

Glenon Mateus Barbosa Araújo

Análise de IDPSs

Brasil

2017

Glenon Mateus Barbosa Araújo

Análise de IDPSs

Trabalho de Conclusão de Curso submetida
a graduação em Ciência da Computação da
UFPA

Universidade Federal do Pará – UFPA

Faculdade de Computação

Bacharelado em Ciência da Computação

Orientador: Dr. Roberto Samarone dos Santos Araújo

Brasil

2017

fichacatalografica

Glenon Mateus Barbosa Araújo Análise de IDPSs/ Glenon Mateus Barbosa Araújo.
– Brasil, 2017- 41 p. : il. (algumas color.) ; 30 cm.
Orientador: Dr. Roberto Samarone dos Santos Araújo
Trabalho de Conclusão de Curso – Universidade Federal do Pará – UFPA
Faculdade de Computação
Bacharelado em Ciência da Computação, 2017.
1. Suricata. 2. Snort. 3. IDPS. I. Orientador. II. Universidade Federal do Pará. III.
Faculdade de Computação. IV. Análise de IDPSs

Errata

Elemento opcional da ??, 4.2.1.2). Exemplo: FERRIGNO, C. R. A. **Tratamento de neoplasias ósseas apendiculares com reimplantação de enxerto ósseo autólogo auto-clavado associado ao plasma rico em plaquetas**: estudo crítico na cirurgia de preservação de membro em cães. 2011. 128 f. Tese (Livre-Docência) - Faculdade de Medicina Veterinária e Zootecnia, Universidade de São Paulo, São Paulo, 2011.

Folha	Linha	Onde se lê	Leia-se
1	10	auto-conclavo	autoconclavo

Glenon Mateus Barbosa Araújo

Análise de IDPSs

Trabalho de Conclusão de Curso submetida a
graduação em Ciência da Computação da UFPA

Trabalho aprovado. Brasil, 24 de novembro de 2012:

Dr. Roberto Samarone dos Santos Araújo
Orientador

Brasil
2017

•

Agradecimentos

Resumo

Palavras-chave: Segurança, Suricata, Snort, Sistema de Detecção de Intrusão, Sistema de Prevenção de Intrusão, IDS, IPS.

Abstract

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

Lista de ilustrações

Figura 1 – Infraestrutura do ambiente de teste	32
Figura 2 – Busca e união dos dados de diferentes fontes	33
Figura 3 – Exemplo de saída do Nmap	34
Figura 4 – Arquitetura do Metasploit	35
Figura 5 – Arquitetura do <i>framework</i> Pytbull	36

Lista de tabelas

Lista de abreviaturas e siglas

IDS	<i>Intrusion Detection System</i>
IPS	<i>Intrusion Prevention System</i>
IDPS	<i>Intrusion Detection and Prevention System</i>
MB	<i>Megabytes</i>
GB	<i>Gigabytes</i>
SO	<i>Sistema Operacional</i>
JSON	<i>JavaScript Object Notation</i>

Sumário

	Introdução	25
1	SEGURANÇA DE REDES DE COMPUTADORES	27
1.1	Cenário Geral	27
1.2	Ataques	27
1.2.1	Varredura de Redes	27
1.2.2	Exploração de Vulnerabilidades	27
1.2.3	Força Bruta	27
1.2.4	Desfiguração de páginas	27
1.2.5	Negação de Serviços	27
1.2.6	Worm	27
1.2.7	Trojan	27
1.2.8	Fraudes - Direitos Autorais	27
2	SISTEMAS DE DETECÇÃO E PREVENÇÃO DE INTRUSÃO	29
2.1	Tipos de IDS/IPS	29
2.2	Snort	29
2.3	Suricata	29
3	DETECÇÃO DE INTRUSÃO EM UM CENÁRIO REAL	31
3.1	Cenário de Testes	31
3.2	Infraestrutura Definida para Testes	31
3.2.1	Nmap	33
3.2.2	Metasploit Framework	34
3.2.3	Pytbull	35
3.3	Testes Realizados	36
3.4	Resultados	37
3.5	Conclusão	37
3.6	Métricas de Comparação	37
4	CONSIDERAÇÕES FINAIS	39
	REFERÊNCIAS	41

Introdução

Objetivos

Trabalhos Relacionados

Motivação

1 Segurança de Redes de Computadores

1.1 Cenário Geral

1.2 Ataques

1.2.1 Varredura de Redes

1.2.2 Exploração de Vulnerabilidades

1.2.3 Força Bruta

1.2.4 Desfiguração de páginas

1.2.5 Negação de Serviços

1.2.6 Worm

1.2.7 Trojan

1.2.8 Fraudes - Direitos Autorais

2 Sistemas de Detecção e Prevenção de Intrusão

2.1 Tipos de IDS/IPS

2.2 Snort

2.3 Suricata

3 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 3.2 será abordado a infraestrutura usada para os testes, ferramentas utilizadas e as configurações feitas. Na seção 3.3 será descrito os testes realizados com suas respectivas justificativas. Na seção 3.4 será apresentado os resultados esperados e obtidos, problemas encontrados e a comparação das ferramentas e por último, na seção 3.5, uma breve conclusão.

3.1 Cenário de Testes

A rede selecionada para ser monitorada tem os valores especificados na tabela. Podemos verificar que em um determinado período do dia o pico de tráfego chega a 107,25 Mbps, valores considerados ideais para o experimento, inclusive para tentar validar os recursos alocados. Figura ??

Em um primeiro momento, selecionou-se uma rede

3.2 Infraestrutura Definida para Testes

No ambiente de teste foi usado uma máquina Dell com 134 Megabytes (MB) de memória RAM e 40 núcleos. Usou-se 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 instabilidade. Outro fator que pesou na escolha do SO foi a experiência que tinha com a plataforma e por existir uma interface para gerência chamada XenCenter que roda no Windows. Uma alternativa *opensource* desse software é o OpenXenManager ([LINTOTT, 2017](#)).

No primeiro momento, foi instalado uma máquina virtual com o sistema operacional Debian 7.11 *codename* Wheezy ([DEBIAN, 2017](#)), uma distribuição linux com uma proposta de ser totalmente livre, usada 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 os IDSs.

Foi alocado 8 MB 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 considerava 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* conseguir pegar o pacotes destinados a rede escolhida para ser monitorada foi necessário uma configuração de espelhamento no roteador B (Figura 1) 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 para instalação das ferramentas usadas para simular ataques a rede. Optou-se pela instalação do sistema Kali Linux (KALI, 2017) para geração de ataques pois nele existe várias ferramentas nativas para testes de penetração e auditoria de segurança. A infraestrutura final pode ser visualizada na Figura 1.

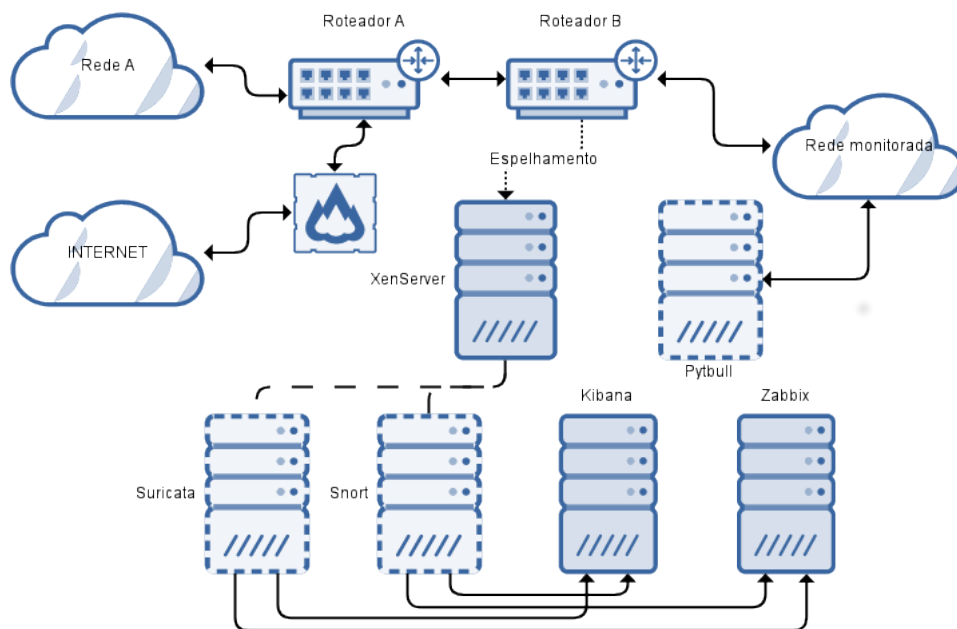


Figura 1 – Infraestrutura do ambiente de teste

Para coleta das informações de uso de recurso de hardware como memória, processamento e I/O das máquinas com os IDSs foi usado o *daemon* Collectd (COLLECTD, 2017). Outra opção para esse fim é a utilização de um servidor de monitoramento com o Zabbix (ZABBIX, 2017). A ideia de usar duas ferramentas para 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 (IDSTOOLS, 2017), uma coleção de bibliotecas na linguagem python que trabalha para auxiliar o IDS, compatível com as ferramentas estudadas. 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, o Kibana (ELASTIC, 2017a), uma plataforma de análise e visualização desenhada para trabalhar com os índices do Elasticsearch (ELASTIC, 2017b), a grosso modo, podemos dizer que ela é uma interface gráfica para o Elasticsearch. O 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. Por último, o Logstash (ELASTIC, 2017c), um motor de coleta de dados em tempo real, unificando os dados de diferentes fontes dinamicamente, normalizando-os nos destinos escolhidos (Figura 2). Dessa forma centralizou-se os *logs*, facilitando a visualização das ocorrências dos IDSs.

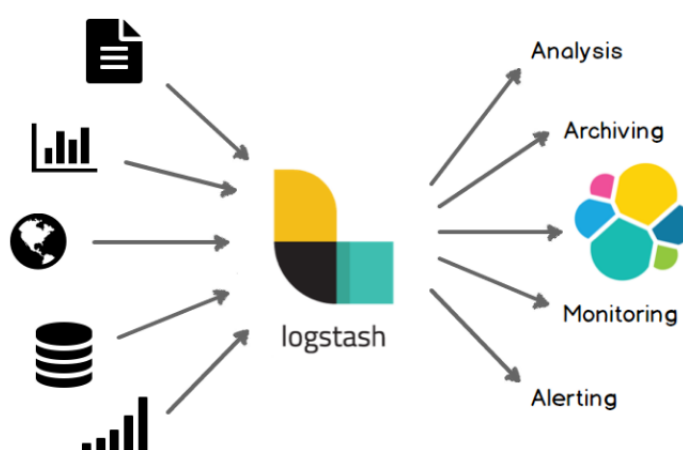


Figura 2 – Busca e união dos dados de diferentes fontes

3.2.1 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 3). As portas podem assumir quatro estados, temos:

- **aberto** (*open*): significa que existe alguma aplicação escutando conexões;
- **filtrado** (*filtered*): há um obstáculo na rede (firewall) que impossibilita que o Nmap determine se a porta está aberta ou fechada;
- **fechado** (*closed*): não possuem uma aplicação escutando nelas;
- **não-filtrado** (*unfiltered*): quando elas respondem porém o Nmap não consegue determinar se estão fechadas ou abertas (NMAP, 2017).

```
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
```

Figura 3 – Exemplo de saída do Nmap

3.2.2 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: **Rex**: É a biblioteca fundamental, a maioria das tarefas executadas pelo *framework* usaram essa biblioteca; **MSF Core**: É o *framework* em si, possui, por exemplo, gerenciador de módulos e a base de dados; **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 arquitetura é mostrada com mais detalhes na figura 4.

Os módulos são divididos da seguinte maneira: **Payload**: São código executados no alvo remotamente; **Exploit**: Explora *bugs* ou vulnerabilidade existente em aplicações do alvo; **Módulos Auxiliares**: Usado para escanear as vulnerabilidades e executar várias tarefas; **Encoder**: Codifica o *payload* para evitar qualquer tipo de detecção pelo anti vírus.

Interface: Disponibiliza a parte gráfica para o usuário;

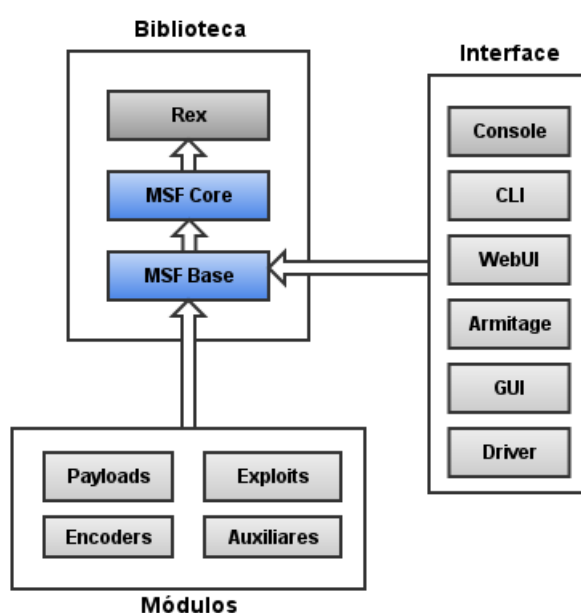


Figura 4 – Arquitetura do Metasploit

3.2.3 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.

O *framework* Pytbull possui cerca de 300 testes agrupados em 11 módulos: **badTraffic**, pacotes não compatíveis com a RFC são enviados para o servidor para testar como os pacotes são processados; **bruteForce**, testa a capacidade do IDPS de rastrear ataques de força bruta; **clientSideAttacks**, usa um *shell* reverso para fornecer ao servidor instruções para baixar arquivos maliciosos; **denialOfService**, testa a capacidade do IDPS de proteger contra tentativas de DoS; **evasionTechniques**, testa a capacidade do IDPS de detectar técnicas de evasão; **fragmentedPackets**, várias cargas úteis fragmentadas são enviadas ao servidor para testar sua capacidade de recomposição e detectar os ataques; **ipReputation**, testa a capacidade do servidor detectar tráfego de servidores com reputação baixa; **normalUsage**, cargas úteis que correspondem a uso normal; **pcapReplay**, permite reproduzir arquivos pcap; **shellCodes**, envia *shellcodes* para o servidor na porta 21/ftp testando a capacidade de detectar/bloquear o mesmo; **testRules**, testa a base de assinaturas configuradas no servidor IDPS.

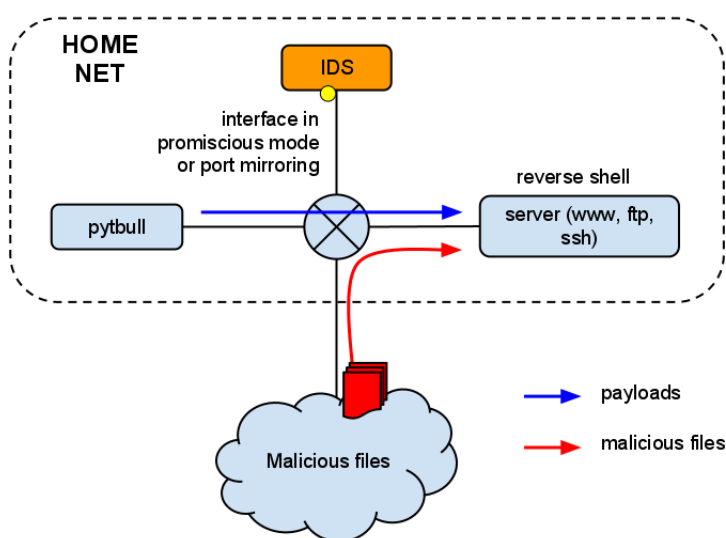


Figura 5 – Arquitetura do *framework* Pytbull

Existe basicamente 5 tipos de testes: **socket**, abre um *socket* em uma porta e envia o *payload* para o alvo remoto na porta especificada; **command**, envia um comando para alvo remoto com a função `python subprocess.call()`; **scapy**, envia cargas úteis específicas baseadas na sintaxe de Scapy; **client side attacks**, usa um *shell* reverso no alvo remoto e envia comandos para serem processados no servidor; **pcap replay**, permite reproduzir tráfego com base em arquivos de pcap.

3.3 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 por engenharia social, analisando as pessoas que trabalharam na organização, enviando spam e phishing na tentativa de capturar dados como senhas de acesso. Posteriormente, verificando os serviços que o alvo oferece e observando (*sniffando*) a rede, 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 um estudo e mapeamento da rede, a procura de um *host* vulnerável. A ferramenta escolhida para essa finalidade é o Nmap 3.2.1. 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`, na instalação padrão esse arquivo vem com 27372 portas descritas. Isso é muito mais rápido que verificar todas as 65535 portas possíveis em um *host*.

```
nmap -F 200.239.82.0/24
```

```
nmap -sV 200.239.82.0/24
```

De posse de um alvo, a pessoa má intencionada irá rodar um scan de vulnerabilidade, em busca de brechas já conhecidas, e que, geralmente por descuido do administrador, não foi fechada. Para esses testes usou-se o *framework* Metasploit [3.2.2](#) nativo do sistema operacional Kali Linux.

3.4 Resultados

3.5 Conclusão

3.6 Métricas de Comparação

4 Considerações Finais

Referências

- ARYA, Y. et al. A study of metasploit tool. *International Journal Of Engineering Sciences & Research Technology*, 2016. Citado na página 34.
- COLLECTD. 2017. Disponível em: <<https://collectd.org/>>. Citado na página 32.
- DEBIAN. 2017. Disponível em: <<http://www.debian.org/>>. Citado na página 31.
- ELASTIC. 2017. Disponível em: <<https://www.elastic.co/guide/en/kibana/current/introduction.html>>. Citado na página 33.
- ELASTIC. 2017. Disponível em: <<https://www.elastic.co/guide/en/elasticsearch/reference/current/getting-started.html>>. Citado na página 33.
- ELASTIC. 2017. Disponível em: <<https://www.elastic.co/guide/en/logstash/current/introduction.html>>. Citado na página 33.
- IDSTOOLS. 2017. Disponível em: <<https://github.com/jasonish/py-idstools>>. Citado na página 32.
- KALI. 2017. Disponível em: <<http://docs.kali.org/introduction/what-is-kali-linux>>. Citado na página 32.
- LINTOTT, D. 2017. Disponível em: <<https://github.com/OpenXenManager/openxenmanager>>. Citado na página 31.
- LOCOCO, M. *Capacity Planning for Snort IDS*. 2011. Disponível em: <<http://mikelococo.com/2011/08/snort-capacity-planning/>>. Citado na página 31.
- NMAP. 2017. Disponível em: <<https://nmap.org/>>. Citado 2 vezes nas páginas 33 e 34.
- XENSERVR. 2017. Disponível em: <<https://xenserver.org/about-xenserver-open-source.html>>. Citado na página 31.
- ZABBIX. 2017. Disponível em: <<http://www.zabbix.com/>>. Citado na página 32.