança é garantida se preservada sob uma operação geral de composição de protocolos. Esta abordagem simplifica consideravelmente o processo da formulação de uma definição de segurança e da análise do protocolo. Além disso, a segurança do protocolo é garantida em ambientes arbitrários, mesmo os que não tenham sido explicitamente considerados.

A abordagem descrita acima, juntamente com sua operação de composição, tem como requisito básico a formulação de um *framework* geral em que se possa representar os protocolos criptográficos e seus requisitos de segurança. Várias definições gerais de protocolos seguros foram propostas, como por exemplo [Goldwasser and Levin 1991], [Micali and Rogaway 1992], [Beaver 1991], [Ben-Or et al. 1993], [Pfitzmann and Waidner 1994], [Canetti 1998], [Hirt and Maurer 2000], [Pfitzmann et al. 2000], [Dodis and Micali 2000], [Pfitzmann and Waidner 2000], sendo candidatas para esse *framework* geral. Entretanto, além de restrições de escopo, seja nas situações ou tarefas às quais se aplicam, elas

não oferecem garantia geral de segurança na composição de protocolos criptográficos. Isto é o caso especialmente em situações em que os adversários são computacionalmente limitados e várias instâncias do protocolo executam simultaneamente com possível controle do adversário.

As motivações listadas acima formam o princípios que levaram ao desenvolvimento do *framework* UC, voltado para a representação e análise de protocolos criptográficos. O *framework* UC define uma metodologia geral para expressar os requisitos de segurança de protocolos criptográficos. Além disso, provê um método geral de composição de protocolos, de forma que a definição de segurança expressa no framework se preserva sob a composição, chamada de *composição universal*.

De forma extremamente resumida, a composição universal pode ser vista de maneira intuitiva como uma generalização da operação de chamada de subrotina em algoritmos sequenciais para ambientes distribuídos com execuções concorrentes de protocolos. A preservação da segurança no *framework* implica que um protocolo seguro permanecerá seguro mesmo quando executando em um ambiente arbitrário e desconhecido multiparte multiexecução.

Recentemente, o *framework* UC tem sido aplicado para estabelecer a segurança composicional de aplicações com funcionalidades mais gerais que protocolos criptográficos. Como exemplo, pode-se citar dentre outras aplicações: serviços de sistemas operacionais [Canetti et al. 2010], produto interno [Dowsley et al. 2010], autenticação [Chari et al. 2011], troca de chaves [Canetti and Gajek 2010] [Canetti et al. 2005] [Gorantla et al. 2009], incoercibilidade [Unruh and Müller-Quade 2009], criptografia simétrica [Kuesters and Tuengerthal 2009], certificação digital [Canetti 2003], assinatura digital [Kurosawa and Furukawa 2008], análise simbólica de protocolos criptográficos [Canetti and Herzog 2004], computação síncrona [Katz et al. 2011], compartilhamento de segredo [Dowsley et al. 2009], análise de protocolos [Green and Hohenberger 2008], [Gajek et al. 2008], entre outras aplicações. Isto reforça a formulação geral do *framework* UC, deixando claro sua aplicabilidade ao contexto mais geral da computação segura multipartes.

4.1.1. Breve descrição deste capítulo

Este capítulo está organizado em seções. Na que segue, é construído o modelo computacional sobre o protocolo e o ambiente, juntamente com o adversário, executam. Depois a *framework* UC é apresentada com os modelos adversariais e na seção seguinte é enunciado o Teorema da Composição. As duas seções seguintes ilustram aplicações do *framework* UC tomando como base protocolos de comprometimento de bit.

Primeiro, é apresentado e demonstrado como o *framework* pode ser utilizado na análise de um protocolo, identificando condições para sua implementação segura. Depois, um protocolo de comprometimento de bit UC seguro é construído, seguindo as recomendações da análise feita anteriormente.

O capítulo é encerrado com conclusões e comentários finais.

4.2. O modelo básico de computação

Antes de iniciar a apresentação do framework UC, o modelo computacional é descrito. A formalização de algoritmos em uma rede de comunicação como máquinas de Turing interativas, ITM, segue a abordagem de [Goldwasser et al. 1989]. Uma descrição detalhada, voltada à formalização das interações entre pares de máquinas pode ser encontrada em [Goldreich 2000].

Uma fita de uma máquina de Turing é dita ser write-once, **wo**, se sua cabeça de leitura se move em apenas uma direção. Se uma fita pode se escrita por outras máquinas de Turing, diz-se que ela é externamente write-once, **ewo**. Se a fita pode ser lida e escrita, diz-se que ela é **rw**

Definição 4.2.1. *Uma máquina de Turing interativa (ITM) M tem as seguintes fitas:*

- *uma fita* **ewo** *de identidade*
- *uma fita* **ewo** *de parâmetro de segurança*
- *uma fita* **ewo** *de entrada*
- *uma fita* **ewo** *de comunicação*
- *uma fita* **ewo** *de saída de subrotina*
- *uma fita de saída*
- *uma fita aleatória*
- *uma fita* **rw** *de ativação (1 bit)*
- *uma fita* **rw** *de trabalho*

O conteúdo da fita de identidade é chamado de identidade de M. A identidade de M é interpretada, segundo alguma codificação, como dois strings: o identificador de sessão SID de M, e o identificador da parte PID de M.

O conteúdo da fita de comunicação modela a informação vinda da rede. Ela é interpretada, segundo alguma codificação, como uma sequência de valores chamados mensagens, onde cada mensagem tem dois campos: o campo com a designação do emissor, que corresponde à identidade de alguma ITM, seguido por um campo arbitrário de conteúdo.

O conteúdo da fita de saída de subrotina modela as saídas das subrotinas de M. Ela é interpretada, segundo alguma codificação, como uma sequência de valores chamados saídas de subrotinas, onde cada uma tem dois campos: o campo com o identificador de subrotina, que corresponde à identidade de alguma ITM junto com o código de alguma ITM, e por um campo arbitrário de conteúdo.

O código de alguma ITM M pode incluir instruções para escreve na fita de outra ITM. A sintaxe e o efeito destas instruções são descritos mais adiante.

Definição 4.2.2. *Um sistema de ITMs $S = (I, C)$ é definido por uma ITM inicial I e uma função de controle $C : \{0,1\}^* \rightarrow \{allow, disallow\}$ que determina o efeito das instruções escritas externamente nas ITMs do sistema.*

Definição 4.2.3. *Uma configuração de uma ITM M é composta por:*

- *código;*
- *estado de controle;*
- *conteúdos de todas as fitas;*
- *posições de todas as cabeças*

Uma configuração está ativa se o bit da fita de ativação tem valor 1; caso contrário está desativada.

Definição 4.2.4. *Uma instância $\mu = \{M, id\}$ de uma ITM, ITI, consiste do código M juntamente com o string de identidade $id \in \{0,1\}^*$*

Diz-se que uma configuração é uma configuração de uma instância μ se o código da configuração é M e o conteúdo da fita de identidade é id.

Definição 4.2.5. *Uma ativação de uma ITI μ é uma sequência de configurações que correspondem a uma computação, que inicia a partir de alguma configuração ativa de μ, até que uma configuração inativa seja alcançada. Nesse caso, diz-se que a ativação se completou e que μ está esperando a próxima ativação.*

Se um estado especial de parada, **`halt`**, for atingido a ITI para; nesse caso, a ITI não executará nada em ativações futuras.

Definição 4.2.6. *Uma execução de um sistema $S = (I, C)$, dado um parâmetro de segurança k e entrada x, é uma sequência de ativações de ITIs. A primeira ativação inicia a partir da configuração com o código de I, a entrada x na fita de entrada, 1^k escrito na fita do parâmetro de segurança, um string aleatório r, longo o suficiente, escrito na fita aleatória, e identidade 0.*

A ITI $(I, 0)$ é chamada de ITI inicial. A execução termina quando a ITI inicial atinge o estado **`halt`**, i.e. quando uma configuração de parada da ITI inicial é atingida. A saída gerada pela execução é o conteúdo da fita de saída da ITI inicial na sua configuração de parada.

Para completar a definição de uma execução ainda é necessário descrever o efeito de uma instrução de escrita externa; e como se determina qual a próxima ITI a ser ativada, após se completar uma ativação.

4.2.1. Escrevendo na fita de outra ITI e chamadas de ITIs

Uma instrução de escrita externa por uma ITM em outra, especifica os seguintes parâmetros: os códigos e identidades das ITIs de origem e destino, a fita do destino que será escrita, os dados que serão escritos. A fita de destino pode ser a fita de entrada ou a fita de comunicação, ou a fita de saída de subrotina. O efeito de uma instrução de escrita externa de uma ITI $\mu = (M, id)$ para uma ITI $\mu' = (M', id')$ é definido como segue:

1. se a função de controle C aplicada à sequência de pedidos de escrita externa até o momento não permite que μ escreva na fita especificada de μ', i.e. retorna um valor *disallow*, então a instrução é ignorada.

2. se C permite a operação e uma ITI $\mu'' = (M'', id'')$ com identidade $id' = id''$ já existe no sistema (uma das configurações na execução até agora tem uma identidade id'), então:

   (a) se a fita do destino é a fita de comunicação de μ', então os dados especificados são escritos na fita de comunicação de μ'', juntamente com a identidade *id* da ITI de origem. Consequentemente, uma nova configuração μ'' é gerada. Essa configuração é a última configuração de μ'' nesta execução, com a nova configuração sendo escrita na fita de comunicação. Isto acontece independente do código M'' de μ'' ser igual ao código M' especificado no pedido de escrita externa.

   Esta regra assume que uma ITI não conhece necessariamente o código da ITI para a qual manda ou da qual recebe mensagens.

   (b) se a fita do destino é a fita de saída de subrotina de μ', então os dados especificados são escritos na fita de saída de subrotina de μ'' juntamente com o código e a identidade de μ. Novamente, isso acontece independente de $M'' = M'$.

   Esta regra assume que uma ITI conhece o código da ITI que escreve para sua fita de saída de subrotina. Entretanto, uma ITI não conhece necessariamente o código da ITI para a qual ela gera uma saída.

   (c) se a fita do destino é a fita de entrada de μ' e $M'' = M'$, então os dados especificados são escritos na fita de saída de subrotina de μ'' juntamente com o código e a identidade de μ. Se $M' \neq M''$ então μ vai para um estado especial de erro.

   Esta regra assume que uma ITI pode verificar o código da ITI para a qual provê entradas. Entretanto, uma ITI não conhece necessariamente o código da ITI para da qual ela recebe entradas.

3. se C permite a operação, e nenhuma ITI com identidade id' existe no sistema, então uma nova ITI com código M' e identidade id' é chamada, i.e. uma nova configuração é gerada, com código M' e identidade id' na fita de identidade, 1^k escrito na fita do parâmetro de segurança, um *string* aleatório r, longo o suficiente, escrito na fita aleatória. Uma vez que a nova ITI é chamada, a instrução de escrita externa é executada como acima. Nesse caso diz-se que μ chamou μ'.

Note que as regras de chamada das ITIs, juntamente com o fato da execução começar com uma única ITI, garante que cada ITI no sistema tenha identidade única. Assim, quaisquer duas ITI não tem mesma identidade, independente de seus códigos. Além disso, uma vez que o código e a identidade da ITI de destino devem ser especificados na instrução de escrita externa, estes devem ser computados pela ITI de origem antes da chamada. Assim, não há restrições para SID e PID.

4.2.2. Determinação da ordem de ativações

A ordem de ativações é bem simples. A cada ativação permite-se que cada ITI execute no máximo uma instrução de escrita externa. A ITI cuja fita foi escrita durante uma ativação será a próxima a ser ativada, i.e. a fita de ativação nesta nova configuração da ITI tem o valor 1. Se nenhuma instrução de escrita externa foi executada, a ITI inicial será ativada na sequência.

Esta regra é simples e natural, permitindo quebrar a computação distribuída em uma sequência de eventos locais onde a cada instante de tempo tem-se apenas uma ITI cujo estado local mudou desde sua última ativação.

Quando uma ITI μ escreve uma mensagem m na fita de comunicação de uma ITI μ', diz-se que μ enviou m a μ'. Quando uma ITI μ escreve um valor x na fita de entrada de uma ITI μ', diz-se que μ passou x a μ'. Quando uma ITI μ' escreve um valor x na fita de saída de subrotina de uma ITI μ, diz-se que μ' passou a saída (ou apenas passou) x para μ. Diz-se que μ' é uma subrotina de μ, se μ passou uma entrada para μ' ou se μ' passou uma saída para μ. μ' é uma subsidiária de μ, se μ' é uma subrotina de μ ou de alguma de suas subsidiárias.

4.2.3. Protocolos

Um protocolo é definido como uma ITM, representando o código a ser executado por cada participante.

Definição 4.2.7. *Dado um estado de um sistema de ITMs, a instância do protocolo multipartes π com SID sid é o conjunto de ITMs no sistema cujo código é π e cujo SID é sid.*

Assim, os PIDs nas instâncias dos protocolos são necessariamente diferentes. Assume-se que π ignore todas as mensagens onde o SID é diferente do SID local. Cada ITI em uma instância de um protocolo é chamada de parte. A definição de estado de um sistema de ITIs é apresentada mais adiante.

Definição 4.2.8. *Uma subparte é uma subrotina de uma parte ou de outra subparte.*

Definição 4.2.9. *Uma instância estendida de π inclui todas as partes e subpartes desta instância.*

Deve-se notar que na definição acima é feita uma distinção entre o protocolo, que representa o código que será executado por cada parte e instância do protocolo que representa uma execução específica de um protocolo. Além disso, a associação de um SID

com cada instância do protocolo, onde SID é conhecido por todas as partes, é conveniente para modelar múltiplas instâncias do protocolo executadas concorrentemente no mesmo sistema.

Definição 4.2.10. *O estado de um sistema de ITMs representa uma descrição completa de um certo instante da execução do sistema. Especificamente, consiste da sequência de todas as configurações de todas as ativações do sistema até o dado instante.*

Definição 4.2.11. *O registro de uma execução de um sistema é o estado final de execução, onde a última configuração é terminal para a ITI inicial.*

Definição 4.2.12. *Um sistema estendido é um sistema em que a função de controle pode alterar instruções de escrita externa. Em um sistema $S = (I,C)$, a função de controle C toma como entrada uma sequência de instruções de escrita externa e gera como saída valores como allowed ou disallowed. Em um sistema estendido, a saída de C consiste em uma instrução de escrita externa completa, que pode ser diferente do pedido original de escrita externa. A instrução executada é a saída de C.*

Essa capacidade da função de controle poder alterar instruções de escrita externa será utilizada para modificar o código de ITIs geradas durante a execução.

4.2.4. Sistemas e ITMs probabilísticas e em tempo polinomial

Apesar de não ter sido dito até então, as ITMs de interesse são limitadas em recurso. Especificamente, o foco da modelagem recai sobre as ITMs que operam em tempo polinomial. Entretanto a definição rigorosa da computação com recursos limitados em um ambiente interativo requer certos cuidados. Isto se aplica ao caso em questão em que o cenário é dinâmico, com ITMs sendo geradas durante a evolução do sistema.

O ponto de partida é considerar uma ITM *M* como PPT, probabilística em tempo polinomial, se seu tempo total de execução, em todas as suas ativações, é polinomial no comprimento da sua entrada. Entretanto, esta definição estaria mais apropriada para capturar situações envolvendo pares de ITMs cujas entradas são fixas durante a computação. Assim ela será estendida a cenários mais gerais, onde as instâncias de ITMs podem escrever arbitrariamente nas fitas de outras instâncias ao longo da execução. Especificamente, será necessário limitar o tempo total de execução de um sistema de ITMs. Em particular, o tempo total de execução de cada instância e o número de instâncias deverá ser limitado.

Assim, para garantir esse limite geral, além da condição do tempo ser polinomial sobre o tamanho da entrada serão levados em consideração o parâmetro de segurança, junto com o número de bits escrito em outras ITMs e o número de ITMs que foram escritas. Desse total deve-se descontar o número de bits escritos na ITM.

Desta forma o *n* para o qual a ITM *M* será polinomial, será a soma do parâmetro de segurança *k*, mais o total de bits escritos até então na fita de entrada de *M*, menos os bits escritos por *M* nas fitas de outras ITMs. Deve-se ainda subtrair *k* vezes o número de máquinas em que *M* escreveu. As subtrações são motivadas por: subtrair os bits escritos por *M* nas fitas de outras ITMs leva em consideração que escrever nas fitas de ITMs existentes não aumenta o tempo de execução; e subtrair *k* vezes o número de máquinas

em que M escreveu leva em consideração que as chamadas a novas instâncias também não aumenta o tempo total de execução.

Além disso, estas condições devem valer durante todo o tempo de execução.

Definição 4.2.13. *Seja $p : N \to N$. Uma ITM M é localmente p-limitada se, a qualquer momento durante sua execução, em qualquer de suas configurações, o número total de passos executados por M é no máximo $p(n)$, onde*

$$n = k + n_I - n_O - k.n_N$$

onde k é o parâmetro de segurança, n_I é o número total de bits escritos na fita de entrada de M, n_O é o número total de bits escritos por M na fita de entrada de outras ITM, e n_N é o número de diferentes instâncias de ITMs cujas fitas foram escritas por M. Se M é localmente p-limitada e, além disso, não faz escritas externas ou cada escrita externa especifica uma ITM que é p-limitada, então se diz que M é p-limitada. M é PPT se existe um polinômio p tal que M é p-limitada. Um protocolo multipartes é PPT se ele é uma ITM PPT.

Pode-se ver que o número total de passos na execução de um sistema, onde a ITM inicial é p-limitada, é limitado por $p(n+k)$, onde n é o tamanho da entrada e k é o parâmetro de segurança. Em particular, o número total de instâncias chamadas é limitado por $p(n+k)$. Se a função de controle C é computável em tempo $q(t)$, onde t é o comprimento da entrada para C, então uma execução de um sistema de ITMs pode ser simulado em uma máquina de Turing sequencial com entrada x e k é o parâmetro de segurança, em $O(q(p(|x|+k)))$ passos, onde $|x|$ é o comprimento de x.

Proposição 4.2.1. *Se a ITM inicial em um sistema (I,C) é p-limitada e a função de controle C é computável em tempo $q(t)$. Então uma execução do sistema pode ser simulada em uma única máquina de Turing não interativa M, com entrada x e parâmetro de segurança k, em $O(q(p(|x|+k)))$ passos (onde $|x|$ é o comprimento de x). Em particular, se I e C são PPT então M também é. Isto também vale para sistemas estendidos de ITMs, se as ITM chamadas forem p-limitadas.*

4.2.5. O modelo de execução de protocolo

O modelo de execução de protocolos é definido em termos de um sistema de ITMs, capturando a noção de uma rede assíncrona, onde as comunicações são não autenticadas e não confiáveis. O modelo em si é parametrizado por três ITMs: o protocolo π a ser executado, o ambiente $\mathscr{E}$ e o adversário $\mathscr{A}$. Assim, dados π, $\mathscr{E}$ e $\mathscr{A}$ o modelo para execução de π é o sistema estendido de ITMs PPT $(\mathscr{E}, C_{EXEC}^{\pi,\mathscr{A}})$, onde a ITM inicial é o ambiente $\mathscr{E}$ e função de controle $C_{EXEC}^{\pi,\mathscr{A}}$ que será definida a seguir.

Inicialmente o ambiente $\mathscr{E}$ recebe alguma entrada, que representa o estado inicial do ambiente no qual a execução do protocolo se realiza. Em particular esta entrada representa todas as entradas externas do sistema, incluindo todas as entradas locais de todas as partes. A primeira ITI a ser chamada por $\mathscr{E}$ é definida pela função de controle para ser o adversário $\mathscr{A}$. Além disso, ao longo da computação, $\mathscr{E}$ pode chamar como subrotinas um

número ilimitado de ITIs, passar-lhes entradas, e obter saídas delas, sujeito à restrição que todas as ITIs chamadas tenham o mesmo SID, que por sua vez é escolhido por $\mathscr{E}$. o código destas ITIs é especificado pela função de controle para ser o código de π, independente do código especificado por $\mathscr{E}$. Consequentemente, todas as ITIs chamadas por $\mathscr{E}$, exceto $\mathscr{A}$, são partes de uma única instância de π. Por outro lado, $\mathscr{E}$ não pode passar entradas para, nem aceitar saídas de, nenhuma outra ITI. Em particular, $\mathscr{E}$ não pode interagir com subrotinas de partes de uma instância única de π.

A função de controle permite que partes e subpartes de π chamem outras ITIs e passem entradas e saídas para qualquer outra ITI que não seja o adversário ou o ambiente. Além disso, elas podem escrever mensagens na fita de comunicação de $\mathscr{A}$. Estas mensagens podem especificar uma identidade de uma parte ou subparte de π como destino final da mensagem.

Além disso, a função de controle permite que o adversário $\mathscr{A}$ envie mensagens para qualquer ITI no sistema. Nesse caso, diz-se que $\mathscr{A}$ entrega a mensagem. Não há necessariamente uma correspondência entre as mensagens enviadas pelas partes e as entregues pelo adversário, que não pode chamar subrotinas.

Execução do protocolo π com ambiente $\mathcal{E}$ e adversário $\mathcal{A}$

Dado um protocolo π, adversário $\mathcal{A}$ e ambiente $\mathcal{E}$ execute um sistema estendido de ITMs, conforme definido anteriormente, com ITM inicial $\mathcal{E}$ e uma função de controle como descrita abaixo. $\mathcal{E}$ inicia com parâmetro de segurança k e entrada x e prossegue como abaixo:

1. A primeira ITI a ser chamada por $\mathcal{E}$ é definida pela função de controle como sendo o adversário $\mathcal{A}$. O código de todas as outras ITIs chamadas por $\mathcal{E}$ é definido como sendo π. Além disso, todas essas ITIs devem ter o mesmo SID.

2. Uma vez que o adversário $\mathcal{A}$ é ativado, ele pode passar saída para $\mathcal{E}$ e também executar as seguintes ações:

   (a) $\mathcal{A}$ pode entregar uma mensagem m a uma parte ou subparte com identidade id; exceto para mensagens **`Corrupt`**, não há restrição quanto ao conteúdo ou identidade do destinatário.

   (b) $\mathcal{A}$ pode corromper uma ITI com identidade id, enviando uma mensagem **`Corrupt`** para a ITI em questão. Esta ação é permitida apenas se $\mathcal{A}$ recebeu de $\mathcal{E}$ uma mensagem ou entrada (**`Corrupt`**,id). A mensagem de $\mathcal{A}$ pode incluir parâmetros adicionais.

3. Uma vez que uma parte ou subparte de π é ativada, ela pode realizar as seguintes ações:

   - envia mensagens para $\mathcal{A}$,
   - passar entradas ou saídas para uma parte ou subparte de π
   - passar saídas para $\mathcal{E}$

   A ativação pode ser motivada por:

   - uma nova mensagem entregue por $\mathcal{A}$;
   - uma nova entrada vinda de $\mathcal{E}$;
   - ou um valor escrito na fita de saída de subrotina.

   A resposta a uma mensagem **`Corrupt`** pode variar de acordo com o modelo específico de corrupção e valores de parâmetros. No modelo de corrupção Bizantino, a parte reporta seu estado interno para o adversário. Em todas as ativações futuras por outras partes diferentes de $\mathcal{A}$, a parte corrompida envia seu estado local para $\mathcal{A}$ e segue as instruções de $\mathcal{A}$ na entrega de valores a outras partes.

O adversário $\mathcal{A}$ pode também corromper partes ou subpartes de π. Formalmente, a corrupção de uma parte ou subparte é modelada por uma mensagem especial **`Corrupt`** entregue por $\mathcal{A}$ para a ITI correspondente. A função de controle permite a entrega da mensagem somente se $\mathcal{A}$ recebeu anteriormente uma mensagem especial (**`Corrupt`**,id) do ambiente $\mathcal{E}$. Isso garante que o ambiente quais as partes que foram corrompidas.

A resposta de uma parte ou subparte a uma mensagem **Corrupt** não é definida no modelo geral, sendo sua definição deixada a cargo do protocolo. Isto permite capturar várias formas de corrupção no mesmo modelo computacional. Para prover uma possibilidade concreta de corrupção de uma parte ou subparte, é definido um possível modelo de corrupção, o Bizantino. Nele, uma vez que uma parte ou subparte recebe uma mensagem **Corrupt**, ela envia ao adversário $\mathscr{A}$ seu estado local corrente e em ativações futuras a parte corrompida segue as instruções de $\mathscr{A}$ independente de escritas externas por outras ITIs.

Deve-se dizer que o modelo permite que $\mathscr{A}$ chame novas ITIs entregando-lhes mensagens. Estas ITIs podem, mas não precisam, ser partes da instância de π com SID *sid*. Esta possibilidade é importante pois permite modelar instâncias de protocolos que são chamadas por mensagens vindas da rede. Poderia ser imposta a restrição permitindo $\mathscr{A}$ chamar apenas as partes da instância corrente. Ainda assim, seria necessário restringir que só as partes da instância de π com SID *sid* podem passar saídas para $\mathscr{E}$.

Sem perda de generalidade, os ambientes retorna apenas um bit de saída denotado pela variável aleatória $OUT_{\mathscr{E},C_{EXEC}^{\pi,\mathscr{A}}}(k,x)$, que também será referida como $EXEC_{\pi,\mathscr{A},\mathscr{E}}(k,x)$. O quadro acima resume o modelo de execução.

4.3. O framework UC

A seguir, o framework é apresentado, culminando com o Teorema da Composição na seção seguinte.

4.3.1. Breve descrição geral

Seguindo [Canetti 2000], o framework UC tem por base três componentes: o modelo real, que formaliza o processo de execução do protocolo na presença de um adversário, em um dado ambiente computacional; o modelo ideal, onde o processo que executa a funcionalidade ideal - aquela que o protocolo real pretende implementar - é formalizado. Neste processo as partes não se comunicam entre si, podendo apenas se comunicar com a funcionalidade ideal. Esta por sua vez captura os requisitos que o protocolo deve satisfazer, sendo tratada como uma terceira parte incorruptível; e a emulação da funcionalidade ideal pelo protocolo real. O protocolo real é dito ser seguro se suas funcionalidades correspondem àquelas prescritas pela funcionalidade ideal.

O protocolo real é representado por um conjunto de máquinas de Turing interativas - ITM, representando as partes envolvidas. Para que o protocolo seja realizável em condições realistas, as ITMs são limitadas em recursos, sendo assim probabilísticas e executando em tempo polinomial - PPT. Nas ITM, as fitas de entrada e saída modelam os valores de entrada e saída que são recebidos de, ou enviados para, outros programas rodando na mesma máquina, enquanto as fitas de comunicação modelam as mensagens recebidas e enviadas pelos outros participantes. O adversário também é modelado por uma ITM.

A rede de comunicação provida entre as partes é assíncrona, sem garantia de entrega das mensagens. Toda comunicação é pública, de forma que o adversário pode ter acesso a todo o tráfego, mas as mensagens trocadas são autenticadas, de forma que o ad-

versário não pode modificá-las. O adversário pode ser adaptativo na corrupção das partes e é ativo no controle destas. Tanto o adversário quanto o ambiente são PPT de forma que são realizáveis.

4.3.2. Execução do protocolo no modelo real

Seja um protocolo π, executado pelas partes $P_1, \ldots, P_n$, com um adversário $\mathscr{A}$ em um ambiente $\mathscr{E}$. Todas as partes tem um parâmetro de segurança $k \in \mathbf{N}$ e são polinomiais em k. A execução é uma sequência de ativações, onde a cada ativação um único participante ($\mathscr{E}$, $\mathscr{A}$ ou algum P_i) é ativado. O participante ativo lê a informação nas suas fitas de entrada (incluídas as de comunicação), executa código e escreve nas fitas de saída (também incluídas as de comunicação). O ambiente $\mathscr{E}$ pode além de isso ler e escrever nas fitas de entrada e de saída, respectivamente, das partes. O adversário $\mathscr{A}$ pode ler mensagens das fitas de saída das partes, copiá-las e entregá-las às partes destinatárias. Note que somente as mensagens geradas pelas partes podem ser entregues, já que o adversário $\mathscr{A}$ não pode alterar ou gerar duplicatas de mensagens. O adversário $\mathscr{A}$ pode corromper as partes, passando a controlá-la e a conhecer as informações internas por ela conhecidas.

Inicialmente, o ambiente $\mathscr{E}$ é ativado. Uma vez ativado, ele pode escreve informação na fita de entrada de alguma parte P_i ou do adversário $\mathscr{A}$. A ativação da entidade se dá assim que o ambiente $\mathscr{E}$ complete sua ativação, i.e. entre em um estado de espera. Se nenhuma fita de entrada for escrita a execução para. Quando uma parte P_i completa a ativação o ambiente $\mathscr{E}$ é ativado novamente. Sempre que o adversário $\mathscr{A}$ entrega uma mensagem a uma parte P_i, a parte é ativada em seguida, que ao se completar leva a nova ativação do ambiente $\mathscr{E}$. Se em alguma ativação o adversário $\mathscr{A}$ não entrega mensagem a alguma parte P_i, o ambiente $\mathscr{A}$ é ativado assim que o adversário $\mathscr{A}$ completar sua ativação. Note que este mecanismo permite que $\mathscr{E}$ e $\mathscr{A}$ troquem mensagens livremente usando suas fitas de entrada e saída, entre duas ativações de alguma parte P_i. A saída da execução do protocolo é a saída de $\mathscr{E}$, que sem perda de generalidade será um bit.

Por $REAL_{\pi,\mathscr{A},\mathscr{E}}(k,x,\vec{r})$ entende-se a saída do ambiente $\mathscr{E}$ interagindo com o adversário $\mathscr{A}$ e partes P_i executando o protocolo π com parâmetro de segurança k, entrada x, e entrada aleatória $\vec{r} = r_{\mathscr{E}}, r_{\mathscr{A}}, r_1 \ldots, r_n$ (x e $r_{\mathscr{E}}$ para $\mathscr{E}$, $r_{\mathscr{A}}$ para $\mathscr{A}$; r_i para P_i). Por $REAL_{\pi,\mathscr{A},\mathscr{E}}(k,x)$ entende-se a variável aleatória descrevendo $REAL_{\pi,\mathscr{A},\mathscr{E}}(k,x,\vec{r})$ quando $\vec{r}$ é escolhido uniformemente. Por $REAL_{\pi,\mathscr{A},\mathscr{E}}$ entende-se a ensemble de distribuições de probabilidade $\{REAL_{\pi,\mathscr{A},\mathscr{E}}(k,x)\}_{k\in\mathbf{N},x\in\{0,1\}^*}$.

4.3.3. Execução no modelo ideal

A segurança de um protocolo é estabelecida comparando sua execução no modelo real com a de um processo ideal que realiza a tarefa desejada. Ao processo ideal está associada uma *funcionalidade ideal* que captura, i.e. especifica, a funcionalidade desejada. A funcionalidade ideal é modelada como mais uma ITM que interage com o ambiente e com o adversário como descrito a seguir.

Especificamente, o processo ideal envolve uma funcionalidade ideal $\mathscr{F}$, um processo adversário ideal $\mathscr{S}$, um ambiente $\mathscr{E}$ com entrada x e um conjunto de partes *dummy* $\tilde{P}_1, \ldots, \tilde{P}_n$. Cada parte *dummy* $\tilde{P}_i$ é modelada por uma ITM fixa e simples: sempre que uma $\tilde{P}_i$ recebe uma entrada, o valor é repassado para $\mathscr{F}$, copiando da fita de entrada para

a fita de comunicação; quando ativado por uma mensagem vida de $\mathscr{F}$, , a mensagem é copiada para a fita de saída. $\mathscr{F}$ recebe mensagens das partes *dummy* lendo das suas fitas de comunicação. $\mathscr{F}$ passa informações para as partes *dummy* enviando para suas fitas de comunicação. O processo adversário ideal $\mathscr{S}$ tem privilégios idênticos ao adversário no modelo real, exceto que não lhe é permitido acesso ao conteúdo das mensagens enviadas por $\mathscr{F}$ às partes *dummy*. $\mathscr{S}$ também pode corromper as partes *dummy*, controlar suas atividades futuras e tomar conhecimento da informação que elas possuam internamente.

A ordem dos eventos no processo ideal é idêntica à que ocorre no modelo real, com uma exceção. No modelo ideal, se uma parte *dummy* $\tilde{P}_i$ é ativada por um valor vindo do ambiente $\mathscr{E}$, este valor é copiado para a fita de comunicação da parte e o ambiente $\mathscr{E}$ é ativado em seguida. Uma vez que $\mathscr{F}$ tenha completado sua ação (possivelmente enviando uma mensagem para $\mathscr{S}$ ou algum $\tilde{P}_j$), a parte $\tilde{P}_i$ é ativada novamente. Deve-se ressaltar que no modelo ideal não há comunicação entre as partes *dummy*. A única comunicação que ocorre são as transferências entre as partes *dummy* e a funcionalidade ideal $\mathscr{F}$.

Por $IDEAL_{\mathscr{F},\mathscr{S},\mathscr{E}}(k,x,\vec{r})$ entende-se a saída do ambiente $\mathscr{E}$ interagindo com o adversário $\mathscr{S}$ e partes P_i executando o protocolo $\mathscr{F}$ com parâmetro de segurança k, entrada x, e entrada aleatória $\vec{r} = r_{\mathscr{E}}, r_{\mathscr{S}}, r_1 \ldots, r_n$ (x e $r_{\mathscr{E}}$ para $\mathscr{E}$, $r_{\mathscr{S}}$ para $\mathscr{S}$; r_i para P_i). Por $IDEAL_{\mathscr{F},\mathscr{S},\mathscr{E}}(k,x)$ entende-se a variável aleatória descrevendo $IDEAL_{\mathscr{F},\mathscr{S},\mathscr{E}}(k,x,\vec{r})$ quando $\vec{r}$ é escolhido uniformemente. Por $IDEAL_{\mathscr{F},\mathscr{S},\mathscr{E}}$ entende-se a ensemble de distribuições de probabilidade $\left\{IDEAL_{\mathscr{F},\mathscr{S},\mathscr{E}}(k,x)\right\}_{k\in\mathbf{N},x\in\{0,1\}^*}$.

4.3.4. Implementação segura de uma funcionalidade ideal

Um protocolo ρ é uma *implementação segura* (ou *realização segura*) de uma funcionalidade ideal $\mathscr{F}$ se para todo adversário $\mathscr{A}$ executando no modelo real existe um adversário $\mathscr{S}$ executando no modelo ideal, tal que nenhum ambiente $\mathscr{E}$, para qualquer valor de entrada, não lhe é possível distinguir com probabilidade não desprezível se está interagindo com $\mathscr{A}$ e as partes que executam ρ no modelo real, ou se está interagindo com $\mathscr{S}$ e a funcionalidade ideal $\mathscr{F}$ no modelo ideal.

Desta forma, para o ambiente, executar o protocolo ρ equivale a interagir com a funcionalidade ideal $\mathscr{F}$.

Definição 4.3.1. *Sejam $\xi = \{X(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ e $\psi = \{Y(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ duas ensembles de distribuições de probabilidade sobre $\{0,1\}$. Diz-se que ξ e ψ são indistinguíveis, denotado por $\xi \overset{c}{\approx} \psi$, se para todo $c \in \mathbf{N}$ existe um $k_0 \in \mathbf{N}$ tal que*

$$|PrX(k,a) = 1| - |PrY(k,a) = 1| < k^{-c}$$

para todo $k > k_0$ e todo a.

Definição 4.3.2. *Sejam $k, c \in \mathbf{N}$, $\mathscr{F}$ uma funcionalidade ideal e π um protocolo executado por n partes. Diz-se que π é uma implementação segura de $\mathscr{F}$ se para todo adversário $\mathscr{A}$ existe um adversário ideal $\mathscr{S}$ tal que para qualquer ambiente $\mathscr{E}$ temos que*

$$REAL_{\pi,\mathscr{A},\mathscr{E}} \overset{c}{\approx} IDEAL_{\mathscr{F},\mathscr{S},\mathscr{E}}.$$

4.4. O Teorema da Composição

Para apresentar o Teorema da Composição, é preciso inicialmente definir o modelo híbrido. No modelo híbrido é definido um modelo de computação em que um protocolo pode ter acesso, chamar, uma funcionalidade ideal.

4.4.1. O modelo híbrido

O modelo híbrido é idêntico ao modelo real, porém com algumas características a mais. Além de trocar mensagens entre si, as partes podem enviar e receber mensagens de um número ilimitado de cópias da funcionalidade ideal $\mathscr{F}$. Cada cópia de $\mathscr{F}$ é identificada por um identificador de sessão (SID) único. Todas as mensagens enviadas a uma dada cópia ou recebidas dela carregam o SID correspondente, que é escolhido pelo protocolo executado pelas partes.

A comunicação entre as partes e cada uma das cópias de $\mathscr{F}$ se dá como no modelo ideal. Assim, uma vez que alguma parte envia uma mensagem para alguma cópia de $\mathscr{F}$, esta cópia é imediatamente ativada e lê a mensagem da fita da parte. Além disso, neste modelo, apesar do adversário ser responsável por entregar as mensagens vindas das cópias de $\mathscr{F}$ para as partes, ele não tem acesso ao conteúdo das mensagens. Deve-se ressaltar que o ambiente não tem acesso direto às cópias de $\mathscr{F}$. Isso decorre da definição de segurança envolvendo o modelo híbrido, que será baseada na incapacidade do ambiente de distinguir entre o modelo real e o híbrido.

Já se pode definir como uma chamada à funcionalidade ideal $\mathscr{F}$ é substituída por uma chamada ao protocolo. Seja π um protocolo no modelo híbrido, e seja ρ um protocolo é uma implementação segura de uma funcionalidade ideal $\mathscr{F}$, com respeito a alguma classe de adversários. O protocolo composto π^ρ é construído substituindo o código de cada ITM de π deforma que a primeira mensagem enviada a cada cópia de $\mathscr{F}$ é substituída por uma chamada a uma nova cópia de ρ, com novos valores aleatórios de entrada, e com o conteúdo da mensagem como entrada. Cada mensagem subsequente para a cópia de $\mathscr{F}$ é substituída pela cópia correspondente de ρ, com o conteúdo passado para ρ como nova entrada. Cada valor de saída gerado pela cópia de ρ é tratado como uma mensagem recebida da cópia correspondente de $\mathscr{F}$.

4.4.2. Enunciado do Teorema

Na sua forma geral, o Teorema da Composição diz que se ρ é uma implementação segura de $\mathscr{F}$ então a execução do protocolo composto π^ρ no modelo real "emula" uma execução de $\pi^\mathscr{F}$, em que π chama cópias de $\mathscr{F}$, no modelo híbrido. Assim, para todo adversário $\mathscr{A}$ no modelo real existe um adversário $\mathscr{H}$ no modelo híbrido tal que nenhum ambiente $\mathscr{E}$ consegue distingue com probabilidade não desprezível se está interagindo com $\mathscr{A}$ e π^ρ no modelo real ou com $\mathscr{H}$ e $\pi^\mathscr{F}$ no modelo híbrido.

Teorema 4.4.1. (Teorema da Composição) *Seja $\mathscr{F}$ uma funcionalidade ideal, e π um protocolo, que composto com $\mathscr{F}$ resulta em $\pi^\mathscr{F}$, no modelo híbrido. Seja ρ um protocolo que é uma implementação segura de $\mathscr{F}$. Então para todo adversário $\mathscr{A}$ no modelo real existe um adversário $\mathscr{H}$ no modelo híbrido tal que para qualquer ambiente $\mathscr{E}$ temos que*

$$REAL_{\pi^\rho,\mathscr{A},\mathscr{E}} \overset{c}{\approx} HIBRIDO_{\pi^\mathscr{F},\mathscr{H},\mathscr{E}}$$

Uma formulação mais específica do resultado geral afirma que se π é uma implementação segura de uma funcionalidade ideal $\mathcal{G}$. $\mathcal{G}$ composto com a funcionalidade $\mathcal{F}$ resulta em $\mathcal{G}^{\mathcal{F}}$ equivale a $\pi^{\mathcal{F}}$ no modelo híbrido. Seja ρ um protocolo que é uma implementação segura de $\mathcal{F}$. Então π^{ρ} é uma implementação segura de $\mathcal{G}^{\mathcal{F}}$ no modelo real.

Teorema 4.4.2. *Sejam $\mathcal{F}$ e $\mathcal{G}$ funcionalidades ideais. Seja π um protocolo de n partes que é uma implementação segura de $\mathcal{G}$ no modelo híbrido e seja ρ um protocolo de n partes que é uma implementação segura de $\mathcal{F}$. Então o protocolo π^{ρ} é uma implementação segura de $\mathcal{G}^{\mathcal{F}}$.*

A prova detalhada se encontra em [Canetti 2001]. Entretanto, as principais ideias serão apresentadas na sequência. Sejam π e ρ protocolos e $\mathcal{F}$ uma funcionalidade ideal, sendo ρ uma implementação segura de $\mathcal{F}$. Seja $\mathcal{A}$ o adversário para π^{ρ} no modelo real. Deseja-se construir um adversário $\mathcal{A}'$ para $\pi^{\mathcal{F}}$ no modelo híbrido tal que $\mathcal{E}$ não conseguirá diferenciar se está interagindo com π^{ρ} e $\mathcal{A}$ no modelo real ou com $\pi^{\mathcal{F}}$ e $\mathcal{A}'$ no modelo híbrido. Para isto, será usado um adversário $\mathcal{S}$, um simulador, que atual sobre uma única instância de $\mathcal{F}$. Essencialmente, $\mathcal{A}'$ executará uma versão simulada de $\mathcal{A}$, seguindo suas instruções. As interações de $\mathcal{A}$ com as várias instâncias de ρ serão simuladas usando várias instâncias de $\mathcal{S}$. Assim, $\mathcal{A}'$ fará o papel do ambiente para as várias instâncias de $\mathcal{S}$. A habilidade de $\mathcal{A}'$ conseguir informações as múltiplas instâncias de $\mathcal{S}$ fazendo o papel do ambiente, é o principal ponto da prova.

A validade da simulação como um todo é demonstrada reduzindo-a à validade de $\mathcal{S}$. O tratamento das diversas instâncias de $\mathcal{S}$ é feito usando um argumento híbrido, que define várias execuções híbridas, nas quais um certo número de instâncias de $\mathcal{F}$ são substituídas por instâncias de ρ. este argumento híbrido é possível pelo fato de $\mathcal{S}$ ser definido independentemente do ambiente. Assim, não muda sob os vários ambientes híbridos.

Na prova em si, a questão referente à construção de um adversário no modelo ideal para todo adversário no modelo real é simplificada. ao invés de se quantificar sobre todos os adversários possíveis, recorre-se à utilização de um adversário *dummy*. Desta forma, é suficiente exigir que o adversário ideal $\mathcal{S}$ seja capaz de simular este adversário *dummy*, que é bem simples. Este adversário *dummy* pode apenas receber todas as mensagens geradas pelo ambiente e repassá-las às partes que são destinatárias. Assim, o adversário *dummy*, dado seu pequeno poder, é o mais difícil de simular pois simulá-lo indica que se pode simular qualquer outro. Intuitivamente, o adversário *dummy* é o mais difícil de simular porque dá ao ambiente controle total sobre as comunicações, deixando pouco espaço de manobra para o simulador.

Especificamente, segue a descrição do adversário *dummy* $\mathcal{D}$. Quando ativado com uma mensagem *m* na sua fita de entrada, $\mathcal{D}$ passa *m* para o ambiente $\mathcal{E}$ (note que a mensagem já inclui a identidade do destinatário). Quando ativado com uma entrada (m, id, c) do ambiente $\mathcal{E}$, onde *m* é uma mensagem, *id* é um destinatário, e *c* um código, $\mathcal{D}$ entrega a mensagem *m* à parte *id* (o código *c* é usado no caso de não existir uma parte com identidade *id*). Assim, $\mathcal{D}$ é capaz de corromper partes quando instruído por $\mathcal{E}$, e coletar informação para *E*.

4.5. Comprometimento de Bit - Bit Commitment BC

Comprometimento (*commitment*) é uma das primitivas criptográficas mais importantes, sendo utilizado na construção de vários outros protocolos criptográficos como zero knowledge (por exemplo [Goldreich et al. 1987], [Brassard et al. 1988], [Damgård 1990]), protocolos gerais de avaliação de funções (por exemplo [Goldreich et al. 1987], [Galil et al. 1988]) e vários outros. Os protocolos de comprometimento em si tem sido objeto de vários estudos como por exemplo [Blum 1982], [Naor 1991], [Dolev et al. 2000], [Naor et al. 1998], [Beaver 1996], [Di Crescenzo et al. 1998], [Fischlin and Fischlin 2000].

A ideia básica da noção de comprometimento é bem simples: uma parte, a que se compromete, entrega ao receptor o equivalente digital de um envelope lacrado contendo um valor x. desse ponto em diante, a parte comprometida não pode alterar o valor dentro do envelope, e enquanto esta parte não passar ao receptor instruções para abrir o envelope, este não tem conhecimento algum sobre o valor de x. O primeiro requisito, que impede que a parte que se compromete altere o valor de x antes da abertura é referida como *binding* enquanto o requisito do receptor só poder tomar conhecimento do valor de x após a abertura pela parte comprometida é referida como *hiding*

Entretanto, a formalização dessa ideia intuitiva não é trivial. O tratamento normalmente dado [Goldreich 2000] se revela insuficiente para algumas aplicações, basicamente por tratar o comprometimento de forma isolada. Assim, em geral, não se garante a segurança quando um protocolo de comprometimento é usado como componente em outros protocolos, ou quando múltiplas de suas cópias executam concorrentemente.

A seguir, o conceito de comprometimento será utilizado para demonstrar duas aplicações do *framework* UC. Inicialmente, o conceito será descrito de forma abstrata como uma funcionalidade ideal. Será demonstrado que como definida, não é possível implementar seguramente (UC segura) a funcionalidade sem que se recorra a hipóteses adicionais. Na sequência, será construído um protocolo de comprometimento UC seguro que inclui uma hipótese de setup. Tanto a impossibilidade da implementação da funcionalidade ideal "pura´´ quanto à segurança UC do protocolo "real´´ são demonstradas.

4.5.1. A Funcionalidade Ideal $\mathscr{F}_{\mathbf{BC}}$

A funcionalidade ideal que captura os requisitos esperados sobre o comprometimento de bit é apresentada a seguir. Na funcionalidade é capturada a ideia de envelope presente na motivação inicial do BC. Como enunciada, a funcionalidade trata do caso de um comprometimento único, podendo ser facilmente estendida para lidar com o caso geral de múltiplos comprometimentos. Outro ponto a destacar é que a funcionalidade lida com o comprometimento de bit, podendo também ser estendida para o caso de *strings* aplicando o Teorema da Composição, sendo também possível estender a funcionalidade diretamente, sem uso do Teorema da Composição. Os protocolos de comprometimento de *string* resultantes seriam certamente realizações mais eficientes da funcionalidade proposta.

A Funcionalidade Ideal $\mathscr{F}_{\mathbf{BC}}$ procede como descrito a seguir. A fase de comprometimento é modelada com a funcionalidade inicialmente recebendo uma mensagem

$(\texttt{Commit}, sid, P_i, P_j, b)$ de P_i, a parte que se compromete. Onde o *sid* é um identificador de sessão usado para diferenciar entre as instâncias da funcionalidade que estejam executando simultaneamente, P_j é a parte receptora, e $b \in \{0,1\}$ é o valor com o qual P_i se comprometeu. Na resposta, a funcionalidade precisa dar conhecimento à outra parte P_n e ao adversário ideal $\mathscr{S}$ de que P_i se comprometeu a algum valor de b, associado À sessão *sid*. Isto é feito enviando a mensagem $(\texttt{Receipt}, sid, P_i, P_j)$, para P_j e $\mathscr{S}$. A fase de abertura é iniciada com P_i enviando a mensagem $(\texttt{Open}, sid, P_i, P_j)$ para $\mathscr{F}_{\mathbf{BC}}$, que por sua vez envia a mensagem $(\texttt{Open}, sid, P_i, P_j, b)$ para P_j e $\mathscr{S}$, completando a execução.

A figura abaixo resume a descrição da funcionalidade:

Funcionalidade Ideal $\mathscr{F}_{\mathbf{BC}}$

$\mathscr{F}_{\mathbf{BC}}$ é definida como segue, executando com as partes $P_1, \ldots, P_n$, na presença de um adversário $\mathscr{S}$:

1. Ao receber uma mensagem $(\texttt{Commit}, sid, P_i, P_j, b)$ de P_i, $b \in \{0,1\}$:
   - armazene o bit b e
   - envie a mensagem $(\texttt{Receipt}, sid, P_i, P_j)$, para P_j e $\mathscr{S}$.

   Ignore mensagens **Commit** subsequentes.
2. Ao receber uma mensagem $(\texttt{Open}, sid, P_i, P_j)$ de P_i, proceda da seguinte forma:
   - se o bit b estiver armazenado,
     - então envie a mensagem $(\texttt{Open}, sid, P_i, P_j, b)$ para P_j;
     - caso contrário, pare a execução.

Vale ressaltar que como definida, a funcionalidade $\mathscr{F}_{\mathbf{BC}}$ não permite a cópia de comprometimento. Suponha que A se compromete com valor x para alguma parte B, e que o protocolo permitisse que B se comprometesse com o mesmo valor para uma parte C, simplesmente repassando o conteúdo da mensagem enviada por A. Tal protocolo não seria uma realização segura da funcionalidade $\mathscr{F}_{\mathbf{BC}}$. este requisito pode parecer difícil de ser satisfeito pois B pode sempre atuar como '*man-in-the-middle*. Deve-se lembrar que as identidades são únicas uma vez que se assume que as comunicações são autenticadas.

4.5.2. Impossibilidade de segurança UC para BC sem hipótese de setup

Esta seção ilustra uma das aplicações do framework UC. Em particular será apresentada sua aplicação para análise de um protocolo, no caso o comprometimento de bit BC. Será mostrado nesta sessão que tal como definida, sem acessar alguma outra funcionalidade ideal, $\mathscr{F}_{\mathbf{BC}}$ não é UC segura. Ou seja, não existe protocolo que seja uma implementação segura sem envolver uma terceira parte na interação e permitir a execução com sucesso quando tanto o emissor e o receptor são honestos. A impossibilidade se mantem mesmo

sob um requisito mais fraco em que para qualquer adversário real e qualquer ambiente deve existir um adversário ideal, onde o simulador pode depender do ambiente.

Esta situação em que uma funcionalidade não pode chamar outra, será referida como modelo simples. Note que existem protocolos de BC que são UC no modelo simples, se fizer uso de terceiras partes, ou servidores, se a maioria destes se mantiver não corrompida. Isto segui do resultado geral de [C01] que mostra que praticamente qualquer funcionalidade pode ser realizada neste arranjo.

Diz-se que o protocolo π com n partes $P_1, \ldots, P_n$ é *bilateral* se apenas duas dentre as n partes trocam mensagens entre si. Um protocolo bilateral de comprometimento π *termina com respeito à execução* se com probabilidade não desprezível, o receptor honesto P_j aceita um comprometimento de um emissor honesto P_i e envia a mensagem $(\texttt{Receipt}, sid, P_i, P_j)$; além disso, quando o receptor honesto recebe uma mensagem válida de abertura do emissor honesto, com valor m e identificador de sessão sid, ele envia a mensagem $(\texttt{Open}, sid, P_i, P_j, m)$ com probabilidade não desprezível.

Teorema 4.5.1. *Não existe protocolo bilateral de comprometimento π convergente com respeito à execução que seja uma implementação segura da funcionalidade $\mathcal{F}_{\mathbf{BC}}$ no modelo simples, mesmo que se permita que o adversário ideal $\mathcal{S}$ dependa do ambiente $\mathcal{E}$.*

Prova: A ideia da prova é a que segue. Considere uma execução no modelo real do protocolo entre uma parte P_i que se compromete com um valor b e o receptor P_j. Suponha que P_i foi corrompido pelo adversário $\mathcal{A}$ e é por ele controlado, enquanto o receptor P_j se mantem honesto. Assuma também que o adversário $\mathcal{A}$ apenas envia as mensagens geradas pelo ambiente $\mathcal{E}$ e repassa a este as mensagens enviadas a P_i. No início do processo, o ambiente escolhe aleatoriamente um bit b, mantendo seu valor em segredo, e gera as mensagens para P_i executando o protocolo para uma parte honesta, que se compromete com b, e as respostas de P_j.

Para simular o comportamento acima descrito, no modelo ideal, o adversário $\mathcal{S}$ deve ser capaz de prover à funcionalidade ideal um valor para o bit comprometido. Ou seja, o ideal $\mathcal{S}$ deve extrair o bit das mensagens geradas pelo ambiente, sem que "rebobine", i.e. volte o estado de execução do ambiente. Entretanto, será mostrado, se o esquema de comprometimento de bit, permite que o simulador extraia com sucesso o bit comprometido, então o esquema não é seguro, pois um receptor corrompido conseguiria obter o valor do bit comprometido interagindo com um emissor honesto.

Mais precisamente, no modelo real, seja o protocolo bilateral π executado com emissor P_i e receptor P_j. Considere o ambiente $\mathcal{E}$ e o adversário real $\mathcal{A}$. No início da execução o adversário $\mathcal{A}$ corrompe P_i, a parte que se compromete no protocolo. Assim, $\mathcal{A}$ faz com que P_i envie toda mensagem recebida de $\mathcal{E}$ e reporta toda resposta recebida por P_j de $\mathcal{E}$.

O ambiente $\mathcal{E}$ escolhe aleatoriamente um bit b, mantendo seu valor em segredo, e gera as mensagens para P_i executando o protocolo para uma parte honesta, que se compromete com b e as respostas de P_j e segue o programa do emissor honesto que se compromete com b, conforme especificado por π. Uma vez que o receptor honesto envia a mensagem `Receipt`, o ambiente faz com que $\mathcal{A}$ abra o comprometimento com valor b.

Uma vez que o receptor envia a mensagem $(\texttt{Open}, sid, P_i, P_j, b')$, o ambiente gera o valor 1 se $b = b'$ e 0 caso contrário.

Já no modelo ideal, uma vez que o receptor envia uma mensagem **Receipt** antes de se iniciar a fase de abertura, o adversário ideal $\mathcal{S}$ para o para $\mathcal{A}$, $\mathcal{E}$ deve enviar a mensagem $(\texttt{Commit}, sid, P_i, P_j, b)$ para funcionalidade ideal $\mathcal{F}_{\mathbf{BC}}$ *antes* de conhecer o valor do bit comprometido b no final da fase de abertura. Entretanto, o receptor honesto envia o valor b' recebido de $\mathcal{F}_{\mathbf{BC}}$ na fase de abertura. Isso implica que, para ter sucesso, $\mathcal{S}$ deve obter o valor correto do bit comprometido b já na fase inicial de comprometimento, o que viola o sigilo do protocolo de comprometimento.

Formalizando, suponha que existe um adversário ideal $\mathcal{S}$ tal que:

$$REAL_{\pi,\mathcal{A},\mathcal{E}} \overset{c}{\approx} IDEAL_{\mathcal{F},\mathcal{S},\mathcal{E}}$$

Será construído um novo ambiente $\mathcal{E}'$ e um novo adversário real $\mathcal{A}'$ para o qual não existe um adversário ideal apropriado para π. nessa nova situação, $\mathcal{A}'$ corrompe o receptor P_j no começo da execução do protocolo. Durante a execução, $\mathcal{A}'$ obtém as mensagens do emissor honesto P_i, passando-as a uma cópia virtual de $\mathcal{S}$. As respostas de $\mathcal{S}$, como se geradas pelo receptor honesto, são encaminhadas a P_i em nome da parte corrompida P_j. Em dado momento, $\mathcal{S}$ envia a mensagem de comprometimento $(\texttt{Commit}, sid, P_i, P_j, b')$ para $\mathcal{F}_{\mathbf{BC}}$. Então o adversário $\mathcal{A}'$ gera o valor de saída b' e para.

O ambiente $\mathcal{E}'$ ordena que a parte honesta se comprometa a um bit b escolhido aleatoriamente e cujo valor é mantido em segredo. Note que em nenhum momento se dá a abertura. O ambiente $\mathcal{E}'$ então conclui gerando o valor 1 se e sós se o valor de saída b' gerado pelo adversário $\mathcal{A}'$ satisfaz $b = b'$.

Pela propriedade de terminação definida acima, obtém-se do simulador $\mathcal{S}$ um bit b' com probabilidade não desprezível. Este bit b' é uma boa aproximação do bit b, já que $\mathcal{S}$ simula o protocolo π com erro desprezível. Consequentemente, a probabilidade do bit b gerado por $\mathcal{A}'$ estar correto é 1/2 mais um valor não desprezível. Porém, é impossível que um adversário ideal $\mathcal{S}'$ consiga acertar o valor de b probabilidade não desprezível maior que 1/2, uma vez que a visão de $\mathcal{S}'$ no modelo ideal é estatisticamente independente do bit b, lembrando que o comprometimento com b não é aberto. □

4.6. Um protocolo UC seguro para comprometimento de bit

Como estabelecido na seção anterior, vimos a impossibilidade de implementação segura (UC segura) da funcionalidade ideal $\mathcal{F}_{\mathbf{BC}}$ por um protocolo que não recorra a pelo menos mais outra funcionalidade. A seguir será apresentada um funcionalidade que auxiliará na construção de um protocolo para BC que é UC seguro, seguido de esquema de comprometimento, que será demonstrado ser UC seguro.

4.6.1. A Funcionalidade Ideal $\mathcal{F}_{\mathbf{CRS}}$

A funcionalidade que será utilizada juntamente com $\mathcal{F}_{\mathbf{BC}}$ para a construção do protocolo UC seguro é descrita a seguir. A funcionalidade ideal $\mathcal{F}_{\mathbf{CRS}}$, onde CRS é uma *string* comum de referência, consiste na distribuição de uma *string* entre todas as partes envolvidas. A *string* é distribuída antes do início da interação entre as partes, sendo escolhida

segundo uma distribuição de probabilidade especificada.

Funcionalidade Ideal $\mathcal{F}_{\mathbf{CRS}}$

$\mathcal{F}_{\mathbf{CRS}}$ é definida como segue, parametrizada por uma distribuição D:

1. Quando ativada pela primeira vez, com entrada (valor, *sid*):
   - escolha um valor $d \xleftarrow{R} D$
   - e envie d para a parte que a ativou.
2. Nas ativações seguintes, envie d para a parte que a ativou.

Note que na definição acima, a funcionalidade está parametrizada por uma distribuição de probabilidade D. Como definida a funcionalidade $\mathcal{F}_{\mathbf{CRS}}$ já está adequada ao modelo híbrido, onde se pode ter acesso à funcionalidade ideal.

Como definida, $\mathcal{F}_{\mathbf{CRS}}$ tem algumas características:

- no modelo real, as partes tem acesso a uma *string* comum e pública que é escolhida previamente de acordo com uma distribuição especificada pelo protocolo executado pelas partes.
- no modelo ideal, pode ser que uma funcionalidade não faça uso de $\mathcal{F}_{\mathbf{CRS}}$, como é o caso de $\mathcal{F}_{\mathbf{BC}}$. assim, um adversário ideal que opere simulando um adversário real pode fazer as vezes de $\mathcal{F}_{\mathbf{CRS}}$ para o adversário simulado. Isto significa que o adversário ideal pode escolher a *string* comum arbitrariamente.

Vale ressaltar que por não usar o *string* comum no modelo ideal, a validade do que acontece no modelo ideal não é afetada já que $\mathcal{F}_{\mathbf{CRS}}$ estaria executando no modelo híbrido. Assim, a validade da noção de segurança se mantem.

Do ponto de vista da composição, é preciso deixar claro alguns aspectos acerca de $\mathcal{F}_{\mathbf{CRS}}$. Cada cópia de ρ no protocolo composto π^{ρ} deve ter sua própria cópia da *string* de referência, i.e. chama uma instância própria de $\mathcal{F}_{\mathbf{CRS}}$. Se isto não for o caso, o Teorema da Composição não poderá ser aplicado.

4.6.2. O Esquema de Comprometimento de Bit com hipótese de setup

O protocolo apresentado a seguir é uma implementação segura da funcionalidade $\mathcal{F}_{\mathbf{BC}}$ se baseia em permutação trapdoor e utiliza cada *string* comum uma única vez, em um único comprometimento. A construção se baseia no esquema proposto em [Di Crescenzo et al. 1998], que por sua vez é uma modificação do esquema apresentado em [Naor 1991].

A ideia do protocolo é a que segue. Seja G um gerador pseudo aleatório que expande n bits de entrada em $4n$ bits de saída. Para o parâmetro de segurança n, o receptor

envia um *string* aleatório σ ao emissor, que escolhe um $r \in \{0,1\}^n$ que computa $G(r)$ e retorna $G(r)$ ou $G(r) \oplus \sigma$ para se comprometer com 0 ou 1 respectivamente. Na abertura, o emissor envia b e r. Pela pseudo aleatoriedade de G o receptor não pode distinguir os dois casos e com probabilidade 2^{-2n} sobre a escolha de r é impossível encontrar r_0 e r_1 tais que:

$$G(r_0) = G(r_1) \oplus \sigma$$

Em [Di Crescenzo et al. 1998] uma versão equivocável do esquema em [Naor 1991] foi proposta. Suponha que σ não é escolhida pelo receptor mas que seja uma comum aleatória. Então, se $\sigma = G(r_0) \oplus G(r_1)$ para r_0 e r_1 aleatórios, e o emissor entrega $G(r_0)$ ao receptor, será mais fácil abrir o comprometimento como 0 para r_0 e 1 para r_1, já que $G(r_0) \oplus \sigma = G(r_1)$. Por outro lado, a escolha de σ daquela forma é indistinguível de uma verdadeira escolha aleatória.

O protocolo UC seguro π_{BC}^{CRS}, descrito a seguir, parte da ideia do esquema acima usando um gerador pseudo aleatório com uma propriedade *trapdoor*. Será usado um gerador Blum-Micali-Yao, porém com permutações *trapdoor* ao invés de permutações *one-way* [Yao 1982] [Blum and Micali 1984]. Seja *KGen* um algoritmo eficiente que com entrada 1^n gera uma chave pública *pk* e um *trapdoor td*. A chave *pk* descreve uma permutação *trapdoor* f_{pk} sobre $\{0,1\}^n$.

Antes de definir o gerador, algumas definições iniciais são necessárias. O primeiro conceito é o de função one-way, que captura a noção intuitiva de uma função não inversível.

Definição 4.6.1. *Uma função $f : \{0,1\}^* \to \{0,1\}^*$, é one-way se:*

- *existe um algoritmo que para a entrada x computa $f(x)$ em tempo polinomial*
- *para todo algoritmo PPT $\mathscr{A}$, a probabilidade de $\mathscr{A}$ ser a inversa de $f(x)$*

$$Pr[f(A(f(x))) = f(x)]$$

é desprezível.

Outro conceito necessário é o de permutação trapdoor, que captura a noção de uma função one-way que com alguma informação adicional se torna fácil inverter.

Definição 4.6.2. *Uma permutação trapdoor é definida pela tripla (G,E,D) tal que:*

1. *G é um algoritmo que para a entrada 1^k gera um par de chaves (pk,td)*
2. *E é um algoritmo determinístico tal que para todo pk, o mapeamento $x \to E(pk,x)$ é uma bijeção*
3. *D é um algoritmo determinístico tal que para todos os pares de chaves possíveis (pk,td) gerados por G, e para todo x:*

$$D(td,E(pk,x)) = x \tag{1}$$

Definição 4.6.3. *Um predicado hardcore B(.) de uma função one-way $f(x)$, é uma função cuja saída é um único bit para qual não existe algoritmo PPT que compute $B(f(x))$ com probabilidade significativamente maior que 1/2 sobre uma escolha aleatória de x.*

Assim, já se pode definir o gerador[1].

Definição 4.6.4. *Seja $B(.)$ um predicado hardcore para f_{pk}. Define-se o gerador que expande n bits de entrada em $4n$ bits de saída com a descrição pública pk como:*

$$G_{pk}(r) = (B(f_{pk}^{(4n)}(r)), B(f_{pk}^{(4n-1)}(r)), \ldots, B(f_{pk}(r)), B(r))$$

onde $f_{pk}^{(i)}(r)$ é a i-ésima aplicação de f_{pk} a r.

Uma importante propriedade deste gerador é que dado um *trapdoor td* para *pk* é fácil dizer se um dado $y \in \{0,1\}^{4n}$ está na imagem de G_{pk}. O *string* público aleatório no esquema π_{BC}^{CRS} consiste em um *string* aleatório de $4n$ bits, junto com duas chaves públicas pk_0 e pk_1 que descrevem os geradores pseudo aleatórios *trapdoor* G_{pk_0} e G_{pk_1}, que expandem n bits de entrada em $4n$ bits de saída. As chaves públicas pk_0 e pk_1 são geradas por duas execuções independentes do algoritmo de geração de chaves *KGen* com entrada 1^n, juntamente com os *trapdoors* correspondentes td_0 e td_1 respectivamente.

Para se comprometer com um bit $b \in \{0,1\}$, o emissor escolhe um *string* aleatório $b \in \{0,1\}^n$, calcula $G_{pk_b}(r)$ e $y = G_{pk_b}(r)$ se $b = 0$, ou $y = G_{pk_b}(r) \oplus \sigma$ se $b = 1$. Na abertura, o emissor envia (b,r) ao receptor, que verifica se $y = G_{pk_b}(r)$ para $b = 0$, ou se $y = G_{pk_b}(r) \oplus \sigma$ para $b = 1$.

Claramente, o esquema abaixo satisfaz as condições de *hiding* e *binding*: a primeira computacionalmente e a segunda estatisticamente. Em uma simulação, σ assume o valor $\sigma = G_{pk_0}(r_0) \oplus G_{pk_1}(r_1)$. Assim, transmitindo $y = G_{pk_0}(r_0)$ posteriormente pode-se abrir esse valor com 0 enviando r_0 e com 1 enviando r_1.

Além disso, caso se conheça td_0 e pk_0, um *string* y^* gerada pelo adversário pode ser checada se está na imagem de G_{pk_0} representando um comprometimento com o valor 0. A menos que y^* esteja na imagem de G_{pk_0} e simultaneamente $y^* \oplus \sigma$ pertença à imagem de G_{pk_1}, o valor extraído do bit comprometido é único e estará correto com respeito a td_0.

[1] para maiores detalhes e uma definição formal de geradores pseudo aleatórios vide apêndice

Esquema de Comprometimento de Bit com hipótese de setup - π_{BC}^{CRS}

- *string pública:*
  - σ - *string* aleatória em $\{0,1\}^{4n}$
  - pk_0, pk_1 - chaves para os geradores $G_{pk_0}, G_{pk_1} : \{0,1\}^n \to \{0,1\}^{4n}$
- *comprometimento para o bit $b \in \{0,1\}$ com SID sid:*
  - emissor P_i
    - calcula $G_{pk_b}(r)$
    - atribui $y = G_{pk_b}(r)$ se $b = 0$, ou $y = G_{pk_b}(r) \oplus \sigma$ se $b = 1$
    - envia mensagem $(\texttt{Commit}, sid, P_i, P_j, y)$ para o receptor P_j
  - receptor P_j
    - ao receber a mensagem $(\texttt{Commit}, sid, P_i, P_j, y)$ do emissor P_i, envia de volta a mensagem $(\texttt{Receipt}, sid, P_i, P_j, b)$
- *abertura para y*
  - emissor P_i
    - envia (b, r) ao receptor
  - emissor P_i
    - verifica se $y = G_{pk_b}(r)$ para $b = 0$, ou se $y = G_{pk_b}(r) \oplus \sigma$ para $b = 1$
    - se a verificação tem sucesso, gera a mensagem $(\texttt{Open}, sid, P_i, P_j, b)$

4.6.3. Segurança UC do protocolo BC com CRS - π_{BC}^{CRS}

Como visto, o esquema π_{BC}^{CRS} suporta a equivocabilidade e a extratibilidade. Já se pode então estabelecer que o esquema de protocolo π_{BC}^{CRS} é UC seguro, i.e. π_{BC}^{CRS} é uma implementação segura da funcionalidade ideal $\mathscr{F}^{\mathbf{BC}}$.

Teorema 4.6.1. *O protocolo π_{BC}^{CRS} é uma implementação segura da funcionalidade ideal $\mathscr{F}^{\boldsymbol{BC}}$ acessando a funcionalidade ideal $\mathscr{F}^{\boldsymbol{CRS}}$.*

Prova: A prova está dividida em partes. Inicialmente, será descrito o adversário ideal, o simulador $\mathscr{S}$. Definido $\mathscr{S}$, será tratada a questão da indistinguibilidade, que compreende três situações distintas. As questões relativas à indistinguibilidade serão reduzidas à distinção entre uma sequência realmente aleatória e outra pseudo aleatória.

Simulação

O adversário ideal $\mathscr{S}$ é descrito a seguir. $\mathscr{S}$ executa juntamente com $\mathscr{E}$ e, em para-

lelo executa uma cópia virtual do adversário real $\mathscr{A}$, que é executado como uma caixa preta. Isto é, $\mathscr{S}$ age como uma interface entre $\mathscr{A}$ e $\mathscr{E}$ emulando uma cópia de uma execução real de π_{BC}^{CRS} para $\mathscr{A}$, incorporando as interações de $\mathscr{E}$ no modelo ideal e, vice-versa, encaminhando as mensagens de $\mathscr{A}$ para $\mathscr{E}$.

1. No início, o simulador $\mathscr{S}$ prepara σ selecionando os pares de chaves

$$\begin{aligned}(pk_0, td_0) &\leftarrow KGen(1^n),\\ (pk_1, td_1) &\leftarrow KGen(1^n)\end{aligned}$$

   atribuindo a σ o valor

$$\sigma = G_{pk_0}(r_0) \oplus G_{pk_1}(r_1)$$

   para $r_0, r_1 \in \{0,1\}^n$ aleatórios.

   Esta *string* σ será chamada de *falsa* com respeito aos valores $pk_0, pk_1, G_{pk_0}(r_0)$ e $G_{pk_1}(r_1)$.

   Na sequência, inicia a execução da simulação de $\mathscr{A}$ e da execução com $\mathscr{E}$ usando a *falsa* σ e pk_0, pk_1.

2. Se em algum ponto da execução:

   - o ambiente $\mathscr{E}$ escreve a mensagem $(\texttt{Commit}, sid, P_i, P_j, b)$ na fita da parte não corrompida $\tilde{P}_i$,
   - e este a copia para a funcionalidade ideal $\mathscr{F}_{\mathbf{BC}}$.

   Então o simulador $\mathscr{S}$, que não pode ler o valor do bit mas toma conhecimento de que houve um comprometimento recebendo a mensagem $(\texttt{Receipt}, sid, P_i, P_j)$,

   - informa $\mathscr{A}$ que P_i enviou a P_j

$$y = G_{pk_0}(r_0)$$

   .

3. Se em algum ponto da execução:

   - o ambiente $\mathscr{E}$ ordena que a parte não corrompida $\tilde{P}_i$ realize a abertura
   - e esta parte, corretamente, se comprometeu com algum bit b, cujo valor foi mantido secreto

   Então o adversário ideal $\mathscr{S}$ deve enviar o valor

$$y = G_{pk_0}(r_0)$$

   fazendo as vezes da parte P_i na simulação (em modo caixa preta) do adversário $\mathscr{A}$.

4. Se o adversário $\mathscr{A}$ simulado

- permite que alguma parte corrompida P_i envie a mensagem $(\texttt{Commit}, sid, y^*)$ a alguma parte honesta P_j
- então $\mathscr{S}$ verifica com o auxílio do *trapdoor* td_0 se y^* está na imagem de G_{pk_0} ou não.

Se for o caso,

- envia a mensagem $(\texttt{Commit}, sid, P_i, P_j, 0)$ para a funcionalidade ideal $\mathscr{F}_{\mathbf{BC}}$
  - como se fosse a parte P_i, para a funcionalidade ideal $\mathscr{F}_{\mathbf{BC}}$; caso contrário, $\mathscr{S}$ envia a mensagem $(\texttt{Commit}, sid, P_i, P_j, 1)$

5. Se $\mathscr{A}$ ordena:

   - que alguma parte corrompida P_i realize a abertura com y^* válido e bit b^* correto,
   - então $\mathscr{S}$ compara com o valor de b^* ao valor previamente extraído e para a execução se eles diferem.
   - caso contrário,
     - $\mathscr{S}$ envia a mensagem $(\texttt{Open}, sid, P_i, P_j)$ em nome de P_i para a funcionalidade ideal $\mathscr{F}_{\mathbf{BC}}$.

   Se P_i precisar realizar a abertura com um valor incorreto, $\mathscr{S}$ também enviará para a funcionalidade uma mensagem de abertura, mesmo com o valor incorreto.

6. Sempre que o simulador $\mathscr{A}$ ordenar que uma parte seja corrompida:

   - $\mathscr{S}$ corrompe esta parte no modelo ideal e toma conhecimento de todas as suas informações internas.

   Assim:

   - $\mathscr{S}$ pode adaptar qualquer possível informação de abertura sobre um comprometimento que ainda não aberto daquela parte, como no caso de uma parte honesta realizando a abertura.

   Depois disso,

   - $\mathscr{S}$ passa toda informação ajustada para $\mathscr{A}$.

Indistinguibilidade

Para mostrar que a saída gerada pelo ambiente no modelo real é indistinguível daquela gerada no modelo ideal, três situações devem ser consideradas sendo associada a cada uma delas uma variável aleatória:

Modelo Real com *string* genuino:

A saída gerada por $\mathscr{E}$ é resultado da execução no modelo real com as partes P_i executando o protocolo na presença do adversário $\mathscr{A}$. Desta forma, σ é escolhido segundo uma distribuição uniforme e pk_0, pk_1 aleatórios são obtidos executando *KGen*, sendo usados como a *string* pública; então o protocolo é executado no modelo real com $\mathscr{A}$ e $\mathscr{E}$ com esse *string*. A esta situação está associada a variável aleatória R_g.

Modelo Real com *string* falso:

A saída gerada por $\mathscr{E}$ é resultado da execução no modelo real. O *string* falso σ é escolhido juntamente com os valores aleatórios pk_0, pk_1 como no simulador, envolvendo os valores previamente selecionados de $G_{pk_0}(r_0)$ e $G_{pk_1}(r_1)$. O protocolo é executado no modelo real com $\mathscr{A}$ e $\mathscr{E}$ usando o *string* falso. Se uma parte honesta for se comprometer com um bit b, ela deve computar o comprometimento usando os valores previamente selecionados: $y = G_{pk_b}(r)$ para $b = 0$, e $y = G_{pk_b}(r) \oplus \sigma$ para $b = 1$. se mais adiante a parte honesta for realizar a abertura, ela deverá fazê-lo enviando b e r_b. Ao final da execução, a saída será que o ambiente $\mathscr{E}$ vier a gerar. A esta situação está associada a variável aleatória R_f.

Modelo Ideal com *string* falso:

A saída gerada por $\mathscr{E}$ é resultado da execução no modelo ideal com $\mathscr{S}$ e $\mathscr{F}^{\mathbf{BC}}$, usando um *string* falso escolhido por $\mathscr{S}$. A esta situação está associada a variável aleatória I_f.

A questão da indistinguibilidade se reduz a estabelecer a indistinguibilidade entre as variáveis aleatórias R_g e R_f, e entre R_f e I_f. Isto estabelecido, por transitividade temos a indistinguibilidade entre R_g e I_f. Em ambos os casos, o problema vai se reduzir a distinguir entre uma sequência aleatória e uma pseudo aleatória.

Indistinguibilidade entre R_g e R_f

Inicialmente, será assumido que o ambiente $\mathscr{E}$ distingue R_g de R_f, com probabilidade não desprezível. Como será visto mais à frente, vai se chegar a uma contradição. A partir dessa hipótese, será construído um algoritmo que decide se uma entrada é verdadeiramente aleatória ou pseudo aleatória.

Dados um parâmetro de segurança n, uma chave pública aleatória pk de um gerador *trapdoor* pseudo aleatório $G_{pk} : \{0,1\}^n \to \{0,1\}^{4n}$ juntamente com um *string* $x \in \{0,1\}^{4n}$, que tanto pode ser escolhido aleatoriamente ou ser gerado por G_{pk}. Deseja-se decidir como x foi gerado.

Para um x aleatório de um x pseudo aleatório, o ambiente $\mathscr{E}$ será usado, distinguindo entre R_g e R_f. Para isto, um *string* σ é gerado por procedimento similar ao de $\mathscr{S}$, mas usado o dado *string* x. Então, emula-se a execução no modelo real simulando o cenário em que todas as partes são honestas, lendo todas as mensagens enviadas por $\mathscr{E}$. Especificamente:

- geração do *string* público:

  - um bit c é escolhido aleatoriamente
  - faz-se $pk_{1-c} = pk$ para uma dada chave pública
    * o bit c será o palpite para o bit ao qual uma parte honesta se compromete
  - um novo par de chaves é gerado $(pk_c, td_c) \leftarrow KGen(1^n)$
  - $r_c \{0,1\}^n$ é escolhido aleatoriamente
  - faz-se $\sigma = G_{pk_c}(r_c) \oplus x$

- emulação:
  - a simulação do protocolo é iniciada com $\mathscr{A}$ e $\mathscr{E}$ usando σ, pk_0, pk_1
  - se $\mathscr{E}$ manda que uma parte não corrompida P_i se comprometa com um bit b,
    * se $b \neq c$, então a execução para imediatamente com saída 0;
    * no caso em que $b = c$, $G_{pk_c}(r_c)$ é enviado se $b = c = 0$,
    * e caso contrário, $b = c = 1$, x é enviado em nome de P_i e a simulação continua;
  - quando mais tarde $\mathscr{E}$ mandar P_i abrir o comprometimento
    * b (que é igual a c) e r_c são enviados
  - de forma similar,
    * no caso da parte P_i ser corrompida, b e r_c são enviados para $\mathscr{A}$ antes da abertura
  - se o adversário $\mathscr{A}$ corrompeu a parte P_i antes desta se comprometer
    * a execução para com probabilidade $1/2$ (tem-se assim simetria com o caso anterior, simplificando a análise);
  - caso contrário,
    * a simulação continua.

- saída:
  - se a execução ainda não parou, a saída de $\mathscr{E}$ é copiada.

Para analisar a vantagem do algoritmo acima, considere-se primeiro o caso em que x é um *string* de $4n$ bits, uniformemente distribuído. Então σ também é aleatório e a predição c não passa nenhuma informação para $\mathscr{A}$ e $\mathscr{E}$ no início da execução. Assim, a probabilidade de uma parada prematura com saída 0 é $1/2$, independente de $\mathscr{A}$ estar dominando o comprometimento, ou deste ser efetuado por uma parte honesta. Condicionada a terminação a se dar no último estágio, a saída de $\mathscr{E}$ tem distribuição idêntica à de R_g.

Seja x agora gerado por G_{pk_c}. Nesse caso, σ corresponde a um *string* falso. Aqui também, o *string* público não revela nada sobre c para $\mathscr{A}$ e $\mathscr{E}$. Conclui-se novamente que a parada prematura ocorre com probabilidade $1/2$, independente de quem realiza o comprometimento. Além disso, dado que se atinge o estágio final, a saída de $\mathscr{E}$ tem distribuição idêntica à de R_f.

Consequentemente, em ambos os experimentos, R_g e R_f, a saída é 1 com metade da probabilidade de $\mathscr{E}$ retornar 1. Segue que se a vantagem de $\mathscr{E}$ distinguir entre R_g e R_f é dada por (onde $\mathtt{Out}(\mathscr{E},X)$ é o valor da saída gerada por $\mathscr{E}$ no experimento X):

$$\varepsilon(n) = \left| Prob\left[\mathtt{Out}(\mathscr{E},R_g) = 1\right] - Prob\left[\mathtt{Out}(\mathscr{E},R_f) = 1\right]\right|$$

assim a vantagem em distinguir entre entradas aleatórias e pseudo aleatórias é igual a $\varepsilon(n)/2$. Se $\varepsilon(n)$ não for desprezível, $\varepsilon(n)/2$ também não é. Entretanto, isto contradiz a pseudo aleatoriedade do gerador.

Indistinguibilidade entre R_f e I_f

Excetuando-se o caso em que o adversário envia algum y^* na imagem de G_{pk_0} para depois abrir o comprometimento com $b^* = 1$, os dois experimentos são idênticos. Assim, é suficiente estimar a probabilidade deste erro acontecer. Como se verá, esta probabilidade é desprezível, dada a pseudo aleatoriedade dos geradores.

Suponha que no experimento R_f a probabilidade de $\mathscr{A}$ comprometer uma parte corrompida usando y^*, tal que y^* e $y^* \oplus \sigma$ estão nas imagens de G_{pk_0} e G_{pk_1}, respectivamente, seja não desprezível. Constrói-se o algoritmo que segue.

As entradas são n, uma chave pública pk, e um *string* x de $4n$ bits. A saída é um bit indicando se x é aleatório ou pseudo aleatório. O algoritmo especifica:

1. faça:
   - $pk_1 = pk$
   - gere outro par de chaves (pk_0, td_0)
   - defina $\sigma = G_{pk_0}(r_0) \oplus x$ para $r_0 \in \{0,1\}^n$
2. emule o experimento I_f com $\mathscr{S},\mathscr{E}$ usando σ, pk_0, pk_1
   - se uma parte honesta for instruída a se comprometer, aborte
3. Se $\mathscr{A}$ instrui uma parte corrompida a se comprometer usando y^*,
   - verificar, usando td_0, se y^* está na imagem de G_{pk_0}
   - se a parte corrompida também faz uma abertura correta de y^* para $b = 1$
     - pare com saída 1.
4. para todos os outros casos,
   - retorne 0.

Observe que o algoritmo retorna 1 se a verificação com td_0 leva a um r_0^* na imagem de G_{pk_0} e se o adversário também revela r_1^* tal que

$$G_{pk_0}(r_0^*) = y^* = G_{pk_1}(r_1^*) \oplus \sigma = G_{pk_1}(r_1^*) \oplus G_{pk_0}(r_0) \oplus x$$

mas para x aleatório a probabilidade de

$$x \in \left\{ G_{pk_0}(r_0) \oplus G_{pk_0}(r_0^*) \oplus G_{pk_1}(r_1^*) |, r_0, r_0^*, r_1^* \in \{0,1\}^n \right\}$$

é no máximo 2^{-n}. Assim, nesse caso, o algoritmo dá saída 1 apenas com probabilidade exponencialmente pequena. Por outro lado, se x é pseudo aleatório então o algoritmo retorna 1 com a mesma probabilidade que o adversário $\mathscr{A}$ incorre na situação descrita para o experimento I_f. Por hipótese, essa probabilidade é não desprezível. Logo, a vantagem geral do algoritmo é não desprezível também, refutando a pseudo aleatoriedade do gerador.

Assim, a prova está concluída. □

4.7. Conclusão

O *framework* UC, conforme apresentado, é uma ferramenta poderosa para o projeto e análise de protocolos criptográficos. O framework proporciona o ferramental para a representação dos protocolos e dos seu requisitos de segurança específicos, relacionados com as tarefas que se pretendem realizar. O conceito de segurança UC, aqui expresso como implementação segura captura a motivação principal de viabilizar o projeto modular de protocolos. O Teorema da Composição sumariza todo o esforço de formalização do *framework*, de certa forma validando a definição de segurança UC. Sua prova, apesar de não ter sido apresentada, serve como um *template* para a prova de segurança UC de protocolos específicos.

Como vimos o *framework* UC inicia sua construção modelando o conceito de protocolo a partir de ITMs, evoluindo para sistemas de ITMs e definindo um modelo de execução. Nesse modelo, além se contemplar a modelagem das chamadas de subrotina, ponto básico para o tratamento da composição, tem-se juntamente com a execução do protocolo, a sua interação com o ambiente e com o adversário.

Para estabelecer o Teorema da Composição são definidos dois refinamentos ao modelo de execução, que incluem as regras de trocas de mensagens entres as partes. Estes refinamentos levam aos modelos real e ideal, que essencialmente diferem quanto ao acesso que o adversário tem aos canais de comunicação. Além disso, a separação entre a funcionalidade ideal, que encapsula de forma abstrata a tarefa a ser realizada, e o protocolo real, que pretende implementar concretamente a funcionalidade desejada, é mais um ponto esclarecedor no tratamento dado ao projeto de protocolos.

Para exemplificação da utilização do framework, que foi o foco principal desta abordagem, foi escolhido o comprometimento de bit. O conceito de comprometimento foi utilizado para demonstrar duas aplicações do *framework* UC. Inicialmente, descrito como uma funcionalidade ideal, foi demonstrado que como definida, não é possível implementar seguramente (UC segura) a funcionalidade sem que se recorra a hipóteses adicionais. Na sequência, foi construído um protocolo de comprometimento UC seguro que inclui uma hipótese de setup. Tanto a impossibilidade da implementação da funcionali-

dade ideal *pura* quanto a segurança UC do protocolo *real* ilustram o poder do *framework*, que como já indicado tem sido usado no contexto mais amplo de computação segura multipartes

4.8. Referências

Referências

[Barak et al. 2001] Barak, B., Goldreich, O., Goldwasser, S., and Lindell, Y. (2001). Resettably-sound zero-knowledge and its applications. In *APPEARED IN 42ND FOCS*, pages 116–125. IEEE Computer Society Press.

[Beaver 1991] Beaver, D. (1991). Secure multi-party protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4:75–122.

[Beaver 1996] Beaver, D. (1996). Adaptive zero knowledge and computational equivocation (extended abstract). In *28th Annual ACM Symposium on Theory of Computing*, pages 629–638, Philadephia, Pennsylvania, USA. ACM Press.

[Beaver and Haber 1992] Beaver, D. and Haber, S. (1992). Cryptographic protocols provably secure against dynamic adversaries. In Rueppel, R. A., editor, *Advances in Cryptology – EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 307–323, Balatonfüred, Hungary. Springer, Berlin, Germany.

[Bellare et al. 1998a] Bellare, M., Canetti, R., and Krawczyk, H. (1998a). A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *30th Annual ACM Symposium on Theory of Computing*, pages 419–428, Dallas, Texas, USA. ACM Press.

[Bellare et al. 1998b] Bellare, M., Pointcheval, D., and Rogaway, P. (1998b). Relations among notions of security for public-key encryption schemes. In *CRYPTO ´98*, pages 26–45. Springer-Verlag.

[Bellare and Rogaway 1994] Bellare, M. and Rogaway, P. (1994). Entity authentication and key distribution.

[Ben-Or et al. 1993] Ben-Or, M., Canetti, R., and Goldreich, O. (1993). Asynchronous secure computation. In *25th Annual ACM Symposium on Theory of Computing*, pages 52–61, San Diego, California, USA. ACM Press.

[Blum 1982] Blum, M. (1982). Coin flipping by telephone. pages 133–137.

[Blum and Micali 1984] Blum, M. and Micali, S. (1984). How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13(4):850–864.

[Brassard et al. 1988] Brassard, G., Chaum, D., and Crépeau, C. (1988). Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, pages 156–189.

[Canetti 1998] Canetti, R. (1998). Security and composition of multi-party cryptographic protocols. *JOURNAL OF CRYPTOLOGY*, 13:2000.

[Canetti 2000] Canetti, R. (2000). Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067. revised Jan 2005 and Dec 2005.

[Canetti 2001] Canetti, R. (2001). Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, USA. IEEE Computer Society Press.

[Canetti 2003] Canetti, R. (2003). Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239. `http://eprint.iacr.org/`.

[Canetti et al. 2010] Canetti, R., Chari, S., Halevi, S., Pfitzmann, B., Roy, A., Steiner, M., and Venema, W. (2010). Composable security analysis of os services. Cryptology ePrint Archive, Report 2010/213. `http://eprint.iacr.org/`.

[Canetti et al. 1996] Canetti, R., Feige, U., Goldreich, O., and Naor, M. (1996). Adaptively secure multi-party computation. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 639–648, New York, NY, USA. ACM.

[Canetti and Gajek 2010] Canetti, R. and Gajek, S. (2010). Universally composable symbolic analysis of diffie-hellman based key exchange. Cryptology ePrint Archive, Report 2010/303. `http://eprint.iacr.org/`.

[Canetti et al. 1999] Canetti, R., Goldreich, O., Goldwasser, S., and Micali, S. (1999). Resettable zero-knowledge. In *In 32nd STOC*, pages 235–244.

[Canetti et al. 2005] Canetti, R., Halevi, S., Katz, J., Lindell, Y., and MacKenzie, P. (2005). Universally composable password-based key exchange. Cryptology ePrint Archive, Report 2005/196. `http://eprint.iacr.org/`.

[Canetti and Herzog 2004] Canetti, R. and Herzog, J. (2004). Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). Cryptology ePrint Archive, Report 2004/334. `http://eprint.iacr.org/`.

[Canetti and Krawczyk 2001] Canetti, R. and Krawczyk, H. (2001). Analysis of key-exchange protocols and their use for building secure channels. In Pfitzmann, B., editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474, Innsbruck, Austria. Springer, Berlin, Germany.

[Chari et al. 2011] Chari, S., Jutla, C., and Roy, A. (2011). Universally composable security analysis of oauth v2.0. Cryptology ePrint Archive, Report 2011/526. `http://eprint.iacr.org/`.

[Damgård 1990] Damgård, I. (1990). On the existence of bit commitment schemes and zero-knowledge proofs. In Brassard, G., editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 17–27, Santa Barbara, CA, USA. Springer, Berlin, Germany.

[Di Crescenzo et al. 1998] Di Crescenzo, G., Ishai, Y., and Ostrovsky, R. (1998). Non-interactive and non-malleable commitment. In *30th Annual ACM Symposium on Theory of Computing*, pages 141–150, Dallas, Texas, USA. ACM Press.

[Dodis and Micali 2000] Dodis, Y. and Micali, S. (2000). Parallel reducibility for information-theoretically secure computation. In Bellare, M., editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 74–92, Santa Barbara, CA, USA. Springer, Berlin, Germany.

[Dolev et al. 2000] Dolev, D., Dwork, C., and Naor, M. (2000). Nonmalleable cryptography. *SIAM J. Comput.*, 30:391–437.

[Dowsley et al. 2009] Dowsley, R., Müller-Quade, J., Otsuka, A., Hanaoka, G., Imai, H., and Nascimento, A. C. A. (2009). Universally composable and statistically secure verifiable secret sharing scheme based on pre-distributed data. Cryptology ePrint Archive, Report 2009/273. `http://eprint.iacr.org/`.

[Dowsley et al. 2010] Dowsley, R., van de Graaf, J., Marques, D., and Nascimento, A. C. A. (2010). A two-party protocol with trusted initializer for computing the inner product. Cryptology ePrint Archive, Report 2010/289. `http://eprint.iacr.org/`.

[Dwork et al. 1998] Dwork, C., Naor, M., and Sahai, A. (1998). Concurrent zero-knowledge. In *30th Annual ACM Symposium on Theory of Computing*, pages 409–418, Dallas, Texas, USA. ACM Press.

[Even et al. 1985] Even, S., Goldreich, O., and Lempel, A. (1985). A randomized protocol for signing contracts. *Commun. ACM*, 28:637–647.

[Fischlin and Fischlin 2000] Fischlin, M. and Fischlin, R. (2000). Efficient non-malleable commitment schemes. In Bellare, M., editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 413–431, Santa Barbara, CA, USA. Springer, Berlin, Germany.

[Gajek et al. 2008] Gajek, S., Manulis, M., Pereira, O., Sadeghi, A.-R., and Schwenk, J. (2008). Universally composable security analysis of tls—secure sessions with handshake and record layer protocols. Cryptology ePrint Archive, Report 2008/251. `http://eprint.iacr.org/`.

[Galil et al. 1988] Galil, Z., Haber, S., and Yung, M. (1988). Cryptographic computation: Secure faut-tolerant protocols and the public-key model. In Pomerance, C., editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 135–155, Santa Barbara, CA, USA. Springer, Berlin, Germany.

[Garay and MacKenzie 2000] Garay, J. A. and MacKenzie, P. D. (2000). Concurrent oblivious transfer. In *41st Annual Symposium on Foundations of Computer Science*, pages 314–324, Redondo Beach, California, USA. IEEE Computer Society Press.

[Goldreich 2000] Goldreich, O. (2000). *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA.

[Goldreich and Krawczyk 1990] Goldreich, O. and Krawczyk, H. (1990). On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25:169–192.

[Goldreich et al. 1987] Goldreich, O., Micali, S., and Wigderson, A. (1987). How to play any mental game, or a completeness theorem for protocols with honest majority. In Aho, A., editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City,, New York, USA. ACM Press.

[Goldreich and Oren 1994] Goldreich, O. and Oren, Y. (1994). Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32.

[Goldwasser and Levin 1991] Goldwasser, S. and Levin, L. A. (1991). Fair computation of general functions in presence of immoral majority. In Menezes, A. J. and Vanstone, S. A., editors, *Advances in Cryptology – CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93, Santa Barbara, CA, USA. Springer, Berlin, Germany.

[Goldwasser and Micali 1984] Goldwasser, S. and Micali, S. (1984). Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299.

[Goldwasser et al. 1989] Goldwasser, S., Micali, S., and Rackoff, C. (1989). The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208.

[Gorantla et al. 2009] Gorantla, M. C., Boyd, C., and Nieto, J. M. G. (2009). Universally composable contributory group key exchange. Cryptology ePrint Archive, Report 2009/300. `http://eprint.iacr.org/`.

[Green and Hohenberger 2008] Green, M. and Hohenberger, S. (2008). Universally composable adaptive oblivious transfer. Cryptology ePrint Archive, Report 2008/163. `http://eprint.iacr.org/`.

[Hirt and Maurer 2000] Hirt, M. and Maurer, U. M. (2000). Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60.

[Katz et al. 2011] Katz, J., Maurer, U., Tackmann, B., and Zikas, V. (2011). Universally composable synchronous computation. Cryptology ePrint Archive, Report 2011/310. `http://eprint.iacr.org/`.

[Kuesters and Tuengerthal 2009] Kuesters, R. and Tuengerthal, M. (2009). Universally composable symmetric encryption. Cryptology ePrint Archive, Report 2009/055. `http://eprint.iacr.org/`.

[Kurosawa and Furukawa 2008] Kurosawa, K. and Furukawa, J. (2008). Universally composable undeniable signature. Cryptology ePrint Archive, Report 2008/094. `http://eprint.iacr.org/`.

[Micali and Rogaway 1992] Micali, S. and Rogaway, P. (1992). Secure computation (abstract). In Feigenbaum, J., editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404, Santa Barbara, CA, USA. Springer, Berlin, Germany.

[Naor 1991] Naor, M. (1991). Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158.

[Naor et al. 1998] Naor, M., Ostrovsky, R., Venkatesan, R., and Yung, M. (1998). Perfect zero-knowledge arguments for np can be based on general complexity assumptions (extended abstract). *JOURNAL OF CRYPTOLOGY*, 11:87–108.

[Naor and Yung 1990] Naor, M. and Yung, M. (1990). Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, STOC '90, pages 427–437, New York, NY, USA. ACM.

[Pfitzmann et al. 2000] Pfitzmann, B., Schunter, M., and Waidner, M. (2000). Secure reactive systems.

[Pfitzmann and Waidner 1994] Pfitzmann, B. and Waidner, M. (1994). A general framework for formal notions of "secure"systems. In *SYSTEM, HILDESHEIMER INFORMATIK-BERICHTE 11/94, UNIVERSITAT*.

[Pfitzmann and Waidner 2000] Pfitzmann, B. and Waidner, M. (2000). Composition and integrity preservation of secure reactive systems. In *In Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254. ACM Press.

[Rabin 1981] Rabin, M. (1981). How to exchange secrets using oblivious transfer. Technical report, Tech. Memo TR-81, Aiken Computation Laboratory, Harvard University.

[Rackoff and Simon 1992] Rackoff, C. and Simon, D. R. (1992). Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 433–444, London, UK. Springer-Verlag.

[Shoup 1999] Shoup, V. (1999). On formal models for secure key exchange. Technical report.

[Unruh and Müller-Quade 2009] Unruh, D. and Müller-Quade, J. (2009). Universally composable incoercibility. Cryptology ePrint Archive, Report 2009/520. `http://eprint.iacr.org/`.

[Yao 1982] Yao, A. C. (1982). Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois. IEEE Computer Society Press.

A. Gerador Pseudo Aleatório

Informalmente, uma distribuição $\mathscr{D}$ é pseudo aleatória se nenhum *distinguisher*, aqui referido como diferenciador *D*, que execute em tempo polinomial consegue detectar se lhe foi passado um *string* amostrado de $\mathscr{D}$ ou se ele foi escolhido uniformemente ao acaso. Na formalização, será exigido que o algoritmo *D*, em tempo polinomial, gere a saída 1 com quase a mesma probabilidade tanto para um valor vindo de $\mathscr{D}$, quanto para uma entrada verdadeiramente aleatória. Essa saída pode ser interpretada como um *guess*.

Um gerador pseudo aleatório é um algoritmo determinístico que recebe na entrada um pequeno *string* verdadeiramente aleatório e o expande em um *string* mais longo que é pseudo aleatório. Dito de outra forma, um gerador pseudo aleatório usa uma pequena quantidade de aleatoriedade para gerar uma quantidade maior pseudo aleatória.

Na definição que segue, n é o comprimento do *string* de entrada e $l(n)$ é o comprimento da saída, sendo interessantes apenas os casos em que $l(n) > n$.

Definição A.1. *Sejam $l(.)$ um polinômio e G um algoritmo determinístico em tempo polinomial tal que, para qualquer entrada $s \in \{0,1\}^n$, o algoritmo G gera como saída um string de comprimento $l(n)$. Diz-se que G é um gerador pseudo aleatório se:*

- *para todo n, $l(n) > n$*
- *para todo diferenciador D em tempo polinomial,*

$$|Pr[D(r) = 1] - Pr[D(G(s)) = 1]|$$

é desprezível. Onde r é escolhido uniformemente ao acaso de $\{0,1\}^{l(n)}$. $l(n)$ é o fator de expansão de G.

Um diferenciador pode ser definido como um algoritmo D em tempo polinomial que recebe um *string* de entrada $x \in \{0,1\}^{l(n)}$ e gera como saída $D(x) = 1$ se x é verdadeiramente aleatório ou $D(x) = 0$ caso contrário.

Definição A.2. *Seja D um algoritmo em tempo polinomial, que recebendo um valor $x \in \{0,1\}^{l(n)}$ como entrada e correspondentemente gera como saída $D(x)$ tal que:*

$$D(x) = \begin{cases} 1, & \textit{se x é verdadeiramente aleatório} \\ 0, & \textit{caso contrário} \end{cases}$$

Capítulo

5

Gerência de Identidades Federadas em Nuvens: Enfoque na Utilização de Soluções Abertas

Guilherme Feliciano[1], Lucio Agostinho[1], Eliane Guimarães[2], Eleri Cardozo[1]

[1]Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas - UNICAMP

[2]Centro de Tecnologia da Informação Renato Archer
13083-970 - Campinas - SP

{gof,larocha,eleri}@dca.fee.unicamp.br, eliane.guimaraes@cti.gov.br

Abstract

Cloud computing emerges as a new paradigm for the offering of services in the Web. The main idea is to transfer most of the processing and storage of user applications to a remote cloud of services. Although this approach leads to new business opportunities, the issue of security is still an open and difficult to solve problem. This is because in a cloud there are several applications available as services, many of which have their own access control systems. Furthermore, applications that support service compositions across distinct domains require authentication mechanisms that take into account this collaborative nature. This short course addresses the offering of federated services under an Identity Management perspective. The main concepts related to this subject are introduced in the context of cloud computing as well as the main solutions employed in open federated cloud environments. Finally, a case study in a field of networked robotics that uses one of the solutions discussed in this short course is presented.

Resumo

A computação em nuvem surge como um novo paradigma para a oferta de serviços na Web. A ideia principal é transferir a maior parte do processamento e armazenamento das aplicações dos usuários para uma nuvem remota de serviços. Apesar desta abordagem trazer novas oportunidades de negócios, a questão da segurança ainda é um problema em aberto e de difícil solução. Isso porque em uma nuvem há diversas aplicações

oferecidas como serviços, sendo que muitas delas possuem seus próprios sistemas para controle de acesso. Além disso, aplicações que suportam a composição de serviços entre domínios distintos exigem mecanismos de autenticação que levem em conta esta realidade colaborativa. Este minicurso explora a oferta de serviços em federações sob a ótica da Gerência de Identidades. São apresentados os principais conceitos relacionados a esse tema no contexto de computação em nuvem e as principais soluções abertas empregadas em nuvens federadas. Ao final, é apresentado um estudo de caso no domínio da robótica em rede que utiliza uma das soluções discutidas neste minicurso.

5.1. Introdução

No final da década de 90, a Gerência de Identidades (IdM - *Identity Management*) e a segurança da informação eram tratadas separadamente. Mais especificamente, a infraestrutura de IdM se destinava à provisão de serviços, especialmente serviços centralizados de autenticação [Suess and Morooney 2009]. Nesta autenticação centralizada empregava-se tecnologias, tais como GSS-API (*Generic Security Service Application Programming Interface*), SASL (*Simple Authentication and Security Layer*) e/ou SSL/TLS (*Secure Sockets Layer/Transport Layer Security*) para o estabelecimento de um contexto de segurança entre dois pares comunicantes [Johansson 2009].

Neste cenário, organizações (empresas ou universidades) empregavam serviços de diretórios baseados em LDAP (*Lightweight Directory Access Protocol*). Esses serviços eram destinados a fornecer mecanismos de autenticação de forma centralizada, com o objetivo de facilitar a gerência deste ambiente e prover uma forma de autenticação única, conhecida como SSO (*Single Sign-On*). Como a identidade do usuário e o seu identificador geralmente eram os mesmos, um serviço de *e-mail*, por exemplo, poderia utilizar o identificador do usuário como chave para armazenar todas as suas mensagens e configurações de sua caixa postal no serviço de diretório [Johansson 2009].

Este modelo ainda é muito utilizado e funciona bem em ambientes onde há uma única organização ou um único serviço de diretório. As limitações deste modelo centralizado surgem quando as organizações precisam utilizar serviços compartilhados, como por exemplo, em cenários de B2B (*business-to-business*) ou necessitam fundir seus repositórios, como por exemplo, em fusões de empresas [Johansson 2009].

A falha deste modelo centralizado em tratar a autenticação em cenários fora do domínio de uma organização, ocorre em razão da falha do LDAP em fornecer um mecanismo de autenticação escalável, que não prenda a organização a uma solução de um fabricante específico ou que não obrigue a organização a permitir acesso externo às suas bases de dados [Johansson 2009].

Com a realidade da computação em nuvem e o surgimento da chamada Web 2.0, o cenário para a IdM tornou-se mais complexo. A Web 2.0 aprofundou os conceitos de colaboração, interoperabilidade entre aplicações e compartilhamento de informações na Internet [Shelly and Frydenberg 2010]. A nuvem surge como uma alternativa para oferecer serviços na Web de forma rápida, escalável e com custos proporcionais à demanda.

A gerência de identidades federadas (*Federated IdM*) têm como desafio oferecer

uma infraestrutura que permita tanto aos usuários uma experiência de SSO, quanto aos administradores um mecanismo de autenticação e controle de acesso aos recursos federados entre parceiros [Olden 2011]. Segundo Stallings [Stallings 2011], um ambiente de gerência de identidades federadas deve ter principalmente os seguintes elementos: autenticação, autorização, *logging*, provisionamento de usuários, automação com *workflow*, delegação, federação, SSO e mecanismo de *password reset* oferecido aos usuários.

Esta infraestrutura é formada por um sistema integrado de políticas, processos de negócios e tecnologias para o tratamento das identidades, definição, certificação e gerência do ciclo de vida das identidades, mecanismos para troca segura e validação destas informações e aspectos legais [Wangham et al. 2010]. A gerência de identidades federadas também têm como meta permitir que os usuários possam estabelecer ligações entre suas diversas identidades. Esta ligação lógica é denominada federação de identidades [Bertino and Takahashi 2011].

Este minicurso tem como principal objetivo explorar os padrões para gerência de identidades federadas, bem como as tecnologias que implementam estes padrões, com vistas à aplicação em ambientes de computação em nuvem. As soluções são apresentadas com um enfoque prático e um estudo de caso ilustra o estabelecimento de uma nuvem federada para a realização de experimentos robóticos via rede.

O capítulo está dividido em sete seções, incluindo esta. A seção 5.2 apresenta os conceitos gerais relacionados à computação em nuvem e suas técnicas fundamentais. A seção 5.3 descreve os principais conceitos e técnicas relacionados a gerência de identidades federadas. A seção 5.4 apresenta o cenário da gerência de identidades federadas em ambientes de computação em nuvem, bem como seus principais desafios e tendências da área. A seção 5.5 apresenta os principais padrões e soluções para a implantação da gerência de identidades federadas em ambientes de computação em nuvem. A seção 5.6 apresenta um estudo de caso detalhando um ambiente de computação em nuvem desenvolvido pelos autores e a motivação para a aplicação da gerência de identidades federadas neste ambiente. Por fim, a seção 5.7 resume os principais assuntos abordados e relata a experiência dos autores no uso de ferramentas para implantação de ambientes federados e integração de recursos em nuvens.

5.2. Visão Geral sobre Computação em Nuvem

Computação em nuvem é um modelo de computação distribuída que deriva características da computação em grades, no que diz respeito à provisão de informação sob demanda para múltiplos usuários concorrentes. Um domínio oferece aplicações na nuvem sem se preocupar com o local onde os serviços estão sediados ou como eles são oferecidos. Fatias do poder computacional dos nós da rede são oferecidas, reduzindo os custos para fornecer uma infraestrutura própria para prover os serviços. Os recursos são cedidos apenas durante o período de uso, reduzindo o consumo de energia quando a utilização não for mais necessária [Endo et al. 2010]. A virtualização fornece a tecnologia base para muitas soluções de nuvem. Além disso, em muitas soluções são oferecidos ambientes onde os usuários são capazes de escolher seus recursos virtualizados tais como linguagem de programação, sistema operacional e outros serviços

personalizados. Os principais benefícios são a redução dos custos de investimento em infraestrutura, dos custos operacionais e a escalabilidade para a provisão de serviços sob demanda [Verdi et al. 2010].

Segundo [Mell and Grace 2010] a computação em nuvem possui características, definições e atributos que ainda estão sendo elaborados. Ainda que possua o agrupamento de muitos modelos, as seguintes características próprias podem ser enumeradas:

1. Oferta de serviços sob demanda: alocação dinâmica dos serviços requisitados (*resource pooling*), sem interação humana com o provedor dos serviços.

2. Amplo acesso aos recursos computacionais: acesso por meio de diversos protocolos padronizados, para uma grande variedade de dispositivos como PCs, *laptops*, dispositivos móveis, dentre outros.

3. Transparência: o usuário não precisa conhecer a localização física dos recursos computacionais oferecidos.

4. Elasticidade: os serviços devem ser alocados e desalocados rapidamente, apenas no decorrer da requisição do usuário.

5. Gerência: a infraestrutura deve oferecer mecanismos para a gerência de recursos, de armazenagem, de processamento e largura de banda, dentre outros.

Outras características comuns de ambientes que utilizam computação em nuvem são [Endo et al. 2010]:

- Escalabilidade: a oferta de recursos sob demanda viabiliza a oferta de serviços para um número maior de usuários. Os recursos serão alocados apenas pelo período contratado, reduzindo a sub-utilização da rede de serviços. Essa característica implica na elasticidade da oferta de recursos para muitos usuários concorrentes.

- Modelo *pay-per-use*: cobrança proporcional ao uso dos recursos. A computação em nuvem é um exemplo de *utility computing* (computação vista como uma utilidade) porque a oferta desses serviços é similar a outros tradicionais, onde o usuário paga pelo fornecimento de eletricidade, água, gás natural ou serviços de telefonia [Breitman and Viterbo 2010].

- Virtualização: o usuário tem a ilusão de que interage com os recursos de um *host* real quando, na verdade, utiliza um ambiente que simula o acesso físico do *host* no qual estão hospedados.

Na computação em nuvem um provedor de serviços pode utilizar um ou mais modelos para a oferta de serviços. Ainda assim, a administração do domínio é responsável por controlar a infraestrutura, sistema operacional, servidores, operações de persistência e demais requisitos para a oferta de serviços para uma grande quantidade de usuários concorrentes. Os modelos para prestação de serviços em nuvem são os seguintes:

- IaaS (*Infrastructure as a Service*): destaca a importância da infraestrutura na provisão de serviços. O provedor é capaz de oferecer uma infraestrutura de armazenagem, processamento e demais recursos de *hardware*, tais como servidores e componentes de rede, de maneira transparente para o usuário. Exemplo de provedores: Amazon AWS [Amazon 2010] e FlexiScale [FlexiScale 2010].

- PaaS (*Platform as a Service*): destaca a importância de plataformas na provisão de serviços. O usuário é capaz de desenvolver suas próprias aplicações, respeitando o modelo de desenvolvimento da plataforma. Também são oferecidos serviços de comunicação (serviços Web, por exemplo), armazenagem e linguagens de programação. Exemplo de provedores: Ning [Ning 2010] e Microsoft Windows Azure Platform [Windows Azure 2010].

- SaaS (*Software as a Service*): o provedor de serviços habilita a execução de aplicações de uso exclusivo do usuário e/ou aplicações fornecidas pelo próprio provedor e/ou terceiros, tais como aplicativos de *e-mail* empresariais, grupos de discussão, ferramentas para edição de sites e demais aplicações que são compartilhadas por um grande número de usuários. Nesse modelo, os serviços são mantidos em um provedor de serviços e são acessíveis pela Internet. O uso de tecnologias como serviços Web, arquiteturas orientadas a serviço (*SOA - Service Oriented Architecture*) e AJAX (*Asynchronous JavaScript and XML*) impulsionaram o desenvolvimento de aplicações remotas que inclusive podem ser incorporadas a Web sites como serviços. Aliado a isso, o aumento da largura de banda entre os usuários finais e os provedores de serviços contribui para o aumento de aplicações sediadas remotamente. Exemplo de provedores: Salesforce [Salesforce 2010] e Google Apps [Google 2010].

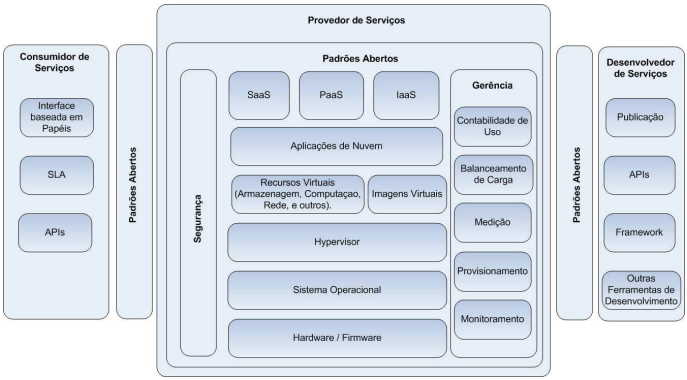

Fig. 5.1. Visão Geral dos Principais Componentes para Ambientes de Computação em Nuvem. Adaptado de [Zappert 2010].

A Fig. 5.1 apresenta uma visão geral dos principais componentes encontrados em ambientes de computação em nuvem. A descrição detalhada de cada um desses

componentes pode ser encontrada em [Zappert 2010]. Nessa figura destaca-se os relacionamentos entre os elementos Provedor de Serviços, Consumidor de Serviços e Desenvolvedor de Serviços, os quais são interligados por tecnologias de comunicação baseadas em padrões abertos. No Provedor de Serviços é observado um nível crescente de abstração em cada uma das camadas. Por exemplo, infraestruturas de nuvem são altamente dependentes do tipo de *hardware* escolhido, além do sistema operacional base de toda a infraestrutura. O suporte a virtualização é oferecido pelo componente Hypervisor que é mantido sobre o sistema operacional. A virtualização de recursos de *hardware* e *software* oferece a base para que um amplo conjunto de aplicações de nuvem sejam oferecidas de várias formas diferentes (*aaS - *Everything as a Service*).

O componente de Gerência deve incluir funcionalidades para contabilidade de uso, balanceamento de carga, medição de tráfego e uso de recursos, provisionamento de máquinas virtuais e demais funções de monitoramento. O componente Segurança deve incluir funcionalidades tanto para a segurança do tráfego de dados, quanto para a segurança relacionada ao acesso ao ambiente. Tanto os componentes de gerência quanto os componentes de segurança devem abordar desde as camadas inferiores do *hardware/firmware* até as camadas superiores no nível das aplicações e infraestrutura de suporte. O Consumidor de Serviços tem acesso aos recursos por meio de interfaces baseadas em regras de acesso compatíveis com os mecanismos de segurança da nuvem. Além disso, acordos de nível de serviço (SLAs - *Service Level Agreements*) são necessários para garantir a qualidade da oferta de serviços e disponibilidade de acesso, o qual é realizado por meio de interfaces de programação para aplicações (APIs - *Application Programming Interfaces*) que se comunicam com os serviços oferecidos por meio de interfaces públicas.

O Desenvolvedor de Serviços utiliza uma gama de serviços para desenvolver novas aplicações para a nuvem. Esses serviços utilizam padrões abertos para se comunicar e estender as funcionalidades dos serviços já existentes. O desenvolvimento de novos serviços demanda o uso de *frameworks* próprios para o desenvolvimento e demais ferramentas de suporte, bem como a publicação dos mesmos com o auxílio de APIs (componentes Publicação e APIs).

A infraestrutura da nuvem é oferecida em diversos níveis de abrangência e disponibilidade, de acordo com as finalidades da organização e de seus usuários. Cada um desses modelos admite que uma entidade terceirizada mantenha a nuvem: a) nuvem privada: centrada no domínio de uma organização; b) nuvem pública: acessível pela Internet, geralmente para uso público em geral; c) nuvem comunitária: nuvens compartilhadas entre várias organizações com finalidades em comum; d) nuvem híbrida: uma combinação das anteriores. O uso desse último modelo é uma alternativa para estender o *pool* de recursos utilizando serviços de outros provedores de nuvem. Nesses casos, a organização utiliza os serviços de uma nuvem pública para prover a maioria das funcionalidades para seus clientes, mas os dados restritos são mantidos e gerenciados em *datacenters* particulares na organização. Outra funcionalidade de destaque é a possibilidade de migração de serviços para redução de custos durante o ciclo de vida das máquinas virtuais (*live migration*) para cumprir os requisitos de armazenagem, desempenho, capacidade de processamento, largura de banda e demais quesitos relacionados a qualidade de serviço (QoS - *Quality of Service*). Isso sugere que

relacionamentos de confiança precisam assegurar SLA entre os domínios parceiros.

De acordo com essa análise, com os recentes avanços das soluções de virtualização e de computação em nuvem, a interoperabilidade será um fator fundamental para manter a cooperação mútua entre provedores de nuvem. Nesse cenário, uma federação de nuvens híbridas é uma solução onde cada domínio é capaz de expandir seus recursos virtualizados sob demanda, requisitando mais recursos computacionais de outros domínios federados, quando necessário. A gerência de identidades federadas aplica-se a esse cenário de forma a gerenciar o uso seguro de recursos por parte dos usuários e sistemas de sites parceiros.

5.2.1. Técnicas Fundamentais em Computação em Nuvem

Virtualização

Em essência, a virtualização consiste em imitar um comportamento, seja por extensão ou substituição de um recurso por outro [Carissimi 2008]. A virtualização também é definida como um sistema ou um método de dividir os recursos computacionais em múltiplos ambientes isolados [OpenVZ 2010]. O conceito de virtualização remonta à virtualização de recursos em sistemas operacionais. Soluções de alto nível como interfaces gráficas, bibliotecas e APIs são exemplos de recursos de *software* que tornam transparente para o usuário o acesso aos recursos de *hardware*, em especial, o acesso aos periféricos de entrada e saída. Ou seja, cria-se a ilusão no sistema operacional de que se tem a interação direta com os recursos de *hardware*. Diz-se também que a virtualização é uma metodologia para dividir os recursos de um computador em múltiplos ambientes de execução conhecidos como máquinas virtuais, aplicando conceitos de particionamento, *time-sharing*, simulação completa ou parcial de máquina, emulação, QoS, entre outros [Carissimi 2008].

Portanto, a virtualização é uma técnica para ocultar características físicas de recursos computacionais, de forma que os sistemas, aplicações e *end users* interajam com esses recursos. Nesta técnica, um único recurso físico, como um servidor, dispositivo de armazenagem ou sistema operacional, passa a ser visto como múltiplos recursos lógicos.

A virtualização remonta às décadas de 60 e 70. Com o uso de máquinas virtuais era possível executar e migrar aplicações legadas entre plataformas distintas, desde que houvesse uma versão da máquina virtual para a plataforma alvo. A principal motivação era ampliar e melhorar a utilização e o compartilhamento de recursos nos *mainframes* [Carissimi 2008].

Com a redução do custo do *hardware* em meados da década de 80, ocorreu uma mudança do foco de processamento centralizado em *mainframes* para o processamento distribuído em microcomputadores. O modelo cliente-servidor foi estabelecido para a computação distribuída, reduzindo a necessidade da virtualização para a integração de recursos computacionais. A redução do custo de aquisição do *hardware* e do compartilhamento de informações tornaram a virtualização pouco explorada por alguns anos [Menascé 2005].

Apenas em meados da década de 90, com o aumento do poder de processamento do *hardware* e dos computadores pessoais (PCs), a virtualização voltou a ganhar

destaque em produtos tais como o VMware [VMware Inc. 2011], User Mode Linux (UML) [Dike 2006], Xen [Xen 2010], KVM [Warnke and Ritzau 2010] e VirtualBox [Oracle 2010]. De certa forma, esses produtos trazem o conceito de virtualização como uma alternativa para executar diversos sistemas operacionais sem a necessidade de se aumentar proporcionalmente o número de *hosts* físicos que os mantêm. Isso implica em reduzir os custos relativos à aquisição de *hardware*, infraestrutura física, consumo de energia, ventilação, suporte e manutenção de vários *hosts*.

A virtualização é recomendada para consolidar múltiplos servidores em um mesmo *host*, isolar diferentes aplicações de usuários em um mesmo *host*, executar/depurar *softwares* e sistemas operacionais construídos para uma arquitetura em outra, além de simplificar a instalação de infraestruturas de *software* em diferentes domínios e testar aplicativos em *hardwares* não existentes.

O sistema operacional que executa o *software* de virtualização é denominado hospedeiro (*host*). O sistema operacional virtualizado é denominado convidado (*guest*). Múltiplos sistemas operacionais convidados podem executar no mesmo hospedeiro, sem interferência entre eles. Uma máquina virtual (VM - *Virtual Machine*) é uma camada de *software* que simula um computador real (físico) e que é capaz de executar instruções como se fosse a máquina física. O núcleo do sistema operacional hospedeiro fornece bibliotecas para suportar múltiplas máquinas virtuais. Sistemas operacionais convidados executam sobre máquinas virtuais.

SOC - Service Oriented Computing

A computação em nuvem supre as necessidades de provedores de serviços na Internet quanto à maneira de se aumentar a capacidade de processamento sob demanda, sem a necessidade de se ampliar os investimentos na própria infraestrutura. Além disso, reduz os custos com o treinamento de pessoal e aquisição de licenças adicionais de *software*. Muitas soluções de nuvem utilizam serviços Web para disponibilizar interfaces Web (APIs) para acesso aos seus serviços. Serviços Web seguem a arquitetura SOA que é baseada em um modelo cliente/servidor em que o cliente faz o papel de requisitante de serviços e o servidor de provedor de serviços. A comunicação é baseada na troca de mensagens síncronas ou assíncronas independentes do protocolo de transporte utilizado [Ma 2005]. A Fig. 5.2 ilustra uma visão geral da arquitetura SOA.

A computação orientada a serviços (SOC - *Service Oriented Computing*) define um conjunto de princípios, modelos arquiteturais, tecnologias e *frameworks* para o projeto e desenvolvimento de aplicações distribuídas. Segundo [Gonçalves et al. 2011], orientação a serviço é um paradigma que propõe a criação de unidades lógicas (ou serviços) bem definidas que podem ser utilizadas coletivamente e repetidamente.

Na arquitetura SOA as aplicações distribuídas utilizam os serviços como blocos de construção [IBM 2011]. Para simplificar a localização desses serviços em provedores distintos, SOA integra um componente para o registro e a descoberta. Com isso, os provedores registram os seus serviços em um repositório centralizado. O registro descreve as informações de cada serviço e a localização do provedor que os mantêm. Quando o cliente deseja utilizar um serviço, ele faz uma consulta nesse repositório e

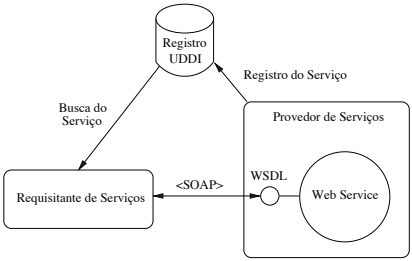

Fig. 5.2. Visão Geral da Arquitetura Orientada a Serviço.

informa quais os requisitos desejados. O repositório então devolve uma lista de serviços que satisfazem os requisitos solicitados. Após a escolha do serviço, o cliente acessa diretamente o provedor de serviços com base na informação de sua localização. Essa localização dinâmica pode ser feita no momento da execução do aplicativo. UDDI (*Universal Description, Discovery, and Integration*) especifica um método padrão para publicação e descoberta de componentes de *software* na Web segundo a arquitetura SOA [OASIS 2006]. Interfaces de serviços são descritas em WSDL (*Web Services Description Language*) e o protocolo SOAP (*Simple Object Access Protocol*) é empregado na interação cliente/servidor. WSDL e SOAP são padrões baseados em XML (*Extensible Markup Language*).

Em SOA, as unidades de *software* podem ser desenvolvidas separadamente. Tais unidades encapsulam a lógica da implementação do serviço e expõem apenas a interface de acesso à funcionalidade. Isso torna possível o desenvolvimento de aplicações distribuídas com fraco acoplamento. Além disso, a possibilidade de reuso de código e de disponibilidade na Web permite o desenvolvimento de aplicações em ambientes colaborativos distintos.

Datacenters

Infraestruturas de virtualização para nuvens geralmente utilizam um ou mais *datacenters*. Os *datacenters* são infraestruturas formadas por componentes que fornecem capacidades em larga escala para processamento, armazenagem e serviços de rede para uma ou mais organizações [Veras 2009]. *Datacenters* podem ser agrupados nas categorias de PDC (*Private Data Center*) e IDC (*Internet Data Center*). Um PDC é voltado para empresas privadas, instituições ou órgãos governamentais, com a finalidade de processar informações internas. Um IDC, por outro lado, geralmente é mantido por um provedor de serviços de telecomunicações ou informação. Um *datacenter* agrupa de centenas a milhares de componentes, desde *switches* e roteadores, a poderosos servidores, balanceadores de carga e dispositivos de armazenagem. Essas características exigem que a infraestrutura tenha suporte adequado para o funcionamento, incluindo a provisão adequada de energia, ventilação/ar condicionado, segurança, monitoramento e redundância de dados. Além disso, equipamentos especializados requerem configuração

e treinamento, o que encarece os custos para a provisão de uma infraestrutura particular [Gonçalves et al. 2011].

Em ambientes de computação em nuvem, o uso de *datacenters* é uma solução adequada para o oferecimento de ambientes virtualizados, visando o desenvolvimento e rápido provisionamento de aplicações empresariais. Empresas não precisam empregar grandes quantias para a aquisição e configuração de infraestruturas de tecnologia que mudam rapidamente, mas sim utilizar infraestruturas pré-configuradas de ambientes, como os da Amazon ou IBM, por exemplo. O custo para o processamento e armazenagem de dados pode reduzir ou ampliar, conforme a demanda.

5.2.2. Sistemas Abertos para Gerência de Nuvens

Atualmente existem diversas soluções de código aberto para a gerência de recursos virtuais em nuvens. Uma classificação detalhada dos principais ambientes é apresentada em [Endo et al. 2010]. Dentre estas soluções destaca-se a plataforma XCP (*Xen Cloud Platform*) [Xen 2010]. A plataforma inclui o agrupamento e o isolamento de recursos de *hardware* e rede, provisionamento de sistemas, APIs para acessar os recursos virtuais e compatibilidade com um grande número de sistemas operacionais. XCP herda características dos ambientes de virtualização para-virtualizados dos produtos Xen.

O projeto Nimbus [Nimbus 2011] disponibiliza um *toolkit* de código aberto para oferecer um ambiente de *cluster* como uma IaaS. Esse *toolkit* oferece interfaces WSDL para o serviço Amazon EC2, Amazon EC2 Query API, e WSRF (*Web Services Resource Framework*) para aplicações em grades. Também são oferecidas interfaces que seguem o padrão arquitetural REST (*Representational State Transfer*) para o acesso à base de dados de recursos e escalonamento de VMs por meio de uma interface de gerência. A implementação da virtualização é baseada em Xen e KVM (*Kernel Virtual Machine*).

O projeto OpenNebula [OpenNebula 2010] oferece um *toolkit* de código aberto para a provisão de nuvens públicas, privadas e híbridas. A infraestrutura do sistema oferece recursos sob demanda para usuários finais, e foi projetada para ser integrada com outras soluções de armazenagem e rede. As VMs são utilizadas em um *pool* de recursos e toda a alocação de recursos é baseada em políticas. O ambiente virtualizado é acessível por meio de interfaces OCCI (*Open Cloud Computing Interface*), definidas pelo Open Grid Forum. Além disso, OpenNebula oferece interfaces para a EC2 Query API da Amazon, através da interface Web OpenNebula Sunstone e um subconjunto de chamadas da vCloud API da VMware. A infraestrutura tem suporte para Xen, KVM/Linux e VMware.

Eucalyptus [Murari 2010] é uma plataforma de *software* de código aberto para a criação e gerência de nuvens públicas e privadas, que possui APIs interoperáveis com as APIs da plataforma Amazon EC2/S3. A empresa Canonical, mantenedora da distribuição Ubuntu, adotou inicialmente o Eucalyptus, juntamente com outros *softwares* de código aberto, como solução de nuvem para o Ubuntu Server Edition. Essa solução ficou conhecida como Ubuntu Enterprise Cloud (UEC). Eucalyptus suporta o uso de Xen, KVM e VMware. A versão UEC, porém, suporta apenas o uso do KVM. Recentemente, a Canonical demonstrou interesse no uso de outra solução para nuvem, conhecida como OpenStack [Rackspace 2010].

É interessante notar que nenhuma dessas soluções contempla funcionalidades necessárias para gerenciar identidades de usuários pertencentes a múltiplas nuvens. Geralmente as iniciativas adotadas para a criação de federações contemplam o uso de outras infraestruturas específicas para essa tarefa, ou seja, infraestruturas para gerência de identidades federadas são agregadas como serviços adicionais ao ambiente de computação em nuvem.

5.3. Gerência de Identidades: Conceitos e Técnicas

O conceito de identidade, segundo [Cao and Yang 2010], é difícil de precisar, uma vez que a definição de identidade está relacionada ao ambiente onde é empregada, a contextos semânticos e a casos de uso. Como uma definição mais geral, pode-se dizer que uma identidade é uma representação de uma entidade ou sujeito que seja suficiente para identificar esta entidade em um contexto particular [El Maliki and Seigneur 2007]. Uma entidade, por sua vez, é qualquer coisa existente no mundo real. Como exemplos de entidades temos uma pessoa, um *host* ou uma aplicação. Em geral, a relação de cardinalidade de uma entidade é que ela pode ter múltiplas identidades. De acordo com a norma ITU-T Y.2720 [ITU-T 2009], uma identidade pode consistir de:

- Identificador: conjunto de dígitos, caracteres e símbolos ou qualquer outra forma de dados usada para identificar unicamente uma identidade. Podem ser delimitados pelo tempo e/ou espaço. Por exemplo, uma URL (*Uniform Resource Locator*) é única ao longo do tempo. Como exemplo de identificadores temos CPF, RG, número de matrícula e número de passaporte.
- Credenciais: uma credencial é um atestado de qualificação, competência ou autoridade, expedida por terceiros com autoridade relevante ou competência para tal ato e que atesta a veracidade da identidade. Na computação, exemplos de credenciais incluem certificados digitais X.509 assinados por uma autoridade certificadora (CA - *Certificate Authority*), senha, asserções SAML (*Security Assertions Markup Language*), dentre outros.
- Atributos: um conjunto de dados que descreve as características fundamentais de uma identidade. Como exemplo temos: nome completo, domicílio, data de nascimento e papéis (*roles*).

A Fig. 5.3 ilustra a relação entre os componentes de uma identidade na notação UML (*Unified Modeling Language*).

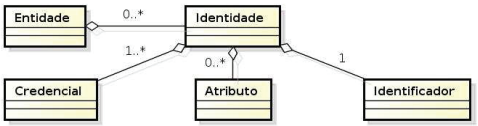

Fig. 5.3. Relação entre os Componentes da Identidade.

5.3.1. Ciclo de Vida da Identidade

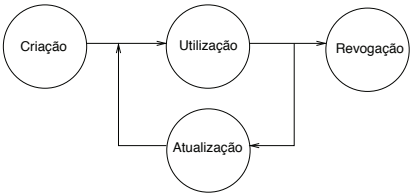

Fig. 5.4. Ciclo de Vida da Identidade.

A Fig. 5.4 ilustra a ciclo de vida de uma identidade e a relação entre as suas fases. De acordo com esta figura o ciclo de vida de uma identidade pode ser categorizado em [Bertino and Takahashi 2011]:

- Criação: a criação de uma identidade está relacionada com a provisão da infraestrutura de tecnologia da informação e comunicação (TIC). É necessário um mecanismo para automatizar a sua criação mediante informação de dados de um usuário e para armazenar estas em um repositório (por exemplo, banco de dados). Esta fase possui três subfases:
  - Verificação dos atributos. Um atributo é verificado por uma autoridade confiável por parte dos destinatários destes atributos. Por exemplo, em situações onde o cadastro em um site requer atributos obrigatórios, tais como RG e CPF. Há casos também onde este passo não ocorre. Por exemplo, em cadastros de *blogs*, onde os atributos requisitados não requerem nenhuma verificação.
  - Emissão da credencial. Após a verificação dos atributos, as credenciais são emitidas. Elas podem ser emitidas por alguma autoridade, pelo próprio sujeito ou pela entidade onde ocorreu o cadastro. As credenciais podem ser de várias formas (certificados digitais, *passwords*, entre outros). É importante ressaltar que cada tipo de credencial implica em um nível de confiança sobre a mesma.
  - Formação da identidade. A identidade é formada pelos atributos verificados, as credenciais emitidas e identificadores que são atribuídos por terceiros ou pelo próprio sujeito.
- Utilização: uma vez que a identidade foi criada, ela pode ser utilizada por vários sistemas e serviços. Neste caso mecanismos para autenticar e autorizar são ações geralmente executadas. Pode também incluir outras ações, tais como bilhetagem. O uso da identidade implica na utilização de forma segura e privativa, principalmente em ambientes federados onde os pares comunicantes devem ser capazes de descobrir, distinguir e autenticar identidades de forma confiável.
- Atualização: a fase de atualização envolve a alteração de valores de atributos já existentes (por exemplo, um novo endereço) ou a inserção de novos atributos para

refletir novas políticas ou regras de negócio. Com a alteração da identidade é necessário também que os sistemas de IdM permitam que estes novos parâmetros sejam propagados para domínios federados. Outra questão interessante é deixar alguns serviços mais procurados à disposição do usuário. Um exemplo é o serviço de *password reset*, que pode ser oferecido ao usuário que esqueceu ou teve a sua senha bloqueada.

- Revogação: identidades e credenciais devem ser canceladas se estas ficarem obsoletas (por exemplo, expirou a data de validade) ou inválidas. Essa fase é importante para manter o sistema de IdM atualizado.

5.3.2. Autenticação e Controle de Acesso (Autorização)

O processo de verificar a existência de uma identidade é chamado de autenticação [Stallings 2011]. Para que tal processo se inicie é necessário que o usuário forneça uma credencial válida para o método de autenticação utilizado pela entidade autenticadora. De posse da credencial válida, a entidade autenticadora poderá verificar em sua base de identidades se há alguma identidade cuja credencial seja a fornecida pelo usuário. Caso a entidade autenticadora encontre alguma entrada referente à credencial, o usuário é autenticado. Com isso entende-se que o usuário provou ser quem ele informou [Stallings 2011].

Uma vez realizada a autenticação do usuário, a entidade autenticadora poderá verificar qual ou quais são as permissões de acesso ao recurso originalmente requisitado. Este processo de verificar as permissões de acesso a um determinado recurso denomina-se autorização [Stallings 2011].

5.3.3. Modelos para Gerência de Identidades

Os modelos para IdM são classificados de acordo com a sua arquitetura. O modelo mais simples é conhecido como modelo isolado ou tradicional. Neste modelo não há divisão de responsabilidade e um único servidor é responsável pelo tratamento de todo o ciclo de vida da identidade e da autenticação e autorização. Uma desvantagem deste modelo é que o usuário precisa se lembrar da identidade que possui em cada serviço que este usuário acessa. Isto torna a experiência do usuário tediosa. Para o provedor de serviços isto representa um custo maior para gerenciar sua infraestrutura de IdM [Wangham et al. 2010].

Para amenizar os problemas existentes no modelo isolado, surge o modelo centralizado. Neste modelo cliente-servidor, há o conceito de provedor de identidades (IdP - *Identity Provider*) e provedores de serviço (SPs - *Service Providers*). Os provedores de serviço não armazenam as identidades dos usuários em seus sistemas e também delegam a etapa da autenticação para um provedor de identidades. Neste modelo todos os SPs utilizam os serviços de identidade oferecidos por um único IdP. Assim, é possível para um usuário que possua uma identidade em um site na Web e que utiliza este modelo, realizar SSO. Com o mecanismo de SSO, o usuário apresenta seu identificador e credenciais apenas uma vez no IdP parceiro do site. O SP, por utilizar o mesmo IdP, pode confiar na autenticação prévia do usúario e assumir que o usuário esteja autenticado sem necessitar de outro processo de verificação das credenciais.

Sistemas tais como Kerberos, PKI (*Public Key Infraestructure*), CAS (*Central Authentication Service*) e Microsoft Passport Network são comunente utilizados em modelos centralizados [Cao and Yang 2010]. Apesar deste modelo oferecer uma separação de responsabilidades, permitindo o SSO, ainda assim oferece desvantagens do ponto de vista da escalabilidade (o IdP é um ponto de falha), da privacidade (o IdP pode ter controle total sobre a identidade) e do fato que os SPs e IdP necessitam estar no mesmo domínio administrativo.

O modelo federado é uma evolução do modelo centralizado. Neste modelo, os SPs e o IdP podem estar em domínios administrativos diferentes. O SSO, no caso federado, também pode ser realizado entre domínios administrativos diferentes (denominado *Cross Domain SSO*). No modelo federado é possível também existir vários IdPs, com uma relação MxN entre IdPs e SPs.

Uma federação representa um conjunto de organizações que cooperam entre si de acordo com regras de confiança pré-estabelecidas para a autenticação de usuários e compartilhamento de recursos [Switch 2011]. Para tal, estas unem-se através de círculos de confiança (CoT - *Circles of Trust*). Uma solução de federação é considerada desejável quando organizações crescem com a aquisição de novos sites, para a manutenção distribuída de repositórios e para simplificar a autenticação e autorização de usuários a recursos e aplicações entre domínios parceiros [Ping Identity 2010a]. As entidades a seguir são normalmente encontradas em uma federação:

- Provedor de serviço: é a entidade mais próxima ao recurso do domínio em questão. É responsável por verificar as permissões de acesso aos recursos por usuários autenticados. Como exemplo, o SP poderia ser um provedor, que oferece serviços de nuvem para seus clientes.
- Provedor de identidades: é um provedor de serviços de identidades. Sua responsabilidade é manter a base de dados de usuários do domínio e validar as credenciais de usuários. Como exemplo, pode ser uma empresa que gerencia contas para um grande número de usuários que precisam de acesso seguro a transações bancárias. Por meio dessa entidade os usuários fornecem as suas credenciais para acessarem os recursos federados.
- *IdP proxy*: é a entidade responsável por interrogar o usuário a respeito de seu domínio de origem. O domínio de origem é o local onde o usuário possui uma identidade em um IdP. Geralmente, o IdP *proxy* apresenta uma lista de IdPs para o usuário selecionar o mais apropriado.

As duas primeiras entidades são geralmente encontradas em federações onde os participantes estão localizados em um mesmo domínio administrativo. Neste caso dizemos que a federação é do tipo intra-domínio e ocorre entre as diferentes aplicações do domínio. Em federações cujos elementos estão dispostos em domínios administrativos distintos, dizemos que a federação é do tipo inter-domínios. Neste caso, e dependendo da arquitetura adotada, o elemento IdP *proxy* atua como uma ponte entre esses domínios.

Por fim, o modelo centrado no usuário, uma evolução do modelo federado, tem como objetivo resolver uma das principais críticas em relação aos modelos anteriores,

que diz respeito à identidade do usuário, ou seja, esta fica armazenada nos provedores de identidades. Estes podem ter total controle sobre estas informações. O modelo centrado no usuário visa então oferecer mecanismos para que um usuário possa escolher que tipo de informações deseja liberar para um determinado provedor de identidades. Exemplos de soluções que utilizam o modelo centrado no usuário são: OpenID [OpenID Foundation 2011], Microsoft CardSpace [Microsoft 2011] e o Projeto Higgins [Eclipse Foundation 2011]. A Fig. 5.5 apresenta os modelos para a IdM.

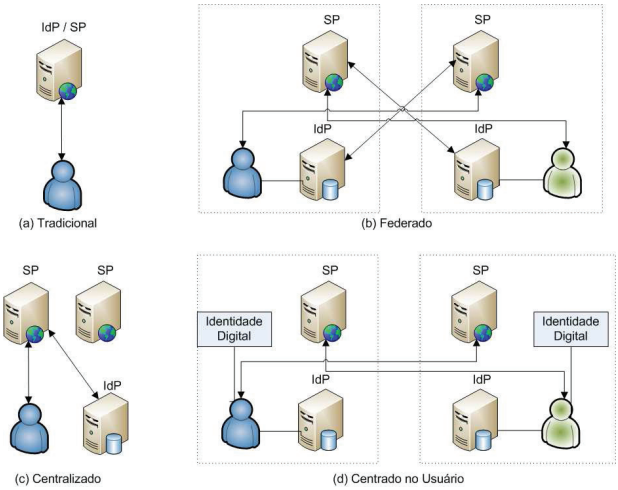

Fig. 5.5. Modelos para a IdM. Baseado em [Wangham et al. 2010]

5.3.4. Mecanismos para Localização de Provedores de Identidades

Em arquiteturas cujo modelo é federado ou centrado no usuário há a problemática da localização de provedores de identidades para a autenticação do usuário. Nestas arquiteturas, há a possibilidade de haver diversos provedores de identidades, e estes, por sua vez, podem estar espalhados em domínios administrativos diferentes. Há duas técnicas empregadas pelos padrões em gerência de identidades federadas:

- Localização baseada em um serviço: nesta técnica utiliza-se um provedor de serviço de localização de identidades. Este serviço conhece todos os possíveis provedores de identidade alcançáveis a partir do provedor. No processo de autenticação de um usuário, este serviço será invocado para apresentar uma lista com os possíveis provedores de identidades conhecidos e confiáveis. O usuário então escolhe um provedor de identidades (onde ele possui uma identidade) e então é redirecionado para este provedor. Caso o usuário não tenha nenhuma identidade em nenhum provedor conhecido, este usuário deverá criar uma identidade. Esta abordagem é utilizada pelo padrão SAML, no perfil *IdP Discovery*.

- Localização baseada em atributos da identidade: esta abordagem utiliza um atributo existente na própria identidade para descobrir onde está o provedor de identidades que autentica esta identidade. O padrão OpenID utiliza como identificador da identidade o formato URL.

5.3.5. Opções para Implantação de SSO

A implantação de sistemas para IdM geralmente envolve a instalação de um *middleware* que ficará responsável pelos serviços de identidade (autenticação, autorização, entre outros). Uma questão importante no processo de implantação de sistemas para IdM diz respeito ao SSO. Principalmente se o SSO envolver diferentes domínios, questões relacionadas à infraestrutura desses domínios, que desejam ingressar na federação, devem ser levadas em consideração. As seguintes opções para a implantação de SSO devem ser avaliadas [Bertino and Takahashi 2011]:

Baseada em *Broker*

Nesta abordagem, há um servidor central responsável por autenticar os usuários (sujeitos) e emitir uma credencial em forma de *ticket*. Através destes *tickets* é possível ao usuário obter acesso aos recursos protegidos. O Kerberos é um exemplo de implantação de SSO que utiliza a abordagem baseada em *broker*. Esta abordagem está relacionada com o modelo centralizado de IdM. A Fig. 5.6 ilustra esta abordagem.

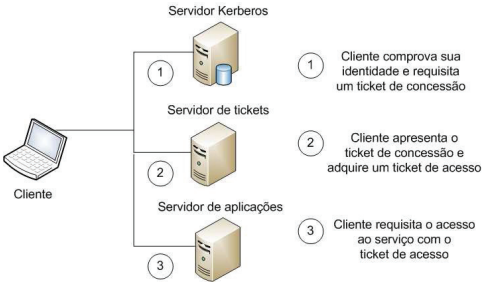

Fig. 5.6. Processo de SSO que Utiliza Abordagem Baseada em *Broker*.

Baseada em Agentes

Esta abordagem utiliza um filtro instalado em um servidor de aplicação Web. Este filtro intercepta todas as requisições que passam neste servidor (passo 1) e, com base em regras pré-estabelecidas, redireciona a requisição para um servidor de autenticação (provedor de identidades, passo 2). Após a autenticação bem sucedida (passo 3), o agente verifica se há alguma credencial válida e, se houver, libera o acesso ao recurso solicitado (passo 4). A Fig. 5.7 ilustra esta abordagem.

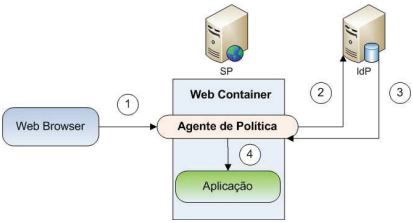

Fig. 5.7. Processo de SSO que Utiliza Abordagem Baseada em Agente.

Baseada em *Proxy* Reverso

Esta abordagem utiliza um *proxy* localizado na borda da rede ou em uma zona desmilitarizada (DMZ). Através deste *proxy*, requisições de acesso a recursos (por exemplo, aplicações) são filtradas (com o uso de um agente, passo 1) e o usuário é redirecionado para um servidor de autenticação (provedor de identidades, passo 2). Após a autenticação bem sucedida, o usuário é redirecionado de volta para o *proxy* (passo 3), que por sua vez estará apto a liberar o acesso redirecionando o usuário para o recurso protegido (passo 4). Nesta abordagem, é possível através do *proxy*, o redirecionamento para qualquer domínio ao qual este tenha alcance. Um problema com esta abordagem é que há uma perda de eficiência uma vez que todas as requisições passam pelo mesmo *proxy*. Uma vantagem é que não é necessário instalar componentes adicionais nos servidores de aplicação para protegê-los. A Fig. 5.8 mostra esta abordagem.

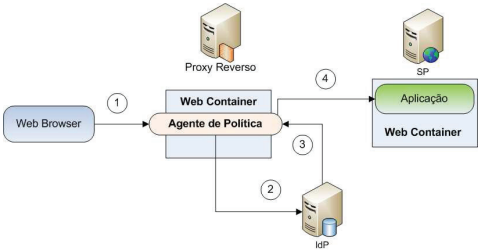

Fig. 5.8. Processo de SSO que Utiliza Abordagem Baseada em *Proxy* Reverso.

Baseada em API

Nesta abordagem há a inclusão de uma API relacionada com o *middleware* para IdM nas aplicações a serem protegidas. Este mecanismo é o mais intrusivo de todos, uma vez que cada aplicação necessita chamar as primitivas oferecidas pela API. Um fator positivo é que nesta abordagem é possível obter um nível de controle de acesso mais sofisticado se comparado com as demais abordagens. No passo 1, a requisição para acessar a aplicação Web é interceptada pela API que está na própria aplicação. No passo 2, a API redireciona

o *browser* para o provedor de serviço de identidades. No passo 3, após a autenticação bem sucedida, este provedor redireciona o *browser* do usuário para a API. No passo 4, a API informa à aplicação que o usuário está autenticado e a aplicação segue o seu fluxo de execução. A Fig. 5.9 ilustra esta abordagem.

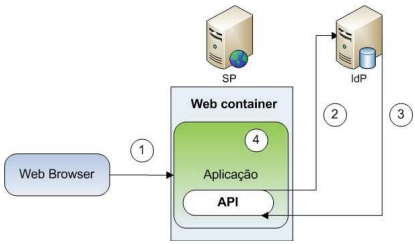

Fig. 5.9. Processo de SSO que Utiliza Abordagem Baseada em API.

Baseada em Serviços

Esta abordagem é parecida com a abordagem baseada em API. A diferença é que neste caso as chamadas são realizadas remotamente (passos 2 e 3), pois são oferecidas diretamente pelo provedor de serviço de identidades. A Fig. 5.10 mostra esta abordagem.

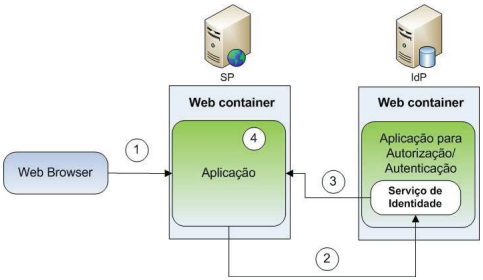

Fig. 5.10. Processo de SSO que Utiliza Abordagem Baseada em Serviços.

5.3.6. Auditoria

O propósito da auditoria para a IdM é auxiliar a resolução de questões relacionadas com o inventário de identidades. A auditoria deve ser realizada periodicamente, por exemplo uma vez ao ano. No processo de auditoria, o inventário é atualizado visando oferecer informações para a avaliação de riscos referente às políticas de segurança, privacidade, entre outras [Windley 2005].

Um desafio importante para que a auditoria funcione em um ambiente de computação em nuvem diz respeito ao problema da falta de visibilidade no acesso de

aplicações oferecidas como SaaS. Este fato é uma consequência dos usuários estarem acessando as aplicações através de uma rede pública (por exemplo, a Internet) ao invés de estarem conectados em uma rede privada (por exemplo, a rede de uma universidade ou empresa). Isto prejudica o uso de ferramentas de monitoramento de rede, uma importante aliada da auditoria [Olden 2011].

5.3.7. Privacidade

O termo privacidade é definido como o direito de uma entidade (sujeito) em determinar o grau de interação que suas informações pessoais (atributos de sua identidade) devem ter perante o contexto onde a identidade está inserida. Este contexto inclui o grau de comprometimento na divulgação destas informações a terceiros [Júnior et al. 2010]. No âmbito da IdM, tendo em vista que as identidades estão logicamente inter-relacionadas entre os diferentes domínios de segurança, a privacidade pode ser vista como a possibilidade de um sujeito determinar quais informações (atributos) de sua identidade serão divulgadas no processo de federação de forma a manter o anonimato entre as diferentes identidades na federação. Uma técnica para a preservação da privacidade em IdM é utilizar pseudônimos. Pseudônimos são identificadores que não permitem inferências em relação à identidade real (e seus atributos) do usuário [Wangham et al. 2010]. Os pseudônimos podem ter significado local (dependente do contexto entre o usuário e um SP específico) ou global, e podem ter validade temporária (durante o tempo de uma sessão) ou permanente.

5.4. Gerência de Identidades Federadas em Nuvens

No setor acadêmico, ambientes de computação em nuvem geralmente pertencem a domínios privados mantidos por centros de pesquisa e universidades. Muitas vezes, nestes domínios, estão recursos cujo acesso pode ser realizado via Internet. Neste caso, é essencial que existam mecanismos para prover a autenticação e a autorização de usuários no acesso aos recursos, tanto para usuários vindos do próprio domínio, quanto de outros domínios parceiros. Estes mecanismos possibilitam a interoperabilidade de recursos localizados em domínios distintos, habilitando então a cooperação entre centros de pesquisa parceiros.

Para que a cooperação seja realizada com êxito, as entidades parceiras devem definir políticas para o compartilhamento de recursos junto aos domínios federados [Gomes and Kowalczyk 2011]. Esse processo envolve mecanismos de SSO para que, uma vez que o usuário esteja autenticado em seu domínio de origem, ele possa acessar recursos em quaisquer outros domínios federados, desde que esse usuário esteja inserido no contexto de autorização do domínio em questão. Em domínios não-federados, o mesmo usuário precisaria ter credenciais em ambos os domínios para ter permissões de acesso aos recursos distribuídos [Cao and Yang 2010].

O uso de padrões abertos assegura a interoperabilidade em ambientes de computação em nuvens híbridas. No entanto, os serviços oferecidos em nuvens ainda estão em processo de desenvolvimento e ainda existem muitos desafios para a provisão de segurança na integração de aplicações de domínios distintos. Outro fato é que nem todas as aplicações de uma empresa podem ser disponibilizadas na nuvem. Ao

mesmo tempo em que alguns serviços podem ser contratados de terceiros (tais como espaço de armazenagem, aplicações Web para *e-mail*, publicação *on-line* de documentos e hospedagem de páginas Web), outros serviços são exclusivos do uso interno da empresa e não devem ser disponibilizados publicamente (folha de pagamentos, relatórios empresariais e aplicativos internos de controle de produção, dentre outros).

Uma área da computação em nuvem onde a IdM tem papel importante é no cenário de *mash-ups*. Um *mash-up* pode ser caracterizado por uma página Web ou aplicação que compartilha, utiliza ou integra dados, apresentação e funcionalidade de uma ou mais fontes criando um novo serviço [Crupi and Warner 2008]. Um exemplo seria um *mash-up* entre um *blog* e um serviço de armazenamento de fotos, oferecendo aos usuários a possibilidade de postar mensagens com foto, sendo que estas fotos podem estar em diferentes sites. Neste cenário, o controle de acesso aos recursos compartilhados (fotos) deve ser baseado em políticas bem definidas e deve avaliar o pedido de acesso ao recurso com base na autenticação prévia do usuário feito pelo *blog*.

Provedores tais como Oracle, Google, Salesforce, IBM, Amazon e Microsoft são exemplos de empresas que começaram a estabelecer novos *datacenters* para a provisão de serviços de nuvem. A infraestrutura desses provedores deve comportar a habilidade de expandir/reduzir a capacidade de suas aplicações sob demanda para o fornecimento de conteúdo para redes sociais, aplicações comerciais, multimídia, jogos, entre muitos outros.

O modelo de acesso a aplicações em plataformas de computação em nuvem segue o modelo *multi-tenant* (multi-locatário), em que uma mesma aplicação passa a ser utilizada por múltiplos locatários (*tenants*), que são desde usuários convencionais a empresas. Esse modelo é interessante para aplicações em nuvem porque múltiplos usuários podem compartilhar os recursos físicos utilizados para prover um aplicativo, ao mesmo tempo em que as operações realizadas por um usuário não são vistas por outro.

Provedores de serviços tais como a Amazon, Salesforce.com e Aol, entre outros, possuem os seus próprios sistemas de gerência de identidades, mas não realizam processos de SSO. Infraestruturas para gerência de nuvens tais como Abiquo, OpenStack, Eucalyptus, Proxmox e OpenNebula não possuem um mecanismo para gerência de identidades federadas padronizado. Os sistemas Web existentes ainda são limitados quanto à distribuição automatizada de recursos entre nuvens, respeitando quesitos de QoS. Ambientes de computação federados devem ser capazes de adotar medidas para a provisão escalável de serviços, respeitando quesitos de QoS, condições de rede e variação de carga entre os diversos servidores associados.

A gerência de identidades federadas pode ser implantada de diversas maneiras. Segundo [Bertino and Takahashi 2011], a primeira delas é o chamado *in-house*. Nesta configuração, as identidades são expedidas e gerenciadas pelo próprio domínio (organização). A outra maneira é oferecer como um serviço tercerizado. Este cenário é chamado de identidade como um serviço (IDaaS - *Identity as a Service*). Nesta configuração, as identidades são expedidas e gerenciadas por provedores IDaaS. Em um cenário de hospedagem gerenciada, a organização contratante exporta toda a sua base de usuários ao provedor IDaaS terceirizado. Em outro cenário, o provedor IDaaS mantém somente pseudônimos das identidades dos usuários da organização, que por sua vez são

mapeadas para identidades de seus usuários reais.

5.4.1. Desafios da Gerência de Identidades Federadas em Nuvens

O desafio da gerência de identidades federadas está relacionado com a problemática de diferentes provedores de serviços utilizarem diferentes mecanismos de mapeamento de usuários em contextos de autenticação [El Maliki and Seigneur 2007]. A gerência de identidades federadas tem o objetivo de unificar a validação de contexto de usuários nos diferentes domínios parceiros. Isso significa que é preciso manter as devidas restrições e controle de acesso de acordo com o contexto em que o usuário está inserido no momento. As identidades de usuários fornecidas para aplicações podem ser do tipo usuário/senha, certificados digitais, *tokens*, biometria, cartões, entre outros [Cao and Yang 2010].

Dentre os principais desafios da gerência de identidades federadas está o acesso seguro a dados de outros domínios, sem que seja necessário a replicação dos serviços administrativos. Por isso é necessário que os mesmos protocolos sejam utilizados pelos domínios parceiros, ou que esses protocolos sejam interoperáveis. Como exemplo, a interface OCCI [Open Grid Forum 2009] é relativamente recente (início do projeto em 2009) e é um dos esforços da comunidade do Open Grid Forum (OGF) para definir como provedores de serviços podem oferecer seus recursos de computação, dados e rede através de interfaces padronizadas. O objetivo do projeto é promover interoperabilidade entre provedores de nuvem, sem a necessidade de tradução de padrões de formatação de dados para a criação, gerência e representação de recursos de um domínio para outro. Infraestruturas de nuvem como OpenNebula e Eucalyptus têm suporte para a interface OCCI, além de oferecerem interfaces para a EC2 Query API da Amazon, bem como APIs próprias de interconexão com outros provedores de nuvem.

Nesse novo modelo, um conjunto de aplicativos, infraestrutura e mesmo plataformas são oferecidos como serviços que não precisam estar fisicamente mantidos nos domínios de uma mesma organização. Os serviços são distribuídos em uma nuvem de recursos armazenados em diversas organizações. Em virtude desse modelo, muitos desafios se tornam inerentes para a provisão de segurança no acesso a serviços de múltiplos parceiros. A seguir são apresentados alguns deles [Buyya et al. 2010]:

- *Single Sign-On*: o acesso a serviços de domínios externos deve ser protegido por meio de serviços de autenticação. O desafio nesse caso está em garantir que uma única credencial seja fornecida e validada entre múltiplos domínios, sem a necessidade do usuário manter uma conta por domínio.

- Segurança para o acesso a recursos distribuídos: o uso de serviços de *datacenters* remotos deve contemplar mudanças nas políticas de provisão de recursos, uma vez que esses recursos poderão ser utilizados por qualquer um dos domínios parceiros. Em nuvens é desejável que tanto os usuários quanto as aplicações da nuvem sejam capazes de acessar múltiplos recursos, e que os usuários de um domínio sejam reconhecidos em outro, sem a replicação das bases de dados de usuários. Ainda que cada domínio possa manter suas próprias políticas de acesso aos serviços por ele oferecidos, outras políticas de controle de acesso podem estar distribuídas em muitas localizações da rede, e um mesmo usuário pode requerer

múltiplas regras para acessar os recursos de um mesmo serviço. Além disso, um mesmo usuário pode ter diferentes identidades, uma por domínio, e o ambiente da nuvem deve ser capaz de mapeá-las para um mesmo usuário. Por outro lado, as declarações (*entitlements*), que definem o conjunto de políticas de acesso, devem ser interoperáveis quanto ao formato de representação, ainda que cada domínio tenha a liberdade de especificar as suas próprias políticas de controle de acesso aos seus recursos locais.

- Dinamicidade: o modelo da computação em nuvem representa o cenário atual do comércio eletrônico, onde os relacionamentos entre os recursos ofertados/produzidos mudam rapidamente. A IdM em nuvens deve considerar a natureza dinâmica da provisão de recursos virtuais e físicos, seja por meio da alteração dinâmica de servidores para fins de balanceamento de carga durante uma migração, ou da provisão sob demanda (aplicações elásticas) de mais recursos para os usuários/aplicações. É comum que nesses ambientes servidores e máquinas virtuais sejam iniciados e finalizados dinamicamente, sendo que a IdM deveria ser informada que o acesso futuro foi permitido ou revogado [Gopalakrishnan 2009]. Além disso, o acesso deve ser monitorado e todos os provedores de serviços devem cumprir os acordos de SLA entre múltiplos domínios. Atrasos na (des)provisão de serviços poderia levar a sérios riscos de segurança [Gopalakrishnan 2009]. Para esses casos, a linguagem SPML (*Service Provisioning Markup Language*) [OASIS 2011] fornece uma descrição em estruturas XML para representar a (des)provisão de requisições para a IdM. Apesar de não ser obrigatório, a SPML utiliza o protocolo SAML.

- Escalabilidade: ambientes de computação em nuvem devem ser capazes de atender a um número relativamente grande de conexões, identidades e transações em um curto espaço de tempo. Dessa forma, o tempo de resposta para realizar as autenticações/autorizações deveria considerar os momentos de pico de consumo de recurso na rede.

- Interoperabilidade: atualmente existe uma corrida em torno de padrões para IdM. Dentre as várias alternativas podem ser citadas as soluções de conectividade da Microsoft e os padrões abertos OAuth, OpenID e SAMLv2, entre outros.

Outro desafio é a oportunidade dos provedores escolherem os domínios mais apropriados para atenderem às suas necessidades atuais. Dessa forma, um típico serviço de rede social, por exemplo, pode utilizar serviços de muitos outros provedores de nuvem, desde a simples armazenagem de dados até o processamento de grande quantidade de informações. Ainda assim, essa oferta deve ser dinâmica, ou seja, os mecanismos de interação com outros domínios devem facilitar tanto a requisição quanto a liberação de recursos adicionais, quando estes não forem mais necessários.

Outro desafio ligado à oferta de aplicações elásticas em ambientes federados é o fato de que os provedores podem estar localizados em diferentes regiões geográficas. Os clientes devem ser capazes de indicar em qual localidade preferem que seus dados sejam armazenados, além de indicar suas preferências sobre aumentar ou não a provisão de seus

recursos de acordo com a demanda. Por outro lado, um provedor pode terceirizar a sua própria oferta de serviços para atender aos quesitos de QoS de seus clientes. De acordo com [Buyya et al. 2010], uma vez que é inviável o estabelecimento físico de servidores em cada uma das regiões onde os serviços são acessados, os seguintes requisitos devem ser atendidos por ambientes de computação em nuvem federados:

- Oferta dinâmica de armazenagem e recursos computacionais de outros provedores de nuvem.
- Negociação de acordos quanto ao nível de serviço ofertado/consumido (SLA) entre os domínios federados.
- Garantias quanto à oferta de serviços para garantir padrões de QoS e minimizar os custos para o cliente final, de forma semelhante às tarifas de água, luz ou gás natural.

De acordo com o consumo de seus clientes a infraestrutura de nuvem deve ser capaz de prever comportamentos para manter-se em plena operação a maior parte do tempo utilizando, por exemplo, algoritmos de aprendizagem, análise estatística do uso da rede e consumo de processamento e/ou armazenagem. Em infraestruturas do tipo SaaS a IdM se preocupa em gerenciar o controle de acesso às aplicações. Por outro lado, infraestruturas do tipo PaaS requerem o controle de acesso à plataforma e às aplicações desenvolvidas na plataforma. Os requisitos para IdM em IaaS são similares às de PaaS, com a preocupação de controlar o acesso à infraestrutura de armazenagem, processamento e demais recursos de *hardware* disponibilizados na nuvem [Gopalakrishnan 2009]. A principal limitação da IdM em todos esses modelos é a perda de interoperabilidade caso os serviços de uma nuvem sejam migrados para outra.

O controle de acesso tradicional, centrado na aplicação, onde cada aplicação mantém e gerencia uma base de dados de usuários ou compartilha esta base de dados entre diversas aplicações não são adequados para ambientes de computação em nuvem federada. Isso porque em tais ambientes há o compartilhamento de informações entre múltiplos provedores de serviços. A replicação de identidades para servir cada aplicação (*one user-to-one app*) não é adequado devido ao crescimento exponencial [Olden 2011]. Além disso, este modelo exige que os usuários memorizem múltiplas identidades para cada novo serviço oferecido pela nuvem. O compartilhamento de bases entre aplicações de um mesmo domínio pode ser uma alternativa, porém entre domínios federados não é adequado, pois implica em questões de segurança no acesso à base. Olden argumenta que para limitar a replicação de identidades em uma nuvem, o modelo de uma identidade para muitas aplicações (*one user-to-many apps*) é o mais adequado, ou seja, a identidade do usuário deve estar federada com estas aplicações.

Outro desafio para a IdM em nuvens é que atualmente cada solução para gerência de recursos da nuvem utiliza uma solução própria para gerir as identidades e o acesso a estes recursos. Dado o contexto de federação/cooperação ao qual os provedores de nuvem estão situados, a demanda por uma solução padronizada, que contemple todas as nuances, ainda é esperada. Por exemplo, para a realização de SSO entre provedores

de nuvem, a Cloud Security Alliance recomenda que os provedores de serviços devem ser flexíveis e que aceitem os formatos utilizados pelos provedores de identidades [Cloud Security Alliance 2009]. A entidade também faz ressalvas no uso de protocolos proprietários.

O IDaaS deve ser capaz de suportar diferentes mecanismos de autenticação (LDAP, RADIUS, Certificados X.509, etc.), autorização (XACML, OAuth, dentre outros), federação (SAML, OpenID, InfoCard, etc.), diversas bases de dados (LDAP, MySQL, OpenDS, etc.), ser fracamente acoplado e não invasivo nas aplicações SaaS [Gopalakrishnan 2009] [Olden 2011]. Olden argumenta também que as organizações não devem integrar diretamente as aplicações oferecidas em nuvem para realizar SSO e acesso a recursos. Ao invés disto, devem federar com o IDaaS, que por sua vez está pré-integrado às diversas aplicações.

5.4.2. Desafios da Privacidade em Nuvens

Os desafios da privacidade em nuvens estão nos modelos de controle de acesso empregados. Os modelos centrados no usuário são mais adequados para ambientes de computação em nuvem: as requisições aos SPs são tratadas de acordo com a identidade do usuário e suas credenciais. O controle de acesso centrado no usuário precisa conter informações que definam unicamente um usuário no domínio. Isso implica em manter um contexto de informação por usuário, ao invés de procurar a melhor forma de reagir, em determinada situação, a determinada requisição. O modelo deve suportar pseudônimos e múltiplas e discretas identidades na proteção da privacidade do usuário [Gopalakrishnan 2009]. O modelo centrado no usuário não coloca apenas o usuário no controle de sua identidade, mas também adiciona mais transparência em relação a quais dados estão sendo compartilhados e com quem. Os usuários ficam mais confiantes dado que estão dando permissões apropriadas a cada serviço utilizado.

5.5. Padrões e Soluções para Implantação de Gerência de Identidades Federadas

Esta seção apresenta alguns dos principais padrões e soluções utilizadas para a implantação de gerência de identidades federadas. Dividimos as subseções a seguir em padrões para autenticação e para autorização.

5.5.1. Padrões para Autenticação

SAML (Security Assertions Markup Language)

SAML é um padrão aberto baseado em XML para a troca de informações de autenticação e autorização em domínios que participam de um mesmo CoT [OASIS 2008]. A especificação SAML define uma linguagem e um protocolo para a troca segura de informações, com a finalidade de prover o SSO e identidades federadas entre os domínios do CoT, independente da arquitetura que utiliza o SAML.

O protocolo SAML é interoperável com outros protocolos, tais como WS-Trust, WS-Federation, e desacopla o provedor de identidades (ou seja, o IdP atua como provedor de asserções SAML) e o provedor de serviços (ou seja, o SP atua como um consumidor

de asserções SAML). Uma asserção SAML é um documento em formato XML que é gerado por um IdP e contém declarações (*statements*) utilizadas pelo SP para tomar as devidas decisões sobre o controle de acesso aos recursos do domínio protegido. As seguintes declarações são definidas em uma asserção SAML: a) Declarações de autenticação: descrição que indica se o usuário foi autenticado pelo IdP em alguma interação anterior; b) Declarações de atributos: descrição de atributos (um par do tipo nome/valor) que é utilizado pelo CoT para tomar decisões sobre o controle de acesso, com base nesses atributos; c) Declaração de decisão de autorização: descrição que indica se o usuário/aplicação, de acordo com uma determinada restrição, pode ou não acessar o recurso.

O protocolo não especifica como os serviços do IdP devem ser implementados, mas espera que o IdP forneça os serviços de autenticação para o domínio local. Ao receber as asserções SAML do IdP, o SP decide quais serão as políticas quanto ao controle de acesso. Mais detalhes serão apresentados nas seções 5.5.3 e 5.6.4.

OpenID

OpenID é um protocolo aberto para a provisão de identidades federadas que permite o SSO para os usuários cadastrados em sites confiáveis do mesmo CoT. Com o OpenID, a credencial do usuário é fornecida para apenas um provedor de identidades, o qual identifica o acesso desse usuário aos outros sites que ele acessa [OpenID Foundation 2011]. Portanto, não é necessário que o usuário crie uma nova conta para acessar recursos de outros sites do mesmo CoT, mas é necessário que o usuário seja autenticado em pelo menos um deles.

Grandes provedores de identidades como Google, Facebook, Yahoo, Microsoft, Aol, MySpace e PayPal, entre muitos outros, fornecem suporte para o OpenID. O SSO é realizado sem a necessidade de relacionamento de confiança pré-existente entre os IdPs e os RPs (*relaying parties*), o que facilita a sua adoção [Ping Identity 2010b].

OpenID é um protocolo que utiliza redirecionamentos HTTP entre o usuário/aplicação e o provedor de identidades. As requisições para acessar o serviço de autenticação são baseadas no uso do protocolo HTTP. Primeiramente, o usuário deve se registrar em um provedor de identidades com suporte ao OpenID. O provedor de identidades, por sua vez, utiliza o nome da conta do usuário para gerar uma URL específica por usuário. Essa URL é utilizada pelo aplicativo cliente do usuário como argumento de localização do serviço remoto de autenticação do usuário, ou seja, a autenticação é vista como um serviço fornecido por um dos provedores de identidade do CoT. A partir dessa URL, o aplicativo cliente é capaz de identificar qual é o provedor de identidades remoto no qual o usuário possui uma conta. A seguir, caso o usuário ainda não tenha se autenticado, ele deve fornecer as suas credenciais no serviço de autenticação do provedor de identidades especificado na URL. OpenID também confere mecanismos de delegação de regras entre os provedores do mesmo círculo de confiança.

A Fig. 5.11 ilustra o mecanismo básico de autenticação federada com o OpenID. No passo 1, o usuário com uma identidade registrada em um provedor OpenID (usuário com conta de *e-mail* no Google, por exemplo), deseja acessar recursos de um Web

site no qual ele não está cadastrado. No entanto, esse Web site possui um serviço de autenticação OpenID. Para proceder com o acesso aos recursos protegidos do site, no passo 2 o usuário fornece a URL para acesso federado que recebeu de seu provedor de identidades. No passo 3, o serviço OpenID do Web site redireciona o *browser* do usuário para o serviço de autenticação do provedor de identidades, informado na URL fornecida anteriormente. No passo 4, duas opções são possíveis. Caso o usuário já esteja autenticado no provedor, o *browser* é redirecionado para o serviço de verificação do endereço de origem que solicitou a autenticação com o OpenID. Caso o usuário não esteja autenticado, o provedor de identidades solicita primeiro o nome de usuário e senha, para depois proceder com o redirecionamento. No passo 5, o Web site do provedor de identidades utiliza um serviço de verificação, para validar o endereço do serviço que solicitou a autenticação com o OpenID. Uma razão é que grande parte dos sites solicitam informações adicionais dos novos usuários, como o nome do usuário, *e-mail* e telefone, dentre outros. Essas informações podem ser fornecidas nesse passo, sem a necessidade de solicitá-las novamente após a autenticação. Caso o usuário permita que essas informações sejam fornecidas, no passo 6 o provedor de identidades redireciona o usuário para o Web site inicial.

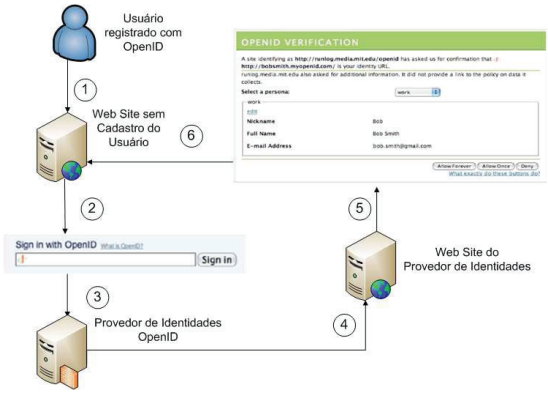

Fig. 5.11. Mecanismo Básico de Autenticação Federada com o OpenID.

5.5.2. Padrões para Autorização

XACML (Extensible Access Control Markup Language)

XACML 2.0 [OASIS 2005] é um padrão que define uma linguagem declarativa, utilizada para a descrição de políticas de controle de acesso, e uma linguagem para formular consultas (e obter respostas) utilizada para verificar se uma determinada ação é ou não permitida. Como resposta à consulta, um dos seguintes valores pode ser retornado: *Permit*, *Deny*, *Indeterminate* (um erro ocorreu, ou algum valor não foi especificado, impedindo o prosseguimento da consulta), ou *Not Applicable* (quando a resposta não puder ser retornada pelo serviço solicitado).

O padrão define também uma arquitetura para a aplicação de políticas baseada no modelo abstrato definido por [IETF 2000] e [ISO/IEC 1996]. Nesta arquitetura estão presentes os seguintes componentes:

- *Policy Enforcement Point (PEP)*: componente da arquitetura responsável por interceptar requisições a recursos/serviços e aplicar a decisão vinda do PDP.
- *Context Handler*: entidade responsável por traduzir as requisições a recursos de um formato nativo ao sistema para o formato XACML e vice-versa.
- *Policy Decision Point (PDP)*: componente responsável por avaliar as políticas aplicáveis e devolver ao PEP uma decisão de autorização.
- *Policy Administration Point (PAP)*: responsável pela criação das políticas.
- *Policy Information Point (PIP)*: componente responsável por obter informações/atributos adicionais e encaminhá-los ao PDP mediante solicitação.

A Fig. 5.12 apresenta uma visão geral, com os componentes centrais, PEP e PDP, atuando em uma requisição a um recurso protegido.

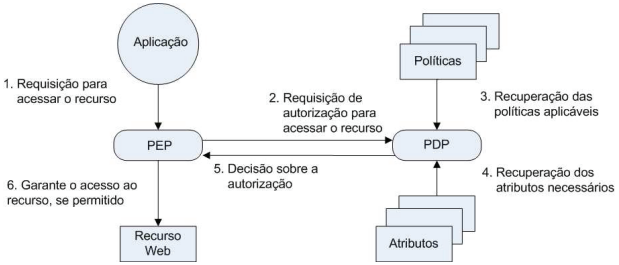

Fig. 5.12. Visão Geral da Arquitetura do XACML. Adaptado de [Anderson 2005]

Uma típica requisição solicita um recurso protegido por um servidor Web, o qual atua como um PEP (passo 1). O PEP forma uma requisição com os atributos da requisição original, que são as informações sobre o(s) recurso(s) que o usuário/aplicativo deseja interagir, a ação para a interação e outras informações específicas da requisição. Essa requisição é enviada para o PDP (passo 2), que verifica se existe uma política que se aplica a essa requisição (passo 3), verifica atributos adicionais para a tomada de decisão (passo 4) e retorna uma resposta para o PEP, contendo a decisão de autorização (passo 5). O PEP, por sua vez, irá aplicar a decisão do PDP de permitir ou negar o acesso ao recurso (passo 6).

Os documentos desta linguagem utilizam o formato XML estruturado em três níveis (*policy language model*): a) *PolicySet*: descrição de um conjunto de políticas ou outros *PolicySets*; b) *Policy*: descrição de uma política específica. Políticas são definidas como um conjunto de regras; c) *Rule*: descrição de uma regra. Regras definem se

um recurso pode ou não ser acessado, a partir dos atributos fornecidos e das restrições definidas em cada regra. Um documento XACML também especifica um *Target*. Caso todas as condições para o *Target* forem satisfeitas, então *PolicySet*, *Policy* e *Rule* relativas a esse *Target* são aplicadas à requisição. As condições para um *Target* são satisfeitas por meio de comparações dos valores da requisição com os definidos para o *Target*.

Os atributos de uma requisição são características do usuário/aplicação (sujeito), tais como o nome, perfil de acesso, recurso que o usuário quer acessar, período (data e hora) em que se quer acessar o recurso, entre outros. Os atributos também são características definidas para o *recurso* a ser acessado, para a *ação* a ser tomada e para o *ambiente* (localização do sujeito). Portanto, o PDP lida com comparações lógicas sobre esses valores (por exemplo, por meio de consultas XPath), de forma a responder à requisição do PEP para o acesso ao recurso protegido. O Quadro 5.1 ilustra um exemplo de documento XACML para descrever uma política de controle de acesso que permite a todos os usuários do domínio *realcloud.dca.fee.unicamp.br* acessarem recursos protegidos.

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy PolicyId="identifier:example:SimplePolicy1"
RuleCombiningAlgId=
  "identifier:rule-combining-algorithm:deny-overrides">
  <Target>
      <Subjects><AnySubject/></Subjects>
      <Resources><AnyResource/></Resources>
      <Actions><AnyAction/></Actions>
  </Target>
  <Rule RuleId="identifier:example:SimpleRule1? Effect="Permit">
      <Target>
        <Subjects>
          <Subject>
              <SubjectMatch MatchId="function:rfc822name-equal">
                <SubjectAttributeDesignator AttributeId="identifier:subject:subject-id"
                 DataType="identifier:datatype:rfc822name"/>
                <AttributeValue DataType="identifier:datatype:rfc822name">
                 *@realcloud.dca.fee.unicamp.br</AttributeValue>
              </SubjectMatch>
          </Subject>
        </Subjects>
        <Resources><AnyResource/></Resources>
        <Actions><AnyAction/></Actions>
      </Target>
  </Rule>
</Policy>
```

Quadro 5.1. Exemplo de Documento XACML para Descrever uma Política. Baseado em [Stoller 2011].

Dentre as vantagens da linguagem XACML para a definição de políticas está o fato dela ser descrita em XML. Com isso XACML pode ser estendida e reutilizada entre diversas aplicações/organizações. Não há a necessidade de reescrever uma política em diversas linguagens diferentes para cada cenário. Outro fator é que ela pode ser utilizada nos mais diversos cenários, uma vez que ela é de propósito geral [Stoller 2011].

OAuth

O protocolo OAuth é um padrão aberto de autenticação que fornece um mecanismo para que um usuário/aplicação possa compartilhar recursos na Web com terceiros sem ter que compartilhar a sua senha [OAuth Community 2011]. Esse protocolo também fornece uma alternativa para autorizar o acesso a esses recursos por um determinado tempo.

Dessa forma, um usuário proprietário de um recurso (um álbum de fotos, por exemplo) pode oferecer acesso temporário a esse recurso sem compartilhar a sua

identidade (nome de usuário e senha). Usuários que desejam ter acesso a esse recurso o fazem por meio de um serviço de delegação. No modelo OAuth existem três elementos principais no processo de autorização: o dono do recurso, o cliente e o servidor. Para o cliente acessar um recurso ele deve primeiramente obter uma autorização do dono do recurso, ou seja, o cliente recebe do dono do recurso uma permissão para acessar temporariamente o recurso, na forma de um *token* e uma chave secreta compartilhada. OAuth também verifica a autorização do dono do recurso e a identidade do cliente que faz a requisição a esse recurso [OAuth Community 2010].

O cenário descrito acima é semelhante ao utilizado pelo OAuth para autorizar o acesso a recursos Web a partir de provedores como o Twitter [Twitter 2011]. A Fig. 5.13 ilustra um cenário onde um usuário publica as suas fotos no site *ServidorRecurso.net* e deseja utilizar outro site para imprimi-las (*Cliente.net*). No entanto, o usuário dono do recurso não quer compartilhar a sua identidade (nome de usuário e senha) com o site de impressão.

No passo 1, ocorre o registro da aplicação cliente do site *Cliente.net* que deseja acessar o site *ServidorRecurso.net*. Um modelo similar aos de chave pública e privada é utilizado. Para cada aplicação registrada será fornecida uma *OAuth consumer key* (chave pública) e uma *OAuth consumer secret* (chave privada). Essas chaves são utilizadas para autenticar o usuário/aplicação (por exemplo, via Twitter API), garantindo que o tráfego seja do usuário portador dessas chaves. Por padrão, a Twitter API envia requisições no cabeçalho das mensagens HTTP (*HTTP Authorization header*).

No passo 2, as aplicações Web registradas devem fornecer uma URL de *callback* que o servidor do recurso irá utilizar para redirecionar o *browser* do usuário após a autenticação. O conjunto de *endpoints* que será utilizado nas interações também deve ser configurado: a) *Temporary Credential Request* - endereço para iniciar a solicitação de acesso ao recurso; b) *Resource Owner Authorization* - endereço para solicitar o serviço de autorização de acesso ao recurso; c) *Token Request URI* - endereço para solicitar *tokens* de acesso.

Antes do cliente ser autorizado a acessar os recursos, no passo 3 ele deve solicitar credenciais temporárias, a partir do endereço informado em *Temporary Credential Request*. O *ServidorRecurso.net* valida a solicitação porque *Cliente.net* possui uma *oath_consumer_key* (chave pública) válida. No passo 3.1, após validar a requisição, o servidor *ServidorRecurso.net* retorna uma mensagem HTTP contendo a *oauth_token* e *oauth_token_secret*.

No passo 4, a aplicação cliente em *Cliente.net* redireciona o *browser* do usuário para o endereço do *Resource Owner Authorization*, com a finalidade de obter a autorização do dono do recurso para acessar as suas fotos. No passo 5, *ServidorRecurso.net/authorize* solicita o acesso do dono do usuário ao recurso, usando o nome de usuário e senha do dono do recurso. Note que esse processo de autenticação ocorre no *ServidorRecurso.net*, ou seja, o dono do recurso se autentica em seu próprio domínio para delegar o acesso ao site *Cliente.net*. Quando confirmada a autenticação, o *ServidorRecurso.net* solicita a autorização do dono do recurso para que *Cliente.net* acesse o recurso. Caso o dono do recurso aprove, o *browser* é redirecionado para a URL de *callback* informada no registro da aplicação.

Até aqui, *Cliente.net* completou o seu processo de autorização. No passo 6, ele solicita o acesso a um *token* utilizando as credenciais temporárias que já obteve até então, ou seja, *oauth_consumer_key*, *oauth_token* e *oauth_verifier*. No passo 6.1, o servidor valida a requisição (como o fez no passo 3.1), porque a requisição possui uma *oauth_consumer_key* válida. O servidor também valida a requisição de acesso ao *token* porque recebeu *oauth_token* e *oauth_verifier* válidos. Após essa validação, o *ServidorRecurso.net* retorna no corpo da mensagem HTTP um conjunto de credenciais temporárias, ou seja, um novo *oauth_token* e um *oauth_token_secret*. Note que o *oauth_consumer_key* não muda durante a interação porque está relacionado com a autenticação do usuário dono do recurso.

No passo 7, o *Cliente.net* possui um conjunto de credenciais temporárias que lhe autorizam a acessar o recurso. Nesse passo, *Cliente.net* solicita um recurso de *ServidorRecurso.net* enviando *oauth_consumer_key* e o novo *oauth_token*. O *ServidorRecurso.net* irá validar o acesso porque recebeu um *oauth_consumer_key* e *oauth_token* válidos. O acesso de *Cliente.net* é permitido enquanto durar o tempo de vida do *token*, ou enquanto o dono do recurso autorizar o acesso.

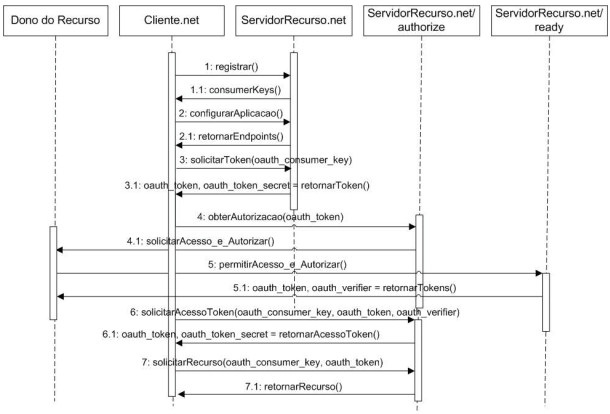

Fig. 5.13. Diagrama de Sequência Simplificado do OAuth.

5.5.3. Soluções

Shibboleth

Shibboleth [Internet2 2011] é um *middleware* gratuito e de código aberto desenvolvido pela comunidade Internet2 que provê uma solução de SSO, baseado em padrões abertos (principalmente XML e SAML) para autenticação e autorização na Web. Shibboleth é um sistema para a criação de federações que oferece funcionalidades para a troca segura de dados para acessar recursos entre domínios. Utilizado inicialmente para integrar instituições acadêmicas, hoje ele é usado por uma variedade de sites em todo o mundo.

Os principais elementos do Shibboleth são o *SP* e o *IdP*. O processo para

determinar o IdP do usuário pode fazer uso de um serviço de descoberta conhecido como *IdP Discovery* que exibe uma lista dos IdP confiáveis cadastrados no círculo de confiança.

Na arquitetura do Shibboleth, o acesso do usuário a recursos é realizado por meio de asserções SAML recebidas de um IdP confiável. Nessa arquitetura, os seguintes componentes são definidos para o SP:

- Recurso Alvo: os recursos Web são protegidos no SP por meio de serviços de Controle de Acesso, o que impede usuários não autenticados/autorizados de acessarem esses recursos.
- Serviço Consumidor de Asserções: é responsável por processar a asserção de autenticação do Serviço de SSO, que foi retornada pelo IdP. Além disso, esse componente redireciona o *browser* do usuário para o recurso desejado.
- Requisitante de Atributos: uma vez que um contexto de segurança tenha sido estabelecido por meio da validação fim-a-fim das asserções SAML trocadas entre IdP e SP, atributos podem ser trocados diretamente entre o SP (com o uso do componente Requisitante de Atributos) e o IdP (com o uso do componente Autoridade de Atributos).

O *IdP Discovery* é tratado na arquitetura como um serviço de *proxy* para realizar o redirecionamento do usuário para o domínio que possui o serviço de autenticação. O *IdP Discovery* também lista o conjunto de IdPs disponíveis para que o usuário escolha o domínio de autenticação mais adequado.

Para o IdP são definidos os seguintes componentes:

- Autoridade de Autenticação: lida com questões relacionadas à autenticação para outros componentes e é responsável por gerar a asserção de autenticação no IdP.
- Serviço de SSO: é responsável por iniciar o processo de autenticação no IdP e verificar a existência e validade dos *cookies* de sessão, e interagir com o componente Autoridade de Autenticação para gerar a asserção de autenticação no IdP.
- Serviço de Resolução: quando o SP define um perfil que utiliza a troca de asserções por referência, esse serviço é responsável por tratar essas requisições. Um artefato é uma referência para uma asserção de autenticação. Ou seja, ao invés de enviar a asserção de resposta de autenticação via o *browser* do usuário, é enviada uma referência a asserção expedida. Esta forma do SP obter a asserção é denominada *artifact binding*, conforme o padrão SAMLv2.
- Autoridade de Atributos: é responsável por autorizar e autenticar requisições que lidam com atributos de usuários.

Como ilustrado na Fig. 5.14, uma requisição de autenticação com o Shibboleth é uma mensagem enviada pelo SP e que contém uma URL enviada para o Serviço de SSO do IdP [Scavo and Cantor 2005]. No passo 1, essa requisição é feita para acessar um recurso protegido pelo SP. Caso já exista um contexto de segurança válido, segue-se para

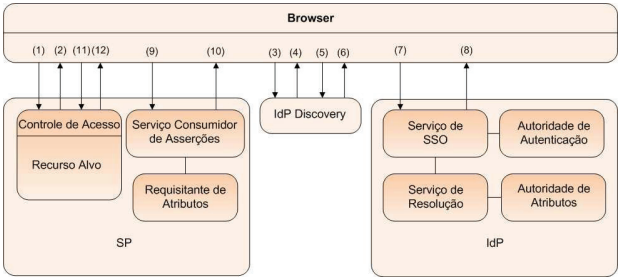

Fig. 5.14. Fluxo de Mensagens na Arquitetura do Shibboleth.

o passo 8. No passo 2, o *browser* do usuário é redirecionado para o *IdP Discovery*. No passo 3, o *IdP Discovery* verifica se existe um *cookie* de sessão e sua validade, e processa uma requisição de autenticação do usuário. Caso o usuário tenha um *cookie* válido, os passos 4 e 5 não são realizados. Caso contrário, um formulário HTML com uma lista de IdPs é fornecida. No passo 4, o *IdP Discovery* fornece uma lista de IdP disponíveis para que o usuário escolha onde deseja ser autenticado. No passo 5, o usuário seleciona o IdP desejado e uma requisição HTTP/GET é enviada novamente para o *IdP Discovery*. No passo 6, o *IdP Discovery* atualiza o *cookie* de sessão com as informações do IdP escolhido pelo usuário e redireciona o *browser* do usuário para o Serviço de SSO do IdP escolhido no passo anterior. No passo 7, o Serviço de SSO é requisitado no IdP e este serviço adquire uma declaração de autenticação (uma asserção SAML) da Autoridade de Autenticação. No passo 8, a asserção SAML é retornada para o *browser* do usuário, após o usuário fornecer as suas credenciais por meio de uma mensagem HTTP/POST. No passo 9, o Serviço Consumidor de Asserções do SP consulta a asserção SAML do IdP. No passo 10, o Serviço Consumidor de Asserções processa a resposta da autenticação (além de outras verificações, a asserção será válida caso o usuário tenha fornecido credenciais válidas no passo 8). Caso o usuário tenha fornecido credenciais válidas, um contexto de segurança é criado no SP, com o posterior redirecionamento do *browser* do usuário para o recurso protegido. No passo 11, o *browser* requisita novamente o acesso ao recurso protegido. No passo 12, com um contexto de segurança válido, o *browser* é redirecionado para a URL do recurso solicitado.

O processo de configuração de federações com Shibboleth não é trivial, uma vez que envolve a adaptação dos serviços do SP, das regras de controle de acesso do domínio, do *IdP Discovery* e da descrição dos atributos necessários para acessar o recurso [Hämmerle 2011]. A configuração de um sistema Shibboleth também envolve a manutenção de um intricado conjunto de arquivos. A federação CAFe [RNP 2011] é um exemplo brasileiro de instituições acadêmicas de ensino e de pesquisa que implementa uma federação com um conjunto de atributos próprios, com o esquema de dados conhecido como brEduPerson. Esses atributos definidos nas asserções SAML estabelecem quais recursos essas entidades poderão requisitar umas das outras, de forma que os provedores de identidades saibam quais atributos devem ser oferecidos.

Projeto Higgins

Higgins é um projeto em desenvolvimento pela Eclipse Foundation [Eclipse Foundation 2011]. Seu objetivo é integrar identidades, perfis e informações relacionadas a redes sociais entre diversos sites, aplicações e dispositivos utilizando componentes extensíveis. Múltiplos protocolos de identidades foram propostos ao longo do tempo, tais como LDAP, WS-Trust, SAML, XDI, OpenID, entre outros. Para os desenvolvedores de sistemas de gerência de identidades federadas, há a necessidade de suportar estes diversos protocolos, o que resulta em complexidade no *software* bem como na gerência de identidades federadas. Para os usuários, este fator pode causar confusão pois estes precisarão gerenciar para qual entidade foi compartilhada qual informação. Por exemplo, o número do cartão de crédito não é interessante de compartilhar em uma rede social, ao passo que em uma loja de vendas *on-line* sim. Para tratar estas questões, o projeto Higgins apresenta um *framework* que provê as seguintes tecnologias [Eclipse Foundation 2011]:

- Um seletor de identidades multiplatforma (Mac OS X, Linux, Windows, etc.) e navegadores (Firefox, Safari, IE, Chrome, etc.) que podem ser utilizados para autenticação em sistemas e sites compatíveis com o modelo centrado no usuário, denominado *Information Card* (i-card). Segundo o projeto, este modelo oferece ao usuário menos senhas, mais conveniência e melhor segurança onde todas as informações do usuário ficam armazenadas em um cartão.

- Provedores de identidades, baseado em serviços Web, que trabalham com os padrões WS-Trust (STS - *Security Token Service*) e SAML 2.0, ambos expedem i-cards. Provê também o código necessário para que provedores de serviço incorporem e aceitem estes i-cards.

- Implementa um modelo de dados chamado na versão 2.0 de *Persona Data Model* (PDM), estendendo o modelo da versão 1.x, denominado *Higgins Data Model* (HDM) 1.0. O Higgins oferece também um serviço de atributos de identidade denominado *Higgins Identity Attribute Service* (IdAS). Com estas soluções, os desenvolvedores possuem uma camada de abstração para obter interoperabilidade e portabilidade entre diferentes silos de informação de identidade, diferentes fontes de dados, tais como diretórios, bancos de dados relacionais e redes sociais.

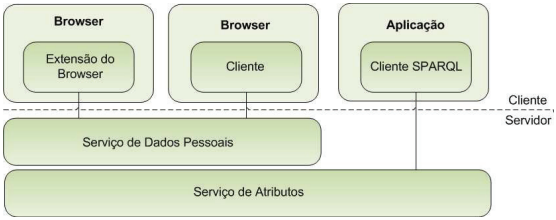

Fig. 5.15. Arquitetura do Higgins. Adaptado de [Eclipse Foundation 2011]

A Fig. 5.15 apresenta a arquitetura em alto nível do Higgins 2.0. Nesta figura, o componente Serviço de Dados Pessoais é responsável por prover serviços para o componente Cliente, localizado no *browser* do usuário. Dentre os serviços prestados estão aqueles para gerência de contas, alteração de *passwords*, etc. O componente Cliente é uma interface em JavaScript desenvolvida para que o usuário possa ver e editar seus cartões. Este JavaScript é carregado do Serviço de Dados Pessoais para o *browser* do usuário via o componente Extensão do Browser. Este componente é responsável por estender as funcionalidades de um *browser* padrão. Ele também provê uma API para que o *script* possa escrever e ler do componente Serviço de Atributos. Isto permite, por exemplo, que o *script* armazene informações vindas de uma página Web. O componente Serviço de Atributos é um repositório do tipo RDF/OWL (*Resource Description Framework*/*Web Ontology Language*) contendo dados do usuário. Estes dados utilizam um vocabulário definido pelo modelo *Persona Data Model* 2.0. O Serviço de Atributos expõe estes dados para o Portal e para o componente Extensão do Browser via uma interface baseada em mensagens HTTP e métodos do tipo *push*. Ele também expõe os dados via uma interface SPARQL (*SPARQL Protocol and RDF Query Language* - acrônimo recursivo) para integração entre servidores [Eclipse Foundation 2011].

OpenAM

O *middleware* OpenAM é uma solução de código aberto responsável por oferecer serviços de autenticação e autorização, federação, SSO, monitoração, *logging* e provisão de identidades. Com a aquisição da empresa Sun Microsystems pela Oracle, o produto originalmente conhecido como OpenSSO passou a ser mantido pelo grupo ForgeRock, porém com o nome de OpenAM. Os serviços de autenticação, autorização, verificação de validade de *tokens*, *logging* e provisão de identidades, oferecidos pelo OpenAM, podem ser acessados tanto via HTTP, quanto via serviços Web, ou por meio de um agente. Este modelo pode ser visto como um IDaaS para controle de acesso e provisão de identidades federadas [ForgeRock 2010]. O OpenAM possui uma arquitetura de *software* cliente/servidor. Desenvolvido em Java e distribuído sob a forma de uma aplicação Web J2EE, ele trabalha com diversos padrões abertos. A Fig. 5.16 ilustra a arquitetura simplificada do OpenAM.

Basicamente, o OpenAM está subdividido em 3 camadas (Interface Cliente, Núcleo e Camada de Integração). Uma descrição mais detalhada da arquitetura pode ser obtida em [Thangasamy 2011]. A primeira camada é responsável por prover interfaces de comunicação para que aplicações possam utilizar os serviços providos pela camada Núcleo. Esta camada pode ser acessada via serviços Web, através de SOAP ou HTTP. A camada Núcleo é onde estão os componentes fundamentais da arquitetura do OpenAM. Esta camada atua como um *broker* entre a Interface Cliente e os componentes servidores [ForgeRock 2010]. Os serviços providos por componentes desta camada são: autenticação, autorização, gerência de sessão (SSO), *logging*, acesso a repositórios de identidades e federação.

Os cinco primeiros serviços são implementados como *servlets* e as requisições e respostas são baseadas em mensagens XML sobre HTTP. O componente de federação

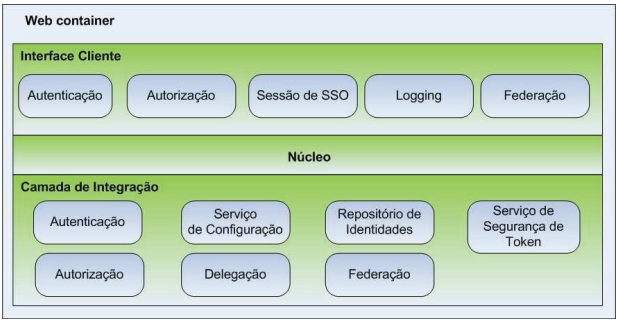

Fig. 5.16. Arquitetura Simplificada do OpenAM.

trabalha com as seguintes especificações: OASIS SAML (SAML1.1 e SAML 2.0), Liberty Alliance (ID-FF 1.2 e ID-WSF 1.1), WS-Security (WS-I BSP), WS-Trust e WS-Federation (WS-Federation 1.1). Cada serviço da camada Núcleo oferece uma SPI (*Service Provider Interface*) para extensão. A Camada de Integração provê as interfaces para extensão destes serviços. Como requisito para sistemas de gerência de identidades federadas, esta camada permite, por exemplo, a extensão para múltiplos mecanismos de autenticação e para múltiplos repositórios de identidades.

O OpenAM suporta também agentes. Estes são parte da arquitetura do OpenAM e são implementados como filtros instalados em servidores de aplicação tais como Apache, Tomcat, WebLogic e Glassfish. Os agentes são utilizados para interceptar requisições a recursos Web e encaminhar estas requisições para o OpenAM. Os agentes também utilizam as interfaces clientes do OpenAM para interagir com o serviço de autenticação, autorização, federação, sessão e *logging*. A Fig. 5.17 ilustra uma implantação básica do OpenAM e do agente protegendo recursos Web.

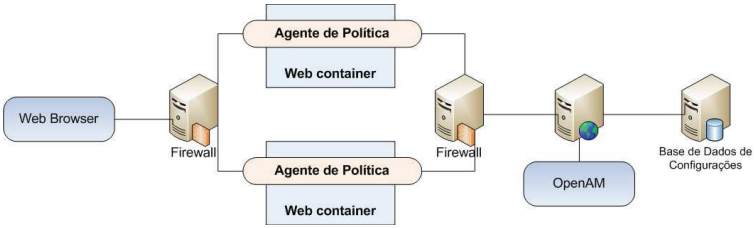

Fig. 5.17. Implantação básica do OpenAM.

O agente é instalado em servidores de aplicação onde os recursos a serem protegidos estão executando. Em geral estes servidores estão localizados atrás de um *firewall* externo, ou seja, em uma zona desmilitarizada. O OpenAM executa em outro servidor de aplicação, juntamente com um servidor de banco de dados, ambos atrás de um *firewall* interno. Uma requisição a uma página, por exemplo, feita por um usuário,

através de um *browser*, será interceptada pelo agente instalado no servidor onde a página é servida e fará com que o agente verifique se o usuário possui credenciais para acesso a este recurso. Em caso negativo, o *browser* do usuário será redirecionado para o OpenAM. O OpenAM fará o processo de autenticação deste usuário e irá redirecionar o mesmo de volta para a página, caso a autenticação seja bem sucedida.

5.5.4. Tabela Comparativa

A Tab. 5.1 apresenta as soluções de gerência de identidades federadas e os padrões estudados neste minicurso.

Soluções	SAML 2.0	OpenID 2.0	XACML 2.0	OAuth 1.0	Paradigma
Shibboleth 2.x	√	-	Extensão	-	Federado
Higgins 2.0	√	-	-	-	C.Usuário
OpenAM 9.5.x	√	*Plugin*	√	*Plugin*	Federado

Tab. 5.1. Tabela Comparativa entre Soluções Abertas em IdM.

Como podemos observar na Tab. 5.1, o padrão para autenticação em gerência de identidades federadas mais utilizado entre as soluções abertas estudadas é o SAML. O OpenID, por sua vez, não está presente no Shibboleth e nem no Higgins, mas pode ser integrado ao OpenAM mediante a instalação de um *plugin*. Com relação aos padrões para autorização, o XACML é contemplado apenas pelo OpenAM. O Shibboleth requer uma extensão baseada em uma solução de repositório do Fedora. O OAuth não é suportado nativamente por nenhuma das soluções. Apenas o OpenAM oferece a possibilidade de instalação de um *plugin*.

Nenhuma das soluções apresentadas oferece suporte nativo a todos os padrões de gerência de identidades federadas estudados. Para ambientes de computação nuvem esta característica é importante, dado que o número de aplicações oferecidas como serviço são variadas. Apesar disso, o OpenAM é uma solução flexível que permite a possibilidade de extensão a partir da criação e instalação de *plugins*. Para um provedor IDaaS esta característica é interessante.

A utilização do OpenID e OAuth são mais adequados para ambientes de redes sociais (*blogs, mash-ups*), já que se propõem a serem soluções sem muita preocupação com a troca confiável de mensagens. O padrão SAML possui uma preocupação maior com a segurança, já que utiliza certificados digitais na troca de mensagens. Uma comparação mais detalhada entre o SAML e o OpenID pode ser encontrada em [Hodges 2009]. O XACML é um padrão que pode ser utilizado em conjunto com o SAML, podendo assim oferecer uma solução de autenticação e autorização em ambientes organizacionais.

O Higgins é a única solução estudada que adota o paradigma centrado no usuário, porém ainda é um projeto que está em evolução. Dentre as vantagens já citadas este paradigma possibilita maior autonomia ao usuário no compartilhamento de informações de sua identidade. O OpenAM, por ter uma arquitetura flexível, pode operar segundo o paradigma centrado no usuário, desde que seja instalado um *plugin* que implemente um padrão neste paradigma.

Observamos que as soluções estudadas, apesar de não suportarem a maioria dos novos padrões e paradigmas de gerência de identidades federadas, estão no processo de evolução para suportá-los. Isto é importante para provedores de nuvem, especialmente provedores IDaaS, que necessitam de uma infraestrutura flexível para atender à demanda por serviços federados.

5.6. Estudo de Caso: Plataforma REALcloud

Uma das motivações para o desenvolvimento de uma arquitetura para computação em nuvem é oferecer a plataforma REALabs como serviço (PaaS). Esta plataforma foi desenvolvida pelos autores para a interação remota com recursos robóticos [Guimarães et al. 2011]. REALabs oferece uma infraestrutura de *software* para o acesso remoto a um laboratório de robótica móvel. Nesta plataforma os experimentos robóticos são executados no computador do usuário e operam os robôs por meio de uma rede de comunicação (usualmente a Internet). Esta forma de operação, apesar de funcional, traz algumas limitações. A principal delas é com relação ao atraso imposto pela rede. Sem uma rede de alta velocidade o controle dos robôs fica prejudicado devido ao atraso na comunicação com o robô. Além disso, os usuários necessitam de CPU e memória consideráveis para execução dos experimentos. Um requisito também necessário é qualidade de serviço que não é garantida na Internet [Cardozo et al. 2010].

Outro ponto a ser considerado é a questão da segurança. Os recursos robóticos são caros e por isso necessitam de um controle de acesso aprimorado para permitir que apenas os usuários autenticados e autorizados possam acessar tais recursos. O acesso a estes recursos necessita de um modelo baseado em políticas compatível com o modelo de utilização dos recursos estabelecido pelo administrador do laboratório. Com a difusão de laboratórios de acesso remoto na Internet (WebLabs) surge a oportunidade de operar estes WebLabs de forma federada. Este conceito reforçou a necessidade de um controle de acesso aprimorado, que precisa contemplar acessos de usuários vindos de domínios parceiros.

Com o advento da computação em nuvem e os conceitos agregados de virtualização, computação orientada a serviços, computação sob demanda, modelo de acesso ubíquo, dentre outros, observou-se que a aplicação destes conceitos e tecnologias no contexto de WebLabs trazem ganhos significativos. A execução de experimentos em nuvem, tendo a máquina virtual do usuário na rede física dos recursos, permite um modelo mais adequado de experimentação. Neste modelo observa-se um ganho de desempenho na interação dos experimentos robóticos com os recursos, dado que tais experimentos executam na máquina virtual do usuário localizada no mesmo enlace de rede dos recursos.

Do ponto de vista da IdM, o desafio é oferecer uma infraestrutura que seja aderente à realidade colaborativa. O oferecimento da identidade como um serviço unificado de nuvem (IDaaS) é essencial para que as aplicações SaaS, PaaS e IaaS possam utilizar seus serviços sem necessitar recorrer a mecanismos individualizados de controle de acesso, fato este que inviabilizaria a gerência à medida que o número de serviços e colaborações aumentam. Outra questão importante é que este serviço permite realizar SSO entre os diferentes domínios federados. Esta infraestrutura também permite a integração de contas

de usuários provindas de diferentes domínios. Outra consideração é que o serviço IDaaS permite utilização de diferentes padrões de federação.

5.6.1. Visão Geral da Plataforma REALcloud

Para tratar os problemas relacionados no tópico anterior, desenvolvemos a plataforma REALcloud [Feliciano et al. 2011] [Agostinho et al. 2011]. Nesta plataforma, há uma infraestrutura de nuvem privada para os estudantes utilizarem os servidores conectados aos robôs através de uma rede local para realizarem os seus experimentos. Como o WebLab é um ambiente colaborativo, são utilizados os conceitos definidos pela gerência de identidades federadas para gerenciar o uso dos recursos distribuídos entre os diferentes domínios parceiros e seus experimentadores.

A Fig. 5.18 apresenta uma visão em alto nível da plataforma REALcloud. A figura apresenta um conjunto de nuvens privadas e públicas conectadas através da Internet e/ou uma rede privada de alta velocidade. A nuvem privada oferece recursos tais como CPU, unidades de armazenamento e conexões de rede para as aplicações que executam os experimentos. Esta nuvem oferece também recursos de *hardware* especializados tais como GPUs (*Graphics Processing Units*) e FPGAs (*Field-Programmable Gate Arrays*) e também plataformas de *software* específicas, tais como Matlab.

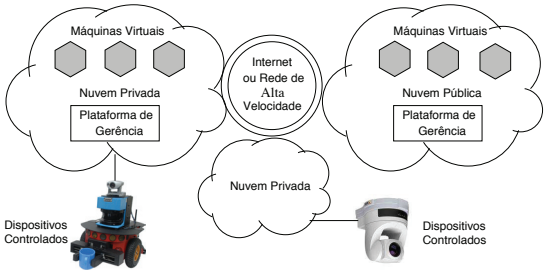

Fig. 5.18. Arquitetura de Alto Nível da Plataforma REALcloud.

A nuvem pública oferece recursos computacionais para aplicações e componentes que não necessitam de um tratamento de qualidade de serviço e segurança aprimorados. Alguns destes recursos podem ser oferecidos como serviços, por exemplo, serviços de gerência de WebLabs. A nuvem pública pode atuar como um provedor de identidades, permitindo que os usuários possam se autenticar, por exemplo, com uma conta do Yahoo! e com isso utilizar recursos habilitados para este tipo de usuário. A nuvem pública pode também servir para ampliar uma facilidade que alguma nuvem privada porventura necessite.

5.6.2. Arquitetura da Plataforma REALcloud

A plataforma REALcloud possui as seguintes características:

- Gerência de identidades federadas, para permitir o compartilhamento seguro de recursos e SSO entre os domínios federados.

- Controle de acesso para prevenir acessos não autorizados e manipulação acidental dos dispositivos físicos controlados.
- Plataforma para gerência de máquinas virtuais.
- Controle de QoS de rede, para que as aplicações tenham baixo atraso e largura banda adequadas.

Cada requisito acima é atendido por um serviço especializado. O Serviço de Gerência de Identidades Federadas é responsável pelo estabelecimento de federações entre as nuvens públicas e privadas e proteção para os serviços restantes. Este serviço provê também SSO e oferece uma API baseada em REST para utilização da identidade como um serviço (IDaaS).

O Serviço de Gerência de Acesso aos Dispositivos permite ao administrador do domínio registrar os dispositivos físicos disponíveis e outros recursos, bem como estabelecer políticas para sua manipulação (por exemplo, uso mediante reserva). Este serviço permite também o estabelecimento de sessões seguras de acesso aos dispositivos controlados.

Virtualização é uma tecnologia chave para a computação em nuvem. Máquinas virtuais oferecem isolamento e recursos computacionais de acordo com a demanda do usuário. No nosso caso, a virtualização também oferece a possibilidade das aplicações estarem próximas dos dispositivos controlados, reduzindo o atraso inerente que degrada a operação dos dispositivos. O Serviço de Gerência de VMs permite aos administradores atribuir VMs aos usuários ou grupo de usuários, bem como permite aos usuários acessar (mediante credenciais) e controlar suas VMs.

Para proteger os dispositivos controlados contra acessos não autorizados, o Serviço de Gerência de Acesso aos Dispositivos utiliza-se de mecanismos de *firewall* providos pelo Serviço de Controle de Rede. Este serviço realiza um controle de acesso no nível de encaminhamento de pacotes que bloqueia o acesso não autorizado e estabelece privilégios no nível de rede para as aplicações que estabeleceram uma sessão de acesso. A Fig. 5.19 ilustra os serviços de gerência definidos para a plataforma REALcloud (Plataforma de Gerência na Fig. 5.18). Cada pacote representa um serviço e as linhas pontilhadas relacionam as dependências entre estes serviços.

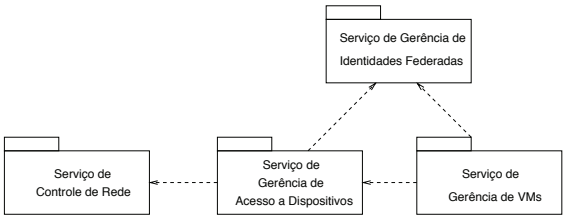

Fig. 5.19. Serviços de Gerência da Plataforma REALcloud e suas Dependências.

5.6.3. Serviços da Plataforma REALcloud

A plataforma REALcloud especializa os serviços identificados na arquitetura proposta (Fig. 5.19).

Gerência de Identidades Federadas

O serviço de Gerência de Identidades Federadas na plataforma REALcloud adota o OpenAM. Este *middleware* foi configurado para funcionar com os perfis *Web SSO* e *IdP Discovery*, definidos pela especificação do SAMLv2. Na plataforma é empregado um modelo para a gerência de identidades federadas onde os componentes SP, IdP e IdP *proxy* estão dispostos em um modelo em camadas. A Fig. 5.20 mostra este modelo acrescido das funcionalidades dos componentes PIP, PAP, PEP e PDP, conforme empregados no padrão XACML.

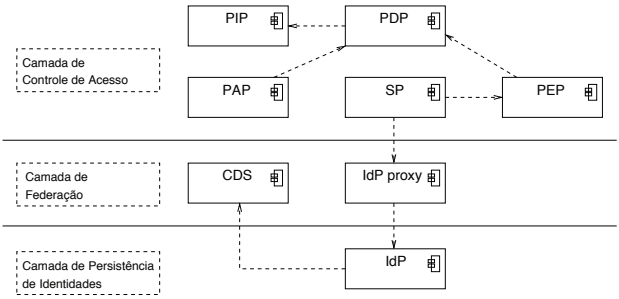

Fig. 5.20. Modelo em Camadas dos Componentes para IdM [Feliciano et al. 2011].

Quando um usuário necessita acessar algum recurso da plataforma, o SP, com o auxílio do componente PEP, intercepta a requisição e verifica se a requisição possui as credenciais necessárias. Se nenhuma credencial for encontrada, o SP redireciona o usuário para o IdP *proxy*, que por sua vez apresenta uma listagem contendo os possíveis IdPs. O usuário seleciona um IdP relacionado ao domínio, onde o mesmo possui uma identidade e é redirecionado para autenticação. Uma vez autenticado, o IdP expede uma credencial (uma asserção SAML) e redireciona o usuário para o componente CDS (*Common Domain Service*), que adiciona um *cookie* de domínio comum (*cookie* de federação). Na sequência, o componente CDS redireciona o *browser* para o IdP *proxy*. Este, por sua vez, apenas verifica qual foi o SP requisitante e, com isso, redireciona o usuário para este SP. O SP novamente verifica a existência de uma credencial de acesso e, desta vez, a presença da credencial permitirá o acesso ao recurso originalmente solicitado. Como a autenticação foi realizada com SSO, o usuário também está apto a acessar recursos existentes nos domínios pertencentes ao círculo de confiança, caso as políticas de autorização permitam tal operação.

Gerência de Acesso a Dispositivos

A Gerência de Acesso a Dispositivos realiza três funções, implementadas como serviço:

1. Registro de Recursos: este serviço permite a criação de uma lista de recursos a serem protegidos. Esta lista armazena, para cada recurso, as suas propriedades (nome, dono do recurso, modelo, etc.), largura de banda requerida para a realização do experimento e a URI (*Uniform Resource Identifier*) que identifica o recurso. Para a realização de um experimento, os recursos podem ser agrupados. Por exemplo, um braço robótico e uma câmera podem ser agrupados para que a câmera mostre os movimentos do braço. Este agrupamento é feito mediante reserva, provida por este serviço. O serviço permite que o administrador separe fatias de tempo para a utilização de cada recurso ou grupo de recursos e a duração máxima da reserva no dia.

2. Reserva: é um serviço que permite aos usuários agendar reservas futuras para a utilização dos recursos, respeitando as fatias de tempo disponíveis e a duração máxima da reserva estipulada pelo administrador.

3. Controle de Sessão: é usado para estabelecer a sessão de acesso aos dispositivos, usualmente durante o período da reserva. As sessões de acesso mantêm o estado da utilização dos recursos, por exemplo, a identidade do usuário, período de acesso e os recursos permitidos.

Gerência de VMs

O serviço de Gerência de VMs na plataforma REALcloud permite que os usuários iniciem e desliguem suas respectivas VMs mantidas nas nuvens públicas e privadas. Os usuários podem acessar suas VMs a qualquer horário, sem necessidade de reserva de horário. Porém, este acesso não habilita o usuário acessar os recursos protegidos do laboratório. O acesso aos recursos protegidos só é liberado se o usuário possuir uma sessão de acesso obtida do serviço de Gerência de Acesso a Dispositivos.

Assim que o usuário estabelece uma sessão, o serviço de Gerência de Acesso a Dispositivos interage com o serviço de Controle de Rede para habilitar o acesso da VM que estabeleceu a sessão para o dispositivo físico e armazenar as regras de encaminhamento de tráfego de acordo com o dispositivo acessado. Esses privilégios de acesso são descartados quando a sessão de acesso terminar. Uma sessão pode terminar explicitamente pelo usuário ou pelo serviço de Gerência de Acesso a Dispositivos, quando o período da reserva terminar.

Virtualizador, Controle de Rede e Encaminhamento de Pacotes

Virtualizador é um serviço para estabelecer a interação com diferentes soluções de virtualização, tais como libvirtd, KVM e VirtualBox. Utiliza-se estes pacotes para abstrair

a solução de virtualização, por exemplo, alterações na tecnologia de virtualização tem impacto reduzido em toda a arquitetura.

O serviço de Controle de Rede é necessário para interagir com o mecanismo de encaminhamento de pacotes, empregar políticas de *firewall* e para realizar diferenciação de tráfego.

O serviço de Encaminhamento de Pacotes filtra o tráfego direcionado aos dispositivos físicos de/para a VM que possui uma sessão de acesso válida. A priorização de tráfego permite estipular classes de serviços (prioridades) de acordo com o dispositivo físico sendo acessado. Por exemplo, imagens em tempo real utilizadas para controlar o robô móvel devem ter maior prioridade do que imagens geradas por uma câmera panorâmica, utilizada para visualização dos experimentos. A classe de serviço e a banda requerida são especificadas através do serviço de Registro de Recursos. A atual versão da plataforma REALcloud não suporta SLAs.

A Fig. 5.21 apresenta os serviços oferecidos pela plataforma REAcloud e a sua inter-relação em notação UML. É importante ressaltar que as linhas tracejadas modelam os fluxos da interação.

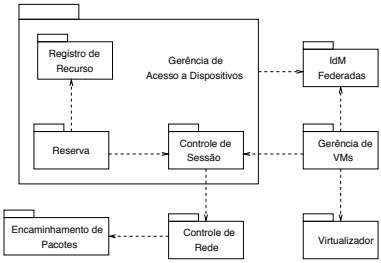

Fig. 5.21. Serviços de Gerência da Plataforma REALcloud.

5.6.4. Implementação da Plataforma REALcloud

O serviço de Gerência de VMs é um *servlet* Java que expõe uma interface Web para controlar o ciclo de vida das VMs, utilizando os serviços do Virtualizador. Os demais serviços oferecidos pela plataforma REALcloud estão descritos na sequência.

Serviço de Gerência de Identidades Federadas

A implementação deste serviço utiliza como base o OpenAM. Na plataforma REALcloud cada componente do OpenAM (SP, IdP proxy e IdP) está instalado em uma VM. A utilização de componentes deste serviço em VMs oferece à plataforma o aumento no tempo de disponibilidade do serviço e também uma diminuição do tempo de instalação deste serviço em outras nuvens.

O estabelecimento do CoT da federação se dá através da configuração de um perfil (SP, IdP ou IdP proxy) em cada instância do OpenAM. Para tal, foi necessário a criação

de um metadado no padrão SAMLv2. Além de conter informações sobre o perfil, este metadado possui um certificado X.509 gerado em cada entidade. A Fig. 5.22(a) ilustra a troca de metadados realizada entre os componentes de IdM da nuvem e entre as nuvens privadas (b), respectivamente. No passo 1, gera-se os metadados do IdP *proxy* com o perfil SP e IdP. No passo 2, o SP importa a parte IdP do metadado do IdP *proxy*. No passo 3, o IdP importa a parte SP do metadado do IdP *proxy*. No passo 4, o IdP *proxy* importa o metadado do SP. Finalmente, no passo 5 o IdP *proxy* importa o metadado do IdP. O IdP proxy atuará, sob o ponto de vista do SP, como um IdP e do ponto de vista de um IdP, como um SP. Isto foi importante para permitir a agregação de múltiplos IdPs (de domínios diferentes) para fazer parte da federação. Na Fig. 5.22(b), a troca de metadados ocorre entre duas nuvens privadas. No passo 1, no IdP *proxy* do domínio 1 é importado o metadado do IdP do domínio 2. No passo 2, é importado no IdP do domínio 2, a parte SP do IdP *proxy* do domínio 1. O mesmo processo (passos 3 e 4) é realizado entre os componentes IdP e IdP *proxy* dos domínios 1 e 2, respectivamente.

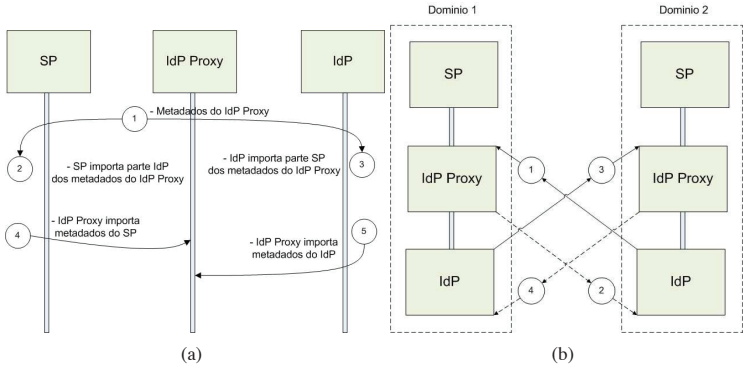

Fig. 5.22. Troca de Metadados: (a) Dentro da Nuvem Privada, (b) entre Nuvens Privadas.

Uma parte importante da infraestrutura de gerência de identidades federadas é o agente. O agente atua como um PEP e a filtragem ocorre na requisição de um recurso hospedado. Na nuvem privada, foi instalado um agente no servidor Apache para proteger páginas de gerência do laboratório (reserva de horário, cadastro de usuários, cadastro de experimentos, provisão de experimentos, etc.). Neste servidor Apache também está instalado o módulo *mod_proxy*. Com isso a plataforma utiliza um modelo de implantação de SSO misto, que combina o uso de *proxy* reverso e um agente instalado neste servidor para filtrar as requisições a todos os recursos protegidos da plataforma.

Serviço de Gerência de Acesso a Dispositivos

O serviço de Gerência de Acesso a Dispositivos providos pela plataforma REALcloud foi implementado com a tecnologia J2EE (*Java 2 Enterprise Edition*), como provido pelo servidor de aplicação Tomcat. Para a persistência dos dados foi utilizado o servidor de banco de dados MySQL.

Serviço de Controle de Rede

O serviço de Controle de Rede foi implementado como um *servlet* que executa um *shell script* para interagir com as regras de *firewall* padrão do Linux, oferecido pelo serviço de Encaminhamento de Pacotes. O serviço de Controle de Rede não provê serviços no nível de usuário, tendo como cliente o serviço de Gerência de Acesso a Dispositivos. Os serviços da plataforma estão distribuídos de acordo com a Fig. 5.23.

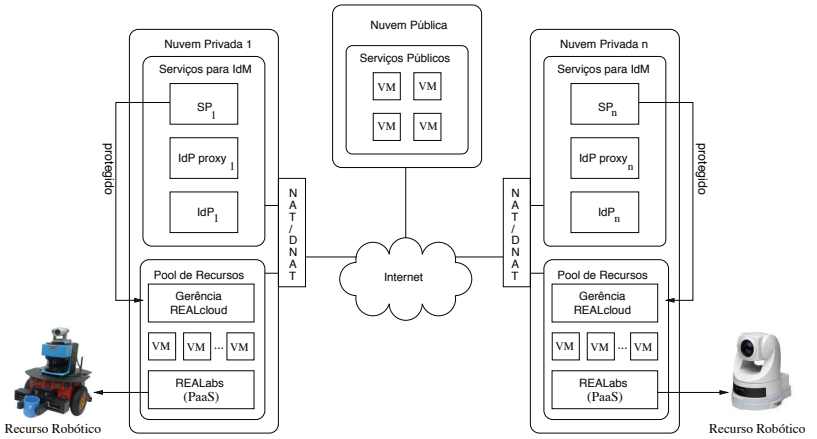

Fig. 5.23. Plataforma REALcloud.

As nuvens privadas 1 e N na Fig. 5.23 representam dois domínios privados que possuem recursos a serem compartilhados. O serviço de Gerência de VM, o serviço de Gerência de Acesso a Dispositivos e o serviço de Controle de Rede estão distribuídos como componentes da plataforma (Gerência REALcloud). Os componentes de gerência da plataforma REALcloud e os recursos tais como robôs, câmeras panorâmicas, máquinas virtuais, a plataforma REALabs, entre outros, localizados no *pool* de recursos, são protegidos pelo componente SP do serviço de gerência de identidades federadas.

A nuvem pública oferece apenas serviços públicos, tais como: hospedagem de páginas, armazenagem de dados e aplicações específicas. Cada serviço está inserido em uma VM ou em VMs que agregam serviços co-relacionados. A nuvem pública pode estabelecer federações entre as nuvens privadas ou entre um provedor de nuvem pública (por exemplo, Amazon). Nesse caso, o serviço de gerência de identidades federadas deve ser instalado na nuvem pública.

Finalmente, cada nuvem está interconectada à Internet. Para se traduzir endereços IP privados para públicos e vice-versa, foi utilizado a função de rede NAT/DNAT. Assim cada requisição originada através da Internet para o par *IP:porta* pública de cada nuvem privada é mapeado para o par *IP:porta* privada, onde os recursos existentes no *pool* estão conectados.

5.7. Considerações Finais

A gerência de identidades acompanha as novas necessidades colaborativas de uma Internet que está em constante evolução. A integração de aplicações na Web com ambientes federados tem o objetivo de suprir as expectativas de uma Web de serviços independentes, porém interoperáveis.

Apesar das vantagens quanto à elasticidade na oferta de serviços em nuvem, existem alguns desafios que ainda precisam ser superados. Dentre estes estão as questões legais quanto à garantia de restrições sobre o tipo de conteúdo armazenado, além de garantias quanto ao nível de acesso do provedor de serviços aos dados hospedados em seus *datacenters*. Por outro lado, o acesso aos recursos do domínio deve ser gerenciado, para que apenas os usuários com contas no domínio, ou em domínios parceiros, tenham acesso a esses recursos. Além disso, o provedor de serviços de nuvem deve comportar o aumento do número de usuários, sem que isso reduza o desempenho das conexões individuais. Também, aplicativos empresariais que envolvem sistemas de intranet, ambientes colaborativos, redes sociais, aplicativos para celular, dentre outros, também podem se beneficiar do uso da nuvem como porta de acesso para uma ampla gama de serviços federados, ou seja, não exclusivos de uma única nuvem. Nesse sentido, a gerência de identidades federadas contribui para assegurar mecanismos confiáveis para a provisão de gerência de múltiplas identidades.

Em nossa experiência na utilização de soluções para implantação de IdM em nuvens verificamos que estas não estão totalmente adaptadas para tratar os desafios da computação em nuvem. A configuração desses sistemas requer um conhecimento prévio dos padrões e, principalmente, da teoria que lhes dá suporte, visto que a terminologia utilizada nestes sistemas é baseada nos conceitos destes padrões. O Shibboleth por ser o sistema precursor, carece de suporte aos padrões emergentes, tais como OpenID e OAuth. A instalação deste sistema é dispendiosa, visto que parte de seus componentes estão em Java e em C/C++. Outra questão é que a instalação é baseada em arquivos XML, o que dificulta o entendimento para um iniciante na área. Já o OpenAM oferece todo o seu pacote de *software* encapsulado de um único arquivo (.war), o que facilita a instalação. A configuração, por sua vez, também demanda um conhecimento prévio dos padrões, principalmente na configuração de federações, que requer um considerável esforço. No caso do Higgins, por adotar o paradigma centrado no usuário e ser um projeto ainda em desenvolvimento, existe pouca informação quanto a sua utilização.

A plataforma REALcloud é um exemplo de solução de código aberto para a realização de experimentos em rede. Como características desta solução, citamos a capacidade de agregar nuvens privadas distintas com compartilhamento de recursos utilizando relações de confiança da federação e também a capacidade de integrar nuvens públicas com a finalidade de aumentar a oferta de recursos computacionais para as demais nuvens privadas.

Referências

[Agostinho et al. 2011] Agostinho, L., Olivi, L., Feliciano, G., Paolieri, F., Rodrigues, D., Guimarães, E., and Cardozo, E. (2011). A Cloud Computing Environment for Supporting Networked Robotics Applications. *The 3rd International Workshop on*

Workflow Management in Service and Cloud Computing.

[Amazon 2010] Amazon (2010). Amazon Elastic Compute Cloud (Amazon EC2). Disponível em: http://aws.amazon.com. Acessado em 10 de Setembro de 2011.

[Anderson 2005] Anderson, A. (2005). A Comparison of Two Privacy Policy Languages: EPAL and XACML. Technical report. Disponível em: http://labs.oracle.com/techrep/2005/smli_tr-2005-147/TRCompareEPALandXACML.html.

[Bertino and Takahashi 2011] Bertino, E. and Takahashi, K. (2011). *Identity Management: Concepts, Technologies, and Systems*. Artech House.

[Breitman and Viterbo 2010] Breitman, K. and Viterbo, H. (2010). Computação na Nuvem - Uma Visão Geral. *Congresso Internacional Software Livre e Governo Eletrônico - Consegi*, pages 17–45.

[Buyya et al. 2010] Buyya, R., Ranjan, R., and Calheiros, R. N. (2010). InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *Proceedings of the 10th ICA3PP*, pages 21–23. Springer.

[Cao and Yang 2010] Cao, Y. and Yang, L. (2010). A Survey of Identity Management Technology. *IEEE International Conference on Information Theory and Information Security (ICITIS)*.

[Cardozo et al. 2010] Cardozo, E., Guimarães, E. G., Rocha, L. A., Souza, R. S., Paolieri, F., and Pinho, F. (2010). A platform for networked robotics. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010), Taipei, Taiwan*.

[Carissimi 2008] Carissimi, A. (2008). Virtualização: da teoria a soluções. In *Mini-curso - SBRC 2008 - Rio de Janeiro - RJ*.

[Cloud Security Alliance 2009] Cloud Security Alliance (2009). Security Guidance for Critical Areas of Focus in Cloud Computing V2.1. Technical report.

[Crupi and Warner 2008] Crupi, J. and Warner, C. (2008). Enterprise Mashups Part I: Bringing SOA to the People . *SOA Magazine*, (18).

[Dike 2006] Dike, J. (2006). *User-Mode Linux*. Prentice Hall.

[Eclipse Foundation 2011] Eclipse Foundation (2011). Higgins - Open Source Identity Framework. Disponível em: http://www.eclipse.org/higgins. Acessado em 20 de Agosto de 2011.

[El Maliki and Seigneur 2007] El Maliki, T. and Seigneur, J.-M. (2007). A Survey of User-centric Identity Management Technologies. In *Proceedings of the The International Conference on Emerging Security Information, Systems, and Technologies*, Washington, DC, USA.

[Endo et al. 2010] Endo, P. T., Gonçalves, G. E., Kelner, J., and Sadok, D. (2010). A survey on open-source cloud computing solutions. *XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - VIII Workshop em Clouds, Grids e Aplicações - Gramado - RS*.

[Feliciano et al. 2011] Feliciano, G., Agostinho, L., Guimarães, E., and Cardozo, E. (2011). Uma Arquitetura para Gerência de Identidades em Nuvens Híbridas. *IX Workshop em Clouds, Grids e Aplicações (WCGA 2011) - XXIX Simpósio Brasileiro de Redes de Computadores (SBRC - 2011) - Campo Grande - MS*.

[FlexiScale 2010] FlexiScale (2010). FlexiScale Public Cloud. Disponível em: http://www.flexiant.com/products/flexiscale. Acessado em 2 de Agosto de 2011.

[ForgeRock 2010] ForgeRock (2010). OpenAM. Disponível em: http://www.forgerock.com/openam.html. Acessado em 9 de Agosto de 2011.

[Gomes and Kowalczyk 2011] Gomes, E. R.and Vo, Q. B. and Kowalczyk, R. (2011). Pure exchange markets for resource sharing in federated clouds. In *Proceedings of Concurrency and Computation: Practice and Experience*.

[Gonçalves et al. 2011] Gonçalves, G. E., Endo, P. T., Cordeiro, T. D., and et al. (2011). Resource Allocation in Clouds: Concepts, Tools and Research Challenges. *XXIX SBRC - Gramado - RS*.

[Google 2010] Google (2010). Google Apps. Disponível em: http://www.google.com/apps. Acessado em 2 de Setembro de 2011.

[Gopalakrishnan 2009] Gopalakrishnan, A. (2009). Cloud Computing Identity Management. *SETLabs Briefings*, 7(7).

[Guimarães et al. 2011] Guimarães, E. G., Cardozo, E., Moraes, D. H., and Coelho, P. R. (2011). Design and Implementation Issues for Modern Remote Laboratories. *IEEE Transactions on Learning Technologies*, 4(1).

[Hodges 2009] Hodges, J. (2009). Technical Comparison: OpenID and SAML - Draft 07a. Technical report. Disponível em: http://identitymeme.org/doc/draft-hodges-saml-openid-compare.html. Acessado em 2 de Agosto de 2011.

[Hämmerle 2011] Hämmerle, L. (2011). Enabling Interfederation Support for a Shibboleth Service Provider (SP) in SWITCHaai. Disponível em: https://www.switch.ch/aai/docs/interfederation/sp-deployment.html. Acessado em 15 de Setembro de 2011.

[IBM 2011] IBM (2011). New to SOA and Web Services. Disponível em: http://www-128.ibm.com/developerworks/webservices/newto/websvc.html. Acessado em 4 de Agosto de 2011.

[IETF 2000] IETF (2000). Framework for Policy-based Admission Control IETF RFC 2753. Technical report. Disponível em: http://www.ietf.org/rfc/rfc2753.txt. Acessado em 25 de Agosto de 2011.

[Internet2 2011] Internet2 (2011). Shibboleth - A Project of the Internet2 Middleware Initiative. Disponível em: http://shibboleth.internet2.edu. Acessado em 15 de Setembro de 2011.

[ISO/IEC 1996] ISO/IEC (1996). Information technology – Open Systems Interconnection – Security frameworks for open systems: Access control framework ISO/IEC 10181-3:1966. Technical report.

[ITU-T 2009] ITU-T (2009). NGN Identity Management Framework. Recommendation Y.2720. Technical report. Disponível em: http://www.itu.int/itu-t/recommendations/rec.aspx?id=9574. Acessado em 20 de Agosto de 2011.

[Johansson 2009] Johansson, L. (2009). It's the F-Word. *IETF Journal*, 6(1).

[Júnior et al. 2010] Júnior, A. M., Laureano, M., Santin, A., and Maziero, C. (2010). Aspectos de segurança e privacidade em ambientes de Computação em Nuvem. In *Mini-curso - SBSeg 2010 - Fortaleza - CE*.

[Ma 2005] Ma, K. J. (2005). Web Services: What's Real and What's Not? *IT Professional*, 7(2):14–21.

[Mell and Grace 2010] Mell, P. and Grace, T. (2010). NIST Working Definition of Cloud Computing. Technical report. Disponível em: http://groups.google.com/group/cloudforum/web/nist-working-definition-of-cloud-computing. Acessado em 15 de Setembro de 2011.

[Menascé 2005] Menascé, D. A. (2005). Virtualization: Concepts, applications, and performance modeling.

[Microsoft 2011] Microsoft (2011). CardSpace. Disponível em: http://msdn.microsoft.com/en-us/library/aa480189.aspx. Acessado em 2 de Setembro de 2011.

[Murari 2010] Murari, K. e. a. (2010). *Eucalyptus Beginner's Guide - UEC Edition - Ubuntu 10.04 - Lucid*. CSS Corp.

[Nimbus 2011] Nimbus (2011). Nimbus - University of Chicago. Disponível em: http://www.nimbusproject.org/. Acessado em 2 de Setembro de 2011.

[Ning 2010] Ning (2010). Ning Inc. Disponível em: www.ning.com. Acessado em 23 de Agosto de 2011.

[OASIS 2005] OASIS (2005). eXtensible Access Control Markup Language (XACML) Version 2.0. Technical report. Disponível em: http://docs.oasis-open.org/xacml/2.0. Acessado em 9 de Setembro de 2011.

[OASIS 2006] OASIS (2006). UDDI 101. Disponível em: http://uddi.xml.org/uddi-101. Acessado em 5 de Setembro de 2011.

[OASIS 2008] OASIS (2008). Security Assertion Markup Language (SAML) V2.0 Technical Overview. Technical report. Disponível em: http://docs.oasis-open.org/security/saml/v2.0. Acessado em 17 de Agosto de 2011.

[OASIS 2011] OASIS (2011). SPML. Disponível em: http://www.oasis-open.org/committees/provision. Acessado em 9 de Setembro de 2011.

[OAuth Community 2010] OAuth Community (2010). The OAuth 1.0 Protocol. Technical report. Disponível em: http://tools.ietf.org/html/rfc5849. Acessado em 18 de Agosto de 2011.

[OAuth Community 2011] OAuth Community (2011). The OAuth 1.0 Guide. Disponível em: http://hueniverse.com/oauth/guide/intro/. Acessado em 2 de Agosto de 2011.

[Olden 2011] Olden, E. (2011). Architecting a Cloud-Scale Identity Fabric. volume 44, pages 52–59. Journal Computer - IEEE Computer Society.

[Open Grid Forum 2009] Open Grid Forum (2009). Occi - open cloud computing interfaces. Disponível em: http://occi-wg.org/. Acessado em 10 de Agosto de 2011.

[OpenID Foundation 2011] OpenID Foundation (2011). OpenID. Disponível em: http://openid.net/get-an-openid/what-is-openid/. Acessado em 2 de Setembro de 2011.

[OpenNebula 2010] OpenNebula (2010). OpenNebula. Disponível em: http://opennebula.org. Acessado em 10 de Setembro de 2011.

[OpenVZ 2010] OpenVZ (2010). OpenVZ Wiki. Disponível em: http://wiki.openvz.org. Acessado em 10 de Agosto de 2011.

[Oracle 2010] Oracle (2010). Virtualbox. Disponível em: http://www.virtualbox.org. Acessado em 10 de Agosto de 2011.

[Ping Identity 2010a] Ping Identity (2010a). About Identity Federation and SSO. Disponível em: http://pingidentity.com. Acessado em 20 de Setembro de 2011.

[Ping Identity 2010b] Ping Identity (2010b). OpenID Tutorial. Disponível em: https://www.pingidentity.com/resource-center/openid.cfm. Acessado em 20 de Setembro de 2011.

[Rackspace 2010] Rackspace (2010). Openstack cloud software. Disponível em: www.openstack.org. Acessado em 10 de Setembro de 2011.

[RNP 2011] RNP (2011). CAFe - Comunidade Acadêmica Federada. Disponível em: http://www.rnp.br/servicos/cafe.html. Acessado em 15 de Setembro de 2011.

[Salesforce 2010] Salesforce (2010). Disponível em: http://salesforce.com. Acessado em 2 de Setembro de 2011.

[Scavo and Cantor 2005] Scavo, T. and Cantor, S. (2005). Technical report. Disponível em: http://shibboleth.internet2.edu/docs/draft-mace-shibboleth-tech-overview-latest.pdf. Acessado em 13 de Agosto de 2011.

[Shelly and Frydenberg 2010] Shelly, G. B. and Frydenberg, M. (2010). *Web 2.0: Concepts and Applications*. Course Technology.

[Stallings 2011] Stallings, W. (2011). *Cryptography and Network Security: Principles and Practice*. Pearson Education.

[Stoller 2011] Stoller, S. D. (2011). XACML. Disponível em: http://www.cs.sunysb.edu/ stoller/cse608/6-XACML.pdf. Acessado em 14 de Agosto de 2011.

[Suess and Morooney 2009] Suess, J. and Morooney, K. (2009). Identity Management and Trust Services: Foundations for Cloud Computing. Technical report. Disponível em: http://www.educause.edu/node/178404. Acessado em 10 de Setembro de 2011.

[Switch 2011] Switch (2011). SWITCH - Serving Swiss Universities. Disponível em: http://www.switch.ch. Acessado em 2 de Setembro de 2011.

[Thangasamy 2011] Thangasamy, I. (2011). *OpenAM*. Packt Publishing.

[Twitter 2011] Twitter (2011). Using OAuth 1.0a. Disponível em: https://dev.twitter.com/docs/auth/oauth. Acessado em 20 de Setembro de 2011.

[Veras 2009] Veras, M. (2009). *Datacenter:Componente Central da Infraestrutura de TI*.

[Verdi et al. 2010] Verdi, F., Rothenberg, C. E., Pasquini, R., and Magalhães, M. F. (2010). Novas Arquiteturas de Data Center para Cloud Computing. *XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - Gramado - RS*.

[VMware Inc. 2011] VMware Inc. (2011). VMware. Disponível em: http://www.vmware.com. Acessado em 22 de Agosto de 2011.

[Wangham et al. 2010] Wangham, M. S., de Mello, E. R., da Silva Böger, D., Gueiros, M., and da Silva Fraga, J. (2010). Gerenciamento de Identidades Federadas. In *Mini-curso - SBSeg 2010 - Fortaleza - CE*.

[Warnke and Ritzau 2010] Warnke, R. and Ritzau, T. (2010). *qemu-kvm & libvirt*. Books on Demand GmbH.

[Windley 2005] Windley, P. (2005). *Digital Identity*. O'Reilly.

[Windows Azure 2010] Windows Azure (2010). Windows Azure Platform. Disponível em: http://www.microsoft.com/windowsazure. Acessado em 2 de Setembro de 2011.

[Xen 2010] Xen (2010). Xenserver. Disponível em: http://www.citrix.com. Acessado em 10 de Setembro de 2011.

[Zappert 2010] Zappert, F. e. a. (2010). Cloud Computing Use Cases White Paper - Version 4.0. Technical report.

Capítulo
6

Live forensics em ambiente Microsoft Windows

Bruno Werneck Pinto Hoelz, Frederico Imbroisi Mesquita e Pedro Auler

Abstract

Conventional digital analysis is being challenged by the presence of encrypted content and the large volume of data to be processed. Live forensics can be applied to secure access to hard disk data and to perform the triage of evidence. However, this kind of analysis is considered complex due to the variety of information to be processed in a short period of time. This work presents the main concepts of live forensics, including its advantages and disadvantages when compared to the conventional methodology. Additionally, it provides a set of procedures grounded on system commands and open source tools for the Windows operating system, installed in more than 90% of desktop computers in Brazil.

Resumo

A presença de conteúdo criptografado ou de grande volume de informações a serem periciados são os novos desafios para a perícia digital convencional. A análise live*, ou* live forensics*, pode ser utilizada para garantir o acesso ao conteúdo do disco rígido e realizar a triagem de evidências. Porém, trata-se de uma perícia complexa devido à grande variedade de informações a serem analisadas em um curto período de tempo. Este trabalho apresenta os principais conceitos da análise* live*, incluindo suas vantagens e desvantagens quando comparada com a perícia digital convencional. Além disso, fornece um conjunto de procedimentos apoiado em comandos e ferramentas de código aberto para o sistema operacional Windows, presente em mais de 90% dos computadores de mesa do Brasil.*

6.1. Introdução

A análise *live*, também chamada de *live forensics*, consiste em uma análise digital realizada por meio de procedimentos periciais e conduzida no equipamento computacional ainda em execução. O procedimento pode ser relativamente simples e rápido, tal como listar as portas de conexões abertas. Porém, pode também ser complexo e demorado,

como, por exemplo, realizar buscas por palavras-chave no disco rígido utilizando expressões regulares e copiar integralmente o conteúdo deste disco por meio da porta USB. O procedimento apresenta vantagens e desvantagens em relação à perícia digital convencional, também chamada de análise *post-mortem*, realizada após o desligamento do sistema.

Coletar dados digitais em um sistema já desligado traz a vantagem de tornar a sobrescrita acidental ou modificação de dados praticamente impossível. Por outro lado, não permite a aquisição de dados voláteis, que são perdidos durante o processo de desligamento do sistema. Além disso, há outras situações em que a recuperação de dados permanentes também é inviabilizada. É o caso, por exemplo, do uso de criptografia, quando só é possível recuperar as informações com o uso da senha de acesso correta. Mais uma vez, esse problema seria contornado caso a aquisição lógica dos dados criptografados tivesse se dado com o sistema ainda ligado. Outro exemplo é a aquisição de informações referentes ao estado da rede e suas portas relacionadas, que também são perdidas ao se desligar o sistema.

Por isso, a coleta de dados com o computador ainda ligado parece ser uma alternativa salvadora. Essa técnica permite a recuperação de valiosas informações que de outra maneira poderiam ser perdidas. Infelizmente, essa abordagem também tem suas limitações. A mais importante é que cada computador analisado possui um sistema operacional diferente instalado. Assim, o analista precisa ter conhecimento de uma grande variedade de *hardware*, *software* e sistemas operacionais. O examinador precisa verificar o sistema em análise e aplicar os princípios forenses corretamente, de maneira a não inviabilizar a futura aceitação das evidências coletadas, quando utilizadas no devido processo legal. Parte do processo de aquisição de dados voláteis consiste em executar aplicativos na CPU do sistema suspeito, podendo levar a potenciais alterações de dados de registros, memória RAM ou do próprio disco rígido. Tais alterações devem ser controladas e documentadas. Dependendo de como se dá a abordagem no local de aquisição dos dados voláteis, a alteração do sistema pode ser tão expressiva que pode inviabilizar o uso futuro das informações coletadas.

A popularização do uso de programas de criptografia, cada vez mais fáceis de utilizar, e, muitas vezes incorporados aos sistemas operacionais, está tornando mais comum o fato de se encontrar sistemas ligados utilizando esta tecnologia. Em geral, não é muito fácil detectar a criptografia em uso no sistema, já que o *software* utilizado pode ser muito discreto, deixando poucos rastros da sua presença. Assim, a abordagem na coleta de dados voláteis em sistemas ligados tem que se ser bastante criteriosa, a fim de detectar a presença de criptografia e, se for o caso, fazer uma cópia lógica do sistema antes de desligá-lo.

A grande capacidade de armazenamento da memória RAM, muitas vezes igual ou superior a quatro gigabytes nos computadores atuais, é capaz de guardar grande quantidade de dados, podendo incluir, entre outras informações, senhas usadas para criptografia. A análise com o volume ainda montado possibilita ainda a aquisição lógica deste volume, que, de outra forma, apareceria como um arquivo criptografado, difícil de ser detectado e praticamente impossível de ser acessado.

Podem ser encontradas várias evidências extremamente úteis na memória RAM como, por exemplo, o conteúdo inteiro ou parcial de arquivos apagados, senhas em texto

claro, *buffers* com conteúdo da área de transferência, informações sobre processos em execução ou já encerrados. Portanto, não é mais possível ignorar a memória volátil dos computadores durante a fase de coleta de dados e a análise subsequente.

Apesar dos recentes progressos da análise de memória, as dificuldades ainda são grandes, devido à falta de flexibilidade das ferramentas existentes, que geralmente só podem ser utilizadas nos sistemas operacionais específicos e nas respectivas versões para as quais foram codificadas. A razão para isso é que as estruturas de dados utilizados pelos sistemas operacionais mudam a cada nova versão, exigindo que as ferramentas forenses também precisem ser atualizadas.

Neste trabalho, serão abordados aspectos fundamentais de perícia digital, incluindo questões de terminologia, um breve histórico da evolução da área e seus principais desafios. Posteriormente, os conceitos da análise *live* – incluindo suas vantagens e desvantagens, quando comparada com a perícia digital convencional – são apresentados. Os procedimentos de coleta e análise de evidências são detalhados, juntamente com um conjunto de ferramentas gratuitas para o ambiente Microsoft Windows, presente em mais de 90% dos computadores de mesa do Brasil, conforme dados da NetMarketShare, apresentados na Figura 6.1.

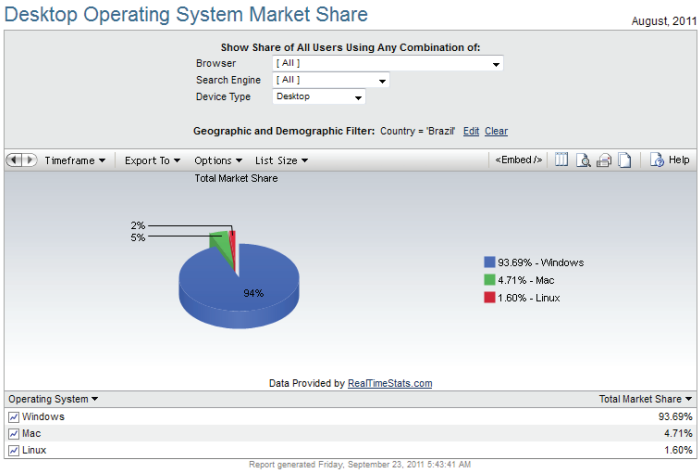

Figura 6.1. Base instalada de sistemas operacionais em computadores de mesa

6.2. Terminologia

Antes de aprofundar a discussão sobre análise *live*, é necessário esclarecer alguns pontos com relação à tradução dos termos em inglês e de outras denominações utilizadas. O termo em inglês associado à perícia em computadores é *Computer Forensics*. Em inglês, o termo *forensics* é definido como:

> a aplicação de conhecimento científico em problemas legais e especialmente

> a análise científica de evidências físicas como as encontradas em uma cena de crime[1].

Uma tradução muito utilizado comercialmente no Brasil é *forense computacional*, embora seja uma substantivação forçada do adjetivo forense, que não explicita a mesma semântica da definição do termo *forensics*. O termo mais próximo de *forensics* com base na definição apresentada seria, de fato, perícia e, por conseguinte, *Computer Forensics* poderia ser traduzido, adequadamente, como perícia em computadores.

Em relação à análise *live*, do inglês *live analysis*, este trabalho optou por manter o termo em inglês, tendo em vista que não há consenso sobre a tradução mais adequada. Opções como "ao vivo"ou simplesmente "viva"carregam uma conotação distinta. Análise "ao vivo"pode parecer mais adequado, no entanto indicaria que todo o processo de análise é realizado com o sistema ainda em execução, quando, na verdade, apenas a coleta é necessariamente feita nessa condição. Análise "viva", por outro lado, faz um contraponto à análise *post mortem*, termo comumente utilizado para se referir a análise de um sistema que foi desligado.

A definição utilizada neste trabalho para a Informática Forense é emprestada da definição de *Digital Forensic Science* discutida e apresentada em Palmer (2001). Assim, a Informática Forense é definida como:

> o uso de métodos cientificamente estabelecidos e comprovados para a preservação, coleta, validação, identificação, análise, interpretação, documentação e apresentação da evidência derivada de fontes digitais para o propósito de facilitar ou promover a reconstrução de eventos que causem a perturbação de operações planejadas.

O *Australian Institute of Criminology* apresenta uma definição mais adequada a esse ponto de vista, no qual a perícia em Informática é definida como:

> o processo de identificar, preservar, analisar e apresentar evidências digitais de uma maneira legalmente aceitável.

Ainda complementando a definição de Informática Forense, é preciso esclarecer a definição de evidência digital, que segundo Huebner et al. (2003) é:

> qualquer informação de valor probatório que é armazenada ou transmitida de forma digital.

Analogamente, o termo perícias em rede (*network forensics*) tem sido cada vez mais utilizado. Em Palmer (2001), é apresentada a seguinte definição de perícias em rede:

> é o uso de técnicas cientificamente comprovadas para coletar, unir, identificar, examinar, correlacionar, analisar e documentar evidências digitais de múltiplas fontes digitais processando e transmitindo ativamente, com o propósito

[1] Definição do dicionário Merriam-Webster disponível em `http://www.merriam-webster.com`

Tabela 6.1. Objetivos da Informática Forense em áreas diversas, adaptada de Palmer (2001)

Área	Objetivo primário	Objetivo secundário	Quando atua
Policial	Persecução penal	–	Depois do fato
Militar	Continuidade	Persecução penal	Tempo real
Comercial	Disponibilidade	Persecução penal	Tempo real

Tabela 6.2. Diferenças entre a segurança de computadores e a perícia em Informática, adaptada de Ruibin and Gaertner (2005)

Segurança	Perícia
Busca proteger o sistema de ataques	Não protege o sistema de ataques
Ação em tempo real ou logo após um incidente	Após os incidentes (*post mortem*)
Ambientes restritos para apresentação dos acontecimentos	A evidência é quase sempre apresentada para pessoal não técnico
Pode ser contornada por indivíduos confiáveis	A integridade da evidência é o mais importante

de descobrir fatos relacionados ao intento planejado ou sucesso apurado de atividades não autorizadas destinadas a perturbar, corromper ou comprometer componentes de sistema, bem como prover informação para auxiliar na resposta ou recuperação após atividades.

A definição das perícias em rede apresenta grande semelhança com as áreas de segurança de redes e resposta a incidentes, cujo objetivo principal é a proteção e a manutenção da disponibilidade de seus sistemas e redes. A Tabela 6.1, adaptada de Palmer (2001), apresenta os principais objetivos de cada área com relação à pesquisa e aplicação da perícia em Informática.

Em um cenário que não envolva o desejo de processar os atacantes, mas de proteger algum patrimônio ou informação, a ação invasora pode ser interrompida enquanto ocorre e, consequentemente, correções no sistema ou rede que implicam perda de evidências do ocorrido podem ser feitas. Ou seja, ações que podem ser boas práticas em respostas a falhas de segurança podem ser devastadoras do ponto de vista pericial. Essa situação é encontrada pelos profissionais de segurança de redes e detecção de intrusão, cujos procedimentos nem sempre estão em sintonia com os procedimentos periciais, já que a persecução penal, nesse caso, é uma preocupação secundária. Portanto, apesar das semelhanças entre a perícia em Informática e a segurança de computadores, tanto em termos de conhecimento quanto de ferramentas, existem algumas diferenças significativas, como as apresentadas por Ruibin and Gaertner (2005), aqui adaptadas e exibidas na Tabela 6.2.

A análise *live* busca aproximar os procedimentos das duas áreas. A realização do exame pericial tende a se aproximar do momento do incidente e a realizar procedimentos que, embora não mantenham o máximo de integridade da evidência, ainda são seguros do

ponto de vista jurídico.

6.3. Histórico

Segundo Huebner et al. (2003), o primeiro caso criminal relacionado ao uso de computadores conhecido foi registrado nos EUA, em 1966, e resultou em uma pena de cinco anos de prisão[2]. Nas décadas de 1970 e 1980, os computadores tornaram-se mais comuns e baratos, permitindo a utilização pessoal e comercial em maior escala. Com isso, a polícia identificou o surgimento de uma nova classe de crimes: os crimes relacionados a computadores. Nas décadas seguintes, a tecnologia passou por diversas evoluções significativas, que exigiram a adaptação contínua dos procedimentos periciais. Na década passada, a perícia em Informática caracterizou-se, predominantemente, pelo tratamento de dispositivos com capacidades de armazenamento relativamente pequenas e poucas quantidades de informação. Isso permitia que cópias completas dos discos rígidos originais fossem feitas para outro disco, sendo o exame realizado sobre a cópia, preservando, assim, a evidência original.

Com o surgimento da Internet comercial no Brasil no início de 1995[3], surge uma nova demanda relacionada à prática de crimes com o auxílio da Internet, hoje conhecidos comumente como crimes cibernéticos. Com o crescimento explosivo da Internet, também cresceram as ocorrências de incidentes relacionados, como a invasão de servidores e a prática de *defacement* de páginas web.

Além do crescimento da Internet, é importante destacar, também, a evolução da telefonia móvel e dos meios de armazenamento de dados. Ambos tornaram-se extremamente acessíveis e hoje fazem parte da vida cotidiana. Da mesma forma, pode-se afirmar que fazem parte do cotidiano de criminosos, que muitas vezes fazem uso dessas tecnologias, mesmo que os crimes que cometam não tenham relação direta com a Informática. Hoje, discos rígidos com capacidade da ordem de terabytes podem ser adquiridos. Esse grande volume de dados e diversidade de mídias é um dos grandes desafios enfrentados pela Informática Forense, que precisa buscar novas soluções para cenários onde não é mais possível realizar uma cópia integral de todos os dados originais, como em ambientes de rede complexos.

No Brasil, cabe ainda considerar a carência de uma legislação específica para punir diversas condutas no meio cibernético, que em muitos países já são consideradas crimes, como a disseminação de programas maliciosos (*malware*). Diversos projetos de lei foram propostos ao longo dos anos, mas até a conclusão deste trabalho nenhum havia sido aprovado em caráter conclusivo.

6.4. Desafios da perícia digital

Muitos dos desafios enfrentados hoje na Informática Forense são produto dos grandes avanços tecnológicos observados nos últimos 15 anos. Esta seção apresenta alguns do desafios mais discutidos e que são os principais alvos de pesquisas.

Nas discussões do *First Digital Forensic Research Workshop*, ocorrido em 2001, e

[2]Tratava-se de um caso de furto de programa de computador.

[3]Considerando a data de criação do Comitê Gestor de Internet em maio de 1995.

apresentado em Palmer (2001), alguns dos desafios de alta prioridade citados então eram a confiabilidade da evidência digital e as perícias em ambientes de rede. Essas questões ainda estão presentes, atualmente, e em escala cada vez maior. As evidências digitais tornaram-se cada vez mais comuns e as redes de computadores, maiores e mais presentes. Comentou-se, então, que a ubiquidade dos sistemas de informática e equipamentos eletrônicos, cada vez mais, indicava que um dia todos os crimes teriam uma "ciberdimensão".

Uma tendência observada nos últimos anos é a expansão da perícia além do simples exame dos discos rígidos. A análise de memória volátil e de sistemas em operação tem recebido bastante atenção em termos de pesquisa e de ferramentas específicas. É interessante notar que a perícia em sistemas em operação "desrespeita"um dos princípios básicos da perícia em mídias de armazenamento, que é a preservação total dos dados originais. Isso porque qualquer atividade em um sistema em operação causa mudanças nos dados armazenados na memória. Como tal situação é inevitável, Huebner et al. (2003) argumentam que evidências coletadas dessa forma têm que ser aceitáveis em juízo.

Além das dificuldades discutidas acima, que afetam de maneira geral o trabalho pericial em Informática, Huebner et al. (2003) apresenta ainda diversos desafios técnicos como:

1. sistemas de arquivos que permitem ocultar dados do usuário comum, sendo visíveis apenas se utilizadas ferramentas especiais;

2. propriedades e mecanismos de sistemas operacionais e aplicativos sem documentação ou utilizados para ocultar dados;

3. armazenamento de dados *online*, que permite o armazenamento de dados em serviços da Internet, cujo acesso pode ser difícil ou pode encontrar-se fora da jurisdição legal daquela polícia e pode até requerer ações demoradas de cooperação internacional;

4. uso extensivo de criptografia forte, que sugere a necessidade de um trabalho maior de investigação para evitar que as evidências digitais do suspeito estejam protegidas dessa forma;

5. dispositivos móveis de alta capacidade e dimensões muito reduzidas, que podem ser facilmente destruídos ou ocultados, ou que podem ser utilizados para evitar que dados importantes fiquem armazenados nos discos rígidos dos computadores utilizados;

6. serviços *online* diversos como *webmail*, redes sociais ou programas de mensagens instantâneas, cujos vestígios encontram-se nas mãos dos provedores dos serviços, o que dificulta sua coleta.

O uso de técnicas cada vez mais sofisticadas de proteção e ocultação de dados, como criptografia integral de disco e esteganografia, também reforçam a necessidade de uma ação mais proativa para identificar e preservar vestígios. A análise *live* surge como uma possível resposta para alguns desses desafios.

6.5. Princípios de perícia digital

Segundo Palmer (2001), por definição, a perícia em Informática tem uma natureza investigativa e seus praticantes devem seguir um processo investigativo na realização de seu trabalho. Ao investigar crimes relacionados a computadores, deve ficar claro que os princípios básicos aplicados a cenas de crime "comuns"também se aplicam. Nesta seção, são apresentados alguns princípios de perícia digital que serão utilizados ao longo do trabalho. Também são detalhadas as fases do trabalho pericial, que se aplicam tanto à análise convencional quanto à análise *live*.

De acordo com Huebner et al. (2003), a primeira coisa de que um investigador deve estar ciente é o *Princípio da Troca de Locard*, segundo o qual

> qualquer pessoa ou coisa entrando em uma cena de crime leva algo da cena consigo ou deixa algo de si para trás quando sai da cena,

ou, como apresentado por Reith et al. (2002), toda atividade em um computador provavelmente produzirá uma modificação no sistema em que foi realizado como, por exemplo, modificações no sistema de arquivos ou, no mínimo, modificações na sua memória principal. Nesse sentido, um princípio básico é o da preservação dos vestígios originais. Sempre que possível, procura-se trabalhar sobre uma cópia integral e exata dos dados originais. Um alto nível de integridade dos vestígios é necessário em todos os exames periciais de Informática, já que materiais digitais são mais facilmente adulterados e forjados do que materiais físicos.

Com o desenrolar dos exames e a descoberta de evidências no material examinado, é importante manter a rastreabilidade dessas descobertas e de suas correlações. Da mesma forma, os dados são transformados e interpretados por ferramentas diversas. É desejável que todos os procedimentos realizados sejam claros e totalmente compreendidos pelos especialistas, embora nem sempre as ferramentas utilizadas permitam uma análise e avaliação mais profunda do seu funcionamento.

6.5.1. Cadeia de custódia

A cadeia de custódia é um processo usado para manter e documentar a história cronológica da evidência, para garantir a idoneidade e o rastreamento das evidências utilizadas em processos judiciais. A cadeia de custódia trata dos procedimentos que buscam garantir a idoneidade das evidências por meio da descrição e documentação detalhada de como a evidência foi encontrada e de como foi tratada dali por diante.

Todo o procedimento deve ser documentado para que fique registrado onde, quando e por quem a evidência foi descoberta, manipulada, coletada e armazenada. Quando a evidência passa para a responsabilidade de outra pessoa, esse fato, com todos os detalhes envolvidos, incluindo número de lacres e outros procedimentos de segurança, deve ser, também, cuidadosamente documentado. Turner (2005) discute extensivamente a importância de comprovar a procedência das evidências obtidas em meios digitais. A incapacidade de demonstrar a continuidade dessa cadeia de custódia em um processo tem um sério impacto na aceitação da prova.

6.5.2. Fases

Beebe and Clark (2005) sugerem uma divisão em seis fases, nas quais um número arbitrário de subfases pode ser definido, considerando a necessidade de sua execução conforme a natureza do caso. As seis fases são:

1. preparação (pré-incidente);
2. resposta ao incidente;
3. coleta de dados;
4. análise;
5. apresentação das descobertas;
6. encerramento do incidente.

A Figura 6.2 apresenta a relação das seis fases com a possibilidade de iteração entre elas ou de todo o processo.

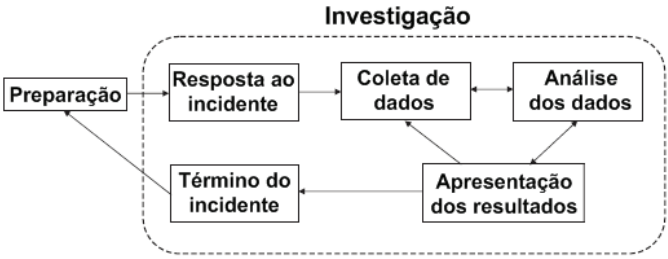

Figura 6.2. Fases do trabalho pericial, adaptada de Beebe and Clark (2005)

A diferença da análise *live* para a convencional está principalmente na fase de coleta de dados e análise, que são realizadas com o sistema ainda em execução. Na análise convencional, o sistema seria desligado antes de qualquer outro procedimento ser realizado.

6.5.3. Funções de *hash*

As funções de *hash* têm uma importância fundamental nos procedimentos da perícia digital. Elas são funções que relacionam uma entrada de tamanho variável a uma saída de tamanho fixo. Essas funções de *hash* são comumente utilizadas em criptografia, autenticação e assinatura digital. Alguns exemplos de algoritmos utilizados são MD5, SHA-1, SHA-256 e SHA-512. Uma característica fundamental das funções de *hash* é que é muito difícil encontrar dois conteúdos de entrada que produzam o mesmo resultado na saída, e, a partir da saída, é computacionalmente inviável encontrar a entrada. Dessa forma, essas funções são utilizadas para garantir a integridade de arquivos digitais. No caso de

modificação no conteúdo de um arquivo, mesmo que mínima, o valor da função de *hash* muda drasticamente, evidenciando a existência de alteração. O quadro a seguir apresenta um exemplo de modificação no conteúdo e seu reflexo no resultado da função de *hash* utilizando o algoritmo MD5.

```
> Conteúdo: <test>
>> MD5: a55ab7512f0d0ff4527d898d06afd5c5

> Conteúdo: <teste>
>> MD5: c0d810f61f37025e600cf41e716c8576
```

A utilização das funções de *hash* para garantir a integridade das evidências encontradas será abordada posteriormente neste trabalho, como parte dos procedimentos a serem executados durante a perícia, assim como uma indicação de ferramentas para realizar o processo.

6.6. Análise *live*

Conforme descrito inicialmente, a análise *live*, também chamada de *live forensics*, consiste em uma análise digital realizada por meio de procedimentos periciais e conduzida no equipamento computacional ainda em execução. Portanto, a análise *live* ocorre quando o sistema é mantido em execução e os investigadores usam o próprio sistema operacional da máquina para acessar os seus dados.

Segundo Anson and Bunting (2007), os ingredientes principais para realizar uma análise *live* bem-sucedida são:

- interagir o mínimo possível com o sistema em análise;
- utilizar ferramentas confiáveis;
- pensar e repensar, pois uma vez feito o procedimento em um sistema em execução, o sistema modificará o estado atual, sendo impossível retornar ao estado inicial;
- documentar todo o procedimento.

Para melhor compreender o potencial da análise *live* com relação a análise convencional, pode-se fazer uma reflexão sobre as fontes de dados existentes em um computador, conforme apresentado na seção a seguir.

6.6.1. Fontes de dados em um computador

Os componentes de um computador são agrupados em três componentes básicos: unidade central de processamento (CPU), a memória principal (RAM) e os dispositivos de entrada e saída. A Figura 6.3 ilustra a interação desses componentes.

Processadores O processador, ou Unidade Central de Processamento (CPU), tem como função principal unificar todo o sistema, controlando as funções realizadas pelos outros

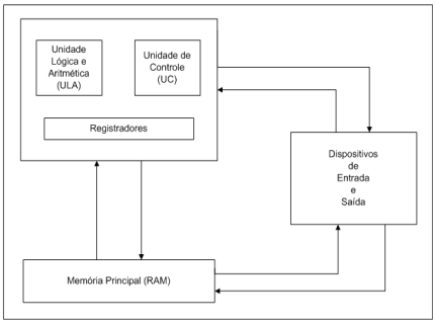

Figura 6.3. Interação entre os componentes básicos do computador

componentes. A CPU é composta por dois componentes básicos, a unidade de controle (UC), e a unidade lógica e aritmética (ULA).

A função da CPU é buscar instruções na memória e executá-las, em seguida. Seu ciclo básico de execução é buscar a instrução da memória, decodifica-la para determinar seus operandos e funções a executar, executá-la, e em seguida, tratar a instrução seguinte, até que o programa pare.

Registradores São dispositivos de alta velocidade, localizados fisicamente na CPU, para armazenamento temporário de dados. Um registrador é o elemento superior da pirâmide da memória, por possuir a maior velocidade de transferência dentro do sistema, menor capacidade de armazenamento e maior custo. O sistema operacional deve estar sempre atento ao estado e ao conteúdo dos registradores. Quando o sistema operacional compartilha a CPU com mais de um programa, necessita, às vezes, interromper um programa e iniciar outro. Nesse caso, é necessário que os dados contidos nos registradores sejam salvos, para que possam ser recuperados posteriormente, quando seu programa de origem voltar a ser executado.

Memória cache Esta memória, hierarquicamente, está abaixo da camada de registradores, sendo controlada, principalmente, por *hardware*. É uma memória de alta velocidade, mais lenta que os registradores, mas mais rápida que a memória principal. Os modernos computadores costumam ter dois ou até três níveis de cache, sendo o seu tamanho limitado pelo alto custo. A cada nível subsequente, diminui a velocidade e aumenta a capacidade de armazenamento. Todas as requisições da CPU que não podem ser atendidas pela memória cache são direcionadas para a memória principal.

Memória principal (RAM) Também conhecida como memória primária, real ou RAM (*random access memory*). Posições de memória frequentemente ou recentemente utilizadas são mantidas na memória cache. Quando o programa precisa ler um dado na memória, o *hardware* verifica se o dado está na memória cache. Se estiver (*cache hit*), nenhuma

requisição adicional é necessária. Caso contrário (*cache miss*), há necessidade de uma requisição adicional, enviada à memória principal, com perda substancial de tempo.

Memória secundária É um meio permanente de armazenamento. Enquanto os dados contidos em registradores, memória cache e memória principal são voláteis – sendo perdidos no momento de desligamento do computador – a memória secundária permanece armazenada mesmo depois do desligamento da máquina. Trata-se de uma memória de acesso bem mais lento, quando comparado às memórias voláteis. Sua vantagem, porém, está no menor custo e na alta capacidade de armazenamento. Exemplos desse tipo de memória são os discos rígidos e os flash drives.

Dispositivos de entrada e saída São os dispositivos que permitem a comunicação entre o computador e o mundo externo. Podem ser divididos em duas categorias: na primeira, estão os dispositivos utilizados como memória secundária e na segunda, os dispositivos que permitem a interação do ser humano com o computador, como teclado, monitor, mouse, impressora e scanners.

O objetivo de discutir os componentes da arquitetura básica do computador é chamar a atenção para as possíveis fontes de evidências digitais. Observa-se que a volatilitade diminui a medida que a capacidade de armazenamento aumenta. Elementos com maior volatilidade como registradores e caches não apresentam vantagem do ponto de vista pericial em relação ao conteúdo da memória RAM, que por ser menos volátil é mais fácil de ser analisada. Assim como os discos rígidos apresentam um volume bem maior de material potencialmente interessante para o exame pericial. Deve-se ter em mente que apesar dos dispositivos de entrada e saída não terem função primordial de armazenamento de dados, alguns deles fazem uso de algum tipo de memória para viabilizar sua operação. Como exemplo, algumas impressoras utilizam um disco rígido que armazena cópias de arquivos a serem impressos.

6.7. Diferenças com relação à perícia convencional

A perícia convencional ocorre com o sistema investigado desligado. Para evitar novas escritas no disco, remoção de arquivos temporários ou qualquer modificação no sistema, aconselha-se desligar o computador utilizando o procedimento *pull the plug*. Esse procedimento consiste na interrupção do fornecimento de energia ao equipamento pela retirada do cabo de energia da tomada. Após a coleta do equipamento computacional, uma cópia integral do disco rígido do sistema é realizada e, a partir daí, essa imagem é analisada em laboratório, utilizando-se um sistema operacional e aplicações forenses confiáveis (Carrier, 2006).

Diferentemente da perícia convencional, que fornece apenas uma visão limitada das informações do sistema, ferramentas para análise *live* podem informar ao investigador um cenário mais completo do estado do computador (Hay et al., 2009). Segundo Adelstein (2006), enquanto a perícia convencional tenta preservar os discos rígidos em um estado inalterado, as técnicas de análises em equipamentos computacionais ligados têm como objetivo tirar *snapshots* do estado da máquina, similar às fotografias de uma

cena de crime.

Ferramentas periciais, na maioria das vezes, são bem-sucedidas na extração de dados dessas mídias, inclusive na recuperação de arquivos apagados não sobrescritos e buscas por palavras-chave. A perícia convencional, apesar de amplamente usada atualmente na persecução penal, apresenta limitações nos seguintes casos:

1. impossibilidade de coletar o equipamento computacional;
2. necessidade de estabelecer o flagrante do suspeito;
3. uso de criptografia forte.

No item 1, existem casos em que não há permissão legal ou viabilidade técnica para coletar o equipamento computacional devido à importância deste para a organização. Equipamentos de grande porte, como *mainframes*, também inviabilizam sua apreensão pela dificuldade de transporte e armazenamento do *hardware*.

No item 2, o desligamento sumário do equipamento computacional inviabilizará o flagrante, já que não haverá a constatação dos requisitos necessários para a sua configuração. Sendo assim, a prova deve ser extraída e documentada antes do procedimento de desligamento e coleta do equipamento.

O item 3 refere-se ao mais recente desafio da perícia digital: o uso, cada vez mais difundido, de esquemas criptográficos robustos nos computadores pessoais, dificultando, consideravelmente, a extração de dados pela perícia convencional, podendo até mesmo inviabilizá-la por completo.

A análise *live* possui vantagens e desvantagens se comparada com a perícia convencional. Segundo Anson and Bunting (2007), o investigador deve determinar qual opção representa uma ameaça maior à perícia: a perda dos dados da memória RAM ou a modificação dos dados no disco rígido.

6.8. Vantagens da análise *live*

Devido à análise em um computador ligado fornecer uma visão mais completa do sistema investigado, com acesso a informações na memória RAM, ela pode ser usada para resolver algumas das limitações encontradas pela perícia convencional. Entre as vantagens obtidas no uso desse tipo análise, é possível elencar como as principais:

- extração de dados voláteis;
- triagem de equipamentos;
- triagem de dados;
- preservação de dados criptografados;
- possiblidade de estabelecer o flagrante.

Cada uma dessas vantagens é detalhada nas seções que se seguem.

6.8.1. Extração de dados voláteis

De acordo com Adelstein (2006), a análise *live* pode resguardar tanto as informações voláteis quanto as informações estáticas sobre o sistema de arquivos. De acordo com Carrier and Spafford (2005), o pré-processamento de dados na cena de crime é apenas uma das fases na investigação digital. A perícia em um computador ainda ligado permite ao investigador analisar um elemento indisponível durante a perícia convencional: a memória RAM. Sendo assim, o investigador terá acesso a informações não tipicamente escritas em disco, tais como: portas abertas, conexões de rede ativas, programas em execução, dados temporários, interação com usuário, chaves criptográficas e conteúdos não criptografados.

Com o uso apenas da perícia convencional, essas informações eram, simplesmente, ignoradas, o que pode ser prejudicial à investigação. Quanto maior a probabilidade de alteração nas informações do dispositivo computacional, maior é a prioridade de extração e preservação desses dados. Como a memória RAM é mais suscetível a mudanças, alerta que a extração deve seguir a ordem de volatilidade. Portanto, na maioria das vezes, é necessário que as informações sejam extraídas da memória antes da extração dos dados do disco rígido.

6.8.2. Triagem de equipamento

A presença crescente de computadores e mídias de armazenamento computacional na vida cotidiana refletiu-se também nas cenas de crime, onde comumente são encontrados computadores que apresentam relação com o fato sob investigação (Hoelz, 2009). Segundo Adelstein (2006), discos rígidos com mais capacidade de armazenamento aumentam o tempo necessário para análise, dificultando e encarecendo-a quando há a coleta de todos os discos rígidos.

A análise *live* permite ao investigador filtrar os equipamentos computacionais que são realmente de interesse à investigação. A busca por palavras-chave no disco rígido ou por aplicativos instalados na máquina, por exemplo, podem evitar a apreensão desnecessária de computadores. Em casos nos quais os resultados da perícia devem ser disponibilizados em um curto espaço de tempo, um modelo para triagem de equipamentos pode ser utilizado durante a análise *live* (Rogers et al., 2006). Esse modelo envolve consultas no computador investigado em busca de informações contidas nos arquivos de registro, histórico de Internet, mensagens eletrônicas entre outros. A triagem de equipamentos ajuda o perito a dedicar-se à perícia dos equipamentos computacionais relevantes, pois reduz a quantidade total de equipamentos apreendidos. Sendo assim, as análises resultam em um relatório com mais qualidade e tempestividade.

6.8.3. Triagem de dados

Devido ao atual aumento da quantidade de evidências digitais disponíveis, em breve será impossível obter todos os dados referentes ao caso. Com isso, o paradigma da análise *live* poderá se tornar o procedimento padrão. Técnicas como mineração de dados e uso de filtros de arquivos conhecidos estão sendo utilizadas para processar casos contendo grande volume de dados, porém não resolvem o problema por completo.

A extração seletiva de dados no computador investigado em execução pode facilitar a perícia posterior, principalmente em casos onde os dados estão em servidores

corporativos (banco de dados, servidores de e-mail), *mainframes* ou máquinas que contenham *hardware* que dificulte a perícia convencional, como RAID (*Redundant Array of Independent Disks*), por exemplo. Segundo Aquilina et al. (2008), nem sempre é possível extrair todos os dados de todas as máquinas envolvidas no incidente, sendo mais eficiente a extração de alguns dados de cada máquina para determinar quais sistemas realmente foram afetados.

6.8.4. Preservação de dados criptografados

A criptografia é um dos melhores métodos para ocultar informação e tem sido amplamente utilizada por criminosos para esconder o conteúdo de seus arquivos. O uso de volumes criptografados complica, significativamente, a perícia convencional. Supondo o uso de algoritmos fortes, métodos convencionais de investigação, em geral, possuem um baixo retorno em função do investimento. Uma vez que o sistema é desligado, a chave criptográfica necessária para acessar a mídia de armazenagem normalmente não está mais disponível (Hay et al., 2009). Houve avanços no processamento de dados criptografados, como o uso de *rainbow tables* e ataques por dicionário. No entanto, técnicas anti-forenses também evoluíram para dificultar a decifração desses conteúdos criptografados. Caso o sistema não seja desligado, a análise *live* permite ao investigador acessar os dados de forma transparente, como um usuário do sistema, e realizar uma cópia para analisá-los posteriormente.

6.8.5. Possibilidade de estabelecer flagrante

Outra grande vantagem do uso de técnicas de análise em computadores ligados é a possibilidade de constatação de uma situação de flagrante. A perícia convencional simplesmente ignorava tal possibilidade e desligava a máquina investigada, perdendo a oportunidade de registrar o estado da máquina, processos em execução e arquivos que serviriam para estabelecer a situação de flagrante.

6.9. Desvantagens da análise *live*

Apesar de resolver alguns problemas encontrados na perícia convencional, a perícia em equipamentos computacionais em execução também introduz novos desafios e possui suas próprias limitações. Além de aumentar a quantidade de informações que o perito deve analisar, é possível citar as seguintes desvantagens no uso desse tipo de análise:

1. aspectos legais e impossibilidade de reprodução do exame;
2. tempo gasto;
3. complexidade e variedade de cenários;
4. mudança de paradigma na investigação;
5. *rootkits* e *malware*.

Analogamente, cada uma dessas devantagens é detalhada nas seções que se seguem.

6.9.1. Aspectos legais e impossibilidade de reprodução do exame

É necessário considerar as indagações sobre aspectos legais como uma parte do esforço de pesquisa nas análises de sistemas em execução (Hay et al., 2009). Caso a prova digital não seja válida legalmente, pouco adianta os resultados da análise *live*.

Durante a realização da perícia em um computador ligado, é muito fácil contaminar a prova no sistema, exigindo que os procedimentos sejam feitos por um profissional qualificado. Durante a realização dos exames, o sistema se modifica continuamente, impossibilitando obter exatamente os mesmos resultados ao se repetir a perícia. A boa prática exige que o investigador, quando realizando um procedimento em uma máquina investigada, minimize o impacto e compreenda o efeito desse procedimento no sistema analisado.

Uma vez que a análise *live* extrai dados da memória volátil, esse exame não poderá ser repetido posteriormente produzindo exatamente os mesmos resultados. Essa impossibilidade de reprodução exige uma documentação precisa dos procedimentos realizados e uma atenção ainda maior na preservação da integridade dos dados extraídos durante a análise dos equipamentos computacionais ligados.

Os dados extraídos durante os exames devem seguir o mesmo tratamento dispensado à perícia convencional. Deve-se utilizar uma função de *hash* para garantir integridade desses dados. Além disso, a análise *live* deve se manter em harmonia com os preceitos da cadeia de custódia. O objetivo de manter, cuidadosamente, a cadeia de custódia não consiste apenas em proteger a integridade da evidência, mas, também, de tornar difícil ao advogado de defesa arguir que a evidência foi mal manipulada enquanto esteve na posse do investigador.

6.9.2. Tempo gasto

A utilização racional do tempo é sempre importante e, para tanto, a escolha entre as atividades de análise que devem ser feitas no ambiente em execução e quais podem ser realizadas posteriormente sem prejuízo às investigações é essencial.

Alguns procedimentos utilizados quando os computadores estão ligados, tais como a cópia integral do disco rígido, podem ser demorados. O local da análise *live*, ao menos no caso policial, não é o local mais adequado para realização de exames periciais. Sendo assim, espera-se que esses procedimentos sejam os mais breves possíveis. Para Adelstein (2006), o investigador pode realizar uma triagem e coletar dados essenciais, examiná-los e usar o resultado dessa análise para decidir o que é mais necessário.

6.9.3. Complexidade e variedade de cenários

A grande quantidade de aplicativos, sistemas operacionais e dispositivos computacionais encontrados durante a perícia em um equipamento ligado frustram a preparação do perito na realização desse tipo de análise. A preparação é requisito fundamental para o sucesso da análise, sendo inadmissível testar ferramentas e procedimentos durante a realização da análise *live* (Mandia et al., 2003).

O sucesso nesse tipo de análise depende de um treinamento constante do perito na área de computação, assim como do estudo e da evolução contínua dos procedimentos

periciais realizados em um computador ligado. Como visto, o investigador deve ter um conhecimento amplo em diversas áreas computacionais, porém é humanamente impossível exigir o domínio em todas as particularidades encontradas durante a análise *live*, sendo necessária a utilização de uma ferramenta para auxiliá-lo nesse processo.

6.9.4. Mudança de paradigma na investigação

Para a realização de uma análise *live*, é indispensável que a máquina esteja ligada. Caso a máquina esteja desligada, não existe nada mais a fazer, além de coletar o equipamento para realização de perícia convencional e esperar que a máquina não esteja utilizando algum algoritmo de criptografia forte. Caso seja necessário realizar a análise de uma máquina ligada, a investigação deve se adaptar à necessidade imposta por esse tipo de análise, ou seja, realizar a coleta apenas se tiver certeza de que o equipamento computacional estará ligado.

6.9.5. *Rootkits* e *malware*

Um *rootkit* é um *software* que permite acesso privilegiado e contínuo a um computador, ao mesmo tempo em que fica invisível aos administradores do sistema, subvertendo as respostas normais e esperadas de comandos do sistema operacional ou de outros aplicativos. O termo *rootkit* é uma concatenação dos termos *root* e *kit*. *Root* é nome tradicional da conta com privilégios de administrador do sistema, nos sistemas operacionais UNIX, enquanto o termo *kit* refere-se aos componentes de *software* com integram a ferramenta. Tipicamente, o *rootkit* é instalado na máquina pelo atacante, após ter obtido poderes de administrador do sistema, explorando alguma vulnerabilidade conhecida ou tendo acesso à senha de administrador. Os *rootkits* são de difícil detecção, já que podem subverter o próprio *software* que supostamente deveria detectá-lo.

Rootkits podem ser classificados em dois níveis: de aplicativo e de *kernel*. O primeiro tipo é encontrado em aplicativos nativos do próprio sistema operacional, sendo capaz de omitir resultados de uma consulta desse aplicativo modificado por meio de um filtro pré-determinado. Já o segundo é incorporado ao sistema operacional e, por isso, utiliza um filtro para omitir resultados independentemente do aplicativo que está sendo executado. Segundo Carrier (2006), os *rootkits* de nível de aplicativo podem ser contornados utilizando executáveis conhecidos e trazidos pelo próprio investigador, mas os *rootkits* de nível de *kernel* necessitam da utilização de uma abordagem mais complexa, sendo necessário buscar inconsistências ao correlacionar diversas estruturas em memória.

Além dos *rootkits*, é importante estar atento para a presença de *malware*. Este termo vem do inglês (*malicious software*), significando *software* malicioso. Refere-se a programas desenvolvidos para alterar ou danificar o sistema, roubar informações ou provocar outras ações não realizadas pelo usuário atual. Exemplos comuns de *malware* incluem vírus, *worms*, *trojans* e *spyware*. Embora não afetem diretamente os resultados do exame, como no caso dos *rootkits*, podem ser utilizados como estratégia de defesa, com a alegação de que "a culpa foi do *malware*".

A análise e engenharia reversa de *malware* é um assunto amplo e de grande valia para o exame pericial, mas que está além do escopo deste trabalho.

6.10. Análise da memória RAM

A grande vantagem do uso da análise *live* em relação à análise convencional é o acesso aos dados residentes na memória do computador investigado. Apesar de ser substancialmente menor que a capacidade de armazenamento dos discos rígidos, a memória RAM é uma rica fonte de informações para investigação. É possível extrair da memória informações como: processos em execução, arquivos abertos, conexões estabelecidas, portas abertas e possíveis senhas. Em sistemas Windows pode-se extrair a lista de DLLs e os arquivos de registro (*hive files*), que permanecem residentes na memória. Essas informações, com exceção dos arquivos de registro – que também são acessíveis pelo disco rígido –, eram simplesmente ignoradas pela perícia convencional. Com o uso da perícia em equipamentos em execução, essas informações ganham papel de destaque, tornando esse tipo de análise uma importante fase da perícia digital.

A RAM é uma memória volátil e bastante dinâmica que exige cuidado durante sua análise. Devido à volatilidade da memória, a análise em equipamentos computacionais ligados oferece ao perito uma breve janela de oportunidade para extração de dados desta, antes que seja efetuado o desligamento e apreensão do equipamento computacional. A memória RAM de um computador ligado permanece continuamente alterando seu conteúdo. Sendo assim, torna-se necessária uma documentação criteriosa das ações realizadas pelo perito.

6.10.1. Análise com ferramentas específicas

Além de aplicativos próprios do sistema operacional, existem outras ferramentas digitais forenses, aplicativos de administração de rede e utilitários de diagnósticos, que podem ser usados durante a análise *live*. Não se pode subestimar a importância do processo monótono de criação de um kit de ferramentas para realizar esse tipo de análise. O tempo gasto nesse processo será compensado pelos resultados mais rápidos, profissionais e bem-sucedidos (Mandia et al., 2003).

Mesmo utilizando ferramentas específicas, é possível a ocorrência de falsos negativos caso um *rootkit* de nível de *kernel* esteja instalado no sistema operacional. Sendo assim, o sistema operacional pode omitir informações de interesse à investigação independentemente da ferramenta que esteja sendo utilizada.

6.10.2. Análise do *dump* da memória

Neste tipo de análise, o conteúdo integral da memória RAM é copiado para um arquivo denominado *dump*. Este arquivo é então processado em um ambiente preparado e controlado por meio do uso de analisadores, responsáveis por percorrer o conteúdo do arquivo em busca de padrões. Na análise do arquivo de *dump* da memória é feita a busca por assinaturas das estruturas de dados em memória.

Por não utilizar as APIs do próprio sistema operacional, essa abordagem de análise apresenta uma desvantagem: incompatibilidade com os diversos sistemas operacionais e versões. As estruturas de dados usadas e a organização dos elementos carregados em memória dependem do sistema operacional do computador. Suas assinaturas variam com cada versão do sistema operacional, incluindo atualizações significativas como *services packs*. É necessário conhecer profundamente essa organização para realizar a extração

desses elementos do arquivo da memória *dump*.

Apesar de não ser possível extrair novamente o conteúdo da memória após o desligamento da máquina investigada, é possível reanalisar o arquivo de *dump*. Essa propriedade é desejável para a filosofia da perícia digital, já que permite uma reprodução parcial do procedimento, além de permitir a extração de outros dados não extraídos durante a primeira análise.

6.10.3. Mecanismos para cópia da memória RAM

Diferentemente da duplicação de discos rígidos pela perícia convencional, a memória RAM não pode ser desconectada do sistema para realização de cópia devido à volatilidade de seus dados. Conforme a arquitetura física, uma vez que a memória não é mais alimentada de energia, o estado dos dados na RAM é desconhecido. Existem duas formas para realizar a cópia total dos dados da memória RAM para um arquivo de *dump*: com suporte de *software* ou de *hardware*.

Cópia com suporte de *software* Consiste na técnica mais utilizada para cópia do conteúdo da memória física. Existem diversos aplicativos que realizam essa cópia, porém como esses aplicativos são executados no próprio sistema operacional investigado, há uma alteração do conteúdo na memória. Sendo assim, o examinador deve dar preferência às ferramentas menos invasivas, ou seja, que alterem o mínimo possível do conteúdo da memória. Um conjunto contendo um bloqueador de escrita para extração de memória volátil de um computador investigado integrado pode ser utilizado para diminuir ainda mais a interação com o sistema.

Cópia com suporte de *hardware* Devido à volatilidade dos dados, não existe um *hardware* específico para realizar uma cópia do conteúdo da memória RAM, porém é possível utilizar a propriedade DMA (*Direct Memory Access*) de alguns dispositivos de *hardware* para realizar essa cópia. O DMA fornece transferência de dados entre o barramento PCI e a memória sem que seja necessário usar recursos do processador. A cópia física do conteúdo da memória RAM é possível pelo acesso direto à memória na porta *Firewire* (IEEE 1394). Apesar de menos invasiva em relação à cópia física por *software*, essa abordagem não é sempre possível. Porém, é recomendada quando o computador encontra-se bloqueado com senha pelo usuário. Essa técnica permite o acesso à memória do computador bloqueado e, por meio da alteração de processos em execução, seu desbloqueio.

6.11. Metodologia

A Figura 6.4 apresenta a metodologia para a realização da análise *live*. Um dos aspectos mais importantes do campo da computação forense é a documentação. Além de documentar seus próprios atos durante a coleta dos dados, o analista deve também documentar o ambiente antes de começar efetivamente a intervir nos sistemas. Para que a documentação seja feita da melhor forma possível, é recomendado que haja uma pessoa responsável exclusivamente por essa tarefa. Alguns itens que devem ser documentados com atenção incluem:

1. telas do computador, com resolução suficiente para leitura de textos ali presentes, se necessário;

2. conexões de rede, mostrando quaisquer cabos de rede conectados ao computador. As duas pontas do cabo devem ser fotografadas, para o caso em que o analista tenha que provar que o computador estava conectado a algum equipamento específico;

3. conexões de periféricos, para provar que estavam conectados ao computador.

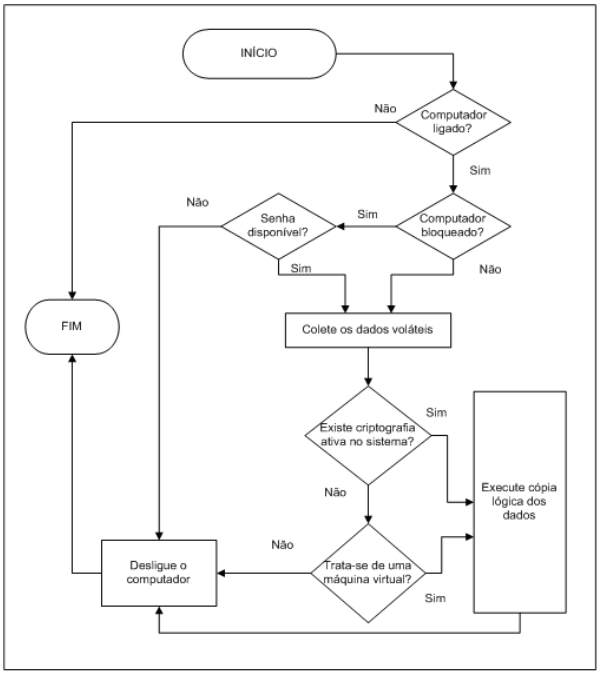

Figura 6.4. Metodologia para a realização da análise *live*

Deve-se impedir que um equipamento ligado seja desligado por intervenção humana ou que fique indisponível, por exemplo, em modo de hibernação automática ou de proteção de tela com senha, já que será alvo de captura de dados voláteis. É necessário, também, impedir que outras evidências sejam ocultadas, adulteradas ou destruídas. Um reconhecimento do local, a fim de localizar equipamentos, verificando as conexões citadas anteriormente deve ser realizado.

É recomendado que sejam tiradas fotografias para ilustrar o estado em que se encontra o ambiente. As fotografias, de preferência digitais, e com bastante resolução, devem fazer parte da documentação. A presença de testemunhas é sempre recomendada e em muitos casos é uma exigência processual. De preferência, devem ser utilizadas

máquinas fotográficas com capacidade para salvar as fotos no formato TIFF, evitando a perda de qualidade causada pela compressão para o formato JPEG.

Se o computador estiver desligado, deve assim permanecer e deve ser levado para exames posteriores, em laboratório. A integridade das fotografias deve ser garantida por meio de uma função de *hash*, cujo resultado deve fazer parte da documentação, permitindo verificação posterior da autenticidade das fotografias. Todas as informações relevantes para futuros exames, tais como senhas, nomes de usuários ou peculiaridades da configuração dos sistemas, devem ser documentadas. Muitas vezes elas podem ser solicitadas ao responsável técnico pelo local. Se o computador estiver ligado, deve-se tirar uma fotografia da sua tela e providenciar algum mecanismo que o impeça de hibernar ou ativar a proteção de tela, pois pode estar protegido por senha e não ser possível voltar a ter acesso ao sistema.

Em seguida, é inserido algum dispositivo externo como *pen drive* ou disco rígido contendo os aplicativos automatizados de coleta de dados voláteis. O simples ato de conectar esse dispositivo gerará alteração no registro do Windows. Portanto, deve-se registrar os dados do dispositivo usado a fim de identificá-lo posteriormente em meio aos resultados. As ferramentas são, então, executadas para realizar a coleta de dados. Os procedimentos de coleta são apresentados na seção seguinte.

Os resultados devem ser armazenados no próprio dispositivo externo. Eles devem ser analisados em sequência para identificar a possível presença de criptografia de volumes, criptografia de sistema ou virtualização do sistema operacional, o que exige procedimentos específicos.

Após a coleta, o dispositivo deve ser inserido em um computador do examinador para os procedimentos finais. Caso tenham sido tiradas fotografias durante o procedimento, elas devem ser juntadas aos resultados da coleta. Deve-se aplicar, então, uma função de *hash* em cada arquivo e gravar todos os valores em um arquivo que acompanhará os demais.

Para fins de logística, sugere-se que todos os arquivos sejam transferidos para mídias óticas, como CD ou DVD. Considerando os tamanhos comuns de memória física, basta um DVD, na maioria dos casos, ou dois, em casos excepcionais. Dessa forma, o analista forense continua com o dispositivo de coleta e a mídia ótica pode acompanhar o trâmite do processo judicial sem qualquer prejuízo. No entanto, um processo adicional é necessário para garantir que não haja substituição dessa mídia ou adulteração do seu conteúdo. Para isso, deve-se calcular o *hash* do arquivo de *hashes* citado anteriormente. O resultado deve ser registrado no relatório da perícia, que conterá também a assinatura do analista. Com isso, é possível garantir a integridade e a autenticidade dos resultados encaminhados.

Nos casos em que houver necessidade de cópia lógica de arquivos para discos rígidos externos, devido ao maior volume de dados, o próprio disco rígido utilizado deve acompanhar o restante do material de informática relacionado apreendido. Novamente, o mesmo procedimento de cálculo de *hashes* dos arquivos coletados deve ser realizado.

6.12. Coleta de dados

A coleta de dados voláteis deve ser feita em uma sequência que parta dos dados mais voláteis para os menos voláteis. Os dados mais voláteis tendem a desaparecer mais rapidamente, tendo a preferência na ordem de coleta. Um exemplo de ordem de coleta, partindo dos dados mais voláteis para os menos voláteis, é apresentado a seguir:

- memória RAM;
- registro do Windows;
- estado da rede;
- processos em execução;
- volumes criptografados montados.

As ferramentas preferencialmente utilizadas para a coleta são as de linha de comando, que comprometem menos recursos da máquina alvo, já que a ferramenta utilizada vai ocupar parte da memória RAM do sistema alvo. Existem diversas ferramentas voltadas para resposta a incidentes e segurança de Tecnologia da Informação (TI). Um dos problemas de se utilizar essas ferramentas está no fato de que o usuário tem que lembrar todos os comandos e parâmetros para executar as ferramentas corretamente em linha de comando. Em seguida, o investigador terá que consolidar os resultados de forma a realizar seu relatório. Assim, para utilizar essas ferramentas em todo o seu potencial, é necessário agregá-las em um aplicativo que as execute de forma automática e na ordem correta, atendendo aos princípios forenses relacionados, e salvando os resultados de forma integrada e lógica em um arquivo, para análise posterior.

Existem algumas soluções integradas para o problema da coleta de dados voláteis no mercado. Entretanto, a maioria delas tem fins comerciais e não permite fácil atualização e adequação. As soluções encontradas, em geral, têm foco na segurança da informação e visam uma resposta imediata, enquanto o foco pericial é na preservação e/ou recuperação de dados, senhas e informações relevantes para a análise.

Devido à rápida evolução da computação forense, dos sistemas de informática e dos crimes relacionados, as ferramentas de análise forense não conseguem acompanhar no mesmo passo. Para cada novo sistema operacional, nova versão ou novo programa de roubo de dados desenvolvido, há necessidade de adequação, atualização e adaptação das ferramentas.

Perícia em sistemas ligados exige uma abordagem muito mais complexa e criteriosa do que o exame tradicional, com o sistema desligado. Deve haver extremo cuidado para minimizar o impacto das ferramentas utilizadas. Na avaliação de ferramentas para coleta de dados voláteis, devem ser considerados, entre outros aspectos:

- o total de memória alocada pela ferramenta;
- o impacto da ferramenta nos Registros do Windows;

- o impacto da ferramenta no sistema de arquivos;
- o uso de DLLs presentes no sistema.

É aconselhável que se capture todos os dados voláteis possíveis. A ordem de coleta pode ser crucial para a investigação. O examinador deve avaliar o caso cuidadosamente, para decidir a ordem de coleta, partindo dos dados mais voláteis para os menos voláteis. Os princípios fundamentais que norteiam a extração de dados são os seguintes:

- deve-se coletar todos os dados que serão perdidos ao desligar o sistema;
- deve-se coletar, primeiramente, os dados mais voláteis, deixando os menos voláteis para o final;
- os dados devem ser coletados no menor tempo possível e levando em conta a sua importância;
- os dados coletados devem permanecer disponíveis para futuras análises, se necessárias, e os exames realizados devem ser tão repetíveis quanto possível;
- deve-se manter a integridade dos dados coletados;
- as ferramentas de coleta devem capturar os dados de forma fidedigna;
- as ações realizadas devem ser pertinentes ao caso.

Cabe lembrar que a inserção de qualquer dispositivo em um computador ligado vai produzir pequenas alterações no sistema. O uso apropriado das ferramentas de captura de dados voláteis e a inserção destes dispositivos não adicionará nenhuma evidência ao sistema. Executar uma ferramenta capaz de realizar a coleta de memória RAM, por exemplo, vai necessitar de uma pequena porção da própria memória a ser capturada. Assim, a inserção de um dispositivo USB também vai adicionar uma entrada no registro do Windows. Todas essas pequenas alterações não produzem grandes consequências no sistema como um todo e podem ser explicadas posteriormente, com o exame minucioso e detalhado do material coletado. Essas pequenas alterações são produzidas pela interação das ferramentas com o sistema operacional do Windows, interferindo, apenas, nos arquivos do sistema operacional, não acarretando nenhuma mudança importante no conteúdo dos dados salvos no sistema. Ainda assim, é preciso enfatizar a necessidade de documentar ao máximo todo o processo.

Os dados voláteis incluem qualquer dado armazenado, na memória ou em trânsito, que será perdido com a interrupção da energia ou quando o sistema for desligado. Dados voláteis são encontrados em registradores, memória cache e memória RAM. Dados voláteis passíveis de coleta incluem:

- data e hora do sistema;
- usuários ativos e suas credenciais de autenticação;

- informação sobre processos em execução;
- informações dos registros do Windows;
- dispositivos conectados ao sistema;
- informações do sistema;
- conexões de rede;
- estado da rede;
- conteúdo da área de transferência;
- histórico de comandos;
- arquivos abertos.

Durante a coleta, se constatada a presença de criptografia de volumes, criptografia de sistema ou virtualização do sistema operacional, deve-se realizar a cópia lógica dos arquivos encontrados no volume criptografado ou a cópia de todo o sistema, dependendo do tipo de criptografia utilizada. Se o sistema estiver sendo executado em máquina virtual, também deve ser realizada cópia lógica de todo o sistema. A cópia lógica pode ser realizada com o programa FTK Imager, apresentado na seção seguinte, e o resultado deve ser direcionado a um disco rígido externo conectado a outra porta USB.

Ao final, deve ser verificado se há mais algum dado a copiar, como arquivos lógicos ou memória física. Caso contrário, o dispositivo externo de coleta deve ser retirado e o computador, desligado – retirando-se o cabo da fonte de alimentação. No caso de computadores portáteis, além de retirar o cabo da fonte de alimentação, deve-se retirar a bateria.

É importante ressaltar que o estado de um computador ligado não é estático, pois este encontra-se em funcionamento e em constante alteração de dados tanto de memória quanto de disco e processos. Assim, uma coleta de memória levará a resultados diferentes, a cada vez que for executada. Consequentemente, não há como executar uma função de *hash* de memória, pois os resultados serão sempre diferentes, quando se baseia no tempo. O que se recomenda, e deve ser feito, é um *hash* do arquivo resultante da coleta de memória. O resultado do *hash* desse arquivo deve ser documentado, buscando garantir a cadeia de custódia e evitar que haja questionamentos futuros. Agindo dessa forma, teremos um arquivo contendo a cópia da memória física, com garantia de integridade, possibilitando a repetição dos exames, caso necessário.

6.12.1. Ferramentas para coleta do *dump* de memória

MDD Utilitário desenvolvido pela Mantech International Corporation, para coleta da memória física (RAM) dos computadores ligados. Esse programa é disponibilizado para órgãos governamentais e uso privado, sob licença GPL e é capaz de coletar imagens de memória dos sistemas Windows 2000, Windows Server 2003, Windows XP, Windows Vista e Windows Server 2008 (MDD, 2011). A memória é coletada em formato binário

e a integridade do arquivo resultante é verificada pelo algoritmo MD5. Posteriormente, o arquivo coletado pode ser verificado em laboratório, com programas específicos para este fim como, por exemplo, o Volatility. O programa foi projetado especificamente para coletar uma imagem da memória física de sistema com até 4 GB de memória. O comando para realizar o *dump* da memória e gravá-lo em um arquivo é apresentado a seguir e ilustrado pela Figura 6.5.

```
mdd_1.3 -o arquivo_de_saída.img
```

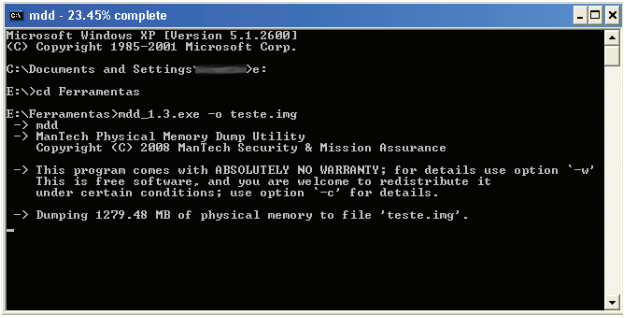

Figura 6.5. Tela de captura de memória do MDD

FTK Imager Utilitário fornecido gratuitamente pela AccessData. É capaz de criar *dumps* de memória em computadores de 32 e 64 bits. A Figura 6.6 apresenta a tela da ferramenta. Embora exista uma versão do FTK Imager para linha de comando, ainda não há suporte para captura de memória.

6.13. Ferramentas para análise indireta

Após realizar a captura da memória, é necessário analizar o conteúdo do *dump*, ou seja, realizar a análise indireta da memória. Duas ferramentas gratuitas para realizar esse procedimento são apresentadas a seguir. Como citado anteriormente, a abordagem de análise indireta da memória via arquivo de *dump* evita a alteração demasiada do estado da máquina, uma vez que substitui vários aplicativos de coleta de dados utilizados na análise direta da memória RAM. Além disso, a análise indireta permite a execução de novas consultas ou confirmações de resultados, o que não é possível utilizando a abordagem direta. Todavia, a análise do *dump* de memória depende fortemente da capacidade dos scripts em reconhecer a organização dos dados na memória RAM, sendo que cada sistema operacional, até mesmo em versões (*services packs*) diferentes, possuem formas distintas de armazenar dados em memória.

MANDIANT Memoryze A ferramenta pode realizar a captura da memória completa do sistema, da memória de um processo, DLL ou *driver* carregado na memória. Com base em um arquivo de *dump* ou a partir do conteúdo da memória em execução, ela pode:

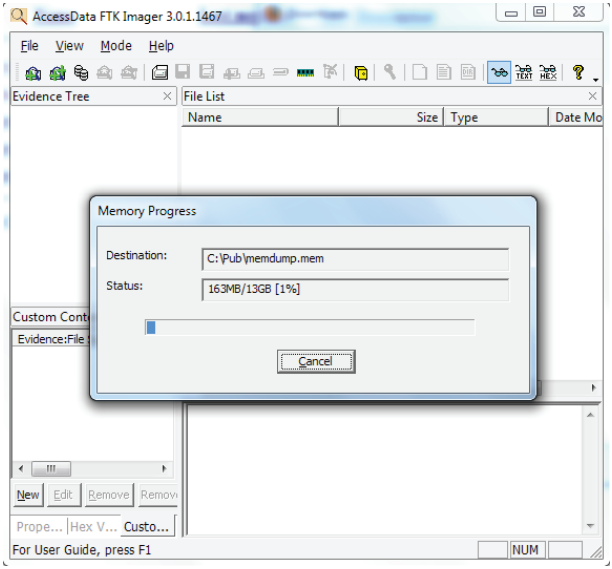

Figura 6.6. Tela de captura de memória do `FTK Imager`

- listar todos os processos em execução, incluindo *rootkits*;
- listar arquivos abertos e chaves de registro em uso;
- especificar funções importadas ou exportadas por executáveis e DLLs;
- listar *strings* em memória para cada processo;
- identificar *hooks*, comumente usados por *rootkits*.

A ferramenta recebe como entrada um arquivo XML com os parâmetros de execução, incluindo a localização do arquivo de *dump*, e gera relatórios em formato XML. Para facilitar seu uso, vários arquivos de lote (`.bat`) são fornecidos. Eles são listados a seguir:

- `MemoryDD.bat`: para obter a imagem da memória física;
- `ProcessDD.bat`: para obter a imagem do espaço de memória de um processo;
- `DriverDD.bat`: para obter a imagem de um *driver*;
- `Process.bat`: para enumerar tudas as informações de um processo incluindo *handles*, memória virtual, portas de rede e *strings*;
- `HookDetection.bat`: para procurar *hooks* no sistema operacional;

- `DriverSearch.bat`: para encontrar *drivers*;
- `DriverWalkList.bat`: para enumerar todos os módulos do *kernel* e *drivers*.

A Figura 6.7 apresenta um exemplo da análise das portas abertas por um processo e o estado da conexão. A saída em XML foi simplificada para facilitar a compreensão.

```
<?xml version="1.0" encoding="UTF-8"?>
<itemList><ProcessItem>
<pid>5400</pid>
<parentpid>5612</parentpid>
<path>Z:\thunderbird</path>
<name>thunderbird.exe</name>
<SecurityID>
S-1-5-21-3064900221-1125138106-471721996-1000
</SecurityID>
<SecurityType>SidTypeUser</SecurityType>
<startTime>2011-09-22T12:22:18Z</startTime>
<PortList>
<PortItem><pid>5400</pid>
    <protocol>TCP</protocol>
    <localPort>64058</localPort>
    <localIP>127.0.0.1</localIP>
    <remotePort>3128</remotePort>
    <remoteIP>10.2.0.50</remoteIP>
    <state>ESTABLISHED</state>
</PortItem>
    <PortItem><pid>5400</pid>
    <protocol>TCP</protocol>
    <localPort>58868</localPort>
    <localIP>127.0.0.1</localIP>
    <remotePort>58869</remotePort>
    <remoteIP>127.0.0.1</remoteIP>
    <state>ESTABLISHED</state>
</PortItem>
</PortList>
</ProcessItem></itemList>
```

Figura 6.7. Análise das portas abertas com a ferramenta `memoryze`

Volatility O *framework* Volatility consiste em um conjunto de *scripts* capazes de extrair informações de execução a partir da análise do arquivo de *dump* de memória do computador investigado. Essa ferramenta percorre o conteúdo do arquivo em busca de assinaturas e estruturas de dados dos elementos contidos em memória, tais como: processos em execução, arquivos abertos, conexões estabelecidas, portas abertas, entre outros. Uma lista completa de parâmetros do Volatility 1.3a é apresentada a seguir:

- `connections`: exibe a lista de conexões estabelecidas com endereço local, endereço remoto e o PID (*Process ID*);
- `connscan`: além da lista de conexões estabelecidas, exibe conexões escondidas e histórico de conexões, caso estejam ainda em memória;
- `connscan2`: exibe as mesmas informações da opção `connscan`;
- `datetime`: exibe hora e data do sistema em que foi feita a cópia da memória RAM;

- `dlllist`: lista as DLLs na memória referenciadas para cada processo;
- `dmp2raw`: converte um arquivo *dump* de falha de sistema em um arquivo de *dump* da memória;
- `dmpchk`: exibe as informações de um arquivo *dump* de falha;
- `files`: lista os arquivos abertos para cada processo;
- `hibinfo`: converte o arquivo de hibernação em arquivo de *dump* da memória;
- `ident`: exibe propriedades do arquivo de *dump* da memória, tais como: nome do arquivo, tipo do sistema operacional e hora de criação do arquivo;
- `memdmp`: extrai a memória endereçada de um processo;
- `memmap`: exibe o mapa de memória;
- `modscan`: busca módulos com os atributos de nome, base e tamanho;
- `modscan2`: exibe as mesmas informações da opção `modscan`;
- `modules`: exibe a lista com os módulos carregados;
- `procdump`: extrai o conteúdo de um processo para um arquivo executável;
- `pslist`: exibe a lista de processos em execução;
- `psscan`: busca por objetos `EPROCESS`;
- `psscan2`: exibe as mesmas informações da opção `psscan`;
- `regobjkeys`: lista as chaves de registro abertas para cada processo;
- `sockets`: lista as *sockets* abertos, contendo os atributos PID, porta, código do protocolo e data de criação;
- `sockscan`: busca por *sockets* abertos;
- `sockscan2`: exibe as mesmas informações da opção `sockscan`;
- `strings`: realiza a correspondência entre os deslocamentos físicos e os endereços virtuais;
- `thrdscan`: busca por objetos `ETHREAD`;
- `thrdscan2`: exibe as mesmas informações da opção `thrdscan`;
- `vaddump`: extrai as seções VAD (*Virtual Address Descriptors*) para um arquivo;
- `vadinfo`: extrai as informações VAD;
- `vadwalk`: percorre a árvore VAD.

A Figura 6.8 apresenta a listagem de processos em execução obtida com a opção `pslist`. É importante notar a presença de dois processos na lista abaixo: `cmd.exe` e `mdd_1.3.exe`, que são produtos do processo de coleta. Além deles, deve-se destacar o uso de criptografia (`truecrypt.exe`) e de máquinas virtuais (`VMWareUser.exe`), que podem ser ameaças ao exame pericial e exigem uma coleta mais cuidadosa.

Name	Pid	PPid	Thds	Hnds	Time
System	4	0	61	261	Thu Jan 01 00:00:00 1970
smss.exe	552	4	3	19	Fri Mar 18 14:23:29 2011
csrss.exe	616	552	12	378	Fri Mar 18 14:23:33 2011
winlogon.exe	640	552	21	644	Fri Mar 18 14:23:34 2011
services.exe	684	640	17	344	Fri Mar 18 14:23:35 2011
lsass.exe	696	640	19	346	Fri Mar 18 14:23:35 2011
vmacthlp.exe	852	684	1	25	Fri Mar 18 14:23:37 2011
svchost.exe	880	684	17	193	Fri Mar 18 14:23:37 2011
svchost.exe	964	684	9	265	Fri Mar 18 14:23:38 2011
svchost.exe	1060	684	71	1414	Fri Mar 18 14:23:39 2011
svchost.exe	1128	684	4	74	Fri Mar 18 14:23:39 2011
svchost.exe	1308	684	15	203	Fri Mar 18 14:23:40 2011
spoolsv.exe	1512	684	12	134	Fri Mar 18 14:23:43 2011
explorer.exe	1612	1596	10	357	Fri Mar 18 14:23:43 2011
VMwareTray.exe	1956	1612	1	76	Fri Mar 18 14:23:46 2011
VMwareUser.exe	1980	1612	6	192	Fri Mar 18 14:23:46 2011
vmtoolsd.exe	1716	684	4	225	Fri Mar 18 14:23:59 2011
VMUpgradeHelper	2036	684	3	98	Fri Mar 18 14:23:56 2011
TPAutoConnSvc.e	324	684	5	99	Fri Mar 18 14:23:58 2011
alg.exe	1172	684	6	108	Fri Mar 18 14:24:00 2011
wscntfy.exe	1456	1060	1	28	Fri Mar 18 14:24:01 2011
TPAutoConnect.e	172	324	1	60	Fri Mar 18 14:24:02 2011
wuauclt.exe	952	1060	3	131	Tue Mar 22 11:45:09 2011
TrueCrypt.exe	216	1612	2	108	Tue Mar 22 15:30:55 2011
cmd.exe	1460	1612	1	32	Tue Mar 22 15:36:17 2011
mdd_1.3.exe	256	1460	1	25	Tue Mar 22 15:36:53 2011

Figura 6.8. Listagem de processos exe execução com o `volatility`

6.14. Ferramentas para análise direta

Esta seção apresenta um conjunto de ferramentas e comandos do sistema operacional para realizar a coleta de dados com o sistema em execução. Essa abordagem se opõe àquela da análise do *dump* de memória. Cabe lembrar que a execução dessas ferramentas, invariavelmente, gerará alterações na memória do sistema e seu uso deve ser documentado detalhadamente.

6.14.1. Comandos do sistema operacional

Os sistemas Windows disponibilizam alguns comandos que podem ser utilizados para diagnosticar e resolver problemas do computador, além de servirem para coletar dados de interesse. Alguns comandos úteis aos procedimentos de coleta de dados são: `cmd`, `time`, `date`, `echo`, `ipconfig`, `netstat` e `tasklist`. Parte desses comandos exige privilégios de administrador do sistema para serem executados. Para a coleta de dados, não é aconselhável que se utilize o `cmd.exe` do sistema investigado, que pode responder de forma imprevisível por estar comprometido por algum *rootkit* instalado na máquina. O kit de coleta de dados utilizado pelo analista deve conter uma cópia do `cmd.exe` sabidamente confiável, para que os comandos ali executados tenham a resposta correta e esperada.

Regedit O editor de registros do Windows possibilita a realização de cópia das chaves da máquina local e do usuário atual com o comando a seguir. O resultado é gravado em

formato texto no arquivo fornecido como parâmetro. A Figura 6.9 apresenta a saída do comando.

```
regedit /e registro.txt
```

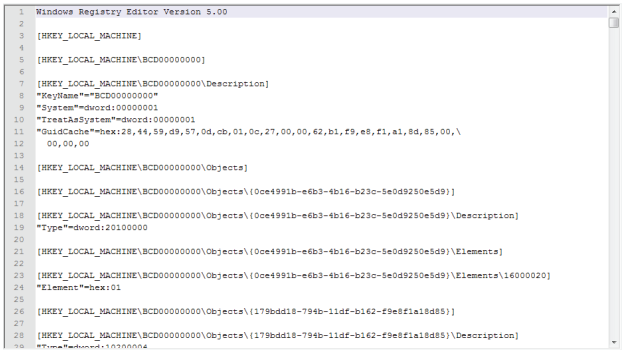

Figura 6.9. Saída do comando `regedit /e`

Outros comandos de sistema do Windows – nome do computador, nome do usuário da sessão, data e hora do sistema e lista de arquivos recentemente abertos pelo usuário da sessão – podem ser executados, conforme especificado a seguir. Todos os resultados são adicionados ao `arquivo_de_saída.txt`.

```
echo %COMPUTERNAME% > arquivo_de_saída.txt
echo %USERNAME% > arquivo_de_saída.txt
dir "%UserProfile%\Recent" > arquivo_de_saída.txt
date /t > arquivo_de_saída.txt
time /t > arquivo_de_saída.txt
```

A lista de processos e serviços em execução pode ser obtida com o comando `tasklist`. A Figura 6.10 apresenta a saída do comando.

```
tasklist /V
```

O comando `netstat` fornece informações sobre conexões de rede, como as portas abertas por processos e as conexões estabelecidas. Um exemplo do comando é exibido na Figura 6.11.

```
netstat -ano
```

Informações sobre as interfaces de rede, incluindo endereços físicos e endereços IP em uso, podem ser obtidas com o uso do comando `ipconfig`. A saída é exibida na Figura 6.12.

```
ipconfig /all
```

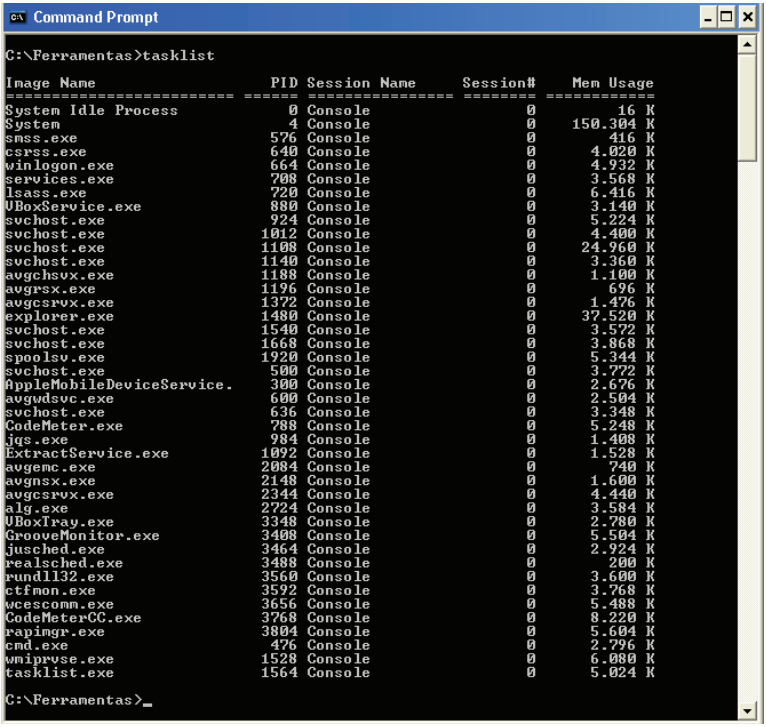

Figura 6.10. Saída do comando `tasklist /V`

O comando a seguir apresenta uma lista dos comandos recentemente utilizados pelo usuário. A Figura 6.13 ilustra o comando.

```
doskey /history > arquivo_de_saída.txt
```

6.14.2. Informações do sistema

Esta seção apresenta utilitários que obtém informações do sistema em execução como arquivos aberto, DLLs carregadas, usuários registrados, data de instalação, dentre outras.

Handle v3.45 Utilitário que mostra a relação de processos com arquivos e pastas abertos. Pode ser executado em sistemas Windows XP ou mais recentes. Necessita ser executado por usuário com poderes de administrador do sistema.

```
handle -accepteula > arquivo_de_saída.txt
```

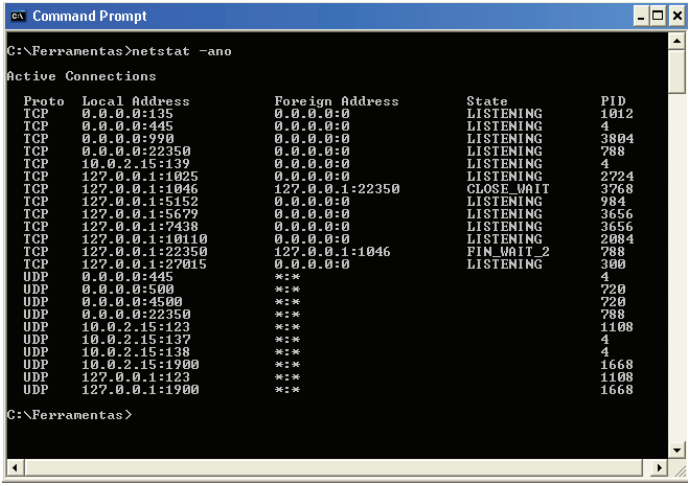

Figura 6.11. Exemplo do comando `netstat`

ListDLLs v3.0 Este utilitário relaciona as DLLs carregadas no sistema. Pode ser executado em sistemas Windows XP ou mais recentes, retornando o nome completo dos módulos carregados. Além disso, sinaliza as DLLs que apresentam número de versão diferente dos seus arquivos correspondentes gravados em disco (isso ocorre quando um arquivo é atualizado depois que o programa carrega suas DLLs), podendo ainda informar quais DLLs foram realocadas.

```
listdlls
```

PsFile v1.02 Este utilitário de linha de comando mostra os arquivos que foram abertos remotamente. Pode ser executado em sistemas Windows XP ou mais recentes.

```
psfile -accepteula
```

PsInfo v1.77 Ferramenta de linha de comando que retorna informações importantes sobre o sistema, incluindo tipo de instalação, usuário registrado, organização, número e tipo de processador, quantidade de memória física, data de instalação do sistema, entre outras. Pode ser executado em sistemas Windows XP ou mais recentes.

```
psinfo -accepteula
```

PsList v1.29 Ferramenta de linha de comando que lista os processos em execução. Pode ser executado em sistemas Windows XP ou mais recentes.

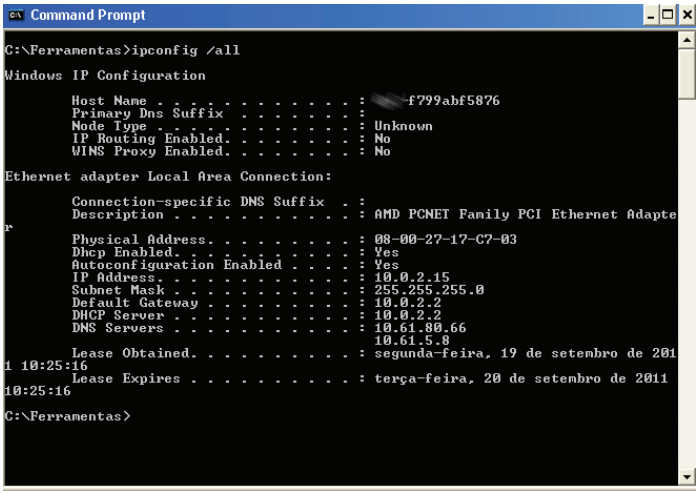

Figura 6.12. Exemplo do comando `ipconfig /all`

```
pslist -accepteula
```

PsLoggedOn v1.34 Este utilitário permite determinar quem está utilizando ativamente o sistema, seja localmente ou remotamente. Pode ser executado em sistemas Windows XP ou mais recentes.

```
psloggedon -accepteula
```

pclip Copia o conteúdo da área de transferência.

```
pclip > arquivo_de_saída.txt
```

6.14.3. Navegadores de Internet

Esta seção apresenta utilitários que extraem informações dos navegadores de Internet como histórico de navegação, senhas gravadas, arquivos temporários do *cache* e buscas realizadas.

IEHistoryView v1.61 Utilitário que permite visualizar as páginas acessadas com o navegador Internet Explorer.

```
iehv /stext arquivo_de_saída.txt
```

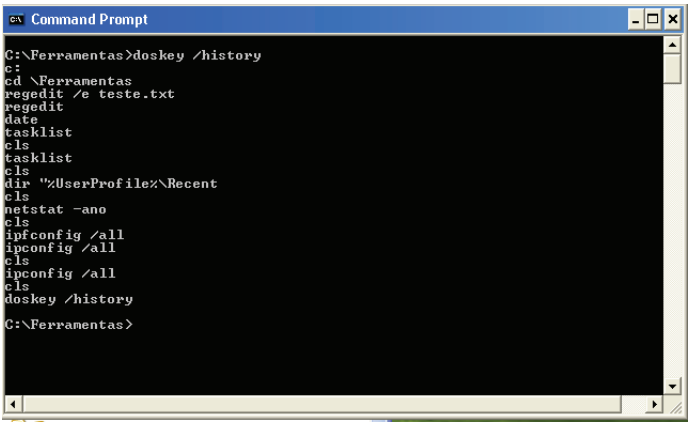

Figura 6.13. Exemplo do comando `doskey /history`

MozillaHistoryView v1.35 Programa utilitário que permite visualizar as páginas acessadas com o navegador Mozilla Firefox.

```
mozillahistoryview /stext arquivo_de_saída.txt
```

ChromeHistoryView v1.00 Utilitário que permite visualizar as páginas acessadas com o navegador Google Chrome. A Figura 6.16 apresenta um exemplo da saída na tela.

```
chromehistoryview /stext arquivo_de_saída.txt
```

IE PassView v1.26 Utilitário que revela as senhas armazenadas pelo navegador Internet Explorer, suportando desde a versão 4.0 até a 9.0.

```
iepv /stext arquivo_de_saída.txt
```

`ChromePass v1.20` Utilitário que revela as senhas armazenadas pelo navegador Google Chrome.

```
ChromePass /stext arquivo_de_saída.txt
```

PasswordFox v1.30 Utilitário que revela as senhas armazenadas pelo navegador Mozilla Firefox, suportando qualquer versão do Windows 2000, XP, Server 2003, Vista, até o Windows 7. Caso uma senha mestre esteja sendo utilizada para proteger as senhas, ela pode ser especificada pelo parâmetro `/master`. Caso ela seja desconhecida, não será possível recuperar as senhas. Ao contrário do Google Chrome, o Firefox permite armazenar senhas incorretas. A Figura 6.17 exibe a tela do aplicativo.

Figura 6.14. Saída do aplicativo `handle`

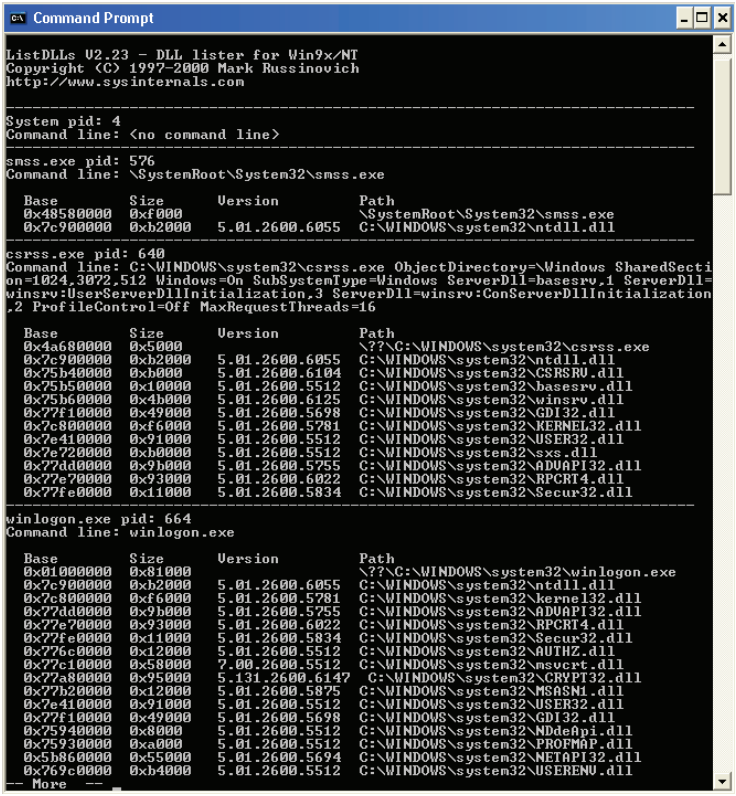

Figura 6.15. Saída do aplicativo `ListDLLs`

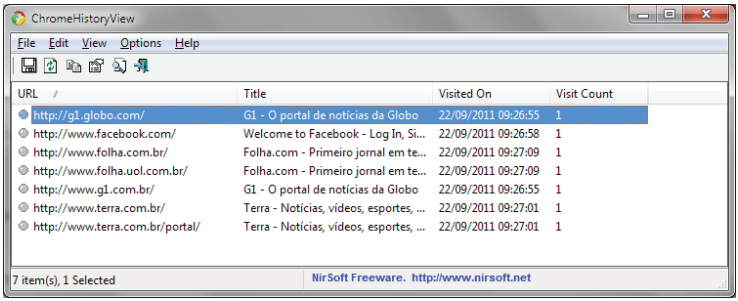

Figura 6.16. Tela do aplicativo `ChromeHistoryView`

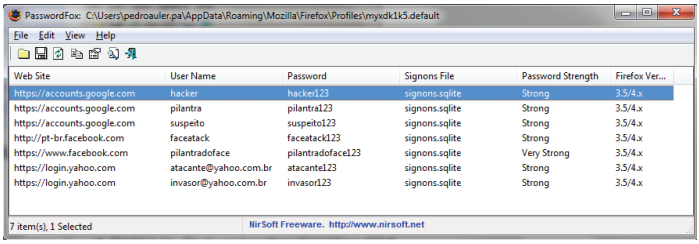

Figura 6.17. Tela do aplicativo `PasswordFox`

```
passwordfox /stext arquivo_de_saída.txt
```

MyLastSearch v1.50 Utilitário que permite para visualizar os últimos termos pesquisados em diversos programas de busca (Google, Yahoo e MSN) e em sites de redes sociais (Twitter, Facebook, MySpace). A Figura 6.18 ilustra os resultados obtidos com esta ferramenta. No caso de buscar realizadas com o recurso de resultados instantâneos, como no navegador Google Chrome, uma busca é registrada para cada letra digitada.

```
mylastsearch /stext arquivo_de_saída.txt
```

6.14.4. Mensageria e comunicação

Utilitários que extraem informações de aplicativos de mensageria e comunicação são apresentados nesta seção, incluindo registros de conversas, ligação utilizando VoIP e senhas gravadas.

SkypeLogView v1.21 Este utilitário acessa os arquivos de log criados pelo Skype e mostra detalhes, como chamadas realizadas ou recebidas, mensagens de chat e transferências de arquivos. A Figura 6.19 ilustra os resultados obtidos com essa ferramenta.

```
skypelogview /stext arquivo_de_saída.txt
```

MessenPass v1.42 Utilitário que permite a recuperação de senhas do usuário atualmente ativo no sistema e somente funciona se o usuário configurar o programa para salvar as senhas utilizadas. Os programas de mensagem instantânea suportados são os seguintes: MSN Messenger, Windows Messenger (em Windows XP), Windows Live Messenger (em Windows XP/Vista/7), Yahoo Messenger (versões 5.x e 6.x), Google Talk, ICQ Lite (versões 4.x/5.x/2003), AOL Instant Messenger (versão 4.6 ou abaixo, AIM 6.x e AIM Pro), Trillian, Trillian Astra, Miranda, GAIM/Pidgin, MySpace IM, PaltalkScene e Digsby.

```
mspass /stext arquivo_de_saída.txt
```

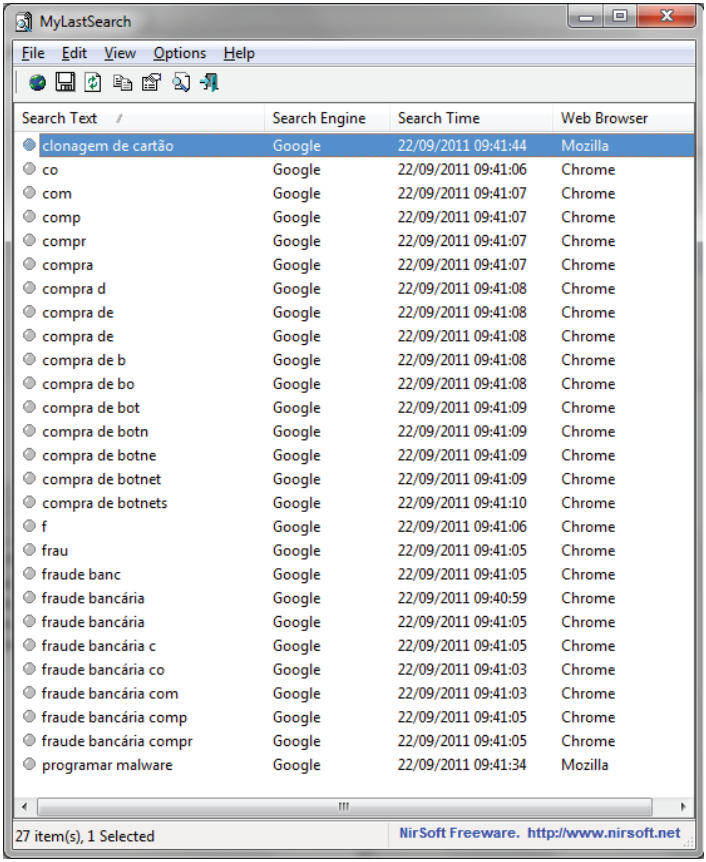

Figura 6.18. Tela do aplicativo `MyLastSearch`

Mail PassView v1.73 Este utilitário pode ser executado em qualquer versão do Windows, desde a versão 98 até o Windows 7, e permite recuperar senhas de programas de e-mail, tais como: Outlook Express, Microsoft Outlook 2000 (POP3 e SMTP), Microsoft Oulook 2002/2003/2007/2010 (POP3, IMAP, HTTP e SMTP), Windows Mail, Windows Live Mail, IncrediMail, Eudora, Netscape 6.x/7.x (se a senha não estiver criptografada com senha mestre), Mozilla Thunderbird (se a senha não estiver criptografada com senha mestre), Group Mail Free, Yahoo! Mail (se a senha estiver salva em alguma aplicação do Yahoo! Messenger) e Gmail (se a senha estiver salva na aplicação Gmail Notifier, Google Desktop ou Google Talk).

```
mailpv /stext arquivo_de_saída.txt
```

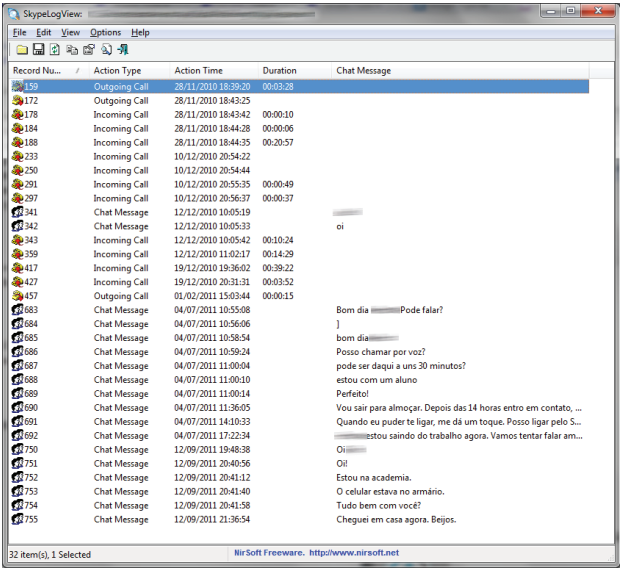

Figura 6.19. Tela do aplicativo `SkypeLogView`

6.14.5. Dispositivos

Informações sobre dispositivos USB, conexões sem fio, volumes criptografados e máquinas virtuais podem ser extraídas com o auxílio dos utilitários apresentados nesta seção.

WirelessKeyView v1.35 Utilitário que recupera as senhas de internet sem fio (WEP/WPA) armazenadas no computador, em sistemas Windows XP e Vista.

```
wirelesskeyview /stext arquivo_de_saída.txt
```

USBDeview v1.89 Utilitário que lista os dispositivos USB conectados ao computador, bem como aqueles que estiveram conectados recentemente.

```
usbdeview /stext arquivo_de_saída.txt
```

Encrypted Disk Detector (EDD) 1.2.0 Ferramenta de linha de comando que verifica a presença de volumes criptografados para os programas TrueCrypt, PGP, Safeboot (McAfee Endpoint Encryption) e Bitlocker. O programa foi testado pelo desenvolvedor em sistemas Windows XP, Vista e 7 (32 e 64 bits), necessitando de apenas 40 KB de espaço em disco e, aproximadamente, 3 MB de memória. Atualmente, o programa é disponi-

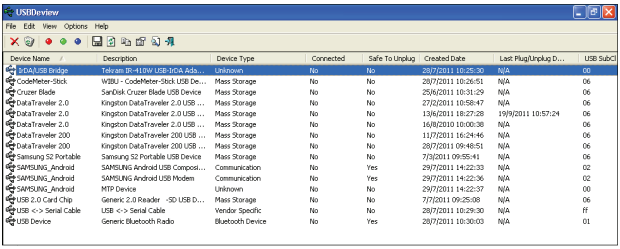

Figura 6.20. Tela do aplicativo `usbdeview`

bilizado gratuitamente. A Figura 6.21 apresenta um exemplo em que não foi detectado qualquer volume criptografado.

```
edd120.exe /batch arquivo_de_saída.txt
```

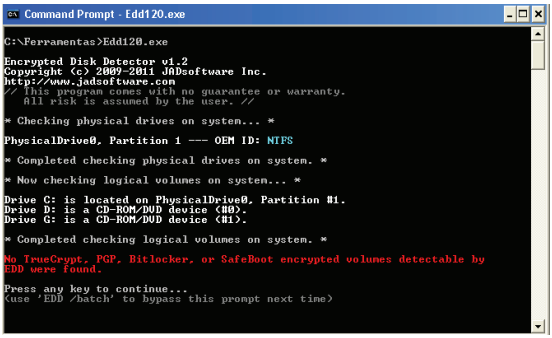

Figura 6.21. Saída do aplicativo `EDD`

ScoopyNG v1.0 Este utilitário combina as técnicas de outras duas ferramentas mais antigas, ScoobyDoo e Jerry, e incorpora algumas novas técnicas para determinar se o sistema operacional corrente está sendo executado dentro de uma máquina virtual VMware ou em um sistema nativo. O aplicativo funciona em qualquer CPU moderna, independentemente do número de processadores utilizados. Além disso, é capaz de detectar a presença do VMware mesmo quando utilizados mecanismos anti-detecção.

```
scoopyng > arquivo_de_saída.txt
```

6.14.6. Ferramentas auxiliares

As ferramentas apresentadas nesta seção auxiliam o trabalho pericial na manipulação dos resultados gerados pelas demais ferramentas.

strings v2.41 Utilitário que permite a extração de caracteres UNICODE (ou ASCII) em arquivos, incluindo arquivos executáveis e DLLs.

```
strings -accepteula arquivo_de_entrada
```

grep Este é um aplicativo de linha de comando proveniente de sistemas Unix/Linux, capaz de fazer buscas no conteúdo de arquivos ou saídas de outros comandos executados, estando também disponível para ambiente Windows. O quadro a seguir apresenta dois exemplos de utilização. No primeiro, é realizada a procura pela palavra "truecrypt"nos processos listados pelo comando `pslist`. A opção `-i` faz com que a diferença entre letras maiúsculas e minúsculas seja ignorada. No segundo, é feita a procura pela palavra "truecrypt"no arquivo `teste.txt`.

```
pslist | grep -i truecrypt
grep -i truecrypt teste.txt
```

fsum v2.52 Utilitário de linha de comando para verificar a integridade de arquivos. Permite a escolha entre várias funções de *hash*. O quadro a seguir apresenta três exemplos. No primeiro, a ferramenta é executada sobre a pasta "Resultados", de forma recursiva (`-r`), e a saída é gravada no arquivo `hashes.txt`. No segundo, a ferramenta calcula o *hash* do arquivo `hashes.txt`, salvando o resultado em `hash_do_hashes.txt`. No terceiro, é pedida a verificação (`-c`) dos *hashes* contidos no arquivo `hashes.txt`, mostrando apenas eventuais falhas de verificação na tela (`-jf`).

```
fsum -d".\Resultados" -r -sha256 * > hashes.txt
fsum -d"." hashes.txt > hash_do_hashes.txt
fsum -jf -c hashes.txt
```

Entre as ferramentas citadas, as distribuídas pela empresa Nirsoft são as seguintes: IEHistoryView v1.60, MozillaHistoryView v1.31, MyLastSearch v1.50, SkypeLogView v1.21, MessenPass v1.41, Mail PassView v1.72, IE PassView v1.26. Todas elas podem ser executadas em linha de comando sem utilização de interface gráfica, podendo o resultado da pesquisa ser direcionado para um arquivo de texto (`/stext arquivo.txt`), para um arquivo HTML (`/shtml arquivo.html`), para um arquivo XML (`/sxml arquivo.xml`) ou outros formatos, que podem ser consultados no site da Nirsoft. Os utilitários são distribuídos gratuitamente, podendo ser utilizados livremente para uso particular ou empresarial, desde que não haja fins lucrativos ou cobranças de qualquer natureza para recuperar senhas de eventuais clientes, a não ser com autorização expressa dos autores do *software*. Os aplicativos podem ser livremente distribuídos, desde que todos os arquivos do pacote sejam incluídos sem qualquer modificação e que não haja nenhum tipo de cobrança financeira.

Algumas ferramentas de linha de comando da Microsoft Sysinternals costumam retornar uma janela perguntando se o usuário aceita as condições da licença de uso. O inconveniente dessa janela é que a coleta automatizada fica interrompida até que o usuário aceite ou não a licença do *software*. Para evitar este inconveniente em uma ferramenta

automatizada de linha de comando, em que os resultados são dirigidos a um arquivo para análise posterior, deve ser utilizada a opção `-accepteula` logo após o comando, aceitando, assim, as condições da licença e evitando o aparecimento da janela.

Algumas vezes, os aplicativos utilizados para fins periciais são classificados incorretamente como vírus ou cavalos-de-troia pelos antivírus eventualmente instalados na máquina em análise. Isso devido ao fato de algumas das ferramentas recuperarem informações sensíveis, como senhas, chaves de instalação e registros do Windows. Logo, esse comportamento é considerado suspeito pelos antivírus. Assim, é recomendável que se desabilite qualquer antivírus instalado antes de iniciar a coleta dos dados, quando possível.

6.15. Considerações finais

A análise de memória RAM tem ser tornado parte importante de qualquer investigação forense computacional, permitindo o acesso aos dados voláteis não encontrados em uma imagem de disco rígido. Apesar dos recentes progressos da análise de memória, as dificuldades ainda são grandes, devido à falta de flexibilidade das ferramentas existentes, que geralmente só podem ser utilizadas nos sistemas operacionais específicos e respectivas versões para os quais foram codificadas.

Como citado anteriormente, a abordagem forense tradicional consiste em retirar o cabo de energia da máquina suspeita, para analisar os dados presentes na mídia de armazenamento posteriormente, em laboratório. Esta técnica pode levar à perda de importantes evidências presentes nos dados voláteis, devido ao crescente uso de criptografia de disco e de sistema. No caso de utilização de criptografia, principalmente de disco, conseguir acesso aos dados com o sistema ainda ligado é de vital importância. Além disso, há uma forte tendência a se utilizar armazenamento remoto de dados, em servidores remotos, por meio de conexões de rede ou Internet. Neste caso, o desligamento precoce do sistema pode inviabilizar a coleta de dados remotos que estão acessíveis somente naquele momento.

Um dos princípios mais importantes da perícia é o *Princípio da Troca de Locard*, que, adaptado à Informática Forense, afirma que ocorrem mudanças em um sistema de informática ativo, simplesmente pela passagem do tempo. Isso ocorre devido aos processos que estão em execução, aos dados que estão sendo gravados na memória ou apagados, às conexões de rede sendo criadas ou finalizadas, e assim por diante. Se as mudanças ocorrem simplesmente pela passagem do tempo, são agravadas quando o investigador executa seus programas de coleta de dados. Afinal, a execução de ferramentas no sistema provoca o seu carregamento na memória RAM, sobrescrevendo outros dados ali presentes.

A coleta de dados em computadores ligados deve ser realizada com impacto mínimo sobre a integridade do sistema. Há pouco tempo, os resultados obtidos dessa forma não eram bem aceitos na Justiça, devido à interferência do analista forense no sistema original. Entretanto, não há mais como fugir dessas técnicas, para que não haja perda definitiva de informações valiosas. A cadeia de custódia deve ser criteriosamente documentada, com a ajuda de *hashes* do material coletado, utilização de fotografias e filmagens e presença de testemunhas durante a coleta. Pequenas interferências no sistema podem e devem ser aceitas, desde que bem documentadas, com o objetivo maior de preservar a informação vital para a investigação e para o processo judicial.

A computação forense é uma área que evolui rapidamente. Como consequência, os crimes de informática também evoluem na mesma proporção. Mais ainda, os sistemas de informática evoluem em velocidade superior à das ferramentas de análise desenvolvidas. Apesar da intensa pesquisa realizada nos últimos anos, a captura e a análise da memória física em sistemas operacionais baseados em Windows ainda está em um estágio inicial de compreensão e desenvolvimento e ainda não existe uma técnica totalmente eficiente. Apesar disso, a análise *live* é capaz de recuperar informações valiosas que, de outra maneira, seriam perdidas se fosse utilizada a técnica de retirar o cabo de energia. É altamente recomendável que a análise tradicional, baseada em disco rígido, seja complementada com a análise *live*, que está se tornando cada vez mais importante e, em alguns casos, determinante, na medida em que as ferramentas de captura e análise se tornam mais sofisticadas e eficientes.

Referências

Adelstein, F. (2006). Diagnosing your system without killing it first. *Communications of the ACM*, 49:63–66.

Anson, S. and Bunting, S. (2007). *Mastering Windows Network Forensics and Investigation*. Sybex.

Aquilina, J. M., Casey, E., and Malin, C. H. (2008). *Malware Forensics - Investigating and Analyzing Malicious Code*. Syngress.

Beebe, N. L. and Clark, J. G. (2005). A hierarchical, objectives-based framework for the digital investigations process. *Digital Investigation*, 2(2):147–167.

Carrier, B. (2006). Risks of live digital forensic analysis. *Communications of the ACM*, 49:56–61.

Carrier, B. D. and Spafford, E. H. (2005). Automated digital evidence target definition using outlier analysis and existing evidence. In *Digital Forensic Research Workshop (DFRWS)*.

FTK Imager (2011). <http://accessdata.com/support/adownloads>. Último acesso em 22/09/2011.

Hay, B., Nance, K., and Bishop, M. (2009). Live analysis: Progress and challenges. digital forensics. *IEEE Security and Privacy*, 7:30–7.

Hoelz, B. W. P. (2009). Madik: Uma abordagem multiagente para o exame pericial de sistemas computacionais. Master's thesis, Universidade de Brasília, Brasília.

Huebner, E., Bem, D., and Bem, O. (2003). Computer Forensics: Past, Present And Future. *Information Security Technical Report*, 8(2):32–36.

Mandia, K., Prosise, C., and Pepe, M. (2003). *Incident Response & Computer Forensics*. McGraw-Hill, 2nd edition.

MANDIANT Memoryze (2011). <http://www.mandiant.com/products/free_software/memoryze>. Último acesso em 22/09/2011.

MDD (2011). <http://sourceforge.net/projects/mdd>. Último acesso em 22/09/2011.

Microsoft Sysinternals (2011). <http://technet.microsoft.com/sysinternals>. Último acesso em 22/09/2011.

NetMarketShare (2011). <http://www.netmarketshare.com/>. Último acesso em 23/09/2011.

Nirsoft (2011). <http://www.nirsoft.com/>. Último acesso em 22/09/2011.

Palmer, G. (2001). A Road Map for Digital Forensic Research. Technical Report DTR - T001-01 FINAL, DFRWS. Report from the First Digital Forensic Research Workshop (DFRWS).

Reith, M., Carr, C., and Gunsch, G. (2002). An Examination of Digital Forensic Models. *International Journal of Digital Evidence*, 1(3).

Rogers, M. K., Mislan, R., Goldman, J., Wedge, T., and Debrota, S. (2006). Computer forensics field triage process model. In *Conference on Digital Forensics, Security and Law*, pages 27–40.

Ruibin, G. and Gaertner, M. (2005). Case-Relevance Information Investigation: Binding Computer Intelligence to the Current Computer Forensic Framework. *International Journal of Digital Evidence*, 4(1).

Turner, P. (2005). Digital provenance - interpretation, verification and corroboration. *Digital Investigation*, 2:45–49.

Volatility (2011). <https://www.volatilesystems.com/default/volatility. Último acesso em 22/09/2011.

PROMOÇÃO

REALIZAÇÃO

PATROCÍNIO