Dissertação de Mestrado em

Engenharia Informática - Computação Móvel

Benchmark para análise de drift no tráfego de rede

João Pedro Leal Santos

Leiria, Março de 2018

Dissertação de Mestrado em

Engenharia Informática - Computação Móvel

Benchmark para análise de drift no tráfego de rede

João Pedro Leal Santos

Dissertação de Mestrado realizada sob a orientação do Doutor Mário João Gonçalves Antunes e da Doutora Catarina Helena Branco Simões Silva, Professores da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria.

Leiria, Março de 2018

Esta página foi intencionalmente deixada em branco.

Nota prévia

Do trabalho efetuado resultou a publicação:

- Generating Benchmark Datasets for Intrusion Detection Systems, in RECPAD 2017

Do trabalho efetuado resultou o seguinte protótipo aplicacional:

- Real Network Dataset Generator (RNDGen), in `http://www.sourceforge.net/p/rndgen`

Esta página foi intencionalmente deixada em branco.

Resumo

As redes de computadores são o meio mais comum, e natural, de utilização para trabalho e comunicação pelas massas levando a um enorme volume de transações de dados. O volume de dados transmitidos tem aumentado bastante nos últimos anos, contendo dados sensíveis e informações confidenciais que têm sido sucessivamente utilizados para fins ilícitos.

Com o objetivo de mitigar a invasão de privacidade dos dados alojados nos sistemas de informação das organizações, têm sido implementados nas redes várias soluções de segurança como os Intrusion Detection System (IDS)s. Estas soluções quando são implementadas necessitam de passar por um conjunto de testes por forma a garantir a sua robustez e capacidade de ação nas várias situações de intrusão. É nesta problemática que se foca este trabalho onde através da geração de *datasets* compostos por pacotes de rede de tráfego normal e anormal (ataques) em formato *pcap* se pretende testar a robustez das soluções implementadas. Sendo que se pretende colmatar a atualmente reduzida existência de *datasets* de teste que representem tráfego aproximado ao real. Esta construção tem por base dados reais, pré-capturados na rede em teste, para serem posteriormente estudados estatisticamente com o objetivo de encontrar um padrão de comportamento/utilização para adaptar os dados resultantes à realidade em causa. Esta abordagem foca-se principalmente na realidade dos dados capturados, permitindo assim produzir *datasets* com um grau de semelhança à realidade igualmente elevado.

Além da abordagem técnica em formato *pcap*, este trabalho pretende também testar mecanismos mais específicos como IDSs produzidos no âmbito de trabalhos de investigação. Para isso é possível criar *datasets* em formato de texto simples, onde os campos que os compõe, bem como a sua ordem, são totalmente personalizáveis consoante as necessidades.

Deste trabalho surgiu um protótipo de uma aplicação, disponível em `http://www.sourceforge.net/p/rndgen`, em *perl*, utilizável para efetuar a criação dos *datasets* pretendidos.

Palavras-chave: Tráfego malicioso; Tráfego de rede; Estatística; Anomalias; Datasets; IDS;

Esta página foi intencionalmente deixada em branco.

Abstract

Computer networks are the most common, and natural, means of use for work and communication by the masses leading to a huge volume of data transactions. The volume of data transmitted has increased significantly in recent years, containing sensitive data and confidential information that has been successively used for illicit purposes.

In order to mitigate the invasion of privacy of data hosted in organizations' information systems, several security solutions such as IDS have been implemented. These solutions, when implemented, need to go through a set of tests in order to guarantee their robustness and ability to act in the various intrusion situations. It is in this problematic that this work focuses where through, the generation of datasets composed of network packets of normal and abnormal traffic (attacks) in pcap format intended to test the robustness of the implemented solutions. Since it is intended to fill the currently gap of reduced existence of test datasets representing approximate traffic to real data. This construction is based on real data, pre-captured in the network under test, to be later studied statistically in order to find a behavior/use pattern to adapt the resulting data to the reality in question. This approach focuses mainly on the reality of the captured data, thus allowing to produce datasets with a similar degree of similarity to reality.

In addition to the technical approach in pcap format, this work also intends to test more specific mechanisms like IDS produced in the scope of research works. For this you can create datasets in plain text format, where the fields that compose them, as well as their order, are fully customizable as required.

With this work was produced a prototype of an application, available at
`http://www.sourceforge.net/p/rndgen`, in perl, usable to create the desired datasets.

Keywords: Abnormal Network Traffic; Normal Network Traffic; Statistics; Anomalies; Datasets; IDS;

Esta página foi intencionalmente deixada em branco.

Lista de Figuras

Esta página foi intencionalmente deixada em branco.

Lista de Tabelas

Esta página foi intencionalmente deixada em branco.

Acrónimos

AD Anomaly-based Detection

API Application Programming Interface

ARP Address Resolution Protocol

CDX Cyber Defense eXercise

CUDA Compute Unified Device Architecture

CVE Common Vulnerabilities and Exposures

DCE/RPC Distributed Computing Environment/Remote Procedure Calls

DDoS Distributed DoS

DNS Domain Name System

DNP3 Distributed Network Protocol

DoS Denial of Service

ESTG Escola Superior de Tecnologia e Gestão

FCFS First Come, First Serve

FTP File Transfer Protocol

GPU Graphics Processing Unit

GRE Generic Routing Encapsulation

GTP GPRS Tunneling Protocol

IDS Intrusion Detection System

HIDS Host-based IDS

HTTP HyperText Transfer Protocol

ICMP Internet Control Message Protocol

IGMP Internet Group Management Protocol

IGRP Interior Gateway Routing Protocol

IMAP Internet Message Access Protocol

IP Internet Protocol

IPLeiria Instituto Politécnico de Leiria

IPS Intrusion Prevention System

MEI-CM Mestrado Engenharia Informática - Computação Móvel

MyCERT Malaysia Computer Emergency Response Team

NBA Network Behaviour Analysis

NIDS Network-based IDS

NIPS Network-based IPS

OSPF Open Shortest Path First

PCAP Packet Capture

PDU Protocol Data Unit

POP Post Office Protocol

PPP Point-to-Point Protocol

PPPoE Point-to-Point Protocol over Ethernet

RIP Routing Information Protocol

RPC Remote Procedure Call

SD Signature-based Detection

SIP Session Initiation Protocol

SMB Server Message Block

SMTP Simple Mail Transfer Protocol

SSH Secure SHell

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

VLAN Virtual LAN

VM Virtual Machine

WIDS Wireless-based IDS

Esta página foi intencionalmente deixada em branco.

Conteúdo

Capítulo 1

Introdução

Esta dissertação insere-se no plano curricular do Mestrado Engenharia Informática - Computação Móvel (MEI-CM) da Escola Superior de Tecnologia e Gestão (ESTG) do Instituto Politécnico de Leiria (IPLeiria). A dissertação incide sobre a temática da análise de tráfego de rede, designadamente sobre o desenvolvimento de uma solução para geração artificial de tráfego para posterior análise de variações de contexto (*drifts*), com vista ao teste dos mecanismos e aplicações de segurança configurados na rede.

A forte utilização das redes de computadores como meio de comunicação e de trabalho gera grande volume de dados resultantes das operações efetuadas pelos equipamentos interligados numa rede. Além do conteúdo, relevante para o utilizador, a ser transportado entre emissor e receptor, são também transportadas informações irrelevantes tais como configurações de rede e dados essenciais para estabelecer comunicação. Assim, estes dados necessitam de ser protegidos de ações mal intencionadas, utilizando-se para esse efeito várias aplicações de segurança como *firewalls*, Intrusion Detection System (IDS), entre outros. Após a configuração destes mecanismos, existe a necessidade de verificar se são eficazes na sua função de deteção das variações contexto para as quais estão configurados. Para tal é necessário que se proceda à simulação de tentativas de intrusão ou de ataques de qualquer tipo, por exemplo utilizando aplicações de injeção controlada de ataques na rede para efeitos de testes e validação. No âmbito do trabalho associado a esta dissertação, pretende-se desenvolver uma aplicação que implementa as seguintes funções:

- Identificar o comportamento de uma rede, através da recolha de tráfego normal durante um período de tempo;
- Produzir *datasets* artificiais com base no tráfego normal recolhido juntamente com o tráfego correspondente a ataques previamente executados em ambiente de testes.

A produção dos *datasets* artificias deve responder às exigências apresentadas pelos vários intervenientes quer da área de administração da rede, quer da área de pesquisa e estudo de mecanismo de segurança de redes. Do ponto de vista de administração, a aplicação deve permitir uma total integração com o meio de aplicação, utilizando e respeitando formatos e características do tráfego, permitindo o correto funcionamento de toda a configuração. Do ponto de vista de pesquisa, existe a necessidade de adaptar os formatos e características de tráfego relacionados com o cenário de aplicação.

Desta forma, com esta dissertação pretende-se atingir o objetivo final de avaliação de eficácia de deteção de intrusões pelos IDS implementados numa rede, através de testes realizados com os *datasets* gerados artificialmente.

A metodologia utilizada inicia-se com uma pesquisa bibliográfica de enquadramento ao tema e de abordagens já estudadas por terceiros, resultando num protótipo da aplicação a criar. A aplicação permite analisar e avaliar o tráfego de uma rede e os seus mecanismos de segurança possibilitando também a alteração de parâmetros com vista a avaliar pontos específicos de segurança. Por fim, foram efetuados testes de utilização e funcionamento da aplicação, cuja análise permitiu tecer as conclusões mais importantes à utilização desta aplicação em contexto empresarial.

1.1 Motivação

A utilização da Internet gera diariamente um grande volume de transações e dados, suscitando vários problemas de segurança, na medida em que a circulação de informação sensível dos seus utilizadores, tais como dados pessoais, transações bancárias e até mesmo conversas privadas a torna num alvo privilegiado a ataques [1].

De forma a garantir a deteção de ações maliciosas contra os dados que circulam na rede, existe a necessidade de implementação de sistemas de deteção de intrusões (Intrusion Detection System (IDS)) sendo também importante monitorar o funcionamento dos dispositivos e garantir que o seu software se encontra devidamente atualizado, mantendo os seus níveis de eficiência e segurança perante as ameaças atuais [2]. Estas ameaças tentam contornar os mecanismos de segurança existentes na rede através da exploração das suas vulnerabilidades, que por sua vez surgem através de falhas, erros de software ou até mesmo por configuração deficiente dos serviços e equipamentos [1].

Este é um tema bastante relevante para o panorama atual onde qualquer rede ativa necessita de estar preparada para um cenário onde será alvo constante de tentativas e intrusão e ataque,

quer seja uma rede de uma pequena-média empresa, ou uma rede governamental com dados sensíveis da gestão de um país assumindo assim um grau de grande importância na sua gestão (empresas/governo), merecendo atenção e investimento.

O tema associado a este trabalho assume um grau de grande importância no panorama tecnológico atual sendo motivado pelo objetivo de criar uma aplicação capaz de testar a segurança dos mecanismos implementados numa rede. De acordo com a bibliografia utilizada neste documento, não existe nenhuma aplicação semelhante que permita a criação automática de métodos de teste de soluções de segurança, nomeadamente recorrendo à criação artificial de *datasets* com pacotes reais. Sendo de maior relevo a sua característica de estudo estatístico realizado sobre os dados, permitindo adaptar os resultados à realidade organizacional em teste.

1.2 Objetivos

Conforme referido anteriormente a segurança é um ponto de extrema importância numa rede sendo necessário aperfeiçoar métodos e ferramentas para esse efeito.

Neste trabalho, pretende-se apresentar um estudo detalhado para geração de *datasets* com base em dados estatísticos, resultando numa aplicação de benchmarking de tráfego de rede que facilite o teste dos mecanismos de segurança de uma rede. A aplicação permite caracterizar e classificar o tráfego previamente capturado, possibilitando a geração de *datasets* com opções parametrizáveis adaptando-se a cada cenário prático. Os *datasets* poderão ser gerados tendo por base o tipo de tráfego e tipo de ataque, permitindo uma maior adaptação aos testes pretendidos. Em suma com esta aplicação pretende-se:

- Caracterizar e classificar tráfego;
- Estudar distribuição estatística do tráfego;
- Gerar tráfego com base numa distribuição estatística;
- Gerar *datasets* constituídos por tráfego normal e de ataques;
- Gerar *datasets* em formato *pcap*;
- Gerar *datasets* em formato texto simples customizável.

Os *datasets* gerados no âmbito deste trabalho têm por base uma rede de testes definida e serão utilizados para testar Intrusion Detection System (IDS).

1.3 Estrutura do documento

Este documento encontra-se estruturado da seguinte forma:

- **Capítulo 1**, introdução ao projeto e suas motivações, desafios e abordagem geral;
- **Capítulo 2**, descrição dos conceitos base associados ao tema do projeto, permitindo facilitar a compreensão dos aspetos futuramente implementados;
- **Capítulo 3**, abordagem prática ao tema e aplicação de conceitos descritos nos capítulos anteriores;
- **Capítulo 4**, descrição e resultados de testes realizados no âmbito do tema do projeto;
- **Capítulo 5**, conclusões retiradas da execução do projeto, bem como indicações para trabalho a realizar futuramente.

Capítulo 2

Conceitos Fundamentais

Neste capítulo serão apresentados e descritos os conceitos base para uma melhor compreensão das tecnologias e protocolos utilizados ao longo deste trabalho.

2.1 A segurança na rede

A forte utilização de uma rede de computadores, torna-a numa importante via de comunicação e informação. Na rede circulam grandes quantidades de dados, sensíveis e não sensíveis, tornando-a assim num alvo apetecível a ataques, com o principal objetivo de aceder e/ou copiar os dados que nela circulam, colocando assim em risco a informação dos seus utilizadores. Estes ataques podem, ainda, levar a que a rede colapse ou que a sua velocidade de resposta seja reduzida de forma notável, através da sua sobrecarga [1]. Estes são problemas de rede que afetam o seu funcionamento normal, denominando-se geralmente como anomalias, que surgem não só por via de ataque, como também devido a configurações deficientes da rede. Assim, podemos considerar dois grandes grupos de anomalias [3]:

- **Anomalias de desempenho** - as que afetam o desempenho da rede, resultantes apenas de problemas de configuração ou capacidade de resposta insuficiente para o volume de tráfego produzido;

- **Anomalias de segurança** - onde existe uma intenção clara, por parte de terceiros, de prejudicar o funcionamento e desempenho da rede.

As anomalias de desempenho focam-se no desempenho dos dispositivos de rede, ocorrendo anomalia quando estes falham devido a, por exemplo, configuração deficiente ou à

sobrecarga de acessos em simultâneo. As anomalias de segurança focam-se no tráfego propriamente dito, detetando irregularidades causadas, na sua maioria, intencionalmente.

A Figura 2.1 representa esquematicamente os tipos de anomalias.

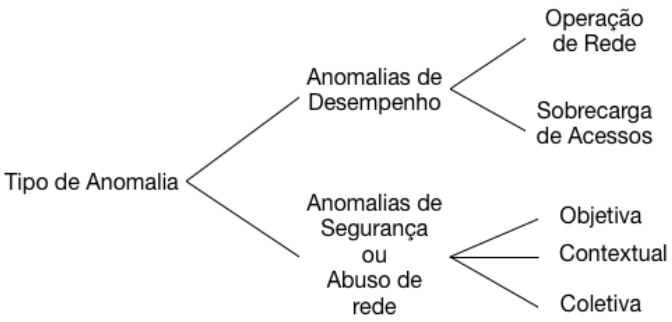


Figura 2.1: Tipos de anomalia.

As anomalias de desempenho podem ser divididas em dois tipos:

- **Operação de rede** - quando uma ação não pode ser realizada por falha dos dispositivos da rede. Estas falhas podem surgir devido a configuração deficiente do dispositivo ou alterações nessa mesma configuração. Um exemplo de anomalias deste tipo são a inclusão de novos equipamentos na rede que, como novos elementos na rede, necessitam de configurações de acordo com as já existentes, de forma a funcionarem em consonância com os restantes. Podem ocorrer problemas de compatibilidade com estes, não estando, por vezes, devidamente configurados [1, 4];

- **Sobrecarga de acessos** - quando existe um número elevado de acessos à mesma rede, em simultâneo, num curto espaço de tempo levando à diminuição da sua performance ou, até mesmo, à sua falha. Este tipo de anomalia pode surgir, de forma involuntária, quando ocorre um acontecimento de grande relevância a nível nacional/internacional despertando o interesse de uma grande parte dos utilizadores da rede, levando a que todos acedam ao mesmo sítio em simultâneo [1, 4]. O Denial of Service (DoS)/Distributed DoS (DDoS) é também uma das causas de sobrecarga da rede, ao contrário da anterior, esta causa ocorre de forma voluntária.

As anomalias de segurança, ou de abuso de rede, podem ser divididas em três tipos:

- **Objetiva** - tipo mais simples de anomalia, foca-se numa parte dos pacotes do fluxo de rede, sendo esta parte considerada irregular (anómala) em relação aos pacotes restantes. Adequando a um cenário do quotidiano, uma compra efetuado com cartão de

crédito com o valor demasiado elevado em comparação com os valores habituais, é considerada anomalia objetiva [4, 5];

- **Contextual** - o contexto dos dados é estipulado pela estrutura presente no tráfego de rede, assim, os dados são considerados anómalos se o seu contexto não corresponder ao definido. O contexto é definido pelo atributo contextual e atributo de comportamento:

  - Atributo contextual - define o contexto dos dados, da mesma forma que a longitude e latitude definem um local no espaço [4, 5];
  - Atributo de comportamento - define parâmetros não contextuais dos dados. Comparando com um cenário real, pode ser a quantidade de chuva num determinado local [4, 5];

  Observando o tempo que passa entre compras efetuadas com um cartão de crédito é possível detetar uma atividade fraudulenta. Este tempo, em situação anómala, está fora do padrão normal de utilização, podendo dizer-se que esses tempos se encontram fora do contexto de utilização [4];

- **Coletiva** - um conjunto de dados que, estando relacionados de alguma forma, são considerados anómalos. Quando vistos de forma isolada são dados normais, que não representam qualquer risco. Consideramos uma anomalia coletiva, por exemplo, um conjunto de comandos que quando observados isoladamente não representam qualquer ameaça, mas quando executados de uma determinada ordem permitem desencadear ações maliciosas [4]. Considerando a sequência de comandos `...http-web, buffer-overflow, http-web, http-web, smtp-mail, ftp, http-web, ssh,-smtp-mail, http-web,` **`ssh, buffer-overflow, ftp`**`, http-web, ftp,-smtp-mail,http-web...` observamos que, de forma isolada, cada comando não representa qualquer ameaça. Se notarmos, a sequência a negrito (**ssh**, **buffer-overflow**, **ftp**) já é possível identificar uma ação fora do normal onde existe um acesso remoto a uma máquina seguida de uma cópia de dados para a máquina de onde é originário o acesso [5].

Uma forma de deteção e combate de anomalias passa pela análise regular do tráfego que circula na rede, embora as anomalias de abuso de rede possam, por vezes, ser difíceis de detetar apenas pela análise do fluxo de rede pois os ataques atuam de forma imprevisível

e sem um padrão facilmente detetável, sendo necessária a utilização de outros mecanismos para a sua deteção ser possível.

Os ataques estão divididos em vários tipos, todos eles com métodos de ação diferentes, mas com um objetivo em comum: invadir a rede e os seus dados. Podemos falar então em *malware*, *DoS/DDoS* sendo estes os tipos mais frequentes de acordo com a bibliografia consultada [2, 6, 7, 8, 9, 10]. Para além destes ataques, mais diretos, podemos considerar também outras vias de atividade maliciosa na rede como o caso do *SPAM* e da fraude informática, como por exemplo o *phishing* [6, 7, 8, 9, 10].

Ao falar de *malware* percebemos, através do seu nome, que se trata da ocorrência de software malicioso ou que parte do código que o constitui é malicioso. Esse código tem como objetivo explorar e aproveitar-se das vulnerabilidades existentes na rede/máquina alvo com a intenção de executar algumas ações sem o conhecimento do utilizador, tais como, o roubo ou destruição de dados e ficheiros. O *malware* pode ser dividido em várias categorias dependendo do tipo de código utilizado, sendo os mais frequentes, conforme ilustra a Figura 2.2 [2, 11, 12]:

- ***Trojans*** - software malicioso, por vezes incorporado em software legítimo. Fica ativo quando o software, aparentemente, legítimo é instalado executando diversos ataques ao ambiente alvo roubando e eliminando dados, bem como infetando o alvo com outros virus. Este possibilita também a criação de *backdoors*[1] para acesso futuro;

- ***Adware*** - é um tipo de *malware*, utilizado por organizações para publicitar os seus produtos. Por vezes vem alojado secretamente em *software* grátis embora seja mais frequente em jogos, existindo a possibilidade de adquirir uma versão paga com o objetivo de eliminar toda a publicidade. Um *adware* manifesta-se na máquina através de *Popups*, mensagens de alerta e *banners*;

- **Vírus** - vêm associados a ficheiros executáveis fazendo parte destes, sendo apenas lançados no sistema quando o ficheiro é executado. É transmissível para outros sistemas via rede, transferência de ficheiros e emails. Um vírus pode, ainda, eliminar software do sistema com o objetivo de ocupar o seu lugar, passando despercebido;

- ***Worms*** - semelhantes aos vírus, embora não estejam camuflados em software. Um *worm* é um software/aplicação específico com o objetivo único de atacar a máquina explorando as vulnerabilidades do sistema usando-as a seu favor na tentativa de levar a que o utilizador o execute.

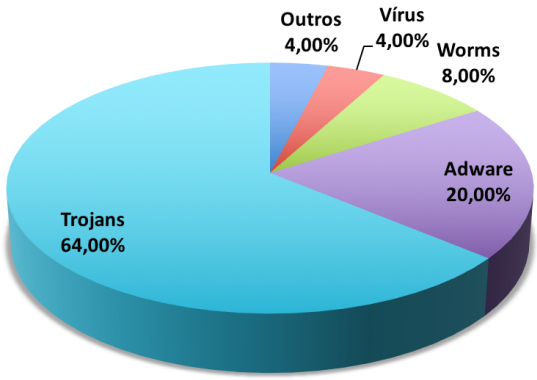
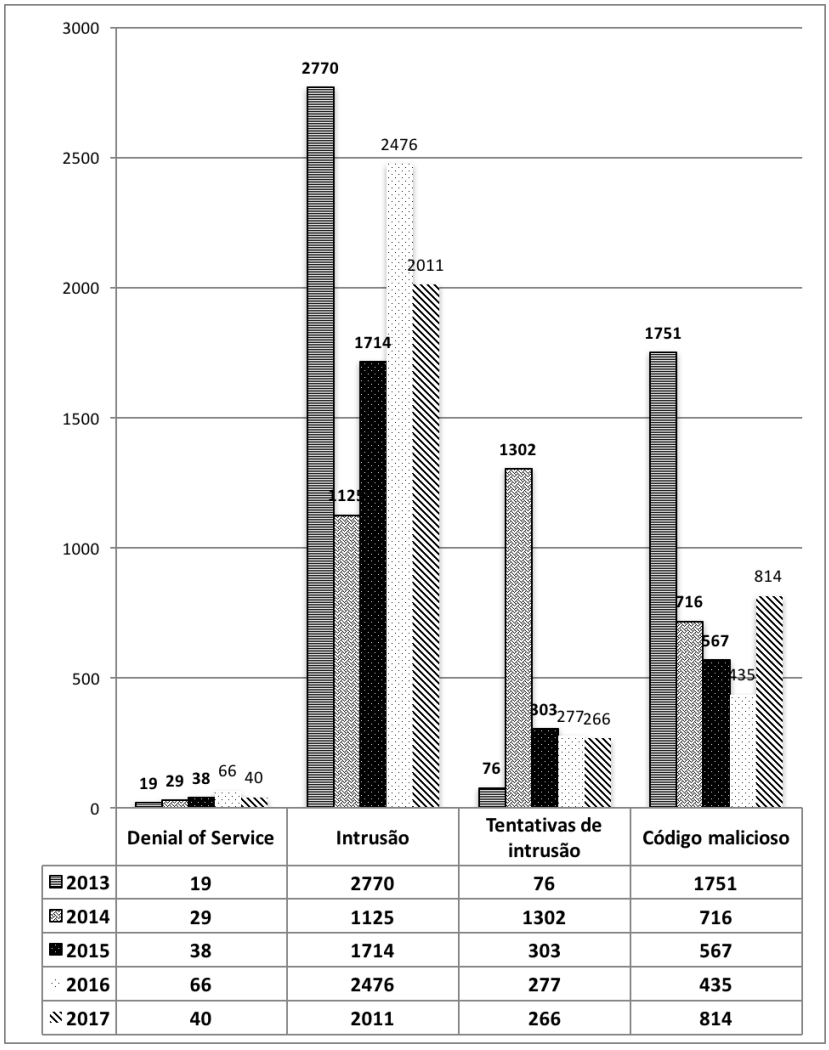


Figura 2.2: Tipos de *malware* mais frequentes de 2013 (adaptada de [13]).

O *DoS/DDoS* é o tipo de ataque mais antigo, utilizado para incapacitar uma rede ou servidor através da sua inundação com tráfego, levando a que fique inacessível (parcial ou completa). Existem dois tipos, o *DoS*, quando o ataque é levado a cabo por um *host* apenas, ou *DDoS* quando apenas um *host* utiliza várias máquinas, infetadas, de qualquer parte do mundo para inundar a rede alvo. A estas máquinas dá-se o nome de máquinas *Zombie* ou *Bots* estando à inteira disposição do atacante pois foram posteriormente infetadas com *malware*. Uma rede de ataque constituída por *Zombies/Bots* é intitulada por *Botnet*, sendo de grande escala constituída por cerca de 100,000 *Zombies/Bots* gerando entre 10Gb a 100Gb de tráfego por segundo [2].

Conforme se pode observar no gráfico da Figura 2.3, que representa a ocorrência dos ataques mais comuns em cada ano (de 2013 ao 1º semestre de 2015) na Malásia, os ataques têm vindo a decrescer ao longo do tempo. Observamos também que quando o número de ocorrência é elevado, no ano seguinte, para esse ataque, o número reduz significativamente. Daqui pode retirar-se que os mecanismos utilizados para os combater têm sido reativos e têm-se adaptado às circunstâncias, permitindo este tipo de resposta. Por outro lado, e considerando as referências utilizadas como exemplo (ver Figura 2.3), podemos também concluir que o nível de ameaça dos ataques *DoS/DDoS* reduziu substancialmente, resultado de mecanismos bastante aperfeiçoados para combater um tipo de ataque com um método de ação que se manteve estável com o passar do tempo.

[1] Vulnerabilidade criada pelos *trojans*. Ponto, frágil, de entrada para futuros ataques.



	Denial of Service	Intrusão	Tentativas de intrusão	Código malicioso
2013	19	2770	76	1751
2014	29	1125	1302	716
2015	38	1714	303	567
2016	66	2476	277	435
2017	40	2011	266	814

Figura 2.3: Evolução do número de ataques entre 2013 e 2017. Dados do *MyCERT* [6, 7, 8, 9, 10].

De todos os ataques acima referidos podemos ressalvar que todos têm algo em comum, o modo ocorrência. Todos esses ataques seguem um conjunto de passos idêntico, aquando do processo de ataque:

1. **Recolha de informação** - recolha de informação das vulnerabilidades da rede a atacar [1, 2];

2. **Pré-ataque** - ataque de alguns nós, tendo em conta a informação reunida no ponto 1 [1, 2];

3. **Ataque** ao alvo (máquina) pretendido [1, 2];

4. **Limpeza** - eliminação de registos e ficheiros log (da máquina) que possam identificar a origem do ataque [1, 2].

Conforme foi referido anteriormente, a melhor defesa para este tipo de anomalia passa pela análise regular do tráfego que circula na rede e pela frequente atualização dos dispositivos e serviços. Para além da constante análise, existe a necessidade de utilizar sistemas de deteção e prevenção de anomalias, tais como Intrusion Detection System (IDS), Intrusion Prevention System (IPS), *firewalls* e/ou sistemas de filtragem de conteúdos, como os *anti-spam* [1, 2].

Um IDS é uma aplicação que deteta tentativas intrusões e ataques a uma rede ou sistema. Um IDS pode ser passivo, detetando apenas esse tráfego e alertando os sistemas ativos. Pode também ser ativo, podendo chamar-se assim IPS, onde, além da deteção da intrusão, o IPS age em conformidade com a situação, rejeitando esses dados ou encerrando a ligação [14].

Uma *firewall* pode, também, ser um dispositivo ou aplicação que analisa o tráfego que circula, verificando se os seus dados, desde os endereços IP (origem, destino), portos, bem como o protocolo utilizado, estão de acordo com as políticas de segurança configuradas. Se as políticas não forem respeitadas os dados são rejeitados.

No caso da filtragem de conteúdos, e utilizando o *anti-spam* como exemplo, existem serviços destinados exclusivamente à análise e filtragem dos dados presentes em cada email. Esta filtragem é feita com base em regras pré-configuradas tendo em conta perfis de *SPAM* já existentes.

Estes ataques/anomalias para além da sua maioria ser causado intencionalmente por *Blackhat Hackers*[2], podem também ser causados por utilizadores comuns de forma involuntária. Estes, através de emails (*SPAM*) que possam ter recebido contendo ficheiros maliciosos, ou através do download de ficheiros em fontes duvidosas podem desencadear um ataque à sua máquina, ou rede, sem intenção [1, 2].

Com este trabalho, pretende-se criar e explorar mecanismos que possibilitem a um administrador de rede testar as soluções de prevenção, contra ataques e intrusões, implementadas. Estes mecanismos são essencialmente a criação de uma aplicação que permite a criação automática de tráfego de rede (criação de datasets) recorrendo a estudo estatístico com o objetivo de tornar os dados, e sua distribuição ao longo do tempo, o mais proximo possível da realidade contando também que este é parametrizável de acordo com as necessidades e ambiente de rede pretendidos.

[2]Hacker que age com o objetivo de prejudicar a rede/sistema atacado e os seus utilizadores.

2.2 Intrusion Detection System (IDS)

Uma das formas utilizadas para detetar e reagir a ataques, consiste na utilização de Intrusion Detection System (IDS). Um IDS é um ponto importante na segurança de um sistema que ocorre sobre forma de software, sendo categorizados de várias formas tendo em conta o tipo de tecnologia usado:

- **Host-based IDS (HIDS)** - recolhe e monitoriza o tráfego transacionado pelo *host*(máquina) e seus serviços, detetando atividade suspeita. Recolhe também informações da própria máquina com o objetivo de perceber o que nela está configurado e a correr [15, 16];

- **Network-based IDS (NIDS)** - presentes na rede recolhendo e analisando o tráfego que nela circula, analisando assim toda a atividade desenvolvida pelas aplicações e protocolos com o objetivo de encontrar atividade suspeita [15, 16];

- **Wireless-based IDS (WIDS)** - semelhante ao NIDS, é responsável pela recolha de tráfego de redes *wireless* [15, 16];

- **Network Behaviour Analysis (NBA)** - responsáveis pela deteção de ataques na rede, pela análise de fluxos de tráfego anormais [16];

- **Mixed IDS** - utiliza vários tipos de tecnologia (das acima mencionadas), preenchendo os requisitos específicos para o ambiente de implementação pretendido [16].

Para além do tipo de tecnologia que cada IDS utiliza, devemos também considerar o método e tipo de ação que este utiliza para análise e resposta aos vários tipos de ocorrências. Estes, podem então, dividir-se em 2 categorias principais:

- **Deteção baseada na assinatura (Signature-based Detection (SD))** - compara o padrão do ataque sofrido com outro já conhecido e catalogado. Por vezes também é chamada de deteção baseado no conhecimento (*Knowledge-based detection*) ou deteção de utilização abusiva (*misuse detection*) [15];

- **Deteção baseada na anomalia (Anomaly-based Detection (AD))** - baseia-se no comportamento adotado pelo ataque, comparando os seus fluxos com os perfis de utilização normal. Pode também ser designada de deteção baseada no comportamento (*behavior-based*) [15].

A Tabela 2.1 apresenta uma análise destas duas categorias. Podemos concluir que para deteção de ataques já conhecidos a escolha deva recair para os IDS baseados em assinatura, sendo esta a solução mais simples e eficaz a aplicar.

Por outro lado, a atualização e manutenção das regras de deteção e base de dados de assinaturas torna-se um pouco difícil e demorada. Se o objetivo for a deteção de ataques "novos", não conhecidos/catalogados, a escolha deve recair sobre um IDS baseado na anomalia, este recorre à comparação dos fluxos com perfis de utilização normal possibilitando a deteção de ataques ainda não catalogados (sem assinatura/padrão registado) embora seja necessário reiniciar o IDS para possibilitar a (re)construção dos perfis e os alertas que este produz sejam um pouco mais retardados no tempo que os de tipo SD.

Tabela 2.1: Comparação dos métodos de deteção dos IDS [16].

	Prós	**Contras**
Baseado na assinatura	• Simples e eficaz na deteção de ataques conhecidos. • Análise contextual detalhada.	• Pouco eficaz na deteção de ataques desconhecidos. • Pouco eficaz na deteção de variantes de ataques conhecidos. • Difícil manter a base de dados de assinaturas atualizada. • Manutenção de regras consome muito tempo.
Baseado na anomalia	• Eficaz na deteção de novos ataques. • Pouco dependente do SO. • Fácil deteção de abuso de privilégios.	• Perfiz pouco exatos ao longo do tempo. • Traçar e analisar estados consume muitos recursos. • Sistema fica em baixo durante construção de perfis. • Alertas retardados no tempo.

Neste trabalho foram analisados três IDS, o *Snort*, o *Suricata* e o *Bro*. Os três IDS são desenvolvidos utilizando o formato Packet CAPture (PCAP) disponível pela API libpcap e utilizam regras, ou mecanismos semelhantes, para a deteção de ataques com base em assinaturas/padrões de ataques conhecidos.

2.2.1 *libpcap*

O *libpcap* é uma biblioteca *open-source* utilizada em sistemas *Unix* para implementação da *API Packet Capture (PCAP)*, em sistemas *Windows* adquire o nome *WinPcap*. A *PCAP*, como o nome indica, é utilizado para capturar e transmitir pacotes de e para a rede. Embora tenha sido criada para as linguagens de programação C e C++, existem vários *wrappers*[3] para possibilitar a sua implementação com outras linguagens como *Perl*, *Java*, *C#*, etc [17].

Os ficheiros de captura criados por uma aplicação que utilize *libpcap* apresentam-se com a extensão `.pcap`, nestes ficheiros estão guardados, de forma estruturada, todos os pacotes capturados, conforme demonstra a Figura 2.4.

[3]Nome dado a uma classe que contém uma instância de outra classe. Muitas vezes usado para modificar a interface de uma *API* para esta ser utilizada numa linguagem diferente da nativa.

Figura 2.4: Estrutura dos dados de um ficheiro de captura *libpcap* (adaptado de [18])

O ficheiro de captura é constituído por duas partes [18]:

1. **Cabeçalho global** - é este cabeçalho que abre o ficheiro de captura, sendo seguido por zero ou mais registos de pacotes. Neste cabeçalho está definida uma estrutura, como exemplifica a Figura 2.5, com a definição dos formatos e definições de captura:

   - ***magic_number*** - define o formato do ficheiro e a ordenação de bytes;
   - ***version_major*** - define a versão do ficheiro (atualmente na 2.4);
   - ***version_minor*** - define a versão do ficheiro (atualmente na 2.4);
   - ***thiszone*** - diferença temporal em segundos entre GMT e hora local;
   - ***sigfigs*** - precisão dos *timestamps* da captura;
   - ***snaplen*** - tamanho máximo de captura de pacotes;
   - ***network*** - define o tipo de cabeçalhos que serão capturados.

```
typedef struct pcap_hdr_s {
        guint32 magic_number;   /* magic number */
        guint16 version_major;  /* major version number */
        guint16 version_minor;  /* minor version number */
        gint32  thiszone;       /* correção da hora de GMT para local */
        guint32 sigfigs;        /* precisão dos timestamps */
        guint32 snaplen;        /* tamanho máximo dos pacotes capturados, em octetos */
        guint32 network;        /* tipo de ligação de dados */
} pcap_hdr_t;
```

Figura 2.5: Estrutura de definição do cabeçalho global do ficheiro *libpcap* (adaptado de [18])

2. **Pacotes capturados** - Os pacotes capturados podem não conter a totalidade dos seus dados, sendo normal serem capturados apenas os primeiros *N* bytes de cada pacote, onde N corresponde ao *snaplen (snapshot lenght)*. O valor de *N* deve ser maior que o maior pacote possível de forma a garantir que nenhum pacote é cortado a meio durante a captura, assim 65535 bytes é o valor frequentemente utilizado. Tal como acontece no cabeçalho global, também aqui existe uma estrutura, como a da Figura 2.6, onde se encontram as definições dos pacotes da captura:

- ***ts_sec*** - horas e data de captura do pacote, valor expresso em segundos a contar a partir de 1 de Janeiro de 1970 às 00:00:00 GMT;
- ***ts_usec*** - tempo de captura em microsegundos, a contar desde o início da captura;
- ***incl_len*** - número de bytes capturados e guardados no ficheiro. Este valor não deve ultrapassar os valores de *origin_len* e *snaplen*;
- ***orig_len*** - tamanho real do pacote aquando da captura. Só é capturado no seu tamanho real, caso seja igual ou inferior ao valor do *snaplen*.

```
typedef struct pcaprec_hdr_s {
        guint32 ts_sec;         /* timestamp segundos */
        guint32 ts_usec;        /* timestamp microsegundos */
        guint32 incl_len;       /* números de octetos do pacote guardado no ficheiro */
        guint32 orig_len;       /* tamanho original do pacote */
} pcaprec_hdr_t;
```

Figura 2.6: Estrutura do cabeçalho dos pacotes capturados do ficheiro *libpcap* (adaptado de [18])

O *libpcap* é o a biblioteca padrão para a captura de pacotes, devido à simplicidade do seu formato e compatibilidade com os demais sistemas operativos (recorrendo a *wrappers*). Apesar de tudo isto, e com base na bibliografia utilizada [18], apresenta algumas limitações tais como a escassa informação sobre a interface de captura ou os comentários do utilizador, existindo a versão *pcapng* que já implementa algumas destas lacunas.

2.2.2 Snort

O *Snort*[4] é uma aplicação de segurança de rede *open-source*, baseado em *libpcap*, podendo atuar como *sniffer* e/ou *logger* de pacotes e, também, como NIDS/NIPS. Este é IDS *single-threaded*, utilizando apenas um núcleo de CPU de cada vez para o processamento dos dados que recebe. Recorre a um conjunto de regras para detetar e identificar padrões no tráfego correspondentes a ataques conhecidos, sendo assim identificado como Signature-based Detection (SD) IDS.

Esta deteção é realizada em tempo real sendo os alertas registados no *log* do sistema, ou então num ficheiro á parte se assim for desejado. Para além da base fornecida pelo Snort existem vários complementos (*add-ons*) que podem ser utilizados sobre este IDS adicionando-lhe capacidades extra, que este não possuía, necessárias para um cenário especifico. Um

[4] https://www.snort.org

exemplo é o facto destes IDS comunicarem via TCP, mas, caso seja necessário que este comunique utilizando outro protocolo basta utilizar um complemento específico para satisfazer essa necessidade [19, 20].

O *Snort* prima pelo seu desempenho e simplicidade de utilização sendo esta a base da sua arquitetura, que está dividida em três mecanismos importantes para garantir sucesso no seu funcionamento. Quando em modo de *sniffer* ou *logger*, o *Snort*, utiliza o descodificador de pacotes, o mecanismo de deteção e ainda nos mecanismos de registo e alerta. No caso de estar a ser utilizado no modo NIDS, este possui ainda um sistema de preprocessamento dos pacotes, antes de estes serem processados pelo mecanismo de deteção.

Assim que o pacote chega à interface destino necessita de ser descodificado, possibilitando a sua interpretação. Após esta descodificação é possível perceber qual o protocolo utilizado pelo pacote, sendo assim comparado com o perfil normal utilizado pelos pacotes daquele protocolo. Caso exista algum problema com o pacote, como por exemplo, ser demasiado longo ou ter alguma malformação no seu cabeçalho, é gerado um alerta [20].

Partindo do pressuposto que está tudo bem com o pacote e que o *Snort* está a ser utilizado como NIDS, o pacote é então reencaminhado para os preprocessadores para que estes analisem o conteúdo do pacote verificado a sua veracidade. Um preprocessador do *Snort* não é mais que um *plug-in* que analisa os pacotes, evitando que ataques camuflados sigam o seu caminho. Estes preprocessadores permitem ainda que a análise e deteção dos ataques seja rápida. Isto é possível pois o despiste realizado pelos preprocessadores permite reduzir o número de ataques identificado indevidamente (falsos positivos), quando ocorrem padrões de ataque muito gerais [20]. O *Snort* encontra-se à data da escrita deste documento, na versão 2.9.7.6 e conta com cerca de 24 preprocessadores implementados (Anexo A). Apresentam-se de seguida dois exemplos:

- **ARP Spoof Preprocessor** - descodifica pacotes ARP para deteção de ataques ARP, pedidos ARP unicast e inconsistências no mapeamento ethernet-IP [21];

- **SMTP Preprocessor** - descodifica pacote SMTP de aplicações do utilizador para detetar os comandos e respostas SMTP. Efetua processamento statefull e stateless, guardando o estado de pacote para pacote [21].

Para além dos preprocessadores existentes, poderão ser criados outros consoante as necessidades do administrador do sistema.

O mecanismo de deteção é o componente principal do *Snort* como NIDS. Após a descodificação e análise ocorridos nos mecanismos de descodificação e preprocessamento, os dados

são comparados com as regras configuradas no ficheiro de configuração do *Snort*, conforme ilustra a Figura 2.7. Assim que os dados chegam ao mecanismo de deteção, é efetuada uma análise aos dados na tentativa de identificar as suas características e encontrar o conjunto de regras que melhor se aplica a esse tipo de pacotes. Assim que as regras são encontradas, estas são aplicadas aos dados e estes seguem o caminho determinado pela(s) ação(ões) configurada(s) na(s) regra(s) [20].

```
# site specific rules
include $RULE_PATH/local.rules

include $RULE_PATH/app-detect.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/backdoor.rules
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/blacklist.rules
include $RULE_PATH/botnet-cnc.rules
include $RULE_PATH/chat.rules
include $RULE_PATH/content-replace.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/experimental.rules
include $RULE_PATH/exploit-kit.rules
include $RULE_PATH/exploit.rules
```

Figura 2.7: Regras configuradas no ficheiro de configuração do *Snort*

Após a aplicação das regras, os pacotes a que foram aplicadas, são registados sendo criado um alerta relativo a essa deteção. Estes alertas são totalmente configuráveis no ficheiro de cada regra do *Snort*, permitindo definir como e quando é que serão desencadeados, permitindo definir também quando é que devem ser ignorados. Estes alertas, tal como acontece com os pacotes, podem ser arquivados. Os sistemas de alerta e registo são *plug-ins*, permitindo adaptar as respostas obtidas ao ambiente de administração utilizado [20].

As regras permitem ao *Snort* a deteção rápida de ataques conhecidos, sendo construídas com base em informação obtida de ataques já ocorridos. As regras podem ser obtidas numa base de dados de regras, como a oficial do *Snort*, e podem também ser criadas pelo administrador de acordo com as necessidades específicas de cada situação.

As regras são constituídas por duas partes lógicas, conforme ilustra a Figura 2.8:

- **Cabeçalho** - onde estão definidas as ações que a regra vai executar, o protocolo a verificar, IP e porto origem e destino bem como as máscaras de rede correspondentes;

- **Opções** - onde se define a mensagem de alerta e que parte do pacote deve ser analisada, é este último campo que define se a ação da regra deve, ou não, ser aplicada.

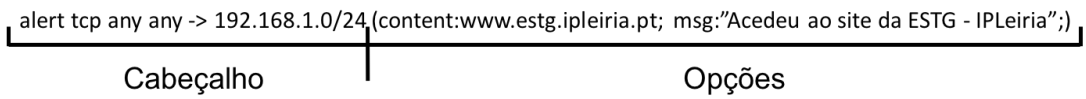


Figura 2.8: Exemplo de regra do *Snort*.

A regra apresentada em cima, lança uma mensagem de alerta sempre que o endereço *web* presente no campo `content` é acedido.

As regras do *Snort* permitem a realização de 5 ações principais, conforme apresentado na Tabela 2.2. Quando é encontrado um pacote que corresponde ao previsto na regra, estas ações definem também o seu tipo [21].

Tabela 2.2: Ações das regras do *Snort*

Ação	**Descrição**
alert	Cria um alerta e de seguida regista o pacote no log
log	Regista o pacote
pass	Ignora o pacote
activate	Cria um alerta e ativa outra regra dinâmica (dynamic)
dynamic	Ativada por uma regra activate, atua como regra de registo (log)

Para além dos tipos de regras predefinidos, é possível criar outros de acordo com a necessidade, permitindo assim a melhor adaptação possível da solução implementada. O código seguinte ilustra um exemplo de um tipo de regra novo, que regista o pacote no ficheiro *suspicious.log*, sempre que esta regra é invocada. Esta regra, pode eventualmente, ser invocado por outras.

```
ruletype suspicious
{
type log
output log_tcpdump: suspicious.log
}
```

A versão atual do *Snort*, permite analisar pacotes dos protocolos TCP, UDP, ICMP e TCP(v4 e v6), estando previsto a análise pacotes de outros protocolos em futuras versões, tais como os de protocolos de encaminhamento (OSPF, RIP, IGRP, etc) [21]. Para definir o IP, ou gama de IP, a analisar utiliza-se a notação standard `IP/Prefixo` (ex: 192.168.1.0/24), este campo pode ainda ser utilizado como `'any'` indicando que analisa qualquer IP. No campo do porto de comunicação basta indicar o número do porto escolhido, ou então `'any'`

analisando os pacotes de qualquer porto. Por fim, o cabeçalho, possui ainda 2 operadores de direção que indicam a direção do tráfego a analisar, o `'->'` é utilizado para tráfego da origem para o destino e o `'<>'` para tráfego bidirecional considerando tráfego da origem->destino e vice-versa [21].

Em relação às opções, estas dividem-se em 4 categorias principais de opções [21]:

- **Gerais** - fornecem informações sobre a regra não possuindo qualquer efeito na deteção;
- **Payload** - analisam os dados transportados no payload do pacote;
- **Non-payload** - analisam todos os dados que não pertencem ao payload do pacote;
- **Pós-deteção** - estas opções definem ações a realizar após a ativação da regra em questão.

Estas regras permitem ainda definir um *threshold*, impondo limites nas ações realizadas. Este limite pode ser temporal ou por número de tentativas [21].

O *Snort* é uma aplicação fortemente documentada com descrições explicativas de cada funcionalidade e parâmetro disponibilizado. Conta também com exemplos passo-a-passo de como utilizar e configurar cada funcionalidade e modo de operação. Apresenta-se como uma aplicação de referência e com uma vasta comunidade de utilizadores.

2.2.3 Suricata

O *Suricata*[5] é um IDS/IPS Signature-based Detection (SD), *open-source*, baseado em regras externas, para monitorização e alerta de atividade maliciosa/suspeita no tráfego da rede onde se encontra. Possui uma forte compatibilidade com os equipamentos de segurança na rede e, possibilita também a interação com outras aplicações. Ao contrário do *Snort*, o *Suricata* é um sistema *multithreading* utilizando os vários núcleo do CPU para processamento dos dados, a aceleração via hardware é também uma característica deste IDS tornando as suas análises mais rápidas e eficientes [22].

O *Suricata* permite monitorizar e registar o tráfego da rede onde se encontra configurado e pode ainda ser configurado como NIDS/Network-based IPS (NIPS). Este é também fortemente compatível a nível de SO, possibilitando a sua configuração em Linux, FreeBSD, OpenBSD, Mac OS X e Windows, disponibilizando versões para cada SO. Possui vários

[5] `http://suricata-ids.org`

mecanismos que lhe permitem analisar e detetar atividade suspeita nos diferentes protocolos utilizados pelos pacotes que circulam na rede, nomeadamente:

- **Mecanismo TCP/IP** - mecanismo de fluxo escalável que fornece total suporte de IP(v4 e v6) e permite ao *Suricata* descodificar tráfego de túneis GRE, IPv4-IPv6, IPv6-IPv4, IP-IP e Teredo. Este possui um mecanismo que permite detetar sessões de transmissão TCP, possibilitando a reconstrução do fluxo/sessão [23];

- **Mecanismo analisador de protocolo** - fornece suporte à descodificação de pacotes IP(v4 e v6), TCP, UDP, SCTP, ICMP(v4 e v6), GRE, Ethernet, PPP, PPPoE, Raw, SLL, VLAN, QINQ. Permite também descodificação de HTTP, SSL, TLS, SMB, SMB2, DCE/RPC, SMTP, FTP, SSH e DNS ao nível da camada de aplicação. Este mecanismo possui também preprocessadores de identificação automática do protocolo a ser utilizado, tornando assim a análise mais rápida e com menos erros [23].

- **Mecanismo HTTP** - permite analisar HTTP Stateful e registar os pedidos HTTP realizados na rede. Permite extrair, identificar e registar ficheiros e possibilita definir as configurações de servidor tais como o limite de pedidos e tempo de resposta [23].

- **Mecanismo de deteção** - é neste mecanismo que se efetua a deteção de atividade maliciosa na rede, esta deteção é feita pela comparação de regras e ficheiros com os pacotes. Este mecanismo possibilita a utilização de algoritmos de padrão múltiplo bem como a actualização das regras utilizadas sem a necessidade de reiniciar o *Suricata*. É possivel ainda a aceleração da comparação de padrões através de CUDA GPU [23].

Tal como outros IDS, este também possui um sistema de alertas. É gerado um alerta por cada regra correspondente ou por thresholding, este alerta pode estar associado exclusivamente a uma regra, ou ao global. É ainda gerado um alerta quando os limites, host/subrede, configurados são atingidos [23].

Como referido anteriormente, este IDS, permite funcionamento multithreading que lhe permite distribuir o processamento por vários *cores*/núcleos, consoante a necessidade. Este IDS utiliza ainda um mecanismo de reputação IP (*IP Reputation*), permitindo identificar, no tráfego de rede, origens maliciosas com base em dados do host. Estes dados podem ser recarregados a qualquer momento sem necessidade de reiniciar o sistema [23].

2.2.4 *Bro*

O *Bro*[6] é um monitor e analisador, *open-source* baseado em *libpcap*, de tráfego de ligações permitindo a deteção de atividade suspeita, funcionando também como Signature-based Detection (SD) IDS. Funciona sobre uma arquitetura single-threaded tal como o *Snort*. Para além desta deteção, o *Bro*, permite também executar tarefas de avaliação de desempenho e ajuda na resolução de problemas. Permite análises *offline*, suporta IPv6 e análise de tráfego de túneis através do desencapsulamento do seu tráfego para posterior análise. Resultante da sua análise e monitorização, surgem vários ficheiros log com informações de tráfego de alto nível, nestes ficheiros está registada toda a informação correspondente a cada ligação realizada, bem como da atividade ocorrida a nível da camada de aplicação tais como sessões HTTP, certificados SSL, etc. Estes logs estão devidamente estruturados para possibilitar o seu processamento por parte de software externo, ou utilização destes noutros formatos ou interfaces [24].

Esta plataforma assenta sobre um domínio de linguagem de scripting proprietária com um modelo baseado em eventos, disponibilizando um conjunto de funcionalidades pre-construídas e possibilitando a construção de outras por parte do administrador. Esta liberdade e tipo de linguagem possibilita a que esta seja uma plataforma parametrizável e extensível. Desta forma, a sua utilização como IDS está um pouco facilitada, pois a linguagem scripting que utiliza permite abordar a deteção de anomalias, análise semântica recorrendo a assinaturas e análise de comportamentos [24].

O *Bro*, possui uma arquitetura separada em dois componentes de maior importância, o mecanismo de eventos e o interpretador de scripts, conforme ilustra a Figura 2.9.

No mecanismo de eventos o fluxo dos dados é separado em eventos que representam a atividade ocorrida na rede. De seguida, estes eventos são enviados para o interpretador onde são interpretados através de *event handlers* criados via scripting. Este scripts representam as políticas de segurança implementadas, e caso seja detetada atividade que as viole, será criado um alerta em tempo-real que desencadeará a ação de contra-ataque. O estado da ligação pode ser mantido ao longo do tempo, possibilitando uma avaliação contínua das várias ligações, permitindo detetar e relacionar evoluções dos ataques entre elas [24].

Apesar de o *Bro* utilizar uma arquitetura single-threaded que limita um pouco o seu funcionamento e desempenho, existem alguns mecanismos e práticas para combater esta limitação. Assim, quando o limite de processamento de um processador (*single core*) é atingindo, o *Bro*, distribui as tarefas de igual forma, através de processos, para os restantes cores/núcleos

[6] `https://www.bro.org`

Figura 2.9: Arquitetura do *Bro* (adaptado de [24]).

ou até mesmo por diversas máquinas, permitindo assim uma maior capacidade de processamento [25].

Este *cluster* segue uma arquitetura base, conforme ilustra a Figura 2.10, constituída por vários pontos, cada um com a sua função [25]:

- ***Tap*** - este mecanismo divide o fluxo de pacote para criar uma cópia para inspeção. Este processo é semelhante ao de um divisor (*splitter*) ótico utilizado nas redes de fibra ótica;

- ***Frontend*** - dispositivo físico, ou software *on-host*, que divide o tráfego em vários fluxos. O *Bro*, não possui capacidades para realizar este processo, sendo necessária a utilização de software de terceiros (ex: cPacket, Netmap, etc);

- ***Manager*** - recebe mensagens e avisos (*log*) de todo o *cluster* combinando os vários ficheiros num só registo (*log*). Desta forma, é possível eliminar alertas duplicados e agir da forma mais correta, de acordo com os avisos recebidos;

- ***Proxy*** - sincroniza as variáveis através dos processos do *Bro*. Um exemplo de estado sincronizado passa pela existência de uma lista completa de *hosts* e serviços da rede, desta forma é facilitada a comunicação de informações de *hosts* e serviços da rede, entre os diversos '*workers*'. A utilização do *proxy* prende-se também com o aumento de desempenho, pelo facto dos seus processos serem 'leves' para o CPU e memória;

- ***Worker*** - é um processo que efetua a análise de tráfego e de protocolos aquando da restruturação do fluxo, sendo assim o ponto mais ativo do *cluster*. Assim, quanto melhor

for o hardware maior será o poder de processamento do *worker*. A este processamento é aplicado um balanceador de carga baseado em fluxos, distribuindo-o pelos vários workers existentes;

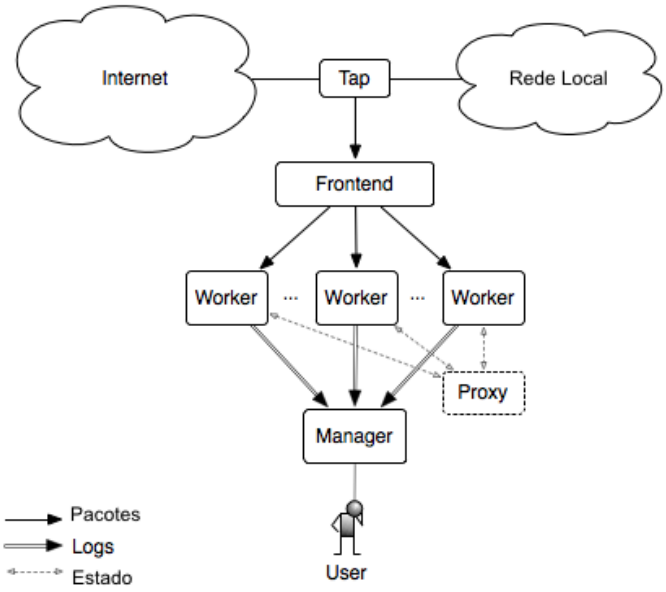


Figura 2.10: Arquitetura de *Cluster Bro* (adaptada de [25]).

Em relação aos *scripts*, estes, seguem uma estrutura base de implementação, em que podemos salientar três níveis, conforme demonstrado na Figura 2.11.

No primeiro, o nível base, incluem-se as bibliotecas necessárias e define-se o *namespace* do *script*. De seguida temos o segundo nível onde são criadas e devidamente identificadas e explicadas as variáveis, e constantes necessárias. No último nível, criam-se as instruções a seguir para o evento (`event evento_exemplo` na Figura 2.11) a que se destina o script [26].

A linguagem de *scripting* do *Bro* é focada na criação de mecanismos para tratar eventos criados pelo *Bro*, enquanto este processa o tráfego da rede manipulando a sua estrutura e decidindo o que fazer com eles com base na informação que recebe. Os eventos são ordenados pelo *core* do *Bro* numa lista de eventos seguindo uma base de processamento First Come, First Serve (FCFS). O *Bro*, como noutras linguagens de *scripting*, permite definir váriáveis globais e locais, constantes, estruturas de dados (ex: tabelas, vectores, etc).

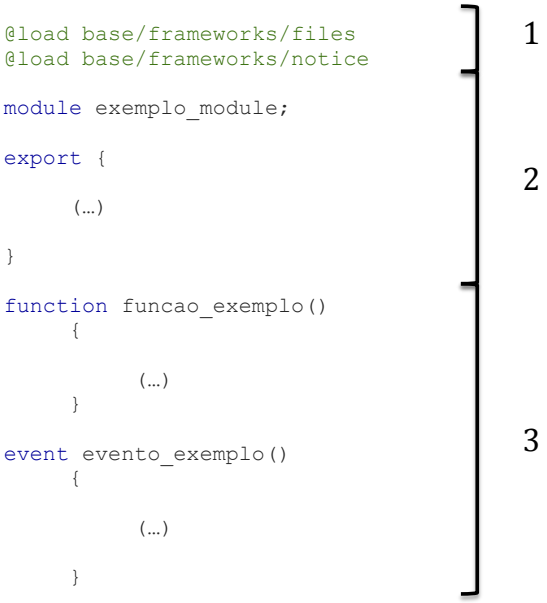


Figura 2.11: Exemplo dos três níveis de *script Bro*.

2.3 Análise comparativa dos IDSs

Um vez feita a descrição individual de cada IDS, existe a necessidade de os comparar, possibilitando assim a observação concreta das funcionalidades. A Tabela 2.3 apresenta um resumo comparativo das características dos IDS apresentados.

A motivação principal para a escolha de um IDS, para o nosso trabalho, passa por este ser open-source, sendo que todos os analisados respeitam este requisito. Em termos de desempenho o *Suricata* apresenta-se em melhor posição perante os outros IDSs devido ao facto de possuir uma arquitetura multithreading e possibilitar a aceleração dos processos recorrendo a processamento sobre hardware (ex. CUDA GPU). Tal possibilita uma análise dos pacotes significativamente mais rápida que nos outros.

No que diz respeito a funcionalidades específicas de análise e monitorização, todos possibilitam as mesmas três funcionalidades (*Sniffer*, *Logger* e NIDS/NIPS), sendo que por acréscimo o *Bro* possui ainda mecanismos extra que lhe permitem a avaliação de desempenho e *troubleshooting* da rede/máquina.

Para análise de tráfego, todos os IDS necessitam de usar um descodificador para aplicação posterior de mecanismos de avaliação próprios de cada IDS. No caso do *Snort*, este tráfego passa ainda pela análise de preprocessadores na tentativa de ataques falsos com o objetivo de ludibriar o sistema, abrindo assim a rede ao ataque verdadeiro.

Tabela 2.3: Funcionalidades de cada IDS

Funcionalidades	**Snort**	**Suricata**	**Bro**
Open-source	X	X	X
***Multithreading* nativo**	-	X	-
Baseado em *libpcap*	X	X	X
Aceleração por *hardware*	-	X	-
Sniffer	X	X	X
Logger	X	X	X
NIDS/NIPS	X	X	X
IP Reputation	-	X	-
Utilização de *addons/plugins*	X	-	-
Utiliza Preprocessadores	X	-	-
Regras ou *scripts* próprios	X	-	X
Alertas em tempo real	X	X	X
Mecanismo de avaliação de desempenho	-	-	X
Mecanismos de *troubleshooting*	-	-	X
Análises *offline*	X	X	X
Linguagem *scripting*	-	-	X
Criação de conteúdo (regras, *scripts*, *addons*, etc)	X	-	X

A avaliação do tráfego da rede tanto no *Snort* como no *Suricata* é realizada com base em regras, sendo que neste último as regras são provenientes de bases de dados externas (ex. base de dados Snort), permitindo ao administrador usufruir de um volume maior de regras. A avaliação de tráfego no *Suricata*, pode também ser realizada com base em mecanismos de *IP Reputation*. No caso do *Bro*, esta avaliação é realizada através de *scripts*, predefinidos ou criados pelo administrador do sistema, sendo que o mesmo acontece com as regras do *Snort*.

Em termos de liberdade de criação e aplicação, o *Snort* e *Bro* são os IDS que se destacam, permitindo aos seus administradores a criação de regras e *scripts* com o objetivo de aproximar a análise o mais possível do pretendido, sendo assim de fácil adaptação à maioria dos casos. No caso do *Snort*, existe ainda a possibilidade de criação de plugins/addons e preprocessadores.

Após esta análise conclui-se que para utilização em cenários mais específicos e onde se necessita utilizar regras e mecanismos fora do considerado normal disponibilizado pelos programadores dos IDS, o *Snort* e o *Bro* destacam-se do *Suricata* pela sua liberdade de criação de regras/scripts e funcionalidades extra. O *Snort* e *Bro* são totalmente independentes de terceiros, permitindo a utilização de qualquer tipo de conteúdo, desde que o formato seja respeitado, ao contrário do *Suricata* que é dependente de bases de dados de regras de

terceiros.

Para desempenho de funções tipo dos IDS (*sniffing, logging* e NIDS/NIPS), apresentam-se todos ao mesmo nível, salientando-se apenas a característica *multi-threading* o *Suricata* que lhe permitirá desempenhar essas funções mais rapidamente.

Por fim salienta-se as funcionalidades extra que o *Bro* possui (ex: *troubleshooting*), disponibilizando ao administrador funções extra que, com os outros IDS, necessitaria de recorrer a *software* externo a estes.

2.4 *Datasets* de *benchmarking* de rede

Os IDS representam um papel importante na proteção, de uma rede, contra várias atividades maliciosas. Por este motivo, estes sistemas devem ser precisos, apresentando um baixo número de falsos positivos (quando tráfego legítimo é identificado como malicioso), e uma taxa de intrusões elevada. Para aperfeiçoar esta deteção são utilizados *datasets* de *benchmarking* desenvolvidos para esta finalidade. Um problema existente nestes datasets consiste no facto de se encontrarem desatualizados para os dias de hoje, bem como pelo facto dos seus dados serem pouco precisos e realistas, não representando devidamente o tráfego real atual [27]. De acordo com a bibliografia utilizada [28, 29, 30, 27, 31], existem alguns *datasets* que pelas suas características específicas merecem especial atenção sendo considerados uma base de partida para trabalhos nesta área. A Tabela 2.4 apresenta um sumário comparativo dos *datasets* analisados no âmbito deste trabalho.

Dataset DARPA O *dataset* DARPA apresenta 3 versões (1998, 1999 e 2000), todas elas constituídas por dados simples de tráfego de rede e por dados de treino essencialmente constituídos por ataques e anomalias. A versão de 1998 serviu de base para as restantes. Os dados de treino foram gerados na versão de 1998 ao longo de 7 semanas, sendo que os diferentes ataques foram injetados na rede em diferentes momentos ao longo das semanas. De entre os ataques podemos salientar ataques de DoS, *buffer overflow*, *SYN Flood*, entre outros[7]. Ao nível de anomalias utilizaram-se acessos não autorizados, alterações suspeitas de permissões bem como acessos a horas fora do normal. No que diz respeito a problemas e inconsistências, este *dataset*, apresenta-os ao nível dos dados que o constituem, pois tanto os dados de tráfego normal como os de tráfego malicioso (ataque) foram adicionados ao *dataset* de forma artificial não sendo então considerado tráfego real. Desta forma, estes dados irão iludir o IDS

[7] `https://www.ll.mit.edu/ideval/docs/attacks.html`

não apresentando falsos positivos, levando a que o tráfego seja corretamente identificado na sua totalidade [27, 28].

Dataset KDD Cup 1999 O *dataset* KDD Cup 1999 foi criado, com base no processamento de registos de *tcpdump* do *dataset* DARPA de 1998, para ser utilizado na terceira competição *Knowledge Discovery and Data Mining Tools*. Esta competição pretendia a criação de um IDS que, recorrendo a este *dataset*, detetasse tráfego legítimo e tráfego malicioso. Este *dataset* surgiu após simulações realizadas em ambiente militar através da geração tráfego normal e tráfego de ataque sendo, posteriormente, unidos. Na lista de ataques estão presentes o DoS, buffer overflow entre outros também presentes no *dataset* DARPA 1998. O principal problema deste *dataset* é o facto de apresentar dados redundantes devido à fusão dos dois tipos de tráfego, este apresenta-se também pouco preciso e atualizado em comparação com tráfego real atual [27, 29].

Dataset CDX O *dataset* Cyber Defense eXercise (CDX) surge em 2009 através de uma competição de segurança na rede que lhe dá nome. Estes dados podem ser utilizados para um teste focado apenas nos ataques com o objetivo de testar as regras dos IDSs. Assim surge a premissa de que as competições de segurança constituem uma boa fonte de datasets para avaliação de IDSs. O desafio passa pelo reduzido número de competições deste tipo, levando a que o volume de dados e a sua diversidade sejam igualmente reduzidos. Falando no caso específico do CDX este apresenta um grande volume de ataques do tipo *rootkit*, em contrapartida apresenta um volume bastante reduzidos de ataques comuns do quotidiano [27, 30].

Dataset Kyoto Em 2009 foi também criado o *dataset* Kyoto com base na captura e análise de tráfego direcionado a *honeypots*[8], evitando assim a necessidade de marcação manual dos pacotes maliciosos. Por outro lado, devido ao tráfego ser direcionado apenas aos *honeypots* não existe tráfego de outras máquinas da rede limitando assim a visão que se tem desta. Também por este motivo não existem falsos positivos pois existe apenas tráfego malicioso [27].

Dataset USB ISCX 2012 Dos datasets referidos pela bibliografia surge, por fim, o *dataset* UNB ISCX 2012, criado, na universidade de New Brunswick, de forma dinâmica para representar tráfego normal e tráfego malicioso. Esta criação foi executada ao longo de 7 dias

[8] Servidores de engodo (chamariz) para recolha de informação sobre o atacante/intruso da rede [32].

com recurso a vários cenários de ataque multi-fase[9], responsáveis pela criação de tráfego de ataques (DoS, DDoS, Brute force SSH, etc), e por perfis de utilizador ativados de forma aleatória e síncrona criando assim tráfego normal minimamente real. O problema deste *dataset* é apenas a falta de tráfego pouco relevante que ocorre na Internet para tornar os dados mais idênticos ao real [27, 31].

Tabela 2.4: *Datasets* de *benchmarking* de rede.

Dataset	**Prós**	**Contras**
DARPA	Deteção de intrusão centrada nos dados	Injeção artificial de tráfego normal/ataque
		Tráfego não real
		Ausência de falsos positivos
KDD Cup 1999	Usado em competições de *data mining*	Tráfegos normal e ataque gerados isoladamente
	Dataset público	Tráfego unido posteriormente
		Ocorrência de dados redundantes
CDX	Proveniente de competição de segurança	Pouco volume de dados
	Focado no teste de regras	Pouca diversidade de dados
Kyoto	Captura dados direcionados a honeypots	Tráfego demasiado direcionado
	Sem marcação manual de pacotes maliciosos	Ausência de falsos positivos
UNB ISCX 2012	Criação dinâmica de tráfego	Ausência de ruído
	Dados minimamente reais	

Estes datasets fornecem um bom ponto de partida para o que se pretende desenvolver neste trabalho, permitindo assim ter uma percepção das falhas a colmatar e pormenores a evitar aquando do desenvolvimento do *dataset*. Assim é possível criar um *dataset* atual e suficientemente completo tendo em conta a autenticidade dos dados de rede, possibilitando o benchmarking de uma rede com recurso a um *dataset* artificial constituído por tráfego bastante semelhante ao real.

A criação do *dataset* pode recorrer a uma abordagem *flow-based* (baseado nos fluxos) ou *packet-based* (baseada nos pacotes). Considerando uma abordagem *flow-based*, o foco dos dados está na comunicação entre IP origem e IP destino, considerando os portos utilizados para comunicarem, a quantidade de dados transmitidos, as *flags* e ainda quando ocorreu esse fluxo. Esta abordagem é frequentemente aplicada em datasets de larga escala, com uma quantidade de dados elevada permitindo a deteção de anomalias através de alterações no fluxo, tais como quantidade de dados demasiado elevada ou até mesmo pela sua direção. Numa abordagem *packet-based*, os pacotes são tratados individualmente tendo o seu conteúdo como ponto importante da implementação existindo o controlo do tipo e quantidade de dados transmitidos. Esta será útil e aplicável para uma quantidade mais reduzida de dados

[9] Ataque com várias fases de ação, todas com o objetivo final de invadir o alvo.

em comparação com o método anterior e as suas anomalias são detetadas através da análise do conteúdo de cada pacote.

2.5 Conclusão

Após este capítulo, e considerando a bibliografia utilizada, não é possível identificar uma solução que possibilite o teste de aplicações de segurança de rede através criação de *datasets* artificiais. As falhas que os *datasets* atualmente apresentam servem de exemplo para o trabalho a desenvolver, com o objetivo de evitar comete-las, nomeadamente a reduzida diversidade de tráfego que estes apresentam.

O estudo dos IDS, presente na Secção 2.2, fornece, ainda, uma forte base de consideração para a escolha de mecanismos de segurança para teste, utilizando os *datasets* criados.

Desta forma, estão reunidas todas a condições para se planear e implementar a solução que se pretende atingir no final deste trabalho.

Capítulo 3

Arquitetura proposta

Neste capítulo encontram-se descritos todos os aspetos relevantes do desenvolvimento da aplicação associada ao tema deste trabalho. Estão descritos os requisitos e arquitetura da aplicação, bem como outros aspetos importantes considerados para o seu desenvolvimento.

3.1 Requisitos

A aplicação a criar pretende responder não só às necessidades de um administrador de sistemas/redes mas também às de um investigador na área da segurança das redes. Por este motivo considerámos dois grupos de requisitos que a aplicação deve respeitar, com o objetivo de disponibilizar todas funções específicas para cada cenário, tais como:

- **Administrador:**
  - Utilizar tráfego previamente recolhido de forma a que possa ser posteriormente utilizado na criação do *dataset* a ser injetado numa rede real ou em ambientes virtualizados;
  - Disponibilizar os pacotes recolhidos em formato `pcap`, para possibilitar a injeção direta na rede;
  - Disponibilizar a possibilidade de analisar o tráfego em fluxos e em pacotes IP;
  - Adicionar ataques, ao *dataset* gerado para obter a configuração objetivo de tráfego normal e ataques;
  - Adicionar novos ataques na aplicação para posterior adição ao *dataset*, deve respeitar o formato utilizado nos existentes;

  - Selecionar o método de injeção dos ataques;
  - Criar um *dataset* constituído por tráfego normal e ataques O *dataset* pode ser gerado em formato *pcap* ou em formato textual permitindo escolher apenas alguns campos e a sua disposição.

- **Investigador:**
  - Utilizar tráfego previamente recolhido de forma a que possa ser posteriormente utilizado na criação do *dataset* a ser injetado numa rede real ou em ambientes virtualizados;
  - O ficheiro de tráfego externo deve estar em formato `pcap`, para possibilitar a injeção direta na rede;
  - Definir template de formatação pretendida para os dados dos pacotes a serem gerados;
  - Criar *dataset* seguindo a estrutura definida, no template, pelo investigador;
  - Adicionar ataques, ao *dataset* gerado para obter a configuração objetivo de tráfego normal e ataques;
  - Adicionar novos ataques na aplicação para posterior adição ao *dataset*, deve respeitar o formato utilizado nos existentes;
  - Selecionar o método de injeção dos ataques;
  - Criar um *dataset* constituído por tráfego normal (ou artificialmente gerado) e ataques com formato textual.

3.2 Arquitetura

A aplicação que surgirá no âmbito deste projeto segue uma arquitetura de implementação constituída por vários processos que executam as suas tarefas de recolha e processamento do tráfego, estando distribuídos por módulos, assegurando a integração e funcionamento correto das suas funcionalidades, conforme demonstrado na Figura 3.1.

Na base do funcionamento da aplicação estão os ficheiros que contêm tráfego de rede considerado normal e tráfego de rede malicioso, constituído por ataques. O primeiro passo de implementação passa pela recolha deste tráfego gerando dois ficheiros isolados com os pacotes necessários para serem analisados e manipulados pelo *script* no passo seguinte.

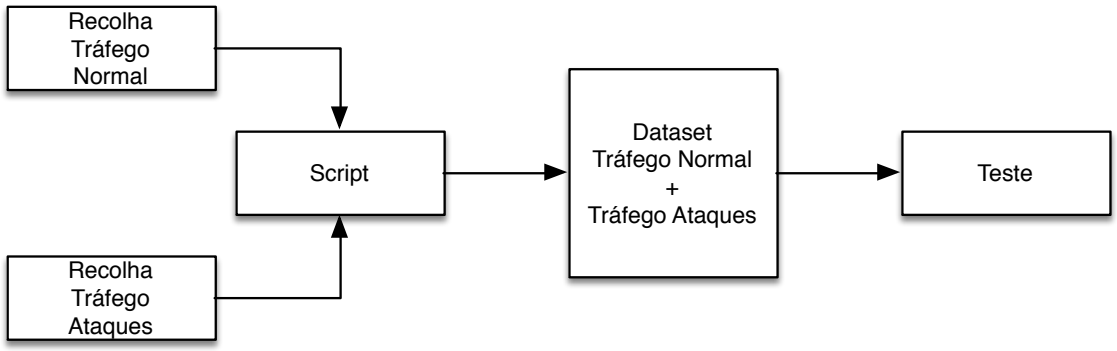


Figura 3.1: Arquitetura geral da aplicação.

Os ficheiros gerados são então analisados e manipulados pelo script analisando os seus pacotes um-a-um possibilitando assim o acesso aos dados que os constituem, tais como *IP* origem e destino, *timestamp*, entre outros. Os pacotes são analisados de um ponto de vista estatístico, permitindo perceber um padrão de comportamento ao longo do tempo para que, após esta análise, seja gerado um ficheiro (*dataset*) final constituído por tráfego de rede normal e por ataques, que respeita a distribuição encontrada.

O *dataset* gerado está, então, pronto para injeção na rede com o objetivo de testar aos soluções de segurança implementadas.

3.2.1 Recolha de tráfego

De acordo com o referido na secção anterior, a primeira fase para criação do *dataset* final passa pela recolha de tráfego de rede, bem como dos pacotes dos ataques.

A recolha do tráfego de rede normal deve ser efetuada sobre uma rede que esteja operacional, sendo esta uma rede real ou projetada e configurada em laboratório para esta finalidade. Para uma rede de laboratório é necessário que esta se assemelhe o mais possível com a realidade, sendo constituída por várias máquinas, dos vários tipos (computadores de utilizador/administração e servidores) possuindo também vários serviços configurados (email, *web*, etc). Também no cenário de rede laboratorial é necessário que existam mecanismos que garantam a ocorrência de tráfego diversificado, pois sendo uma rede artificial não existe atividade espontânea como se de uma rede real se tratasse.

Garantindo estes requisitos é necessário utilizar uma ferramenta de captura de pacotes de rede, sendo que nesta fase se tenha em conta que os pacotes devem ser capturados no formato *pcap* garantindo a possibilidade de injeção direta do *dataset* final na rede. Para além

da ferramenta desenvolvida neste trabalho, que também possibilita a captura de pacotes, sugere-se a utilização do *Wireshark*[1] ou do *tcpdump*[2]. Para ter uma melhor percepção do tipo de pacotes capturados e por forma a filtrá-los e analisá-los, a escolha deve recair sobre o *Wireshark* pois, disponibiliza uma interface gráfica que facilita todo este processo.

Para recolha dos pacotes de ataques, o processo é semelhante, e a ferramenta de captura utilizada pode ser a mesma utilizada na recolha de tráfego normal. A acrescentar ao processo de recolha de tráfego normal, nesta fase, é também necessário lançar ataques sobre a rede para que os seus pacotes sejam capturados. Uma vez lançados, são capturados estando prontos para serem identificados. Mais uma vez, para este processo é aconselhável o uso do *Wireshark*. Para facilitar a sua identificação são considerados os tempos em que o ataque foi lançado, bem como os *IP* origem e destino do ataque, assim que os pacotes forem identificados é possível isolá-los e exportá-los para um segundo ficheiro de captura contendo apenas os pacotes referentes ao ataque, estando prontos para adição ao *dataset* final.

Para efetuar a simulação dos ataques para a sua posterior captura e utilização no *dataset*, foi utilizado *Metasploit Framework*[3]. Esta framework é constituída por vários procedimentos *plug 'n play* que utilizam vulnerabilidades bem identificadas em diversas ferramentas, aplicações e sistemas operativos de forma a que seja possível testar a sua robustez. Os procedimentos implementados por esta framework necessitam de dados, fornecidos por quem a está a utilizar, para definir a origem e o alvo do ataque. Os dados podem ser, por exemplo, *IP* origem do ataque, *IP* da vítima, payload a enviar nos pacotes de ataque, ou apenas porto origem e porto da vítima conforme exemplo na Figura 3.2. A definição destes dados vai posteriormente facilitar a identificação dos pacotes de ataques no *Wireshark* aquando da captura.

A escolha do ataque a simular deve ter em conta o ambiente alvo, considerando atualizações de segurança que a aplicação/sistema operativo possa ter recebido para corrigir a vulnerabilidade pretendida o que poderá inviabilizar o teste.

3.2.2 Processamento de dados

A aplicação resultado deste trabalho consiste num *script*, desenvolvido na linguagem de programação *perl*, que executa todas as tarefas de processamento e análise dos dados recolhidos, conforme descrito na subsecção anterior. Para possibilitar a leitura e manipulação dos ficheiros *pcap*, é necessário utilizar um módulo especifico do *perl* criado para esse objetivo, o

[1] `https://www.wireshark.org/`
[2] `http://www.tcpdump.org`
[3] https://www.metasploit.com/

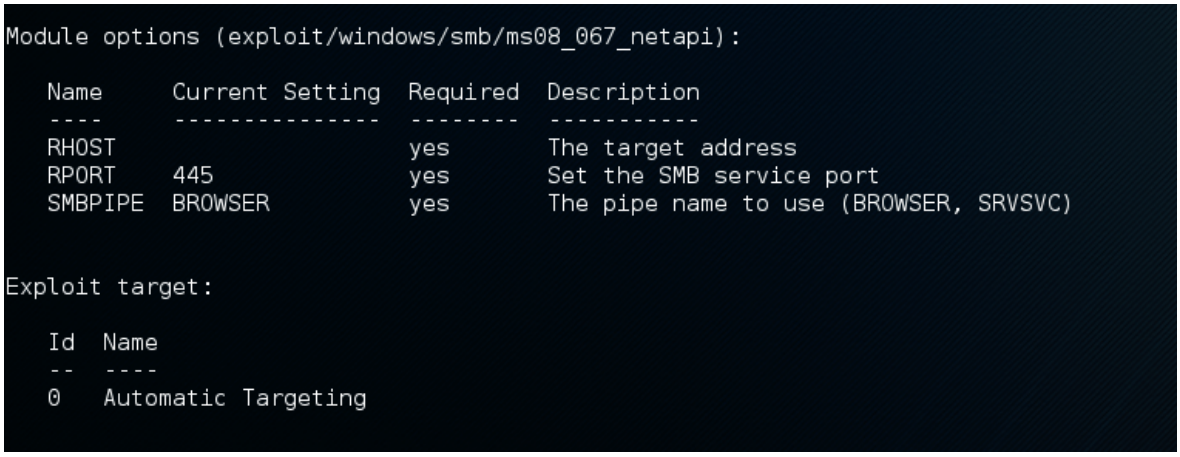


Figura 3.2: Opções de um ataque em *Metasploit Framework*.

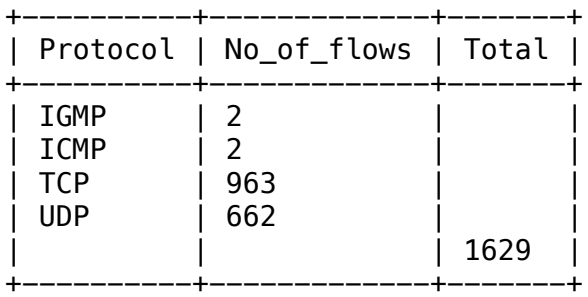
Net::Pcap[4]. Este módulo é necessário para o correto funcionamento do *script* pois permite a abertura e leitura de ficheiros *pcap* e também o processamento de pacotes.

Antes de qualquer tarefa de criação de um novo *dataset*, é conveniente para o utilizador da ferramenta ter noção do tipo de tráfego que ocorre na sua rede. Por este motivo, o *script* possui um módulo de estatística de tráfego analisando os dados de tráfego normal recolhidos considerando o fluxo de transmissão de pacotes, e ainda uma estatística de análise pacote a pacote.

Estatística por fluxo Neste tipo de estatística é apenas considerado o fluxo de pacotes criado entre dois pontos (origem e destino) descartando-se o seu conteúdo. O ficheiro de recolha é aberto para leitura e processamento, utilizando o *Net::Pcap* que vai analisar os dados de origem e destino de cada pacote de forma a serem agregados numa tabela de hashes. A tabela de hashes é constituída por uma **chave=[IP origem+porto,IP destino+porto]** que fica associada a todos os pacotes que lhe pertençam, **valor=pacotes**. Ao longo deste processamento são considerados os *timestamps* dos pacotes de forma a ser possível determinar o tempo de duração do fluxo, e são ainda considerados os protocolos *IGMP*, *ICMP*, *TCP* e *UDP*. O resultado final apresentado por esta funcionalidade corresponde a uma tabela semelhante à da Figura 3.3 que demonstra o número de fluxos por protocolo e ainda um conjunto de linhas com cada fluxo no formato ***IP origem:porto origem <-> IP destino:porto destino (número de pacotes):protocolo–duração***.

Para ser possível processar e analisar as características dos pacotes (*timestamp*, IP, porto e protocolo) é também necessário recorrer a módulos do *perl* específicos para esse função:

[4]https://metacpan.org/pod/Net::Pcap



Protocol	No_of_flows	Total
IGMP	2	
ICMP	2	
TCP	963	
UDP	662	
		1629

Figura 3.3: Exemplo de resultado apresentado pela análise estatística por fluxo.

- **NetPacket::Ethernet**[5] - fornece funções para desconstruir e construir pacotes *Ethernet*;
- **NetPacket::IP**[6] - fornece funções para desconstruir e construir pacotes *IP*. Permite aceder ao IP e protocolo utilizados;
- **NetPacket::TCP**[7] - fornece funções para desconstruir e construir pacotes *TCP*. Permite aceder ao porto de comunicação utilizado;
- **NetPacket::UDP**[8] - fornece funções para desconstruir e construir pacotes *UDP*. Permite aceder ao porto de comunicação utilizado;
- **NetPacket::ICMP**[9] - fornece funções para desconstruir e construir pacotes *ICMP*. Permite aceder ao porto de comunicação utilizado;
- **NetPacket::IGMP**[10] - fornece funções para desconstruir e construir pacotes *IGMP*. Permite aceder ao porto de comunicação utilizado;

Estatística por pacote Com esta estatística, pretende-se analisar todo o tipo de pacotes com um porto de comunicação compreendido no intervalo de portos de sistema e de utilizador [0-49151], possibilitando uma análise mais pormenorizada do tipo de tráfego que circula na rede. É considerado apenas este intervalo de portos, por serem portos fixos associados a serviços ou aplicações de sistema facilitando assim a identificação do serviço/aplicação a ele associado. O funcionamento desta estatística é semelhante à anterior, embora aqui o que

[5]https://metacpan.org/pod/NetPacket::Ethernet
[6]https://metacpan.org/pod/NetPacket::IP
[7]https://metacpan.org/pod/NetPacket::TCP
[8]https://metacpan.org/pod/NetPacket::UDP
[9]https://metacpan.org/pod/NetPacket::ICMP
[10]https://metacpan.org/pod/NetPacket::IGMP

interessa é aceder ao porto de comunicação do pacote e recolhe-lo. Os portos são recolhidos de todos os pacotes e guardados num *array*, esse mesmo *array* será processado para calcular o número de pacotes existentes para cada porto e para remover os portos repetidos para no fim ser apresentado o resultado estatístico conforme demonstra o exemplo da Figura 3.4. Nesta funcionalidade são também utilizados alguns módulos do *perl* que são também utilizados na análise estatística por fluxo, como é o caso do **NetPacket::Ethernet**, **NetPacket::IP**, **NetPacket::TCP** e **NetPacket::UDP**.

Port	No_of_packets	Total
53	567	
67	21	
68	32	
80	8771	
123	14	
137	140	
138	62	
139	108	
443	101932	
445	12	
1073	81	
1900	211	
4070	314	
5222	15011	
5223	17	
5353	174	
11793	6	
17500	478	
31445	15	
41800	45	
		233250

Figura 3.4: Exemplo de resultado apresentado pela análise estatística por pacote.

3.2.3 Geração de *dataset* para Administrador

O ponto crucial do *script* desenvolvido, e ponto mais importante deste trabalho, é a geração de *datasets* compostos por tráfego normal e por ataques.

A geração dos *datasets* baseia-se na análise estatística da amostra de tráfego previamente capturada e guardada em formato *raw* (pcap), conforme descrito na Secção 3.2.1, fornecendo assim os dados necessários para se proceder à geração de perfis de tráfego que servem de base

à geração de *benchmarks* de tráfego de rede.

A Figura 3.5 demonstra o algoritmo utilizado pelo *script* para processar os dados e criar *datasets* novos.

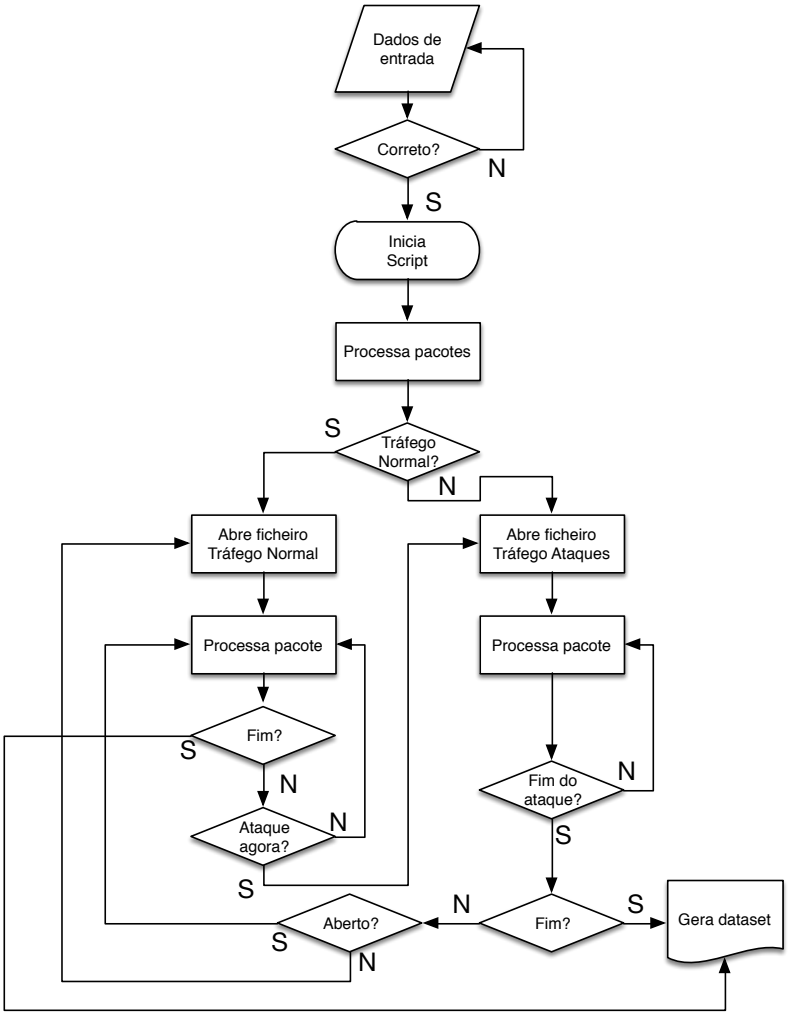


Figura 3.5: Algoritmo de ação da ferramenta na geração de *datasets* de administrador de rede.

Para iniciar a criação do *dataset* é necessário fornecer alguns dados de entrada que guiam todo o processo:

- **Ficheiro de tráfego normal** - ficheiro obtido através da recolha de tráfego de rede normal, servindo como base do *dataset*;
- **Ficheiro de ataque** - ficheiro de ataque a injetar no *dataset*. Poderão ocorrer vários ficheiros, correspondendo a vários ataques;

- **Frequência de geração** - define o número de pacotes pretendidos por segundo;

- **Tempo de geração de pacotes** - define a duração em segundos para gerar pacotes para o *dataset*. Este tempo é multiplicado pela frequência de geração, totalizando o número de pacotes que constituirá o *dataset*.

O primeiro passo após a captura do tráfego da rede é a análise e processamento dos pacotes capturados, servindo de base à construção dos perfis *N*, referentes ao tráfego de rede normal.

A análise e processamento dos pacotes passa por duas etapas principais: agregação de fluxos e construção dos perfis, compostas por várias subetapas, como apresentado na Figura 3.6 e descrito de seguida.

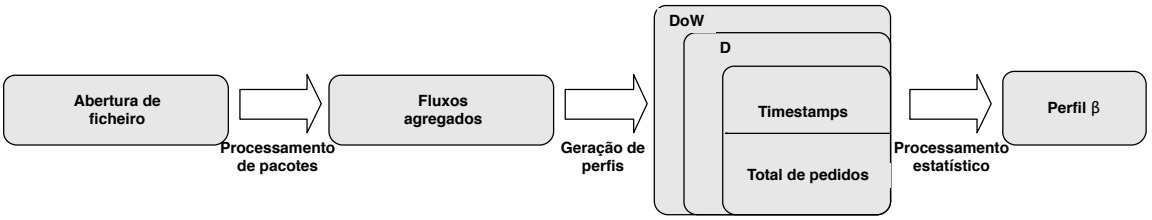


Figura 3.6: Etapas de construção de perfis *N*.

1. Agregação de fluxos Nesta etapa o ficheiro capturado é aberto para leitura e processamento dos fluxos. Todos os pacotes passam por uma fase onde são analisados os parâmetros de *IP*(origem e destino), porto (origem e destino) e protocolo de transporte (*TCP* e *UDP*). Estas informações são guardadas num *array* de *hashes* com a estrutura: *IP origem-porto origem -> IP destino-porto destino -> protocolo = pacotes* . Desta forma é possível identificar os fluxos para cada serviço utilizado na rede, permitindo efetuar um estudo mais detalhado e focado ao nível do serviço preferido para geração de tráfego para o *dataset*.

2. Construção dos perfis *N* Utilizando os fluxos agregados na etapa anterior, procede-se à construção dos perfis de tráfego *N* (tráfego normal). Este tipo de perfil não é mais que uma tabela de *hashes* onde os dados importantes para o perfil são guardados: dia da semana (DoW), identificador do utilizador (D), *timestamps* dos pedidos do utilizador à rede e número total de pedidos do utilizador naquele dia (ver Figura 3.7).

Os dados do perfil seguem uma norma, constituída por *DoW, D, Timestamps e Número de pedidos*, respeitada aquando da construção do perfil:

- **DoW** - identifica o dia da semana em que um determinado utilizador realizou os seus pedidos à rede. Representado por um valor compreendido de 1 a 7 (1 corresponde a segunda-feira e 7 a domingo);
- **D** - identifica o utilizador através do seu endereço *IP*
- ***Timestamps*** - engloba todos os *timestamps* relativos aos pedidos do serviço pretendido (ex: *http*);
- **Número de pedidos** - engloba o número total de pedidos efetuados pelo utilizador correspondente naquele dia.

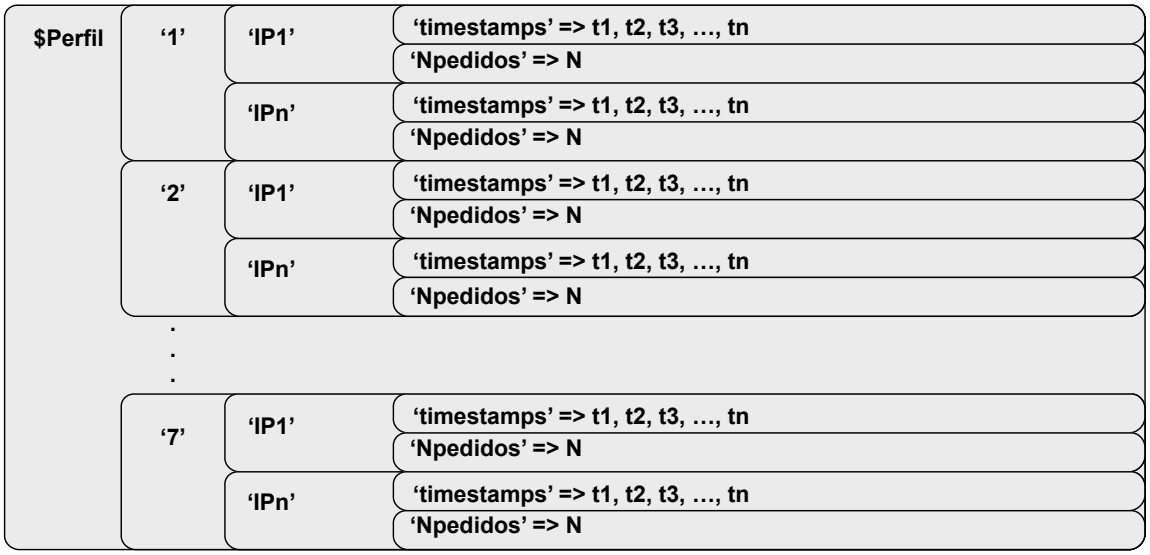


Figura 3.7: Estrutura de um perfil *N*.

Por fim todos os utilizadores (D) com o mesmo dia (DoW) são agrupados numa única entrada (chave) na tabela de *hashes*, formando assim o perfil *N* sobre o qual serão aplicados métodos estatísticos para estudar a distribuição do tráfego.

Distribuições de tráfego No que diz respeito à distribuição de tráfego que mais se adequa ao tráfego recolhido, observando o trabalho realizado por [33], não existe uma que defina o tráfego de uma forma constante ao longo dos dias, como tal são consideradas várias distribuições: Normal, Weibull, Gamma, Exponencial, Uniforme e Longnormal.

Considerando as distribuições nomeadas, existe necessidade de as testar para o tráfego de cada dia em específico determinando qual a que se adequa melhor à distribuição dos

dados. A distribuição que apresenta um erro quadrado (*square error*) menor é a que mais se adequa aos dados analisados, a todo este processo dá-se o nome de *fitting* (encaixe) e as várias ferramentas utilizadas para fins estatísticos (R, excel, Matlab, etc) dispõem de funções próprias para esta determinação.

Dos testes de *fitting* realizados sai uma distribuição que mais se adequa ao tráfego recolhido e a ela estão associados valores que a caracterizam tais como máximo, mínimo, desvio padrão, etc. São estes valores de caracterização da distribuição que serão usados para gerar pontos ao longo do tempo (*timestamps*) onde se situarão os pacotes do *dataset* final. Esta abordagem socorreu-se da ferramenta *R*[11] e das funções da biblioteca *fitdistrplus*[12] para fazer o *fitting* do tráfego e gerar pontos com base nos dados obtidos.

Geração de tráfego Após o estudo estatístico efetuado nas etapas anteriores, surgem vários pontos (*timestamps*) espaçados no tempo seguindo a distribuição obtida para o perfil *N*. Estes pontos necessitam de ser preenchidos com pacotes de tráfego de rede, podendo ser gerados seguindo determinados parâmetros definidos ou pode ser tráfego real gerado por pedidos efetuados por *web crawlers*. O pontos podem também ser preenchidos com pacotes do tráfego previamente recolhido, existindo apenas o cuidado de manter os fluxos intactos e em ordem. Nestes pontos constarão também pacotes de tráfego não legítimo (malicioso), recolhido em 3.2.1, para inclusão no *dataset*, podendo ocorrer quantas vezes tenham sido definidas pelo administrador da rede. O *script* através do número de ataques a inserir e do número de pacotes total a gerar calcula automaticamente o posicionamento temporal dos ataques, distribuindo-os de igual forma ao longo do tempo, desta forma é garantida a coexistência de pacotes e a sua sequência temporal.

Com os pontos todos preenchidos com pacotes, está obtido o *dataset* final com tráfego realista para teste de rede.

3.2.4 *Dataset* para investigador

Para além de gerar *datasets* em formato *PCAP*, esta solução permite gerá-los em formato de texto com os campos que um investigador entender. Esta funcionalidade torna-se uma mais valia no teste de *IDSs* próprios com um input de dados personalizado.

Permite ao investigador a criação de um *dataset* respeitando um protocolo próprio por ele definido, podendo utilizar os seguintes campos dos pacotes: **IP origem, IP destino, Porto**

[11] https://www.r-project.org/

[12] https://cran.r-project.org/web/packages/fitdistrplus/index.html

origem, Porto destino, Pacote/Payload e Timestamp.

Esta funcionalidade é suportada pelos dados e processos utilizados para a criação de *datasets* para administradores de rede (ver Secção 3.2.3) diferenciado apenas na análise estatística que só é considerada caso o campo *timestamp* faça parte das escolhas, caso não faça serão gerados apenas para possibilitar a distribuição dos pacotes ao longo do tempo e definir quando surge a injeção dos ataques pretendidos, conforme exemplifica a Figura 3.8.

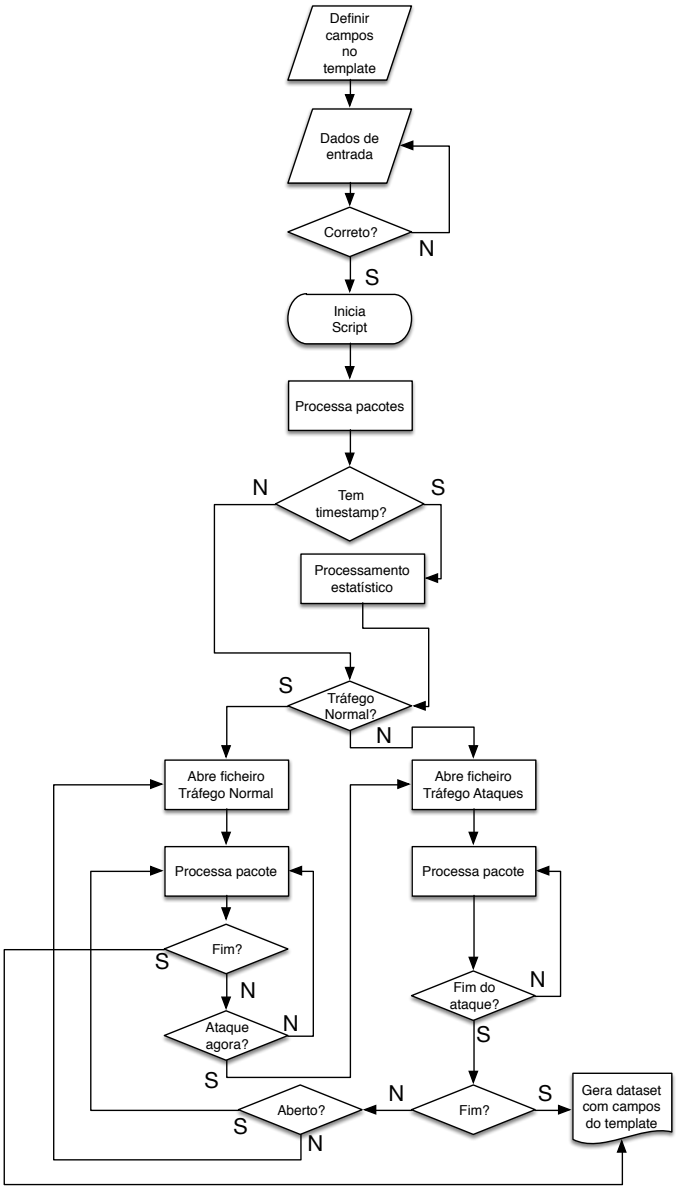


Figura 3.8: Algoritmo de ação da ferramenta na geração de *datasets* para investigadores.

Como é possível observar na Figura 3.8 o algoritmo de ação para gerar um *dataset* com

protocolo personalizado, para investigadores, é semelhante ao de geração de um para administradores de rede como é apresentado na Figura 3.5. As diferenças relevantes assentam sobre o facto da consideração, ou não, dos *timestamps* sendo este o fator decisivo para aplicação de métodos estatísticos.

A diferença mais notória assenta sobre a necessidade de existir um template externo para ser consumido pelo *script* na criação do *dataset*, surgindo pela utilização do módulo **Template Toolkit**[13] do *perl*. Este módulo permite, com base num ficheiro auxiliar, criar ficheiros seguindo um determinado formato de saída acedendo diretamente aos dados de uma tabela de *hash* para preencher os campos presentes num template como o da Figura 3.9.

```
Sip,dip,timestamp
[% FOREACH line = data -%]
[% line.src_ip %], [% line.dst_ip %], [% line.timestamps %]
[% END %]
```

Figura 3.9: Template para criação de *dataset* para investigadores.

O *script* está preparado para criar ficheiros com base nos campos anteriormente listados devendo ser colocados/retirados no template consoante a necessidade. O ficheiro de template é composto por várias partes:

- **Primeira linha** - cabeçalho do ficheiro que será gerado. Corresponde ao nome dos campos para facilitar identificação;
- **Segunda linha** - ciclo para aceder aos dados armazenados na tabela de *hash*. Não deve ser alterado;
- **Terceira linha** - acesso aos dados para escrever os diversos campos no ficheiro (*dataset*) gerado. Os campos terão de seguir o formato apresentado na Figura 3.9 *[% line.tag %]*;
- **Quarta linha** - fecha o ciclo e o template. Não deve ser alterado.

As *tags* utilizadas no template têm uma correspondência direta com os campos dos pacotes conforme apresentado na Tabela 3.1.

[13] https://metacpan.org/pod/Template

Tabela 3.1: Correspondência tag-campo de pacote.

Campo	**Tag**	**Utilização**
IP Origem	src_ip	*[% line.src_ip %]*
IP Destino	dst_ip	*[% line.dst_ip %]*
Porto Origem	src_port	*[% line.src_port %]*
Porto Destino	dst_port	*[% line.dst_port %]*
Pacote	pkt	*[% line.pkt %]*
Timestamp	timestamp	*[% line.timestamp %]*

O ficheiro criado vai respeitar os campos escritos no template e a sua ordem, conforme demonstra a Figura 3.10, desta forma torna-se flexível aos cenários pretendidos pelo investigador, desde que utilizem os campos apresentados. É possível usar todos em simultâneo, ou apenas um, o importante é que o template esteja bem construído para possibilitar o processamento efetuado pelo *script*.

```
dip,sip,timestamp
17.248.144.74,192.168.1.93,33.1269739263
17.248.144.74,192.168.1.93,136.9780052841
17.248.144.74,192.168.1.93,300.6517738752
17.248.144.74,192.168.1.93,478.3360489524
```

Figura 3.10: Exemplo para criação de *dataset* para investigadores.

3.2.5 Considerações finais

Os perfis *N* gerados permitem definir um padrão da utilização que os utilizadores dão à rede, desta forma é possível prever com alguma precisão o comportamento de outros utilizadores nas mesmas circunstâncias. Desta forma, o estudo estatístico efetuado sobre o tráfego recolhido confere ao *dataset* final o grau de realismo pretendido, na medida em que é gerado recorrendo a *timestamps* obtidos através dos perfis de utilização da rede. O facto de os *timestamps* gerados serem preenchidos por pacotes reais (obtidos pelo *web crawler* ou tráfego recolhido) ou mesmo por pacotes gerados respeitando as regras de construção, confere ainda mais realismo ao *dataset*.

Com a possibilidade de criação de *datasets* em formato personalizável a solução permite a aplicação tanto na área de administração de redes como nas área de investigação, tornando-se assim mais abrangente. Desta forma é possível a sua utilização em vertentes mais focadas em diversos aspetos do tráfego de rede, onde existe necessidade de aplicação de protocolos próprios e formatos de dados diferentes do *standard pcap*.

Capítulo 4

Testes

Neste capítulo encontram-se descritos todos os testes realizados à aplicação desenvolvida. Estão descritos os pressupostos para cada teste, resultados esperados e efetivos, bem como alterações resultantes das conclusões retiradas em cada teste.

4.1 Rede de testes

Para facilitar o desenvolvimento da ferramenta, e tendo em conta que seria necessário efetuar a recolha de tráfego de rede normal e de tráfego de ataques para testar as funcionalidades de análise de tráfego e criação de *datasets*, foi necessário criar um ambiente de rede, simulado, conforme demonstra a Figura 4.1

O cenário foi simulado em ambiente virtual utilizando a ferramenta *GNS3*[1] e recorrendo à instalação de máquinas virtuais em *VirtualBox*[2].

O *GNS3* é uma ferramenta *freeware* que permite a virtualização de redes de pequena e média dimensão para os mais diversos cenários de aplicação. Toda a rede é construída e configurada em equipamentos virtuais, dispensando o uso de *hardware*, possibilitando também a sua integração com rede/equipamentos físicos.

O *VirtualBox* é uma ferramenta de virtualização de sistemas operativos, permitindo o uso de vários sistemas operativos na mesma máquina física. Permite virtualização de *Microsoft Windows*, *Linux*, *Mac OS*, *BSD*, entre outros.

O cenário de rede foi construído utilizando um *router cisco c3600* responsável por tarefas de encaminhamento de tráfego e para possibilitar a comunicação entre as duas redes confi-

[1] `https://www.gns3.com/`
[2] `https://www.virtualbox.org`

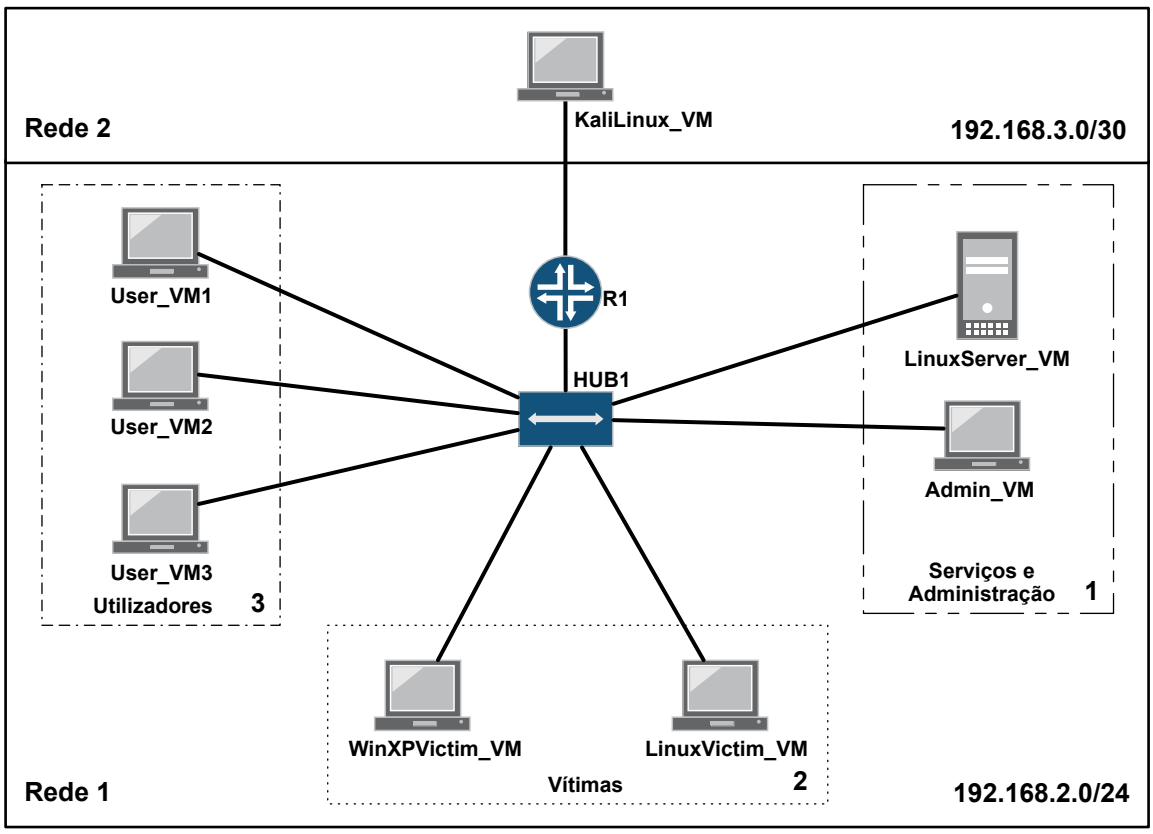


Figura 4.1: Rede para realização de testes.

guradas, garantindo também uma futura (se necessária) conexão da rede virtual a uma rede física. Foi também utilizado um *hub ethernet* genérico que interliga todas as máquinas virtuais (VM) da rede 1. Estes equipamentos foram configurados de acordo com as informações da Tabela 4.1.

Tabela 4.1: Informações de configuração do cenário de rede

Rede	Endereço de Rede	Máscara de Rede	Endereço de Broadcast	Default Gateway
1	192.168.2.0/24	255.255.255.0	192.168.2.255	192.168.2.254
2	192.168.3.0/30	255.255.255.252	192.168.3.3	192.168.3.2

A rede 1 aquando da sua configuração foi dividida em três áreas (1, 2 e 3 conforme Figura 4.1), cada uma composta por VM distintas com funções e requisitos próprios, de acordo com a Tabela 4.2. Esta rede funciona como rede principal, sendo o alvo dos ataques aquando da realização de testes. É também nesta rede que se efetua a captura de tráfego quer ele seja normal ou malicioso, bem como o processamento dos dados para criação dos *dataset*.

Tabela 4.2: Requisitos de configuração das VM da rede 1

VM	SO	Memória RAM	CPUs
Admin_VM	Ubuntu Desktop 14.04.3 LTS	2048MB	2
LinuxServer_VM	Ubuntu Server 14.04.3 LTS	512MB	1
LinuxVictim_VM	Ubuntu 8.04	1024MB	1
WinXPVictim_VM	Windows XP sem Service Pack	512MB	1
User_VM1	Ubuntu Server 14.04.3 LTS	512MB	1
User_VM2	Ubuntu Server 14.04.3 LTS	512MB	1
User_VM3	Ubuntu Server 14.04.3 LTS	512MB	1

A rede 2, conforme ilustra a Figura 4.1, apresenta apenas uma máquina virtual responsável por desempenhar o papel de atacante. Conforme demonstra a Tabela 4.3, trata-se de uma VM de *Kali Linux*[3] criada com o objetivo de testar e detetar vulnerabilidades em máquinas e redes, sendo por várias aplicações que para essa finalidade bem como para a injeção de ataques, como por exemplo:

- *Metasploit-framework*[4] - utilizado para lançar ataques sobre serviços e máquinas;
- *Wireshark* - para além de funções de recolha de tráfego conforme explicado na Secção 3.2.1, pode ser utilizado para *sniffing* de pacotes;
- *HTTrack*[5] - utilizado para analisar aplicações *web*, esta aplicação permite fazer *download* de uma página e todos os diretórios e ficheiros associados;

Esta máquina aparece fora da rede principal (rede 1) com o objetivo de possibilitar a injeção de ataques através de uma máquina externa, aproximando os dados capturados a um cenário real. Os ataques serão lançados sobre as VM da área 2 da rede 1.

Tabela 4.3: Requisitos de configuração da VM da rede 2

VM	SO	Memória RAM	CPUs
KaliLinux_VM	Kali GNU/Linux Rolling 64-bit	2048MB	2

Para efeitos de diversidade de testes, os ataques foram também lançados do interior da rede 1 através da máquina **Admin_VM**.

[3] `https://www.kali.org`
[4] `http://www.metasploit.com`
[5] `https://www.httrack.com`

4.1.1 Área 1 - Serviços e Administração

A área 1 englobou duas VM *Linux*, conforme demonstra a Figura 4.2, uma com funções de administração (**Admin_VM**) a outra com funções de servidor para disponibilização de serviços (**LinuxServer_VM**).

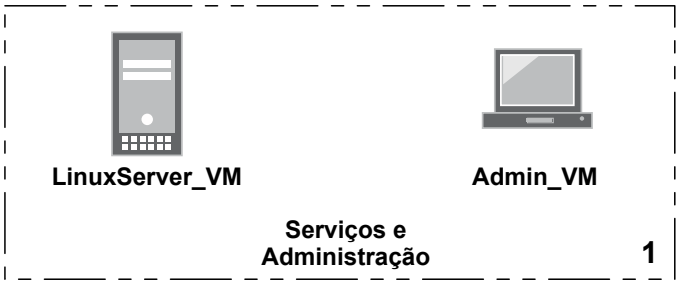


Figura 4.2: Área 1 - Serviços e Administração

A **Admin_VM** ficou responsável pela monitorização da rede, controlando os fluxos de tráfego e era, também, nesta máquina onde se encontravam configurados os mecanismos de segurança, mais propriamente os IDS. A recolha de tráfego para construção dos *dataset*, foi também realizada nesta VM. Para além das funções de administração da rede e recolha de tráfego, esta máquina permitiu também executar funções de atacante em alternativa à **KaliLinux_VM** da rede 2, lançando ataques às VM da área 2, utilizando apenas a *Metasploit-framework*.

A **LinuxServer_VM** ficou responsável por ser a máquina servidor de serviços tipo de uma rede empresarial real, nomeadamente servidor *web* e servidor de *mail*. Esta máquina surgiu com o intuito de permitir gerar tráfego normal variado para que a recolha de tráfego de rede normal apresentasse tráfego de vários tipos.

4.1.2 Área 2 - Vítimas

A área 2 englobou duas VM, uma *Windows* e outra *Linux*, conforme demonstra a Figura 4.3, ambas com o objetivo de serem o alvo de ataques por parte da VM **Admin_VM**, aquando da captura de tráfego de ataques.

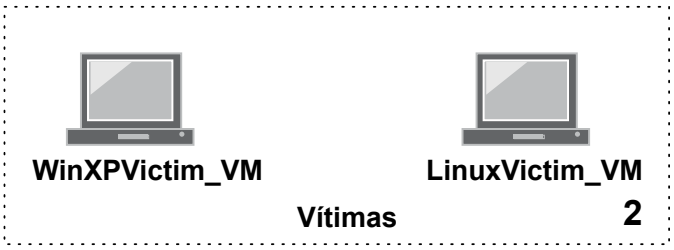


Figura 4.3: Área 2 - Vítimas

A **WinXPVictim_VM** é uma VM *Windows XP* sem *Service Pack*, quer com isto dizer que apresentava inúmeras vulnerabilidades de segurança, apresentando-se como um alvo fácil para atacar.

A **LinuxVictim_VM** é uma VM *Linux* vulnerável, esta máquina é disponibilizada pelos criadores da *Metasploit-framework*, e encontra-se disponível na *web* com o nome *Metasploit2*[6]. Esta máquina, tal como a anterior, apresenta-se como um alvo fácil para atacar, e pelo facto de terem sido utilizados dois SO, possibilitou a recolha de ataques diversificados.

4.1.3 Área 3 - Utilizadores

A área 3 englobou três VM *Linux*, conforme demonstra a Figura 4.4, todas elas com o simples objetivo de gerar tráfego.

Figura 4.4: Área 3 - Utilizadores

Estas máquinas foram pensadas com o principal objetivo de gerar tráfego o mais diversificado possível, através de pedidos realizados aos serviços (email, web) da **LinuxServer_VM**. Este tráfego permite que a recolha de tráfego normal apresentasse tráfego diversificado, originários dos vários serviços configurados no servidor.

Nos primeiros testes foi utilizado um *dataset* de dados de navegação web manual, de um utilizador real.

[6] `https://information.rapid7.com/metasploitable-download.html`

4.2 Testes executados

4.2.1 Teste 1

O primeiro teste efetuado foi realizado apenas com um ataque, o ataque escolhido explora uma vulnerabilidade no ficheiro *NetAPI32.dll* do Windows que permite ignorar o *NX* em alguns sistemas operativos e aceder remotamente aos mesmos [34]. Neste teste a máquina vítima utilizada foi a Windows XP sem Service Pack não contendo correções para esta falha permitindo assim o sucesso do ataque.

O ataque foi executado utilizando a *Metasploit Framework* e os pacotes do ataque foram capturados conforme descrito anteriormente na Secção 3.2.1. Para lançar o ataque primeiro temos de o selecionar no repositório da *framework* e de seguida definir os parâmetros necessários para este ser bem sucedido, neste caso é apenas necessário definir o endereço *IP* da máquina vítima, conforme demonstra a Figura 4.5.

```
msf exploit(ms08_067_netapi) > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > show options

Module options (exploit/windows/smb/ms08_067_netapi):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   RHOST                     yes       The target address
   RPORT    445              yes       Set the SMB service port
   SMBPIPE  BROWSER          yes       The pipe name to use (BROWSER, SRVSVC)
```

Figura 4.5: Opções do ataque.

Após lançado o ataque, foi possível verificar que se obteve acesso remoto à vítima permitindo a execução de comandos na shell do Windows. Uma vez capturados os pacotes de ataque, foi executado o *script* com um *dataset* de tráfego normal (de navegação na *web*) com cerca de 230000 pacotes e o de ataque capturado. Para este teste foi definido um fluxo de dois pacotes por segundo durante uma hora, totalizando 7200 pacotes para o *dataset* final e foram injetados 2 ataques igualmente espaçados no tempo.

A criação do *dataset* final foi concluída com sucesso procedendo-se à ligação do *Snort* para efetuar a deteção da injeção direta dos pacotes na rede. O *Snort* utilizado encontra-se na versão 2.9.8.0 e com regras de Janeiro de 2016, tendo ainda os pré-processadores pré-definidos ativos (ver Anexo A).

Com esta configuração espera-se que o *Snort* detete sem problemas 70 Protocol Data

Unit (PDU)s (35 de cada iteração), específicos de sessão DCE/RPC correspondentes ao ataque e deixe passar o restante tráfego. Após o termino da injeção dos pacotes na rede, verificou-se a análise efetuada pelo *Snort* onde foi possível observar um número demasiado elevado de pacotes rejeitados. A rejeição dos pacotes deveu-se a uma má reutilização dos *timestamps* existentes em conjunto com os gerados, levando a que os pacotes estivessem associados a uma data muito anterior à atual sendo consequentemente rejeitados pelo *Snort*. Esta má construção do *dataset* levava também a que os pacotes fossem colocados de forma desordenada, não respeitando os *timestamps* fornecidos.

4.2.2 Teste 2

No segundo teste foi utilizado o mesmo ataque que no Teste 1, a máquina vítima também foi a mesma tal como as versões do *Snort* e respectivas regras deteção. Para este teste o *script* foi alterado para que os *timestamps* passem a considerar o tempo atual e o gerado pelos métodos estatísticos para assim evitar os conflitos ocorridos no primeiro teste.

Espera-se que com as alterações efetuadas, o *Snort* detete os 70 PDUs correspondentes ao ataque e permita a passagem do restante tráfego. Após o término da injeção dos pacotes na rede, foi possível verificar que o número de pacotes rejeitados é substancialmente inferior que obtido no Teste 1 confirmando que a alteração efetuada à utilização dos *timestamps* surtiu efeito. Contudo apenas a primeira injeção de ataque foi detetada (35 PDUs), após alguma análise e *troubleshooting* foi possível observar no *Wireshark* que os pacotes da segunda injeção foram marcados como retransmissão.

Os pacotes estão a ser considerados retransmissão devido aos da segunda injeção serem exatamente iguais aos da primeira exceptuando-se o *timestamp* o que leva a que o *Snort* não os marque como ataque.

A solução para resolver o problema prende-se com a alteração das propriedades *sequence number* e *packet ID* dos pacotes da segunda injeção para que assim exista a diferença necessária para evitar que sejam considerados retransmissão.

4.2.3 Teste 3

O cenário e condições utilizados nos dois primeiros testes manteve-se no terceiro exceptuando-se apenas as alterações ao *script* para que seja possível criar um *dataset* final com repetições do mesmo ataque. Com as alterações efetuadas às propriedades *sequence number* e *packet ID* dos pacotes será possível ao *Snort* detetar devidamente os 70 PDUs específicos

do ataque injetado no *dataset*.

O teste foi então executado e após a injeção do *dataset* na rede, foi possível verificar que o número de deteções não sofreu qualquer alteração em comparação com o Teste 2. Após análise dos pacotes recorrendo ao *Wireshark* foi possível verificar que os pacotes que sofreram alteração encontravam-se mal construídos, com falta de dados.

A API utilizada (*NetPacket*) implementa funções que permitem descodificar e recodificar os pacotes a nível protocolar, como por exemplo *TCP, UDP* ou *Ethernet*, sendo que existem módulos adaptados a cada protocolo e quando os pacotes são alterados o seu *checksum* é automaticamente recalculado pela API. Para este cenário é necessário utilizar o módulo *NetPacket::Ethernet* para possibilitar o acesso e edição das propriedades anteriormente referidas. Contudo, após análise do manual e código fonte do módulo, foi possível verificar que a função de recodificação dos pacotes alterados não se encontra implementada, inviabilizando assim a solução para o problema.

Foram exploradas diversas API's alternativas, entre elas a *Net::Packet*[7] foi a que se apresentou mais promissora, mostrando-se também bastante semelhante à anteriormente utilizada, mas também falhou. Com os testes de implementação efetuado percebeu-se que esta API não possibilita a utilização dos nossos dados pré-carregados para serem editar e recodificados.

Desta forma, não nos é possível continuar com os testes para comprovar o funcionamento e aplicabilidade do *script* para teste de mecanismos de deteção de intrusões utilizando a linguagem *perl*.

4.2.4 Teste 4

Chegamos ao teste 4 com uns resultados diferentes do que era pretendido. O estudo estatístico e construção do *dataset* final correram conforme o planeado contudo a injeção do mesmo ataque mais do que uma vez resultou em erros de duplicação de pacotes. O objetivo deste teste centrou-se na alteração de abordagem ao tema de forma a ser possível provar que podemos gerar *datasets* constituídos por tráfego normal e ataques para testar mecanismos de segurança de uma rede e atingir o objetivo da dissertação.

Não sendo possível alterar os pacotes de um ataque e voltar a codificá-lo com dados novos, a abordagem neste teste utilizou 2 ataques diferentes. Todo o cenário de testes é igual ao já utilizado nos anteriores sendo que para além do ataque *NetAPI32.dll* utilizou-se também um ataque de exploração de vulnerabilidade a um servidor *Tomcat* de forma a obter acesso ao

[7] https://metacpan.org/release/Net-Packet

mesmo.

Desta forma os resultados esperados são a deteção dos PDUs do *NetAPI32.dll*, específicos da sessão DCE/RPC criada pelo ataque, e também a deteção do ataque ao servidor *Tomcat*. Após a injeção do *dataset* na rede verificou-se que o *Snort* detetou com sucesso os 35 PDUs de DCE/RPC e despoletou 59 alertas provenientes do ataque ao servidor *Tomcat*. Estes alertas provém de uma regra criada manualmente para detetar acessos não autorizados ao porto específico do servidor (ver Anexo B).

Podemos concluir que o objetivo foi atingindo, embora não permita a injeção do mesmo ataque várias vezes, o teste pode se executado com sucesso se os ataques utilizados forem diferentes evitando assim erros de duplicação de pacotes.

4.3 Conclusão

Durante o teste do *script*, vários foram os problemas que surgiram, conforme seria esperado. Primeiro ocorreram incoerências nos *timestamps*, pois devido ao cálculo dos novos recorrerem à reutilização dos que tinham sido previamente capturados juntando-os com gerados pela distribuição obtida, os resultantes ocorriam em datas demasiado anteriores à atualidade. Por esta razão mais de metade dos pacotes do *dataset* eram rejeitados pelo *Snort* quer fossem legítimos ou não. Após a alteração do cálculo para utilizar o *timestamp* atual ao invés do capturado, o problema ficou resolvido e os pacotes passaram a ser corretamente identificados pelo *Snort*.

No seguimento dos testes ocorreu um problema de retransmissão de pacotes, problema que ocorre devido a existirem pacotes integralmente iguais a ocorrerem em tempos diferentes. A solução seria alcançável se existisse uma API compatível que permiti-se a alteração das propriedades específicas do pacote para evitar retransmissão. O mesmo não foi possível, já que a API utilizada não permite recodificar pacotes ao nível do *Ethernet* e as alternativas também não permitiram responder à necessidade.

Desta forma, por limitação técnica, ficámos impossibilitados de provar empiricamente que a solução desenvolvida possibilita a injeção de pacotes do mesmo ataque, diversas vezes, num *dataset* composto também por pacotes legítimos, com o objetivo de testar mecanismos de segurança de uma rede. Contudo a solução apresentada permite atingir os objetivos quando os ataques utilizados são todos diferentes, o *dataset* é corretamente construído e quando injetado na rede os pacotes são todos identificados, e devidamente marcados, conforme seria esperado.

Capítulo 5

Conclusões e trabalho futuro

Nesta dissertação foi abordada a temática da segurança nas redes de dados, mais concretamente, foi estudada a possibilidade de criação de mecanismos que permitam testar a segurança e robustez destas bem como dos mecanismos/aplicações de que esta dispõe para a sua proteção. Para tal foi efetuado um estudo relativo às anomalias que tipicamente ocorrem nas redes, mais precisamente nas empresariais, bem como do tipo de mecanismos de segurança utilizado. O estudo dos mecanismos focou-se essencialmente nos IDSs *open-source* disponíveis, existindo vários e com vários métodos de deteção tendo a nossa escolha recaído sobre os que baseiam as deteções nas assinaturas dos ataques. Em suma, o foco do trabalho foi na geração de *datasets* de rede, compostos por pacotes de tráfego normal e de ataques, que permitam fazer *benchmark* às soluções IDS implementadas.

Neste trabalho foi desenvolvido um protótipo com o objetivo de provar que é possível implementar uma solução de *benchmark* de IDS recorrendo a tráfego de rede normal previamente capturado, em formato *pcap*, e a tráfego de ataques. Posteriormente, os pacotes capturados, sofrem um tratamento estatístico com o intuíto de os enquadrar com uma determinada distribuição estatística, permitindo assim prever o comportamento dos pacotes da rede e gerar um novo *dataset* constituído por tráfego normal e de ataques. Encontrada a distribuição, é gerado um conjunto de pontos representativos dos *timestamps* a atribuir aos pacotes gerados para o novo *dataset*. Após gerado, o novo *dataset*, é injetado na rede de testes onde existe um IDS (*Snort*) à escuta para detetar as anomalias/ataques ocorridos. Um dos objetivos deste trabalho prendia-se com a possibilidade de ocorrência do mesmo ataques diversas vezes ao longe do tempo, no mesmo *dataset*, tal cenário revelou-se impossivel de implementar devido à abordagem e linguagem de desenvolvimento escolhidas. Com a nossa abordagem, apenas a primeira ocorrência anómala é considerada, todas a outras são iden-

tificadas como retransmissões/duplicações sendo automaticamente descartadas pelo *IDS*, a forma de evitar este cenário passaria por editar campos específicos dos pacotes e voltar a reconstruí-los com os novos dados, tornando-os únicos a cada ocorrência, tal não é possível por limitações do módulos disponíveis na linguagem de desenvolvimento escolhida (*perl*). De forma a minimizar o impacto desta situação, optou-se pela utilização de 2 ataques diferentes para efeito de teste, o que se revelou numa abordagem com resultados positivos permitindo-nos provar ser possível a criação do novo *dataset* para testes de segurança e robustez de uma rede.

Para além dos testes recorrendo a dados em formato standard, *pcap*, surgiu a possibilidade de munir a o protótipo desenvolvido de capacidades de criação de *datasets* em formato personalizado em texto simples, permitindo a sua aplicação em cenários de investigação científica. Desta forma é possível escolher quais os dados que constituem os *dataset* e qual a sua disposição/ordem, tudo isto apresentado em texto simples. Com esta abordagem é possibilitado, ao investigador, a criação dos dados seguindo um protocolo próprio e adaptar assim os dados a vários cenários de investigação diferentes.

Podemos concluir que o objetivo principal desta trabalho foi atingindo, ainda que com alguns ajustes foi construído um protótipo que permite provar que é possível criar *datasets* para teste de mecanismos de segurança de uma rede. A solução apresentada é totalmente *open-source*, recorre a métodos estatísticos com o objetivo de tornar os dados resultantes os mais próximos possíveis de dados reais e disponibiliza um módulo quer permite personalizar os dados resultantes.

Embora se tenham atingindo os objetivos, o trabalho não se encontra concluído, existindo margem para trabalho futuro. Essencialmente existe a possibilidade de adaptar o protótipo na íntegra para outra linguagem de desenvolvimento com o objetivo principal de possibilitar a edição/reconstrução dos pacotes de rede, algo que não seria possível fazer no âmbito desta dissertação tendo em conta o tempo disponível. Para além da adaptação do protótipo, também será necessário a reformulação do atual tornando-o mais eficiente na realização das tarefas disponíveis.

Para além das questões relacionadas com o protótipo, seria interessante realizar um estudo comparativo dos diversos IDSs disponíveis (mantendo sempre o pressuposto *open-source*) utilizando *datasets* gerados pela solução desenvolvida, na tentativa de perceber quais os pontos fortes e fracos de cada um.

Bibliografia

[1] N. Hoque, Monowar H. Bhuyan, R.C. Baishya, D.K. Bhattacharyya, and J.K. Kalita. Network attacks: Taxonomy, tools and systems. *Journal of Network and Computer Applications*, 40:307–324, 2014.

[2] David Schneider. The state of network security. *Network Security*, 2012(2):14–20, 2012.

[3] Monowar H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys & Tutorials*, 16(1):303–336, 2014.

[4] Paul Barford and David Plonka. Characteristics of network traffic flow anomalies. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, IMW '01, pages 69–73, New York, NY, USA, 2001. ACM.

[5] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection. *CSUR*, 41(3):7–9, 2009.

[6] MyCERT. *Reported Incidents based on General Incident Classification Statistics 2013*. 2015.

[7] MyCERT. *Reported Incidents based on General Incident Classification Statistics 2014*. 2015.

[8] MyCERT. *Reported Incidents based on General Incident Classification Statistics 2015*. 2018.

[9] MyCERT. *Reported Incidents based on General Incident Classification Statistics 2016*. 2018.

[10] MyCERT. *Reported Incidents based on General Incident Classification Statistics 2017*. 2018.

[11] Cisco. What is the difference: Viruses, worms, trojans, and bots?, 2015.

[12] Neil Anderson and Jim Doherty. Tip 5: Lock out spyware and adware > what is spyware and adware?, 2016.

[13] Cisco. *Cisco 2014 Annual Security Report*. Cisco Annual Security Reports. 2014.

[14] SANS Institute. *Understanding IPS and IDS: Using IPS and IDS together for Defense in Depth*. 2015.

[15] Frederic Massicotte, Francois Gagnon, Yvan Labiche, Lionel Briand, and Mathieu Couture. Automatic evaluation of intrusion detection systems. pages 361–370, Dec 2006.

[16] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[17] Luis Martin Garcia. Programming with libpcap - sniffing the network from our own application. *Hakin9*, 3(2):39, 2008.

[18] Wiki.wireshark.org. Development/libpcapfileformat - the wireshark wiki, 2016.

[19] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[20] Jay Beale. *Snort 2.1 Intrusion Detection, Second Edition*. Syngress, 2004.

[21] Martin Roesch and Chris Green. *SNORT Users Manual 2.9.7.3*. The Snort Project.

[22] Redmine.openinfosecfoundation.org. Suricata - suricata - open information security foundation, 2015.

[23] NSM engine Open Source IDS, IPS. Features, 2012.

[24] Bro.org. Introduction — bro 2.4.1 documentation, 2015.

[25] Bro.org. Bro cluster architecture — bro 2.4.1 documentation, 2015.

[26] Bro.org. Writing bro scripts — bro 2.4.1 documentation, 2015.

[27] Richard Zuech, Taghi Khoshgoftaar, Naeem Seliya, Maryam Najafabadi, and Clifford Kemp. A new intrusion detection benchmarking system, 2015.

[28] https://www.ll.mit.edu/ideval/. Mit lincoln laboratory: Cyber systems & technology: Darpa intrusion detection, 2015.

[29] Kdd.ics.uci.edu. Data mining and knowledge discovery competition, kdd cup 1999 data, 2015.

[30] Benjamin Sangster, T. J. O'Connor, Thomas Cook, Robert Fanelli, Erik Dean, William J. Adams, Chris Morrell, and Gregory Conti. Toward instrumenting network warfare competitions to generate labeled datasets. In *Proceedings of the 2Nd Conference on Cyber Security Experimentation and Test*, CSET'09, pages 9–9, Berkeley, CA, USA, 2009. USENIX Association.

[31] Unb.ca http://www.unb.ca/research/iscx/dataset/. Unb iscx dataset, 2015.

[32] Sans.org. Sans: Intrusion detection faq: What is a honeypot?, 2015.

[33] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers and Security*, 31(3):357–374, 2012.

[34] https://www.rapid7.com/db/modules/exploit/windows/smb, ms08-067 microsoft server service relative path stack corruption, 2017.

Apêndice A

Preprocessadores do *Snort*

- Frag3 - módulo de fragmentação IP, criado com o objetivo de tornar a análise de tráfego rápida para gerir dados menos complexos e possuindo técnicas anti-evasão;
- Session - disponibiliza API de criação e gestão de sessões de fluxo, evitando que fluxos de transferência de grande quantidade de dados sejam ignorados após algum tempo. Controla informações importantes do fluxo (direção, protocolo, etc);
- Stream - módulo de remontagem de TCP, utilizado para monitorar sessões TCP e UDP;
- sfPortscan - criado pela *Sourcefire*[1], é utilizado para detetar a fase de reconhecimento (1ª fase) de uma ataque;
- RPC Decode - utilizado para normalizar vários registos Remote Procedure Call (RPC) fragmentados, juntando-os em apenas um registo não fragmentado;
- Performance Monitor - monitoriza o desempenho do *Snort* em tempo real e comparando-o com um máximo teórico. As estatísticas apresentadas contemplam *timestamp*, número de alertas, número de registos, entre outros;
- HTTP Inspect - descodifica dados HTTP de aplicações de utilizador para os normalizar. Compatível com pedidos de cliente e respostas de servidor;
- SMTP Preprocessor - descodifica dados SMTP de aplicações de utilizador. Descodifica os dados para analisar comandos e respostas SMTP;

[1]

- POP Preprocessor - descodifica dados POP3 de aplicações de utilizador. Descodifica os dados para analisar comandos e respostas POP3;
- IMAP Preprocessor - descodifica dados IMAP de aplicações de utilizador. Descodifica os dados para analisar comandos e respostas IMAP;
- FTP/*Telnet* Preprocessor - oferece melhorias ao descodificador de *Telnet* e inspeciona fluxos de dados FTP/*Telnet*. Descodifica os fluxos para identificação de comandos e respostas *FTP*, identifica também sequências de escape do *Telnet* e normaliza os dados;
- SSH - deteta a ocorrência dos ataques: *Challenge-Response Buffer Overflow*. *CRC 32*, *Secure CRT* e *Protocol Mismatch*;
- DNS - descodifica respostas DNS e deteta os ataques: *DNS Client RData Overflow*, *Obsolete Record Types* e *Experimental Record Types*;
- SSL/TLS - o *Snort* deve ignorar tráfego cifrado por razões de performance e para evitar a ocorrência de falsos positivos. Este pré-processador descodifica tráfego SSL/TLS com o objetivo de determinar quando será aconselhável parar a inspecção dos dados;
- ARP Spoof Preprocessor - descodifica pacotes *ARP* e deteta ataques *ARP*, pedidos *unicast* e mapeamentos *Ethernet-IP* inconsistentes;
- DCE/RPC 2 Preprocessor - efetua a desfragmentação de pacotes SMB e DCE/RPC para evitar técnicas de evasão às regras;
- Sensitive Data Preprocessor - módulo que deteta e filtra informações pessoais, como números de cartão de crédito, endereço e-mail, etc;
- Normalizer - normaliza pacotes para evitar evasão;
- SIP Preprocessor - fornece meios para atacar Common Vulnerabilities and Exposures (CVE)s identificados ao longo do tempo, relacionados com SIP, tornando também mais fácil a deteção de ataques novos;
- Reputation Preprocessor - fornece uma *blacklist/whitelist* IP para analisar os pacotes decidindo se estes são bloqueados, perdidos ou aceites;
- GTP Decoder and Preprocessor - descodifica os pacotes GTP para detetar tentativas de intrusões através desse protocolo;

- *Modbus* Preprocessor - descodifica os pacotes do protocolo *Modbus* e fornece regras de acesso aos seus campos específicos;
- DNP3 Preprocessor - descodifica os pacotes do protocolo DNP3 e fornece regras de acesso aos seus campos específicos;
- AppId Preprocessor - fornece capacidades ao *Snort* para gerir a rede ao nível aplicacional. Atribui identificadores às aplicações da rede para facilitar a criação de regras e fornece estatísticas de uso de largura de banda para cada aplicação.

Apêndice B

Regra de deteção de acesso não autorizado ao servidor *Tomcat*

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 8180 (msg:"Acesso não autorizado ao
Apache Tomcat"; classtype:successful-user; sid:15104;)
```