

Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais
Faculdade de Computação

Glenon Mateus Barbosa Araújo

**Avaliação de Sistemas de Detecção de Intrusão em uma
Rede Acadêmica**

Belém - PA

2018

Glenon Mateus Barbosa Araújo

Avaliação de Sistemas de Detecção de Intrusão em uma Rede Acadêmica

Trabalho de conclusão de curso apresentada ao Curso de Ciência da Computação da Faculdade de Computação do Instituto de Ciências Exatas e Naturais da Universidade Federal do Pará como pré-requisito para a obtenção do título de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Roberto Samarone dos Santos Araújo

Belém - PA

2018

Glenon Mateus Barbosa Araújo

Avaliação de Sistemas de Detecção de Intrusão em uma Rede Acadêmica/ Glenon Mateus Barbosa Araújo. – Belém - PA, 2018

Orientador: Prof. Dr. Roberto Samarone dos Santos Araújo

Trabalho de Conclusão de Curso – Universidade Federal do Pará - UFPA
Instituto de Ciências Exatas e Naturais
Faculdade de Computação, 2018.

1. Suricata. 2. Snort. 3. IDPS. I. Orientador. II. Universidade Federal do Pará. III. Faculdade de Computação. IV. Análise de IDPSs

Glenon Mateus Barbosa Araújo

Avaliação de Sistemas de Detecção de Intrusão em uma Rede Acadêmica

Trabalho de conclusão de curso apresentada ao Curso de Ciência da Computação da Faculdade de Computação do Instituto de Ciências Exatas e Naturais da Universidade Federal do Pará como pré-requisito para a obtenção do título de Bacharel em Ciência da Computação

Trabalho aprovado. Belém - PA, 12 de julho de 2018:

Prof. Dr. Roberto Samarone dos Santos Araújo
Orientador

Prof. Dr. Josivaldo de Souza Araujo
Avaliador

Prof. Dr. Raimundo Viégas Junior
Avaliador

Belém - PA
2018

Resumo

A todo momento novos ataques ou mesmo variações de ataques existentes surgem e são lançados às redes. Somente a utilização de *firewall* é insuficiente, pois o atacante pode utilizar portas, na qual rodam serviços fornecidos pela empresa/instituição, para realizar ataques. Diante desse cenário, é indispensável, para um administrador de rede, o uso de ferramentas auxiliares como um Sistema de Detecção e Prevenção de Intrusão (IDPS), que alertam ou até mesmo bloqueiam ataques. Esse trabalho tem o objetivo de avaliar os Sistemas de Detecção de Intrusão *open source* mais populares, Snort e Suricata. O Snort é um sistema de detecção, lançado em 1998 por Martin Roesch e foi uma das primeiras em seu segmento a realizar análise de tráfego em tempo real e registro dos pacotes de forma leve, utilizando recursos mínimos de processamento. Já o Suricata, foi lançado em 2010, e tem como principal característica, a utilização da tecnologia *multithreading*, tirando maior proveito dos processadores, visando melhorar o desempenho. A avaliação foi realizada comparando o desempenho e as detecções de intrusões as quais foram colocadas em funcionamento em uma rede de produção real, verificando suas vantagens e desvantagens. Nos testes realizados, foram utilizadas ferramentas auxiliares para simular ataques a uma máquina alvo instalada para essa finalidade.

Palavras-chave: Segurança da Informação, Suricata, Snort, Sistema de Detecção de Intrusão, Sistema de Prevenção de Intrusão.

Abstract

At any moment new attacks or even variations of existing attacks arise and are launched only on networks. Only the use of firewall is insufficient, because the attacker can use ports, in which services provided by the company / institution run, to execute attacks. Given this scenario, it is essential for a network administrator to use tools such as a Intrusion Detection and Prevention System that alert or even block attacks. This work aims to evaluate the most popular open source Intrusion Detection Systems, Snort and Suricata. Snort is a detection system launched in 1998 by Martin Roesch and was one of the first in its segment to perform real-time traffic analysis and package logging in a lightweight manner using minimal processing capabilities. Suricata was launched in 2010 and its main feature is the use of multithreading technology, taking advantage of processors to improve performance. The evaluation was performed comparing the performance and intrusion detections that were put into operation in an actual production network, verifying their advantages and disadvantages. In the tests performed, auxiliary tools were used to simulate attacks on a target machine installed for this purpose.

Keywords: Security Information, Suricata, Snort, Intrusion Detection System, Intrusion Prevention System, IDS, IPS.

Lista de ilustrações

Figura 1 – Estatísticas de ataques reportadas ao CERT.br	17
Figura 2 – Estatísticas de incidentes reportados ao CAIS	18
Figura 3 – Topologia geral de uma rede de computadores	28
Figura 4 – Estatísticas de Desfiguração de Páginas	32
Figura 5 – Ataque de Negação de Serviço Distribuído	34
Figura 6 – Exemplo de saída do Nmap	36
Figura 7 – Arquitetura do Metasploit	37
Figura 8 – Arquitetura <i>Remote Mode</i> do <i>framework</i> Pytbull	39
Figura 9 – Arquitetura <i>IPS mode</i> do <i>framework</i> Pytbull	39
Figura 10 – Arquitetura <i>IPS mode with attacked server in dmz</i> do <i>framework</i> Pytbull	39
Figura 11 – Arquitetura <i>IDS mode with attacked server in dmz</i> do <i>framework</i> Pytbull	39
Figura 12 – Arquitetura OpenVAS	41
Figura 13 – Exemplo de arquitetura de NIDS passivo	45
Figura 14 – Exemplo de Arquitetura de NIDS ativo	45
Figura 15 – Arquitetura do Snort	47
Figura 16 – Uso de <i>plugins</i> no pré-processador	48
Figura 17 – Motor de Detecção do Snort	49
Figura 18 – Arquitetura <i>Multithread</i> do Suricata	51
Figura 19 – Tráfego da Rede de Teste	54
Figura 20 – Infraestrutura do ambiente de teste	55
Figura 21 – Definição do alvo no OpenVAS	58
Figura 22 – Configuração de um <i>task</i> de <i>scan</i> no OpenVAS	59
Figura 23 – Resultados do teste executando o NMAP no modo <i>fast</i>	61
Figura 24 – Resultados do teste executando o NMAP no modo de detecção do SO e versões dos serviços	62
Figura 25 – Resultados do teste executando um o <i>scan</i> de vulnerabilidades	62
Figura 26 – Resultado da execução do <i>framework</i> Pytbull sobre o Suricata	63
Figura 27 – Resultado da execução do <i>framework</i> Pytbull sobre o Snort	63
Figura 28 – Resultados do teste executando um o <i>scan</i> de vulnerabilidades	64

Lista de tabelas

Tabela 1	–	Classificação dos ataques passivos e ativos	25
Tabela 2	–	Tabela de regras aplicadas no <i>firewall</i>	27
Tabela 3	–	Resultado do uso de recurso de <i>hardware</i> do Suricata	64
Tabela 4	–	Resultado do uso de recurso de <i>hardware</i> do Snort	64

Lista de abreviaturas e siglas

IDS	<i>Intrusion Detection System</i>
IPS	<i>Intrusion Prevention System</i>
SDI	Sistema de Detecção de Intrusão
SPI	Sistema de Prevenção de Intrusão
IDPS	<i>Intrusion Detection and Prevention System</i>
HIDS	<i>Host Based Intrusion Detection Systems</i>
NIDS	<i>Network Based Intrusion Detection Systems</i>
MB	<i>Megabytes</i>
GB	<i>Gigabytes</i>
SO	Sistema Operacional
JSON	<i>JavaScript Object Notation</i>
CAIS	Centro de Atendimento a Incidentes de Segurança
DoS	<i>Denial of Services</i>
DDoS	<i>Distributed Denial of Services</i>
URL	<i>Uniform Resource Locator</i>
LAN	<i>Local Area Network</i>
WAN	<i>Wide Area Network</i>
DMZ	<i>Demilitarized Zone</i>
SPF	<i>Stateful Packet Filter</i>
SQL	<i>Structured Query Language</i>
XSS	<i>Cross-site scripting</i>
OISF	<i>Open Information Security Foundation</i>
ET	<i>Emerging Threats</i>

NVT	<i>Network Vulnerability Tests</i>
OMP	<i>OpenVAS Management Protocol</i>
OAP	<i>OpenVAS Assistant Protocol</i>
GSD	<i>Greenbone Security Desktop</i>
GSA	<i>Greenbone Security Assistant</i>
VM	<i>Virtual Machine</i>

Sumário

1	INTRODUÇÃO	17
1.1	Motivação	19
1.2	Objetivos	19
1.2.1	Geral	19
1.2.2	Específicos	19
1.3	Metodologia	20
1.4	Trabalhos Relacionados	20
1.5	Organização do Trabalho	21
2	SEGURANÇA EM REDES DE COMPUTADORES	23
2.1	Definições	23
2.2	Cenário Geral	26
2.3	Pontos de Vulnerabilidade	27
2.4	Ataques Comuns à Redes de Computadores	29
2.4.1	<i>Scanners</i>	29
2.4.2	<i>Exploit</i>	30
2.4.3	Força Bruta	31
2.4.4	Desfiguração de páginas	31
2.4.5	Negação de Serviços	33
2.4.6	<i>Malwares</i>	33
2.5	Ferramentas para Avaliação de Segurança	35
2.5.1	Nmap	35
2.5.2	Metasploit Framework	36
2.5.3	Pytbull	37
2.5.4	OpenVAS	40
2.6	Conclusão	41
3	SISTEMAS DE DETECÇÃO E PREVENÇÃO DE INTRUSÃO	43
3.1	Definições de IDPS	43
3.2	Tipos de Sistemas de Detecção e Prevenção de Intrusão	44
3.2.1	Sistemas de Detecção de Intrusão Baseados em Host (HIDS)	44
3.2.2	Sistemas de Detecção de Intrusão Baseados em Rede (NIDS)	44
3.2.3	Formas de Detecção	46
3.3	Principais Ferramentas de IDS	46
3.3.1	Snort	46
3.3.2	Suricata	50

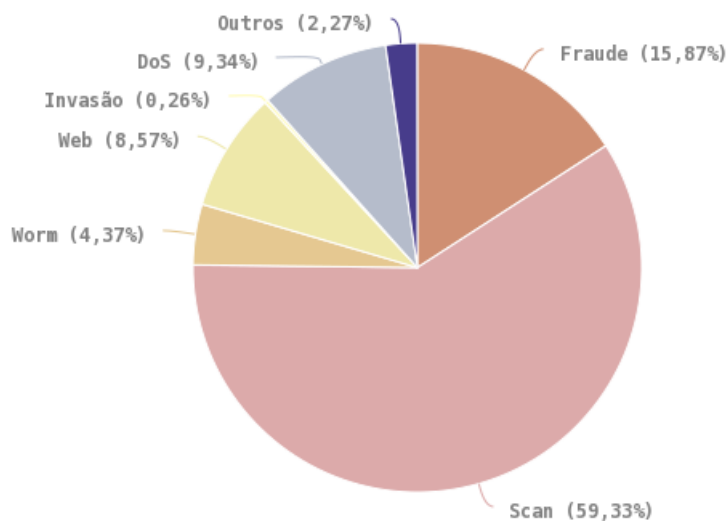
3.4	Conclusão	52
4	DETECÇÃO DE INTRUSÃO EM UM CENÁRIO REAL	53
4.1	Metodologia dos Testes	53
4.1.1	Cenário de Testes	53
4.1.2	Infraestrutura	53
4.2	Testes Realizados	56
4.3	Resultados	60
4.4	Conclusão	64
5	CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS	65
	REFERÊNCIAS	67

1 Introdução

A Internet é um conjunto de redes físicas heterogênea (uma variedade de dispositivos conectados, *smartphones*, *desktops*, *notebooks*, servidores, *switches*, roteadores, entre outros) funcionando como uma rede lógica única de alcance mundial. O grande e contínuo crescimento da Internet gerou um aumento da sua complexidade, que a expõe a diversas vulnerabilidades.

A todo momento, novos ataques ou mesmo variações de ataques já existentes surgem e são lançados a várias redes indiscriminadamente em busca de vulnerabilidades. Em 2015, foram reportados ao Centro de Estudos, Respostas e Tratamento de Incidentes de Segurança no Brasil (CERT.br) cerca de 722.205 incidentes, em 2016, esse número diminuiu, chegando a 647.112 (Figura 1). Apesar da diminuição, esse número é considerado grande, presumindo-se que há muitos incidentes que não são reportados e/ou identificados (INTERNET, 2017c).

Figura 1 – Estatísticas de ataques reportadas ao CERT.br



Fonte: (INTERNET, 2017d)

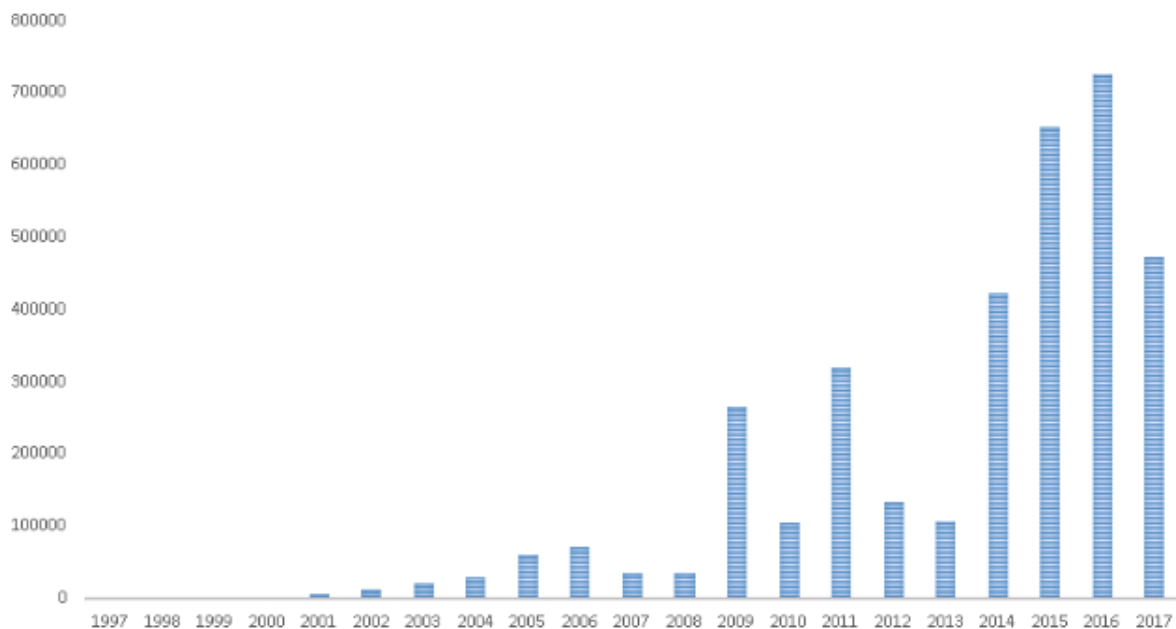
O CERT.br é o grupo de resposta a incidentes de segurança para a internet brasileira, mantido pelo Comitê Gestor da Internet no Brasil. Atua na notificação e tratamento de incidentes de segurança dando apoio no processo de resposta. Além disso, faz um trabalho de conscientização e treinamento sobre problemas de segurança no Brasil.

Paralelamente ao CERT.br temos o Centro de Atendimento a Incidentes de Segurança (CAIS), mantido pela Rede Nacional de Ensino e Pesquisa (RNP). O CAIS é responsável por zelar pela segurança da rede Ipê (infraestrutura de rede dedicada à

comunidade brasileira de ensino superior), detectando, resolvendo e prevenindo incidentes de segurança. Além disso, tem o papel de orientar (através de publicações de cartilhas) e disseminar boas práticas de segurança da informação, educando e conscientizando usuários de todos os níveis sobre os principais riscos em segurança da informação (SEGURANÇA, 2017).

Desde 2008, todas as fraudes identificadas pelo CAIS estão sendo ordenadas e disponibilizadas para consulta (Figura 2). Adicionalmente, são enviados alertas através de uma lista quando uma fraude mostra-se, particularmente, perigosa aos usuários e computadores.

Figura 2 – Estatísticas de incidentes reportados ao CAIS



Fonte: (SEGURANÇA, 2017)

As empresas, de qualquer segmento e tamanho, devem ter o trabalho de manter os ativos seguros e isso vai além da utilização de anti-vírus nos computadores. Ter uma política de atualização de *software* em estações de trabalho e servidores, aliados com boas práticas nas configurações de serviços, dificultam a exploração de vulnerabilidades.

A utilização de *firewall*, não pode ser encarado com uma solução definitiva, passando uma falsa sensação de segurança, pois muitas portas legítimas podem estar vulneráveis, como acontece, por exemplo, com a porta 80 que hospeda *websites* vulneráveis (MARTINELO; BELLEZI, 2014).

Diante desse cenário, torna-se cada vez mais importante para o administrador de rede e/ou segurança da informação o uso de ferramentas de IDPS, permitindo identificar e tratar de forma automatizada os incidentes de segurança.

Para implementar tais ferramentas em uma rede, deve-se levar em consideração a flexibilidade e a administração simplificada para não resultar que a empresa/instituição fique na dependência de um único fabricante ou fornecedor da solução. Além disso, a ferramenta deve ter um bom desempenho e eficiência para não passar a falsa sensação de proteção ou degradar o desempenho da rede.

1.1 Motivação

A motivação desse trabalho surgiu da necessidade de implantação de um Sistema de Detecção e Prevenção de Intrusão por parte da recém formada equipe de segurança da Universidade Federal do Pará (UFPA). Tal necessidade visa melhorar a segurança de modo geral da rede acadêmica da instituição.

Assim, diante de uma pesquisa e devido questões orçamentárias, procurou-se ferramentas de uso gratuito e com bom desempenho, capaz de atender, de forma satisfatória, as necessidades da instituição.

Como desdobramento, encontrou-se duas ferramentas, o Snort, que se encontra no mercado desde 1998, foi a primeira no seu segmento a realizar análise de tráfego em tempo real, e o Suricata, lançado em 2010, com uma arquitetura similar ao Snort, no entanto, utiliza *multithreading*, visando melhorar ainda mais o desempenho.

1.2 Objetivos

1.2.1 Geral

Este trabalho tem como objetivo geral, avaliar e fazer um comparativo entre as soluções de código aberto de Sistemas de Detecção e Prevenção de Intrusão: Snort e Suricata.

1.2.2 Específicos

Como desdobramento do objetivo geral, os seguintes objetivos específicos foram definidos:

- a) Apresentar conceitos sobre segurança da informação e sua importância em um ambiente corporativo;
- b) Descrever problemas relacionados a ataques envolvendo redes de computadores;
- c) Descrever as ferramentas, compreendendo requisitos, características, modos de atuação e funcionalidades;
- d) Descrever o ambiente experimental;

- e) Realizar experimentos e coletar dados para validar o funcionamento e a eficiência das ferramentas;

1.3 Metodologia

O método de pesquisa utilizado é o qualitativo, apoiando-se em técnica de coleta e análise de dados a partir de ferramentas. O estudo foi desenvolvido a partir de:

- a) Pesquisa bibliográfica: Os conceitos e referências teóricas para entendimento do trabalho analisados foram: "Redes de Computadores", documentação e trabalhos referentes as ferramentas em estudos e a cartilha do CERT.br que descreve os ataques lançados as redes;
- b) e a pesquisa de campo feita através da implementação de uma infraestrutura capaz de coletar dados, utilizando ferramentas auxiliares, sobre os objetos estudados.

A infraestrutura montada utiliza várias tecnologias, dentre elas temos virtualização, monitoramento de servidores, espelhamento de tráfegos de rede, etc. Para obter uma conclusão, os dados coletados foram comparados e ao final verificou-se quais das ferramentas obtiveram melhores resultados.

1.4 Trabalhos Relacionados

O uso de ferramentas IDS *opensource* Snort e/ou Suricata, importantes na área de segurança, já foi abordado e apresentou alguns resultados satisfatórios em pesquisas anteriores, por exemplo, nas pesquisas de Nagahama *et al.* (2013), Cléber *et al.* (2014) e Martín *et al.* (2014).

No trabalho do Nagahama *et al.* (2013), é usado redes definidas por *softwares*, que desacopla os planos de controle e de dados, permitindo adaptar o funcionamento da rede de acordo com a necessidade de cada um. O *software* utilizado na pesquisa foi o OpenFLOW. Tal uso, tem como objetivo mitigar a falta de integração do IDS com os equipamentos de rede como switches e roteadores, o que limita a atuação destas ferramentas.

O trabalho do Nagahama foi de fundamental importância para o entendimento dos conceitos relacionados a Sistemas de Detecção e Prevenção de Intrusão e sobre a ferramenta Snort. No entanto, ele não faz um comparativo de desempenho entre as ferramentas proposto nesse trabalho.

Já Cléber *et al.* (2014), foi feito uma comparação de desempenho das ferramentas de IDS (Snort e Suricata), porém usou-se dados sintéticos fornecidos pela *Defense Advanced Research Projects Agency* (DARPA) e devido ser uma base com ataques antigos, não houve

resultados satisfatórios. Ao final, listou-se as vantagens e desvantagens existentes de cada ferramenta. No trabalho proposto, no entanto, serão usados dados mais próximo de um ambiente de produção de uma rede acadêmica.

Por fim, Martín *et al.* (2014), utiliza-se do BroFlow que possui uma série de vantagens, como, detecção de intrusão através de algoritmos simples, modular e flexível, reação imediata a um ataque descartando pacotes dos atacantes os mais próximo da origem. Dentre os resultados obtidos, destacam-se que a ferramenta conseguiu garantir o encaminhamento de pacotes legítimos na rede na taxa máxima do enlace e reduziu, em até dez vezes, o atraso na rede provocado pelo ataque.

1.5 Organização do Trabalho

O restante desse trabalho está organizado da seguinte forma.

No [Capítulo 2](#), são definidos conceitos sobre redes de computadores e segurança da informação, também são descritos os ataques comuns e ferramentas utilizadas para validar e avaliar as soluções de IDPS.

Em seguida, no [Capítulo 3](#), a definição IDPS, os tipos existentes, as funcionalidades e descrição das ferramentas avaliadas: Snort e Suricata.

O [Capítulo 4](#) detalhará o cenário real e a infraestrutura utilizada para os realização dos testes das ferramentas, os testes realizados e o resultados obtidos.

Por fim, no [Capítulo 5](#), as considerações finais e trabalhos futuros.

2 Segurança em Redes de Computadores

Esse capítulo contextualiza o trabalho e apresenta os principais conceitos relacionados à segurança e rede de computadores, mostrando seus principais componentes e os ataques mais utilizados contra essas redes. Ao final, foram apresentadas as ferramentas usadas para simular ataques com o intuito de avaliar o comportamento dos IDPS.

A organização deste capítulo segue da seguinte forma: A próxima seção apresenta as definições sobre segurança da informação. Na [seção 2.2](#) será apresentado uma topologia de rede comum, que existe nas organizações. Na [seção 2.3](#) será abordado os pontos vulneráveis em uma rede. Na [seção 2.4](#) será apresentado o principais ataques a rede de computadores. Por fim, na [seção 2.5](#) será apresentada as ferramentas usadas para gerar os ataques.

2.1 Definições

Quando se fala em segurança de sistemas computacionais, logo vem a mente da maioria dos usuários da rede, roubo de número de cartões de crédito, *hackers* danificando páginas e aplicações *Web* e ataques de negação de serviço. Também vem a imagem dos *malwares*, como vírus, cavalos de tróia e *worms*. Esses possuem maior visibilidade pois representam uma parte significativa das ameaças existentes na Internet.

Existem, porém, outros problemas que apresentam riscos que normalmente não são levados em consideração, como administradores desonestos, funcionários descontentes e usuários que utilizam dados sigilosos de forma equivocada.

Para um melhor entendimento sobre segurança da informação, precisa-se entender alguns elementos listados abaixo (COELHO; ARAUJO; BEZERRA, 2014):

- a) **Incidente de segurança:** qualquer evento oposto a segurança; por exemplo, ataques de negação de serviços (*Denial of Service* - DoS), roubo de informações, vazamento e obtenção de acesso não autorizado a informações;
- b) **Ativo:** qualquer dado, informação e equipamento que tenha valor para a organização e para seus negócios. Alguns exemplo: banco de dados, softwares, equipamentos (computadores e notebooks), servidores, elementos de redes (roteadores, switches, entre outros), pessoas, processos e serviços;
- c) **Vulnerabilidade:** qualquer fraqueza que possa ser explorada e comprometer a segurança de sistemas ou informações. Fragilidade de um ativo ou grupo de ativos que pode ser explorada por uma ou mais ameaças. Vulnerabilidades são falhas que permitem o surgimento de deficiências na segurança geral do computador

ou da rede. Configurações incorretas no computador ou na segurança também permitem a criação de vulnerabilidades. A partir dessa falha, as ameaças exploram as vulnerabilidades, que, quando concretizadas, resultam em danos para o computador, para a organização ou para os dados pessoais;

- d) **Ameaça:** qualquer evento que explore vulnerabilidades. Causa potencial de um incidente indesejado, que pode resultar em dano para um sistema ou organização;
- e) **Risco:** probabilidade de uma ameaça se concretizar;
- f) **Ataque:** qualquer ação que comprometa a segurança de uma organização;
- g) **Impacto:** consequência de um evento.

Diante desses elementos, podemos definir segurança da informação como sendo a proteção das informações, sistemas, recursos e demais ativos contra desastres, erros (intencionais ou não) e manipulação não autorizada, objetivando a redução da probabilidade e do impacto de incidentes de segurança.

Segundo (TÉNICAS, 2013), segurança da informação é a preservação da confidencialidade, da integridade e da disponibilidade da informação, adicionalmente, outras propriedades, tais como autenticidade, responsabilidade, não repúdio e confiabilidade, podem também estar envolvidas.

O conceito mais básico e considerado o pilar de toda a área de segurança corresponde à sigla CID (Confidencialidade, Integridade e Disponibilidade), de modo que um incidente de segurança é caracterizado quando uma dessas áreas é afetada (PEIXINHO; FONSECA; LIMA, 2013). Abaixo será detalhado cada item.

- a) **Confidencialidade:** termo ligado à privacidade de um ativo ou recurso, que deve ser acessível somente por pessoas ou grupos autorizados;
- b) **Integridade:** possui duas definições, a primeira está relacionada com o fato da informação ter valor correto, a segunda, está ligada à inviolabilidade da informação;
- c) **Disponibilidade:** está relacionada ao acesso à informação, que deve estar disponível quando necessária.

Dois dos termos citados são fáceis de ser monitorados pois é perceptível para o usuário: a integridade (capacidade de identificar se uma informação foi alterada) e a disponibilidade (através da tentativa de acesso a um serviço e verificando se o mesmo está respondendo adequadamente). No entanto, só é possível identificar se houve quebra da confidencialidade com auditorias, analisando os registros de acesso (se houver), isso torna a identificação custosa e em muitos casos impossível (PEIXINHO; FONSECA; LIMA, 2013).

Além dos conceitos listados, a literatura considera mais alguns conceitos auxiliares:

- a) **Autenticidade:** garantia que uma informação, produto ou documento foi elaborado ou distribuído pelo autor a quem se atribui;
- b) **Legalidade:** garantia de que ações sejam realizadas em conformidade com os preceitos legais vigentes e que seus produtos tenham validade jurídica;
- c) **Não repúdio:** conceito bastante utilizado em certificação digital, onde o emissor de uma mensagem não pode negar que a enviou;
- d) **Privacidade:** habilidade de uma pessoa controlar a exposição e a disponibilidade de informações acerca de si.

Os ataques são classificados em passivo e ativo. Em um ataque passivo não há interação direta (modificações de arquivos ou afetando os recursos) com o sistema alvo, o atacante apenas monitora com o objetivo de obter informações. Por outro lado, os ataques ativos há modificações de dados que afetam as operações do sistema. Os ataques podem ser divididos em categorias apresentadas na tabela.

Tabela 1 – Classificação dos ataques passivos e ativos

Ataque	Categoria	Descrição
Passivo	Liberação de conteúdo da mensagem	Ocorre quando uma informação é captada e seu conteúdo é lido pelo atacante
	Análise de tráfego	Ocorre quando o tráfego da troca de uma informação (criptografada ou não) é analisado para identificar padrões nas mensagens
Ativo	Disfarce	Ocorre quando uma entidade finge ser outra entidade
	Repetição	Ocorre quando os dados são capturados passivamente e, subsequentemente, retransmitidos para produzir um efeito não autorizado
	Modificação da mensagem	Ocorre quando alguma parte da mensagem original é alterada para produzir um efeito não autorizado
	Negação de serviço	Ocorre quando há um impedimento ou inibição do uso ou gerenciamento normal das instalações de comunicação

Fonte: Autoria própria

Além disso, pode-se dividir os ataques em quatro categorias, que são (CLARO, 2015):

- a) **Interrupção:** Esse ataque tem como objetivo interromper ou destruir o serviço, afetando a disponibilidade da informação, como ocorre, por exemplo, nos ataques de negação de serviço (DoS) e ataques de negação de serviço distribuído (DDoS);
- b) **Interceptação:** Esse ataque visa capturar informações que estão em trânsito sem a percepção do vítima, comprometendo sua privacidade. Seu objetivo

principal é gerar cópias de informações, arquivos e programas de forma não autorizada. Um exemplo desse tipo de ataque é o *Man-in-the-Middle*.

- c) **Modificação:** Esse ataque ocorre quando as informações transmitidas são alteradas, após serem captadas, afetando sua integridade. Como exemplo desse ataque temos o *Replay Attack*.
- d) **Falsificação:** Esse ataque tem como finalidade se passar por um usuário do sistema para obter informações e transmiti-las na rede, comprometendo a autenticidade da informação. Como exemplo desse ataque temos o IP *Spoofing*.

2.2 Cenário Geral

Nessa seção serão apresentados conceitos sobre redes de computadores e descrever uma topologia de rede genérica conectada a internet.

Uma rede de computadores é um conjunto de dispositivos interconectados para compartilhar recursos como *hardware*, *software*, interação e interatividade, onde existem máquinas que desempenham os papéis de clientes, servidores e/ou parceiros dependendo do serviços disponíveis na rede (JUSTO; TAMARIZ, 2012).

Um administrador, com o mínimo de consciência sobre segurança, coloca em sua rede, um *firewall* de borda. Um *firewall* sempre é colocada na divisa entre duas ou mais redes, pode ser entre redes privadas ou entre uma rede privada e a Internet. Uma empresa pode ter muitas LANs conectadas de forma arbitrárias, mas todo o tráfego de saída ou de entrada da empresas para a internet deve ser feito através do *firewall*, permitindo assim, que alguns pacotes passem e bloqueando outros (TANENBAUM; WETHERALL, 2011).

Há três tipos básicos de *firewall*, os mais tradicionais são os filtros de pacotes e os *proxies*. O terceiro tipo é uma evolução do filtro de pacotes tradicional chamado de filtro de estados de pacotes ou *stateful packet filter* (SPF) (ULBRICH; VALLE, 2007).

Um *firewalls* de filtros de pacotes são baseados em tabelas configuradas pelo administrador da rede. Essas tabelas listam as origens e os destinos aceitáveis e/ou bloqueados e as regras padrões que orientam o que deve ser feito com os pacotes recebidos de outras máquinas ou destinados a elas, ou seja, o *firewall* tem como função controlar o tráfego entre as redes (TANENBAUM; WETHERALL, 2011).

Há vários *softwares* que implementam filtro de pacotes. Alguns são instalados em *hardwares* como roteadores outros são programas que rodam em computadores comuns (ULBRICH; VALLE, 2007). Um utilitário bastante conhecido e utilizado para essa finalidade é o *iptables*. A Tabela 2 apresenta um exemplo de uma tabela de regras.

No exemplo, são permitidas conexões na Intranet no Servidor Web pelas portas 80 e 443 (padrão nos protocolos HTTP e HTTPS) para todos os *Flags* TCP (ACK, ACK/SYN,

Tabela 2 – Tabela de regras aplicadas no *firewall*

IP Origem	IP Destino	Porta Origem	Porta Destino	Protocolo	Flag TCP	Ação
Rede Externa	Servidor Web	Todas	80,443	TCP	Todos	Permitir
Rede Externa	Servidor Web	Todas	21,3000:3070	TCP	Todos	Permitir
Todas	Todas	Todas	Todas	Todos	Todos	Negar

Fonte: Autoria própria

SYN e FIN). Além disso, podemos definir um range de portas, como na linha 2, que são abertas as portas 21 e todas as portas entre 3000 e 3070, utilizadas por padrão pelo protocolo FTP.

Um *proxie* trabalha na camada de aplicação interagindo com o programa e seus protocolos, independente de como esse protocolo será encapsulado na pilha TCP/IP. Por exemplo, um *proxy* para Web trabalha apenas com o protocolo HTTP, bloqueando os demais. Além disso, pode-se configurar-lo para controlar quem pode ou não acessar serviços externos (ULBRICH; VALLE, 2007).

No *firewall* de filtros de pacotes por estado (SPF) uma nova tecnologia de análise de pacotes foi agregada, permitindo que eles lembrem-se de pacotes anteriores antes de permitir outro mais recente entrar. Isso é implementado na forma de uma tabela de conexões ativas. Quando uma conexão é iniciada, todos os dados do pacote são guardados nela. Se um novo pacote chegar em direção à mesma máquina, o SPF consulta a tabela. O novo pacote é aceito caso seja dada a continuação da conexão ou rejeitado, se não for (ULBRICH; VALLE, 2007).

Na Figura 3 apresenta uma típica rede composta por um roteador de núcleo que interliga roteadores (A, B e C) de outras redes da Intranet, que por sua vez interliga os clientes e/ou servidores. Todo tráfego de saída e entrada da Intranet para a Internet passa pelo roteador de núcleo, além disso, os pacotes são tratados por um *firewall* de borda, que determina o que entra e o que sai da rede local.

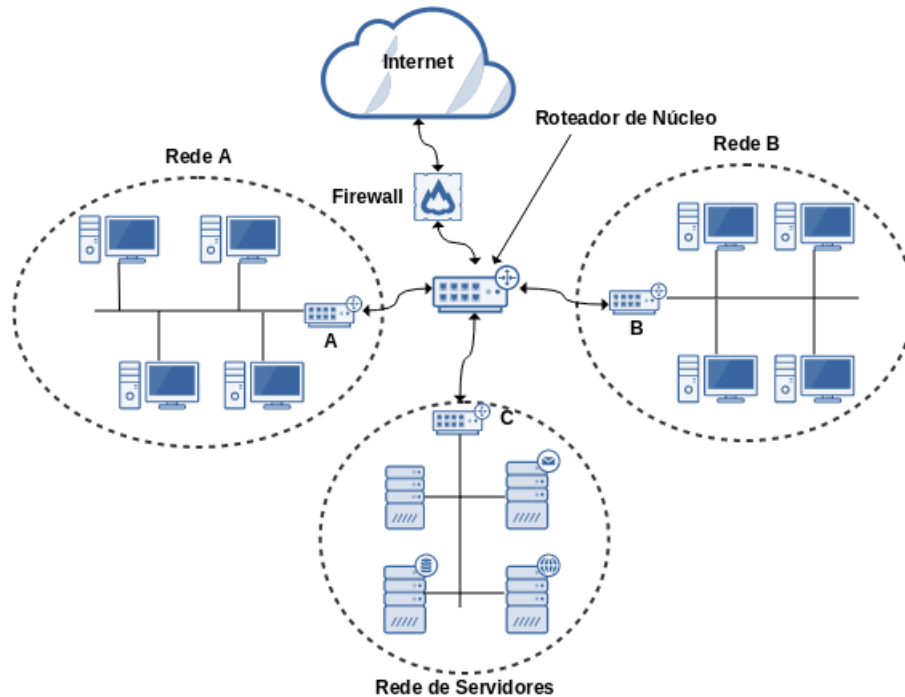
2.3 Pontos de Vulnerabilidade

Nessa seção será abordado os pontos fracos que uma pessoa má intencionada pode explorar para ter um ataque bem sucedido a um rede de computador.

Apesar da preocupação dos administradores em proteger suas redes de ataques, devido a sua heterogeneidade, sempre haverá uma fragilidade a ser explorada. A literatura considera o ser humano como elo mais fraco, é bastante comum o usuário cadastrar senhas fracas, por conveniência, e fácil memorização.

Um problema muito comum é o desenvolvimento de aplicações *web* sem nenhuma preocupação com segurança, podendo comprometer, não somente o serviço, mas também,

Figura 3 – Topologia geral de uma rede de computadores



Fonte: Autoria própria

em casos mais extremos, o servidor inteiro. Para tal, o atacante pode usar vários artifícios, os mais conhecidos são *sql injection* e *cross-site script*.

A inserção de *Structured Query Language* (SQL) via formulário na aplicação *web* resulta num ataque de *sql injection*. O atacante injeta um código dentro dos campos de entrada, como usuário e senha, de uma aplicação onde a declaração condicional sempre será verdadeiro quando executado. Em casos bem sucedidos, o atacante pode alterar o banco de dados, acessar informações sensíveis ou ter acesso ao sistema (S; S; M, 2014).

No exemplo abaixo, a declaração condicional 'OR 1=1' torna toda a clausura WHERE verdadeiro pois a expressão 1=1 é uma tautologia. A consulta retorna todos os dados da tabela *user_info*. Perceba que os dois hífen fornecidos no final da entrada comenta o resto da linha.

```
SELECT * FROM user_info WHERE logID="" OR 1=1 — AND pass1=""
```

O *Cross-site scripting* (XSS) é uma forma de ataque que permite utilizar um aplicação vulnerável para transporta códigos maliciosos até o navegador de outros usuário. O navegador da vítima entende que o código recebido é legítimo e, por isso, informações sensíveis, como o identificador de sessão do usuário, por exemplo, podem ser acessadas programaticamente (UTO, 2013).

Com o XSS pode-se roubar histórico de navegação, fazer uma varredura de redes

privadas, descobrir consultas realizadas em mecanismos de busca, escravizar o navegador *web* e proliferar *worms* (subseção 2.4.6) baseados em XSS (UTO, 2013).

2.4 Ataques Comuns à Redes de Computadores

Nessa seção será descritos os ataques mais comuns à redes e serviços de organizações privadas e públicas, financeiras ou acadêmicas. Para licitar os ataques dessa seção, levou-se em consideração as estatísticas divulgada pelo CERT.br (Figura 1).

2.4.1 Scanners

Uma vulnerabilidade é a fraqueza em sistemas de informação, procedimentos de segurança do sistema e controles internos, ou aplicação que pode ser explorada tendo como origem uma ameaça.

Scanners são programas usados para varrer uma rede à procura de computadores (tanto pessoais como servidores) com alguma vulnerabilidade. Podemos dividir os *scanners* em dois tipos (ULBRICH; VALLE, 2007):

- a) **scanner de portas TCP/IP abertas (ou portscanner)**: cada serviço de rede que estiver disponível em uma determinada máquina é uma porta de entrada em potencial. Existem um total de 128 mil porta, sendo 65536 portas para o protocolo TCP e 65536 portas para o protocolo UDP. O *portscanner* verifica quais portas TCP/IP estão abertas com o objetivo de determinar quais serviços de rede TCP/IP disponíveis. Quase todas as técnicas de *portscanning* valem-se de sinais (ou *flags*), TCP, UDP ou ICMP, e a partir da análise desses sinais, os *scanners* retiram informações sobre o sistema; (ULBRICH; VALLE, 2007)
- b) **scanner de vulnerabilidades conhecidas**: Um vez determinados os serviços que uma máquina disponibilizada na rede entra em cena o *scanner* de vulnerabilidade. A ideia é checar, através de uma lista de falhas conhecidas, se o sistema está ou não executando um serviço com problemas (ULBRICH; VALLE, 2007).

Normalmente, essas ferramentas funcionam em três estágios (BASSO, 2010):

- a) **Configuração**: aqui será definido o endereço IP do alvo ou a URL (*Uniform Resource Locator*) da aplicação Web e demais parâmetros, como, por exemplo, utilização de *proxy*.
- b) **Rastreamento**: esse estágio, em *scanners* de vulnerabilidade de aplicações web, o *scanner* chama a primeira página web e então examina seu código procurando

links. Cada *link* encontrado é registrado e este procedimento é repetido várias vezes até que *links* e páginas não sejam mais encontrados.

- c) **Exploração:** vários testes são executados e as requisições e respostas são armazenadas e analisadas. Ao final, os resultados são exibidos ao usuário e podem ser salvos para uma análise posterior.

Um bom *scanner* de vulnerabilidade verifica itens como (ULBRICH; VALLE, 2007):

- a) **Erros comuns de configuração:** portas não utilizadas por nenhum serviço abertas;
- b) **Configurações e senhas-padrões:** instalação de softwares deixando-os com as configurações de fábrica (com usuário e senha-padrão), por exemplo, usuário: admin, senha: admin. Outro problema é deixar serviços desnecessários ativados;
- c) **Combinação óbvias de usuário e senha:** Usuário comuns tendem a colocar senhas fáceis de lembrar;
- d) **Vulnerabilidades divulgadas:** Sempre que uma falha de segurança é divulgada há uma corrida dos desenvolvedores para saná-las. Em paralelo, existem *hackers* que querem chegar aos sistemas vulneráveis antes de serem consertados.

Os *scanners* de vulnerabilidades automatizados contêm, e atualizam regularmente, enormes bancos de dados de assinaturas de vulnerabilidades conhecidas para basicamente tudo o que está recebendo de informações em uma porta de rede, inclusive sistemas operacionais, serviços e aplicativos web (MCCLURE; SCAMBRAY; KURTZ, 2014).

2.4.2 Exploit

Os atacantes exploram *bugs* ou vulnerabilidades em programas para ter acesso ao sistema alvo. Infelizmente, existem milhares de *bugs*, em 2013, por exemplo, foram reportados 103,000 *bugs* no sistema operacional Ubuntu. Outros projetos de códigos fechados, possuem estatísticas similares (AVGERINOS et al., 2014).

Diante das vulnerabilidades obtidas por um *scanners* (subseção 2.4.1), o passo seguinte seria usar um *exploit* adequado. Os *exploit* são pequenos utilitários usados para explorar vulnerabilidades específicas, podendo ser utilizados de forma "*stand alone*", ou seja, diretamente, ou podem ser incorporados à *malwares* (NUNES, 2011).

Para alguns *exploits* funcionar, é necessário ter acesso ao *shell* da máquina-alvo. Tal artifício pode ser conseguido através da execução de um cavalo de tróia (subseção 2.4.6) pela vítima em seu sistema. O *trojan* abre uma porta de comunicação e permite que o invasor tenha total controle sobre a máquina, dessa forma é possível executar *exploits* para quebrar outros níveis de segurança (ULBRICH; VALLE, 2007).

2.4.3 Força Bruta

Na segurança da informação, a autenticação é umas das áreas-chaves onde há a distinção de usuários autorizados de outros não-autorizados, tendo como principal vantagem ser de fácil implementação, não requerendo equipamentos, como leitores biométricos (SILVA; STEIN, 2007).

Na literatura sobre segurança da informação, o fator humano é considerado o elo mais fraco. Muitos usuários, por conveniência, criam senhas de acesso fáceis e, em muitos casos, única para acessar diversos sistemas. Nesse ponto que *hackers* iram atuar para ter acesso não-autorizado ao sistema.

Existem três métodos mais usados por programas de quebra de senha: ataques de dicionário (ou lista de palavras), ataques híbridos e ataques de força-bruta. Nos ataques por dicionários, utilizam-se listas de palavras comuns: nomes próprios, marcas conhecidas, gírias, nomes de canções, entre outros, tais elementos conseguidos por engenharia social (ULBRICH; VALLE, 2007).

Um ataque de força bruta consiste em gerar todas as permutações e combinações possíveis de senha, criptografar cada uma e comparar a senha gerada com a senha criptografada original até encontrar uma que seja igual (SCHARDONG; ÁVILA, 2012).

Esse tipo de ataque é facilmente detectável pois, além de gerar uma alta carga no servidor, gera uma grande quantidade de registros de logs. No entanto, caso a pessoa má intencionada, de alguma outra forma, tenha acesso ao arquivo de *hash* ou a tabela de usuário de um banco de dados, com as senha criptografadas do sistema, ela pode usar o ataque de força bruta no arquivo em qualquer máquina, assim, impossibilitando a detecção do ataque.

Muitos sistemas já possuem formas de contornar esse tipo de ataque, por exemplo, bloqueio de usuário ao errar a palavra-chave por uma certa quantidade de vezes. Outra forma, é colocar um tempo de expiração da senha, por exemplo, a senha deve ser trocada a cada trinta dias por uma diferente e nunca usada anteriormente, dessa maneira, inviabilizando a quebra de senha por força bruta.

2.4.4 Desfiguração de páginas

A desfiguração de páginas, *defacement* ou pichação ocorre quando o conteúdo da página *web* de um site é alterado. O atacante (*defacer*) consegue fazer alterações em páginas explorando vulnerabilidade nas aplicações *web* que permite injeção de *script* malicioso ou através de furto de senha de acesso à interface *web* usadas para administração remota (INTERNET, 2017a).

Nos serviços *web*, como por exemplo, apache2 existe um usuário especial, comumente

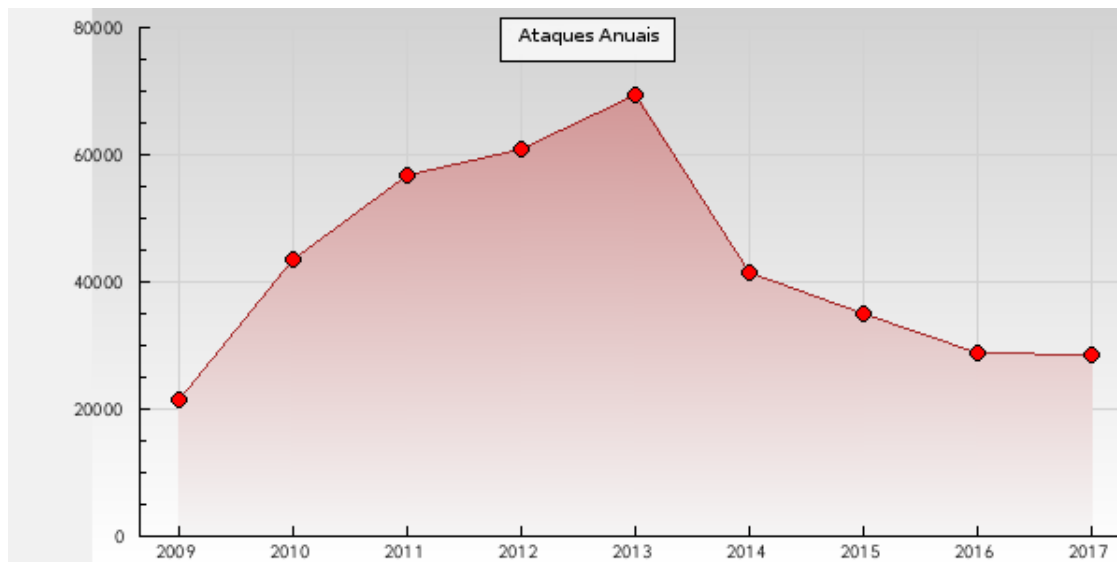
chamado de *www-data* ou algo semelhante. O usuário *www-data*, na maioria das vezes, precisa apenas de permissões de leitura nos arquivos porém muitos gerentes de sistemas cujo a conscientização sobre segurança é insuficiente, designa permissões errôneas (escrita ou alteração), e caso haja um comprometimento, através, por exemplo, de injeção de código remoto PHP, do servidor, o atacante poderá alterar a maioria dos arquivos. A ocorrência amplamente disseminada de ataques de desfiguração de páginas Web é uma consequência direta dessa prática (STALLINGS; BROWN, 2014).

Esse tipo de ataque pode trazer sérias consequências à instituição, entre elas (CERON, 2015):

- a) **Constrangimento:** A instituição pode ter a imagem de confiabilidade afetada, em certos casos, refletir o descaso com que as informações críticas são tratadas;
- b) **Disseminação de inverdades:** Algumas alterações no *website*, por exemplo, alterações de preços de produtos, podem resultar em consequências negativas;
- c) **Prejuízo de serviços:** Pode indisponibilizar serviços prestados pela instituição, por exemplo, em *e-commerce*.

Existem ferramentas que automatizam esse tipo de ataque, elas identificam aplicações web populares vulneráveis, de modo explorar falhas de segurança e alterar o conteúdo da página (CERON, 2015).

Figura 4 – Estatísticas de Desfiguração de Páginas



Fonte: (ZONE-H, 2017)

O site Zone-H mantém um arquivo de páginas alteradas. Os próprios *hackers* submetem os *websites* comprometidos no intuito de ter seus minutos fama. Nas submissões, os sites são espelhados para o Zone-H, então os moderadores verificam a veracidade do *defacement*. Em 2013, foram identificados cerca de 70.000 páginas comprometidas, desde

então houve uma redução nesse número (Figura 4). Esse tipo de ataque é considerado passivo pois é gerado somente uma mensagem na tela (NUNAN, 2012).

2.4.5 Negação de Serviços

Um ataque de negação de serviço (*Denial of Service* - DoS) tem como principal objetivo deixar um serviço (servidor *web*, banco de dados) ou recurso (memória, processador) indisponível, impossibilitando que usuário legítimos tenham acesso a esses recursos. Para tal, o atacante gera diversas requisições inúteis para o servidor, consumindo seus recursos até que o serviço não esteja mais disponível ou degradando a qualidade do serviço (STALLINGS, 2011).

Pode-se dividir os ataques de Dos em três categorias (KUROSE; ROSS, 2013):

- a) **Ataque de vulnerabilidade:** Envolve o envio de um série de mensagens a uma aplicação ou sistema operacional vulnerável, como consequência o serviço pode parar ou, no pior caso, o hospedeiro pode pifar;
- b) **Inundação na largura de banda:** O atacante envia um grande quantidade de pacotes ao hospedeiro, fazendo com que o enlace de acesso do alvo fique indisponível, impedindo os pacotes legítimos de alcançarem o servidor;
- c) **Inundação na conexão:** O atacante estabelece um grande número de conexões no hospedeiro-alvo fazendo-o deixar de aceitar conexões legítimas.

Esse tipo de ataque pode gerar grandes prejuízos financeiros para as empresas, principalmente *e-commence*, pois enquanto o sistema está fora ou com uma resposta lenta, as transações financeiras são prejudicadas. Com isso, cria-se também, uma insatisfação pelo usuário do serviço prestado pela empresa.

Existe uma forma mais sofisticada de ataque de DoS chamada Negação de Serviço Distribuído (*Distributed Denial of Services* - DDoS), enquanto o DoS básico as requisições partem de apenas uma fonte, no entanto, no DDoS o atacante tem acesso a um grande número de computadores (*zombies*) explorando suas vulnerabilidades criando o que chamamos de *botnet* (Figura 2.4.5). Com isso, basta o atacante indicar as coordenadas de um ou mais alvos para o ataque (ZARGAR; JOSHI; TIPPER, 2013). O DDoS são mais difíceis de detectar e de prevenir do que um ataque DoS de um único hospedeiro.

2.4.6 Malwares

Os *malwares*, também conhecidos como *softwares* maliciosos, são um grande problema para sistemas de informação, sua existência ou execução tem consequências negativas ou involuntárias. Nessa seção será apresentado os *malwares* mais popularmente conhecidos que são os vírus, *worms*, *trojans* e, devido sua repercussão, os *ransomwares*.

Figura 5 – Ataque de Negação de Serviço Distribuído



Fonte: Autoria própria

É importante entender o funcionamento e o comportamento desses códigos maliciosos para, a partir daí, buscar soluções contra esse ataque. Existe dois tipos de análise: análise estática, requer uma verificação linha a linha do código malicioso, geralmente o código não está disponível e até mesmo se estiver, o autor do *malware* muitas vezes ofusca o código, tornando esse tipo de análise difícil. Por outro lado, existe a análise dinâmica, o analista monitora a execução e o comportamento do *malware*, esse tipo de análise é imune a ofuscação de código (TILBORG; JAJODIA, 2011).

O Vírus é um programa que se propaga inserindo cópias de si mesmo e se tornando parte de outros programas e arquivos. Para dar continuidade ao processo de infecção, o vírus depende da execução do programa ou arquivo hospedeiro. O principal meio de propagação desse tipo de *software* malicioso são as mídias removíveis, como, por exemplo, *pen-drives* (INTERNET, 2017b).

O *Worm* é um *malware* que se propaga através de e-mails, sites ou *software* baseados em rede, explorando as vulnerabilidades das aplicações. Uma das principais características desse tipo de *software* é a propagação automática, ou seja, sem a intervenção do usuário (JAIN; PAL, 2017).

O *Trojan* ou Cavalo de Troia são programas que precisam ser explicitamente executados para serem instalados no computador. Esse *malware* se disfarça de um programa benigno, por exemplo, cartões virtuais animados, álbuns de fotos, jogos e protetores de tela que ao serem executados o *trojan* é instalado sem o consentimento do usuário. No entanto, o atacante, após invadir um computador, pode instalar o *trojan* alterando as funções já

existentes de programas para executarem ações maliciosas ([INTERNET, 2017b](#)).

Por fim, temos os *ransomwares*. O *ransomware* é um *malware* que criptografa os dados de um computador ou uma rede. A pessoa ou a organização responsável pelo ataque pede um resgate, geralmente pago em cripto moedas, como por exemplo, bitcoin, para manter sua anonimidade, fornecendo uma chave para descriptografar os arquivos mediante o pagamento ([ENIS, 2017](#)).

A melhor medida contra esse tipo de *malware*, uma vez que, não há garantias que o atacante irá fornecer a chave depois do pagamento, além de manter o sistema sempre atualizado, é ter uma política de *backup* regular. O armazenamento de arquivos importantes em outros tipos de mídias não conectadas regularmente ao sistema (removíveis) ou *backup* baseados em nuvens ([ENIS, 2017](#)).

2.5 Ferramentas para Avaliação de Segurança

Nessa seção será descrito as ferramentas auxiliares utilizadas para geração de ataques abordados na [seção 2.4](#) com objetivo de testar e validar as configurações das ferramentas de IDPS estudadas.

2.5.1 Nmap

O Nmap é uma ferramenta de código aberto utilizada para auditoria de segurança e descoberta de rede. A ferramenta é capaz de determinar quais *hosts* estão disponíveis na rede, quais serviços cada *host* está oferecendo, incluindo nome e versão da aplicação, o sistema operacional usado, dentre outras características.

Muitos administradores de sistemas utilizam o Nmap para tarefas rotineiras como, criação de inventário de rede, gerenciamento de serviços, visto que é de suma importância manter os mesmos atualizados e monitoramento de *host*.

Diversos parâmetros podem ser utilizados com o Nmap, possibilitando realizar varreduras das mais variadas maneiras, dependendo do tipo desejado. A lista completa de opções podem ser consultadas na documentação oficial que vem junto da ferramenta ou no site do projeto ([NMAP, 2017](#)).

Na execução do Nmap, o que não for opção ou argumento da opção é considerado especificação do *host* alvo. O alvo pode ser um ou vários, usando uma notação de intervalo por hífen ou uma lista separada por vírgula. Os *hosts* alvos também podem ser definidos em arquivos.

O resultado do Nmap é uma tabela de portas e seus estados ([Figura 6](#)). As portas podem assumir quatro estados, temos ([NMAP, 2017](#)):

- a) **open**: significa que existe alguma aplicação escutando conexões;
- b) **filtered**: há um obstáculo na rede, podendo ser algum *firewall*, que impossibilita que o Nmap determine se a porta está aberta ou fechada;
- c) **closed**: não possui aplicação escutando na porta;
- d) **unfiltered**: a porta responde requisição porém o Nmap não consegue determinar se estão fechadas ou abertas.

Figura 6 – Exemplo de saída do Nmap

```
Starting Nmap 7.40 ( https://nmap.org ) at 2017-06-12 10:30 -03
Nmap scan report for portal.ufpa.br (200.239.64.160)
Host is up (0.00041s latency).
Other addresses for portal.ufpa.br (not scanned): 2801:80:240:8000::5e31:160
rDNS record for 200.239.64.160: marahu.ufpa.br
Not shown: 94 filtered ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
81/tcp    open  hosts2-ns
443/tcp   open  https
3000/tcp   closed ppp

Nmap done: 1 IP address (1 host up) scanned in 1.65 seconds
```

Autoria própria

2.5.2 Metasploit Framework

O Metasploit é um *framework* de código aberto cujo principio básico é desenvolver e executar *exploit* contra alvos remotos e fornecer uma lista de vulnerabilidades existentes no alvo. É uma ferramenta que combina diversos *exploits* e *payloads* dentro de um local, ideal para levantamento de segurança de serviços e testes de penetração (ARYA et al., 2016).

O Metasploit possui uma biblioteca dividida em três partes:

- a) **Rex**: É a biblioteca fundamental, a maioria das tarefas executadas pelo *framework* usarão essa biblioteca;
- b) **MSF Core**: É o *framework* em si, possui, por exemplo, gerenciador de módulos e a base de dados;
- c) **MSF Base**: Guarda os módulos, sejam eles, *exploit*, *encoders* (ferramentas usadas para desenvolver o *payloads*) e os *payloads*. Além disso, são guardadas informações de configuração e sessões criadas pelos *exploits*.

A Interface permite que o usuário interaja com o *framework*. Nele há o *msfconsole* uma interface de linha de comando interativa, o *msfcli* interface de linha de comando

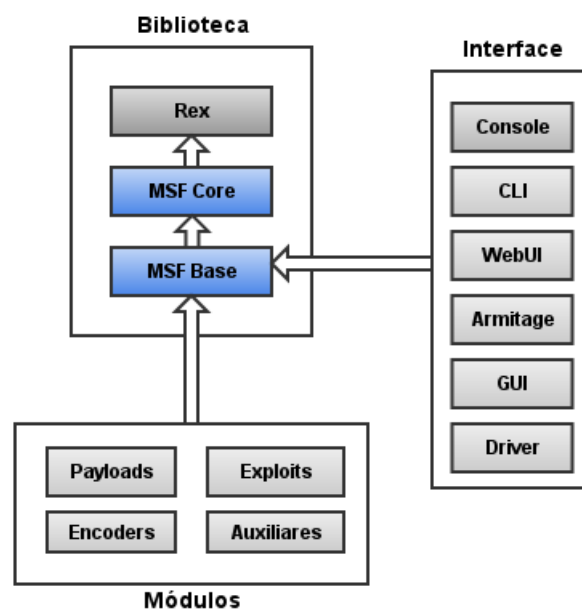
não-interativa, e o *msfweb* interface baseada em *web* (MAYNOR et al., 2007). Por fim, temos o Armitage, que é uma interface gráfica baseada em Java desenvolvido por Raphael Mudge.

A arquitetura é mostrada com mais detalhes na Figura 7.

Os módulos são divididos da seguinte maneira:

- a) **Payload**: são código executados no alvo remotamente;
- b) **Exploit**: explora *bugs* ou vulnerabilidade existente em aplicações do alvo;
- c) **Módulos Auxiliares**: usado para escanear as vulnerabilidades e executar várias tarefas;
- d) **Encoder**: codifica o *payload* para evitar qualquer tipo de detecção por antivírus.

Figura 7 – Arquitetura do Metasploit



Fonte: Autoria própria

2.5.3 Pytbull

O Pytbull é um *framework* para teste de IDPS, capaz de determinar a capacidade de detecção e bloqueio do mesmo, além de fazer uma comparação entre diversas soluções e verifica as configurações (DAMAYE, 2016). O *framework* Pytbull possui cerca de 300 testes agrupados em 11 módulos, temos:

- a) **badTraffic**: pacotes não compatíveis com a RFC são enviados para o servidor para testar como os pacotes são processados;

- b) **bruteForce**: testa a capacidade do IDPS de rastrear ataques de força bruta;
- c) **clientSideAttacks**: usa um *shell* reverso para fornecer ao servidor instruções para baixar arquivos maliciosos;
- d) **denialOfService**: testa a capacidade do IDPS de proteger contra tentativas de DoS;
- e) **evasionTechniques**: testa a capacidade do IDPS de detectar técnicas de evasão;
- f) **fragmentedPackets**: várias cargas úteis fragmentadas são enviadas ao servidor para testar sua capacidade de recomposição e detectar os ataques;
- g) **ipReputation**: testa a capacidade do servidor detectar tráfego de servidores com reputação baixa;
- h) **normalUsage**: cargas úteis que correspondem a uso normal;
- i) **pcapReplay**: permite reproduzir arquivos pcap;
- j) **shellCodes**: envia *shellcodes* para o servidor na porta 21/ftp testando a capacidade de detectar e/ou bloquear o mesmo;
- k) **testRules**, testa a base de assinaturas configuradas no servidor IDPS.

Existem basicamente 5 tipos de testes ([DAMAYE, 2016](#)):

- a) **socket**: Abre um *socket* em uma porta e envia o *payload* para o alvo remoto na porta especificada;
- b) **command**: Envia um comando para alvo remoto com a função `python subprocess.call()`;
- c) **scapy**: Envia cargas úteis específicas baseadas na sintaxe de Scapy;
- d) **client side attacks**: Usa um *shell* reverso no alvo remoto e envia comandos para serem processados no servidor;
- e) **pcap replay**: Permite reproduzir tráfego com base em arquivos de pcap.

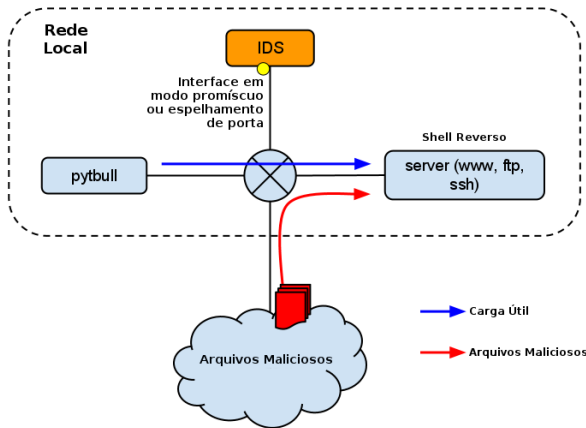
Segundo ([DAMAYE, 2016](#)), o Pytbull pode ser usado seguindo as seguintes arquiteturas:

- a) **Remote Mode**: Nesse modo, o IDS é conectado em um espelhamento de porta no *switch* de núcleo e com a interface em modo *promisc*, analisando todo o tráfego que passa no *switch*. Os arquivos maliciosos podem ser baixados pelo Pytbull ou pelo servidor ([Figura 8](#));
- b) **Local Mode**: Nesse modo, os arquivos maliciosos são baixados pelo Pytbull;
- c) **IDS mode with attacked server in DMZ**: Nessa configuração, o *firewall* divide a rede em três segmentos (lan, wan, DMZ). O IDS é conectado em um

espelhamento do *switch* com um interface em modo *promisc*, analisando todo o tráfego enviado a interface LAN do *firewall* (Figura 11);

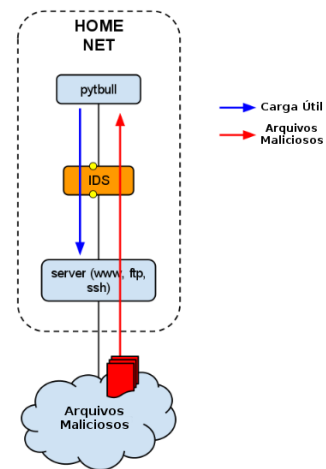
- d) **IPS Mode:** Nessa configuração, o IDS é colocado entre o Pytbull e o firewall. O Pytbull deve fazer o *download* dos arquivos maliciosos para dar chance o IDS detectar (Figura 9);
- e) **IPS mode with attacked server in DMZ:** O mesmo do modo anterior porém com uma DMZ (Figura 10).

Figura 8 – Arquitetura *Remote Mode* do *framework* Pytbull



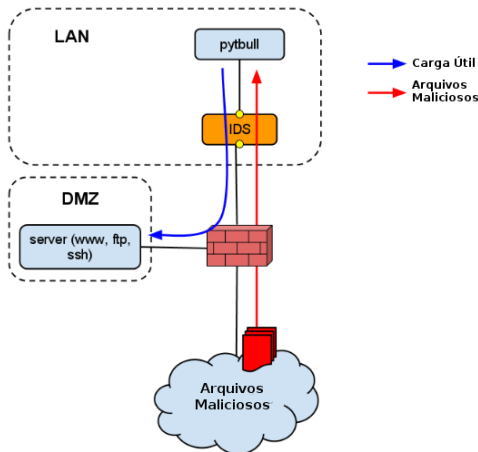
Adaptado de (DAMAYE, 2016)

Figura 9 – Arquitetura *IPS mode* do *framework* Pytbull



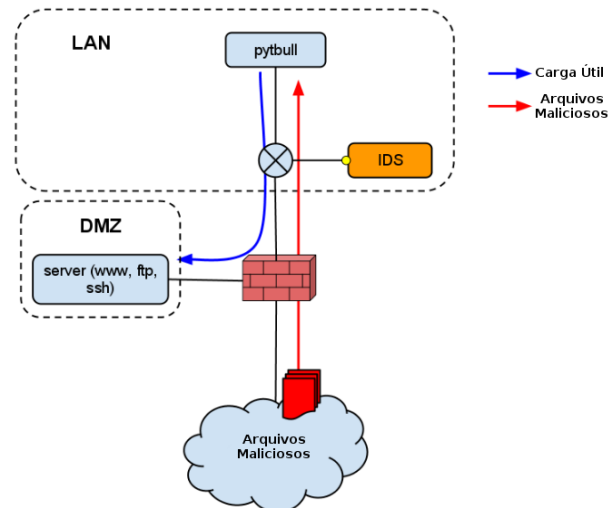
Adaptado de (DAMAYE, 2016)

Figura 10 – Arquitetura *IPS mode with attacked server in dmz* do *framework* Pytbull



Adaptado de (DAMAYE, 2016)

Figura 11 – Arquitetura *IDS mode with attacked server in dmz* do *framework* Pytbull



Adaptado de (DAMAYE, 2016)

2.5.4 OpenVAS

Uma das ferramentas mais populares para buscas de vulnerabilidades é o Nessus, da Tanable Network Secutiry. No entanto, a decisão da Tenable de alterar a licença do Nessus para uso comercial gerou um insatisfação pelo profissionais de segurança.

Em 2005, com o anuncio oficial de fechamento do código do Nessus, foi lançado um *fork* que resultou no OpenVAS. A comunidade que mantém o OpenVAS é crescente e bem ativa e conta com indivíduos e corporações de todo o mundo (BROWN; GALITZ, 2010). Além disso, o OpenVAS é membro da *Software in the Public Interest*, uma organização sem fins lucrativos dedicada a ajudar no desenvolvimento de *Software* Livre.

Segundo o site oficial, o OpenVAS é definido como: "[...] é um *framework* com vários serviços e ferramentas que oferecem um análise de vulnerabilidade abrangente e poderosa solução de gerenciamento de vulnerabilidade." (OPENVAS, 2017).

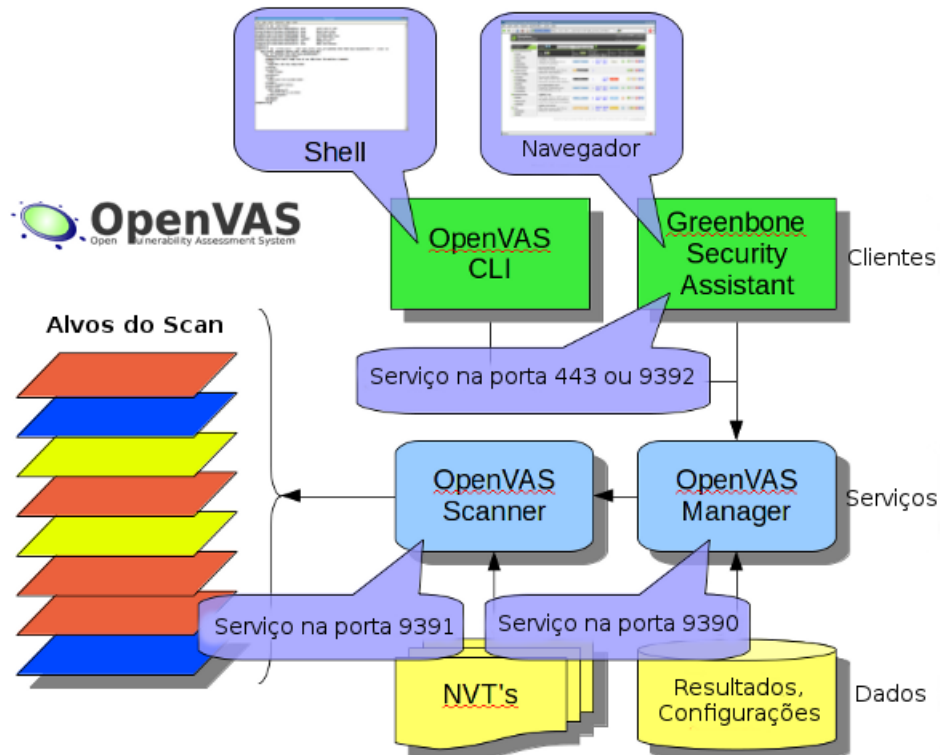
O OpenVAS possui uma arquitetura cliente-servidor, onde o servidor é responsável por todo o processamento, procurando vulnerabilidades e armazena as configurações de varreduras realizadas. Já, do lado do cliente, é disponibilizado uma interface para o administrador realizar as configurações e visualizar os relatórios. Vale ressaltar que toda essa comunicação é tunelada por SSL com cifras fortes para garantir que apenas pessoas autorizadas acesse os dados gerados pelo OpenVAS (BROWN; GALITZ, 2010).

A Figura 12 ilustra a arquitetura do OpenVAS e seus componentes, abaixo segue a descrição de cada um:

- a) **Network Vulnerability Tests (NVT)**: São testes de segurança desenvolvidos na linguagem de script do Nessus, *Nessus Attack Scripting Language* (NASL). Existe um serviço que disponibiliza esses testes diariamente chamado OpenVAS NVT Feed, atualmente contendo 50.000 NVTs. No entanto, é possível desenvolver sua própria NVT para realizar testes para alguma necessidade específica, como, por exemplo, testar sistemas próprios;
- b) **OpenVAS Scanner**: Programa que executa os testes NVTs no alvo;
- c) **OpenVAS Manager**: Executa e gerencia as varreduras feitas pelo OpenVAS Scanner. Os resultados são armazenados em um banco de dados baseados em SQLite. A conexão entre o cliente e o Manager é feita através do protocolo OpenVAS Management Protocol (OMP);
- d) **OpenVAS Administrator**: Módulo que gerencia os usuários e as atualizações dos NVTs. Sua comunicação é feita através do protocolo OpenVAS Administrator Protocol (OAP);
- e) **OpenVAS CLI**: Interface de linha de comando;
- f) **Greenbone Security Desktop (GSD)**: Interface *desktop* baseada em Qt;

- g) **Greenbone Security Assistant (GSA)**: Interface web onde é possível rodar em qualquer navegador.

Figura 12 – Arquitetura OpenVAS



Adaptado de (OPENVAS, 2017)

2.6 Conclusão

Este capítulo apresentou definições sobre segurança da informação, mostrando os elementos envolvidos. Definindo os tipos de ataques existentes e suas categorias e os possíveis impactos. Mostrou-se um cenário geral de uma rede de computador e seus componentes assim como suas definições e os ataques comuns envolvendo essas redes. Ao final do capítulo, mostrou-se as ferramentas auxiliares usadas para simular ataques com objetivo de analisar o comportamento dos IDPS.

3 Sistemas de Detecção e Prevenção de Intrusão

Os sistemas de detecção e prevenção de intrusão (*Intrusion Detection and Prevention System* - IDPS) são ferramentas de importância reconhecida pela comunidade da segurança da informação, pois elas são capazes de detectar se alguém está tentando entrar no sistema ou se algum usuário legítimo está fazendo mau uso do mesmo.

Nesse capítulo, vamos apresentar os principais conceitos relacionados a IDS e IPS, uma breve descrição do funcionamento e classificação, para melhor entendimento das ferramentas que iremos apresentar e avaliar em um ambiente de real.

Esse capítulo está organizado da seguinte forma: Na [seção 3.1](#) será apresentado os conceitos sobre IDS e IPS, seus componentes gerais, os tipos existentes e suas diferenças. Na [seção 3.3](#) será apresentado as ferramentas avaliadas, suas arquiteturas e diferenças. Por ultimo, na [seção 3.4](#), a conclusão do capítulo.

3.1 Definições de IDPS

Intrusion Detection Systems (IDS) ou Sistemas de Detecção de Intrusão (SDI) são ferramentas utilizadas para monitoramento de eventos que ocorrem em redes e sistemas computacionais, analisando sinais de possíveis ataques que podem levar a uma violação das políticas de segurança da organização, alertando os administradores do sistema que estes eventos estão ocorrendo ([NAGAHAMA, 2013](#)).

O *Intrusion Detection Systems* (IPS) ou Sistema de Prevenção de Intrusão (SPI) possui todas as funcionalidades do IDS com uma diferença, ele é capaz de deter os incidentes, minimizando os impactos causados por sistemas comprometidos ([MUKHOPADHYAY; CHAKRABORTY; CHAKRABARTI, 2011](#)).

Os IDS's são compostos basicamente por quatro componentes:

- a) **Sensor ou Agente:** responsável pelo monitoramento e análise do tráfego capturado;
- b) **Base de Dados:** usado como repositório das informações de eventos detectados pelo sensor e que posteriormente serão processados;
- c) **Gestor:** é o dispositivo central que recebe, analisa e gerencia as informações de eventos vindo do sensor;
- d) **Console:** é uma interface para administração e monitoramento das atividades.

3.2 Tipos de Sistemas de Detecção e Prevenção de Intrusão

Os IDPS's são classificados de acordo com o local onde o sensor é instalado, *Host Based Intrusion Detection Systems* (HIDS) e *Network Based Intrusion Detection Systems* (NIDS), e a técnica utilizada para o monitoramento, baseado em assinaturas e anomalias (NAGAHAMA, 2013).

3.2.1 Sistemas de Detecção de Intrusão Baseados em Host (HIDS)

Em um HIDS o sensor é instalado no *host*, monitorando as informações contidas na própria máquina. Esse tipo de IDS não observa o tráfego que passa pela rede (somente o tráfego que passa pela placa de rede do *host*), seu uso volta-se a verificação de informações relativas aos eventos e registros de logs e sistemas de arquivos (permissão, alteração, acesso a arquivos não autorizados) (NAGAHAMA, 2013).

As vantagens do HIDS são:

- a) Evita a execução de códigos maliciosos;
- b) Bloqueia tráfego de entrada e saída contendo ataques e uso não autorizado de protocolos e programas;
- c) Evita que arquivos possam ser acessados, modificados e deletados impedindo a instalação de *malwares* e ataques envolvendo acesso inapropriado a arquivos;

Por outro lado, o HID possui alguns desvantagens como (SCARFONE; MELL, 2007):

- a) Difícil instalação e manutenção;
- b) Interfere no desempenho do *hosts*;
- c) Demora para identificar eventos consequentemente a resposta ao incidente terá um atraso.

3.2.2 Sistemas de Detecção de Intrusão Baseados em Rede (NIDS)

No NIDS, o sensor é instalado na rede e a interface de rede atua em um modo especial chamado “promíscuo”, tendo a capacidade de capturar o tráfego mesmo que os pacotes não sejam destinados ao sensor. Dessa forma, o NIDS monitora e analisa todo o tráfego no segmento da rede, detectando atividades maliciosas, como ataques baseados em serviço, *portscans*, entre outros, além de detectar se algum usuário legítimo está fazendo mau uso da rede (NAGAHAMA, 2013).

Quanto a localização o NIDS pode ser classificado como passivo ou ativo. No modo passivo (Figura 13), o IDS monitora cópias dos pacotes da rede que passam pelo *switch* ou

hub onde está conectado, ficando limitado somente a gerar notificações quando encontrado algum tráfego malicioso.

No entanto, no modo ativo (Figura 14), o IDS é instalado da forma que o tráfego da rede passe através do sensor parecendo com o fluxo de dados associado com um *firewall*. Dessa forma, ele é capaz de parar ataques bloqueando o fluxo malicioso.

É necessário uma análise minuciosa na instalação de um IDS ativo pois um mal dimensionamento de *hardware* pode degradar a rede, adicionando atrasos excessivos aos pacotes.

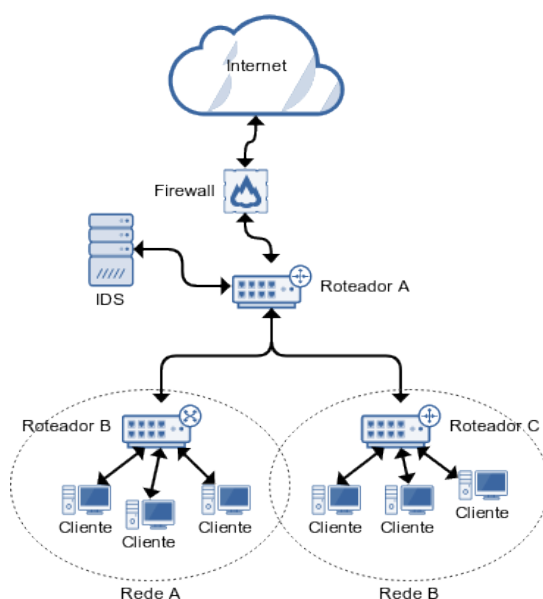
As principais vantagens do um NIDS são:

- a) São independentes de plataformas;
- b) Não interfere no desempenho do *host*;
- c) Fácil implantação e transparente para o atacante.

Dentre as desvantagens, temos:

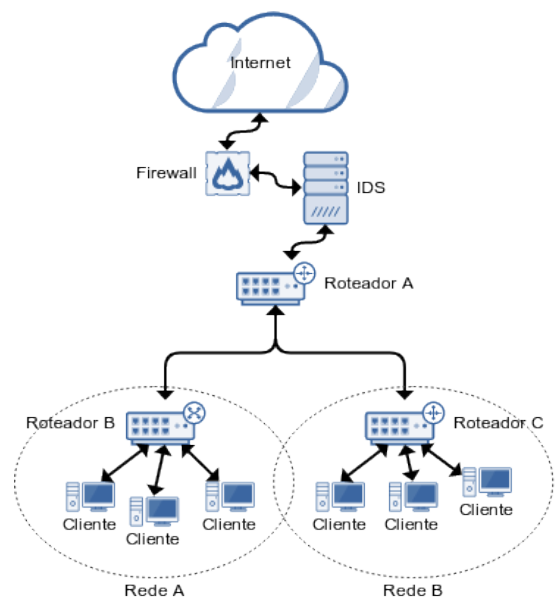
- a) Pode adicionar retardados aos pacotes quando instalado no modo ativo;
- b) Dificuldade de tratar dados de redes de alta velocidade;
- c) Trata apenas segmentos de rede;
- d) Dificuldade de tratar dados criptografados.

Figura 13 – Exemplo de arquitetura de NIDS passivo



Fonte: Autoria própria

Figura 14 – Exemplo de Arquitetura de NIDS ativo



Fonte: Autoria própria

3.2.3 Formas de Detecção

Quanto a técnica de monitoramento utilizado, o IDS pode ser baseado em assinaturas ou anomalias. IDSs baseados em assinaturas compara os pacotes com uma base de assinaturas de ataques previamente conhecidos e reportados por especialistas, cada assinatura identifica um ataque (NAGAHAMA, 2013).

As vantagens de um IDS baseado em assinaturas são:

- a) Usa pouco recurso de hardware do servidor;
- b) Possui, de certa forma, um rápido processamento.

Dentre as desvantagens temos:

- a) Exige uma atualização constante da base de assinaturas;
- b) Para a geração de uma base própria, a equipe precisa de um alto conhecimento técnico;
- c) Possui altos índices de falsos positivos e negativos.

Os IDS baseados em anomalias, procuram determinar um comportamento normal na fase de aprendizagem do sistema computacional ou rede e sempre que existir um desvio desse padrão alertas são gerados.

Possui a vantagem de detectar novos ataques sem necessariamente conhecer a fundo a intrusão através dos desvios de comportamento. Porém, tem como desvantagem a geração de um grande número de falsos alertas em decorrência a modificações na rede ou *host* nem sempre representar um tráfego malicioso.

3.3 Principais Ferramentas de IDS

Nessa seção, será apresentado as ferramentas de IDPS. A escolha dessas ferramentas deu-se devido ser de código aberto e de livre uso, e também, por essas ferramentas não precisarem, para seu funcionamento, de um *hardware* robusto.

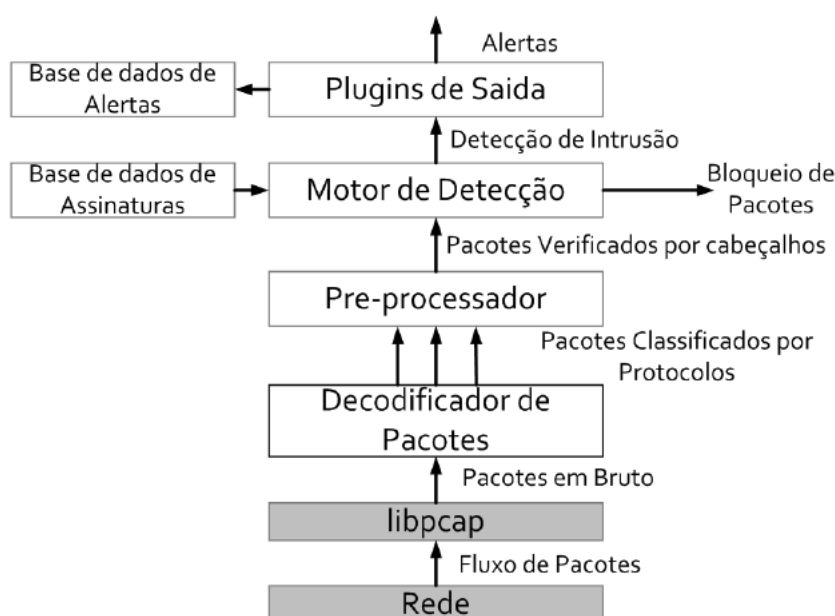
3.3.1 Snort

O Snort é um sistema de detecção e prevenção de intrusão de código fonte aberto escrita na linguagem de programação C. Seu primeiro *release* foi lançado em 1998 e desde então passa por constantes revisões e aperfeiçoamentos, com o passar dos anos se tornou o IDS mais utilizado no mundo. Ele combina análise baseada em assinaturas e anomalias, podendo operar em três modos: *sniffer*, *packet logger* e de sistema de detecção de intrusão (NIDS) (ROESCH; GREEN, 2017).

No modo *Sniffer*, o Snort captura os pacotes e exibe as informações no console de forma contínua. No modo *Packet Logger*, além de capturar o tráfego, o Snort escreve essas informações em arquivos (chamados de logs) que são armazenados no disco. Por fim, o *Network Intrusion Detection System* - NIDS, sendo o modo mais complexo e completo, permitindo capturar e analisar os pacotes de rede em tempo real (ROESCH; GREEN, 2017).

Existe quatro componentes no Snort: O *sniffer*, o pré-processador, o motor de detecção e módulos de saída. A Figura 15 mostra a arquitetura e disposição dos componentes (BAKER; CASWELL; BEALE, 2007).

Figura 15 – Arquitetura do Snort



Fonte: (LOPEZ, 2014)

O pré-processador, o motor de detecção e os componentes de alerta do Snort são todos *plugins*. Os *Plugins* são programas escritos em conformidade com a API de *plugins* do Snort. Esses programas são usados no core do Snort, mas eles são separados para que as modificações feitas no *core* sejam mais confiáveis e mais fáceis de realizar (BAKER; CASWELL; BEALE, 2007).

O *sniffer* é um dispositivo (*software* ou *hardware*) usado para ver o tráfego passante em algum segmento de rede. No caso da Internet, consiste geralmente de tráfego IP (composto por diferentes protocolos de alto nível como, TCP, UDP, ICMP, protocolos de roteamento e IPSec). Os pacotes são analisado, interpretados e exibidos de uma forma legível para os humanos.

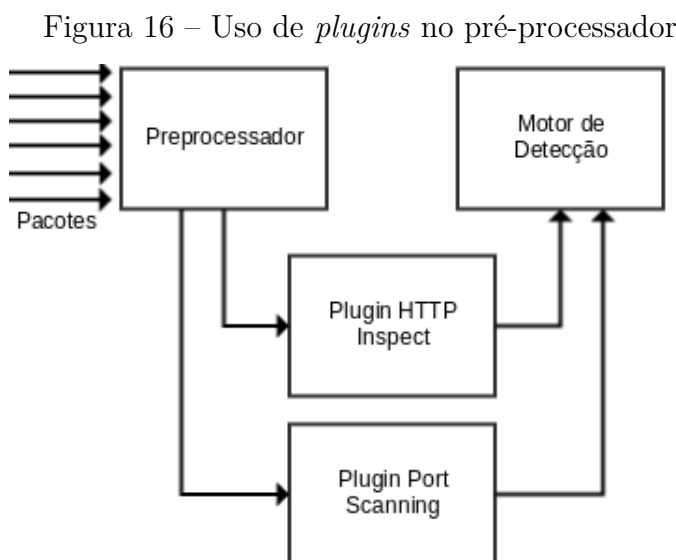
Um *sniffer* tem os seguintes usos:

- a) Analisador de rede e resolução de problemas;
- b) Analisador de desempenho e avaliação comparativa;
- c) Capturar senhas em texto plano e outros dados sensíveis.

Assim como qualquer outra ferramenta de rede, os *sniffers* podem ser usados tanto para o bem quanto para o mal. Então, criptografar o tráfego de rede previne que pessoas sejam capazes de lerem os pacotes capturados (BAKER; CASWELL; BEALE, 2007).

O pré-processador pega o pacote bruto e faz uma checagem utilizando um determinado *plugin*. Esses *plugins* verificam se o pacote tem um tipo particular de comportamento, uma vez determinado, o pacote é enviado para o motor de detecção caso contrário é descartado.

A Figura 16, apresenta como o pré-processador utiliza *plugins* para checar pacotes. O Snort suporta muitos tipos de pré-processadores, cobrindo vários protocolos comumente usados como, IP *fragmentation handling*, *port scanning* e controle de fluxo.



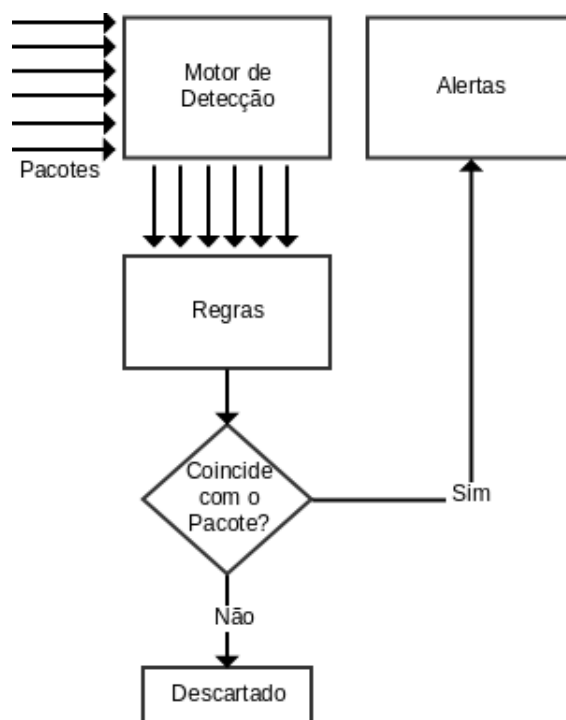
Fonte: Autoria própria

O uso de *plugins* é uma característica muito útil para o IDS, pois os *plugins* podem ser ativados e desativados a medida do necessário, otimizando a utilização dos recursos computacionais e geração de alertas (BAKER; CASWELL; BEALE, 2007).

Os pacotes, após passarem por todos os pré-processadores, são entregues para o motor de detecção. O motor de detecção recebe esses dados e faz uma checagem utilizando uma base de regras pré-configurada pelo administrador. Se a regra for compatível com os dados do pacote, eles são enviados para o processador de alertas, caso contrário, são descartados (BAKER; CASWELL; BEALE, 2007).

Na Figura 17, temos os pacotes saindo dos pré-processadores e chegando no motor de detecção. No motor de detecção há uma base de regras configurada, os pacotes são comparados com as assinaturas da base, se coincidirem, uma ação é tomada, caso contrário, o pacote é descartado.

Figura 17 – Motor de Detecção do Snort



Fonte: Autoria própria

A base de regras é um conjunto de assinaturas de ataques conhecidos e catalogados. As regras são escritas em formato texto em uma única linha e constituídas por duas partes:

- Cabeçalho:** São definidos que ações serão tomadas, tipo de pacote (TCP, UDP, ICMP, etc), o IP de origem e destino e suas respectivas portas;
- Opções:** É o conteúdo do pacote que faz ele ser compatível com a regra.

Dentre as ações que podem ser tomadas temos:

- Activation:** Alerta e chama regra do tipo *dynamic*;
- Dynamic:** permanece inativa até ser ativado por uma regra *activate*, registrando o tráfego;
- Alert:** Gera um alerta usando um método selecionado e então registra os pacotes e dados;
- Pass:** Ignora os pacotes;
- Drop:** Descarta o pacote (quando configurado para atuar de forma ativa (IPS));

f) **Log**: Registra e não alerta.

Abaixo temos um exemplo de regra.

```
alert icmp any any -> any any (msg:"Ping suspeito ";  
sid:1; resp:icmp_all;)
```

Com a regra acima, o Snort gerará um alerta de qualquer pacotes ICMP que estiver passando de qualquer máquina e porta origem (**any any**) para qualquer máquina e porta destino (**any any**) e enviará pacotes ICMP para a máquina de origem com as mensagens *host unreachable*; *network unreachable*.

Se um dado for compatível com uma regra é gerado um alerta. Os alertas podem ser enviados para arquivos de *logs*, através da rede, através de *sockets* UNIX ou Windows Popup (SMB). Os alertas também podem ser armazenados em banco de dados SQL como MySQL e Postgres. Existem vários *plugins* para Perl, PHP e servidores Web para exibir os *logs* através de um interface Web (BAKER; CASWELL; BEALE, 2007).

3.3.2 Suricata

O Suricata é um sistema de código aberto com funções de IDS, IPS e NSM (*Network Security Monitor*). Sua primeira versão oficial foi lançado em 2010 e foi desenvolvido e atualmente é mantido pela *Open Information Security Foundation* (OISF). A OISF é uma fundação sem fins lucrativos formada por um grupo multinacional de desenvolvedores (OISF, 2017).

Uma característica marcante desse IDS é a utilização de uma tecnologia de processamento *multithread* para ter benefício dos múltiplos núcleos de um computador. Além de utilizar *hardware* de aceleração para ter um melhor desempenho (LOPEZ, 2014). Este motor incorpora um normalizador HTTP e analisador que fornece um processamento avançado de fluxos HTTP, permitindo a compreensão do tráfego na camada 7 do modelo OSI (MURINI, 2014).

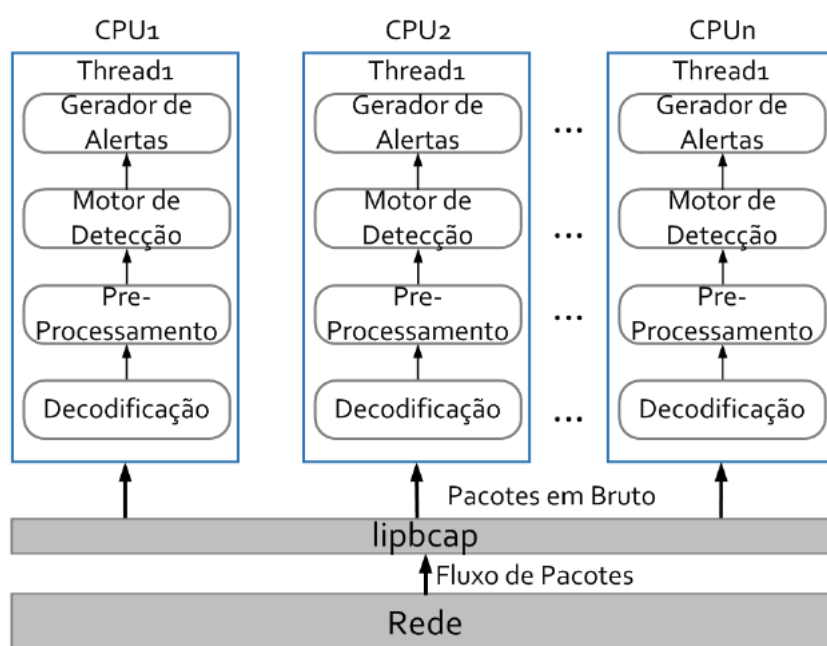
O Suricata utiliza detecção baseada em assinatura e em anomalias. As assinaturas desenhadas para o Snort funcionam no Suricata, podendo ser otimizadas para o uso em seu motor de detecção. As anomalias dos protocolos são fornecidas pelos pré-processadores, e quando implementado no modo ativo, atua na modalidade de prevenção (LOPEZ, 2014). Além disso, implementa uma completa linguagem de assinaturas ligadas a ameaças conhecidas, violação de políticas de segurança e comportamentos maliciosos de *malwares* e outros ataques.

Embora o código do Suricata ser original, os desenvolvedores não hesitam afirmar que a arquitetura foi inspirada no Snort. Na Figura 18 representa a mesma arquitetura

do Snort porém com o mecanismo de *multithread*, sendo essa característica a principal diferença entre as ferramentas (LOPEZ, 2014).

As assinaturas são de grande importância tanto no Snort quanto no Suricata. Muitas pessoas, por conveniência, utilizam conjunto de regras prontas. As mais usadas são da Emerging Threats (ET) e Talos (anteriormente chamado VRT) (OISF, 2017). Talos é um grupo de especialistas em segurança de rede trabalhando o tempo todo para descobrir, avaliar e responder de forma proativa as últimas tendências em atividades de *hacking*, tentativa de intrusão, *malware* e vulnerabilidades (TALOS, 2017).

Figura 18 – Arquitetura *Multithread* do Suricata



Fonte: (LOPEZ, 2014)

A base de assinaturas ET é mantida pela Proofpoint. A Proofpoint é uma empresa especialista em segurança da informação, no site oficial há vários produtos que visam proteger pessoas e dados, detectando e bloqueando ataques e respondendo a essas ameaças (PROOFPOINT, 2017).

O fato de existir equipes dedicadas para o desenvolvimento de uma base de regras, torna o uso dessas bases confiáveis, menos custoso, tem termos de tempo, recursos financeiros e de pessoal e fácil implementação.

Conforme (OISF, 2017) existem três razões para utilizar a ferramenta Suricata:

- a) **Altamente escalável:** O Suricata é *multithreaded*, isso significa que é possível executar instancia do programa e equilibrar a carga de processamento em cada processador em um sensor, permitindo que o *hardware* alcance velocidades de

até 10 Gigabit sobre o tráfego, sem sacrificar a cobertura do conjunto de regras;

- b) **Protocolo de identificação:** Os protocolos mais comuns são reconhecidos pelo Suricata. Além disso, graças a palavra-chave dedicadas, pode combinar em campos que vão desde protocolo HTTP para um identificador de certificado SSL;
- c) **Identificação e extração de arquivos:** Esse software pode identificar milhares de tipos de arquivos durante a varredura na rede. Pode-se também marcar esse arquivo para a extração, e assim o mesmo será gravado em disco como arquivo de metadados que descreve a situação de captura e fluxo.

Além disso, o motor fornece as seguintes funcionalidades ([MURINI, 2014](#)):

- a) Opções de idioma (regra);
- b) Saída unificada permitindo a interação com sistemas externos de registro e gestão, por exemplo, um servidor de *log* ou SIEM;
- c) Baseado em regras de reputação de IP, dessa forma otimizando e reduzindo a geração de alertas falso positivo e falso negativo;
- d) Possui biblioteca ficha-capacidade de interação com outras aplicações e a disponibilização de estatísticas de desempenho.

3.4 Conclusão

Este capítulo apresentou definições sobre IDPS e seus componentes, os tipos existentes e a forma de detecção utilizada. Também foi descrito as ferramentas avaliadas nesse trabalho (Snort e Suricata), destacando suas diferenças e arquiteturas.

4 Detecção de Intrusão em um Cenário Real

Este capítulo está organizado da seguinte forma: A próxima seção apresenta o cenário de testes, descrevendo características gerais da rede selecionada para os testes. Na [seção 4.1](#) será abordado a infraestrutura usada para os testes, ferramentas utilizadas e as configurações feitas. Na [seção 4.2](#) será descrito os testes realizados com suas respectivas justificativas. Na [seção 4.3](#) será apresentado os resultados esperados e obtidos, problemas encontrados e a comparação das ferramentas e por último, na [seção 4.4](#), uma breve conclusão.

4.1 Metodologia dos Testes

Nessa seção será descrito o cenário utilizado para a realização dos testes, descrevendo a rede onde os sensores foram instalados e algumas de suas estatísticas de uso. Posteriormente, descreve-se a infraestrutura montada para realização dos testes, os equipamentos utilizados e as ferramentas adicionais. Na [seção 4.2](#), será descrito os testes realizados. Na [seção 4.3](#) abordará os resultados esperados e obtidos com os testes. Por último, na [seção 4.4](#) a conclusão.

4.1.1 Cenário de Testes

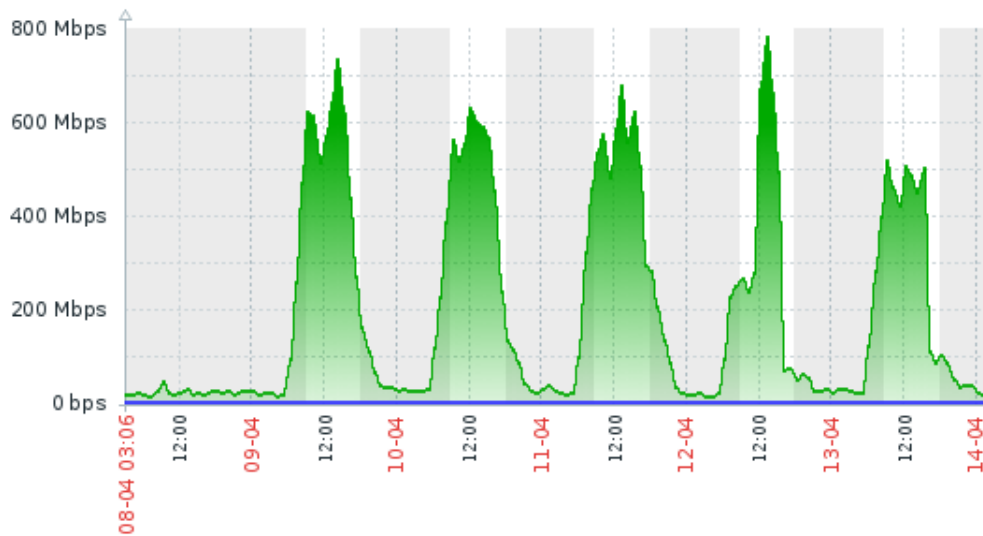
O tráfego da rede na qual os sensores foram colocados para a avaliação possui um pico de taxa de transferência, girando entorno de 800 Mbps ([Figura 19](#)). A quantidade de usuários nessa rede é indeterminado, devido a grande quantidade de dispositivos (servidores, roteadores, rádios) e também, por envolver, redes sem fio, que a todo momento clientes entram e saem da rede.

4.1.2 Infraestrutura

No ambiente de teste foi usado uma máquina Dell com 134 Gigabytes (GB) de memória RAM e 40 núcleos. Foi utilizado XenServer ([XENSERVER, 2017](#)) versão 7, sistema operacional (SO) *opensource* da Citrix voltado para virtualização. Foram testados outros SOs, porém somente o XenServer possuía, na época da instalação do ambiente, *firmware* da placa de rede do *host* compatível e que funcionava com estabilidade.

Outro fator importante na escolha do SO foi a experiência com a plataforma e por existir uma interface para gerencia chamada XenCenter que roda no Windows. Uma alternativa *opensource* desse software é o OpenXenManager que funciona nos sistemas Unix ([LINTOTT, 2017](#)).

Figura 19 – Tráfego da Rede de Teste



Fonte: ([ZABBIX](#), 2017)

No primeiro momento, foi instalado uma máquina virtual com o sistema operacional Debian 9.3 *codename* Stretch ([DEBIAN](#), 2017), uma distribuição linux com uma proposta de ser totalmente livre.

Essa VM foi utilizada como base para instalação de outras máquinas utilizando o recurso de *snapshot*, uma cópia de uma máquina virtual rodando em um certo momento, do XenServer. O uso desse recurso foi necessário para criar um ambiente igual para as ferramentas em estudo.

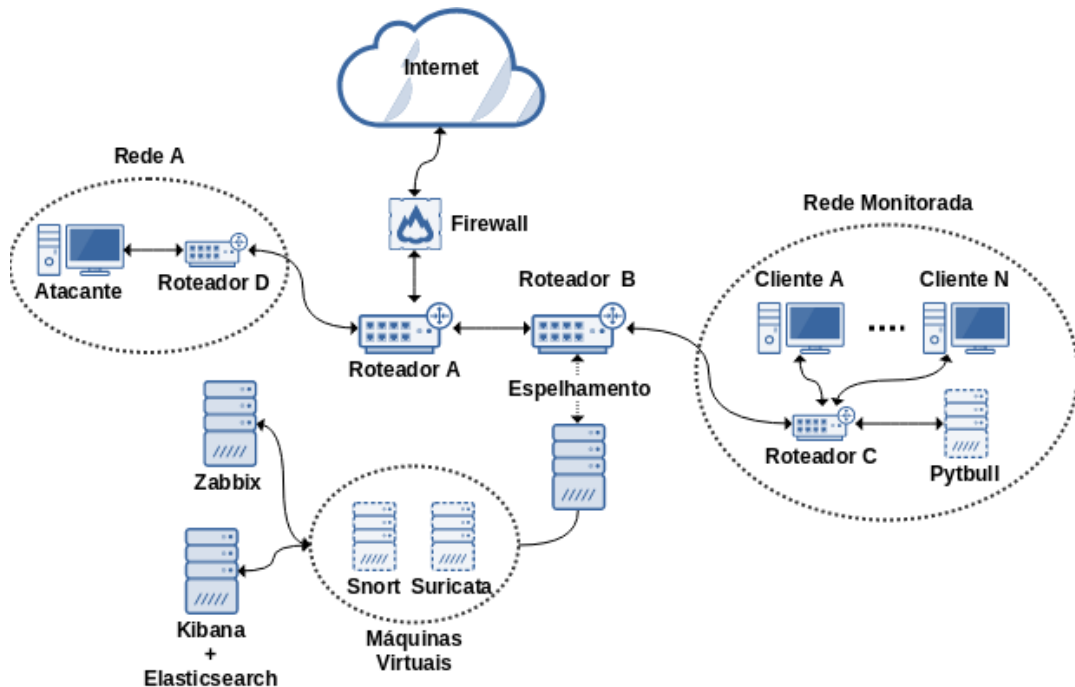
Foi alocado 8 GB memória RAM, 4 processadores e 100 Gigabytes(GB) de espaço em disco para o *snapshot*. Esses valores foram definidos com base em um estudo ([LOCOCO](#), 2011) que considera vários fatores, como largura da rede, localização do IDS e versão, tipo do capturador de tráfego e tamanho da base de assinaturas para dimensionar os recursos de memória e processamento, aplicado especificamente ao Snort. A mesma regra foi aplicada ao Suricata.

Para o *host* obter os pacotes destinados a rede escolhida para o experimento, foi necessário uma configuração de espelhamento no roteador B ([Figura 20](#)) que consiste na copia dos pacotes que saem pela porta dessa rede no roteador para a porta conectada no *host* que possui uma largura de banda de 10 Gigabits. A interface de rede do *host* precisou ser configurada no modo *promisc*.

Posteriormente criou-se três máquinas virtuais, duas usadas para instalação dos IDSs (Suricata e Snort) e a terceira será usada para rodar o *framework* Pytbull. Numa quarta máquina, instalou-se o sistema Kali Linux ([KALI](#), 2017) para geração de ataques, esse SO possui ferramentas nativas para testes de penetração e auditoria de segurança

(Metasploit, NMAP e OpenVAS). A infraestrutura final do ambiente de teste poder ser visualizada na [Figura 20](#).

Figura 20 – Infraestrutura do ambiente de teste



Fonte: Autoria própria

Para coleta das informações de uso de recurso de hardware como memória, processamento e I/O das máquinas com os IDSs utilizou-se o *daemon* Collectd ([COLLECTD](#), 2017). Outra opção para essa finalidade é a utilização de um servidor de monitoramento como o Zabbix ([ZABBIX](#), 2017). A ideia de ter duas ferramentas para essa análise é fazer um comparativo e validar as informações coletadas.

O formato usado para facilitar a análise do *logs* foi JavaScript Object Notation (JSON), um formato simples, leve e de fácil leitura. O Motor de Saída do Suricata já tem suporte a esse tipo de formato o que não acontece no Snort. Para tal, usou-se o IDSTools (ISH, 2017), uma coleção de bibliotecas na linguagem python que trabalha para auxiliar o IDS Snort. Dentre os utilitários presentes nessa coleção, temos o idstools-u2json, que converte, de forma contínua, arquivo no formato unified2, uma das saídas disponível no Snort, para o formato JSON.

Para analisar os *logs*, usou-se uma infraestrutura que combina três ferramentas:

- a) **Kibana** (ELASTIC, 2017b): Uma plataforma de análise e visualização desenhada para trabalhar com os índices do Elasticsearch (ELASTIC, 2017a), a grosso modo, podemos dizer que ela é uma interface gráfica para o Elasticsearch.

- b) **Elasticsearch**: Um motor de busca e análise altamente escalável, capaz de armazenar, buscar e analisar uma grande quantidade de dados em tempo próximo ao real.
- c) **Logstash** ([ELASTIC, 2017c](#)): Um motor de coleta de dados em tempo real, unificando os dados de diferentes fontes dinamicamente, normalizando-os nos destinos escolhidos.

Dessa forma centralizou-se os *logs*, facilitando a visualização e análise das ocorrências geradas pelos IDSs.

4.2 Testes Realizados

Os testes realizados são simulações de passos que uma pessoa má-intencionada iria tomar para alguma tentativa de invasão. Entende-se por invasão, qualquer tipo de violação e alteração não autorizada de um serviço ou *host*.

O passo inicial seria um estudo do alvo utilizando várias técnicas mas principalmente a engenharia social, analisando as pessoas que trabalharam na organização, enviando spam e *phishing* na tentativa de capturar dados como senhas de acesso.

Posteriormente, o atacante iria observar o tráfego da rede, verificando os serviços que o alvo oferece, a procura de alguma senha desprotegida (não criptografada). Esse passo inicial não será aplicado nos testes pois seria impossível o IDS detectar.

O passo seguinte seria uma garimpagem de informações e mapeamento da rede, a procura de um *host* vulnerável. A ferramenta escolhida para essa finalidade é o NMAP.

No primeiro teste de *scan*, usou-se o parâmetro '-F', habilitando a modo *fast* do Nmap. Nesse modo, são verificadas apenas as portas especificadas no arquivo nmap-services, que, na instalação padrão, possui 27372 portas descritas. Isso é muito mais rápido que verificar todas as 65535 portas tcp e 65535 portas udp, possíveis em um *host*. Abaixo segue o comando usado.

Comando NMAP no modo *fast*

```
nmap -F 200.239.72.19
```

O segundo teste, usou-se o parâmetro '-A' do NMAP. Essa opção habilita opções adicionais avançadas e agressivas, descobrindo versões dos serviços, determinando protocolos de serviços, o nome da aplicação, o número da versão, o nome do *host* (utilizando o DNS reverso), tipo de dispositivo, sistema operacional, entre outras informações. Ter um número de versão exacto ajuda substancialmente na determinação de quais *exploits* o servidor está vulnerável ([NMAP, 2017](#)).

Comando NMAP no modo de descoberta de versões

```
nmap -A 200.239.72.19
```

Uma forma encontrada para automatizar o processo de execução dos comandos citados acima foi a criação de um código em *shell script*. O *Shell Script* é uma linguagem de script usada por vários sistemas operacional, principalmente os baseados em UNIX. Abaixo segue o código.

Automatização das execuções do comando NMAP

```
#!/bin/bash

echo "\nIniciando ..."
STARTIME=$(date)

for i in $(seq 1 100)
do
    echo "\n$i-Executando ..."
    nmap $1 200.239.72.19 > /dev/null
    sleep 5
done

ENDTIME=$(date)
echo "\nInicio-$STARTIME"
echo "\nFim-$ENDTIME"
```

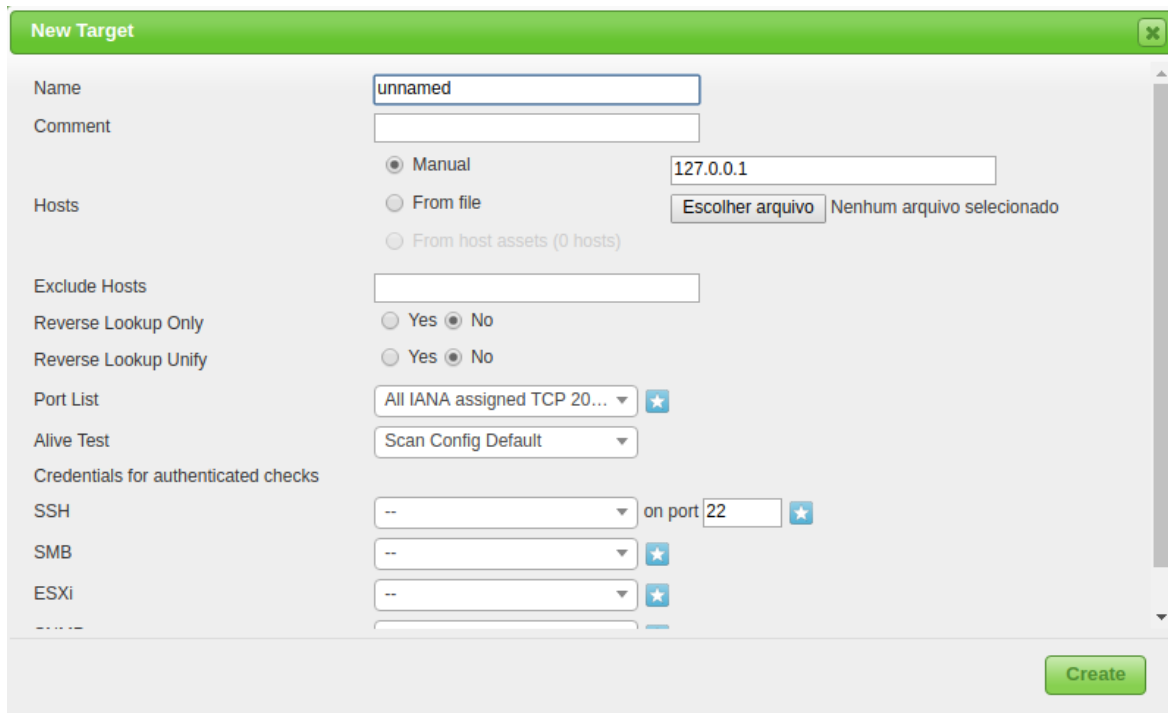
Na primeira linha de um código em *shell script*, precisamos especificar o interpretador utilizado (`#!/bin/bash`). Criou-se duas variáveis, uma para armazenar a hora do início do *script* e uma para armazenar termino, dessa forma, a pesquisa pelos alertas gerados pelas ferramentas será menos dispendiosa. O comando NMAP foi colocado dentro de um laço *for*, executado 100 vezes.

De posse de um alvo em potencial, próximo passo seria executar um *scanner* de vulnerabilidade, em busca de brechas já conhecidas, e que, geralmente por descuido do administrador, não foi fechada. Essas brechas podem ter várias origens, desde uma versão do serviço com *bugs* ou uma má configuração. Para esse teste, usou-se o OpenVAS.

É necessário algumas configurações no *scanner* nessa etapa. Primeiramente, deve-se definir o alvo, tal configuração é feita através do caminho "*Configuration > New target*", a [Figura 21](#) mostra a janela aberta, nela temos que definir um nome para o alvo e o IP ou a faixa de IP's, as outras configurações serão deixadas com o padrão.

A realização do *scan* é feita através do caminho "*Scans > Tasks > New Task*", na

Figura 21 – Definição do alvo no OpenVAS



The screenshot shows the 'New Target' window in OpenVAS. The 'Name' field is set to 'unnamed'. The 'Hosts' section has 'Manual' selected, with the IP '127.0.0.1' entered in the adjacent field. Other options like 'From file' and 'From host assets' are unselected. The 'Exclude Hosts' field is empty. 'Reverse Lookup Only' and 'Reverse Lookup Unify' are both set to 'No'. The 'Port List' is set to 'All IANA assigned TCP 20...'. The 'Alive Test' is set to 'Scan Config Default'. Under 'Credentials for authenticated checks', 'SSH' is set to '--' on port '22'. 'SMB' and 'ESXi' are also set to '--'. A 'Create' button is at the bottom right.

Fonte: Autoria própria

Figura 22 mostra a janela aberta, nela precisamos definir no nome da *task* e o alvo, que foi definido anteriormente, as outras opções serão deixadas com o padrão.

Por padrão, quando uma *task* é criada, o *scan* é automaticamente executado. Esse processo pode demorar, isso depende de quantos alvos foram definidos. Ao final, um relatório é exibido com as vulnerabilidades encontradas e categorizadas (*high*, *medium*, *low*) de acordo com a sua severidade. Além disso, o OpenVAS exibe um sumário, descrevendo a(s) falha(s) encontrada(s) e como resolver ou mitigar o problema.

Outro teste realizado determinará a capacidade das ferramentas de detectar ataques de negação de serviço. Existem várias técnicas para realizar esse tipo de ataque, o usado nesse trabalho foi o TCP SYN FLOOD.

Quando um cliente tenta começar uma conexão TCP com um servidor, são trocados, entre eles, uma série de mensagens (*Three-Way Handshaker*). O TCP SYN FLOOD consiste em enviar uma sequência de requisições SYN para o alvo, sobrecarregando-o diretamente na camada de transporte e indiretamente na camada de aplicação.

Para tal, usou-se o Metasploit Framework. Primeiramente, entrou-se no console de linha de comando, para acessar, basta, no terminal, digitar *msfconsole*. Existe um módulo no Metasploit chamado *synflood*, para utilizar esse módulo, é necessário definir o IP do alvo e o IP de onde partirá o ataque. Abaixo segue os comandos usados no *msfconsole*.

Figura 22 – Configuração de um *task* de *scan* no OpenVAS

The screenshot shows the 'New Task' configuration window in OpenVAS. The form includes fields for Name, Comment, Scan Targets, Alerts, Schedule, Add results to Assets, Apply Overrides, Min QoD, Alterable Task, Auto Delete Reports, and Scanner. The 'Create' button is located at the bottom right.

Fonte: Autoria própria

Comando usados no Metasploit para ataque de negação de serviço

```
use auxiliary/dos/tcp/synflood

msf auxiliary(synflood) > set rhost 200.239.72.19 (target IP)

msf auxiliary(synflood) > set shost 10.15.10.20 (attack IP)

msf auxiliary(synflood) > exploit
```

No ultimo teste, utilizou-se o *framework* Pytbull. O Pytbull foi desenvolvido especificamente para realizar testes e validar configurações em IDPS. A documentação oficial exemplifica algumas arquiteturas que podem ser usadas, nesse trabalho usou-se a arquitetura *remote mode* onde o IDS é colocado para escutar todo o tráfego que passa no *switch* através de uma porta espelhada e com uma interface configurada em modo *promisc*.

Por padrão, todos os testes do *framework* estão ativados. Como o teste de DoS foi realizado utilizando o Metasploit, e para diminuir o tempo da experimentação, o módulo *denialOfService* do Pytbull foi desativado. Outro módulo desativado, devido demora na execução, foi o *testRules*, a execução desse módulo levava mais de uma hora. Para desativar esses módulos precisa-se editar o arquivo **pytbull/config/config.cfg**, alterando, na parte dos testes, os valores de 1 para 0.

Outra mudança feita, somente nessa fase, com o intuito de melhorar a eficácia do teste, foi a configuração das ferramentas de IDPS para analisar somente pacotes com o IP do *host* que executará o Pytbull.

Antes de executar o *framework*, é necessário mover, para os *hosts* que estão executando as ferramentas de IDPS, o script **pytbull/server/pytbull-server.py** encontrado no arquivo `.tar.bz2`, baixado no site oficial do Pytbull ([DAMAYE, 2016](#)). Esse executável abre um shell reverso no servidor que é usado pelo módulo `clientSideAttacks` do Pytbull, fazendo-o baixar arquivos maliciosos da Internet.

Feito isso, basta executar os comandos abaixo, substituindo **suricata** e **snort** pelos seus respectivos IPs:

Comandos usados para execução do Pytbull

```
cd pytbull/  
  
sudo ./pytbull -t suricata  
  
sudo ./pytbull -t snort
```

4.3 Resultados

O Snort e Suricata são Sistemas de Detecção de Intrusão baseados em assinaturas, ambos gratuitos e multiplataformas. No entanto, caso seja necessário uma base de assinaturas robusta e com suporte, o mesmo deve ser adquirido mediante a um pagamento.

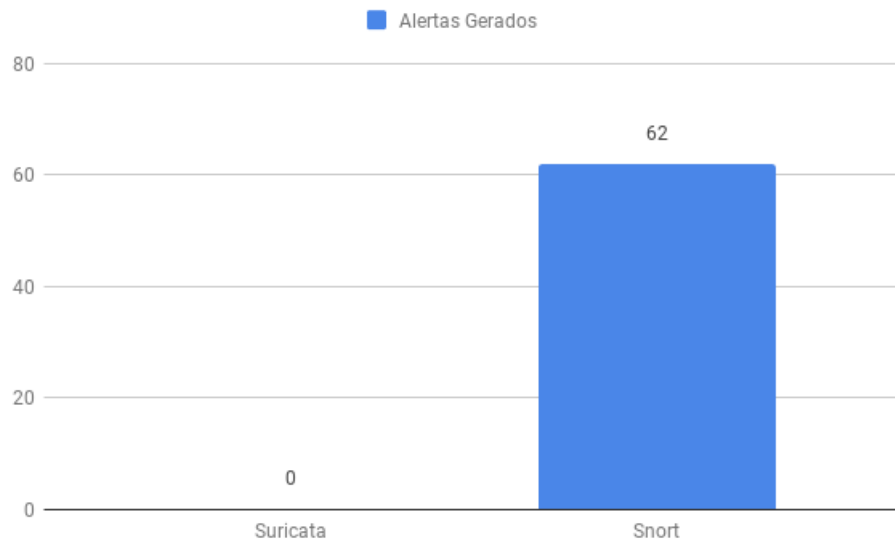
Nada impede o desenvolvimento de uma base própria, porém, é necessário uma equipe altamente capacitada e dedicada para esse fim. Essa equipe é necessária pois, diariamente, são lançados na rede novos ataques ou variações de ataques existentes, tornando a atualização da base de assinaturas de fundamental importância além de ser um processo árduo e custoso.

O Snort encontra-se consolidado no mercado há vários anos e com muitas versões, já o Suricata surgiu em 2010 como um novo motor, com a tecnologia `multithreading` (processa várias tarefas ao mesmo tempo), aproveitando o paradigma multinúcleos dos processadores. Por esse motivo, espera-se que o desempenho do Suricata seja superior ao do Snort.

As métricas usadas para análise e comparação das ferramentas são a utilização dos recursos de *hardware* (memória, processamento), taxa de detecção, independente de ser falso positivo e falso negativo. No primeiro momento, havia a ideia de comparar utilizando essa métrica, no entanto, devido a uma quantidade muito grande de alertas e devido a rede ser grande (vários clientes), essa comparação tornou-se inviável.

Os testes foram realizados no período de 2 de abril de 2018 à 27 de abril de 2018. No primeiro experimento realizado, utilizando o NMAP no modo *fast*, a execução do *script*, gerou um total de 62 alertas, dos quais 100% deles foram do Snort (Figura 23). O Snort teve um desempenho melhor pois ele foi capaz de detectar a execução do comando.

Figura 23 – Resultados do teste executando o NMAP no modo *fast*



Fonte: Autoria Própria

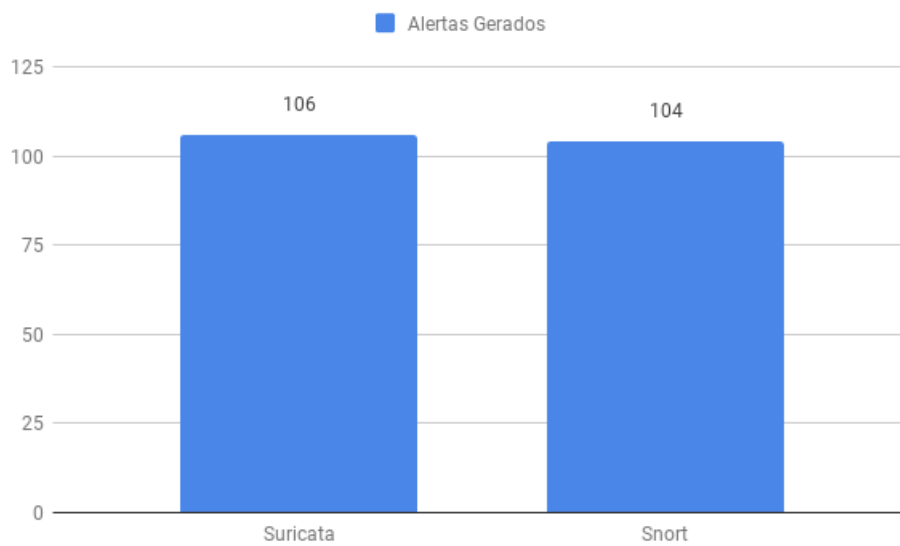
No segundo teste de varredura, utilizando o NMAP no modo no qual é possível identificar as versões dos serviços e até o SO do alvo, as ferramentas geraram, com a execução do *script*, um total de 210 alertas, desses, 106 (50.5%) foi gerado pelo Suricata e 104 (49.5%) pelo Snort (Figura 24). Nesse cenário, houve quase uma equidade na taxa de detecção. No geral, nesse experimento com NMAP, o Snort teve um desempenho melhor, pois conseguiu gerar alertas no primeiro teste.

Já no teste realizado usando um *scan* de vulnerabilidade, os IDPS's geraram um total de 3071 alertas, sendo que 2142 (69.74%) veio do Suricata e 929 (30.26%) do Snort (Figura 25). Aqui, houve uma disparidade grande na taxa de detecção, assim, nesse cenário, o Suricata apresentou um desempenho melhor.

Para determinar se os IDPS's tinham capacidade de identificar ataques de negação de serviço utilizou-se um módulo do *Metasploit Framework*. Nesse teste, nenhuma uma das ferramentas alertou sobre a execução desse tipo de ataque.

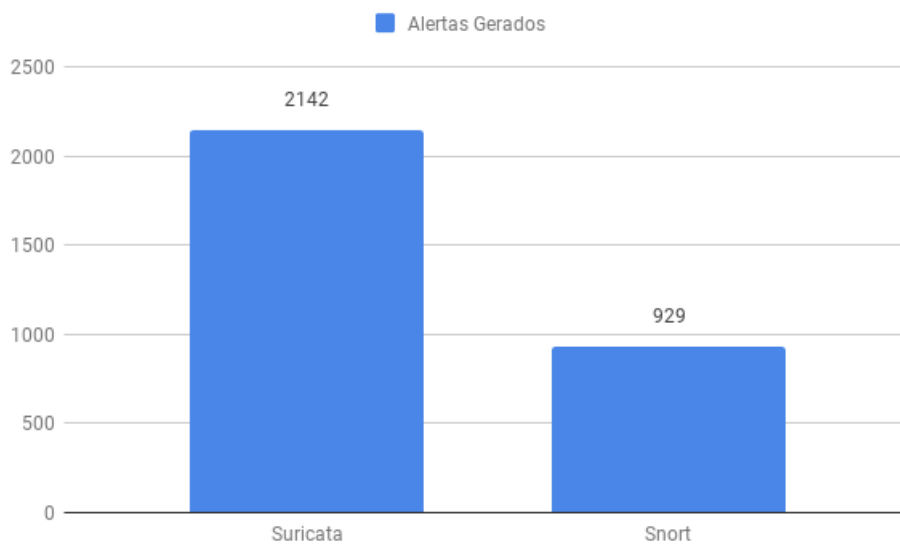
Por último, temos o teste utilizando *framework* Pytbull. Para melhorar os resultados, configurou-se os IDPS's para apenas gerar alertas de eventos relacionados ao *host* alvo. O resultado mostrou que houve uma igualdade na taxa de detecção total, no entanto, o Suricata mostrou-se superior pois a quantidade não detectado do Snort foi maior, 81%

Figura 24 – Resultados do teste executando o NMAP no modo de detecção do SO e versões dos serviços



Fonte: Autoria Própria

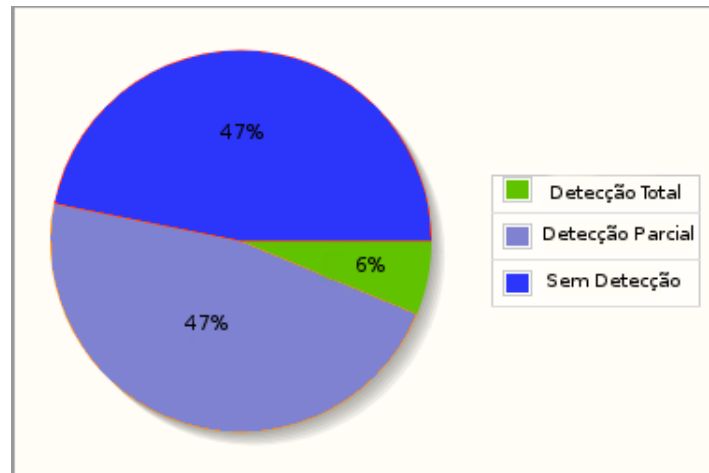
Figura 25 – Resultados do teste executando um o *scan* de vulnerabilidades



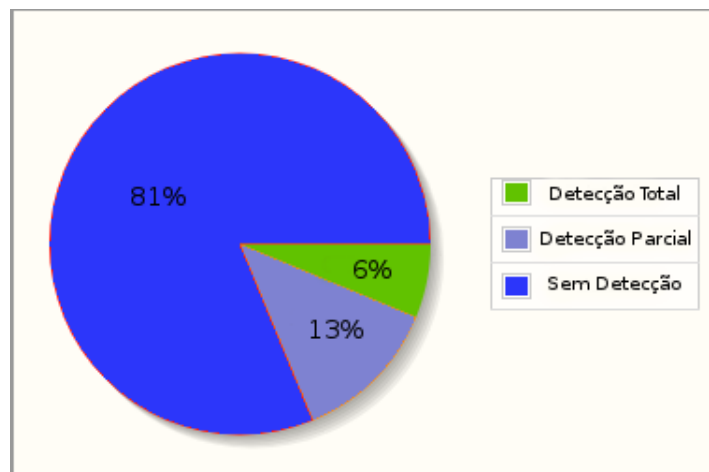
Fonte: Autoria Própria

contra 47% do Suricata. Abaixo segue os resultados, o gráfico é fornecido pelo próprio Pytbull ao final da execução.

Na [Tabela 3](#) está presente os dados coletados pelo serviço de monitoramento Zabbix do Suricata no decorrer de um mês. Foram coletados quatro informações, a carga do processador por núcleo nos tempos de um, cinco e quinze minutos e o uso de memória. Na

Figura 26 – Resultado da execução do *framework* Pytbull sobre o Suricata

Fonte: Pytbull

Figura 27 – Resultado da execução do *framework* Pytbull sobre o Snort

Fonte: Pytbull

Tabela 4, há as mesmas informações, no entanto, do Snort.

Quanto a utilização de recurso de *hardware*, conclui-se que o Snort utiliza um quantidade menor de memória, no entanto, possui um média de carga de processamento maior. Tal característica é válida devido a arquitetura do Snort usar apenas uma *thread* por processo. O fato do processamento mínimo do Suricata ser zero, dar-se a um comportamento da ferramenta, no qual o mesmo era finalizado de forma inesperada.

Quanto a taxa de detecção, analisou-se os registros de alertas gerados no Kibana, selecionando um período de uma semana. Houve um total de 47.778 de alertas gerados, desses 30.097 (62.99%) vieram do Suricata e 17.681 (37.01%) do Snort (Figura 28). Assim, o Suricata teve um desempenho melhor neste cenário. Vale salientar que, não esta sendo levado em consideração se o alerta é um falso negativo ou falso positivo.

Tabela 3 – Resultado do uso de recurso de *hardware* do Suricata

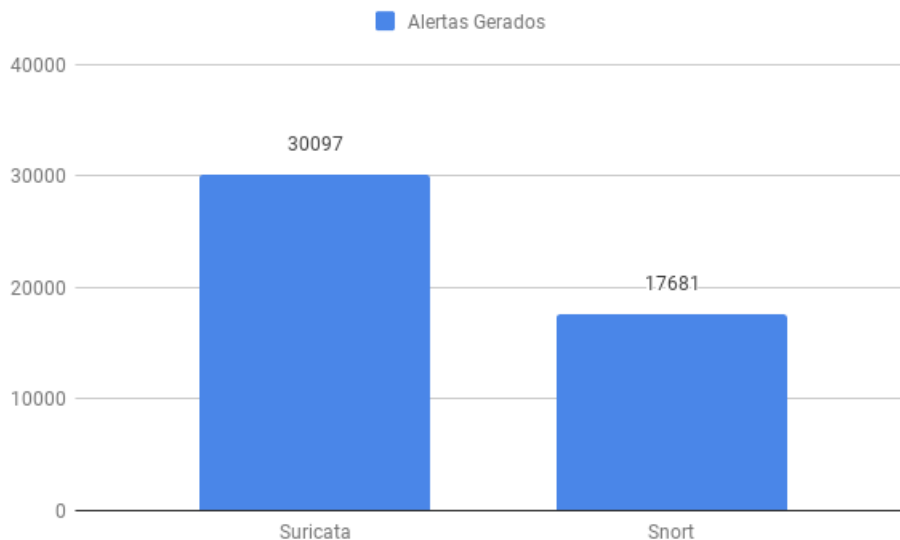
	mínimo	média	máximo
Carga do Processador (1 min) por core	0	0.0326	0.545
Carga do Processador (5 min) por core	0	0.0326	0.545
Carga do Processador (15 min) por core	0	0.0326	0.545
Uso de Memória	3.16 GB	5.48 GB	7.78 GB

Fonte: Zabbix

Tabela 4 – Resultado do uso de recurso de *hardware* do Snort

	mínimo	média	máximo
Carga do Processador (1 min) por core	0	0.1798	0.4625
Carga do Processador (5 min) por core	0.0075	0.1788	0.3225
Carga do Processador (15 min) por core	0.0125	0.1774	0.2825
Uso de Memória	2.09 GB	3.81 GB	5.06 GB

Fonte: Zabbix

Figura 28 – Resultados do teste executando um o *scan* de vulnerabilidades

Fonte: Autoria Própria

4.4 Conclusão

Esse capítulo apresentou o cenário no qual as ferramentas estudadas foram colocadas para análise. Descreveu-se a infraestrutura montada e os componentes envolvidos para a realização do estudo. Além disso, foi detalhado os testes realizados, simulações de ataques a uma máquina alvo dentro da rede e ao final os resultados obtidos.

5 Considerações Finais e Trabalhos Futuros

Foram encontrados vários desafios durante o desenvolvimento desse trabalho. Dentre elas, destacam-se a dificuldade de encontrar material e experimentos referente ao Suricata, a documentação oficial do sistema é pobre de conteúdo, limitando-se ao básico de instalação e algumas particularidades de configuração.

Outra dificuldade que se destaca foi com relação a configuração do cenário de teste, devido a alta complexidade e por requerer configurações em vários equipamento, no qual, em alguns casos, a gerência pertencia a terceiros. Depois de um tempo, observou-se que nem todos os pacotes da interface espelhada estava passando para as VM's, isso afetaria os resultados dos testes. Após análise, concluiu-se que era necessário uma configuração de espelhamento no *openvswitch* (utilizado pelo XenServer) do *host*.

Diante dos resultados obtidos na [seção 4.3](#), conclui-se que, apesar da ferramenta Suricata (versão 4.0.0) ter, em aspectos gerais, um desempenho melhor, justificando assim, que a arquitetura nela implementada (utilizando *multithread*) ajuda na melhoria da detecção e geração de alertas.

No entanto, recomenda-se a utilização do Snort (versão 2.9.8.3) em um ambiente de produção real. Tal afirmação, é sustentada devido a um comportamento do Suricata observado durante o desenvolvimento desse trabalho, que, por algum motivo não investigado, era finalizado. Dessa forma, os alertas não eram gerados, deixando a rede um pouco desprotegida.

Apesar disso, os dados coletados do Suricata, correspondem a um período no qual a ferramenta estaria funcionando adequadamente, sem prejudicar assim, os resultados obtidos no trabalho. Assim, caso opte-se por implantar o Suricata em um ambiente de produção, será necessário um esforço maior, pois a equipe de segurança, deve está sempre monitorando se o processo da ferramenta está executando e gerando os alertas corretamente.

A realização desse trabalho foi de grande valor para aquisição de conhecimento na área de segurança de redes. Concluindo-se que, embora haja na rede um IDS, somente a implementação deste não é suficiente para se ter uma segurança total, e sim, que ele deva atuar em paralelo com outros sistemas de segurança, vindo assim, a somar na dura batalha pela obtenção de segurança em redes.

Como trabalhos futuros, pode-se avaliar as ferramentas em um ambiente mais controlado para tentar identificar se os alertas gerados são realmente de ataques lançado à rede. Assim, determinando a quantidade de falso negativos e falsos positivos e a precisão

das ferramentas na geração de alertas.

Referências

- ARYA, Y. et al. A study of metasploit tool. *International Journal Of Engineering Sciences & Research Technology*, 2016. Citado na página 36.
- AVGERINOS, T. et al. Automatic exploit generation. *Communications of the ACM*, 2014. Citado na página 30.
- BAKER, A. R.; CASWELL, B.; BEALE, J. *Snort IDS and IPS Toolkit*. 1. ed. [S.l.]: Syngress, 2007. Citado 3 vezes nas páginas 47, 48 e 50.
- BASSO, T. Uma abordagem para avaliação da eficácia de scanners de vulnerabilidades em aplicação web. Faculdade de Engenharia Eletrica e de Computação - UNICAMP, 2010. Citado na página 29.
- BROWN, T.; GALITZ, G. O farejador de vulnerabilidades openvas. *Linux Magazine* 67, 2010. Citado na página 40.
- CERON, J. *Tratamento de Incidentes de Segurança*. 2. ed. [S.l.]: Rede Nacional de Ensino e Pesquisa - RNP, 2015. 46 p. Citado na página 32.
- CLARO, J. R. Sistemas ids e ips: Estudo e aplicação de ferramenta *OPEN SOURCE* em ambiente linux. Instituto Federal de Educação, Ciência e Tecnologia Sul-Rio-Grandense, 2015. Citado na página 25.
- COELHO, F. E. S.; ARAUJO, L. G. S. de; BEZERRA, E. K. Gestão da segurança da informação - nbr 27001 e nbr 27002. *Escola Superior de Redes - RNP*, 2014. Citado na página 23.
- COLLECTD. *Collectd – The system statistics collection daemon*. 2017. Disponível em: <<https://collectd.org/>>. Acesso em: 12 jul. 2017. Citado na página 55.
- DAMAYE, S. *Oficial Documetation*. 2016. Disponível em: <<http://pytbull.sourceforge.net/index.php?page=documentation>>. Acesso em: 02 ago. 2017. Citado 4 vezes nas páginas 37, 38, 39 e 60.
- DEBIAN. *Afinal de contas, o que é o Debian?* 2017. Disponível em: <<https://www.debian.org/intro/about>>. Acesso em: 12 jul. 2017. Citado na página 54.
- ELASTIC. *Elasticsearch Reference*. 2017. Disponível em: <<https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html>>. Acesso em: 12 jul. 2017. Citado na página 55.
- ELASTIC. *Kibana User Guide*. 2017. Disponível em: <<https://www.elastic.co/guide/en/kibana/current/introduction.html>>. Acesso em: 12 jul. 2017. Citado na página 55.
- ELASTIC. *Logstash Reference*. 2017. Disponível em: <<https://www.elastic.co/guide/en/logstash/current/introduction.html>>. Acesso em: 12 jul. 2017. Citado na página 56.
- ENIS, M. Ransomware hits govt., libraries. *Library Journal*, Maio. 2017. Acesso em: 19 out. 2017. Citado na página 35.

- INTERNET, C. G. da. *Ataques na Internet*. 2017. Disponível em: <<https://cartilha.cert.br/ataques/>>. Acesso em: 21 jul. 2017. Citado na página 31.
- INTERNET, C. G. da. *Codigos maliciosos Malware*. 2017. Disponível em: <<https://cartilha.cert.br/malware/>>. Acesso em: 12 jul. 2017. Citado 2 vezes nas páginas 34 e 35.
- INTERNET, C. G. da. *Estatisticas dos Incidentes Reportados ao CERT.br*. 2017. Disponível em: <<https://www.cert.br/stats/incidentes/>>. Acesso em: 30 nov. 2017. Citado na página 17.
- INTERNET, C. G. da. *Incidentes Reportados ao CERT.br - Tipos de Ataques*. 2017. Disponível em: <<https://www.cert.br/stats/incidentes/2016-jan-dec/tipos-ataque.html>>. Acesso em: 02 ago. 2017. Citado na página 17.
- ISH, J. *py-idstools*. 2017. Disponível em: <<https://github.com/jasonish/py-idstools>>. Acesso em: 12 jul. 2017. Citado na página 55.
- JAIN, J.; PAL, P. R. Detecting worms based on data mining classification technique. IJESC, 2017. Citado na página 34.
- JUSTO, J. E. da S.; TAMARIZ, A. del R. Modelo de agente racional para auxiliar na gestão de serviços em redes de computadores. Universidade Federal do Norte Fluminense, 2012. Citado na página 26.
- KALI. *What is Kali Linux ?* 2017. Disponível em: <<http://docs.kali.org/introduction/what-is-kali-linux>>. Acesso em: 12 jul. 2017. Citado na página 54.
- KUROSE, J. F.; ROSS, K. W. *Redes de Computadores e a Internet: uma abordagem top-down*. 7. ed. [S.l.]: Pearson Education do Brasil Ltda, 2013. Citado na página 33.
- LINTOTT, D. *OpenXenManager introduction*. 2017. Disponível em: <<https://github.com/OpenXenManager/openxenmanager>>. Acesso em: 12 jul. 2017. Citado na página 53.
- LOCOCO, M. *Capacity Planning for Snort IDS*. 2011. Disponível em: <<http://mikelococo.com/2011/08/snort-capacity-planning/>>. Acesso em: 12 jul. 2017. Citado na página 54.
- LOPEZ, M. E. A. Um arquitetura de detecção e prevenção de intrusão para redes definidas por software. Programa de Engenharia Eletrica - COPPE - UFRJ, 2014. Citado 3 vezes nas páginas 47, 50 e 51.
- MARTINELO, C. A. G.; BELLEZI, M. A. Analise de vulnerabilidades com openvas e nessus. Universidade Federal de São Carlos - UFSCar, 2014. Citado na página 18.
- MAYNOR, D. et al. *Metasploit Toolkit for penetration testing, exploit development, and vulnerability research*. 1. ed. [S.l.]: Syngress, 2007. Citado na página 37.
- MCCLURE, S.; SCAMBRAY, J.; KURTZ, G. *Hackers Expostos Segredos e Soluções para a Segurança de Redes*. 7. ed. [S.l.]: Bookman Editora LTDA, 2014. Citado na página 30.
- MUKHOPADHYAY, I.; CHAKRABORTY, M.; CHAKRABARTI, S. A comparative study of related technologies of intrusion detection & prevention systems. JOURNAL OF INFORMATION SECURITY, 2011. Citado na página 43.

- MURINI, C. T. Análise dos sistemas de detecção de intrusão em redes: Snort e Suricata comparando com dados da DARPA. Universidade Federal de Santa Maria, 2014. Citado 2 vezes nas páginas 50 e 52.
- NAGAHAMA, F. Y. Ipsflow: Um framework para sistema de prevenção de intrusão baseado em redes definidas por software. 2013. Citado 3 vezes nas páginas 43, 44 e 46.
- NMAP. *Nmap Manual*. 2017. Disponível em: <<https://nmap.org/>>. Acesso em: 12 jul. 2017. Citado 2 vezes nas páginas 35 e 56.
- NUNAN, A. E. Detecção de cross-site scripting em páginas web. Instituto de Computação - UFAM, 2012. Citado na página 33.
- NUNES, C. H. F. Exploit e ferramentas para sua utilização. FATEC OURINHOS, 2011. Citado na página 30.
- OISF. *About*. 2017. Disponível em: <<https://suricata-ids.org/about/>>. Acesso em: 20 dez. 2017. Citado 2 vezes nas páginas 50 e 51.
- OPENVAS. *About OpenVAS*. 2017. Disponível em: <<http://www.openvas.org/about.html>>. Acesso em: 26 dez. 2017. Citado 2 vezes nas páginas 40 e 41.
- PEIXINHO, I. de C.; FONSECA, F. M. da; LIMA, F. M. M. Segurança de redes e sistemas. *Escola Superior de Redes - RNP*, 2013. Citado na página 24.
- PROOFPOINT. *About Proofpoint*. 2017. Disponível em: <<https://www.proofpoint.com/us/company/about>>. Acesso em: 22 dez. 2017. Citado na página 51.
- ROESCH, M.; GREEN, C. *Snort Users Manual*. 2017. Disponível em: <<http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node2.html>>. Acesso em: 29 set. 2017. Citado 2 vezes nas páginas 46 e 47.
- S, S.; S, S.; M, R. Review on sql injection attacks: Detection techniques and protection mechanisms. *International Journal of Computer Science and Information Technologies*, 2014. Acesso em: 26 out. 2017. Citado na página 28.
- SCARFONE, K.; MELL, P. Guide to intrusion detection and prevention systems *idps*. National Institute Of Standards and Technology, 2007. Citado na página 44.
- SCHARDONG, F.; ÁVILA, R. Interface de apoio para ataques de força bruta com o gpu md5 crack. ERAD, 2012. Citado na página 31.
- SEGURANÇA, C. de Atendimento a Incidentes de. *Segurança*. 2017. Disponível em: <<https://www.rnp.br/servicos/seguranca>>. Acesso em: 12 jul. 2017. Citado na página 18.
- SILVA, D. R. P. da; STEIN, L. M. Segurança da informação: uma reflexão sobre componentes humanos. *Ciência e Cognição*, 2007. Citado na página 31.
- STALLINGS, W. *Cryptography and Network Security: Principles and Practice*. [S.l.]: Prentice Hall, 2011. Citado na página 33.
- STALLINGS, W.; BROWN, L. *Segurança de Computadores: Princípios e Práticas*. 2. ed. [S.l.]: Elsevier Editora LTDA, 2014. Citado na página 32.

- TALOS. *Talos*. 2017. Disponível em: <<https://www.snort.org/talos>>. Acesso em: 22 dez. 2017. Citado na página 51.
- TANENBAUM, A.; WETHERALL, D. *Redes de Computadores*. 5. ed. [S.l.]: Editora Pearson, 2011. Citado na página 26.
- TÉNICAS, A. B. de N. Nbr iso/iec 27002:2013. *ABNT*, 2013. Citado na página 24.
- TILBORG, H. C. A. van; JAJODIA, S. *Encyclopedia of Cryptography and Security*. [S.l.]: Springer Science+Business Media, 2011. Citado na página 34.
- ULBRICH, H. C.; VALLE, J. D. *Universidade Hacker*. 5. ed. [S.l.]: Digerati Books, 2007. Citado 5 vezes nas páginas 26, 27, 29, 30 e 31.
- UTO, N. *Teste de Invasão de Aplicações Web*. 1. ed. [S.l.]: Rede Nacional de Ensino e Pesquisa - RNP, 2013. 179 p. Citado 2 vezes nas páginas 28 e 29.
- XENSERVR. *About Xenserver*. 2017. Disponível em: <<https://xenserver.org/about-xenserver-open-source.html>>. Acesso em: 12 jul. 2017. Citado na página 53.
- ZABBIX. *What is Zabbix*. 2017. Disponível em: <<https://www.zabbix.com/product>>. Acesso em: 12 jul. 2017. Citado 2 vezes nas páginas 54 e 55.
- ZARGAR, S. T.; JOSHI, J.; TIPPER, D. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE*, 2013. Citado na página 33.
- ZONE-H, E. *Estatísticas*. 2017. Disponível em: <<http://www.zone-h.com.br/stats>>. Acesso em: 22 nov. 2017. Citado na página 32.