Raphael Campos Silva

Malflow: um *framework* para geração automatizada de assinaturas de *malwares* baseado em fluxo de dados de rede

São José do Rio Preto
2017

Raphael Campos Silva

Malflow: um *framework* para geração automatizada de assinaturas de *malwares* baseado em fluxo de dados de rede

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Campus de São José do Rio Preto.

Financiadora: CAPES

Orientador: Prof. Dr. Adriano Mauro Cansian

São José do Rio Preto
2017

Silva, Raphael Campos.
Malflow : um framework para geração automatizada de assinaturas de malwares baseado em fluxo de dados de rede / Raphael Campos Silva. -- São José do Rio Preto, 2017
103 f. : il., tabs.

Orientador: Adriano Mauro Cansian
Dissertação (mestrado) – Universidade Estadual Paulista “Júlio de Mesquita Filho”, Instituto de Biociências, Letras e Ciências Exatas

1. Computação. 2. Redes de computadores - Medidas de segurança. 3. Sistemas de detecção de intrusão (Medidas de segurança) 4. Malware (Software de computador) 5. Algoritmos de computador. 6. Assinaturas digitais. I. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. II. Título.

CDU – 681.3.025

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

Raphael Campos Silva

Malflow: um *framework* para geração automatizada de assinaturas de *malwares* baseado em fluxo de dados de rede

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

Comissão Examinadora

Prof. Dr. Adriano Mauro Cansian
UNESP – São José do Rio Preto
Orientador

Prof. Dr. André Ricardo Abed Grégio
UFPR – Curitiba

Prof. Dr. Geraldo Francisco Donega Zafalon
UNESP – São José do Rio Preto

São José do Rio Preto
23 de janeiro de 2017

Dedico este trabalho

Aos meus pais, Ademir e Sandra e os meus irmãos Ricardo e Christina, pelo incentivo e amor durante todos os tempos.

AGRADECIMENTOS

Agradeço primeiramente a minha família pelo amor, ensinamento e suporte durante todos estes anos.

Ao meu orientador, Prof. Dr. Adriano Mauro Cansian, pela orientação, paciência e puxadas de orelha, que contribuíram significativamente para minha formação pessoal e profissional.

Ao André Ricardo Abed Grégio pelas discussões ao longo do projeto. Ainda, agradeço ao Geraldo Francisco Donegá Zafalon pela ajuda prestada no artigo.

Aos meus amigos Vinícius Vassoler Galhardi e Vinícius Oliveira Ferreira, pela troca de conhecimento realizada durante toda essa jornada que passamos juntos.

A todos os colegas de laboratório que estiveram presente em minha vida durante esses últimos anos: Adriano Ribeiro, Amanda Barbosa, Bruno Leal, Bruno Rogério, Leandro Gonçalves, Matheus Andrade, Pedro Ferracini e Rafael Carreira pelas horas de estudo e longas risadas durante o almoço. Ainda, agradeço os antigos membros do laboratório pelos conselhos e pelos grandes conhecimentos que foram passados.

Ao PPGCC – Programa de Pós Graduação em Ciência da Computação e a todos os docentes do Departamento de Ciências da Computação e Estatística (DCCE) pelas disciplinas e conhecimentos transmitidos durante minha formação.

Não poderia deixar de agradecer as pessoas que cruzaram meu caminho ao longo desses anos, e aos *hackers* que conheci e que se tornaram uma grande fonte de inspiração: Rodrigo Rubira Branco, Gabriel Negreira Barbosa e Ygor Da Rocha Parreira.

Por fim, agradeço à CAPES pela bolsa de mestrado concedida para realização deste projeto.

“Há uma força motriz mais poderosa que o vapor, a eletricidade e a energia atômica: a vontade.”
Albert Einstein

RESUMO

A garantia de segurança em ambientes computacionais é complexa, uma vez que a expertise dos atacantes e o número de ameaças têm aumentado. De forma a lidar com o aumento de incidentes de segurança, é necessária uma metodologia que automatize o processo de análise de ameaças e forneça assinaturas de ataques para os ambientes de defesa. Este projeto propõe uma metodologia para criação automatizada de assinaturas para *malware* baseado em fluxo de dados de redes. A partir de múltiplas execuções de uma amostra de *malware*, são encontradas semelhanças entre o tráfego de rede gerado em cada uma de suas execuções. O processo de encontrar semelhanças baseia-se em: (i) geração de um *hash* para cada uma das conexões realizadas pelo *malware*, no qual cada *hash* irá representar um elemento de uma sequência e (ii) utilização do algoritmo LCS para encontrar uma subsequência em comum mais longa entre duas sequências geradas a partir das conexões realizadas pelo *malware* durante cada uma de suas execuções. Uma vez encontrada a subsequência em comum mais longa, os descritores das conexões realizadas pelo *malware* são recuperados, os quais irão compor os passos de uma assinatura. Por fim, as assinaturas geradas serão testadas para identificação de falsos-positivos e verdadeiros-positivos, para que sejam selecionadas com o intuito de alimentar um Sistema de Detecção de Intrusão.

Palavras-chave: *Malware*. Análise dinâmica. Mecanismo de geração de assinatura. Geração automatizada de assinatura. Sistemas de Detecção de Intrusão baseados em Rede.

ABSTRACT

The guarantee of security in computing environments is complex, since the expertise of the attackers and the numbers of threats have increased. In order to handle the increased security incidents, is required a methodology to automate the process of threat analysis and provide signatures to defense environments. This project proposes a methodology to generate signatures automatically, based on network flows. From multiple execution of a malware sample, similarities are found between the network traffic generated in each of its executions. The process of finding similarity is based on: (i) Generation of a hash for each connection performed by the malware, where each hash will represent an element of a sequence and (ii) application of the LCS algorithm to find the longest common subsequence between two sequences generated from the connections performed by the malware during each of its executions. Once the longest common subsequence is found, the descriptors of the connections performed by the malware are retrieved, which will compose the steps of a signature. Finally, the generated signatures will be tested for false positive and true positive identification, so that they are selected with the intention of feeding an Intrusion Detection System.

Keywords: Malware. Dynamic Analysis. Signature Generation Mechanism. Automated Signature Generation. Network Intrusion Detection Systems.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API: Application Programming Interface

Cert.br: Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil

CSRF: Cross-site request forgery

CR: carriage return

C&C: Command-and-Control

DDoS: Distributed Denial of Service

DLL: Dynamic-link library

DNS: Domain Name System

DoS: Denial of Service

FN: False Negative

FP: False Positive

FTP: File Transfer Protocol

HIDS: Host Intrusion Detection System

HTTP: Hypertext Transfer Protocol

HTTPS: Hypertext Transfer Protocol Secure

IDS: Intrusion Detection System

IEEE: The Institute of Electrical and Electronics Engineer

IETF: Internet Engineering Task Force

IP: Internet Protocol

IPFIX: IP Flow Information Export

IPv4: Internet Protocol version 4

IRC: Internet Relay Chat

KVM: Kernel-based Virtual Machine

LCS: Longest Common Substring

LF: Line Feed

NIDS: Network Intrusion Detection System

QEMU: Quick EMUlator

RFC: Request for Comments

SGBD: Sistemas Gerenciadores de Banco de Dados

SGM: Signature Generation Mechanism

SMTP: Simple Mail Transfer Protocol

SO: Sistema Operacional

SQL: Structured Query Language

SSH: Secure Shell

SSDT: System Service Dispatch Table

TCP: Transmission Control Protocol

TN: True Negative

TP: True Positive

UDP: User Datagram Protocol

URL: Uniform Resource Locator

VM:Virtual Machine

XSS: Cross-site scripting

YAF: Yet Another Flowmeter

CAPÍTULO 1 - Introdução

1.1 Considerações Iniciais

Incidentes de segurança, além de representarem grandes riscos para as organizações, têm se tornado mecanismos complexos, podendo-se utilizar de diversas técnicas para o seu êxito. De acordo com o CERT.br (CERT.br, 2016), as estatísticas apontam que dentre a variedade de ataques, a prospecção de redes e a proliferação de artefatos maliciosos (*malwares*) encontram-se entre as ameaças mais reportadas. Os *malwares* são programas maliciosos que realizam ações prejudiciais no sistema de computação comprometido, e ainda, em muitas vezes, são responsáveis por desencadearem diversos tipos de ataques a outros sistemas.

O número de exemplares de *malwares* vem crescendo cada vez mais em larga escala, ultrapassando a marca de 550 milhões de arquivos maliciosos em 2016 (AVTEST, 2016). De acordo com (MCAFEEa, 2016), só no último trimestre de 2015 foram identificados 42 milhões de novos *malwares*, cerca de 10% a mais que no trimestre anterior. Segundo a empresa de antivírus McAfee, que possui um laboratório com mais de 600 milhões de amostras, a quantidade de *malwares* cresceu cerca de 32% ao longo do ano de 2015 (MCAFEEb, 2016).

De modo a garantir a vitalidade das redes de computadores e a proteção dos bens das organizações, mecanismos têm sido criados e melhorados ao longo dos anos. Dentre os mecanismos, os Sistemas de Detecção de Intrusão

(IDS – *Intrusion Detection System*) têm auxiliado administradores na detecção de diversos tipos de ataques e mostrado resultados satisfatórios como meio de defesa para infraestrutura. O principal objetivo de um IDS é o monitoramento de sistemas de computação, a fim de detectar a ocorrência de incidentes de segurança.

Embora existam diferentes metodologias de detecção presentes nos IDS, a maioria apresenta uma estrutura similar que, segundo (WU; BANZHAF, 2010), pode se dividir entre: ambiente monitorado, que se refere à origem das informações analisadas, podendo estas serem oriundas de hospedeiros ou de uma rede; e método de detecção, que compreende o mecanismo utilizado para detectar eventos maliciosos, podendo ser detecção por abuso ou anomalia; ações de respostas, que podem ser passivas, como a emissão de alertas, ou ativas, ações direcionadas a sistemas computacionais, como por exemplo um *firewall*.

Em relação ao ambiente e aos dados a serem monitorados, em se tratando de monitoramento de redes, destacam-se as metodologias que fazem a análise do conteúdo de pacotes e as que não fazem. O uso de IDS que fazem a análise do conteúdo do pacote, como é o caso do IDS SNORT (SNORTa, 2016), pode causar latência ao processo de detecção por exigirem um grande esforço computacional, que se torna maior conforme aumenta a quantidade de informações que trafega pela rede, podendo chegar ao ponto em que alguns pacotes não sejam analisados.

Uma opção atrativa para solucionar o problema do monitoramento de redes com grande volume de dados é a utilização de fluxos de dados de rede, baseada no padrão IPFIX (*IP Flow Information Export*) (CLAISEa, 2016). Os fluxos de dados de rede contêm informações sumarizadas sobre todo o tráfego de um ambiente de rede, e não fornecem quaisquer informações sobre o conteúdo do pacote. Os fluxos de dados de rede são coletados e armazenados, passando a constituir uma rica fonte de dados para detecção de eventos e análise de segurança. Vários trabalhos na área de detecção de incidentes de segurança a partir da análise de fluxo de redes foram bem sucedidos, conforme pesquisas realizadas em (CORRÊA, 2009), (OLIVEIRA, 2012) e (BATISTA, 2012).

1.2 Identificação do problema e justificativa

Embora todos os elementos de um IDS sejam importantes, é possível afirmar que o mecanismo ou metodologia de detecção é o componente principal, uma vez que ele é responsável em dizer se houve uma ação maliciosa em um ambiente computacional. As metodologias de detecção são divididas entre: detecção por anomalia, em que o ambiente é monitorado à procura de eventos que fogem do padrão normal do uso de recursos do sistema; e detecção por abuso, que se caracteriza por detecções baseadas em regras, ou assinaturas de ataque, e descrevem exatamente o comportamento de uma ação maliciosa. Em (ELSHOUSH; OSMAN, 2010), os autores fazem uma análise comparativa entre os dois métodos, mostrando suas vantagens e desvantagens. As metodologias baseadas em anomalia têm a capacidade de detectar novos padrões de ataques, porém apresentam uma alta taxa de falso-positivo. As metodologias baseadas em abuso, apresentam um baixo nível de falso-positivo, entretanto é possível detectar apenas padrões de ataques (ou *malwares*) previamente conhecidos por meio de assinaturas. As assinaturas descrevem detalhadamente o funcionamento de um padrão de ataque ou de um comportamento de um *malware*, logo, é necessário o conhecimento prévio dos mesmos, para só então, poder criar uma assinatura.

A geração de assinaturas de *malwares* pode ser feita de forma manual, em que há intervenção de um analista de segurança para analisar a amostra, ou de forma automática, na qual mecanismos auxiliam na criação das assinaturas. No entanto, com o aumento na quantidade de *malwares* como apresentado em (MCAFEEb, 2016), o processo de geração manual de assinaturas tornou-se improdutivo, uma vez que cada assinatura é criada de forma a descrever características específicas de um único *malware*, ou de uma família de *malwares*.

As pesquisas nos últimos anos se concentraram em métodos que utilizam mecanismos de geração automatizada de assinaturas em resposta ao grande crescimento de incidentes de segurança (KAUR; SINGH, 2013). Porém, as pesquisas focam no uso de mecanismos capazes de gerar

assinaturas utilizando o conteúdo dos pacotes. Logo, verifica-se que existe uma carência de metodologias que empreguem fluxo de dados para a geração automática de assinaturas, objetivando a detecção de *malwares*.

1.3 Hipótese e objetivos

A hipótese deste trabalho é que a geração de assinaturas de rede para *malwares*, baseadas em fluxo de dados de rede, pode ocorrer de forma automatizada, possibilitando detectar, de forma parcial ou integral, a comunicação de *malware* em nível de rede.

Diante dos problemas apresentados na seção 1.2, este trabalho visa arquitetar um modelo capaz de elaborar assinaturas de ataques baseadas em fluxo de redes de forma automatizada, estando apto a lidar com o rápido crescimento na quantidade de *malwares*, tendo como preocupação a análise de grande quantidade de dados e a detecção de ameaças previamente desconhecidas.

1.4 Organização da dissertação

Esta dissertação está dividida em seis capítulos incluindo o atual. O capítulo 2 traz a fundamentação teórica, abrangendo os conceitos e as tecnologias utilizadas para execução deste projeto. No capítulo 3 constam os trabalhos relacionados ao tema que foram utilizados de suporte para elaboração da metodologia. No capítulo 4 é apresentada a metodologia utilizada para geração automatizada de assinaturas de *malwares* baseadas em fluxo de dados de rede. No capítulo 5 constam os experimentos que foram realizados juntamente com os resultados obtidos. Por fim, no capítulo 6 são feitas algumas considerações gerais sobre a pesquisa, além de propostas futuras.

CAPÍTULO 2 - Fundamentação Teórica

2.1 Segurança de Computadores e Redes

De acordo com (GOLLMANN, 1999), segurança de computadores e redes é baseada nos seguintes requisitos: **autenticidade** – garante ao destinatário que o originador dos dados é quem diz ser; **confidencialidade** – garante o acesso a uma determinada informação somente para quem possui autorização; **integridade** – garante que os dados não foram modificados, comprometendo o conteúdo; **disponibilidade** – garante que um determinado serviço/recurso de um sistema computacional estará disponível quando for necessário.

Os requisitos citados são essenciais para garantir a segurança em um determinado sistema computacional e constituem os quatro pilares da segurança da informação. Dessa forma, um ataque tem como objetivo subverter pelo menos um dos quatro requisitos citados.

2.1.1 Prospecção de rede

O processo de prospecção de rede pode ser entendido como uma sub-etapa do processo de coleta de informações (ou *Information Gathering*) de um ou mais alvos. Esse processo também pode ser chamado de varredura de

redes (*scans*) ou sondagem. Geralmente é utilizado para identificar possíveis alvos em uma determinada rede, e ainda, utilizado para identificar possíveis serviços que estão em execução nos sistemas de computação de interesse.

Essa etapa é amplamente utilizada por atacantes ou por *malwares* que realizam varreduras em endereços ou *ranges* de IP (*Internet Protocol*) em busca de serviços (CERT.br, 2016). Uma vez encontrado os serviços de interesse, são realizadas algumas ações maliciosas nesses serviços, como por exemplo: ataques de dicionário, ataques de força bruta e exploração de serviços vulneráveis.

Ataques de dicionário geralmente são utilizados em serviços onde há a requisição de algum tipo de identificação do usuário. Nesse tipo de ataque, o atacante possui uma base de dados com possíveis usuários e senhas para serem testados. Dessa forma, um atacante ou um *malware* realiza diversas tentativas para conseguir se autenticar utilizando os usuários e senhas presentes no banco de dados. Serviços comuns que recebem esse tipo de ataques são: SSH, Telnet e FTP. Os ataques de força bruta são uma variante dos ataques de dicionário, uma vez que o atacante tenta todas as possibilidades de combinações de palavras, cobrindo um número maior de possibilidade na formação de palavras (OLIVEIRA, 2012).

A exploração de serviços vulneráveis é muito utilizada por *malwares* ou atacantes, uma vez que permitem acesso ao alvo. Após o processo de prospecção, é possível saber quais serviços estão sendo executados, e em alguns casos, quais são suas versões. A partir do momento que a versão de um *software* é identificada, é possível saber se existe algum programa que explore aquela determinada versão de *software* ou ainda, é possível escrever um programa que a explore. O programa que explora uma determinada vulnerabilidade é conhecido como *exploit* (VAN DER VEEN et al., 2012).

Do ponto de vista do comportamento de rede, se comparado os quatros ataques citados anteriormente, no geral, os ataques de prospecção de rede, de dicionário e de força bruta produzem um comportamento diferente (superior) em relação a quantidade de pacotes trocados se comparado com os *exploits*.

2.1.2 Exploits

Durante o processo de desenvolvimento de um *software*, é possível que surjam diversos *bugs* que permanecem no *software* devido à falta de testes necessários ou por falta de experiência dos desenvolvedores. Ainda, esses *bugs* podem se tornar vulnerabilidades, podendo ser exploradas por trechos de código chamados *exploits* (VAN DER VEEN et al., 2012), que são capazes de tirar um programa de seu fluxo normal de execução, fazendo com que o programa execute códigos arbitrários, geralmente dando acesso à máquina para um invasor ou um *malware*. Em (VAN DER VEEN et al., 2012), o cenário de vulnerabilidades é dividido em duas principais partes, sendo a primeira, vulnerabilidades do tipo corrupção de memória, ou *memory corruption*, em que se enquadram as vulnerabilidades presentes em linguagens compiladas, como é o caso de C e C++. Os tipos mais comuns são:

- *buffer overflow* – onde um *buffer* recebe mais dados do que consegue armazenar, podendo sobrescrever endereços e valores de regiões de memória diferentes das alocadas para o *buffer*. Ainda, essa classe pode se dividir em, baseado em pilha (*stack-based*) (PHRACK, 2016), onde regiões de memória do segmento *stack* são sobrescritos, ou baseado em *heap* (*heap-based*) [(CGSECURITY, 2016), (OPENWALL, 2016)], memória esta que é alocada dinamicamente em tempo de execução. *Buffer overflow* baseado em *heap* geralmente ocorre com a sobrescrita de memória adjacente ou com a sobrescrita de metadados utilizados pelos algoritmos de alocação dinâmica (VAN DER VEEN et al., 2012).
- *format string* – ocorrem quando há mau uso de funções que utilizam *string* de formatação (*format string*), como é o caso da família de funções `printf`, por exemplo [(BUGTRAQ, 2016), (TESO, 2016)]. Geralmente esse tipo de vulnerabilidade pode ser utilizada para leitura de memória arbitrária ou para sobrescrita de

regiões de memória, fazendo com que códigos arbitrários sejam executados.

- *integer overflow* – ocorre quando há a incorreta manipulação de inteiros, podendo acarretar em outro tipo de vulnerabilidade, como é o caso de *buffer overflow*. Esses tipos de vulnerabilidades são comuns em servidores e sistemas operacionais (VAN DER VEEN et al., 2012).

A segunda categoria apresentada engloba as vulnerabilidades presentes em aplicações web, que são muito exploradas atualmente pelos atacantes, como podem ser vistas em (OWASP, 2016). Os tipos mais comuns e conhecidos são: vulnerabilidades na formatação e manipulação de *querys* SQL (denominadas vulnerabilidades do tipo SQL *Injection*), que normalmente têm como alvo de ataque um banco de dados; XSS ou *Cross-site scripting,* normalmente tem como alvo de ataque os usuários. Isso é possível em aplicações *web* que não validam dados fornecidos por usuários, fazendo com que um usuário mal intencionado armazene dados (geralmente *JavaScript*) no banco de dados. Esses dados serão executados no navegador das vítimas após requisitarem as páginas *web* que contêm o código malicioso injetado pelo atacante ou *malware*. Outros tipos de vulnerabilidades web podem ser vistas em (OWASP, 2016).

Em (CHANG et al., 2013) é realizada uma extensa revisão bibliográfica acerca dos tipos de vulnerabilidades exploradas por códigos maliciosos.

2.1.3 Código Malicioso

Códigos maliciosos, ou *malwares*, são programas desenvolvidos com objetivos específicos, porém compartilham uma mesma característica: eles são usados para violar a integridade, a confiabilidade ou a disponibilidade de sistemas computacionais. Segundo (ANDERSON, 2008), há um debate em relação à definição exata da classificação dos artefatos maliciosos, mas comumente são usadas as seguintes terminologias:

- ***Worms*:** são programas que se replicam de forma automática. Como mecanismo de infecção, esses artefatos se propagam através de exploração de vulnerabilidades, ou seja, fazem uso de *exploits*.
- **Vírus:** são códigos maliciosos que, para serem executados, necessitam que o usuário execute o arquivo no qual o vírus está contido. Após executado, o vírus se espalha para outros programas.

Os *worms* ou vírus possuem dois componentes (ANDERSON, 2008): mecanismo de replicação e *payload*. Em relação ao mecanismo de replicação, é possível ocorrer tanto através da propagação via exploração de serviços vulneráveis (*worms*) ou se anexando a um e-mail ou outros programas (vírus). O segundo componente (*payload*) é de fato a ação maliciosa que será realizada. Dependendo da ação específica realizada pelo *malware*, é adotado pela comunidade uma classificação que, quando houver, será mencionado ao longo do texto. A título de exemplo, é possível elencar algumas ações maliciosas:

- Mudanças seletivas ou randômicas nos estados de proteção da máquina;
- Encriptação de arquivos dos usuários, sendo geralmente arquivos que possuem algum valor para o usuário, tais como: fotos, planilhas, documentos, senhas, etc. Esse tipo de *malware* é conhecido como ***ransomware*** dado que o mesmo cobra um valor de resgate, para que, só então, seja liberada uma chave que descriptografe os arquivos.
- Conhecidos como ***keylogger***, esse são capazes de capturar as teclas pressionadas. Ainda, existe a evolução desse tipo de *malware* (no quesito de monitorar novas atividades) que é o ***screenlogger***, onde o mesmo monitora a imagem que está sendo exibida na tela e a posição do clique do *mouse* quando houver, sendo efetivo para captura de senhas que são inseridas por meio de um teclado virtual.
- Monitorar quaisquer atividades do sistema e reporta-las para o atacante, comprometendo assim a privacidade do usuário. Esse tipo de *malware* é referenciado como ***spyware*** (programa para espionagem).

- Exibir propagandas e anúncios sem a autorização do usuário, persuadindo-o a clicar em tais anúncios para serem redirecionados para sites maliciosos. Esse tipo de *malware* é conhecido como ***adware***.
- Realizar a instalação de outro programa – que pode estar contido dentro do próprio *malware* ou ainda ser obtido por meio de um *download* de algum servidor remoto ou local. São referenciados como ***dropper*** e ***downloader*** respectivamente.
- Manter-se persistente no sistema alvo instalando-se como uma ***backdoor***, assegurando que o acesso ao alvo se mantenha. Com a utilização de uma *backdoor*, não é necessário recorrer novamente ao método de invasão utilizado para obter acesso ao sistema, uma vez que, o acesso, agora, pode ser feito de forma direta através do uso da *backdoor*.
- ***Bot*** e ***botnets***, referem-se à máquinas ou dispositivos computacionais que foram capturados e fazem parte de uma rede (*botnets*), onde comumente são alugadas para serem utilizadas para outros fins, tais como: utilizá-las em ataques direcionados, envio de spam, *phishing* ou DDoS (*Distributed denial of service attacks*).

Ainda, é possível ocorrer a instalação de um ***rootkit***. Segundo (BLUNDEN, 2013), os *rootkits* podem ser entendidos uma “tecnologia” de “multiplicador de forças”. Um dos principais recursos de um *rootkit* são: capacidade de se esconder do sistema operacional, de forma que ferramentas padrões e mecanismos de segurança não os capturem; e de ser controlado remotamente. Dadas as suas características, os *rootkits* são muito utilizados para manter a comunicação e manutenção das máquinas que compreende as *botnets*. Esse tipo de artefato pode sofrer atualizações remotas, de modo que continue indetectável por mecanismos de segurança, ou, ainda, não deixar que nenhum outro tipo de *malware* infecte o sistema computacional no qual o *rootkit* está instalado (ANDERSON, 2008).

2.2 Coleta e análise de código malicioso

Sistemas de coleta de artefatos maliciosos (*malwares* ou quaisquer outros tipos de ataque) são amplamente utilizados como meio para entender o funcionamento dos incidentes de segurança. A coleta de artefatos maliciosos pode ocorrer de diversas maneiras, porém as duas principais, como definido em (LIN et al., 2014), são: 1) modo passivo, onde se enquadram nessa categoria os *honeypots* que disponibilizam serviços vulneráveis, à espera de ataques; e a disseminação de *malware* via e-mail, uma vez que este e-mail não é solicitado; 2) modo ativo, onde a coleta de *malwares* é feita através da análise de conteúdo de URL e compartilhamento P2P, a procura de arquivos maliciosos. Um exemplo de ferramenta criada para esta categoria é a *Honey-Inspector* (HONEYINSPECTOR, 2016).

Além das formas descritas, uma outra forma de se obter exemplares de *malwares* é através do uso de repositórios públicos. Alguns exemplos de repositórios são: VirusShare (VIRUSHARE, 2016), VXHeaven (VXHEAVEN, 2016), Offensive Computing (OFFENSIVECOMPUTING, 2016), VirusTotal (VIRUSTOTAL, 2016). O interessante em utilizar amostras de *malwares* provenientes de repositórios públicos é que, em geral, estes repositórios recebem amostras de diversas localidades, abrangendo uma diversidade grande de *malwares* que possuem diferentes localidades como alvo[1]. Além disso, a utilização de repositórios públicos evita o investimento em recursos para preparação do ambiente de coleta, caso seja utilizado *honeypots* por exemplo. Em contrapartida é importante certificar-se de que os exemplares presentes em tais repositórios não sejam antigos, podendo influenciar nos resultados por não representar padrões de técnicas/ataques atuais.

Se tratando do processo de análise de *malwares*, a literatura dividi-o em duas grandes áreas: estática e dinâmica. Ambas as abordagens possuem suas vantagens e desvantagens. Na Tabela 2-1 é realizado uma comparação entre ambos os métodos. Para uma extensa revisão acerca dos métodos

[1] Para estudos de comportamento, tendo como foco uma localidade em específico, talvez não seja interessante utilizar uma base de dados pública, a menos que na base de dados se tenha informações referentes a localidade. Desta forma será possível separar as amostras de uma determinada região, por exemplo.

utilizados para análise de *malware*, verificar (EGELE et al., 2012). Segundo (SIKORSKI; HONIG, 2012), é interessante utilizar as duas abordagens em conjunto para se ter uma análise completa

Tabela 2-1. Comparação entre análise estática e dinâmica.

Análise estática	**Análise dinâmica**
Malware não é executado.	*Malware* é executado.
Todos os caminhos de execução (*execution paths*) possíveis podem ser examinados.	Pode não ser possível cumprir todas as condições exigidas para executar um determinado caminho de execução (exemplo de condições: checagens de datas especificas ou programas específicos instalados).
Pode ser lenta (análise interpretando o código *assembly*) ou rápida (checagens automáticas de *strings*, *bytes* no arquivo ou funções utilizadas).	Pode ser lenta (execução manual de um *malware* ou usando um sistema de análise que executa o *malware* por horas ou dias) ou rápida (execução de um *malware* por alguns segundos ou minutos).
Nenhuma interação possível do *malware* com sua infraestrutura.	Possível interação do *malware* com a infraestrutura possibilitando o estudo dos dados transmitidos durante a execução.
Pode ser totalmente automatizada, embora a análise (feita por um humano) em nível de código *assembly* seja necessária.	Pode ser totalmente automatizada, embora alguns *malwares* possam não funcionar ou parar sua execução durante a análise.
Pode exigir um analista experiente, principalmente para a análise em nível de código *assembly*. No entanto, ferramentas automatizadas podem identificar informações relevantes, facilitando o trabalho.	Em se tratando de uma análise dinâmica convencional, pode exigir um nível inferior de experiência se comparado com a análise estática.

Pode dificultar a análise caso o *malware* esteja obfuscado ou empacotado (*packer*).	O *malware* irá se desempacotar (processo de *unpacking*) durante sua execução.
Pouco provável que o *malware* infecte a máquina de análise (a menos que exista vulnerabilidades nas ferramentas utilizadas durante a análise estática).	Muito provável que a máquina de análise seja infectada. *Sandboxes* podem retornar o estado anterior da máquina (antes do processo de infecção) após o término da análise.

Fonte: Tabela traduzida e adaptada de NELSON, 2016.

Em linhas gerais, o processo de análise estática compreende o estudo do artefato malicioso sem que haja a execução do mesmo. Dessa forma é possível analisar o seu código fonte (quando disponível) ou analisar o código *assembly* que pode ser obtido pela conversão do código de máquina (*machine code*) a partir do arquivo executável. Este processo de tradução de código de máquina para código *assembly* é conhecido como *disassembly* e é realizado por uma ferramenta chamada *disassembler*, tal como a ferramenta IDA Pro (EAGLE, 2011). Com isso é possível analisar as instruções que serão executadas pela CPU e as chamadas de sistema, revelando as capacidades que um *malware* possui.

A análise dinâmica é o processo de se analisar o artefato malicioso por meio de sua execução. Esta execução pode ocorrer de forma manual, onde existe intervenção humana, ou de forma automática, que é realizada através do uso de sistemas de análise, comumente conhecidos como *sandbox*.

Dada a natureza desse trabalho, que se baseia na análise do tráfego de rede para geração de assinaturas de rede, a discussão será centrada em mecanismos para efetuar a análise dinâmica como forma de se obter o tráfego de rede. Embora a análise estática possa fornecer detalhes dos protocolos a serem utilizados (por meio da análise das funções importados e de *strings*, por exemplo) a mesma não é realmente capaz de gerar o tráfego de rede necessário para este projeto (NELSON, 2016).

Os sistemas de análise são ambientes utilizados para execução de *malwares* com o objetivo de monitorar o seu comportamento, seja este em

nível de sistema operacional ou de rede. Esses sistemas são conhecidos por fazerem a análise dinâmica de um *malware*, ou seja, o *malware* é executado da mesma forma que um usuário o executaria, porém, neste tipo de ambiente, diversos subsistemas são monitorados, tais como: sistema de arquivo, redes, registros, memória, entre outros. O *malware* pode ser executado em um ambiente virtual ou ambiente físico (conhecido como *bare-metal* (KIRAT; VIGNA; KRUEGEL, 2014)). Embora as execuções em ambientes físicos se aproximam mais do comportamento real do *malware*, com o uso de ambientes virtuais o processo de recuperação do estado da máquina (anterior a infecção) é mais rápido.

Dentre os sistemas que realizam a análise de *malware*, é possível citar alguns: Anubis (ANUBIS, 2016), BehEMOT (FILHO et al., 2010), Sandbox Cuckoo (CUCKOO, 2016), Malwr (MALWR, 2016) e CWSandBox (WILLEMS; HOLZ; FREILING, 2007). Cada sistema utiliza uma abordagem diferentes para realizar a captura dos dados para futuras análises. A título de exemplo, serão explicadas duas abordagens que são bastante utilizadas nestes tipos de sistemas. Ambas as abordagens que serão explicadas utilizam *hooking*, porém existem outras abordagens que podem ser consultadas em (FILHO et al., 2011) e (EGELE et al., 2012).

> A técnica *hooking* pode ser definida como um meio de se alterar as requisições e respostas das interações realizadas em um sistema operacional ou por suas aplicações, através da interceptação das funções ou eventos utilizados. (FILHO et al., 2011).

A técnica de *hooking* caracteriza-se como sendo em nível de usuário (*userland hooking*) ou em nível de *kernel* (*kernel hooking*), onde esse último possui um nível mais elevado de privilégio em comparação ao primeiro. O sistema BehEMOT utiliza a técnica chamada *SSDT hooking* que se caracteriza como uma técnica de *kernel hooking*. *Hooking* da *System Service Dispatch Table*, ou como é conhecida SSDT *hooking*, é implementada por meio de um driver de *kernel* que aplica esta técnica modificando a tabela que armazena os endereços das chamadas de sistemas (*system calls* ou *syscalls*). Dessa forma, o endereço da função original será substituído pelo

endereço da função que realizará o monitoramento, monitorando assim tanto as requisições quanto os valores retornados. É possível verificar um esquema ilustrativo da técnica na Figura 2-1. Nesta figura é possível observar um exemplo de *hooking*, onde a função que irá realizar o monitoramento (`NewZwCreateFile`) se interpõem entre a função original e a respectiva entrada da SSDT. Esta técnica também é utilizada por *rootkits*, porém, além de monitorar as requisições e respostas, um *rootkit* geralmente modifica as informações em ambas as direções.

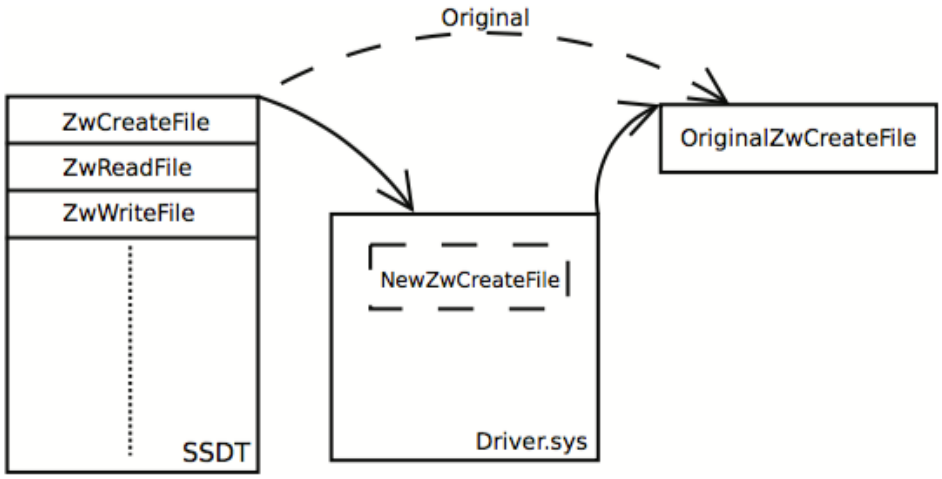


Figura 2-1. Desenho ilustrativo da técnica SSDT *hooking*.

Fonte: FILHO et al., 2011.

Um exemplo de *userland hooking* é a técnica chamada de *inline hooking*[2], técnica esta utilizada pela Sandbox Cuckoo. Essa técnica ocorre por meio do redirecionamento do fluxo de execução normal de um programa para uma região de memória onde se tenha total controle. Nesta técnica, é realizada a modificação de uma porção de instruções ao longo de uma função, onde a inserção das novas instruções é responsável por transferir o fluxo de execução para outra localização de memória. Para que esta técnica (*inline hooking*) seja utilizada, a Sandbox Cuckoo realiza o carregamento de uma DLL (técnica conhecida como DLL *injection*) no processo[3] que se deseja

[2] Esta técnica pode ser utilizada tanto em *userland hooking* quanto em *kernel hooking*, porém é comumente utilizado em *userland hooking* (FILHO et al., 2011).

[3] O processo monitorado pode ter sido criado por um executável malicioso ou não, por isso foi utilizado a palavra "processo" ao invés da palavra "*malware*". Ao final da análise dinâmica, o sistema irá fornecer alguns indicadores e um *score*, com base na análise, que podem ser utilizados na tentativa de classificar um executável como malicioso ou não.

realizar o monitoramento. Após carregada a DLL, a mesma se encarrega de realizar o *inline hooking* nas funções a serem monitoradas. Esta DLL é carregada no processo por um *script* em *python* (chamado *Agent*) que fica em execução na máquina de análise e se comunica com a Sandbox Cuckoo para transferir os executáveis a serem analisados, as informações capturadas e os comandos de controle.

Embora existam diferentes tipos de sistema de análise dinâmica, optou-se por utilizar a Sandbox Cuckoo uma vez que é possível realizar o *download* do mesmo e instalá-lo em um ambiente de pesquisa controlado. Ainda, outro fator que colaborou para sua escolha foi pelo fato do mesmo ser de código aberto e estar em constante desenvolvimento, possibilitando ajustes no seu funcionamento quando necessário. Além desses fatores, foi possível observar que em alguns trabalhos recentes, como é o caso de (NELSON, 2016), (JANG; WOO; BRUMLEY, 2013) e (KHARRAZ et al., 2016), a Sandbox Cuckoo tem sido utilizada como sistema de análise. Outros sistemas de *Sandbox* foram encontrados, como é o caso de: Ether (DINABURG et al., 2008), Anubis (ANUBIS, 2016) e CWSandBox (WILLEMS; HOLZ; FREILING, 2007). Porém, esses sistemas, ou são de código fechado, ou não possui atualizações recentes.

Além dos sistemas de análise citados, ainda existem os sistemas de análise que funcionam como serviço de *cloud*, bastando apenas realizar a submissão do *malware* a ser analisado. Entretanto, ao utilizar este tipo de sistema, dois fatores são introduzidos na pesquisa nos quais não favorecem a escolha deste tipo de serviço, são eles: 1) restrição na quantidade de *malwares* que podem ser analisados diariamente; 2) falta de transparência no ambiente de rede no qual o sistema esta implementado, fator importantíssimo segundo (ROSSOW et al., 2012).

Sendo assim, o processo de coleta foi realizado por meio da obtenção de *malwares* presentes em base pública e o sistema de análise escolhido para realizar a análise dinâmica foi a Sandbox Cuckoo dado os fatores apontados anteriormente. Por fim, segundo (OLIVEIRA, 2012), o processo de coleta em conjunto com os sistemas de análise, proveem um importante ambiente para pesquisas voltadas ao entendimento de ações maliciosas.

2.3 Fluxo de dados de rede

Com a finalidade de monitorar redes de computadores, equipamentos como roteadores ou ferramentas para captura de pacotes, podem realizar a medição do tráfego que flui em uma rede de computadores. Devido à grande quantidade de *softwares* e dispositivos para realizar tal tarefa, o IETF (*Internet Engineering Task Force*), órgão regulador de padrões da Internet, propôs que fosse criado um padrão para a obtenção e análise de tráfego de rede. Este padrão foi publicado em 2004 no RFC 3917 (QUITTEK, 2016) contendo os requisitos para o futuro estabelecimento de um padrão. O padrão a ser estabelecido foi chamado de IPFIX, que define como um fluxo IP deve ser formatado e transferido de um exportador para um coletor. As especificações do padrão IPFIX surgiram no RFC 5101(CLAISEb, 2016) (em 2008) e foram atualizadas no RFC 7011 (em 2013).

Após a definição do IPFIX, os padrões preexistentes e os novos começaram a se adequar à proposta do IETF, como é o caso do Netflow definido no RFC 3954 (CLAISEc, 2016), da Cisco Systems. O padrão Netflow define um fluxo como sendo uma sequência unidirecional de pacotes que flui através de um ponto de observação (normalmente um roteador). Na Figura 2-2, é possível observar como é realizado o processo de exportação dos fluxos por um roteador Cisco.

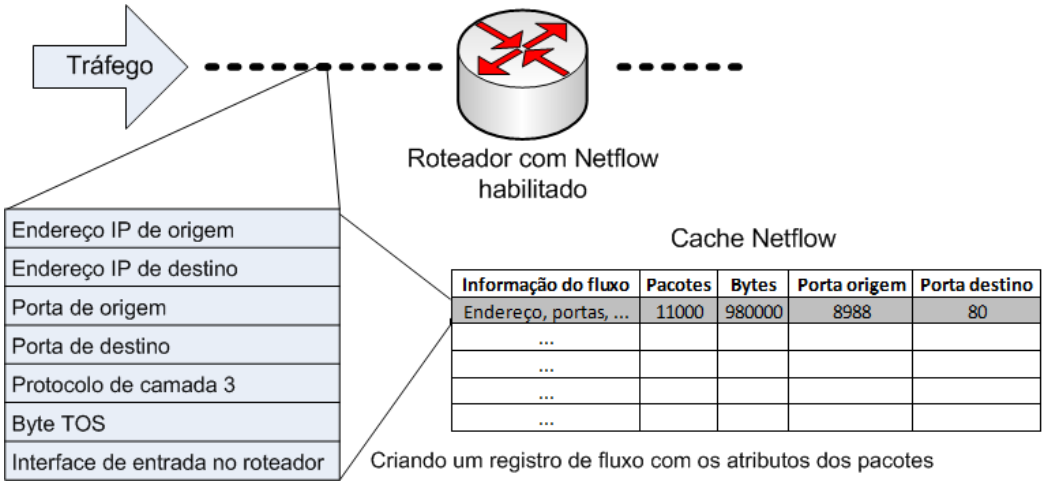


Figura 2-2. Esquema do padrão Cisco Netflow de fluxos de rede.

Fonte: Figura adaptada de CISCO, 2016.

Além do padrão Netflow que define um fluxo como sendo uma sequência unidirecional, também emergiu a tecnologia de fluxos bidirecionais, representada pelo Biflow, em que um fluxo representa pacotes fluindo em ambas as direções de uma conexão de rede. Na Figura 2-3 é possível observar a correlação de dois fluxos unidirecionais (originados a partir de uma mesmo conexão) em um único fluxo bidirecional.

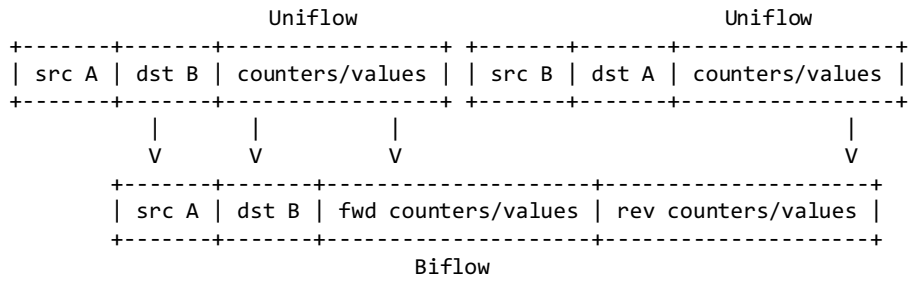


Figura 2-3. Construção de um fluxo bidirecional a partir de fluxos unidirecionais.

Fonte: TRAMMELL e BOSCHI, 2016.

Um fluxo de dados pode ser entendido como um tupla que possui alguns campos constantes, tais como: endereço IP de origem e destino; porta de origem e destino (protocolo da camada de transporte); o valor do campo Protocolo do datagrama IP; campo *Type of Service* do datagrama IP; e por fim, interface lógica de entrada do datagrama no roteador (ou *switch*).

Outros campos, tais como: quantidade de pacotes e quantidade de *bytes*, são contabilizados em cada tupla (fluxo) à medida que cada pacote flui pelo ponto de observação. Além de roteadores da Cisco, existem outras ferramentas *Open Source* para o mesmo fim de exportação e coleta de fluxo, como é o caso dos seguintes *softwares*: Argus (QOSIENT, 2016), IPFIXCol (LIBEROUTER, 2016), YAF (YAF, 2016) e SiLK (SILK, 2016). O IPFIXCol, por exemplo, é um coletor IPFIX que segue o padrão definido no RFC 7011.

Dessa forma é possível prover meios para a análise de rede sem acrescentar latência, uma vez que não é necessário realizar o desencapsulamento dos pacotes.

2.4 Sistema de detecção de intruso

Um sistema de detecção de intruso, ou *Intrusion Detection System* (IDS) tem como principal objetivo o monitoramento de sistemas de computação, seja apenas um computador ou um parque com uma grande quantidade de computadores, a fim de detectar a ocorrência de ações maliciosas. Tais ações podem compreender tentativas de ataques ou ataques bem sucedidos, que acabam resultando no comprometimento do ambiente, na violação da integridade, da confiabilidade ou da disponibilidade de recursos.

A maioria dos IDS apresenta uma estrutura similar em relação aos seus componentes (WU; BANZHAF, 2010). Dentre os componentes, os dois principais são: a origem dos dados analisados e a metodologia de detecção.

De acordo com a origem dos dados analisados, os IDSs se dividem em dois grandes grupos: 1) os *Host-based Intrusion Detection System* (HIDS), onde estes realizam o monitoramento de informações obtidas de um Sistema Operacional. Usualmente são monitoradas informações exclusivas de um único hospedeiro, compreendendo a análise de arquivos de *logs*, checagem de integridade dos arquivos, monitoramento de registros, detecção de *rootkis*, monitoramento de processos, dentre outras informações (OSSEC, 2016); 2) conhecidos como *Network Intrusion Detection System* (NIDS), esses realizam o monitoramento do tráfego de rede, podendo este monitoramento ser relativo a um único hospedeiro ou de uma rede inteira. Para ambos os grupos existem ferramentas que são bastante utilizadas e de código aberto, como é o exemplo do HIDS OSSEC (OSSEC, 2016), e dos NIDS SNORT (SNORTa, 2016) e Suricata (SURICATA, 2016).

Uma vez que este trabalho objetiva a análise do tráfego de rede, a explicação da metodologia de detecção será centrada nos NIDS, que é dividida em duas grandes subáreas:

- **Detecção por anomalia:** envolve os IDS que realizam sua detecção através da ocorrência de anomalias no uso de recursos do sistema, ocorrências essas que fogem do padrão normal de uso de aplicações. Com o uso desse tipo de técnica não é necessário existir uma base de dados com ataques previamente conhecidos (ZHANG; YANG; GENG,

2009). Esse tipo de detecção geralmente é baseado em limiares e não identifica aplicações ou eventos específicos.

- **Detecção por abuso:** caracteriza-se por detecções baseadas em regras que descrevem exatamente o comportamento de uma ação maliciosa. Tais regras são conhecidas como assinatura de um ataque e detalham um ataque previamente conhecido. Ainda, este grupo pode se dividir em duas categorias: as que fazem a análise do conteúdo dos pacotes (por exemplo os NIDS SNORT e Suricata); e as que não fazem a análise do conteúdo dos pacotes e são conhecidas como baseadas em fluxo de dados. Em (CORRÊA, 2009) é feito o rastreamento de fluxos de dados, onde são criadas assinaturas, utilizando *querys* SQL, baseadas em fluxo de dados de rede. Cada assinatura pode compreender diversos passos, onde cada passo corresponde a uma característica causada pelo evento a ser detectado. Por exemplo, se um evento realiza *n* conexões, uma assinatura pode conter *n* passos, responsáveis por detectar cada uma das conexões.

Embora as metodologias baseadas em anomalia possuam uma capacidade de detectar ataques desconhecidos, uma vez que os ataques causam um comportamento anômalo ao comportamento normal de rede, elas apresentam uma alta taxa de falso-positivo. Já as metodologias baseadas em abuso apresentam um baixo nível de falso-positivo. No entanto, há uma necessidade de se ter um banco de assinaturas para ser utilizado, ou seja, é preciso ter um conhecimento prévio dos ataques. Uma discussão mais detalhada pode ser encontrada em (ELSHOUSH; OSMAN, 2010).

Tratando-se dos NIDS baseados em abuso que utilizam informações de *payload* para detecção de certos comportamentos, na Figura 2-4 é possível observar um exemplo de uma assinatura utilizada por um NIDS que realiza a análise do conteúdo dos pacotes. Esta assinatura irá emitir um alerta quando detectar a ocorrência da *string* `Wefa7e` no campo `User-Agent` do protocolo HTTP.

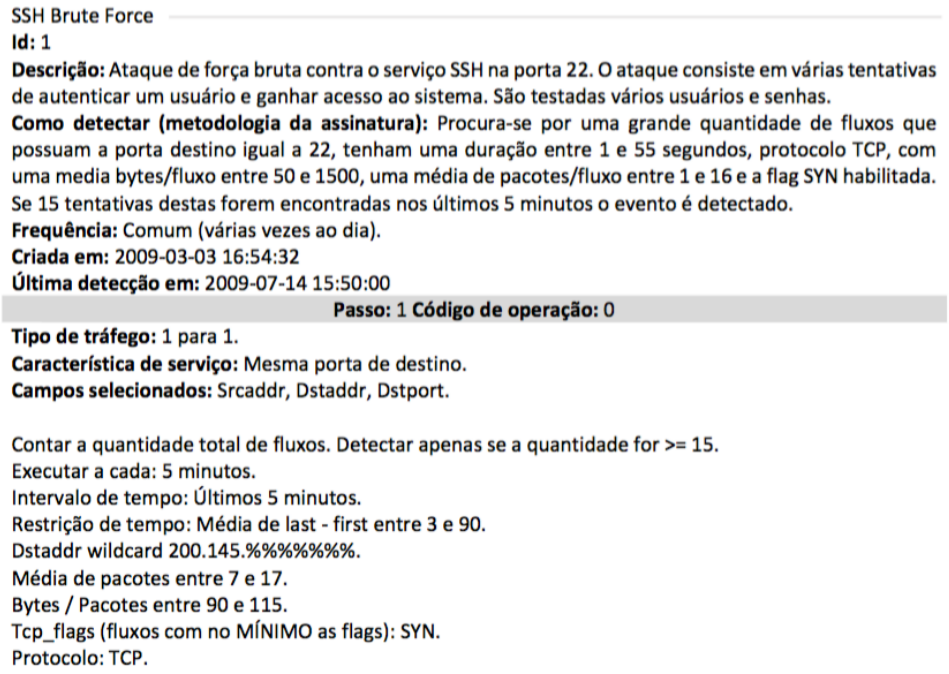
```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"TROJAN Malicious User-Agent"; content:"|0d 0a|User-Agent\: Wefa7e"; classtype:trojan-activity; sid:2000001; rev:1;)
```

Figura 2-4. Exemplo de assinatura baseado em *payload*.

Fonte: Assinatura retirada de SIKORSKI e HONIG, 2012.

Como dito anteriormente, o uso de NIDS que realizam detecção baseada em abuso necessitam de um banco de dados com as assinaturas, que pode ser atualizado através da obtenção de assinaturas que são criadas por organizações que cobram para tal criação, como é o caso da (SNORTb, 2016) e (PROOFPOINT, 2016); ou obtidas sem quaisquer custos (EMERGINGTHREATS, 2016) ou ainda, criadas individualmente seguindo a sintaxe exigida pelo NIDS.

Na Figura 2-5 é possível observar uma assinatura baseada em fluxo de redes para detectar ataque de força bruta ou dicionário, onde é procurado a ocorrência de fluxos de rede em uma base de dados que tenha como destino a porta 22, a mesma utilizada por padrão para o serviço SSH. A metodologia e sintaxe da assinatura apresentada na Figura 2-5 foi elaborada em (CORRÊA, 2009).


SSH Brute Force
Id: 1
Descrição: Ataque de força bruta contra o serviço SSH na porta 22. O ataque consiste em várias tentativas de autenticar um usuário e ganhar acesso ao sistema. São testadas vários usuários e senhas.
Como detectar (metodologia da assinatura): Procura-se por uma grande quantidade de fluxos que possuam a porta destino igual a 22, tenham uma duração entre 1 e 55 segundos, protocolo TCP, com uma media bytes/fluxo entre 50 e 1500, uma média de pacotes/fluxo entre 1 e 16 e a flag SYN habilitada. Se 15 tentativas destas forem encontradas nos últimos 5 minutos o evento é detectado.
Frequência: Comum (várias vezes ao dia).
Criada em: 2009-03-03 16:54:32
Última detecção em: 2009-07-14 15:50:00
Passo: 1 **Código de operação:** 0
Tipo de tráfego: 1 para 1.
Característica de serviço: Mesma porta de destino.
Campos selecionados: Srcaddr, Dstaddr, Dstport.

Contar a quantidade total de fluxos. Detectar apenas se a quantidade for >= 15.
Executar a cada: 5 minutos.
Intervalo de tempo: Últimos 5 minutos.
Restrição de tempo: Média de last - first entre 3 e 90.
Dstaddr wildcard 200.145.%%%%%%%.
Média de pacotes entre 7 e 17.
Bytes / Pacotes entre 90 e 115.
Tcp_flags (fluxos com no MÍNIMO as flags): SYN.
Protocolo: TCP.

Figura 2-5. Exemplo de assinatura baseado em fluxo de dados para detectar ataque de força bruta ou dicionário no serviço de SSH.

Fonte: Figura adaptada de CORRÊA, 2009.

No contexto de *malware*, uma assinatura pode detectar algumas atividades comumente realizadas por *malwares,* tais como: tráfego de rede que representa uma comunicação antes de uma infecção bem sucedida (ex: durante o processo de *scanning*); durante o processo de *download* de extensões de um *malware,* tarefa realizada por *downloaders*; comunicação com um C&C (*Command and Control*), entre outros (SIKORSKI; HONIG, 2012). Ainda, segundo (NELSON, 2016), uma assinatura pode ser abrangente, possibilitando detectar uma família de *malwares*, ou especifica, possibilitando detectar a comunicação de uma versão em particular de um determinado *malware*.

2.4.1 Extração de características

De acordo com a metodologia utilizada por cada NIDS, os mesmos são criados para tentar encontrar determinadas características na comunicação analisada. Essas características podem ser: 1) específicas a estrutura interna

dos protocolos (em nível de *payload*); 2) voltadas para o comportamento de comunicação realizada pelo uso dos protocolos (em nível de fluxo de dados de rede).

Tratando-se das características específicas à estrutura interna dos protocolos, estas devem ser extraídas e analisadas por metodologias ou ferramentas que realizam a extração e análise do conteúdo dos pacotes. As ferramentas Suricata e Bro (BRO, 2016) são exemplos de NIDS que podem ser utilizadas tanto para extração de características de protocolo quanto para detecção de determinados eventos por meio do uso de assinaturas. Tomando como exemplo o protocolo HTTP, é possível citar algumas características que são possíveis de serem extraídas utilizando as ferramentas citadas. Algumas dessas características são: método HTTP, *Uniform Resource Locator* (URL) e *User-Agent*.

Em relação a extração de características do comportamento de comunicação realizada pelo uso dos protocolos, em geral, são extraídas características que envolvem as mesmas informações presentes nos fluxos de dados. Tais características podem ser extraídas sem que haja a análise do conteúdo dos pacotes, compreendendo informações dos endereços envolvidos na comunicação, quantidade de *bytes* trocados, entre outras informações (verificar na seção 2.3 as informações que podem ser extraídas). Embora as ferramentas previamente citadas (Suricata e Bro) são comumente utilizadas para extração de informações presentes no conteúdo dos pacotes, as mesmas também possui a capacidade de extrair informações que podem ser utilizadas para identificar um fluxo de dados de rede. A título de exemplo, a ferramenta Bro, por padrão, cria um arquivo chamado `conn.log` contendo todas as comunicações (fluxos de dados de rede) presentes em um determinado *dump* de rede.

Referente às características em si, as mesmas podem ser utilizadas em seu formato bruto (exemplo, endereço de origem da comunicação) ou pré-processadas (exemplo, razão entre a quantidade de *bytes* trafegados pela quantidade de pacotes). Em (NELSON, 2016) é comentado sobre a utilização de um NIDS baseado em *payload* para extração de característica, onde informações presentes nos alertas (após a detecção de um ataque) emitidos

são utilizadas como características. Desta forma é possível saber se uma determinada comunicação apresenta padrões de comportamento realizado por *malwares* ou por determinado tipos de ataque.

A abordagem utilizada neste projeto consiste na extração e análise de características em ambos os níveis: *payload* e fluxo de dados de rede. Ainda, será utilizado o NIDS Suricata para extração de características, dado que o mesmo possui uma base de dados de assinaturas para *malwares* que é constantemente atualizada pela comunidade e distribuída gratuitamente.

2.5 Subsequência comum mais longa

No contexto de aplicações biológicas, em muitas vezes, é preciso determinar o grau de semelhança entre dois filamentos de DNA. Dentre as diferentes formas de se calcular essa semelhança, encontra-se o cálculo da subsequência comum mais longa, que, dado dois filamentos S_1 e S_2 é encontrado um terceiro filamento S_3, onde a base desse último aparece em cada um dos filamentos S_1 e S_2 (consultar (CORMEN, 2009) para saber as demais formas de se encontrar semelhança entre filamentos de DNA). Portanto, quanto mais longo o filamento S_3, maior será a semelhança entre S_1 e S_2. Um detalhe importante é que essas bases não precisam ser necessariamente consecutivas, mas devem aparecer na mesma ordem.

Quando zero ou mais elementos são omitidos de uma sequência dada, tem-se uma subsequência. De modo formal, dada uma sequência $X = \langle x_1, x_2, \dots, x_m \rangle$, uma outra sequência $Z = \langle z_1, z_2, \dots, z_k \rangle$ é uma ***subsequência*** de X se existir uma sequência estritamente crescente $\langle i_1, i_2, \dots, i_k \rangle$ de índices de X tais que, para todo $j = 1, 2, \dots, k$, temos $x_{i_j} = z_j$. Tomando como exemplo a sequência $X = \langle A, B, C, B, D, A, B \rangle$, $Z = \langle B, C, D, B \rangle$ é uma subsequência de X com índices correspondentes $\langle 2, 3, 5, 7 \rangle$ (o valor inicial do índice é 1).

Se o objetivo for encontrar uma subsequência Z entre duas sequências X e Y, logo a sequência Z é uma ***subsequência comum*** de X e Y. Se tomarmos a sequência $X = \langle A, B, C, B, D, A, B \rangle$ e $Y = \langle B, D, C, A, B, A \rangle$, a

sequência $\langle B, C, A \rangle$ é uma subsequência comum das sequências X e Y. A sequência $\langle B, C, A \rangle$ possui comprimento igual a 3, e, embora seja uma sequência de comum das sequências X e Y, a mesma não é uma subsequência comum *mais longa* (LCS – *Longest Common Subsequence*). Neste caso, as sequências $\langle B, C, B, A \rangle$, $\langle B, D, A, B \rangle$ e $\langle B, C, A, B \rangle$ são LCS (comprimento igual a 4) de X e Y visto que não existe nenhuma subsequência comum de tamanho 5 ou maior.

$$c[i,j] = \begin{cases} 0 & se\ i = 0\ ou\ j = 0 \\ c[i-1, j-1] + 1 & se\ i, j > 0\ e\ x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & se\ i, j > 0\ e\ x_i \neq y_j \end{cases} \quad (2.1)$$

Tendo como base a equação 2.1, construída a partir de um teorema que pode ser conferido em (CORMEN, 2009), é construído o algoritmo LCS-LENGTH (Figura 2-6). É utilizado programação dinâmica para calcular as soluções do problema, dado que o problema da LCS tem somente $\Theta(mn)$ subproblemas distintos (CORMEN, 2009). A programação dinâmica resolve problemas combinando as soluções para subproblemas.

> O procedimento LCS-LENGTH toma duas sequências $X = \langle x_1, x_2, \ldots, x_m \rangle$ e $Y = \langle y_1, y_2, \ldots, y_n \rangle$ como entradas. Armazena os valores c$[i,j]$ em uma tabela c$[0..m, 0..n]$ e calcula as entradas em ***ordem orientada por linha*** (isto é, preenche a primeira linha de c da esquerda para a direita, depois a segunda linha, e assim por diante). O procedimento também mantém a tabela b$[1..m, 1..n]$ para ajudar a construir a solução ótima. Intuitivamente, $b[i,j]$ aponta para a entrada da tabela correspondente à solução ótima de subproblema escolhida ao se calcular $c[i,j]$. O procedimento retorna as tabelas b e c; $c[m,n]$ contém o comprimento de uma LCS de X e Y. (CORMEN, 2009).

```
LCS-LENGTH (X,Y)
1     m = X.comprimento
2     n = Y.comprimento
3     sejam b[1..m, 1..n] e c[0..m, 0..n] tabelas novas
4     for i = 1 to m
5       c[i,0] = 0
6     for j = 0 to n
7       c[0,j] = 0
8     for i = 1 to m
9       for j = 1 to n
10        if x_i == y_j
11          c[i,j] = c[i-1, j-1] + 1
12          b[i,j] = "↖"
13        elseif c[i-1, j] ≥ c[i, j-1]
14          c[i,j] = c[i-1, j]
15          b[i,j] = "↑"
16        else c[i,j] == c[i, j-1]
17          b[i,j] = "←"
18    return c, b
```

Figura 2-6. Trecho de código referente a função LCS-LENGTH.

Fonte: CORMEN, 2009.

A partir do algoritmo LCS-LENGTH descrito, são calculadas duas tabelas, *b* e *c* (apresentadas na Figura 2-7), que são utilizadas como explicado a seguir. O tempo de execução do algoritmo LCS-LENGTH é $\Theta(mn)$.

> A tabela *b* retornada por LCS-LENGTH nos habilita a construir rapidamente uma LCS de $X = \langle x_1, x_2, \dots, x_m \rangle$ e $Y = \langle y_1, y_2, \dots, y_n \rangle$. Simplesmente começando em $b[m, n]$ e percorremos a tabela seguindo as setas. Sempre que encontramos uma "↖" na entrada $b[i, j]$, ela implica que $x_i = y_j$ é um elemento da LCS que LCS-LENGTH encontrou. Com esse método, encontramos os elementos da LCS em ordem inversa. O procedimento recursivo a seguir imprime uma LCS de X e Y na ordem direta adequada. A invocação inicial é PRINT-LCS($b, X, X.comprimento, Y.comprimento$).
> (CORMEN, 2009).

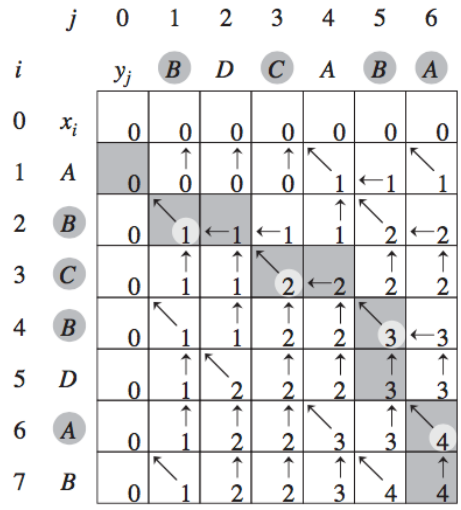


Figura 2-7. Tabelas c e b calculadas por LCS-Length para as sequências X = <A, B, C, B, D, A, B> e Y = <B, D, C, A, B, A>.

Fonte: CORMEN, 2009.

Segundo (CORMEN, 2009), o algoritmo de impressão da LCS demora o tempo $O(m+n)$, uma vez que decrementa no mínimo um de *i* e *j* em cada etapa da recursão. Para a tabela *b* da Figura 2-7, o procedimento (apresentadas na Figura 2-7) irá imprimir $\langle B, C, B, A \rangle$.

```
PRINT-LCS(b,X,i,j)
1    if i == 0 ou j == 0
2        return
3    if b[i,j] == "↖"
4        PRINT-LCS(b,X,i-1,j-1)
5        print x_i
6    elseif b[i,j] == "↑"
7        PRINT-LCS(b,X,i-i,j)
8    else PRINT-LCS(b,X,i,j-1)
```

Figura 2-8. Trecho de código referente a função PRINT-LCS.

Fonte: CORMEN, 2009.

Embora o algoritmo Print-LCS (Figura 2-7) descrito, por padrão, imprima apenas uma LCS, é possível realizar algumas alterações (verificar (WIKIBOOKS, 2016)) fazendo com que o mesmo imprima todas as LCS

possíveis de tamanho máximo. Ainda, segundo (BERGROTH, 2000), pode ser utilizado busca em largura a partir do grafo resultante para encontrar todas as LCS possíveis. Para o caso da Figura 2-7, além da LCS extraída, é possível extrair outras duas: $\langle B, D, A, B \rangle$ e $\langle B, C, A, B \rangle$.

2.6 Considerações finais

O objetivo deste capítulo foi introduzir os principais conceitos utilizados nesta pesquisa, além das tecnologias relacionadas ao tema. Além da revisão geral acerca de algumas taxonomias de *malwares*, foram apresentadas algumas tecnologias que serão utilizadas nesta pesquisa, tais como: fluxo de dados de redes, sistemas de detecção de intrusão e sistemas de análise de artefatos maliciosos, no qual este último servirá para gerar os dados necessário para essa pesquisa.

Por fim, foi apresentado o funcionamento do algoritmo LCS, que, nesta pesquisa, será utilizado para encontrar semelhanças nas conexões resultantes a partir de múltiplas execuções de um *malware*.

CAPÍTULO 3 - Trabalhos Relacionados

3.1 Considerações iniciais

Ao longo deste capítulo serão apresentados os trabalhos relacionados a esta pesquisa. Embora existam diversos trabalhos que propõem metodologias para geração automatizada de assinatura de rede, nenhum deles propõem uma metodologia para geração de assinatura de rede para ser utilizada em Sistema de Detecção de Intrusão baseado em fluxo de dados de rede, objetivando a detecção de *malwares*.

O processo de geração automatizada de assinatura surgiu com grande impacto em 2004 com a ferramenta Honeycomb (KREIBICH; CROWCROFT, 2004). Desde então, diversas metodologias que objetivam a criação automatizada de assinaturas têm sido propostas como indicadas nos *surveys* publicados em 2005 (WARAICH, 2005), 2006 (D1.2, 2016) e 2013 (KAUR; SINGH, 2013).

Os trabalhos aqui descritos serviram de base para a definição da metodologia e para execução dos experimentos. Mais especificamente, os trabalhos serviram de base nos seguintes pontos: 1) arquitetura genérica dos mecanismos de geração de assinaturas (SGM – *Signature Generation Mechanism*) e fluxo de dados de rede; 2) similaridade entre diferentes tipos de protocolo; 3) melhores práticas utilizadas pelas pesquisas no contexto de análise de *malwares*.

3.2 Geração automatizada de assinatura

Em (KREIBICH; CROWCROFT, 2004) é desenvolvida uma ferramenta para geração automatizada de assinatura baseada em *payload*, chamada Honeycomb. Tal ferramenta foi desenvolvida como uma extensão da ferramenta honeyd (HONEYD, 2016), e utiliza as informações de tráfego de rede capturadas pela honeyd. Após a análise dos protocolos, a metodologia apresentada em Honeycomb realiza a remontagem do conteúdo dos pacotes e, em seguida, é utilizado o algoritmo LCS entre sequências de *bytes* presentes nos pacotes. Após detectado alguns padrões (semelhanças de sequências de *bytes*) pelo algoritmo LCS, as assinaturas são montadas e utilizadas em ambos os NIDS, SNORT e Bro. Outras duas pesquisas, apresentadas em [(PARK et al., 2008), (WANG; STOLFO, 2004)], também utilizam o algoritmo LCS na etapa de extração de assinatura, onde são extraídos *strings* ou padrões de *bytes* em hexadecimal de aplicações específicas.

Em (PORTOKALIDIS; SLOWINSKA; BOS, 2006) foi elaborado um sistema para geração automatizada de assinaturas, chamado Argos. O sistema Argos é um *honeypot* de alta interatividade baseado em um emulador x86 (QEMU) (QEMU, 2016), e é capaz de detectar quando uma vulnerabilidade é explorada. Os tipos de vulnerabilidades detectadas pelo Argos são do tipo *buffer overflow* (baseados em *stack* e *heap*) e do tipo *format string*. Quando uma vulnerabilidade é explorada, o Argos é capaz de realizar um *dump* na sequência de *bytes* utilizada pelo *exploit* e correlacioná-la com os dados de rede capturados, dessa forma, são geradas assinaturas para o *exploit* utilizado. Apesar de gerar diversas assinaturas, o mesmo utiliza um subsistema chamado SweetBait, que tem o objetivo de identificar semelhanças entre as assinaturas, realizando um processo de fusão entre elas quando semelhanças são encontradas. Sendo assim, a quantidade total de assinaturas é reduzida. Por fim, essas assinaturas são testadas no IDS SNORT. O processo completo pode ser visto na Figura 3-1.

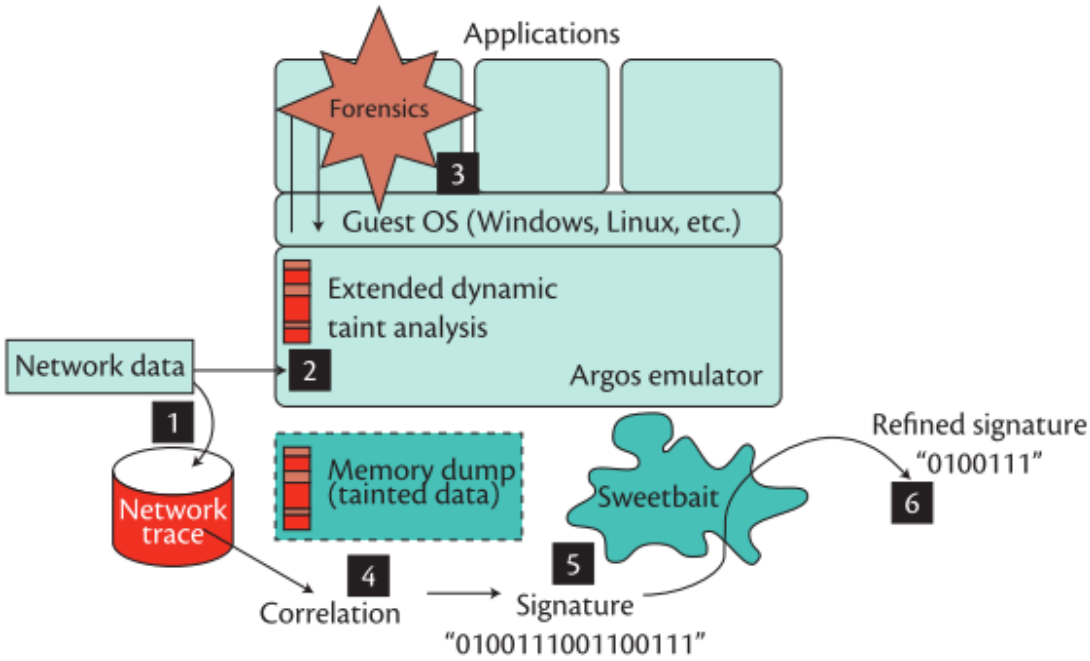


Figura 3-1. Arquitetura do sistema Argos.

Fonte KAUR e SINGH, 2013

Em (RIECK et al., 2010) é proposta uma abordagem para geração automatizada de assinaturas para *malwares*, onde os mesmos são executados repetidas vezes em ambiente com diferentes configurações (por exemplo, diferentes Sistemas Operacionais), visando produzir atividade de rede diferente entre as execuções. As assinaturas em (RIECK et al., 2010) utilizam informações de *payload*, onde o tráfego de rede capturado é dividido em duas partes, sendo que uma parte é utilizada para geração das assinaturas e a outra parte para medição de taxa de detecção e de falso-positivo. Inicialmente são extraídas sequências de *bytes* do tráfego de rede (gerado durante a execução do *malware*) que são chamados de *tokens*, para, posteriormente, serem comparados com o tráfego normal. O objetivo de se comparar tais *tokens* com o tráfego normal é remover a parcela dos *tokens* que se associam com o tráfego normal, reduzindo assim a taxa de falso-positivo.

Em (RAFIQUE; CABALLERO, 2013) é criada uma metodologia capaz de gerar assinaturas de *malwares* baseadas em *payload* de forma automatizada. São identificadas as semelhanças com base em características extraídas de alguns protocolos (para o protocolo HTTP por exemplo, são agrupadas as

requisições com a mesma URL requisitada), que, por sua vez, são separadas em *clusters* diferentes. Dessa forma é criado um conjunto inicial de assinaturas para cada *cluster*, de forma a identificá-los. Após esse processo, são identificadas as características, que são compartilhadas entre os *clusters*, por consequência, entre as assinaturas e, posteriormente, é realizando o agrupamento dessas assinaturas. Por fim, essas assinaturas são testadas, utilizando o IDS SNORT, o que possibilita encontrar e eliminar as assinaturas que foram criadas e as que contêm uma alta taxa de falsos-positivos.

3.3 Fluxo de dados de redes

A seguir serão apresentados alguns trabalhos que fazem uso de informações presentes em fluxo de dados de rede nas áreas de detecção de intrusão e geração automatizada de assinaturas.

Em (CORRÊA, 2009) é desenvolvida uma metodologia para detecção de eventos de rede com base em assinaturas de abuso ou anomalia. De modo geral, o sistema realiza a coleta e o armazenamento dos fluxos de dados, em seguida, compara as assinaturas previamente cadastradas com os fluxos de dados armazenados, verificando assim se alguma assinatura identificou algum evento. Como descrito anteriormente, cada assinatura pode compreender diversos passos, onde cada passo corresponde a uma característica causada pelo evento a ser detectado. O processo de geração de assinaturas para a metodologia proposta em (CORRÊA, 2009) ocorre de forma manual, onde são observadas as características presentes nos fluxos após a simulação de um ataque. Após observada tais características, é possível descrevê-las por meio das assinaturas, detectando assim futuras ocorrências do ataque.

Em (CHHABRA; JOHN; SARAN, 2005) é elaborado um algoritmo que a partir do tráfego de rede, utiliza técnicas de *clustering* para agrupar fluxos em categorias com base em valores de campos em comum dentre os pacotes. Se o número total de pacotes em um *cluster* for maior que um limite (*threshold*) especificado, então os campos em comum e seus respectivos valores

presentes no *cluster* são utilizados para gerar uma assinatura. A captura das informações ocorre por amostragem, sendo utilizada informações que são obtidas em nível de fluxo (endereço IP, *flags* TCP, porta, etc). As assinaturas geradas possuem pares de campo e valor (por exemplo, {(src_addr, 11.0.0.1), (src_port,80)}), contendo informações de fluxo de rede. Tal metodologia é capaz de gerar assinaturas para *worms* com base em apenas informações presentes em fluxo. Entretanto, como tais assinaturas possuem apenas informações de fluxo, essas são menos específicas se comparadas com as assinaturas baseadas em *payload*.

Em (OLIVEIRA, 2012) é elaborado um modelo para geração de assinatura de comportamento e de rede, baseado no sistema imunológico humano. São geradas assinaturas a partir de informações obtidas em nível de sistema operacional e tráfego de rede. Para geração de assinaturas de comportamento, são utilizadas informações obtidas através de chamadas de sistema (*syscall*) monitoradas. Assinaturas baseadas no tráfego de rede utilizam as seguintes informações obtidas a partir de fluxo de dados de rede: total de *bytes*, número da porta de destino e protocolo da camada de transporte utilizado.

3.4 Outros

Além dos trabalhos previamente descritos que envolvem a geração automatizada de assinatura de ataques e detecção de eventos baseados em fluxo de dados de rede, é importante citar alguns trabalhos que realizam o agrupamento de *malwares* com base em algumas características do tráfego de rede. Além de agrupar *malwares* com base em similaridade presente no tráfego de rede, tais trabalhos fornecem métodos para extração e detecção de similaridade entre as conexões dos *malwares*, que podem ser úteis para o processo de geração de assinatura, como é o caso do trabalho previamente citado (RAFIQUE; CABALLERO, 2013).

No contexto de similaridade entre protocolos, em (PERDISCI; ARIU; GIACINTO, 2013) e (PERDISCI; LEE; FEAMSTER, 2010), foram propostas metodologias para realizar o agrupamento de exemplares de *malwares* com

base nas requisições HTTP similares. Em (CABALLERO et al., 2011) são utilizadas informações do método da requisição HTTP e os parâmetros da URL requisitada, visando agrupar os exemplares de *malwares* em famílias.

Em (NELSON, 2016) é realizado o processo de *clustering* em nível de pacote e sessão HTTP. O processo de *clustering* em nível de pacote é baseado no método proposto em (RAFIQUE; CABALLERO, 2013), onde são extraídas algumas informações em nível de rede e transporte dos protocolos UDP e TCP. Já o processo de *clustering* em nível de sessão HTTP, é baseado no método proposto em (PERDISCI; LEE; FEAMSTER, 2010), onde após extraídas informações da linha de requisição (método e URL), são calculadas algumas distâncias. Com base em tais distâncias é possível identificar o quão próxima uma requisição HTTP esta das demais.

Como visto nos trabalhos citados, geralmente são utilizados algoritmos de *clustering* para agrupar requisições HTTP em *clusters* que sejam semelhantes entre si. Esta ideia de agrupamento geralmente é utilizada para identificar características que são semelhantes entre diferentes exemplares de *malwares* que podem pertencer a uma mesma família. Ainda, alguns trabalhos estendem esta ideia de agrupamento por semelhança através de mecanismos responsáveis por criar assinaturas que identifiquem tais *clusters*, detectando desta forma famílias de *malwares*.

No contexto de melhores práticas para serem utilizadas ao longo do processo de análise de *malwares*, uma pesquisa bastante referenciada foi publicada no ano de 2012 em (ROSSOW et al., 2012), onde são fornecidas algumas diretrizes para se projetar experimentos prudentes em pesquisas que envolvem análise de *malware*. Com base em tais diretrizes, é analisado o rigor metodológico e a prudência de 36 trabalhos acadêmicos publicados entre os anos de 2006 e 2011. Em relação as diretrizes, a seguir foram destacadas as principais que possuem relação direta com o trabalho que será desenvolvido:

- **Conjunto inicial de dados confiável** – importância de não haver executáveis benignos no conjunto inicial de *malwares* e a quantidade de *malwares* deve estar balanceada entre as famílias

analisadas. Essa pré-seleção pode ser facilmente realizada pela ferramenta AVClass[4].

- **Transparência** – declarar os nomes das famílias e os *hashes* dos *malwares* analisados. Além disso, descrever o sistema operacional utilizado, *softwares* adicionais e politicas de contenção de rede. Exemplos de trabalhos que fornecem informações em relação ao ambiente utilizado, são: (ROSSOW et al., 2011) e (RIECK et al., 2010).

- **Realismo –** destaca a importância de permitir que o *malware* tenha conectividade com a Internet e, se utilizado algum serviço emulado, é necessário descrevê-lo. Além disso, é aconselhado não generalizar os resultados caso tenha sido utilizado apenas um único tipo de sistema operacional. Ainda, é indicado a utilização de famílias relevantes de *malwares* e que ainda possuam *malwares* recentes.

- **Segurança do ambiente –** implementar e descrever as políticas de contenção, realizando o redirecionamento de tentativas de SPAM e de infecção, além de tentar reprimir ataques do tipo DoS (*Denied of Service*). Em (ROSSOW et al., 2011), a largura de banda é limitada na tentativa de evitar ataques do tipo DoS, enquanto que ataques do tipo SPAM e tentativa de infecção são direcionadas para uma *honeypot*.

3.5 Considerações finais

Neste capítulo foram apresentados alguns trabalhos relacionados ao tema de análise de *malware*, geração automatizada de assinatura e Sistema

[4] https://github.com/malicialab/avclass

de Detecção de Intrusão. Inicialmente foram apresentadas algumas pesquisas que têm como foco a geração automatizada de assinaturas, sejam elas para *malwares* ou *exploits*, além dos métodos utilizados por tais trabalhos, que em sua maioria utilizam técnicas de clusterização dada suas características.

Como apresentado anteriormente, alguns trabalhos utilizam o algoritmo LCS durante a etapa de geração de assinatura, de modo a encontrar similaridade entre sequências de *bytes* presentes nos pacotes. O algoritmo LCS será utilizado na metodologia desenvolvida neste trabalho para encontrar similaridade, porém, diferentemente dos trabalhos apresentados, tal similaridade não será baseada entre os *bytes* dos pacotes, e sim entre as conexões de múltiplas execuções de um *malware*, como será visto na próxima seção.

Além dos trabalhos que utilizam LCS, foram apresentados alguns trabalhos que utilizam informações obtidas de fluxo de dados rede. Em se tratando de geração de assinatura para *malware* utilizando fluxo, será utilizado como base, em partes, o trabalho desenvolvido em (OLIVEIRA, 2012) que, dentre os tipos de assinatura gerada, realiza a geração automatizada de assinatura para fluxo de dados de rede. Das informações utilizadas para geração de assinatura de rede em (OLIVEIRA, 2012), tais como quantidade total de *bytes*, número da porta de destino e protocolo da camada de transporte, na metodologia que será desenvolvida serão adicionados os seguintes campos às assinaturas: endereço IP de destino e comparador de *bytes*. Essa adição será possível graças a análise de *payload* que será realizada para alguns protocolos como será visto na seção 4.5. Os trabalhos apresentados que abordam similaridade entre protocolos foram utilizados como base no processo de extração e análise de *payload*.

Além disso, foram citados alguns trabalhos que visam a contenção e o controle de rede durante o processo de análise dinâmica de *malware.* Tais trabalhos foram fundamentais para o desenvolvimento da metodologia, fornecendo alguns guias e melhores práticas no contexto de análise de *malware*.

Em relação aos trabalhos apresentados, a metodologia proposta objetiva gerar, de forma automatizada, assinaturas para serem utilizadas em Sistemas

de Detecção de Intrusão que faz uso de assinaturas baseadas em fluxo de dados de rede.

CAPÍTULO 4 - Metodologia

4.1 Objetivos

O principal objetivo deste projeto é a geração automatizada de assinaturas baseadas em fluxo de dados de redes para detecção de *malware*s. Tais assinaturas serão criadas com base em características semelhantes entre múltiplas execuções da amostra analisada. Inicialmente serão utilizadas algumas técnicas para extrair características presentes no tráfego de rede para posteriormente serem utilizadas pelo algoritmo LCS, que irá desempenhar o papel de encontrar semelhanças entre execuções diferentes do mesmo *malware*, caso existam.

Embora existam metodologias apresentadas na literatura que realizam a criação de assinaturas de detecção de *malwares*, nenhuma possui como objetivo a criação de assinaturas automatizada que possuem apenas informações de fluxo de dados de rede. Em geral, as metodologias se preocupam apenas em criar assinaturas baseadas em *payload*, como apresentado em (RAFIQUE; CABALLERO, 2013). Portanto, a principal contribuição deste projeto é apresentar uma metodologia, juntamente com os desafios e as respectivas soluções utilizadas, que seja apta a gerar assinaturas baseadas em fluxo de dados de rede para *malwares*, a partir do tráfego de rede gerado pelo *malware* em múltiplas execuções.

Por meio do processo de geração de assinaturas de forma automatizada, espera-se criar assinaturas que não possuam falso-positivo, descrevendo as

características de comunicação exercidas pelo *malware*. Após criadas tais assinaturas, estas podem (embora esteja fora do escopo deste trabalho) ser utilizadas em um IDS que utilize assinaturas baseadas em informações presentes em fluxo de dados rede, como é o caso do trabalho desenvolvido em (CORRÊA, 2009).

4.2 Arquitetura Proposta

Primeiramente será descrito brevemente o funcionamento geral do sistema e a interligação entre os componentes que compõem este sistema. Nas seções seguintes serão detalhados cada componente de forma individual fornecendo detalhes técnicos para um melhor entendimento do sistema. Na Figura 4-1 é fornecida uma visão geral da arquitetura proposta.

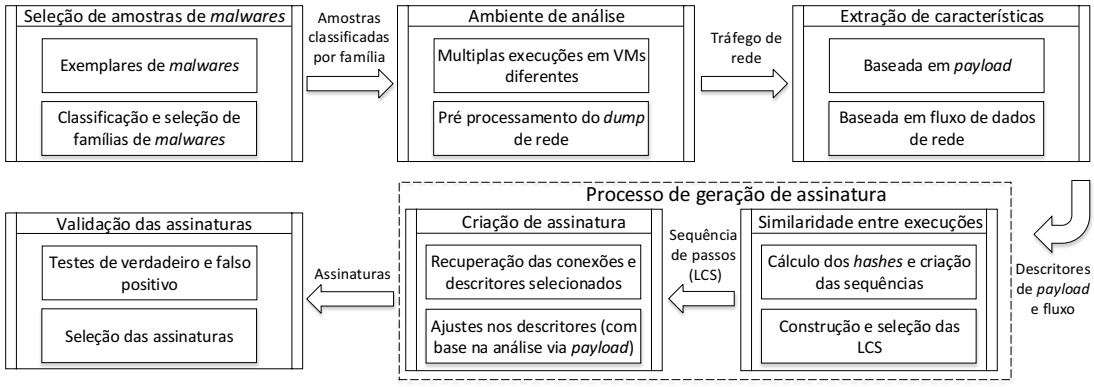


Figura 4-1. Visão geral da arquitetura proposta.

Inicialmente os *malwares* são obtidos de um repositório público e selecionados utilizando a ferramenta AVClass[5]. Os *malwares* são classificados em famílias, e, a partir de cada família, é obtido uma mesma quantidade de amostras. Detalhes acerca da seleção de amostras de *malwares* são fornecidos na seção 4.3. Posteriormente essas amostras de *malwares* são submetidas ao ambiente de análise que é responsável por escalonar a amostra para ser executada em uma das máquinas virtuais

[5] https://github.com/malicialab/avclass

presentes no ambiente de análise. Cada amostra de *malware* é executada múltiplas vezes em uma máquina virtual escolhida aleatoriamente. Após executada, é obtido o *dump* do tráfego de rede gerado durante a execução (caso haja) e então são removidas todas as conexões que não foram originadas pelo *malware*, ou seja, conexões de atualizações do próprio SO (Sistema Operacional) ou de *softwares* de terceiros instalados. Detalhes técnicos do ambiente de análise juntamente com a política de contenção de rede utilizada serão detalhados na seção 4.4.

Ao fim do ambiente de análise, para cada execução de cada amostra, será gerado um *dump* do tráfego de rede. Para cada *dump*, serão extraídas características em nível de *payload* e em nível do fluxo de dados de rede. Após a extração de características em ambos os níveis, é realizada a relação entre o fluxo de dados de rede juntamente com o *payload* de cada conexão. Na seção 4.5 será apresentada quais informações serão extraídas em ambos os níveis. Para cada conexão gerada durante a execução da amostra será selecionado um *parser* (dos implementados) para o protocolo utilizado na comunicação. A tarefa do *parser* é utilizar as características extraídas em nível de *payload* de forma individual/específica para cada protocolo. Ao fim dessa etapa, será construída uma sequência de elementos onde cada elemento representa uma conexão de rede (dentre todos as conexões) originada durante a execução da amostra. Cada elemento da sequência será representado por um *hash*, sendo este *hash* calculado pelos *parsers*.

Para cada *malware* é feita uma combinação dois a dois entre todas as suas execuções, onde, para cada combinação, é calculada a sequência de elementos mais longa (LCS – *Longest Common Substring*). O cálculo da LCS ocorre com base nas duas sequências de elementos referentes às duas execuções. Na subseção 4.6.1, são fornecidos detalhes de como os *parsers* para cada protocolo são implementados, além das informações necessárias para gerar as LCSs. Após obtidas todas as LCSs resultantes (dentre todas as combinações), para cada elemento (que representa uma conexão) de cada LCS, são recuperadas características referentes a conexão em ambos os níveis (*payload* e fluxo). Para algumas conexões são realizadas algumas etapas de pré-processamento, de modo a ajustar alguns descritores, que

serão detalhados na seção 4.6.2. Por fim, após geradas e atualizadas as LCSs, as mesmas são testadas com o objetivo de se descobrir quais possuem verdadeiro-positivo com ausência de falso-positivo. O processo completo em relação a validação das assinaturas está descrito na seção 4.7.

4.3 Seleção de amostras de *malwares*

Função da etapa **Seleção de amostras de *malwares***: Esta etapa inicia-se com a obtenção dos executáveis e conclui-se com a seleção dos *malwares* e suas respectivas famílias.

Como previamente descrito na seção 2.2, sobre sistemas de coleta de artefatos maliciosos, uma das formas de se obter amostras de *malwares* é por meio da utilização de repositórios públicos de *malwares*. Neste trabalho os *malwares* foram obtidos de repositórios públicos pelas seguintes motivações: fácil obtenção, quantidade e variedade[6] elevada. Embora tenha sido utilizado repositórios públicos de *malwares*, a metodologia elaborada neste trabalho não se limita a *malwares* provenientes de tais repositórios, podendo compreender também *malwares* provenientes de outros meios (verificar o modo passivo e ativo para obtenção de *malwares* na seção 2.2). Alguns exemplos de repositórios públicos, já previamente citados, são: VirusShare, VXHavens, Offensive Computing, VirusTotal.

Após a obtenção inicial das amostras, é feita uma consulta no site do VirusTotal através da API disponível[7]. A base de dados do VirusTotal é amplamente utilizada por diversos trabalhos [(ROSSOW et al., 2011), (KHARRAZ et al., 2016)] uma vez que a mesma fornece diversos indicadores a respeito do executável submetido. Uma das maneiras de verificar se a amostra possui relatório no VirusTotal é por meio da consulta de seu *hash* MD5. Caso exista um relatório associado a este *hash* MD5, o mesmo é

[6] Foi possível observar a variedade de famílias de *malwares* após o processo de classificação em famílias que será descrito em seguida.

[7] Para utilizar a API do VirusTotal é necessário uma chave publica que é fornecida mediante cadastro no próprio site do VirusTotal. Descrição da API - https://www.virustotal.com/pt/documentation/public-api/.

retornado. Neste relatório existem várias informações referentes ao executável, e dentre essas informações estão: a data e horário da primeira submissão[8] feita, a data e o horário da última análise, indicadores provenientes de análise estáticas e dinâmica, entre outras informações.

Além das informações previamente citadas, o relatório fornece também o rótulo (*label*) retornado por cada fabricante (*vendor*) de antivírus que classificou o executável como malicioso, além da quantidade de fabricantes que classificou o executável como malicioso dentro todos os fabricantes que analisaram a amostra. Diversos trabalhos [(SEBASTIÁN et al., 2016), (ROSSOW et al., 2012)] utilizam o relatório que é provido pelo VirusTotal para realizar uma triagem inicial das amostras, fazendo uma pré-seleção dessas antes do processo de análise. Duas triagens iniciais bastante utilizadas são:

1. Remoção de executáveis benignos: para tal, é utilizada a quantidade de fabricantes que detectou tal executável para saber se o mesmo é ou não malicioso, dado que é comum existir executáveis benignos em uma base pública de *malwares*. Um exemplo seria: caso um ou mais antivírus classifique o executável como malicioso, o mesmo é selecionado, caso contrário, o executável é considerado benigno. Em (ROSSOW et al., 2012) é discutido a importância de se remover arquivos benignos das amostras iniciais para evitar problemas com os resultados.
2. Obter os rótulos retornados pelos antivírus na tentativa de realizar uma pré-classificação do executável em questão. O rótulo aplicado depende da sintaxe que é emprega por cada antivírus. Comumente os rótulos possuem as seguintes informações: família do *malware*, sua variante e qual classe o *malware* pertence (as mesmas citadas na seção 2.1.3). O processo de *labeling* de *malware* é um problema recorrente (SEBASTIÁN et al., 2016) e bastante discutido uma vez que cada fabricante utiliza uma sintaxe própria, dificultando a padronização.

[8] Para obter a informação de data e horário da primeira submissão de um determinado executável foi necessário solicitar uma chave privada para a equipe de suporte do VirusTotal, que a concedeu por tempo limitado mediante justificava. O uso dessas informações será discutido em breve.

> Além disso, é muito comum haver divergência de informação entre os fabricantes em relação ao nome de família ou classe do *malware*.

No presente trabalho, a triagem inicial 1) é feita da seguinte forma: o executável é escolhido apenas se a porcentagem de antivírus que classificá-lo como malicioso ultrapassar 50%. É comum alguns trabalhos tomarem esta decisão com base na resposta de apenas um único antivírus. Porém, foi escolhido utilizar uma porcentagem de 50% na tentativa de se ter uma maior garantia de que o executável seja malicioso. Para resolver o problema apontado em 2), foi utilizado o trabalho chamado AVClass (SEBASTIÁN et al., 2016).

O AVClass é uma ferramenta disponibilizada em 2016 para a comunidade e tem como objetivo automatizar o processo de rotulação, obtendo a família mais provável a qual o *malware* pertença, a partir dos rótulos retornadas por todos os fabricantes de antivírus (presentes no VirusTotal) que classificaram o executável como malicioso. O AVClass realiza tal processo através da normalização dos rótulos, remoção de *tokens* genéricos e detecção de *alias* utilizados pelos fabricantes de antivírus.

Segundo (ROSSOW et al., 2012), é interessante que o conjunto inicial de amostras de *malwares* esteja balanceada entre as famílias presentes, evitando que haja predominância de *malwares* de certa(s) família(s) no conjunto inicial. No presente trabalho, após a classificação dos *malwares* em famílias, foi obtida uma quantidade fixa de *malwares* de cada família, dentre todas as famílias, para compor o conjunto inicial das amostras.

Além disso foi utilizada a informação de data e horário da primeira submissão proveniente do relatório do VirusTotal. Desta forma, foram selecionados apenas os *malwares* que foram submetidos no ano de 2016. É importante lembrar que, embora o *malware* tenha sido submetido no ano de 2016, não é garantido afirmar que o mesmo foi criado em 2016. O objetivo de se analisar *malwares* recentes é que existe uma chance maior da infraestrutura destes *malwares* ainda existir. Segundo (GRAZIANO; LEITA; BALZAROTTI, 2012), a análise de exemplares depois de semanas (ou dias) após a coleta do artefato pode resultar em uma análise imprecisa, uma vez

que a “infraestrutura” utilizada pelo *malware*, disponível na Internet, pode deixar de existir. Isso acontece porque as máquinas contactadas pelos *malwares* são voláteis por natureza, podendo ser desligadas por provedores ou em decorrência da aplicação de medidas legais ao serem detectadas atividades maliciosas (GRAZIANO; LEITA; BALZAROTTI, 2012).

4.4 Ambiente de análise

Função da etapa **Ambiente de análise**: Esta etapa inicia-se ao receber as amostras de *malware* classificadas por família e é concluída após gerar o tráfego de rede (*dump*) para todas as execuções de todas as amostras (de todas as famílias).

Para executar os testes necessários a essa pesquisa, será utilizado o ambiente esquematizada na Figura 4-2.

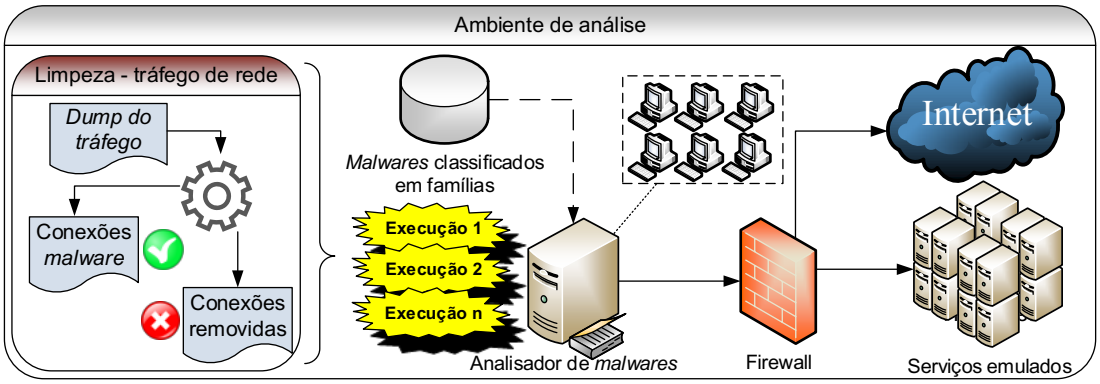


Figura 4-2. Ambiente de análise.

Após selecionado o conjunto inicial de amostras, as mesmas são submetidas para o ambiente de análise. No ambiente de análise, o *malware* será executado diversas vezes, com o objetivo de se obter semelhanças no comportamento de rede exercido pelo *malware* em cada execução, como realizado em (RAFIQUE; CABALLERO, 2013) e (RIECK et al., 2010). Ainda, as execuções são realizadas em diferentes máquinas virtuais (VM) com diferentes configurações. Como descrito em (SIKORSKI; HONIG, 2012) e (RIECK et al., 2010), alguns tipos de *malwares* produzem comportamentos

distintos ao serem executados em ambientes/configurações diferentes, uma vez que utilizam informações estáticas do sistema operacional para tomar decisões sobre quais funcionalidades executar e quais dados transmitir.

As amostras de *malwares* foram executados na Sandbox Cuckoo, que é responsável por realizar a análise dinâmica do artefato e capturar todas as ações executadas pelo mesmo no ambiente, incluindo as ações realizadas na rede. A Sandbox Cuckoo foi configurada para utilizar o *hypervisor* Qemu (QEMU, 2016) com a opção de KVM (*Kernel Virtual Machine*) ativada. Foram utilizadas 6 VMs Windows, sendo 2 Windows XP e 4 Windows 7. Para detalhes técnicos de versões e *softwares* instalados em cada uma das máquinas, verificar o Apêndice A deste documento. Embora tenha sido utilizado 6 VMs, a metodologia elaborada é independente da quantidade de VMs utilizada e do Sistema Operacional instalado em cada uma das VMs.

Em relação à politica de contenção de acesso a rede, algumas considerações foram avaliadas antes de se tomar a decisão de qual política seria utilizada. Em (GRAZIANO; LEITA; BALZAROTTI, 2012) são apresentadas duas abordagens: 1) comunicação em sua totalidade com a Internet; 2) uso de uma *honeynet* para emular alguns serviços comuns, que possuem vulnerabilidades conhecidas, fornecendo assim uma “infraestrutura” para o *malware*. A comunicação com a Internet referente à última abordagem, só ocorre quando o *malware* precisa realizar o *download* de algum binário, ou seja, apenas quando necessário. Segundo (GRAZIANO; LEITA; BALZAROTTI, 2012), a segunda abordagem possui vantagens em relação à contenção do comportamento do *malware* e, por isso, será utilizada nesta pesquisa.

De forma técnica e objetiva, a política de contenção utilizada foi a seguinte: conexões utilizando protocolo HTTP e DNS foram liberadas para a Internet, e as demais conexões foram encaminhas para o INetSim (INETSIM, 2016), *software* utilizado para fins de análise de *malware* [(SIKORSKI; HONIG, 2012), (JANG; WOO; BRUMLEY, 2013)], responsável por emular alguns serviços de rede. Além disso, foi feito o bloqueio de intercomunicação de rede entre as VMs, fazendo com que cada máquina opere de forma isolada (apenas podendo se comunicar com a Internet ou com os serviços emulados).

Na tentativa de evitar ataques do tipo DoS (*Denial of Service*), foi realizado um controle de banda para cada uma das VMs. A limitação foi feita em nível de *interface* de rede uma vez que cada VM utiliza uma *interface* própria[9]. Além de tentar evitar ataques do tipo DoS, o objetivo dessa limitação foi garantir que, independente da existência ou não de uma análise ocorrendo em paralelo (dado que existem 6 VMs fazendo a análise em paralelo), a banda disponível para cada VM será sempre a mesma. Em outras palavras, essa limitação garante que uma VM não "roube" banda de outras VMs.

Por padrão, a Sandbox Cuckoo fornece um *dump*, no formato pcap, do tráfego de rede gerado pelo *malware* durante o período de análise. Esse *dump* do tráfego de rede é gerado pela ferramenta tcpdump (TCPDUMP, 2016). Este arquivo criado contém todas as conexões de rede que ocorreram durante o período de análise da amostra, resultando na fonte de dados que será utilizada na metodologia desenvolvida neste projeto. Segundo (ROSSOW et al., 2011), aproximadamente 44% dos *malwares* de fato produzem tráfego de rede. Já os demais, não produzem pelos seguintes motivos: 1) são arquivos inválidos/corrompidos; 2) operam apenas no sistema local (encriptação de disco), sem produzir atividade de rede; 3) necessitam de interação do usuário; ou 4) utilizam técnicas de anti-análise e finaliza sua execução. Portanto, para as análises que produziram tráfego de rede, foi feita uma limpeza[10] no *dump* de rede gerado, na tentativa de deixar apenas o tráfego de rede originado pelo *malware* executado. Dessa forma, para cada amostra irá existir *n dumps* de tráfego de rede, sendo *n* a quantidade de vezes que uma amostra foi executada.

[9] Foi feita uma limitação de 1.5MBps para cada VM. No pior caso o consumo total da banda será de 9MBps (somando o consumo de todas as 6 VMs).

[10] No Apêndice B é feita uma explanação a cerca de como pode ocorrer a limpeza do tráfego de rede. Além disso, são fornecidos detalhes de como a mesma foi realizada neste trabalho.

4.5 Extração de característica

Função da etapa **Extração de característica**: Esta etapa é iniciada ao receber o tráfego de rede de cada execução das amostras e finaliza-se emitindo os descritores de *payload* e fluxo de dados de rede.

A partir do *dump* do tráfego de rede obtido após a análise de cada amostra, algumas características das conexões são extraídas em ambos os níveis: *payload* e fluxo[11]. Na Figura 4-3 é possível observar o fluxograma do esquema de extração de característica.

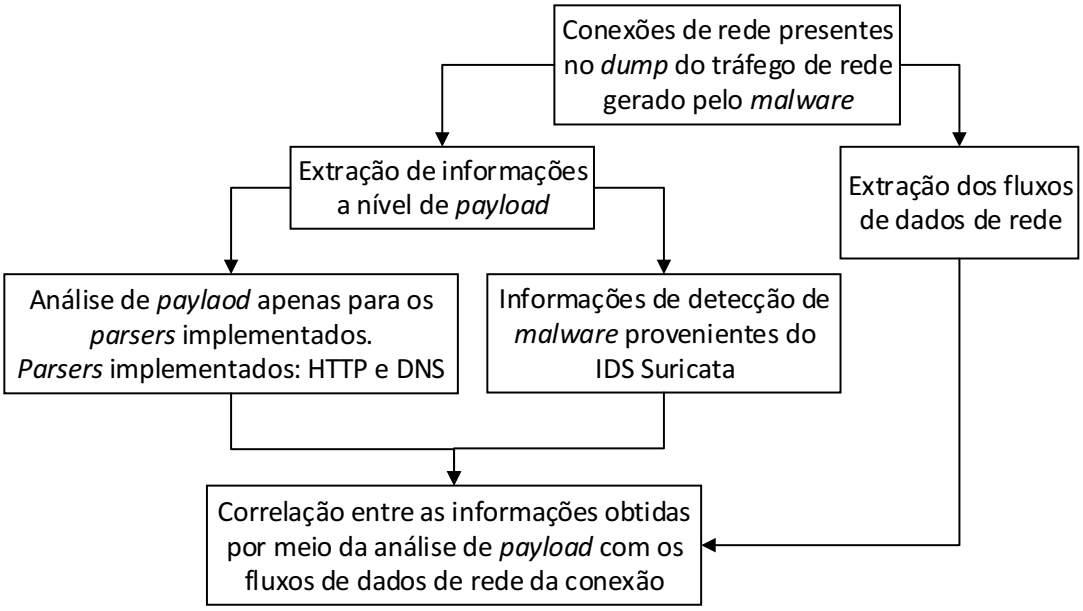


Figura 4-3. Esquema lógico do processo de extração de características a partir do tráfego de rede gerado a partir da execução de cada amostra.

Para extração de características em nível de fluxo, foi utilizada a ferramenta Argus (QOSIENT, 2016), e para a extração em nível de *payload* foi utilizada a ferramenta Bro. Em (BAYER et al., 2009) é utilizada a ferramenta Bro para extrair informações dos seguintes protocolos: HTTP, SMTP, IRC e FTP. Como é possível observar na Figura 4-3, o processo de extração de característica se repete para cada conexão encontrada no arquivo *dump*. Para

[11] Quando utilizada ao longo do texto, a palavra “fluxo” faz referência a “fluxo de dados de rede”.

cada conexão, é checado se a extração em nível de *payload* foi implementada. Caso tenha sido implementada, são extraídas informações em nível de *payload* e fluxo, caso contrário, apenas informações de fluxo são extraídas. No caso onde é feita a extração de características em nível de *payload*, é realizado o relacionamento entre essas características e as características extraídas em nível de fluxo. Dessa forma, a partir de um fluxo é possível recuperar características extraídas em nível de *payload*.

Para este trabalho, foram extraídas características em nível de *payload* dos seguintes protocolos: HTTP e DNS. A motivação para escolher ambos os protocolos foi originada após observar que diversas pesquisas apontam a predominância do uso de ambos os protocolos por *malwares*, como é possível observar nas seguintes pesquisas: (ROSSOW et al., 2011) e (NELSON, 2016). Além das características em nível de *payload*, foi utilizado o NIDS Suricata em conjunto com o conjunto de assinaturas de rede para *malwares* do *Emerging Threats* (EMERGINGTHREATS, 2016) para extrair características de severidade dos alertas emitidos pelo Suricata.

O objetivo em se utilizar o Suricata foi tentar detectar algumas características presentes nas conexões, descritas por meio de assinaturas. A título de exemplo, essas características podem ser: uma sequência específica de *bytes* para um determinado protocolo ou uma *string* específica no cabeçalho do protocolo HTTP. Após detectada tal característica, o Suricata emite um alerta informando a detecção. A partir deste alerta é possível correlacionar o título e a severidade do alerta com o fluxo da conexão correspondente, fornecendo, desta forma, mais informações a respeito de um determinado fluxo.

Em relação aos campos presentes no fluxo, foram extraídos os campos apresentados na Tabela 4-1, que são utilizados para representar o fluxo em formato bidirecional. Apenas alguns deles serão utilizados para compor os passos[12] de uma assinatura, como será discutido na seção 4.6.2.

[12] O mesmo conceito de “passos” utilizado em (CORRÊA, 2009) será utilizado neste trabalho. Em (CORRÊA, 2009) foi elabora um modelo de rastreamento de fluxos, onde cada passo de uma assinatura corresponde a uma característica causada por um evento a ser detectado. Por exemplo, se um evento realiza *n* conexões, uma assinatura pode conter *n* passos, responsáveis por detectar cada uma das conexões.

Tabela 4-1. Campos do fluxo bidirecional extraídos.

Campos	Descrição
stime	*Timestamp* inicial do fluxo em Unix *time*
ltime	*Timestamp* final do fluxo em Unix *time*
saddr	Endereço IPv4 de origem
sport	Porta de origem
daddr	Endereço IPv4 de destino
dport	Porta de destino
proto	Protocolo da camada de transporte
spkts	Quantidade de pacotes enviados pela origem
sbytes	Quantidade de *bytes* enviados pela origem
sappbytes	Apenas a quantidade de *bytes* da camada de transporte enviadas pelo destino
dpkts	Quantidade de pacotes enviados pelo destino
dbytes	Quantidade de *bytes* enviados pelo destino
dappbytes	Apenas a quantidade de *bytes* da camada de transporte enviadas pelo destino

As características extraídas do protocolo DNS estão presentes na Tabela 4-2, enquanto as do protocolo HTTP estão presentes na Tabela 4-3. Para o protocolo DNS, foi extraído o *timestamp* inicial da requisição para que fosse possível correlacioná-lo com o seu fluxo. Além disso foi extraído o nome do domínio requisitado e a resposta obtida.

Tabela 4-2. Campos extraídos do protocolo DNS.

Campos	Descrição
stime	*Timestamp* inicial da requisição DNS
dns_query	Domínio requisitado
dns_response	Informação retornada na resposta

Do protocolo HTTP foram obtidas as características de *timestamp* inicial do protocolo TCP, além do identificador da sessão HTTP e, caso existam várias requisições HTTP dentro de uma mesma sessão HTTP, as mesmas

são identificadas por um índice indicando a profundidade da requisição, informação esta presente no campo *http_depth*. Quando existem múltiplas requisições em uma mesma sessão HTTP, a mesma é referenciada como conexão persistente, caso contrário é chamada de conexão não persistente (KUROSE; ROSS, 2013).

Tabela 4-3. Campos extraídos do protocolo HTTP.

Campos	Descrição
stime_tcp	*Timestamp* inicial da requisição HTTP
http_stream_id	Identificador da sessão HTTP
http_depth	Profundidade da requisição dentro da sessão HTTP
http_request_line	Linha de requisição (ou *request-line*) do protocolo HTTP
http_header_fields[]	Vetor contendo cada linha de cabeçalho (ou *header fields*) do protocolo HTTP

O campo *http_request_line* contém a linha de requisição, presente na primeira linha do protocolo HTTP. Essa linha possui três campos: método utilizado, a URL (*Uniform Resource Locator*) requisitada e a versão do protocolo HTTP utilizado. Diversos trabalhos [(NELSON, 2016); (PERDISCI; LEE; FEAMSTER, 2010), (PERDISCI; ARIU; GIACINTO, 2013)] utilizam a linha de requisição do HTTP para realizar o processo de *clustering* com o objetivo de encontrar semelhança entre as requisições e, por consequência, classificar essas requisições em grupos que podem ser utilizados para encontrar famílias de *malwares*.

O campo *http_header_fields[]* é um vetor contendo todas as linhas de cabeçalho, onde cada linha possui o nome do campo e seu valor, ambos separados entre si pelo caractere ":". Alguns exemplos de linhas de cabeçalho HTTP: "*USER-AGENT: Microsoft-CryptoAPI/6.1"*, *"CONNECTION: Keep-Alive"* e "HOST: www.googletagservices.com".

A primeira linha (de requisição) e as demais linhas do cabeçalho HTTP são separadas entre si por meio dos caracteres *carriage return* (CR) e *line*

feed (LF). A última linha do cabeçalho HTTP é seguida de um comando adicional de CR/LF. Para detalhes acerca do protocolo HTTP, consultar o RFC 1945 (BERNERS-LEE; FIELDING; FRYSTYK, 2016) e o RFC 2616 (FIELDING et al., 2016).

4.6 Geração de assinatura

Função da etapa **Processo de Geração de Assinatura**: Esta etapa inicia-se recebendo os descritores de *payload* e fluxo e finaliza-se emitindo as assinaturas criadas.

Este processo se divide em dois sub-processos, que são responsáveis por encontrar similaridades no tráfego de rede dentre todas as execuções de uma determinada amostra, e, após encontradas, é criada uma assinatura com o objetivo de detectar tais conexões que ocorrem entre as execuções. A etapa **similaridade entre execuções** utiliza características de *payload* e fluxo, e a etapa **criação de assinatura** irá produzir descritores para serem utilizados em assinaturas baseadas em fluxo. Embora as assinaturas criadas por esta metodologia utilizem apenas descritores presentes em fluxo, são utilizadas informações em nível de *payload* apenas para orientar e tornar o método de geração de assinatura mais preciso, dado que as informações de *payload* não estarão contidas nas assinaturas.

4.6.1 Similaridade entre execuções

Para que seja possível encontrar similaridade nas conexões entre as execuções do mesmo *malware*, antes foi calculado um *hash* para cada conexão de cada execução, para posteriormente ser realizada a busca por similaridade entre as execuções. Para geração do *hash*, algumas informações são utilizadas dependendo do tipo da conexão, como será descrito mais adiante.

A análise de todas as amostras é passada para a função `Hash-Amostras` (Figura 4-4), que irá iterar sob todas as execuções de todas as amostras e irá passar as conexões de cada amostra para a função `Hash-Calc` (Figura 4-5).

```
Hash-Amostras(amostras)
1     for amostra in amostras
2         execucoes = amostra.execucoes
3            for execucao in execucoes
4                 Hash-Calc(execucao.conexoes)

5            combinacoes = Obter-Combinacao(execucoes)
6            for combinacao in combinacoes
7                execucao_conexoes1 =
                 execucoes[combinacao.indice1].conexoes

8                execucao_conexoes2 =
                 execucoes[combinacao.indice2].conexoes

9                if Existe-Conexao-HTTP(execucao_conexoes1,
                 execucao_conexoes2)
10                   Resolver-Hashes-HTTP(execucao_conexoes1,
                     execucao_conexoes2)
11               hashes1, hashes2 = Checar-Sequencia(conexoes1.hash,
                                                     conexoes2.hash)
12               LCSs = LCS(hashes1, hashes2)
13               Todos_LCSs = Contabilizar-LCSs(Todos_LCSs, LCSs)
```

Figura 4-4. Algoritmo principal que irá iterar sob todas as conexões de todas as amostras e irá criar as LCS.

A função `Hash-Calc` possui a tarefa de verificar se uma determinada conexão possui análise em nível de *payload*, e, caso possua, o *hash* da conexão é calculado com base em informações obtidas em nível de *payload*. Caso a conexão não possua análise baseada em *payload*, são obtidas informações em nível de fluxo para realizar o cálculo do *hash*.

```
Hash-Calc(conexoes)
1     for conexao in conexoes
2         if Possui-Analise-Payload(conexao)
3             switch Protocolo-Conexao(conexao)
4                 case "DNS":
5                     conexao.hash = Hash-DNS(conexao)
6                 case "HTTP":
7                     conexao.hashes = Hash-HTTP(conexao)
8                 default:
9                     return NULL
10        else
10            conexao.hash = Hash-Fluxo(conexao)
```

Figura 4-5. Algoritmo que irá selecionar a função correta para o cálculo de *hash* para cada conexão.

Quando não há análise de *payload* para a conexão, são obtidas as seguintes informações do fluxo da conexão: destino, porta de destino e protocolo. A informação de destino foi obtida da seguinte forma: caso exista alguma requisição DNS que tenha tido como resposta o endereço IP utilizado na conexão, o domínio é considerado o destino, caso contrário o endereço IP da conexão é considerado o destino. Verificar a função `Hash-Fluxo` no Apêndice C deste documento.

Como foram extraídas informações em nível de *payload* para os protocolos DNS e HTTP, quando uma conexão possuir análise via *payload*, a forma de calcular o *hash* vai depender de um desses dois protocolos utilizados. Para realizar a análise via *payload* para outros protocolos, como o protocolo SMTP por exemplo, basta adicionar uma função que irá calcular o *hash* para o protocolo específico na função `Hash-Calc`. O *framework* elaborado nesta pesquisa foi projetado para trabalhar de forma modular, facilitando a inserção de análise de *payload* para outros protocolos.

Para calcular o *hash* de conexões DNS é utilizado apenas o nome do domínio requisitado. Para o protocolo HTTP são utilizadas as informações: 1) linha de requisição; 2) linha de cabeçalho que inicia-se com “HOST:”. Ainda, foi calculado um *hash* para a linha de requisição sem os valores dos parâmetros e outro *hash* para a linha de requisição na sua forma original (com

parâmetros e valores). Portanto, caso a requisição HTTP possua parâmetros, dois *hashes* serão criados, caso contrário, apenas o *hash* da linha de requisição inteira é criado (o *hash* referente a requisição sem os valores dos parâmetros permanece nulo).

Os dois *hashes* para o protocolo HTTP foram utilizados uma vez que os *malwares* podem utilizar os parâmetros da requisição HTTP para passar valores, realizando assim a troca de informações com os nós envolvidos na comunicação. Sendo assim, caso o(s) valor(es) de(os) parâmetro(s) mudem entre as execuções do *malware*, o *hash* calculado para a linha de requisição sem os valores não irá mudar (apenas o *hash* da requisição na sua forma original irá mudar). Dessa forma, é possível encontrar a mesma requisição em outra execução do *malware* através do *hash*. As funções `Hash-DNS` e `Hash-HTTP` presentes no Apêndice C deste documento detalham o funcionamento para cada um dos protocolos descritos acima.

Como descrito anteriormente, para cada conexão HTTP podem existir um ou dois *hashes* gerados a partir da requisição HTTP. Entretanto, como descrito na seção 4.5, é possível que existam conexões persistentes que possuem múltiplas requisições em uma mesma sessão HTTP. Para tal, foram concatenados apenas os *hashes* referentes a linha de requisição na sua forma original, e, a partir de então um novo *hash* foi gerado. O mesmo procedimento foi feito para as linhas de requisição sem os valores dos parâmetros. Ao final, a conexão HTTP persistente irá possui um ou dois *hashes* da mesma forma que uma conexão HTTP não persistente.

Após calculado o *hash* para cada conexão de todas as execuções de cada amostra, é possível obter todos os *hashes* de todas as conexões de uma determinada execução. Com a obtenção de todos esses *hashes* é formada uma sequência respeitando a ordem na qual as conexões ocorreram. Em outras palavras, para cada execução será produzida uma sequência formada de um ou mais *hashes*, caso a execução produza atividade de rede. Após produzido a sequência de elementos para a execução, será realizada uma combinação dois a dois entre todas as execuções de cada amostra. Desta forma, por exemplo, se houver 3 execuções de uma determinada amostra, teremos as seguintes combinações: execução 1 com execução 2, execução

1 com execução 3 e execução 2 com execução 3. Sendo assim, a sequência de elementos de uma execução será testada com a sequência de elementos de todas as outras execuções.

Portanto, para cada combinação, existirão duas sequências da seguinte forma: $X = \langle hash1_1, hash1_2, \dots, hash1_m \rangle$ e $Y = \langle hash2_1, hash2_2, \dots, hash2_n \rangle$. As sequências X e Y serão obtidas de acordo com a combinação selecionada. A partir de ambas as sequências X e Y, é feita a remoção dos elementos que não aparecem em ambas as sequências (linha 11), ajudando a diminuir o tempo gasto pelo algoritmo LCS. Ambas as sequências serão fornecidas para o algoritmo LCS (*Longest Common Substring*), (linha 12) para que o mesmo realize os procedimentos descritos na seção 2.5. Após a execução do algoritmo LCS, o mesmo irá retornar a subsequência comum mais longa, $Z = \langle hash1\_2_1, hash1\_2_2, \dots, hash1\_2_k \rangle$, com base nas sequências X e Y. Logo, para cada combinação, será produzida uma ou mais LCSs, sendo necessário guardá-las até que todas as combinações sejam testadas. Ao fim do algoritmo `Hash-Amostras`, é possível que existam diversas LCSs, dependendo da quantidade de sequências e combinações resultantes (linha 13).

Na Figura 4-6 é ilustrado o procedimento descrito, onde o algoritmo LCS recebe duas sequências diferentes como entrada, de tamanho 8 e 6, gerando duas LCSs de tamanho 4. Na mesma figura, é possível verificar que cada uma das conexões presentes nas LCSs geradas (LCS 1 e LCS 2), ocorrem na mesma ordem cronológica se comparadas com a sequência de tamanho 8 e 6. A partir das LCSs geradas, serão extraídos os descritores responsáveis por identificar cada um dos fluxos de cada conexão, possibilitando desta forma criar uma assinatura, como será descrito na seção 4.6.2.

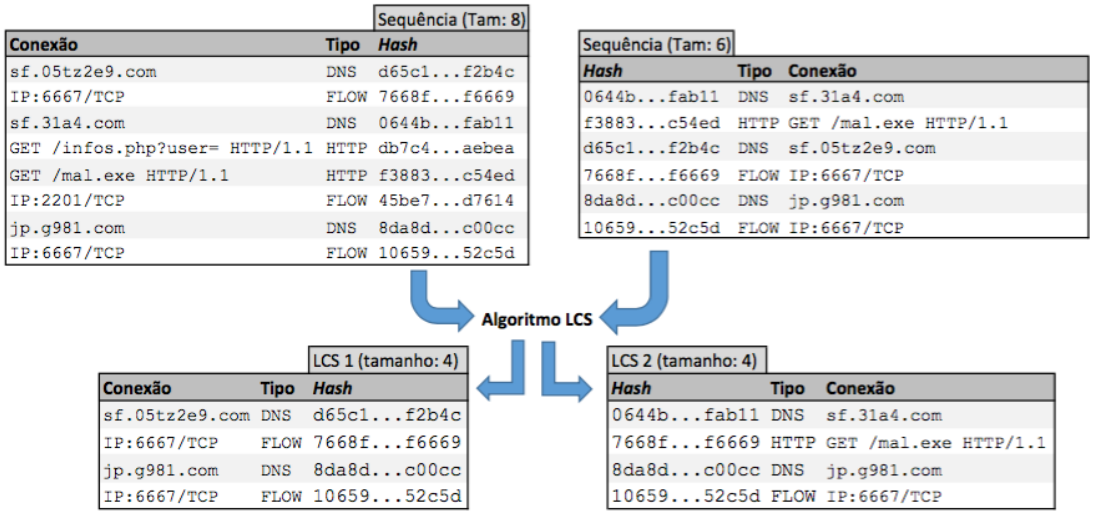


Figura 4-6. Ilustração do processo de criação das LCSs.

Para cada *malware* pode haver a produção de múltiplas LCSs, que, dentre as geradas, foram escolhidas as que aparecem em um maior número de execuções. Por exemplo, na combinação entre a execução 1 e 3 de um *malware* foram geradas três LCSs diferentes (LCS X, LCS Y e LCS Z), e na combição entre a execução 2 e 3 foram geradas as LCSs (LCS X e LCS Y). Desta forma, a LCS X e a LCS Y, apareceram em três execuções (1, 2 e 3), enquanto a LCS Z aparece em apenas duas execuções (1 e 3). Desta forma, foram escolhidas apenas as LCSs que apareceram no maior número de execuções, que no caso foram as LCSs: LCS X e LCS Y.

Como para cada requisição HTTP existe a possibilidade de se criar dois *hashes* como descrito anteriormente, é necessário escolher qual dos dois *hashes* será utilizado para a conexão. A escolha do *hash* dependerá da combinação em questão e ocorrerá na linha 10 do algoritmo `Hash-Amostras` (Figura 4-4), através da chamada da função `Resolver-Hashes-HTTP`. Em linhas gerais, se houver duas requisições iguais (que utilizem parâmetro/valores) entre duas execuções de uma mesmo amostra, é utilizado o *hash* da linha de requisição. Caso haja diferença de valores nos parâmetros da requisição entre duas execuções diferentes, é utilizado o *hash* da linha de requisição sem os valores. Para maiores detalhes e algumas exceções do funcionamento do algoritmo, checar a função `Resolver-Hashes-HTTP` presente no Apêndice C deste documento.

4.6.2 Criação de assinatura

Nesta sub-etapa serão recebidas as LCSs pré-selecionados na etapa anterior, e, a partir de então, serão recuperadas as conexões (a partir dos *hashes*) que foram selecionadas. Para conexões do tipo DNS e Flow, a recuperação dos descritores que serão utilizados em nível de fluxo seguem algumas etapas simples. Para conexões do tipo HTTP, antes de retornar os descritores, é necessário realizar um pré-processamento como será descrito mais adiante. Os descritores que podem (dependendo do tipo da conexão) ser utilizados para compor cada passa da assinatura estão presentes na Tabela 4-4.

Tabela 4-4. Descritores utilizados em cada passo da assinatura.

Campos	Descrição
daddr	Endereço IPv4 de destino
dport	Porta de destino
proto	Protocolo da camada de transporte
sappbytes	Apenas a quantidade de *bytes* da camada de transporte enviadas pelo destino
comparador	Comparador, pode assumir um dos dois valores: “igual” ou “maior igual”

Como discutido anteriormente, os descritores de cada conexão serão utilizados para compor um determinado passo da assinatura. Dessa forma, a partir de todas as conexões resultantes serão criados todos os passos da assinatura para uma determinada amostra de *malware*. A seguir será descrito quais descritores são obtidos a partir de cada tipo de conexão:

- **Tipo Fluxo**: protocolo e porta de destino. O descritor de endereço IP de destino é utilizado no passo da assinatura apenas se não existir uma resposta DNS com o endereço IP em questão. A título de exemplo, a seguir será exemplificado algumas das razões pelas quais um *malware* não realiza requisições DNS para se obter o endereço IP: o *malware* já possui o endereço IP de destino no binário; o *malware* descobriu o seu

endereço IP público e realiza uma prospecção de rede na sub-rede que abrange o seu enderço IP; ou ainda, o *malware* recebeu o endereço IP de uma C&C. Os descritores de comparador e de *bytes* enviados são nulos para esse tipo de conexão uma vez que não há análise de *payload*.

- **Tipo DNS**: protocolo, porta de destino, comparador e quantidade de *bytes* enviados. O endereço IP de destino é utilizado apenas nos casos em que é feita a requisição DNS para um IP diferente do endereço IP do servidor DNS local. No caso de conexão DNS, o comparador utilizado é “igual” dado que não há mudança na quantidade de *bytes* entre uma execução e outra, já que o *hash* é o mesmo em ambas as execuções.
- **Tipo HTTP**: protocolo, porta de destino, comparador e quantidade de *bytes*. O descritor de endereço IP de destino é utilizado no passo da assinatura apenas se não existir uma resposta DNS com o endereço IP em questão. O comparador e a quantidade de *bytes* utilizada na assinatura irão depender de uma análise feita de forma automatizada, como será descrito mais adiante.

Embora seja possível extrair outros descritores (além dos descritos na Tabela 4-4), optou-se por não extrair alguns dos seguintes descritores pelos seguintes motivos:

- Quantidade de *bytes* enviados/recebidos (*sbytes*/*dbytes*) e quantidade de pacotes enviados/recebidos (*spkts*/*dpkts*): caso haja retransmissões de pacotes durante a execução do *malware*, tais valores podem mudar, não fornecendo estabilidade para serem utilizados nos passos das assinaturas.
- Quantidade de *bytes* da camada de transporte enviadas pelo destino (*dappbytes*): não foi utilizado este campo uma vez que é comum o tamanho da resposta mudar mediante atualizações feitas no lado do servidor. Exemplo: um binário ou uma página *web* que é requisitada pode sofrer alterações, modificando assim a quantidade de *bytes* de resposta.

Após obter os descritores dos tipos de conexões mencionados anteriormente, será necessário realizar um pré-processamento especificamente para o protocolo HTTP. Segundo (SIKORSKI; HONIG, 2012) é comum que um *malware* utilize alguns campos das linhas de cabeçalho do protocolo HTTP para transferir algumas informações, como é o caso do campo de *User-Agent*. A utilização de tais campos das linhas de cabeçalho pode resultar no uso de diferentes valores entre uma execução e outra, acarretando em um acréscimo ou decréscimo na quantidade de *bytes* resultantes no fluxo de dados de rede da conexão. Para exemplificar, considere: 1) um *malware* é executado em um Sistema Operacional Windows 7 e utiliza o valor "Windows7SP2" no campo *User-Agent*; 2) o mesmo *malware* é executado no sistema operacional Windows XP e agora o valor utilizado no campo *User-Agent* é "WindowsXPSP1". Embora a diferença entre uma requisição e outra (em diferentes execuções) tenha sido de apenas um único *byte* no tamanho da *string* informada, criar uma assinatura que utilize uma quantidade fixa de *bytes* não irá detectar algumas comunicações do *malware*.

Para evitar esse tipo de problema, a solução utilizada foi identificar quando ocorre cada um dos casos descritos adiante, e, dependendo do caso, realizar uma subtração na quantidade de *bytes* quando necessário. Inicialmente são obtidas todas as requisições iguais (mesmo *hash*) de todas as execuções de cada amostra e então é feita a verificação das linhas de cabeçalho. Foram mapeados três casos que podem ocorrer se comparada duas requisições iguais entre execuções diferentes. Para cada caso será fornecido um exemplo e uma solução para resolver o problema:

- **Caso 1:** Linha do cabeçalho (campo e valor) não muda entre duas requisições iguais (mesmo *hash*).
    - Exemplo: ambas as requisições utilizam "*User-Agent*:*downloader-exe*".
    - **Solução:** não há remoção de *bytes* para este caso.

- **Caso 2:** Apenas o valor muda entre duas requisições iguais (mesmo *hash*).

  - Exemplo: como citado anteriormente, "*User-Agent*:*WindowsXPSP1*" para Windows XP e "*User-Agent*:*Windows7SP2*" para Windows 7.
  - **Solução:** remover apenas a quantidade de *bytes* referente ao valor da linha de cabeçalho de cada requisição, ou seja, para o exemplo citado haverá a remoção de 12 *bytes* e 11 *bytes,* respectivamente.
- **Caso 3:** O campo da linha de cabeçalho não existe em todas as requisições iguais (mesmo *hash*).
  - Exemplo: utiliza "*User-Agent*: *Windows XP*" se executado no Windows XP mas não utiliza "*User-Agent*:" se executado no Windows 7.
  - **Solução:** remover a quantidade de *bytes* referente a linha de cabeçalho (campo e valor) de todas as requisições que possuem tal linha de cabeçalho.

Após o processo descrito, caso haja remoção de *bytes* para uma determinada requisição HTTP, é feita a subtração da quantidade total de *bytes* da conexão e o comparador utilizado será "maior igual". Caso não haja remoção de *bytes* entre as requisições, a quantidade de *bytes* se mantém inalterável e o comparador utilizado será "igual". Ainda, existem outros dois casos nos quais são utilizados o comparador "maior igual": 1) *hash* selecionado é referente a uma requisição sem os valores de parâmetro – logo é provável que haja diferentes valores entre diferentes execuções; 2) requisição HTTP utiliza o método POST, onde é possível enviar dados no corpo do protocolo HTTP. Como não foi feita análise deste conteúdo, a quantidade de *bytes* referente ao conteúdo da requisição POST foi removido.

4.7 Validação das assinaturas

Função da etapa **Validação das assinaturas**: Esta etapa é iniciada ao receber as assinaturas geradas e finaliza-se após selecionar as assinaturas que possuem verdadeiro-positivo e não possuem falso-positivo.

Para avaliar a qualidade das LCSs selecionadas, e, por consequência, transformá-las em assinaturas as que apresentam melhores resultados, foi utilizado métricas que são usadas para avaliar os métodos baseados em detecção por anomalias (WU; BANZHAF, 2010). Quando o tráfego de rede de um *malware* for detectado corretamente por uma LCS, tem-se um verdadeiro-positivo (*True Positive* – TP), caso contrário tem-se um falso-negativo (*False Negative* – FN). Caso uma LCS criada para um *malware* classifique um fluxo normal como sendo tráfego malicioso, tem-se um falso-positivo (*False Positive* – FP), caso contrário tem-se um verdadeiro-negativo (*True Negative* – TN). Diferentemente da utilização de tais métricas no contexto de Sistemas de Detecção de Intruso (utilizadas para avaliar a taxa de detecção), tais métricas possuem um papel diferente neste trabalho. Ao utilizar tais métricas, espera-se selecionar as melhores LCSs e descartar as piores. As melhores LCSs serão consideradas assinaturas de rede para os respectivos *malwares*.

Após criada as LCSs, serão testadas em uma base de dados contendo fluxos normais, provenientes de um ambiente isolado[13]. Para cada passo de uma assinatura é feita uma busca no banco de fluxos informando os descritores presentes no passo da assinatura. Caso a busca retorne resultados, significa que houve uma correspondência entre os descritores do passo da assinatura e os fluxos presentes no banco de fluxos. A partir do resultado retornado, é obtido o menor tempo (*stime*) dentre os fluxos retornados, para que a próxima busca possa ser realizada, tendo como início o menor tempo retornado na busca anterior. O menor tempo (*stime*) é utilizado a partir do segundo passo para garantir a ordem cronológica de ocorrência

[13] O tráfego de rede obtido foi referente à rede do laboratório ACME!, laboratório onde esta pesquisa foi desenvolvida.

dos passos, em outras palavras, é possível garantir que o $passo_x$ ocorre depois do $passo_{x-1}$.

O processo descrito se repete até que uma das duas condições sejam atingidas, e, dependendo da condição, a assinatura é caracterizada com a existência de falso-positivo, são elas: 1) algum passo da assinatura não retorna resultado, ou seja, esta assinatura é considerada um verdadeiro-negativo; 2) todos os passos da assinatura são testados, e, o último passo retorna resultado – portanto esta assinatura possui falso-positivo.

Para os *malwares* onde houve a criação de assinatura, os mesmos foram executados novamente para que fosse possível realizar os testes com as LCSs criadas. Esse teste tem como objetivo avaliar se uma LCS possui verdadeiro-positivo. Após realizada a nova execução do *malware* no ambiente de análise (Figura 4-2), foi feita a limpeza do *dump* do tráfego de rede, para, posteriormente, obter-se apenas as conexões de rede geradas pelo *malware* durante sua execução.

Para verificar se uma LCS possui verdadeiro-positivo, a seguinte abordagem foi utilizada: todo o tráfego de rede produzido, durante a nova execução do *malware*, foi transformado em fluxo. Em seguida, todos os passos de cada uma das LCSs (produzidas para o *malware* durante o processo de geração de assinatura) foram testados nos fluxos de rede referentes à nova execução deste *malware*. Se os descritores respectivos a cada passo encontrarem uma correspondência com os fluxos testados, significa que a assinatura é apta a detectar as atividades de rede geradas pelo *malware*, sendo considerado um verdadeiro-positivo, caso contrário a assinatura é considerada um falso-negativo.

Portanto, ao fim da etapa de validação de assinatura, para cada LCS que possuir verdadeiro-positivo com ausência de falso-positivo, a mesma estará apta a ser transformada em assinatura.

CAPÍTULO 5 - Testes e Resultados

5.1 Considerações iniciais

Neste capítulo são apresentados os experimentos realizados para validar a metodologia proposta e os resultados obtidos. De modo a validar a metodologia proposta, para os *malwares* que produziram atividade de rede, foram geradas LCSs com o objetivo de identificar tais atividades de rede. Em seguida, as LCSs foram testadas para identificar quais detectavam a nova execução do *malware* e quais possuíam falso-positivo, removendo essas últimas. Na seção 5.2 são apresentados os resultados acerca dos exemplares de *malwares* obtidos e selecionados para serem executados. Na seção 5.3 é possível verificar a quantidade de *malware* que produziu tráfego de rede dentre as famílias de *malwares* que foram executadas. Na seção 5.4 são fornecidas algumas estatísticas em relação as características extraídas das atividades de rede exercidas pelos *malwares* durante suas execuções. Na seção 5.5 são apresentados os resultados obtidos com o processo de geração de assinaturas. Na seção 5.6, para as LCSs selecionadas, são feitos alguns testes seguindo as métricas apresentadas na seção 4.7. Por fim, na seção 5.7, são feitas as considerações finais sobre os resultados obtidos.

5.2 Exemplares analisados

Como discutido na seção 4.1, foram utilizados *malwares* provenientes de repositórios públicos. Além disso, foram obtidos os relatórios de análise retornados pelo VirusTotal e utilizados no AVClass para realizar a triagem inicial das amostras. O repositório público utilizado foi o VirusShare devido a grande quantidade de amostras de *malwares*, além da facilidade e rapidez para se obtê-los. Dado a grande quantidade de amostras, foram obtidos apenas os arquivos disponíveis pelo VirusShare entre as datas 05/07/2016 e 16/10/2016. Foram obtidos 12 arquivos compactados compreendendo um total de 292GB de *malwares*[14].

Os arquivos compactados resultaram em 367.664 amostras de executáveis para plataforma Windows, que, após a triagem inicial, resultou em 166.271. Como explicado na seção 4.2, a triagem inicial envolve a seleção dos arquivos que foram vistos pela primeira vez pelo VirusTotal no ano de 2016 e que a quantidade de antivírus que realizou a detecção excede 50%.

Para todos os relatórios retornados do VirusTotal, o AVClass obteve 3.001 famílias diferentes de *malwares*, porém, grande parte das famílias possuem poucos exemplares diferentes de *malwares*[15]. Além disso, das 3.001 famílias, há três famílias (*sytro*, *multiplug* e *hotbar*) que juntas possuem 107.831 (29,32%) *malwares*. Para evitar a escolha de famílias que possuam poucas amostras, e evitar que haja predominância de *malwares* de determinada(s) família(s), como discutido na seção 4.3, o seguinte procedimento foi realizado: de todas as famílias presentes, foram escolhidas 200 famílias que possuem no mínimo 10 *malwares* diferentes. Das 200 famílias escolhidas, foram obtidas 10 amostras de *malwares* diferentes de cada uma, resultando em um total de 2.000 *malwares*[16]. Desta forma, o conjunto inicial das amostras de *malwares* estão balanceadas entre as 200 famílias selecionadas.

[14] Os arquivos obtidos foram todos entre os índices entre 00258 e 00269. Exemplo do nome inteiro do arquivo: VirusShare_00258.zip.

[15] 62,11% (1.864) das famílias possuem menos de quatro *malwares* diferentes.

[16] As famílias e os respectivos *hashes* dos *malwares* estão disponíveis em: http://pastebin.com/sqF7f5q8

5.3 Ambiente de análise

Após obtidos os 2.000 *malwares*, cada um foi executado 5 vezes no ambiente de análise, resultando em 10.000 execuções. Uma vez que esta pesquisa envolve achar similaridade entre diferentes execuções, houve a necessidade de se executar uma amostra diversas vezes. Em (RIECK et al., 2010) os *malwares* são executados múltiplas vezes com o mesmo objetivo. O motivo pelo qual foi escolhido 5 execuções, foi visando gerar um número significativo de combinações entre as execuções, mesmo quando, por algum motivo, uma das execuções não gere atividade de rede.

De um total de 10.000 execuções, 5.208 (52,08%) execuções exibiram alguma atividade de rede, totalizando 144.483 fluxos na base de dados de fluxo, se consideradas todas as execuções. Na Figura 5-1 é possível verificar a quantidade de *malwares* que produz atividade de rede caso seja executado novamente. Nesta figura é possível observar a importância de dar uma nova chance de execução para o *malware*, dado que o mesmo, por algum motivo, possa não ter produzido atividade de rede em sua execução. Se comparada a segunda execução com a primeira, 172 (16,5%) *malwares* produziram tráfego de rede durante a nova execução. Ainda, nesta mesma figura, verifica-se que 1.381 (69,05%) *malwares* produziram atividades de rede em pelo menos uma execução, enquanto 619 (30,95%) *malwares* não produziram atividade de rede em nenhuma das 5 execuções.

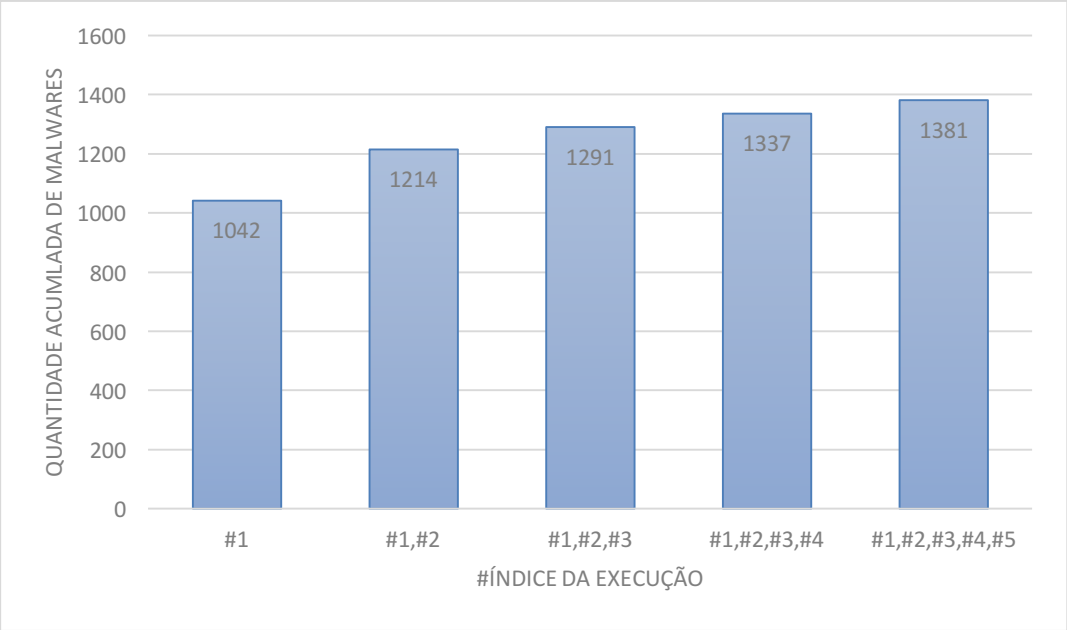


Figura 5-1. Quantidade de *malware* que produz tráfego de rede caso seja reexecutado novamente.

Das 200 famílias executadas, 18 famílias (180 *malwares*) não produziram nenhum tipo de atividade de rede, sendo que todos os *malwares* destas 18 famílias compreendem aproximadamente 29,08% de todos os *malwares* que não produziram atividade de rede, como é possível observar na Figura 5-2. As demais 182 (91%) famílias tiveram pelo menos um *malware* que produziu atividade de rede.

Segundo (ROSSOW et al., 2011), existem alguns motivos (já previamente citados) pelos quais um exemplar de *malware* pode não executar ou ainda, não produzir atividade de rede, são eles: 1) são arquivos inválidos/corrompidos; 2) operam apenas no sistema local (encriptação de disco), sem produzir atividade de rede; 3) necessitam de interação do usuário; ou 4) utilizam técnicas de anti-análise e finaliza sua execução. De modo a tentar identificar tais técnicas de anti-análise, que são bastante utilizadas entre os *malwares* (BRANCO; BARBOSA; NETO, 2012), foram utilizadas algumas assinaturas do Yara (YARA, 2016) e da Sandbox Cuckoo[17].

[17] As assinaturas da Sandbox Cuckoo são utilizadas após o período de execução do *malware*, enquanto que as assinaturas do Yara são utilizadas sem que haja a execução do *malware*.

Dos 619 *malwares* que não produziram atividade de rede em nenhuma execução, 560 (90,46%) *malwares* foram detectados por pelo menos uma regra do Yara referente a anti-análise[18]. Em relação as assinaturas presentes na Sandbox Cuckoo, dos 619 *malwares* que não produziram tráfego de rede, 295 (47,65%) *malwares* tiveram pelo menos uma assinatura que identificou algumas das atividades suspeitas realizadas pelos *malwares* durante sua execução, tais como: tentativa de atrasar a análise; tentativa de detectar a Sandbox Cuckoo pela presença de arquivos; tentativa de realizar *unhook* de funções utilizadas pela Sandbox Cuckoo; tentativa de obtenção de informações (registros, dispositivos, disco, instruções, aplicações instaladas, etc), utilizadas em técnicas de anti-análise.

Como observado por meio das assinaturas, grande parte dos *malwares* possuem comportamento suspeito em se tratando de técnicas de anti-análise, podendo ser um dos motivos pelos quais os mesmos não produziram atividade de rede. Entretanto, não é possível garantir com certeza que tais executáveis não produziram atividade de rede pelos motivos previamente citados, uma vez que um *malware* pode detectar que esta sendo analisado e mesmo assim produzir atividade de rede (ou ainda outro comportamento qualquer). Como será sugerido na parte de trabalhos futuros, é interessante utilizar outras Sandboxes (que utilizam ou não outras técnicas de análise) na tentativa de evitar técnicas específicas de anti-análise que objetivam detectar modelos específicos de Sandbox.

[18] Arquivo do Yara que possui as assinaturas de anti-análise - https://github.com/Yara-Rules/rules/blob/master/Antidebug_AntiVM/antidebug_antivm.yar

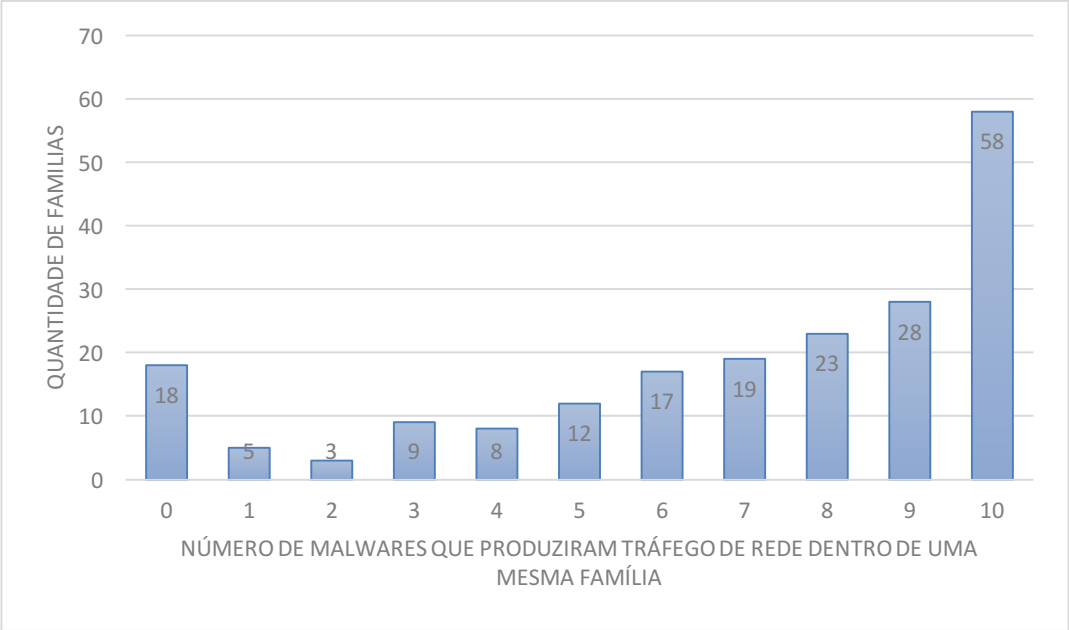


Figura 5-2. Número de *malwares* de uma família que produziram tráfego de rede.

O balanceamento do conjunto inicial de amostras, como feito neste trabalho, é essencial para garantir a heterogeneidade de comportamentos presente em diferentes famílias, e a não predominância de alguns comportamentos que poderiam comprometer o resultado do projeto.

5.4 Extração de características

Em relação ao uso dos protocolos utilizados pelos 1.381 *malwares*, que produziram atividade de rede, em pelo menos uma das execuções é possível observar a porcentagem de *malware* que utiliza cada um dos protocolos: DNS (94,35%), HTTP (65,45%), HTTPS (8,03%) e SMTP (1,15%). Como relatado em (ROSSOW et al., 2011), ambos os protocolos (DNS e HTTP) são os mais utilizados por *malwares*, que em (ROSSOW et al., 2011) tiveram as seguintes porcentagens de uso: DNS (92,3%) e HTTP (58,6%).

Em relação ao protocolo DNS houve 1.497 domínios diferentes requisitados e 1.532 endereços IPv4 distintos obtidos como resposta. Entretanto, no banco de fluxo houve a ocorrência de 13.887 endereços IPv4 distintos. A diferença entre a quantidade de endereços IPv4 obtidos por meio de resposta DNS (1.532) e a quantidade presente no banco de fluxo (13.887)

indica que diversos endereços IPv4 foram contactados sem haver requisição DNS, evidenciando que: 1) diversos endereços IPv4 estavam armazenados no *malware*; e 2) foi feito prospecção de rede em diversas subredes. Dentre as requisições DNS, 11,97% dos *malwares* realizam requisições para ambos os buscadores, Bing e Google, que, segundo (ROSSOW et al., 2011), são utilizados para teste de conectividade.

Houve 38.317 sessões HTTP, totalizando 55.468 requisições HTTP. Dentre as sessões HTTP, 4.049 possuem múltiplas requisições e as demais (34.268) sessões HTTP possuem apenas uma requisição. Dentre todas as requisições, houve predominância dos seguintes métodos HTTP: GET (91,84%), POST (6,51%) e HEAD (0,55%). Das requisições HTTP, 10.161 (18,32%) fazem uso de parâmetros e 45.307 (81.68%) não fazem, resultando no cálculo de dois *hashes* para 18.32% das requisições, enquanto que para o resto foi calculado apenas um único *hash*. Na Tabela 5-1 é possível observar um resumo das informações sobre os protocolos previamente citadas.

Tabela 5-1. Informações sobre protocolos.

Protocolo	**Porcentagem de *malwares***	**Observações**
DNS	94,35%	- 1.497 domínios diferentes requisitados e 1.532 endereços obtidos como resposta.
HTTP	65,45%	- 38.317 sessões HTTP, totalizando 55.468 requisições HTTP. - Métodos HTTP: GET (91,84%), POST (6,51%) e HEAD (0,55%). - 10.161 (18,32%) requisições fazem uso de parâmetros, enquanto que 45.307 (81.68%) não fazem.
HTTPS	8,03%	*sem informações adicionais*
SMTP	1,15%	*sem informações adicionais*

Para todo o tráfego de rede produzido, referente a todos os 1.381 *malwares*, foram detectados padrões referentes a 112 diferentes assinaturas do Suricata, totalizando 10.860 alertas emitidos. Tais alertas foram correlacionados com os fluxos referente a conexão que gerou o alerta, atribuindo um grau de severidade (extraído do alerta do Suricata) para cada um dos fluxos.

5.5 Geração de assinatura

Dos 2.000 *malwares* que foram executados, houve a geração de LCSs para 1.111 (55,55%) *malwares*. A diferença entre a quantidade de *malwares* que geraram atividade de rede (1.381) e a quantidade de *malwares* que houve a produção de LCSs (1.111) foi de 270 *malwares*. Ou seja, não houve a produção de LCSs para esses 270 *malwares*. Dentre os 270 *malwares*, houve a produção de LCSs de tamanho zero para 250, enquanto que, para 20 *malwares* não foi possível gerar nenhuma LCS uma vez que o algoritmo LCS não convergia para o resultado, por possuir, como entrada, sequências com muitos elementos.

Esses 20 *malwares* representam apenas 1,44% dos 1.381 *malwares* que produziram atividade de rede. Essa quantidade é pouco significativa se comparada com todos os *malwares* que produziram atividade de rede, não influenciando no resultado dos testes de assinatura, uma vez que os testes serão realizados com as LCSs geradas para os 1.111 *malwares*.

Como para cada *malware* pode haver a geração de mais de uma LCS, dado que o algoritmo LCS pode retornar múltiplas LCSs, foram geradas 3.659 LCSs para os 1.111 *malwares*. A decisão de qual LCS será escolhida para ser utilizada como assinatura irá depender da taxa de falso-positivo e verdadeiro-positivo para cada uma das LCSs de um determinado *malware*. Portanto, é necessário realizar alguns testes para saber qual LCS escolher dentre as criadas para o *malware* em questão. Os resultados de tais testes serão descritos na seção seguinte.

Na Figura 5-3, a barra azul mostra a quantidade de *malwares* que executou pelo número de execuções que produziram tráfego de rede. Como foram geradas LCSs apenas para os *malwares* que produziram tráfego de rede em pelo menos duas execuções (resultando em apenas uma única combinação), nota-se que não houve a produção de LCSs para os 208 *malwares* que produziram atividade de rede em apenas uma execução. A quantidade de assinaturas criadas é indicada pela barra vermelha. Para duas execuções, foram criadas LCSs apenas para 90 (70,86%) *malwares* dos 127, pois, para algumas execuções, pouca atividade de rede foi produzida. Quando pouca atividade de rede é gerada, a quantidade de elementos por sequência é pequena, podendo acarretar em LCSs de tamanho igual a zero caso não haja correspondência das mesmas conexões na outra execução.

O cenário começa a mudar quando existem 3 ou mais execuções que produzam atividade de rede. A porcentagem de assinaturas geradas para esses casos aumenta devido a quantidade de combinações resultantes. Para esses casos, mesmo que uma única execução produza pouca atividade de rede, resultando em uma LCS de tamanho zero, podem existir outras combinações que resultem em LCS com tamanho maior do que zero. Desta forma, quanto maior o número de execuções do *malware* que produzirem atividade de rede, maior é a chance de se gerar uma LCS com tamanho maior do que zero.

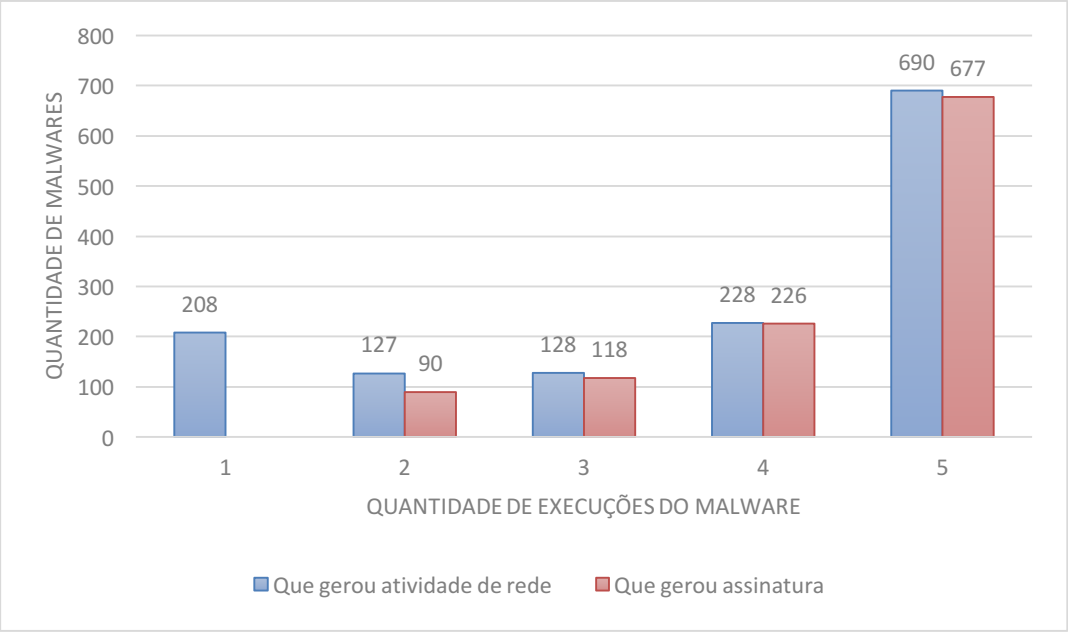


Figura 5-3. Número de execuções de um *malware* que produziram atividade de rede e assinaturas em relação a quantidade de *malwares*.

Para cada *malware* pode haver a produção de múltiplas LCSs, que, dentre as geradas, foram escolhidas as que aparecem em um maior número de execuções. Por exemplo, na combinação entre a execução 1 e 3 de um *malware* foram geradas três LCSs diferentes (LCS X, LCS Y e LCS Z), e na combição entre a execução 2 e 3 foram geradas as LCSs (LCS X e LCS Y). Desta forma, a LCS X e a LCS Y, apareceram em três execuções (1, 2 e 3), enquanto a LCS Z aparece em duas execuções (1 e 3). Desta forma, foram escolhidas apenas as LCSs que apareceram no maior número de execuções, que no caso foram as LCSs: LCS X e LCS Y. Portanto, 2.106 LCSs foram selecionadas (das 3.659 produzidas) uma vez que aparecem no maior número de execuções para cada amostra. Os testes a seguir foram realizados com as 2.106 LCSs selecionadas.

Segundo (CORRÊA, 2009), um dos fatores importante que descrevem o quão específica é uma assinatura é a quantidade de passos que a mesma possui. Na Figura 5-4 observa-se que há uma predominância de LCSs que possuem entre 1 e 10 passos.

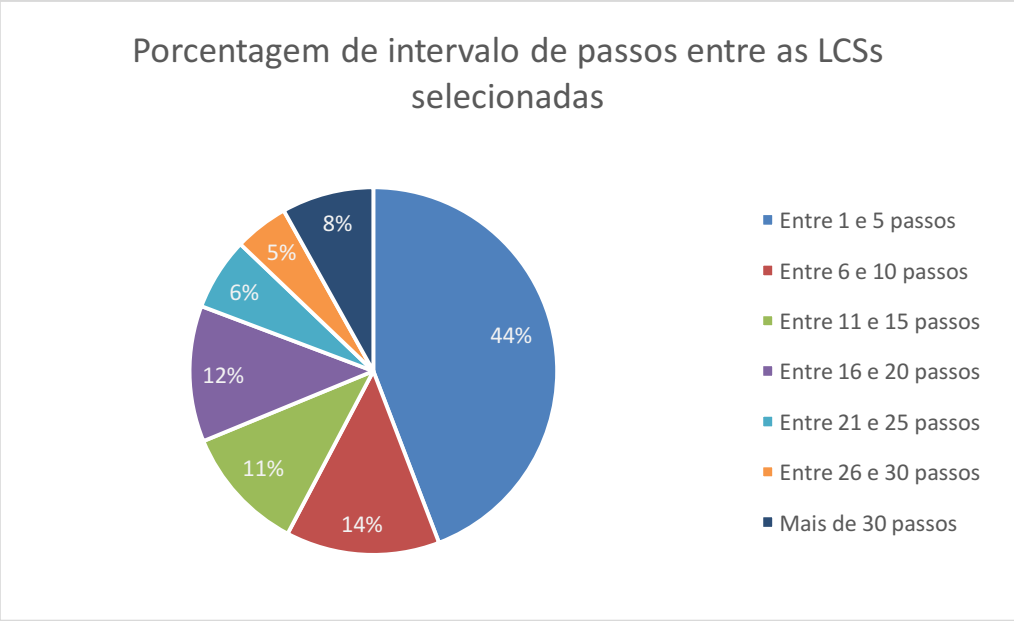


Figura 5-4. Das LCSs selecionadas, porcentagem dos intervalos de passos presentes.

Além da quantidade de passos, outro fator importante são os descritores utilizados em cada passo. Na Figura 5-5 é possível verificar, nas 2.106 LCSs selecionadas, a quantidade de LCSs que possuem cada um dos descritores. Como é possível existir um passo que utilize ambos os descritores (descritor "IP estático" e um dos demais descritores), foi feita a seguinte separação: para contabilização das LCSs que possuem endereço estático no passo da assinatura, foram selecionados as LCSs que possuem no campo "IP estático" o valor referente a algum endereço IP. Para contabilizar os demais descritores foram selecionados os passos das LCSs que não possuem valor no campo "IP estático". Na Figura 5-5, é possível observar que os descritores que aparecem em um maior número de LCSs são os descritores de comparador "maior igual" para porta 80 e comparador "igual" para porta 53 (1.216 e 1.974 LCSs, respectivamente). Os demais descritores são menos recorrentes nas LCSs, entretanto, como será visto na seção 5.6, tais descritores possuem um nível mais elevado de especificidade, influenciando na taxa de falso-positivo da assinatura.

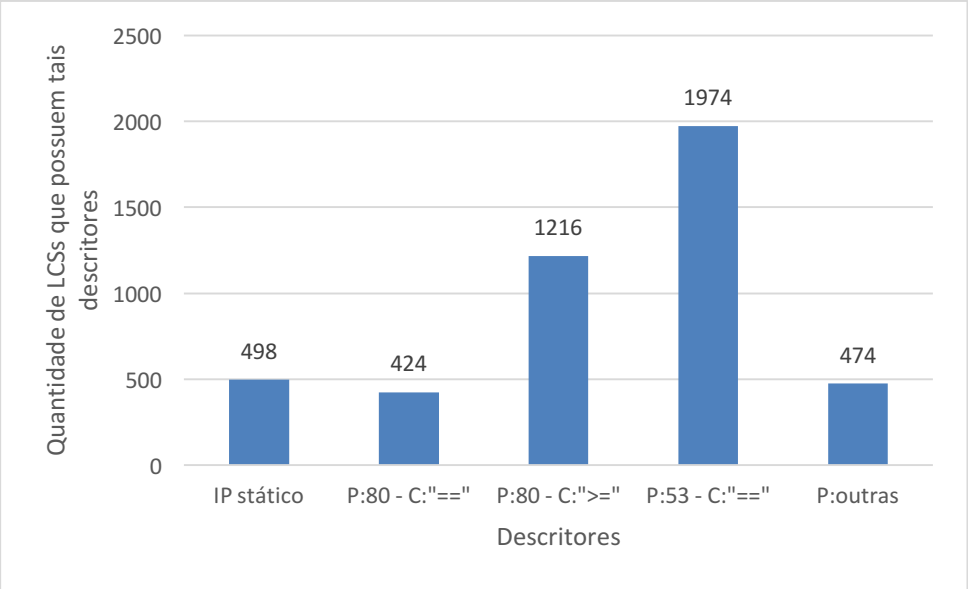


Figura 5-5. Das LCSs selecionadas, quantidade de LCSs que possuem os diferentes tipos de descritores[19].

Na seção 5.5, para as LCSs selecionados, foram feitos alguns testes com o objetivo de verificar a quantidade de LCSs que possuem falsos-positivos e verdadeiro-positivo. Os testes de falso-positivo serão realizados em relação a quantidade de passos que cada LCS possui, enquanto que, para cada um dos descritores, será testado o nível de especificidade que cada um possui.

Para fins práticos, foi obtida uma assinatura que engloba diferentes descritores: endereço IP de destino; comparadores “igual” e “maior igual”; portas que possuem análise por *payload* como é o caso das portas 53 e 80, e portas que não possuem análise de *payload*. A assinatura esta esquematizada na Tabela 5-2.

[19] Para os descritores que possuem porta de destino, é utilizado a letra “P” para simplificar a exibição, enquanto que, para os que possuem comparador, é utilizado a letra “C”.

Tabela 5-2. Exemplo de uma assinatura com 10 passos.

Passo	Porta Destino	Protocolo	Comparador	*Bytes*	IP Destino
#1	53	udp	==	32	
#2	80	tcp	==	666	
#3	53	tcp	==	35	
#4	80	tcp	>=	134	
#5	53	udp	==	31	
#6	80	tcp	==	99	
#7	2012	tcp			205.209.186.48
#8	80	tcp	>=	134	
#9	80	tcp	==	99	205.209.186.48
#10	2012	tcp			

Embora não tenha sido implementado, é possível obter os endereços IP de destino e consultá-lo no VirusTotal de forma automática, para saber se o endereço IP em questão esteve envolvido em alguma atividade maliciosa. Foi feita uma consulta no VirusTotal com o endereço IP presente na assinatura esquematizada na Tabela 5-2. A consulta retornou que o endereço IP “205.209.186.48” esteve relacionado a uma comunicação realizada por um *malware* (cuja o MD5 é b33745e540910f5e9fcf47a7b53df4b6). Dos fabricantes de antivírus que analisaram e classificaram a amostra, 89,5% a classificou como maliciosa.

5.6 Testes das assinaturas

De forma a testar as assinaturas criadas, houve uma nova execução para os 1.111 *malwares* que resultaram na criação de LCSs com tamanho maior que zero. Os *malwares* foram executados apenas uma única vez, e, dos 1.111 *malwares*, 980 (88,20%) produziram atividades de rede. A partir da Figura 5-3, foi inserida uma nova coluna (Figura 5-6) que representa a quantidade de *malwares* que, em relação a quantidade de execuções que foi

necessária para criar a assinatura, produziram atividade de rede. É possível verificar que a porcentagem de *malwares* que produziram atividade durante sua nova execução aumenta conforme aumenta a quantidade de execuções que foi necessária para criar a assinatura do *malware*. Isso acontece porque a chance do *malware* produzir atividade de rede em sua nova execução esta diretamente ligada a quantidade de execuções que geraram atividade de rede durante a etapa de geração de assinatura. Em outras palavras, se o *malware* executou atividade de rede em diversas VMs que foi executado, a chance do mesmo produzir atividade de rede durante sua nova execução é grande.

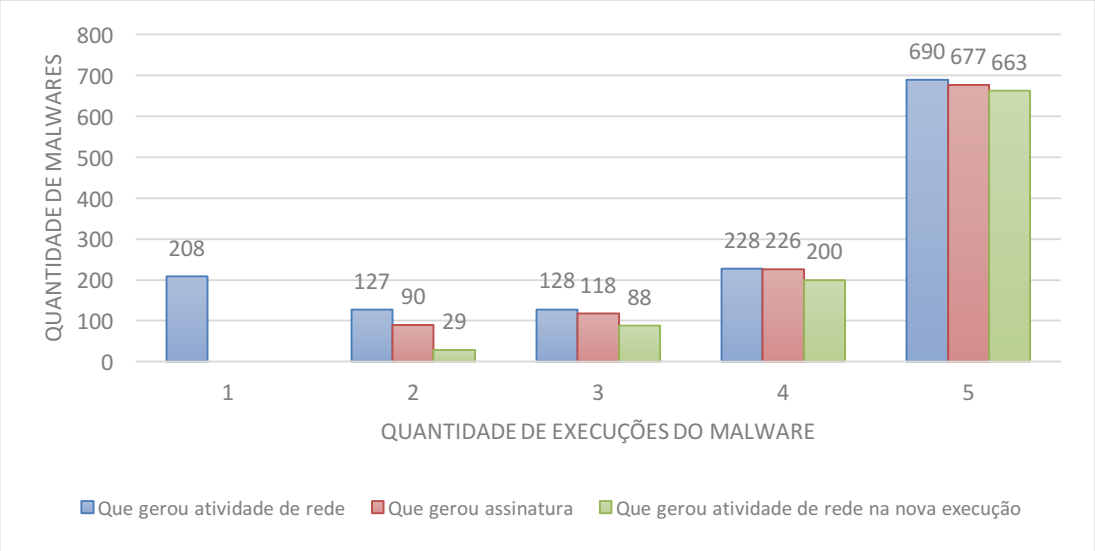


Figura 5-6. Quantidade de *malwares* que produziram: atividade de rede; assinaturas; e atividade de rede durante a nova execução.

Os testes a seguir foram realizados com o objetivo de avaliar as 2.106 LCSs que foram selecionadas. As LCSs que foram selecionadas englobam o maior número de execuções para cada amostra, como discutido na seção 5.4. Para avaliar a qualidade das LCSs selecionadas, foram utilizadas as métricas descritas na seção 4.7.

Todas as LCSs foram testadas em um banco de fluxo de dados de rede, obtidos de um ambiente controlado. Este banco possui 1.145.124 fluxos normais, ou seja, ausentes de quaisquer tipos de ataque. Embora as 2.106 LCSs geradas possuam entre 1 e 3.470 passos, para facilitar a visualização da Figura 5-7, foram selecionadas apenas as LCSs que possuem entre 1 e 40

passos, que correspondem a 95.44% de todas as LCSs geradas. Na Figura 5-7 é possível observar a relação entre a quantidade de passos que uma LCS possui versos a taxa de falso-positivo.

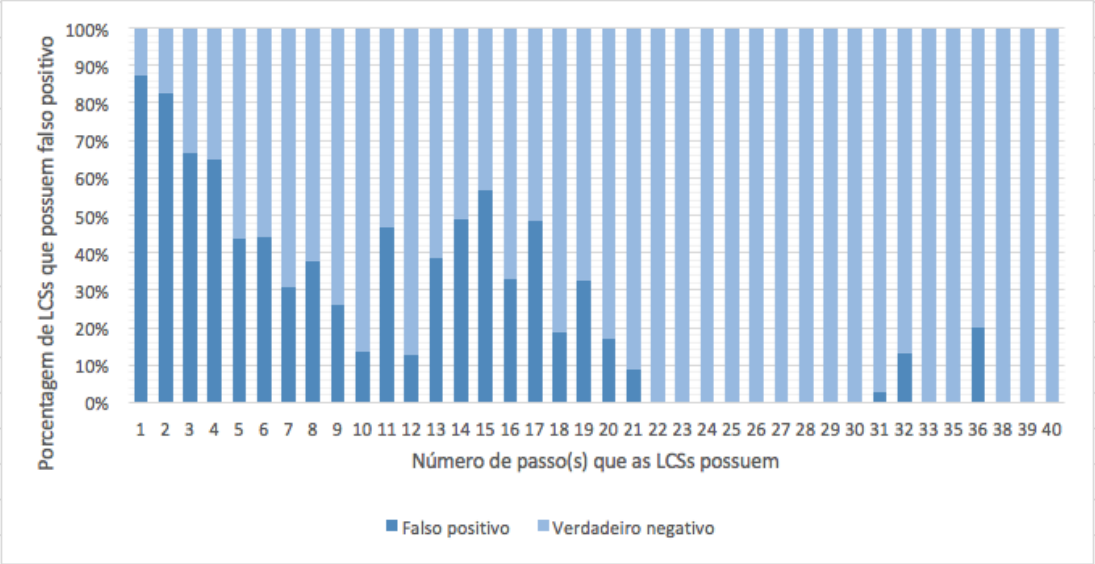


Figura 5-7. Relação entre o número de passo(s) que as LCSs possuem e a porcentagem de falso-positivo.

Na Figura 5-7 observa-se que quanto menos passos uma LCS possui, mais será a porcentagem de falso-positivo, ou seja, menos especifica ela é, como ressaltado em (CORRÊA, 2009). Este comportamento fica claro no gráfico para as quantidades de passos menores, por exemplo entre 1 e 7 passos (que representam aproximadamente 50% das assinaturas criadas). Entretanto, para algumas LCSs que possuem uma quantidade maior de passos (como é o caso das LCSs de tamanho 15 por exemplo), verificar-se que as mesmas possuem uma taxa de falso-positivo maior que as LCSs de tamanho 10 por exemplo. Este comportamento acontece porque existe outro fator que influencia na especificidade das LCSs.

Além da quantidade de passos presentes em uma assinatura, outro fator que influencia na especificidade de uma assinatura são os descritores utilizados em cada passo. Se utilizarmos como exemplo as assinaturas que possuem apenas um único passo (que representam 15% de todas as assinaturas), 87,22% das assinaturas possuem falso-positivo enquanto 12,78% não possui. Dos 12,78% que não possui falso-positivo, todas as

assinaturas possuem o descritor de endereço IP de destino, ou seja, o *malware* se comunicou (ou tentou se comunicar) com esses endereços de destino sem haver qualquer requisição DNS. Das LCSs produzidas, 23,64% delas utilizam pelo menos um endereço IP fixo na sua composição.

Em relação aos descritores, não foi possível mensurar a relação entre cada descritor e a taxa de falso-positivo, uma vez que a taxa de falso-positivo esta relacionado a LCS como um todo e não especificamente a um único passo (que contém o(s) descritor(es) em questão). Como o objetivo deste teste é medir o impacto (na taxa de falso-positivo) que cada descritor, **individualmente**, tem ao ser utilizado em um passo da assinatura, foi utilizada a seguinte abordagem: para cada um dos descritores foi obtida a quantidade média de fluxo normal retornada. A geração do histograma (Figura 5-8) foi realizada da mesma forma que o histograma da Figura 5-5, havendo a separação apenas dos passos que possuem endereço IP de destino para calcular a quantidade de resultados retornado para este descritor. Para os demais descritores foram selecionadas as LCSs que não possuem endereço IP de destino.

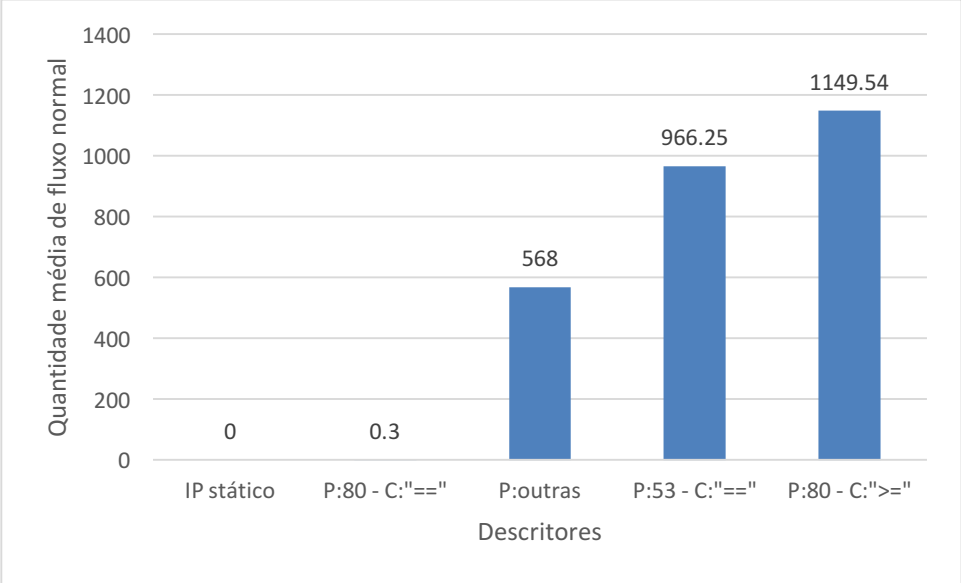


Figura 5-8. Relação entre a quantidade média de fluxo normal retornado e o descritor utilizado.

No teste realizado em relação aos descritores, observa-se que quanto menor a média de fluxo retornado, mas especifico é o passo que contém tal descritor. Desta forma, caso uma assinatura possua endereço IP estático em seu(s) passo(s) e/ou porta de destino 80 e comparador “igual”, existe uma grande chance do passo não retornar resultados, sendo um fator positivo para que a assinatura não possua falso-positivo.

Cada uma das 2.106 LCSs selecionadas foram testadas com as conexões produzidas pelas execuções dos respectivos *malwares*. Ao final deste teste, para cada LCS testada foi possível saber se a mesma detectou ou não a nova atividade de rede exercida pelo *malware*. Logo, uma LCS terá 0 ou 1 como verdadeiro-positivo.

Dos 1.111 *malwares* que resultaram na produção de LCSs, 980 (88,20%) *malwares* produziram atividade de rede quando executados novamente. Para esses 980 *malwares*, haviam sido geradas 1.847 LCSs. Entretanto, dessas 1.847 LCSs que foram checadas no tráfego de rede da nova execução do *malware*, apenas 1.124 (60,85%) LCSs identificaram corretamente o tráfego de rede. Essas 1.124 LCSs correspondem a LCSs de 817 *malwares*, ou seja, para os 980 *malwares* que geraram atividade de rede, 817 (83,36%) *malwares* tiveram pelo menos uma LCS que identificou seu tráfego de rede. Ainda, das 1.124 LCSs, apenas 367 (32,65%) tiveram ausência de falso-positivo (quando testadas na base de dados com fluxo normal). Essas 367 LCSs correspondem a LCSs de 256 *malwares*. O esquema geral acerca da produção de LCSs em cada uma das etapas pode ser visualizado na Figura 5-10.

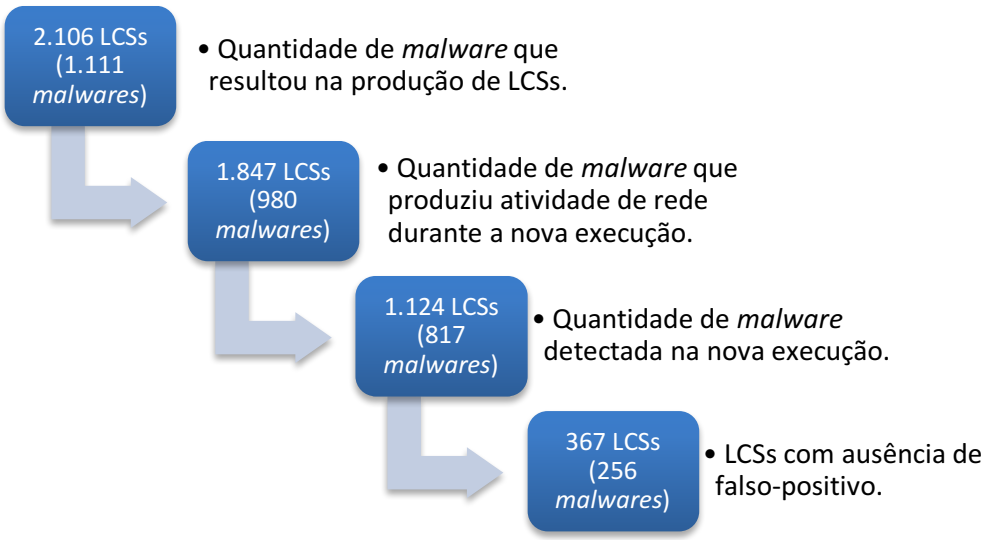


Figura 5-9. LCSs resultantes após cada etapa de teste.

Na Figura 5-10 observa-se a relação entre a quantidade de diferentes execuções que colaboraram para geração de uma mesma LCS e a quantidade das seguintes LCSs: LCSs selecionadas; LCSs que possuem verdadeiro-positivo; e LCSs que não possuem falso-positivo. Quanto maior a quantidade de diferentes execuções utilizadas para criar a mesma LCS, maior será a porcentagem de LCSs que possuem verdadeiro-positivo. Isso ocorre uma vez que a LCS criada para o *malware* foi genérica o suficiente para envolver um maior número de execuções do *malware*, resultando em uma maior probabilidade de detectar sua nova execução.

O gráfico da Figura 5-10 é bastante semelhante ao gráfico da Figura 5-6, porém a diferença é que um *malware* pode produzir tráfego de rede em todas as 5 execuções (Figura 5-6), por exemplo, porém uma LCS gerada pode estar presente em apenas 3 execuções (Figura 5-10).

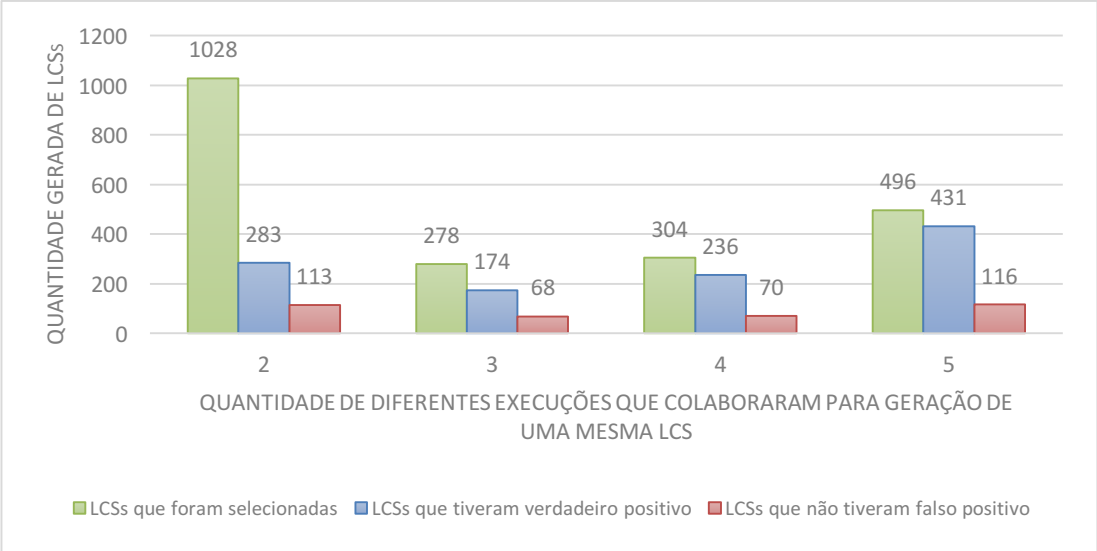


Figura 5-10. Quantidade de LCSs geradas de acordo com a quantidade de diferentes execuções que foram necessárias para gerarem a LCS.

Portanto, dos 980 *malwares* que produziram atividade de rede, foi possível detectar 817 (83,36%) *malwares*. Entretanto, foi possível gerar assinaturas livres de falso-positivo para detectar 256 (31,33%) *malwares* dos 817 *malwares*.

Na Figura 5-11 observa-se que, a maioria das LCSs possuem entre 1 e 10 passos. Das 124 LCSs (34%) que possuem entre 1 e 5 passos, 55 (44,35%) possuem endereço IP de destino na LCS. Para as LCSs de apenas um único passo, 100% delas possuem endereço IP de destino na LCS. Já as demais LCSs (entre 2 e 5 passos) possuem grande quantidade de passos que contém o comparador “igual” para ambas as portas (53 e 80), ou ainda, alguns passos possuem portas de destino que não são comuns, como por exemplo: 6667, 2010 e 5358.

Em relação as LCSs que possuem entre 6 e 10 passos (91 LCSs), a quantidade de LCSs que possuem endereço IP de destino na LCS cai para 17 LCSs (18,68%) se comparado com as LCSs entre 1 e 5 passos. Ainda, para as LCSs de 6 e 10 passos, o comparador “maior igual’ está presente em 68 (74,72%) das 91 LCSs, enquanto que para as LCSs entre 1 e 5 passos, apenas 23 (18,54%) de 124 possuem o comparador “maior igual”.

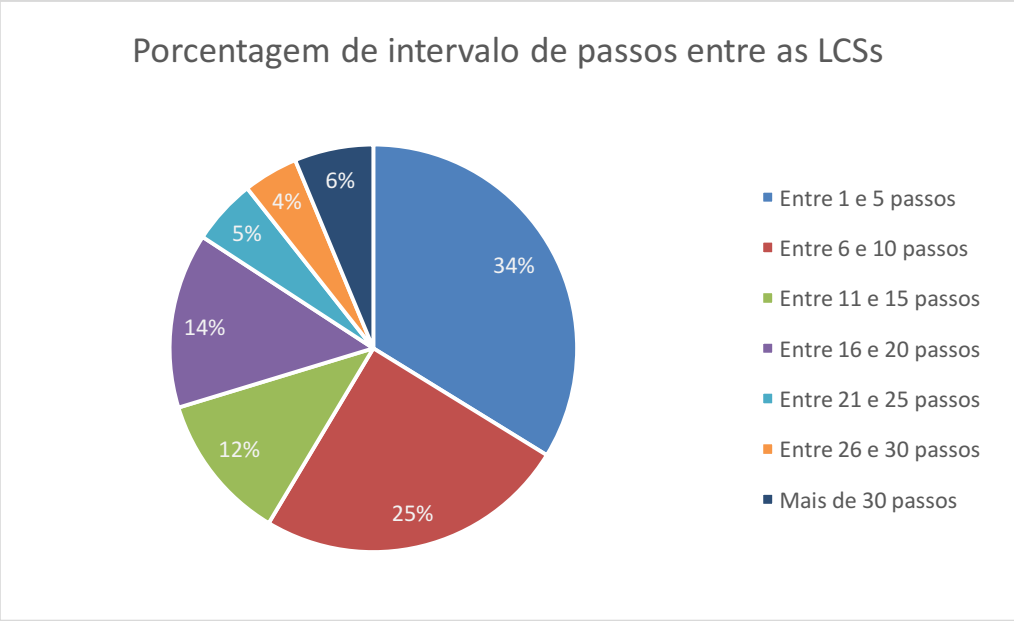


Figura 5-11. Porcentagem dos intervalos de passos presentes nas LCSs que podem ser utilizadas como assinatura.

Os testes realizados ajudaram a verificar a quantidade de LCSs que foram efetivas (1.124 LCSs), referente a 817 *malwares*, para identificação do tráfego de rede gerado durante a nova execução dos 1.111 *malware* analisados. Ainda, das 1.124 LCS, 367 LCS (referente a 256 *malwares*) possuem ausência de falso-positivo, sendo altamente indicadas para serem utilizadas como assinatura. Por fim, como observado por meio dos testes, foi possível obter um resultado promissor uma vez que as assinaturas foram criadas de forma automatizada, sem qualquer intervenção humana.

Por fim, em relação ao tempo gasto para gerar a assinatura, o método elaborado nesta pesquisa depende de dois principais fatores: 1) tempo gasto para analisar a amostra e 2) tempo gasto para geração das LCS.

1) **Tempo gasto para analisar a amostra** – o tempo médio gasto para cada execução de cada amostra foi de 393 segundos (6 minutos e 33 segundos), resultando em 1965 segundos (32 minutos e 45 segundos) caso seja considerada as 5 execuções de uma mesma amostra. A estimativa resultante das 5 execuções (de 1965 segundos) é levando em consideração o pior caso, onde as análises não ocorrem em paralelo. Caso seja considerado o melhor caso, o tempo decorrido será

de 393 segundos para cada amostra uma vez que existem 6 máquinas virtuais realizando a análise dos *malwares* em paralelo.

2) **Tempo gasto para geração das LCS** – o tempo médio gasto para a geração de todas as LCSs de cada amostra foi de 0.04 segundos (para as 1.111 assinaturas que foram criadas). O menor tempo médio gasto para geração das LCSs para uma única amostra foi de 0.00001 segundos, enquanto que o maior tempo médio gasto foi de 10.67 segundos.

5.7 Considerações finais

Neste capítulo foram apresentados os resultados obtidos por meio dos experimentos realizados. Para os 980 *malwares* que produziram atividade de rede durante sua nova execução, foi possível gerar pelo menos uma assinatura para detectar 817 (83,36%) *malwares*. Entretanto, das assinaturas geradas para os 817 *malwares*, assinaturas geradas para 256 (31,33%) *malwares* possuem ausência de falso-positivo. Das LCSs geradas, observou-se que a maioria possui entre 1 e 10 passos, onde estas devem possuir a presença de alguns tipos de descritores para que não haja a presença de falso-positivo.

Com base nos testes realizados foi possível verificar que além da quantidade de passos que uma assinatura possui, outro fator que influência na taxa de falso-positivo são os descritores utilizados. Quanto mais diferenciados forem os descritores utilizados (em relação aos descritores extraídos a partir de um tráfego normal) em cada passo das assinaturas, menor será a probabilidade de uma assinatura possuir falso-positivo.

Por fim, é importante ressaltar que embora as métricas utilizadas para detecção de falso-positivo e verdadeiro-positivo são amplamente utilizadas no contexto de detecção de intrusão, nesse projeto tais métricas foram utilizadas para selecionar as assinaturas que possuem verdadeiro-positivo e remover as que possuem falso-positivo. Sendo assim, tais métricas não foram utilizadas para avaliar a taxa de detecção (como comumente são utilizadas), uma vez que a metodologia proposta neste trabalho diz respeito a geração

automatizada de assinaturas baseado em fluxo de dados de rede e não um à Sistemas de Detecção de Intrusão.

CAPÍTULO 6 - Conclusões

De forma a lidar com a grande quantidade de *malwares* presentes atualmente, o emprego de metodologias automatizadas de geração de assinatura se faz necessário, uma vez que os ambientes de defesa precisam ser alimentados.

Como visto na literatura, as metodologias de geração automatizada de assinaturas compartilham uma mesma característica: o uso de algoritmos que buscam por similaridades. Uma vez identificadas, são criadas assinaturas de forma a detectar tais ações maliciosas.

Nesse trabalho foi elaborada uma metodologia capaz de gerar assinaturas de *malwares* de forma automatizada. A criação de tais assinaturas foi possível uma vez que foi identificada características semelhantes no tráfego de rede gerado entre múltiplas execuções de um mesmo *malware*. A identificação de semelhanças no tráfego de rede ocorreu, de modo geral, em duas etapas: 1) geração de *hash* para cada tipo de conexão, onde a maneira que o *hash* é gerado depende do tipo da conexão; 2) utilização do algoritmo LCS para encontrar uma subsequência em comum entre duas sequências geradas a partir das conexões realizadas pelo *malware* durante cada uma de suas execuções.

Uma vez que a geração de assinatura automatizada para *malwares* ocorreu com sucesso, e sem intervenção humana, foi possível confirmar a hipótese inicial proposta. Dos 980 *malwares* que produziram tráfego de rede durante a nova execução, foi possível detectar 817 (83,36%) deles.

Em comparação com os trabalhos presentes na literatura, a metodologia proposta gera assinaturas para *malwares* possibilitando a utilização em IDS que fazem detecção baseada em fluxo de dados de rede. Ainda, tal metodologia utiliza análise de *payload* para gerar alguns descritores para serem utilizados em nível de fluxo. A combinação dos fatores citados resulta na principal contribuição da metodologia desenvolvida neste trabalho.

Uma desvantagem de utilizar as assinaturas geradas pela metodologia proposta, é que, caso o *malware* não gere conexões que possuam características (em nível de fluxo) que se diferenciem do tráfego de rede normal, tais assinaturas terão uma grande probabilidade de possuir falso-positivo, uma vez que as assinaturas baseadas em fluxo são menos específicas se comparadas com as baseadas em *payload*. Dos 817 *malwares* que foram detectados durante a nova execução, foi possível produzir assinaturas livres de falso-positivo para 256 (31,33%). Para os demais casos, é necessário a utilização de metodologias que realizam a geração de assinaturas baseadas em *payload*, uma vez que, caso haja a geração de assinaturas para serem utilizadas a nível de fluxo, as mesmas irão possuir falso-positivo.

É importante ressaltar que o fato de uma assinatura possuir falso-positivo esta diretamente ligada às limitações presentes na tecnologia de fluxo de dados de rede, e não a metodologia desenvolvida, uma vez que, dependendo do caso, as informações extraídas em nível de fluxo não são especificas o suficiente, resultando em falso-positivo. Sendo assim, é necessário testar as assinaturas criadas de forma a remover as que possuem falso-positivo.

Desta forma o uso das assinaturas geradas pela metodologia proposta, se tornam vantajosas apenas para alguns tipos de *malwares*, ou seja, para *malwares* que produzam tráfego de rede com algumas características específicas, ou ainda, *malware*s que realizam diversas conexões, resultando em assinaturas com diversos passos.

6.1 Problemas encontrados durante o desenvolvimento da pesquisa

Ao longo desta pesquisa foram encontrados alguns problemas, sendo que o principal foi a limpeza do *dump* do tráfego de rede. Para resolver tal problema foi necessário realizar algumas alterações no arquivo `modules/processing/network.py` para que a limpeza pudesse ser feita de forma automatizada, como explicado no Apêndice B deste documento.

Outro problema encontrado foi em relação ao tamanho das sequências criadas a partir das conexões de algumas execuções dos *malwares*. Para 20 (1,44%) *malwares* não foi possível gerar LCS uma vez que houve a produção de sequência muito grande em alguma das execuções.

6.2 Trabalhos futuros e contribuições

Algumas ideias de trabalhos futuros:

- Utilizar outros sistemas operacionais e *Sandbox* como tentativa de generalizar as assinaturas criadas, sendo independente do sistema de análise e da plataforma utilizada. Além disso, a utilização de diferentes configurações (*sandbox*, sistema operacional e *softwares* de terceiros) pode resultar na execução de *malwares* que utilizam técnicas de anti-análise para alguns tipos específicos de ambiente.
- Paralelizar o processo de geração de LCS de forma a diminuir o tempo necessário para encontrá-las. Para o caso de sequências muito longas, caso o paralelismo não resolva tal problema, verificar se é viável reduzir alguns elementos da sequência com base na importância dos descritores de cada elemento.
- Implementar análise de *payload* para outros protocolos, como por exemplo: SMTP, FTP, IRC, entre outros.

- Produzir assinaturas genéricas para famílias de *malwares* com base em LCSs criadas para dois ou mais *malwares* que pertençam a mesma família.
- Integrar a metodologia produzida por esse trabalho com o trabalho desenvolvido em (CORRÊA, 2009). Para tal, será necessário colocar um componente intermediário entre a saída do gerador de assinatura com a entrada do módulo de cadastramento de assinatura desenvolvido em (CORRÊA, 2009).
- Atualização das assinaturas. Após geradas e testadas as LCSs para um *malware*, é possível atualizá-las, caso necessário. O processo de atualização da assinatura pode ocorrer da seguinte forma: aplicar o algoritmo LCS entre a sequência resultante de uma nova execução do *malware* e a LCS criada durante a etapa de geração de assinatura. Sendo assim, a nova LCS resultante pode ser considerada uma atualização da antiga LCS.

Uma das contribuições foi feita em relação a ferramenta Argus, em que foi reportado para os desenvolvedores um problema simples no agrupamento de fluxos DNS. Mais especificamente a ferramenta `racluster` tinha um problema ao interpretar uma opção específica do arquivo de configuração. Quando utilizada a opção *filter* para o protocolo DNS, o mesmo estava agrupando todas as requisições DNS que usavam a mesma porta efêmera, sendo que o correto era apenas correlacionar a requisição com a resposta. Na Tabela 6-1 é possível verificar as informações de fluxo de dados de rede antes da correção feita na ferramenta, enquanto na Tabela 6-2 é possível observar os fluxos resultantes após aplicar a correção na ferramenta Argus.

Tabela 6-1. Informações de fluxo antes da correção.

StartTime	Flgs	Proto	SrcAddr	Sport	Dir	DstAddr	Dport	TotPkts	TotBytes	State
21:53:13.609647	e	udp	192.168.100.106.1040		<->	192.168.100.1.domain		8	1041	CON

Tabela 6-2. Informações de fluxo após correção.

StartTime	Flgs	Proto	SrcAddr	Sport	Dir	DstAddr	Dport	TotPkts	TotBytes	State
21:53:13.609647	e	udp	192.168.100.106.	1040	<->	192.168.100.1.	domain	2	240	CON
21:53:14.135221	e	udp	192.168.100.106.	1040	<->	192.168.100.1.	domain	2	224	CON
21:53:15.079598	e	udp	192.168.100.106.	1040	<->	192.168.100.1.	domain	2	368	CON
21:53:16.315496	e	udp	192.168.100.106.	1040	<->	192.168.100.1.	domain	2	209	CON

Referências

ANDERSON, Ross. Security engineering. John Wiley & Sons, 2008.

ANUBIS. ISECLAB. Disponível em: < https://anubis.iseclab.org/>. Acesso em: 07 set. 2016.

AVTEST. The independente IT-Security Institute. Disponível em: < https://www.av-test.org/en/statistics/malware/>. Acesso em: 4 dez. 2016.

BATISTA, M. L. Análise de eventos de segurança em redes de computadores utilizando detecção de novidade. 2012. 72 f. Dissertação (Mestrado em Ciências da Computação)–Instituto de Biociências, Letras e Ciências Exatas, Universidade Estadual Paulista, São José do Rio Preto, 2012.

BAYER, U., COMPARETTI, P. M., HLAUSCHEK, C., KRUEGEL, C., & KIRDA, E. (2009, February). Scalable, Behavior-Based Malware Clustering. In NDSS (Vol. 9, pp. 8-11).

BERGROTH, LASSE, HARRI HAKONEN, AND TIMO RAITA. "A survey of longest common subsequence algorithms." String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on. IEEE, 2000.

BERNERS-LEE, T., FIELDING, R., & FRYSTYK, H. RFC 1945: Hypertext Transfer Protocol -- HTTP/1.0. Disponível em: <https://tools.ietf.org/html/rfc1945/>. Acesso em: 19 dez. 2016.

BLUNDEN, Bill. The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System. Jones & Bartlett Publishers, 2013.

BRANCO, R. R., BARBOSA, G. N., & NETO, P. D. (2012). Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies. Black Hat.

BRO. The Bro Network Security Monitor. Disponível em: < https://www.bro.org/>. Acesso em: 17 dez. 2016.

BUGTRAQ. Exploit for proftpd 1.2.0pre6. Disponível em: < http://seclists.org/bugtraq/1999/Sep/ 328>. Acesso em: 02 jan. 2016.

CABALLERO, J., GRIER, C., KREIBICH, C., & PAXSON, V. (2011, August). Measuring Pay-per-Install: The Commoditization of Malware Distribution. In Usenix security symposium (p. 15).

CERT.br. Cento de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil. Estatísticas sobre notificações de incidentes. Disponível em: <http://www.cert.br/stats/>. Acesso em: 07 jun. 2016.

CGSECURITY. w00w00 on Heap Overflows. Disponível em: < http://www.cgsecurity.org/exploit/ heaptut.txt>. Acesso em: 10 fev. 2016.

CHANG, J., VENKATASUBRAMANIAN, K. K., WEST, A. G., & LEE, I.. Analyzing and defending against web-based malware. ACM Computing Surveys (CSUR), 45(4), 49, (2013).

CHHABRA, P., JOHN, A., & SARAN, H. (2005, May). PISA: automatic extraction of traffic signatures. In International Conference on Research in Networking (pp. 730-742). Springer Berlin Heidelberg.

CISCO. Introduction to Cisco IOS Netflow - A technical overview. 2009. Disponível em:<http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/prod_white_paper0900aecd80406232.pdf >. Acesso em: 20 jul. 2016.

CORMEN, THOMAS H. Introduction to algorithms. MIT press, 2009.

CUCKOO. Cuckoo Sandbox: Automated Malware Analysis. 2014. Disponível em: < http://www.cuckoosandbox.org/> Acesso em: 17 fev. 2016.

CLAISE, B. (a). RFC 7011: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. Disponível em: <https://tools.ietf.org/html/rfc7011/>. Acesso em: 19 jan. 2016.

CLAISE, B. (b). RFC 5101: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. Disponível em: <https://tools.ietf.org/html/rfc5101/>. Acesso em: 19 jan. 2016.

CLAISE, B. (c). RFC 3954: CISCO Systems Netflow Services Export version 9. Disponível em: <http://www.ietf.org/rfc/rfc3954.txt>. Acesso em: 14 mar. 2016.

CORRÊA, J. L. Um modelo de detecção de eventos em redes baseado no rastreamento de fluxos. 2009. 111 f. Dissertação (Mestrado em Ciências da Computação)–Instituto de Biociências, Letras e Ciências Exatas, Universidade Estadual Paulista, São José do Rio Preto, 2009.

D1.2: Attack Detection and Signature Generation. NoAH. Disponível em: < https://www.fp6-noah.org/publications/deliverables/D1.2.pdf>. Acesso em: 30 nov. 2016.

DINABURG, A., ROYAL, P., SHARIF, M., & LEE, W. (2008, October). Ether: malware analysis via hardware virtualization extensions. In Proceedings of the 15th ACM conference on Computer and communications security (pp. 51-62). ACM.

EAGLE, C. The IDA Pro book: the unofficial guide to the world's most popular disassembler. No Starch Press. (2011).

EGELE, M., SCHOLTE, T., KIRDA, E., & KRUEGEL, C. A survey on automated dynamic malware-analysis techniques and tools. ACM Computing Surveys (CSUR), 44(2), 6. (2012).

ELSHOUSH, H. T.; OSMAN, I. M. Reducing false positives through fuzzy alert correlation in collaborative intelligent intrusion detection systems: a review. In: IEEE INTERNATIONAL CONFERENCE ON FUZZY SYSTEMS (FUZZ), 2010, Barcelona. Proceedings... Barcelona, 18-23 July 2010. p.1-8.

EMERGINGTHREATS. Emerging Threats list. Disponível em: < https://lists.emergingthreats.net/mailman/listinfo>. Acesso em: 03 abr. 2016.

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., & BERNERS-LEE, T. RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1. Disponível em: <https://tools.ietf.org/html/rfc2616/>. Acesso em: 19 dez. 2016.

FILHO, D. S. F., GRÉGIO, A. R., AFONSO, V. M., SANTOS, M. J., & DE GEUS, P. L. Análise Comportamental de Código Malicioso Através da Monitoração de Chamadas de Sistema e Tráfego de Rede. In: X Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg). Fortaleza - CE 2010, pp. 311-324.

FILHO, D. S. F., AFONSO, V. M., MARTINS, V. F., GRÉGIO, A. R. A., DE GEUS, P. L., JINO, M., & DOS SANTOS, R. D. C. Técnicas para Análise Dinâmica de Malware. Anais do XI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. (2011).

GOLLMANN, D. Computer Security, 1a Edição, John Wiley & Sons, 1999.

GRAZIANO, M.; LEITA, C.; BALZAROTTI, D. Towards network containment in malware analysis systems. In: Proceedings of the 28th Annual Computer Security Applications Conference. ACM, 2012. p. 339-348.

HONEYD. Disponível em: <http://www.honeyd.org/>. Acesso em: 11 nov. 2016.

HONEYINSPECTOR. SOURCEFORGE. Disponível em: <http://honeyinspector.sourceforge.net/>. Acesso em: 09 fev. 2016.

INETSIM. Internet Services Simulation Suite. Disponível em: < http://www.inetsim.org/>. Acesso em: 4 set. 2016.

JANG, J., WOO, M., & BRUMLEY, D. Towards automatic software lineage inference. In Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13) (pp. 81-96). (2013).

KAUR, S.; SINGH, M. Automatic attack signature generation systems: A review. Security & Privacy, IEEE, v. 11, n. 6, p. 54-61, 2013.

KIRAT, D., VIGNA, G., & KRUEGEL, C. (2014). Barecloud: bare-metal analysis-based evasive malware detection. In 23rd USENIX Security Symposium (USENIX Security 14) (pp. 287-301).

KHARRAZ, A., ARSHAD, S., MULLINER, C., ROBERTSON, W., & KIRDA, E. (2016). UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. In 25th USENIX Security Symposium (USENIX Security 16) (pp. 757-772).

KREIBICH, C.; CROWCROFT, J. "Honeycomb: creating intrusion detection signatures using honeypots." ACM SIGCOMM Computer Communication Review 34.1 (2004): 51-56.

KUROSE, J. F., ROSS, K. W. (2013). Redes de Computadores e a Internet: uma abordagem top-down. Pearson. 6.ed.

LIBEROUTER. IPFIXcol. Disponível em: <https://www.liberouter.org/technologies/ipfixcol/>. Acesso em: 04 jun. 2016.

LIN, Y. et al. Active versus Passive Malware Collection. IEEE Computer, v. 47, n. 4, p. 59-65, 2014.

MALWR. Disponível em: < https://malwr.com/>. Acesso em: 07 fev. 2016.

MCAFEE (a). Relatório do McAfee Labs sobre ameaças – Março de 2016. Disponível em: <http://www.mcafee.com/br/resources/reports/rp-quarterly-threats-mar-2016.pdf>. Acesso em: 11 dez. 2016.

MCAFEE (b). Relatório do McAfee Labs sobre ameaças – Setembro de 2016. Disponível em: <http://www.mcafee.com/br/resources/reports/rp-quarterly-threats-sep-2016.pdf>. Acesso em: 11 dez. 2016.

NELSON, A. Sandnet++ - A framework for analysing and visualising network traffic from malware. Technical report. 2016.

OFFENSIVECOMPUTING. Open Malware - Georgia Tech Information Security Center. Disponível em: <http://www.offensivecomputing.net/ >. Acesso em: 29 nov. 2016.

OLIVEIRA, I. L. Sistema de coleta, análise e detecção de código malicioso baseado no sistema imunológico humano. 2012. 87 f. Dissertação (Mestrado em Ciências da Computação)–Instituto de Biociências, Letras e Ciências Exatas, Universidade Estadual Paulista, São José do Rio Preto, 2012.

OPENWALL. JPEG COM Marker Processing Vulnerability. Disponível em: <http://www.openwall .com /articles/JPEG-COM-Marker-Vulnerability>. Acesso em: 13 fev. 2016.

OSSEC. Open Source HIDS SECurity. Disponível em: < http://ossec.github.io/>. Acesso em: 20 dez. 2016.

OWASP. OWASP Top Ten Project. Disponível em: < https://www.owasp.org/index.php/Category: OWASP_Top_Ten_Project>. Acesso em: 03 abr. 2016.

Park, Byung-Chul, et al. "Towards automated application signature generation for traffic identification." *NOMS 2008-2008 IEEE Network Operations and Management Symposium*. IEEE, 2008.

PERDISCI, R., ARIU, D., & GIACINTO, G. (2013). Scalable fine-grained behavioral clustering of http-based malware. Computer Networks, 57(2), 487-500.

PERDISCI, R., LEE, W., & FEAMSTER, N. (2010, April). Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In NSDI (pp. 391-404).

PORTOKALIDIS, G.; SLOWINSKA, A.; BOS, H. "Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation." ACM SIGOPS Operating Systems Review. Vol. 40. No. 4. ACM, 2006.

PHRACK. Smashing The Stack For Fun And Profit. Disponível em: <http://phrack.org/issues/49/14.html>. Acesso em: 25 fev. 2016.

PROOFPOINT. Emerging Threats Intelligence. Disponível em: < https://www.proofpoint.com/us/products/et-intelligence>. Acesso em: 03 abr. 2016.

QEMU. Disponível em: <http://wiki.qemu.org/Main_Page>. Acesso em: 2 nov. 2016.

QOSIENT. ARGUS. Disponível em: <http://www.qosient.com/argus/index.shtml>. Acesso em: 03 out. 2016.

QUITTEK, J. et al. RFC 3917: Requirements for IP Flow Information Export. Disponível em: <http://www.ietf.org/rfc/rfc3917.txt>. Acesso em: 23 fev. 2016.

RAFIQUE, M. Z.; CABALLERO, J. "Firma: Malware clustering and network signature generation with mixed network behaviors." Research in Attacks, Intrusions, and Defenses. Springer Berlin Heidelberg, 2013. 144-163.

RIECK, K., SCHWENK, G., LIMMER, T., HOLZ, T., & LASKOV, P. (2010, March). Botzilla: Detecting the phoning home of malicious software. In Proceedings of the 2010 ACM Symposium on Applied Computing (pp. 1978-1984). ACM.

ROSSOW, C., DIETRICH, C. J., BOS, H., CAVALLARO, L., VAN STEEN, M., FREILING, F. C., & POHLMANN, N. Sandnet: Network traffic analysis of malicious software. In Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (pp. 78-88). ACM. (2011, April).

ROSSOW, C., DIETRICH, C. J., GRIER, C., KREIBICH, C., PAXSON, V., POHLMANN, N., ... & VAN STEEN, M. (2012, May). Prudent practices for designing malware experiments: Status quo and outlook. In 2012 IEEE Symposium on Security and Privacy (pp. 65-79). IEEE.

SEBASTIÁN, M., RIVERA, R., KOTZIAS, P., & CABALLERO, J. (2016, September). Avclass: A tool for massive malware labeling. In International Symposium on Research in Attacks, Intrusions, and Defenses (pp. 230-253). Springer International Publishing.

SNORT (a). Lightweight intrusion detection for networks. Disponível em: <https://www.snort.org/>. Acesso em: 21 dez. 2016.

SNORT (b). SNORT Products. Disponível em: <https://www.snort.org/products>. Acesso em: 21 dez. 2016.

SIKORSKI, M.; HONIG, A. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. No Starch Press, 2012.

SILK. CERT. Disponível em: <https://tools.netsa.cert.org/silk/>. Acesso em: 09 mar. 2016.

SURICATA. Open Source IDS / IPS / NSM engine. Disponível em: < https://suricata-ids.org/>. Acesso em: 20 dez. 2016.

TCPDUMP. Disponível em: <http://www.tcpdump.org/>. Acesso em: 12 fev. 2016.

TESO. Exploiting Format String Vulnerabilities. scut / team teso. Disponível em: < http://crypto.stanford.edu/cs155/papers/formatstring-1.2.pdf>. Acesso em: 07 jan. 2016.

TRAMMELL, B., & BOSCHI, E. RFC 5103: Bidirectional Flow Export Using IP Flow Information Export (IPFIX). Disponível em: <https://tools.ietf.org/html/rfc5103/>. Acesso em: 29 nov. 2016.

VAN DER VEEN, V., CAVALLARO, L., & BOS, H. Memory errors: the past, the present, and the future. In International Workshop on Recent Advances in Intrusion Detection (pp. 86-106). Springer Berlin Heidelberg. Springer Berlin Heidelberg, 2012.

VIRUSHARE. Because Sharing is Caring. Disponível em: <https://virusshare.com/>. Acesso em: 2 nov. 2016.

VIRUSTOTAL. Disponível em: < https://www.virustotal.com/ >. Acesso em: 29 dez. 2016.

VXHEAVEN. Disponível em: <http://vxheaven.org/>. Acesso em: 21 dez. 2016.

WANG, K., & STOLFO, S. J. (2004, September). Anomalous payload-based network intrusion detection. In International Workshop on Recent Advances in Intrusion Detection (pp. 203-222). Springer Berlin Heidelberg.

WARAICH, R. (2005). Automated Attack Signature Generation: A Survey. Technical report, ETH Zurich, Computer Engineering and Networks Laboratory.

WILLEMS, C.; HOLZ, T.; FREILING, F., "Toward Automated Dynamic Malware Analysis Using CWSandbox," IEEE Security and Privacy, vol. 5, no. 2, pp. 32-39, Mar./Apr. 2007.

WIKIBOOKS. Algorithm Implementation/Strings/Longest common subsequence. Disponível em: < https://en.wikibooks.org/w/index.php?title=Algorithm_Implementation/Strings/Longest_common_subsequence&oldid=3108703#Reading_out_all_LCSs>. Acesso em: 15 dez. 2016.

WU, S. X.; BANZHAF, W. The use of computational intelligence in intrusion detection systems: A review. Applied Soft Computing, v. 10, n. 1, p. 1– 35, Jan. 2010.

YAF. CERT. Disponível em: <https://tools.netsa.cert.org/yaf/>. Acesso em: 09 nov. 2016.

YARA. The pattern matching swiss knife for malware researchers. Disponível em: <http://virustotal.github.io/yara/>. Acesso em: 07 fev. 2016.

ZHANG, W.; YANG, Q.; GENG, Y. A Survey of Anomaly Detection Methods in Networks. Computer Network and Multimedia Technology. International Symposium on , vol., no., pp.1-3, 18-20 Jan. 2009.

APÊNDICE A – Configuração de cada máquina do ambiente e os *softwares* adicionais instalados

Máquina Virtual	Sistema Operacional	Configurações	*Softwares* adicionais
VM01	Windows XP *Professional* SP2	32-bit, 512MB de RAM	Nenhum
VM02	Windows 7 *Professional* SP1	32-bit, 768MB de RAM	Nenhum
VM03	Windows 7 *Professional* SP1	64-bit, 1GB de RAM	Nenhum
VM04	Windows 7 *Professional* SP1	32-bit, 768MB de RAM	Acrobat Reader 10, Java RE 1.6.0.22 (jre6u22), Adobe Flash Player 11, Microsoft .NET Framework 4.6
VM05	Windows 7 *Professional* SP1	32-bit, 1GB de RAM	Acrobat Reader 10, Java RE 1.6.0.22 (jre6u22), Adobe Flash Player 11, Microsoft .NET Framework 4.6
VM06	Windows XP *Professional* SP2	32-bit, 1GB de RAM	Acrobat Reader 10, Java RE 1.6.0.22 (jre6u22), Adobe Flash Player 11, Microsoft .NET Framework 3.5

A configuração da máquina para suportar as VMs é: Intel Zeon 64-bit (4 cores), 8GB de RAM e HD de 500GB. Sistema operacional Linux (Ubuntu – 15.10).

APÊNDICE B – Metodologia utilizada para realizar a limpeza do *dump* de rede

Foi possível observar no tráfego de rede que as instâncias virtuais que estavam sendo utilizadas para análise dinâmica dos *malwares* estavam se comunicando com a Internet mesmo antes da execução do *malware*. Uma vez que foram utilizadas máquinas Windows XP Service Pack 2 (x86) e máquinas Windows 7 Service Pack 1 (x86 e x64), foi possível observar que essas comunicações envolviam atualizações do Sistema Operacional, teste de conectividade (por meio do serviço *Network Connectivity Status Indicator*) além de tentativas de atualizações de *softwares* de terceiros.

Uma primeira possível solução considerada e testada foi: desativar algumas chaves do *Registry* do Windows (responsáveis por testes de conectividade) e desabilitar atualização de *softwares* de terceiros. Porém, notou-se que esta não era a solução mais adequada, uma vez que havia necessidade de desabilitar todos os mecanismos de atualização do Windows e *softwares* de terceiros, e mesmo assim, foi possível notar que alguns *softwares* mantinham algum estado de comunicação com a Internet. Além de não ser um método garantido, esta solução seria um problema visando a escalabilidade do ambiente de análise, uma vez que seria necessário repetir o mesmo processo para cada nova máquina criada.

A segunda possível solução foi deixar todas as máquinas executando por um período de tempo onde alguns programas eram executados na tentativa de se obter todas os *domains* envolvidos nas comunicações de rede. Após obtido todos os *domains*, os mesmos foram inseridos (utilizando expressões regulares) no arquivo `network.py` presente no seguinte diretório [`modules/processing/network.py`] da Sandbox Cuckoo. Além disso, foi feita uma alteração na Sandbox Cuckoo para que o mesmo criasse um arquivo chamado `hosts.txt` no diretório de cada *malware* analisado. Através deste arquivo foi possível verificar todos os *domains* e *subdomains* que estavam no filtro e se envolveram de alguma forma com as comunicações de rede. Após recuperar esses *domains*, foi possível remover toda comunicação geradas para os IPs em questão utilizando o *software* tshark. Desta forma, foi possível

produzir o *dump* final, chamado de `cleanup_dump.pcap`, onde fora removido comunicações não originadas pelo *malware*. Embora tenha-se encontrada uma solução viável e que possibilite a escalabilidade do sistema de análise, não há uma garantia de 100% a cerca do método uma vez que é possível que o próprio *malware* realize uma requisição para um dos *domains* contidos no filtro.

APÊNDICE C – Trechos de algoritmos

```
Hash-Fluxo(conexao)
1    ip_destino = conexao.fluxo.ip_destino

2    if Originado-de-Dns(ip_destino)
3        destino = Obter-dominio(ip_destino)
4    else
5        destino = ip_destino

6    md5 = MD5-Calc(destino+";"+conexao.fluxo.porta_destino+";"+
     conexao.fluxo.protocolo)
7    return md5
```

```
Hash-DNS(conexao)
1    md5 = MD5-Calc(conexao.dns.dominio)
2    return md5
```

```
Hash-HTTP(conexao)
1    http_cabecalho_host = Buscar-Linha-Host
     (conexao.http.linhas_cabecalho)

3    if Possui-Parametros(conexao.http.linha_requisicao)
4        linha_requisicao_sem_valores = Remover-Valores-Linha-
         Requisicao(conexao.http.linha_requisicao)

5        md5_sem_valores = MD5(linha_requisicao_sem_valores+";"+
         http_cabecalho_host)
6    else
7        md5_sem_valores = ""

8    md5_linha_inteira = MD5(conexao.http.linha_requisicao+";"+
     http_cabecalho_host)

9    hashes.linha_inteira = md5_linha_inteira
10   hashes.sem_valores   = md5_sem_valores

11   return hashes
```

```
Resolver-Hashes-HTTP(conexoes1, conexoes2)
1    conexoes_hashes1 = Obter-Conexoes-Multi-Hash(conexoes1)
2    conexoes_hashes2 = Obter-Conexoes-Multi-Hash(conexoes2)

3    for conexao_hashes1 in conexoes_hashes1
4        buscar = conexao_hashes1.hashes.sem_valores
5        hashes_iguais_conexoes2 = Buscar-Hashes-Sem-Valores
         (buscar, conexoes_hashes2)

6        quantidade_linha_inteira = 0

7        for hashes_iguais_conexao2 in hashes_iguais_conexoes2
8                if conexao_hashes1.hashes.md5_linha_inteira ==
                 hashes_iguais_conexao2.hashes.md5_linha_inteira
9                    conexao_hashes1.hash =
                     conexao_hashes1.hashes.md5_linha_inteira
10                   hashes_iguais_conexao2.hash =
                     hashes_iguais_conexao2.hashes.md5_linha_inteira
11                   quantidade_linha_inteira++
12               else
13                   if quantidade_linha_inteira == 0
14                       conexao_hashes1.hash =
                         conexao_hashes1.hashes.md5_sem_valores

15                   hashes_iguais_conexao2.hash =
                     hashes_iguais_conexao2.hashes.md5_sem_valores
```