

LIGA DE ENSINO DO RIO GRANDE DO NORTE
CENTRO UNIVERSITÁRIO DO RIO GRANDE DO NORTE
ESPECIALIZAÇÃO EM REDES DE COMPUTADORES

JORGE FERNANDO BRITO DA SILVA

IMPLEMENTANDO UM SISTEMA DE PREVENÇÃO DE INTRUSÃO USANDO
SURICATA

NATAL/RN

2013

LIGA DE ENSINO DO RIO GRANDE DO NORTE
CENTRO UNIVERSITÁRIO DO RIO GRANDE DO NORTE
ESPECIALIZAÇÃO EM REDES DE COMPUTADORES

JORGE FERNANDO BRITO DA SILVA

IMPLEMENTANDO UM SISTEMA DE PREVENÇÃO DE INTRUSÃO USANDO
SURICATA

NATAL/RN

2013

JORGE FERNANDO BRITO DA SILVA

**IMPLEMENTANDO UM SISTEMA DE PREVENÇÃO DE INTRUSÃO USANDO
SURICATA**

Trabalho de Conclusão de Curso
apresentado ao Centro Universitário do
RN - UNI-RN como requisito final para
obtenção do título de Especialista em
Redes de Computadores.

Orientador: Prof M.Sc.
Aluizio Ferreira da Rocha Neto

NATAL/RN
2013

004.056.52
5586i

UNI-RN BIBLIOTECA
RB: <u>36195</u>
Cód de Barra: <u>MP00053</u>
Data: <u>09/04/13</u>

Catálogo na Publicação – Biblioteca do UNI-RN
Setor de Processos Técnicos

Silva, Jorge Fernando Brito da.
Implementando um sistema de prevenção de intrusão usando suricata /
Jorge Fernando Brito da Silva. – Natal, 2013.
59 f.

Orientador: Aluizio Ferreira da Rocha Neto (Msc.)
Monografia (Pós-Graduação em Redes de Computadores) – Centro
Universitário do Rio Grande do Norte.

- Segurança da Informação – Monografia. 2. Suricata – Monografia.
- 3. Firewall – Monografia. I. Rocha Neto, Aluizio Ferreira da. II. Título.

RN/UNI-RN/BC

CDU 004.056.52

JORGE FERNANDO BRITO DA SILVA

**IMPLEMENTANDO UM SISTEMA DE PREVENÇÃO DE INTRUSÃO USANDO
SURICATA**

Trabalho de conclusão de curso
apresentado ao curso de Especialização
em Redes de Computadores do Centro
Universitário do RN - UNI-RN como
requisito final para a obtenção do título de
Especialista em Redes de Computadores.

Aprovado em: ____/____/____

BANCA EXAMINADORA

Aluizio Ferreira da Rocha Neto, M.Sc.
Orientador

Membro examinador

Membro examinador

RESUMO

Este trabalho aborda a implementação de um Sistema de Prevenção de Intrusão em um cenário fictício. Esta solução baseia-se principalmente no IDS/IPS de código aberto Suricata, uma alternativa promissora ao IDS/IPS Snort. Em um ambiente Debian Wheezy, foram utilizados os mecanismos de aquisição de dados nativos NFQ e AF_PACKET, o Suricata como IPS, o framework de testes de IDS/IPS Pytbull como ferramenta de ataques e o front-end Snorby para a visualização dos resultados. O presente trabalho limita-se a apresentar o Suricata como uma promissora, versátil e moderna ferramenta para detecção e prevenção de intrusão, abordando os conceitos envolvidos, detalhes de configuração e implementação, bem como cenários de ataque.

Palavras-chaves: IPS. Suricata. NFQ. AF_PACKET. Pytbull. Snorby.

ABSTRACT

This work discusses the implementation of a Intrusion Prevention System in a fictional scenario. This solution is based mainly on the open source IDS/IPS Suricata, a promising alternative to the Snort IDS/IPS. In a Debian Wheezy environment, data acquisition was accomplished through the GNU/Linux native mechanisms NFQ and AF_PACKET, Suricata was employed as IPS, the Pytbul IPS/IDS testing framework was used as attacking tool and the front-end Snorby was used for results visualization. The present work limits itself on presenting the Suricata as a modern, promising and versatile intrusion detection and prevention tool, by covering the utilized concepts, configuration and implementation details as well as attack scenarios.

Keywords: IPS. Suricata. NFQ. AF_PACKET. Pytbul. Snorby.

LISTA DE FIGURAS

Figura 1: Arquitetura PF_RING.....	19
Figura 2: Suricata inline funcionando como IPS.....	21
Figura 3: Exemplo de modo de operação IDS.....	22
Figura 4: Exemplo de modo PF_RING.....	23
Figura 5: Diagrama completo da solução.....	33
Figura 6: Cenário utilizando AF_PACKET.....	35
Figura 7: Cenário usando NFQ.....	36
Figura 8: Tela inicial do Snorby.....	40
Figura 9: Pytbull no Backtrack.....	43
Figura 10: Executando Pytbull.....	44
Figura 11: Resultados do Pytbull.....	45
Figura 12: Detalhes de um ataque de SQL Injection visto no Pytbull.....	46
Figura 13: Snorby após ataque.....	47
Figura 14: Detalhe de um ataque de SQL Injection visto no Snorby.....	48
Figura 15: Arquivo fast.log mostrando o bloqueio de um ataque de SQL Injection.....	48
Figura 16: Ataques de shellcodes bloqueados pelo IPS.....	49

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

DHS	Department of Homeland Security
DoS	Denial of Service
FTP	File Transfer Protocol
GPL	General Public License
HIPS	Host Based Intrusion Prevention System
HTTP	Hyper Text-Transfer Protocol
IDMEF	Intrusion Detection Message Exchange Format
IDS	Intrusion Detection System
IDPS	Intrusion Detection and Prevention System
IP	Internet Protocol
IPS	Intrusion Prevention System
LSB	Linux Standard Base
NIPS	Network Based Intrusion Prevention System
OISF	Open Information Security Foundation
PCAP	Packet Capture
PCRE	Perl Compatible Regular Expressions
SMB	Server Message Block
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VRT	Vulnerability Research Team
WIPS	Wireless Based Intrusion Prevention System

SUMÁRIO

1 INTRODUÇÃO	9
1.1 MOTIVAÇÃO	9
1.2 OBJETIVOS	10
1.2.1 Objetivos Específicos	10
2 REFERENCIAL TEÓRICO	12
2.1 IPTABLES	12
2.2 SISTEMA DE PREVENÇÃO DE INTRUSÃO - IPS	12
2.2.1 Vulnerabilidades	12
2.2.2 Métodos de Detecção	13
2.2.2.1 Assinaturas	14
2.2.2.2 Comportamento	14
2.2.2.3 Validação de Protocolos	14
2.2.2.4 Tecnologias de Detecção	14
2.2.3 Quanto ao Tipo de IPS	15
2.2.3.1 Baseado na Estação	15
2.2.3.2 Baseado na Rede	15
2.3 SURICATA	15
2.3.1 Recursos	16
2.3.2 Aquisição de Pacotes	18
2.3.2.1 PF_RING	18
2.3.2.2 AF_PACKET	20
2.3.2.3 NFQ	20
2.3.3 Arquitetura	21
2.3.4 Principais Arquivos	23
2.3.4.1 Classification.config	23
2.3.4.2 Suricata.yaml	24
2.3.4.3 Threshold.config	28
2.3.4.4 – Arquivo de Regras	29
2.4 PYTBULL	30
2.5 SNORBY	31
3 DESENVOLVIMENTO DA SOLUÇÃO	32
3.1 INSTALAÇÃO	33
3.1.1 Cenário com Aquisição via AF_PACKET	34
3.1.2 Cenário com Aquisição via NFQ	34
3.1.2.1 Configurando o Firewall	35
3.1.3 Instalação do front-end	35
3.1.4 Inicialização do Suricata e dependências	40
3.2 ATACANDO COM PYTBULL	40
3.3 RESULTADOS	42
4 CONCLUSÕES E TRABALHOS FUTUROS	48
REFERÊNCIAS BIBLIOGRÁFICAS	49
APÊNDICE A – ARQUIVO SURICATA.YAML	50
APÊNDICE B – SCRIPT DE INICIALIZAÇÃO DO FIREWALL	56
APÊNDICE C – ARQUIVO RULES	58
APÊNDICE D – ARQUIVO OPEN	59
APÊNDICE E – ARQUIVO CLOSE	60
APÊNDICE F – ARQUIVO DATABASE.YML	60

APÊNDICE G – ARQUIVO SNORBY_CONFIG.YML.....	60
APÊNDICE H – ARQUIVO DEFAULT.....	61
APÊNDICE I – ARQUIVO BARNYARD2.CONF.....	62
APÊNDICE J – ARQUIVO IPS.....	62
APÊNDICE K – ARQUIVO CONFIG.CFG.....	63

1 INTRODUÇÃO

As redes estão cada vez maiores, mais complexas e interconectadas através da internet. Toda a facilidade da interconexão mundial proporcionou o aumento da complexidade quanto à administração e gerenciamento destas redes. Além de grandes em número de equipamentos, os dados trafegados e a exigência de disponibilidade também acompanharam o crescimento tornando ainda mais difícil a tarefa de gerenciamento destes ambientes.

Neste cenário veio também a evolução e diversificação das ferramentas de ataques às redes, como vírus, *worms*, *exploits* etc. Do mesmo modo, a proteção eficaz dessas redes se faz com o uso de várias ferramentas sempre aliada a uma política adequada de segurança.

Tradicionalmente, firewalls e programas anti-vírus atuam tentando bloquear ataques, enquanto que um Sistema de Detecção de Intrusão - IDS tenta identificar ataques assim que eles ocorrem. Estas duas tecnologias apesar funcionarem relativamente bem, tem suas limitações.

Um firewall pode, entre outras coisas, bloquear portas ou endereços de Internet (IP), mas tem atuação limitada nas portas que precisam estar liberadas para que determinado serviço funcione. Um IDS pode avaliar melhor o tráfego que passa por estas portas liberadas, porém não pode bloquear o tráfego danoso. Um Sistema de Prevenção de Intrusão – IPS tem o melhor dos dois mundos, pois pode de forma proativa avaliar e bloquear esse tráfego, sendo mais um componente fundamental para o especialista em segurança de redes.

1.1 MOTIVAÇÃO

Com o aumento da demanda por serviços e portanto um crescimento significativo no fluxo de dados nas redes, um ponto extremamente importante em qualquer IPS é o desempenho. Com esse aumento de tráfego são necessárias soluções cada vez mais robustas de hardware e software bem como melhorias no processo de aquisição dos dados.

Existem vários ótimos projetos de segurança na área de detecção e

prevenção da intrusão em software livre tais como o Snort¹, Bro² e o Suricata, entre outros. A opção foi então descrever o Suricata, pois apesar de não ser tão popular, prometia uma solução com design moderno, desenvolvimento bastante ativo e ainda, excelente desempenho sobretudo em processadores multi-cores.

1.2 OBJETIVOS

O objetivo geral do trabalho é implementar uma solução de IPS (Sistema de Prevenção de Intrusão) com Suricata obtendo dados via NFQUEUE, fazendo uso do kernel Linux 3.7.2 em situações que podem ultrapassar a taxa de gigabits por segundo de tráfego.

1.2.1 Objetivos Específicos

Para atingir o objetivo geral, as seguintes etapas foram seguidas no desenvolvimento deste trabalho:

- Inicialmente foi realizada uma breve pesquisa sobre as novas tecnologias de sistemas IDS/IPS disponíveis em software livre que prometiam bom desempenho extraindo o máximo de sistemas multi-cores, levando também em consideração disponibilidade de documentação e desenvolvimento bastante ativo.
- Após escolhida a ferramenta, foram feitos estudos mais aprofundados sobretudo em requisitos, recursos, cenários de utilização e compatibilidade com outras tecnologias.
- Por fim foi implementado um cenário de testes com o Suricata funcionando como gateway, protegendo um servidor que foi atacado pelo software Pytbull. Toda a parte de exibição de alertas foi feita usando o front-end Snorby.

¹Snort (www.snort.org) é um sistema completo de detecção e prevenção de intrusão.

²Bro (www.bro-ids.org) é um framework de análise de redes.

2 REFERENCIAL TEÓRICO

2.1 IPTABLES

O Iptables³ é um software em nível de espaço de usuário (*userspace*) que controla o framework Netfilter do *kernel* Linux porém, com os módulos adequados habilitados pode-se controlar praticamente todo o conjunto de funcionalidades do Netfilter. Com ele é possível construir um firewall *stateful* fazendo filtro de pacotes, tradução de endereços e/ou portas, contagem de pacotes, logs e manipulação de diversos outros campos nos cabeçalhos de pacotes como por exemplo ttl, tunelamento, marcação de pacotes etc.

2.2 SISTEMA DE PREVENÇÃO DE INTRUSÃO - IPS

São também conhecidos por Sistemas de Detecção e Prevenção de Intrusão (IDPS), são aplicações que tem como principais funções detectar e prevenir ataques ou atividades maliciosas às redes, ou hosts pertencentes a estas redes, bem como o registro destas atividades.

Tanto um Sistema IDS como um IPS monitoram o tráfego de uma rede. A principal diferença é que o IPS, após a detecção de uma ameaça, pode tomar alguma atitude como tentar bloquear o ataque.

2.2.1 Vulnerabilidades

Segundo (BRANCO, 2009), de maneira geral, as vulnerabilidades podem ter os estados de *pública com correção*, *não pública* e *pública sem correção*.

Pública com correção na maioria das vezes significa que a vulnerabilidade só foi divulgada quando já existia correção para ela.

É dito que uma vulnerabilidade existe mas não é pública quando ela ainda não tem correção e é conhecida por poucas pessoas que podem ser por exemplo desenvolvedores do software ou pesquisadores de segurança.

³Site do projeto: www.netfilter.org

É dito que uma vulnerabilidade é pública sem correção em dois casos distintos:

- Quando por exemplo um dos pesquisadores do caso anterior que descobriram a vulnerabilidade tentaram contatar os desenvolvedores e obtiveram respostas insatisfatórias, demora na correção ou o software é descontinuado; então eles divulgam a falha mesmo que ainda não tenha correção.
- O pior caso, profissionais de segurança descobrem que alguma vulnerabilidade não pública foi explorada em seus sistemas, e examinam os detalhes da exploração para descobrir qual a falha explorada. Neste caso a vulnerabilidade já está sendo explorada e as empresas fornecedoras de TI nem sequer sabem de sua existência.

O problema é que existe um intervalo de tempo entre a divulgação e a efetiva correção da vulnerabilidade em que as empresas podem ficar expostas e que pode ser fatal dada a velocidade com que as falhas têm sido exploradas. Isso sem falar das vulnerabilidades públicas sem correção.

A principal função de um sistema IPS é a de reduzir ao máximo esse tempo em que as empresas ficam expostas à determinada vulnerabilidade, não substituindo a efetiva necessidade de aplicação de patches de correção.

2.2.2 Métodos de Detecção

As tecnologias de prevenção de intrusão, de uma forma em geral, evoluíram para proporcionar a detecção com base em diversos fatores: assinaturas, comportamento, validação de protocolos e tecnologias de detecção. Sendo mais interessante as soluções híbridas por serem mais completas.

2.2.2.1 Assinaturas

Muitos entendem assinaturas como simples busca de padrões, ou busca de strings. Embora em algumas tecnologias isso muitas vezes seja verdade, isso não

se aplica a todas as soluções. Uma assinatura pode ser um conjunto elaborado de condições e regras para validação dos dados que passam pelo sistema. Por isso é importante entender quais os protocolos analisados e entendidos pelo sensor a ser avaliado. O sensor verifica os pacotes utilizando as assinaturas, e assim identifica os ataques conhecidos. Para cada novo ataque é criada uma nova assinatura.

2.2.2.2 Comportamento

A detecção baseada em comportamento envolve a criação de um *baseline* ou um padrão normal do comportamento da rede. Mudanças nesse padrão são indicativas de um ataque. Este tipo de detecção é muito importante no caso de ataque de *worms* que mudam o comportamento da rede, mas não é muito eficiente no caso de um ataque a um alvo específico pois não causa mudanças perceptíveis na rede.

2.2.2.3 Validação de Protocolos

Neste caso, independente da verificação de assinaturas o IPS seria capaz por exemplo de remontagem de sessões, ou seja, a sequencia de dados pertencentes a uma mesma transação por parte de um protocolo específico; a remontagem de pacotes na camada de rede, entre outras.

2.2.2.4 Tecnologias de Detecção

Um exemplo é a verificação de instruções de máquina passando pela rede e a emulação das instruções para evitar falsos positivos. Este método serve para a detecção genérica de explorações com inserção de código arbitrário - *shellcodes*.

2.2.3 Quanto ao Tipo de IPS

Quanto ao tipo, um sistema de prevenção da intrusão pode ser baseado na

estação ou na rede.

2.2.3.1 Baseado na Estação

Fica instalado na própria estação a ser monitorada, chamado de *Host Based Intrusion Prevention System* – HIPS.

2.2.3.2 Baseado na Rede

Monitoram todo tráfego da rede em questão e portanto são colocados normalmente inline, ou seja, todo o tráfego daquele trecho deverá passar através do IPS. Podem ser do tipo *Network Based Intrusion Prevention System* (NIPS) monitorando os protocolos de rede física ou *Wireless Intrusion Prevention Systems* (WIPS) monitorando os protocolos de rede sem fio.

São fatores fundamentais para um sistema IPS o desempenho e um conhecimento profundo da topologia da rede a ser protegida, pois um dos maiores problemas nas Soluções IPS é o alto número de falsos positivos – eventos normais e inofensivos detectados como ataques, gerando bloqueios indevidos. Existem ainda os falsos negativos, que são ataques que de alguma forma conseguem passar pelo sensor sem serem detectados. O ideal é que a solução seja ajustada de forma equilibrada.

2.3 SURICATA

A *Open Information Security Foundation* - OISF⁴ descreve o Suricata como (traduzido) "a ferramenta Open Source de Detecção e Prevenção da Intrusão da próxima geração, não se destina apenas a substituir ou imitar as ferramentas existentes na indústria, mas trazer novas idéias e tecnologias para esse campo". A OISF conta também com apoio de várias empresas privadas de TI além de

⁴ A OISF (Open Information Security Foundation) é uma fundação sem fins lucrativos, financiada pelo Department of Homeland Security – DHS juntamente com o Space and Naval Warfare Systems Command – SPAWAR, ambos do governo americano.

comunidades, sobretudo a *Emerging Threats Open Source*⁵, um de seus principais objetivos é o desenvolvimento do Suricata.

2.3.1 Recursos

Segundo a OISF, o Suricata tem como principais características e recursos:

- Executa aplicações Multi-threading;
- Suporte nas plataformas Linux, FreeBSD, OpenBSD, MAC OS X e WINDOWS;
- Detecção automática de protocolos (IP, TCP, UDP, ICMP, HTTP, SSL, TLS, DCERPC, SMTP, SSH, FTP, SMB e SMB2).
- O suricata tem palavras chave pra estes protocolos, assim o usuário pode escrever regras que procurem determinada palavra chave no protocolo tal, e até um determinado numero de vezes, e em caso positivo tomar alguma atitude, por exemplo gerar um alerta. Recurso extremamente importante na detecção de *Malwares*;
- Biblioteca HTP – Um “analisador HTTP”, licenciado pela GPL versão 2, escrito por Ivan Ristic⁶, que apesar do nome irá cobrir todos os protocolos usados em aplicações web. Será de grande utilidade no processamento de streams de camada de aplicação;
- Suporte nativo a IPS (modo inline) – NFQueue para Linux e Windows; ipfw para FreeBSD e NetBSD;
- Descompressão Gzip – A Biblioteca HTP irá decodificar streams com compressão Gzip;
- Comparação rápida de IP - Quando estiver usando regras baseadas apenas em comparação de endereços IP, elas serão executadas em um

⁵ A **Emerging threats Open Source** (<http://www.emergingthreats.net>) é uma comunidade criada por Matt Jonkman que desenvolve trabalhos com IPS/IDS, sobretudo na criação e distribuição de assinaturas para SNORT e Suricata.

⁶Disponível em: <http://blog.ivanristic.com/2009/11/http-parser-for-intrusion-detection-and-web-application-firewalls.html>, Acessado em 20/01/2012.

pre-processador implementado especialmente para esse fim;

- suporte nativo a IPV6;
- Saída de logs formato unified2 – pode ser lido pelo Barnyard⁷ e assim exportados para algum front-end como por exemplo o Snorby⁸;
- Tem ainda saídas do tipos fastlog, HTTP log (estilo apache) e Intrusion Detection Message Exchange Format – IDMEF;
- Compatível com a maioria das regras escritas para o Snort, inclusive as da Sourcefire VRT⁹ e da Emerging Threats;
- Flowint – É um recurso que permite armazenar variáveis e ainda realizar operações matemáticas. Poderá ser utilizado, por exemplo para contar, incrementar ou decrementar ocorrências, sendo que até o momento esse recurso está restrito ao mesmo fluxo de dados. O Suricata considera pertencentes do mesmo fluxo de dados, pacotes que tenham mesmo protocolo, IP de origem, IP de destino, porta de origem e porta de destino. No futuro, esse recurso será implementado de forma global possibilitando operações entre fluxos distintos, abrindo espaço para novas possibilidades de regras;

Dos módulos que estão bem avançados em seu desenvolvimento destacam-se: *IP Reputation*, *GPU Acceleration* na plataforma CUDA da NVIDIA, suporte as placas de rede da Napatech, *GeoIP Lookups*, *Global Variables/Flowbits* e *HTTP Blocklist Lookups*.

2.3.2 Aquisição de Pacotes

Até a versão 1.4 temos consolidados como mecanismos de aquisição de pacotes o AF_PACKET, o PF_RING, o NFQ para o Linux e o IPFW para os sistemas

⁷O **Barnyard** (<http://barnyard.sourceforge.net/>) é um software livre que, entre outras coisas, pode interpretar logs em formato binário

⁸O **Snorby** (<https://snorby.org/>) é um front-end para Sistemas IDS/IPS.

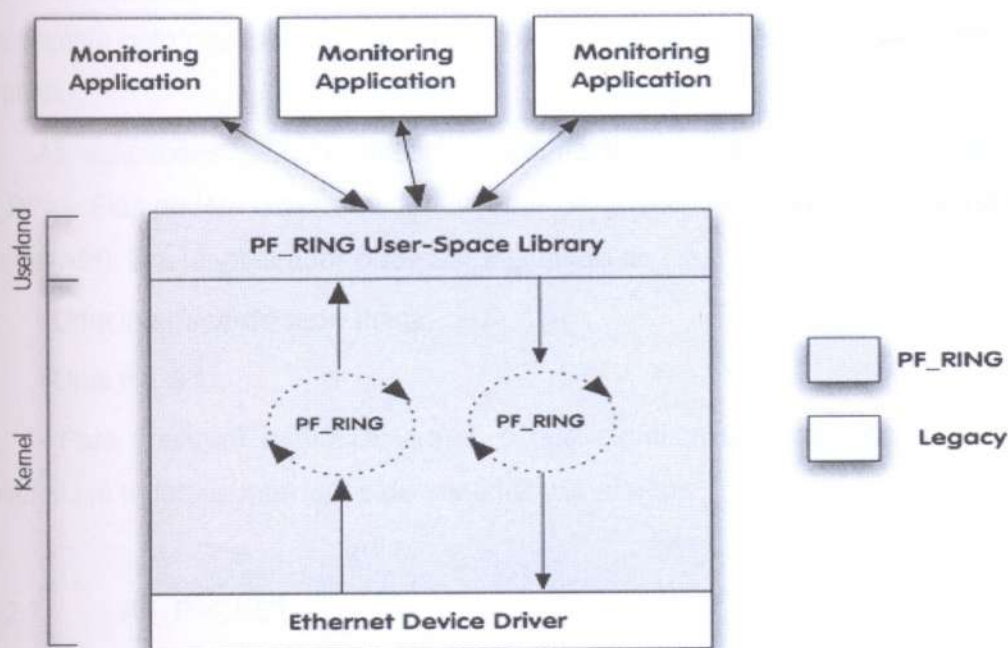
⁹A **Sourcefire Vulnerability Research Team - VRT** (<http://www.snort.org/vrt>) é uma equipe de especialistas da Sourcefire que disponibiliza regras atualizadas para Snort mediante cadastro ou subscrição.

BSD's. O Suricata pode ainda receber pacotes pela libpcap padrão e analisar arquivos do tipo PCAP. Este último e o IPFW não serão abordados neste trabalho.

2.3.2.1 PF_RING

Segundo a comunidade ntop¹⁰, desenvolvedora dessa solução *open source*, o PF_RING¹¹ é um tipo de soquete de rede que melhora drasticamente a velocidade de captura de pacotes em máquinas multi-processadas. A maioria dos drivers escritos para placas de rede consideram filas únicas de pacotes o que serializa o trabalho das aplicações e reduz consideravelmente a performance de aplicações de rede em PCs.

Figura 1: Arquitetura PF_RING



Fonte: https://svn.ntop.org/svn/ntop/trunk/PF_RING/doc/UsersGuide.pdf

¹⁰A Ntop (<http://www.ntop.org>) é uma comunidade que desenvolve vários projetos em software livre como o ntop, pfring entre outros.

¹¹Extraído de **PF_RING User Guide** (https://svn.ntop.org/svn/ntop/trunk/PF_RING/doc/UsersGuide.pdf). Acessado em 10/12/2012.

O trabalho do PF_RING é criar múltiplas filas de estruturas circulares e disponibiliza-las para cada CPU aumentando a paralelização dos processos e melhorando significativamente a performance.

A estrutura mostrada na Figura 1 é basicamente composta de:

- Bibliotecas para prover suporte transparente aos aplicativos no espaço de usuário.
- Kernel otimizado PF_RING com buffers que tem a estrutura de uma fila circular.
- Drivers PF_RING-aware, que não são obrigatórios, porém quando usados permitem melhorias na performance copiando os pacotes direto para os anéis de memória sem ter de passar pelas estruturas de dados do kernel do Sistema Operacional.

Os pacotes capturados são copiados nesses buffers circulares e então disponíveis a serem lidos pelas aplicações, uma vez lida aquela posição de memória é sobrescrita pelo próximo pacote recebido, ou seja, não há alocação e liberação de memória.

As aplicações se comunicam diretamente com essa estrutura do kernel PF_RING. Elas podem obter um identificador de anel e com isso emitir chamadas de sistema(API). Um identificador pode ser vinculado a:

- Uma interface de rede física
- Uma fila RX
- Para qualquer interface virtual, o que significa que pacotes recebidos ou enviados em todas as interfaces de sistema são aceitos.

2.3.2.2 AF_PACKET

Um tipo de *raw socket* que permite que um software do espaço de usuário possa interagir diretamente com os pacotes na camada de enlace, sem ser obrigado a usar a estrutura padrão de cabeçalhos do kernel. Em outras palavras, a aplicação ganha em desempenho pois neste caso ela pode ler ou escrever pacotes direto no

driver da placa de rede sem passar pelas camadas de rede e de transporte do kernel.

Em alguns aspectos até se parece com o PF_RING, porém o AF_PACKET vem de forma modular no kernel Linux e vem sofrendo constantes melhorias sobretudo a partir da versão 3.

Na versão 1.4 já é possível implementar o modo IPS com AF_PACKET, e existem implementações com AF_PACKET, modo IDS em links de 10Gbps¹² mas espera-se ainda mais melhorias no suporte ao AF_PACKET.

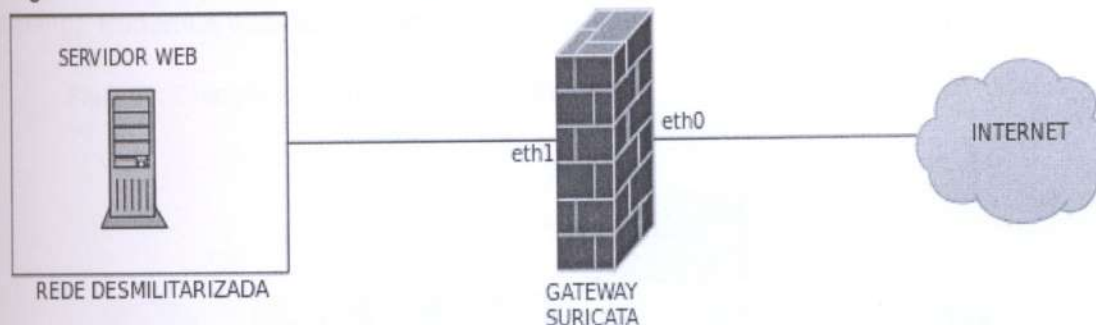
2.3.2.3 NFQ

É um manipulador de pacotes que usa a biblioteca netlink-queue do Netfilter (a partir do kernel linux 2.6.14), possibilitando acesso direto aos pacotes a partir do espaço de usuário. Ele trabalha apenas com IP e quando usado como uma target do iptables, empurra os pacotes do kernel para o espaço de usuário. Diferente de seu antecessor, o ip_queue, suporta várias filas simultâneas.

No GNU/Linux, fazendo uso do Iptables pode-se criar regras que enviem um pacote com determinadas características para o espaço de usuário.

Ex: iptables -I FORWARD -i eth0 -p tcp --dport 80 -j NFQUEUE --queue-num 1

Figura 2: Suricata inline funcionando como IPS



Na figura 2, imaginando o Suricata como Gateway os acessos ao servidor WEB, quando recebidos pelo Suricata, seriam redirecionados para o espaço de usuário e seriam testados conforme a configuração escolhida. A opção `--queue-num`

¹²Essa implementação é descrita em <https://home.regit.org/2012/07/suricata-to-10gbps-and-beyond/>. Acessado em 26/12/2012

especifica qual fila será enviada o pacote. Caso não seja informada, o iptables coloca a fila default "0", porém esse número pode ser especificado entre 0 e 65535.

2.3.3 Arquitetura

Em uma descrição superficial, o Suricata é composto em componentes básicos que são threads, módulos-thread e filas.

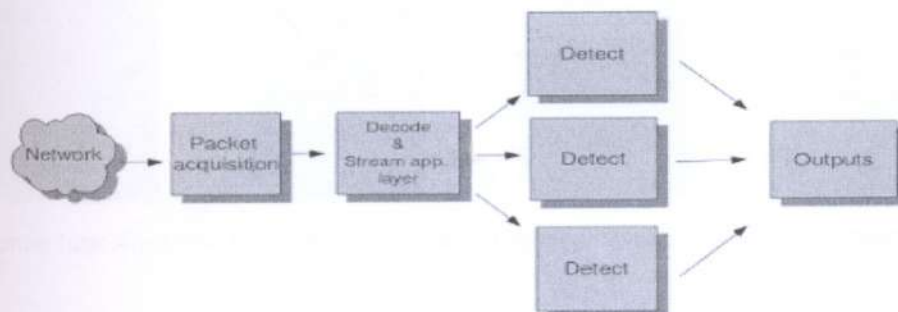
No Suricata, uma thread é da mesma forma que um processo rodando em um computador, sendo que uma thread pode ter um ou mais módulos-thread, mas apenas um fica ativo de cada vez.

Um módulo-thread é parte de uma funcionalidade. Por exemplo, um módulo é para decodificar um pacote, outro é para detecção e outro para saída.

Um pacote pode ser processado por mais de uma thread, sendo que apenas por uma de cada vez e a passagem de uma thread a outra se dá por meio de filas. Como o Suricata é multi-thread, várias threads ficam ativas simultaneamente e portanto vários pacotes são processados ao mesmo tempo.

A forma como threads, módulos e as filas são organizadas em conjunto é o que denominamos modos de operação do Suricata ou simplesmente "runmodes". Em geral estão ligados a escolha do dispositivo de captura e se irá atuar como IDS ou IPS. Em geral, esses modos vem com as opções auto, single e autofp (auto flow pinning). Podemos visualizar todos os modos com a opção – list-runmodes.

Figura 3: Exemplo de modo de operação IDS

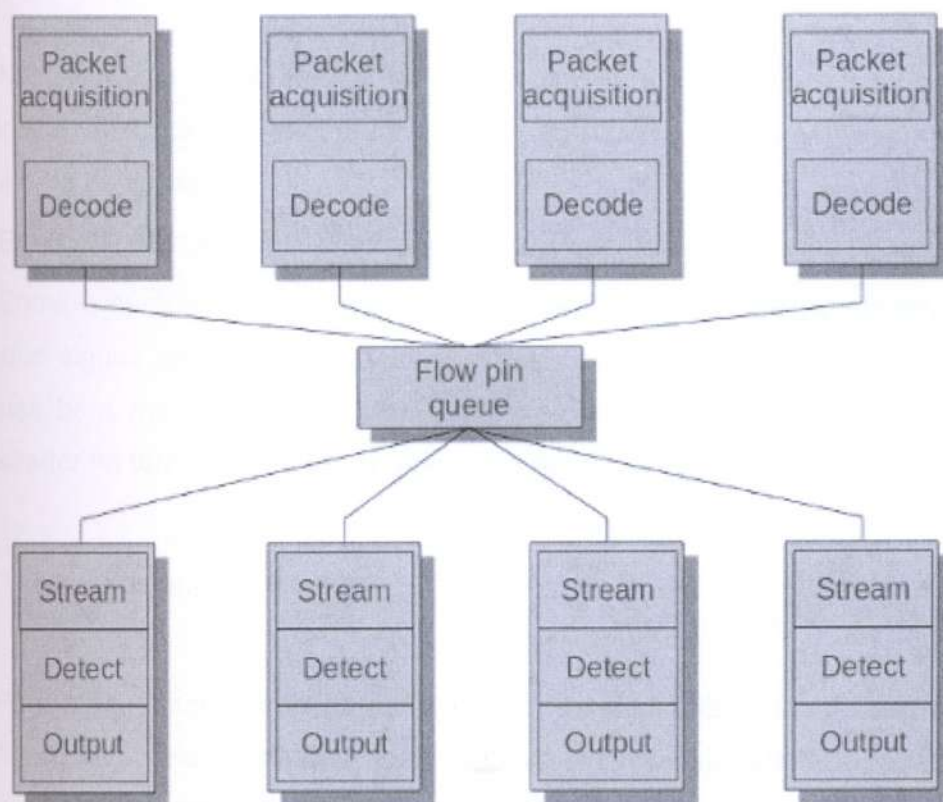


Fonte: <https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Runmodes>

A figura 3 mostra uma arquitetura em modo IDS e seus módulos-thread: Aquisição de pacotes, Decodificador, fluxo de camada de aplicação, detecção e saídas.

Após serem capturados pelo módulo de aquisição, os pacotes são decodificados, em seguida os fluxos de dados TCP são devidamente identificados e reconstruídos, para então serem enviados para os módulos de detecção onde a camada de aplicação será inspecionada a fim de comparar assinaturas, note-se que na detecção podemos ter várias threads operando simultaneamente e portanto vários fluxos sendo inspecionados ao mesmo tempo, por fim serão processados na saída os eventos e alertas.

Figura 4: Exemplo de modo PF_RING



Fonte: <https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Runmodes>

A figura 4 mostra o modo PF_RING onde cada fluxo segue sua própria rota definida no modo de operação, por exemplo caso na escolha do runmode pfiring auto

os pacotes pertencentes ao mesmo fluxo podem ser processados por qualquer thread de detecção, diferente do runmode pfring autofp onde esses pacotes serão processados por uma única thread de detecção. Ambas as opções são Multi-thread, já o runmode pfring single teríamos todos os fluxos numa única thread serializando o processo.

2.3.4 Principais Arquivos

São os arquivos mais importantes do Suricata, apesar de que muitos parâmetros podem ser passados via linha de comando.

2.3.4.1 Classification.config

Neste arquivo serão colocadas prioridades para cada regra. A sintaxe inclui um apelido, descrição e prioridade padrão que pode ser alterada conforme a conveniência do usuário.

Ex: *config classification: shellcode-detect,Executable code was detected,1*

Como o número de regras e seus alertas podem crescer significativamente, sendo que alguns alertas são apenas alarmes simples e outros acusam falhas de segurança bem mais graves, o uso deste mecanismo de diferenciação ajuda o administrador na tomada de ações conforme a gravidade do alerta.

2.3.4.2 Suricata.yaml

Principal arquivo de configuração, por ser extenso a ênfase será dada apenas as configurações mais importantes pertinentes a solução apresentada, ademais ele vem com vários comentários.

MAX-PENDING-PACKETS: Numero de pacotes processados simultaneamente, quanto maior melhor será o desempenho e mais uso será feito da memória RAM. Quando estiver em uso uma Unidade de Processamento Gráfico - GPU com suporte à plataforma CUDA da NVIDIA utilizando do algoritmo b2g_cuda,

poderia ser escolhido um numero ainda maior de pacotes processados pois eles utilizariam todo o poder de processamento da GPU.

RUNMODE: Modo de operação do Suricata. Para visualizar todos os modos disponíveis basta fazer:

```
# suricata --list-runmodes
```

No caso do modo NFQ, por exemplo, temos as opções auto, autofp e worker. No auto qualquer pacote pode ser processado por qualquer thread, já no modo autofp os pacotes pertencentes ao mesmo fluxo serão processados pela mesma thread, por fim no worker o trabalho fica serializado em filas onde os teremos uma única thread por fila.

ACTION-ORDER: Dependendo da natureza da regra, em seu cabeçalho existe uma ação a ser tomada no caso do pacote se enquadrar nessa regra. Essas ações podem ser pass, drop, reject e alert.

- pass: Quando a regra combinar com o pacote e a ação for pass, o pacote não será mais escaneado pelo suricata;
- drop: Usado como IPS, o pacote será barrado sem nenhuma resposta;
- reject: como no drop porém com envio de resposta para o remetente;
- alert: Nesse caso no casamento da regra com o pacote, será gerado um alerta e ele continuará a ser escaneado normalmente pelas demais regras;

DEFAULT-LOG-DIR: Diretório onde serão armazenados por padrão os logs do suricata. Dentro de cada seção de OUTPUTS, caso seja especificado um caminho absoluto para o arquivo de log, esta diretiva é ignorada.

OUTPUTS: São vários tipos diferentes de logs mas a estrutura é bem parecida basta habilitar ou desabilitar no campo enabled. Destaque aqui para o tipo de log unified2.log, que é um log binário bem rápido utilizado pelo banyard2 e ideal para redes de alta performance. É preciso atenção pois dependendo do que for habilitado os arquivos de log podem crescer e ocupar muito espaço em disco.

NFQ: Para o uso do Suricata Inline em modo IPS em sistemas GNU/Linux é necessário que se faça uso do iptables com regras direcionadas aos pacotes desejados encaminhando-os ao userspace onde esses pacotes serão analisados pelo Suricata.

MODE: Aqui pode-se colocar accept, repeat, route e fail-open. Em accept, por default, o pacote será analisado e haverá tomada de alguma ação drop, accept ou reject. Caso seja usado repeat o pacote será enviado para o suricata, marcado e devolvido para o iptables e novamente testado em todas as regras exceto aquela em que ele recebeu a marca. Em route o pacote, após ser analisado pelo suricata, poderá ser roteado para outra aplicação que esteja rodando em outra fila (queue) no espaço de usuário. Por fim, habilitando a opção fail-open ocorre que caso o fluxo de pacotes seja muito intenso e a aplicação não dê conta, então o suricata dirá ao kernel para aceitar todos os pacotes. Essa opção só funciona a partir do kernel linux 3.6.

AF_PACKET: Configuração de aquisição de pacotes via raw socket af_packet, que em modo IPS é relativamente novo no Suricata, mas que poderia funcionar como IDS ou as duas configurações simultâneas. Para o modo IPS as duas interfaces de rede devem ser preferencialmente iguais e ter configurações similares. Destaque para:

- copy-mode: Pode ser TAP onde os pacotes serão copiados de uma interface para outra, ou IPS que só haverá cópia após constatação de não existir nenhum drop para aquele pacote;
- copy-iface: Interface para onde serão copiados os pacotes;
- cluster-id: O af_packet irá balancear os pacotes baseado no fluxo, amarrado a uma thread que no caso proposto será única para cada interface de rede.
- use-mmap: mecanismo de bufferização circular similar ao do PF_RING, deve ser ativado nas duas interfaces para que o modo IPS funcione bem.

DETECT-ENGINE: O Suricata constrói internamente grupos de assinaturas, são verificados de qual rede e que tipo de tráfego serão comparados e a partir disso são criados esse grupos, por exemplo, não faz sentido inspecionar um pacote UDP com assinaturas TCP.

Com essa divisão, quanto maior o número de grupos, melhor o desempenho porém aumenta o consumo de memória, por outro lado, uma quantidade menor de grupos de assinaturas piora o desempenho mas diminui o consumo de memória. A opção profile aceita os valores low, medium e high; que dizem respeito a esse

balanço entre desempenho e uso de memória.

MPM-ALGO: Opção multi-pattern-matcher algorithm, ou seja, aqui é feita a escolha do algoritmo de busca de padrões pra ser usado dentro do Suricata. São suportados b2g, b2gc, b2gm, b3g, wumanber, ac e ac-gfbs. Logo abaixo podem ser feitas configurações específicas para cada algoritmo.

DEFRAG: Ocorre que alguns pacotes podem chegar fragmentados, nesse caso, o Suricata primeiro precisa remontar esses pacotes pra poder inspeciona-los. Aqui são configurados os parâmetros do defragmentador do Suricata como memória reservada para armazenar fragmentos, tamanho do hash usado para identificar fragmentos, quantidade de fragmentos, tempo máximo de espera desses fragmentos entre outros.

FLOW: Dizemos que pacotes de mesma tupla (protocolo, ip de origem, ip de destino, porta de origem e porta de destino) fazem parte do mesmo fluxo. Para cada pacote novo com uma nova tupla é necessário que o Suricata prepare a criação de um novo fluxo, quanto maior o número de fluxos maior o consumo de memória. Então existe uma thread de gerenciamento de fluxos que além de monitorar a memória utilizada, aloca previamente um número de fluxos de leitura (prealloc) para novas tuplas.

Numa situação de ataque ou anomalia, a memória utilizada chegará ao limite (memcap), o Suricata entra num modo de emergência onde os fluxos existentes expiram de maneira mais rápida permitindo alocar novos fluxos. Essa situação perdura até que a porcentagem livre de alocação de fluxos atinja o valor especificado em emergency-recovery.

FLOW-TIMEOUTS: Tempo para que os fluxos expirem. Aqui esses tempos são configurados de forma diferente e independente em modo normal e modo de emergência, para fluxos TCP, UDP, ICMP. São especificados parâmetros de expiração de tempo em segundos, referentes ao estado do fluxo. Para fluxos TCP é considerado o estado NEW durante o handshake, ESTABLISHED após o estabelecimento do handshake e CLOSED para o fechamento do fluxo que pode ser por meio de um FIN ou RESET. Já para fluxos UDP são admitidos os estados NEW para conexões em um sentido e ESTABLISHED para conexões nos dois sentidos.

STREAM: O comportamento é um pouco parecido com o dos fluxos, mas esta

seção é bastante específica e exclusiva para os pacotes TCP, que precisam ser primeiro monitorados e depois reconstruídos e esses dois passos são tratados de forma independente nesta seção. Destaque para os parâmetros verificação de checksum e o comportamento de monitorar ou não sessões já iniciadas (midstream).

DEFAULT-RULE-PATH: Diretório onde ficam armazenadas as regras

RULE-FILES: Arquivos contendo as assinaturas. Aqui podem ser adicionadas ou criadas novas regras.

VARS: Aqui são definidas algumas variáveis que serão usadas pelo Suricata e suas assinaturas.

ENGINE-ANALYSIS: É uma ferramenta para auxiliar na depuração de erros do Suricata. Será criado no diretório padrão de logs um arquivo de depuração para cada seção habilitada. Para ativar este recurso é necessário rodar o Suricata com a opção `--engine-analysis`:

```
suricata -c /etc/suricata/suricata.yaml --engine-analysis
```

PCRE: É uma biblioteca de implementação de expressões regulares. Aqui são configurados os parâmetros de limite de pesquisa e recursão.

LIBHTTP: A biblioteca libhttp é usada pelo Suricata para analisar sessões http. Para processar o tráfego é necessário lidar com diferentes tipos de servidores, tais como apache, IIS, etc. Cada tipo de servidor apresenta diferentes anomalias no processamento de tráfego HTTP, portanto é importante que o Suricata conheça o IP, rede e o tipo de servidor que terão tráfego inspecionado, sendo que para cada tipo de servidor será atribuída uma personalidade que será configurada nesta seção. A personalidade padrão é a IDS, que é bastante genérica e tentará cobrir de forma razoável a maioria das situações.

Merece destaque os parâmetros `request-body-limit` e `response-body-limit` que limitam até quantos bytes serão inspecionados do corpo da requisição e da resposta HTTP. Caso seja atribuído o valor zero, todo o corpo da requisição ou da resposta será verificado, porém dependendo do tamanho dessa requisição ou resposta, isso pode consumir muito recurso do hardware.

2.3.4.3 Threshold.config

Arquivo onde são configurados comandos que limitam a frequência de vezes que um evento em particular é logado. O objetivo principal é diminuir de forma conveniente a quantidade de logs para regras que geram muitos alertas lotando os arquivos de logs ou eventos que por sua natureza não necessitem de logs constantes.

Este mecanismo pode ser configurado de forma global no arquivo `threshold.config` ou nos próprios arquivos de regras. Sendo que dependendo da quantidade de regras, bem como suas atualizações, fica mais fácil de controlar a partir do `threshold.config` que tem a seguinte sintaxe:

- `threshold gen_id <gen_id>, sig_id <sig_id>, type <limit|threshold|both>, track <by_src|by_dst>, count <n>, seconds <t>`
- `suppress gen_id <gid>, sig_id <sid>, track <by_src|by_dst>, ip <ip|subnet>`

Onde tem-se:

Threshold – Limita o numero de vezes que um evento é logado durante um intervalo de tempo específico.

Type – Pode ser do tipo `limit`, `threshold` ou `both`. No `limit` são alertados tantos eventos num determinado intervalo de tempo, e após este intervalo os demais eventos são ignorados. No tipo `threshold` são emitidos alertas a cada numero de vezes que o evento ocorre num determinado intervalo de tempo. Em `both` temos os dois tipos presentes.

Suppress – Usado para suprimir alertas para uma regra, um host ou uma rede.

Tem-se alguns exemplos:

1 – Limita alertar um evento a cada 120 segundos para a assinatura com sid 24040:

```
threshold gen_id 1, sig_id 24040, type limit, track by_src, count 1, seconds 120
```

2 - Limita a alertar cada vez que 5 eventos ocorrerem no intervalo de 120 segundos para a assinatura 24040:

```
threshold gen_id 1, sig_id 24040, type threshold, track by_src, count 5, seconds 120
```

3 - Limita alertar um evento a cada 60 segundos, mas apenas se ocorrerem pelo

menos 50 eventos no intervalo de 60 segundos:

```
threshold gen_id 1, sig_id 24040, type both, track by_src, count 50, seconds 60
```

4 – Suprime evento da assinatura 20300 para o destino 10.0.0.0/8:

```
suppress gen_id 1, sig_id 20300, track by_dst, ip 10.0.0.0/8
```

Pode-se também colocar essas limitações junto com as assinaturas dentro dos arquivos de regras. Sua sintaxe muda um pouco mas o funcionamento é igual.

2.3.4.4 Arquivo de Regras

No diretório rules são colocadas os arquivos de regras referenciados pelo *suricata.yaml*. Com alterações mínimas, a sintaxe de regras do Suricata é perfeitamente compatível com a do Snort e existem comunidades que disponibilizam regras prontas como a comunidade opensource Emerging Threats.

Por exemplo, na seguinte assinatura:

```
alert tcp 10.0.0.0/8 5060 -> $EXTERNAL_NET any (msg:"ET VOIP Multiple Unauthorized SIP Responses TCP"; flow:established,from_server; content:"SIP/2.0 401 Unauthorized"; depth:24; threshold: type both, track by_src, count 5, seconds 360; reference:url,doc.emergingthreats.net/2003194; classtype:attempted-dos; sid:2003194; rev:6;)
```

Essa regra monitora respostas "SIP/2.0 401 Unauthorized" na porta tcp 5060 partindo da rede 10.0.0.0/8, porém devido ao threshold tipo both será gerado um alerta se dentro de 6 minutos existirem 5 ou mais eventos e irá alertar apenas uma vez a cada 6 minutos. Vem ainda a classificação, o sid que é número identificador dessa assinatura e por fim a versão.

2.4 PYTBULL

Pybull¹³ é um framework de testes para sistemas de detecção e prevenção da intrusão como Snort, Suricata ou qualquer outro que gere alertas. Pode ser usado para testar a detecção e capacidade de bloqueio; comparação entre sistemas

¹³Extraído do site do projeto (<http://pytbull.sourceforge.net/index.php?page=documentation>). Acessado em 16/02/2013.

IDS/IPS, comparar mudanças nas configurações e validação destas configurações.

Na versão 2.0, o Pytbul é equipado com aproximadamente 300 testes organizados em 11 módulos:

- 1- *BadTraffic*: Pacotes em não-conformidade com as RFC's são enviados para testar como eles são processados pelo sistema IDS/IPS.
- 2- *BruteForce*: Testa a habilidade do servidor em lidar com ataques de força bruta.
- 3- *ClientSideAttacks*: Esse módulo usa um shell reverso para instruir o servidor a baixar arquivos maliciosos de um servidor remoto testando a proteção do IDS/IPS contra ataques deste tipo.
- 4- *DenialOfService*: Testa o IDS/IPS contra tentativas de ataques de negação de serviço.
- 5- *evasionTechniques*: Uso de várias técnicas de invasões/evasões para checar se são detectadas pelo IDS/IPS.
- 6- *FragmentedPackets*: São enviados vários payloads fragmentados para testar a habilidade de remontagem de pacotes e detecção de ataques.
- 7- *ipReputation*: Testa a habilidade para detectar tráfego de envio ou recebimento com servidores de baixa reputação.
- 8- *NormalUsage*: Payloads que correspondem ao uso normal da rede.
- 9- *pcapReplay*: habilita a repetição de arquivos tipo pcap
- 10- *ShellCodes*: Envio de vários shellcodes na porta TCP 21 para verificar a capacidade do servidor em detectar ou rejeitar shellcodes.
- 11- *TestRules*: Teste básico de regras. Supõe-se que esses ataques devem ser detectados pelas regras default do IDS/IPS.

2.5 SNORBY

O Snorby¹⁴ é um front-end utilizado para visualizar os alertas gerados por Sistemas IDS como Snort, Sagan e Suricata. É um software livre licenciado sob a GPL versão 3, desenvolvido utilizando a linguagem de programação ruby on rails

¹⁴Extraído do site do projeto (<https://snorby.org>). Acessado em 18/02/2012.

para plataforma web.

Ele também contém gráficos e estatísticas dos alertas coletados, tem uma interface rápida e intuitiva, podendo receber dados de vários sensores ao mesmo tempo, mesmo que sejam sensores diferentes como por exemplo, um Snort e um Suricata.

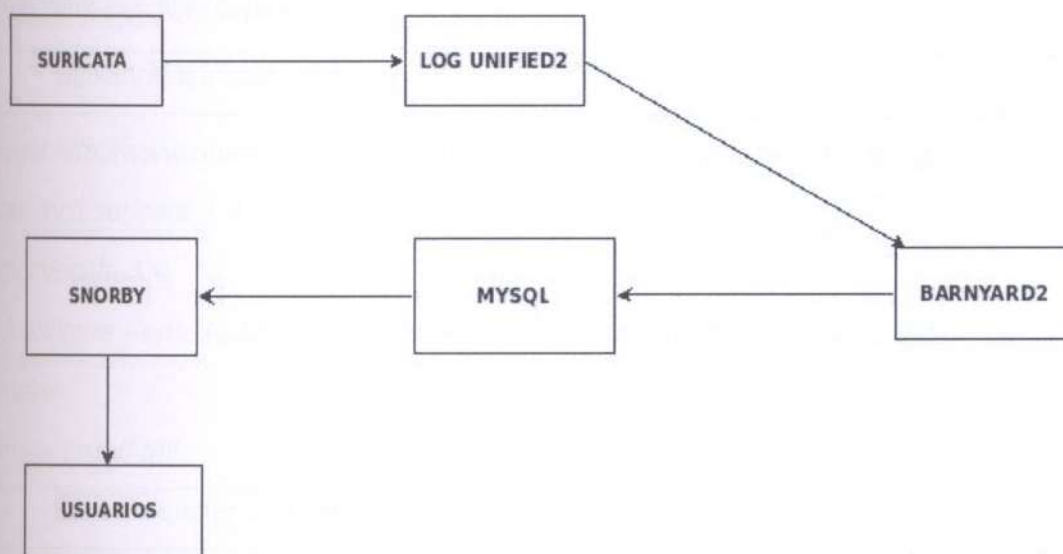
3 DESENVOLVIMENTO DA SOLUÇÃO

Existe a opção de baixar e rodar distribuições Linux prontas com o Suricata como o Smooth-Sec¹⁵ e a Security Onion¹⁶. Há também Suricata disponível nos repositórios Debian, Ubuntu, Gentoo e FreeBSD. Porém, normalmente estas versões do Suricata estão desatualizadas em relação a última versão disponibilizada pela OISF e por isso não foram utilizadas aqui.

A implementação do IPS foi feita em dois cenários distintos, sendo no primeiro a aquisição de dados foi via AF_PACKET e no segundo via NFQ. Num cenário em produção deve-se afinar as configurações para o melhor funcionamento de acordo com o ambiente, principalmente no que tange a escolha das regras, threshold, configurações da libhttp, fluxos, stream etc.

A figura 5 mostra uma visão completa da solução.

Figura 5: Diagrama completo da solução



O Suricata gera logs binários no formato unified2 para serem lidos pelo

¹⁵O Smooth-Sec (<http://bailey.st/blog/smooth-sec/>) é uma distribuição GNU/Linux que vem com uma solução de IPS/IDS pronta composta pelo Suricata, Snorby e Barnyard2.

¹⁶Security Onion (<http://code.google.com/p/security-onion/>) é uma distribuição GNU/Linux baseada em Ubuntu contendo Snort, Suricata, Sguil, Squert, Snorby, Bro, NetworkMiner, Xplico e outras ferramentas de segurança.

Barnyard2 que disponibiliza esses alertas para um banco mysql. Por fim o Snorby acessa essa base mysql e exibe em alto nível para o administrador da rede. Em um ambiente de produção onde normalmente existem vários sensores e de diferentes plataformas, é mais interessante que o Snorby juntamente com o mysql fiquem em um servidor diferente do servidor que hospeda os sensores.

3.1 INSTALAÇÃO

Nesta instalação foi usada a distribuição GNU/LINUX Debian Wheezy com kernel recompilado para o 3.7.2 e Suricata 1.4. Os procedimentos de recompilação do kernel são facilmente encontrados na internet e não serão abordados aqui.

São necessários os seguintes pacotes:

```
#apt-get -y install libpcre3 libpcre3-dbg libpcre3-dev build-essential autoconf
automake libtool libpcap-dev libnet1-dev libyaml-0-2 libyaml-dev zlib1g zlib1g-dev
libmagic-dev libcap-ng-dev pkg-config libnetfilter-queue-dev libnetfilter-queue1
libnfnetlink-dev libnfnetlink0
```

Baixando e instalando:

```
# wget http://www.openinfosecfoundation.org/download/suricata-1.4.tar.gz
# tar -xvzf suricata-1.4.tar.gz
# cd suricata-1.4
# ./configure --enable-nfqueue --prefix=/usr --sysconfdir=/etc --localstatedir=/var
# make
# make install-full
```

Para gerenciar a atualização das regras foi usado o software oinkmaster:

```
# apt-get install oinkmaster
```

Foi acrescentada a seguinte linha no arquivo /etc/oinkmaster.conf:

```
url = http://rules.emergingthreats.net/open/suricata/emerging.rules.tar.gz
```

E para a sincronização:

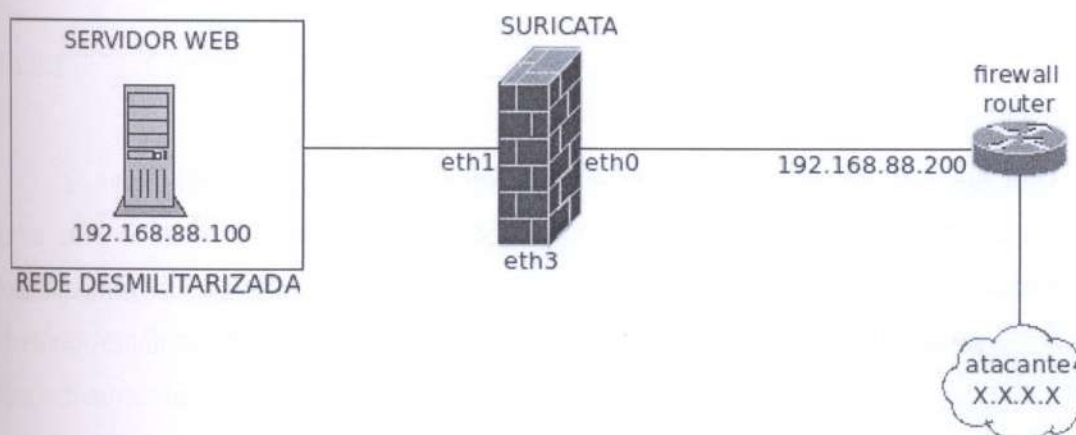
```
# oinkmaster -C /etc/oinkmaster.conf -o /etc/suricata/rules
```

O arquivo /etc/suricata/suricata.yaml usado encontra-se no Apêndice A.

3.1.1 Cenário com Aquisição via AF_PACKET

Neste cenário (figura 6), não há necessidade de configuração do Iptables, pois os dados brutos são interceptados em nível de enlace, sendo que regras de firewall em nível de rede são ineficazes neste caso, ficando o gateway invisível como se fosse uma bridge. Os logs podem ser visualizados por uma terceira interface de rede de forma segura.

Figura 6: Cenário utilizando AF_PACKET



Para inicializar o Suricata em modo de operação AF_PACKET basta fazer:

```
# suricata -c /etc/suricata/suricata.yaml -af-packet
```

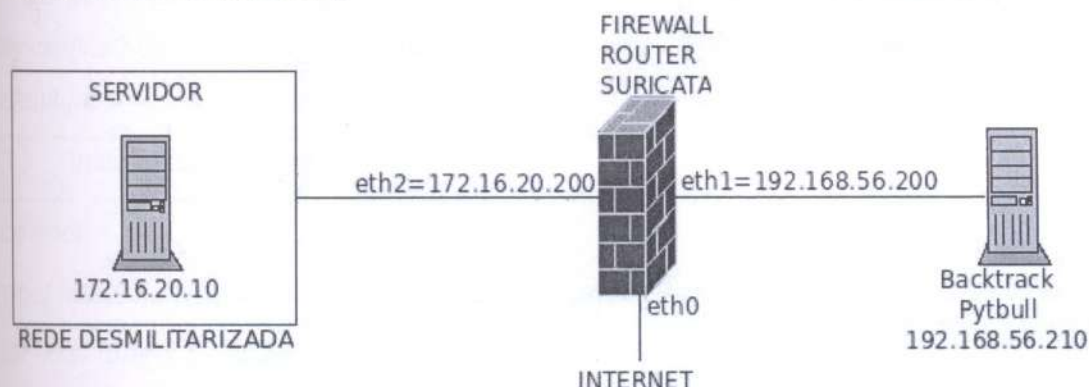
3.1.2 Cenário com Aquisição via NFQ

Neste cenário (figura 7), o Suricata está fazendo papel de firewall e roteador sendo fundamental a instalação do iptables. Como no cenário anterior o acesso aos logs seria através de uma terceira interface de rede.

Para inicializar o suricata em modo NFQ basta fazer:

```
# suricata -c /etc/suricata/suricata.yaml -q 0
```

Figura 7: Cenário usando NFQ



3.1.2.1 Configurando o Firewall

É necessário construir um firewall com iptables de forma que ele inicialize junto com o micro. Primeiro foi criado e inicializado (usando o insserv) o arquivo `/etc/init.d/firewall` conforme mostrado no Apêndice B. Em seguida, foi criado no diretório `/etc/firewall` os arquivos `rules`, `open` e `close` conforme Apêndices C, D e E respectivamente.

O iptables foi editado de forma a não interferir nos testes realizados sendo que todo tipo de tráfego foi roteado passando pelo Suricata por meio do NFQUEUE. A ideia é que passada a fase de testes, o iptables encaminhará para o Suricata apenas os pacotes referentes aos serviços a serem monitorados.

Se for necessária alguma modificação no iptables basta editar o arquivo `rules` e reiniciar o firewall como qualquer script de inicialização nos padrões Linux Standard Base - LSB. Desta forma em momentos de crise ou de teste de alguma regra nova do Suricata o firewall pode ser gerenciado de forma fácil e dinâmica.

3.1.3 Instalação do front-end

Antes de instalar o Snorby, é necessária a instalação de alguns softwares:

```
# apt-get install gcc g++ build-essential libssl-dev libreadline6-dev zlib1g-
dev libsqlite3-dev libxslt-dev libxml2-dev git-core libmysqlclient-dev mysql-server
default-jre
```

Instalação do Ruby 1.9.3:

```
# cd /root
# wget http://ftp.ruby-lang.org/pub/ruby/1.9/ruby-1.9.3-p286.tar.gz
# tar -xvzf ruby-1.9.3-p286.tar.gz
# cd ruby-1.9.3-p286
# ./configure
# make && make install
# cd ext/openssl
# ruby extconf.rb
# make
# make install
# cd /root
```

Instalação do Rubygems 1.8.25:

```
# wget production.cf.rubygems.org/rubygems/rubygems-1.8.25.tgz
# tar -xvzf rubygems-1.8.25.tgz
# cd rubygems-1.8.25
# ruby setup.rb
# cd ..
```

Instalação do Rails e outras dependências:

```
# gem install rubygems-update
# update_rubygems
# gem install rails bundler thor i18n tzinfo builder memcache-client rack rack-
tester rubis mail text-format rack-mount sqlite3 pdftk
```

Instalação do ImageMagick 6.8.3-2:

```
# wget www.imagemagick.org/download/ImageMagick.tar.gz
# tar -xvzf ImageMagick.tar.gz
# cd ImageMagick-6.8.3-2
# ./configure
# make && make install
# ldconfig /usr/local/lib/
# make check
# cd ..
```

Instalação do wkhtmltopdf 0.11:

```
# wget wkhtmltopdf.googlecode.com/files/wkhtmltopdf-0.11.0_rc1-static-amd64.tar.bz2
# tar -xjvf wkhtmltopdf-0.11.0_rc1-static-amd64.tar.bz2
# cp wkhtmltopdf-amd64 /usr/bin/wkhtmltopdf
```

E finalmente o Snorby 2.5.6

```
# wget https://github.com/Snorby/snorby/zipball/v2.5.6
# unzip v2.5.6
# cp -r Snorby-snorby-42dd6d5 /var/www/snorby
```

Os arquivos database.yml e snorby_config.yml devem ser criados conforme Apêndices F e G, respectivamente:

```
# vim /var/www/snorby/config/database.yml
# vim /var/www/snorby/config/snorby_config.yml
# cd /var/www/snorby
# bundle update activesupport railties rails
# gem install arel ezprint && bundle install
# bundle exec rake snorby:setup
```

Neste momento para rodar o snorby em produção basta executar o comando:

```
# bundle exec rails server -e production
```

E acessar o servidor na porta 3000, através de um navegador qualquer. Porém, pela pouca flexibilidade dessa solução foi preferível usar o apache através do Passenger:

```
# apt-get install apache2 apache2-prefork-dev libapr1-dev libaprutil1-dev libopenssl-
ruby libcurl4-openssl-dev
# gem install --no-ri --no-rdoc passenger
# /usr/local/bin/passenger-install-apache2-module -a
```

Criação do arquivo `/etc/apache2/mods-available/passenger.load` com o seguinte conteúdo:

```
LoadModule passenger_module /usr/local/lib/ruby/gems/1.9.1/gems/passenger-
3.0.19/ext/apache2/mod_passenger.so
```

Criação do arquivo `/etc/apache2/mods-available/passenger.conf` com o seguinte conteúdo:

```
<IfModule mod_passenger.c>
    PassengerRoot /usr/local/lib/ruby/gems/1.9.1/gems/passenger-3.0.19
    PassengerRuby /usr/local/bin/ruby
</IfModule>
```

Carregando os módulos:

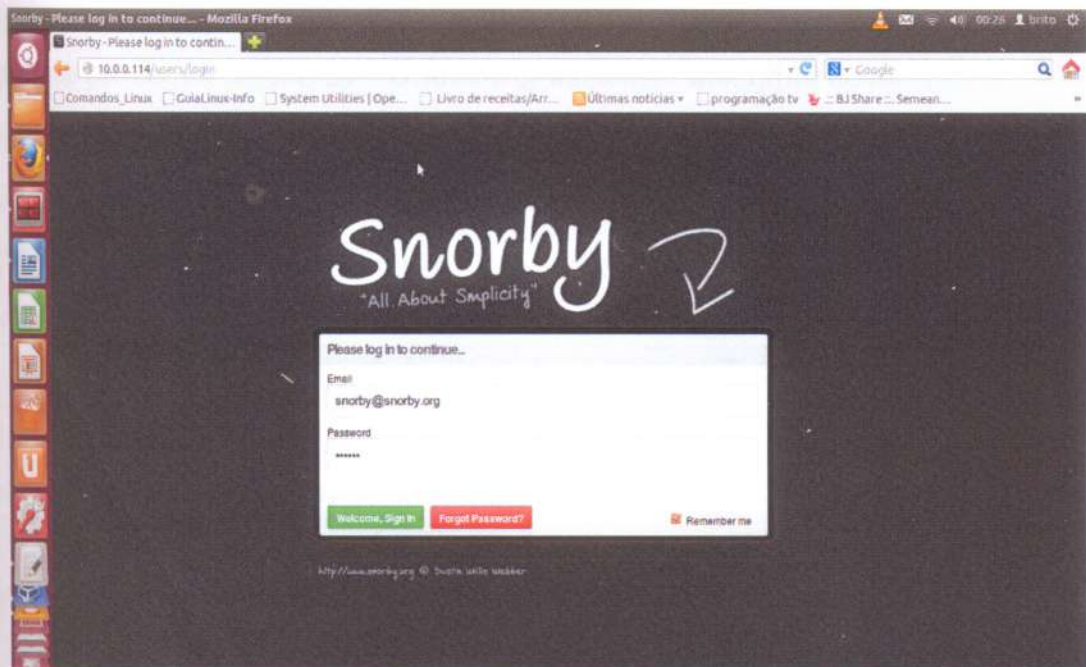
```
# a2enmod passenger
# a2enmod rewrite
# a2enmod ssl
```

Edição do conteúdo do arquivo `/etc/apache2/sites-available/default` conforme Apêndice H:

```
# vim /etc/apache2/sites-available/default
# cd /var/www/snorby
# bundle pack && bundle install --path vendor/cache
# chown -R www-data:www-data /var/www/snorby
# /etc/init.d/apache2 restart
```

Neste momento a página do Snorby está acessível através do apache na porta 80 conforme figura 8 inicialmente com login `snorby@snorby.org` e senha `snorby`.

Figura 8: Tela inicial do Snorby



Instalação do Barnyard2:

```
# apt-get install mysql-client libpcrc3 libpcrc3-dbg libpcrc3-dev build-essential
autoconf automake libtool libpcap-dev libnet1-dev libmysqlclient-dev

# cd /root

# wget www.securixlive.com/download/barnyard2/barnyard2-1.9.tar.gz

# tar -xvf barnyard2-1.9.tar.gz

# cd barnyard2-1.9

# ./configure --with-mysql-libraries=/usr/lib/x86_64-linux-gnu/ --with-mysql

# make && make install
```

Criação do arquivo /etc/suricata/barnyard2.conf conforme Apêndice I:

```
#vim /etc/suricata/barnyard2.conf
```

Configuração do mysql:

```
# mysql -u root -p
# create user 'fulano'@'localhost' IDENTIFIED BY 'password';
# grant all privileges on snorby.* to 'fulano'@'localhost' with grant option;
# flush privileges;
# GRANT ALL ON snorby.* TO fulano@'localhost' IDENTIFIED BY 'password';
# quit
# /etc/init.d/mysql restart
```

Após inicializar o Suricata, basta iniciar o Barnyard2 com o comando:

```
# barnyard2 -c /etc/suricata/barnyard2.conf -d /var/log/suricata -f unified2.alert -w
/var/log/suricata/suricata.waldo -D
```

Agora com um navegador basta acessar o servidor, inicializar o worker e verificar que os contadores deverão ser atualizados.

3.1.4 Inicialização do Suricata e dependências

Os comandos de inicialização do Suricata, barnyard, bem como a sincronização das regras pelo oinkmaster podem ser colocados como um script do tipo LSB aqui chamado de ips, conforme Apêndice J.

Este script foi colocado dentro do diretório /etc/init.d e inicializado com o comando insserv.

Para iniciar, parar ou atualizar as regras do Suricata basta fazer:

```
#/etc/init.d/ips <opção>
```

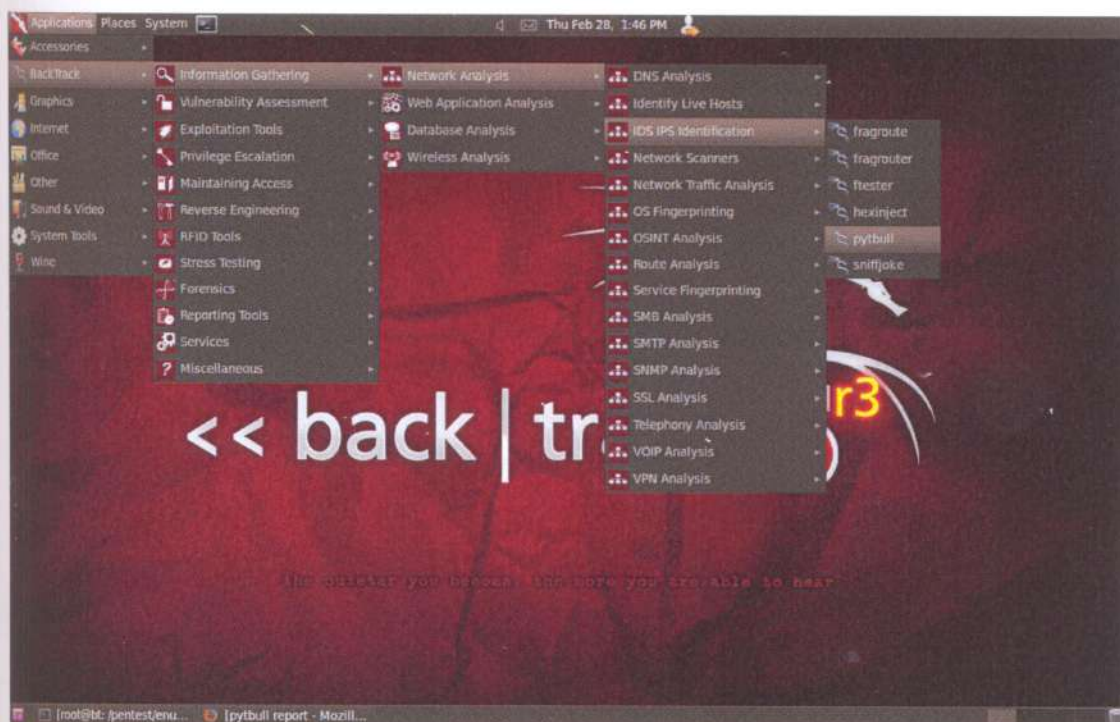
3.2 ATACANDO COM PYTBULL

O cenário de ataque usado foi o mesmo da figura 7. Como o objetivo aqui era apenas mostrar o funcionamento da ferramenta sem fazer comparações, foi demonstrado apenas o ataque contra o Suricata em modo NFQ. Apesar de que com algumas pequenas alterações no cenário e no modo de operação do Suricata, pode-se chegar a resultados similares usando AF_PACKET.

Para o ataque foi utilizado o framework Pytbull que vem disponibilizado na distribuição Backtrack¹⁷ 5 R3. A instalação do Backtrack não será demonstrada aqui, pois é bastante intuitiva, foi realizada usando uma imagem ISO, além de instruções¹⁸ disponíveis no site do desenvolvedor.

Após a instalação do Backtrack foi executado o Pytbull conforme figura 9:

Figura 9: Pytbull no Backtrack



Devido a um erro apresentado, foi necessário acrescentar a seguinte linha no início do arquivo `/usr/lib/pymodules/python2.6/scapy/sendrecv.py`:

```
from base_classes import SetGen
```

Por fim, basta alterar o arquivo `/pentest/enumeration/ids/pytbull/conf/config.cfg` conforme Apêndice K. Agora o Pytbull está apto a executar:

```
#./pytbull -c conf/config.cfg -t 172.16.20.10 -m gateway
```

O resultado do comando acima foi o da figura 10:

¹⁷Backtrack (<http://www.backtrack-linux.org/>) é uma distribuição GNU/Linux que vem com várias ferramentas de segurança, sobretudo forense, testes de penetração etc...

¹⁸http://www.backtrack-linux.org/wiki/index.php/Install_BackTrack_to_Disk

Figura 10: Executando Pytbull

```

^ v x root@bt: /pentest/enumeration/ids/pytbull
File Edit View Terminal Help

          Pytbull
          Sebastien Damaye, aldeid.com

What would you like to do?
1. Run a new campaign (will erase previous results)
2. View results from previous campaign
3. Exit
Choose an option: 1
(gateway mode)

-----
| pytbull will set off IDS/IPS alarms and/or other security devices |
| and security monitoring software. The user is aware that malicious |
| content will be downloaded and that the user should have been    |
| authorized before running the tool.                               |
-----

Do you accept (y/n)? y

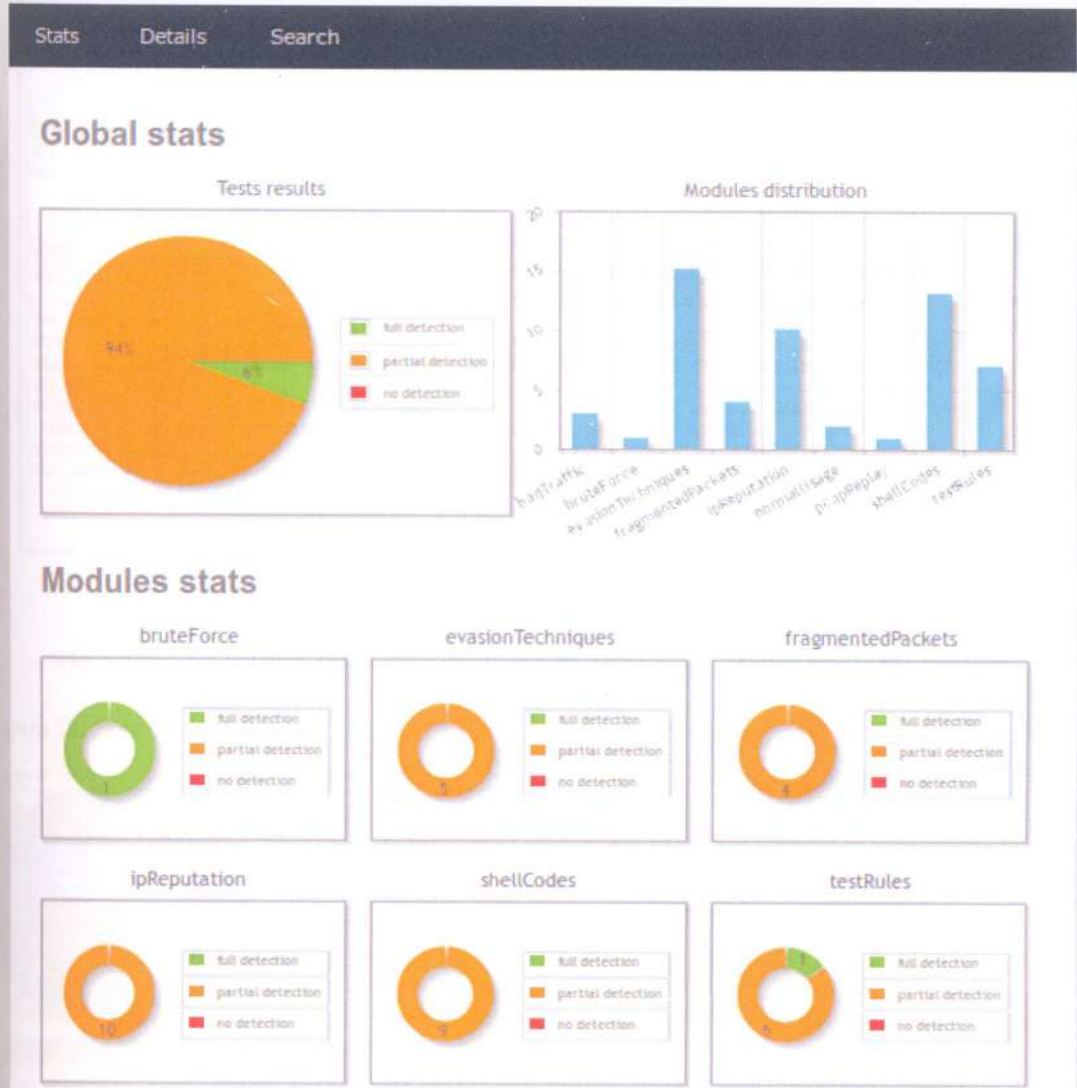
```

O próprio Pytbull faz um comparativo entre o ataque realizado e o log gerado pelo Suricata naquele momento, porém para isso, ele necessita de acesso de leitura ao arquivo `/var/log/suricata/fast.log` via FTP. Como o objetivo era a efetivação de testes e já havia um servidor FTP no servidor atacado, foi preferível compartilhar os logs do Suricata com este servidor através de uma conexão NFS a parte.

3.3 RESULTADOS

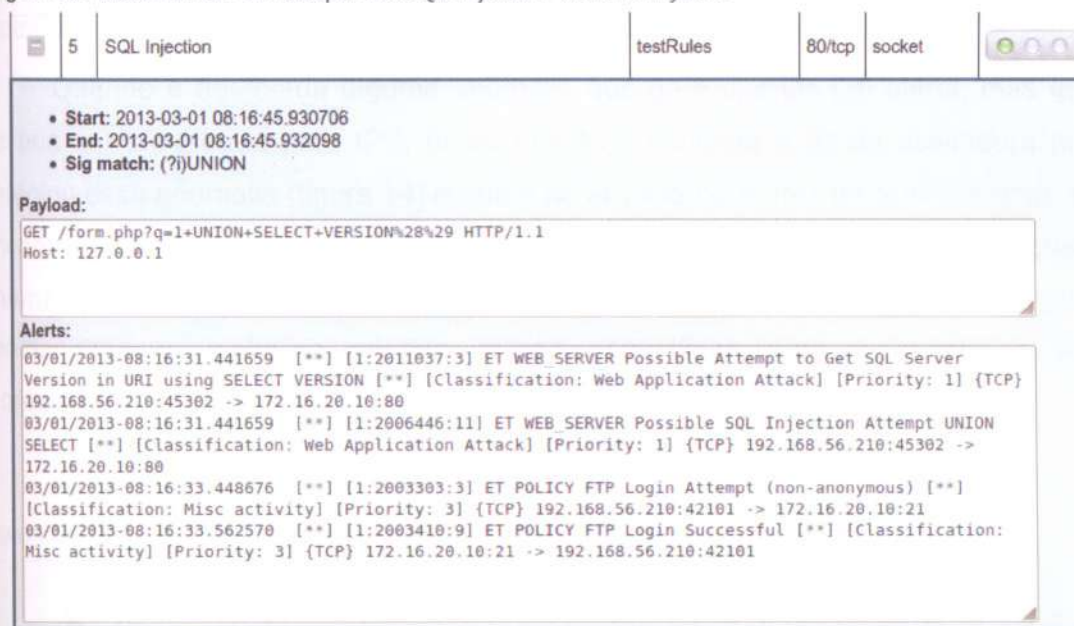
Pela máquina onde está instalada o Backtrack, no final do ataque é disponibilizada uma pagina web contendo os resultados do ataque acessada por um cliente web em <http://127.0.0.1:8080> (figura 11).

Figura 11: Resultados do Pytbull



Clicando em *details* (figura 12), podemos ver informações mais específicas de cada tipo de ataque e a regra do Suricata que fez a detecção dele. Este recurso de exibição do Pytbull é muito útil para realizar testes em sistemas IPS/IDS, porém serve para mostrar apenas dados referentes aos ataques realizados pela ferramenta naquele momento. O campo *Alerts* mostra o log gerado pelo Suricata para aquele ataque.

Figura 12: Detalhes de um ataque de SQL Injection visto no Pytbull



Na figura 13 temos uma tela do Snorby logo após o ataque com Pytbull:

Figura 13: Snorby após ataque



O Snorby, por outro lado, é bastante robusto e disponibiliza várias opções na

exibição dos logs, mantendo histórico, mostra os ataques por origem, destino, protocolo, classificação e ainda de que sensor foi feita a detecção.

Quando é detectada alguma anomalia que gere apenas um alerta, mas que deveria ser bloqueada pelo IPS, basta que seja checada a ID da assinatura que detectou essa anomalia (figura 14) e então no arquivo de regras do Suricata onde se encontra essa assinatura trocar a palavra alert por drop, a partir disso aquela anomalia será bloqueada pelo IPS. Na prática deve-se atualizar e afinar as regras constantemente, inclusive criando regras específicas para o serviço a ser monitorado.

Figura 14: Detalhe de um ataque de SQL Injection visto no Snorby

The screenshot displays the Snorby interface for a detected SQL Injection attack. The event is titled "ET WEB_SERVER Possible SQL Injection Attempt UNION SELECT" and occurred at 8:15 AM. The source IP is 192.168.56.210 and the destination is 172.16.20.10.

IP Header Information:

Source	Destination	Ver	hlen	Tos	Len	ID	Flags	Off	TTL	Proto	Csum
192.168.56.210	172.16.20.10	4	5	0	126	30349	0	0	63	6	2904

Signature Information:

Generator ID	Sig. ID	Sig. Revision	Activity (4-6252)	Category	Sig Info
1	2006446	11	0.00%	web-application-attack	Query Signature Database View Rule

TCP Header Information:

Src Port	Dst Port	Seq	Ack	Off	Res	Flags	Win	Csum	UHP
45302	80	2147483647	2115416488	8	0	24	913	50088	0

References:

Type	Value
url	en.wikipedia.org/wiki/SQL_injection
url	doc.emergingthreats.net/2006446

Payload:

```

00000000: 47 45 54 20 2f 66 6f 72 6d 2e 70 68 70 3f 71 9d 31 2b 55 4e 49 4f 4e 2b 53 45  GET./form.php?q=1+UNION+SE
00000010: 4c 45 43 54 2b 56 45 52 53 49 4f 4e 25 32 36 25 32 39 20 48 54 54 50 2f 31 2e  LECT+VERSION%28%29.HTTP/1.
00000034: 31 0d 0a 48 6f 73 74 3a 20 31 32 37 2e 30 2e 30 2e 31 0d 0a 0d 0a 1..Host::127.0.0.1....
  
```

Notes:

This event currently has zero notes - You can add a note by clicking the button below.

[Add A Note To This Event](#)

Por exemplo, no ataque tipo SQL Injection da figura 14, após colocar o Suricata para bloquear este tipo de assinatura e refazendo o ataque, foi verificada a

saída do arquivo fast.log mostrado na figura 15.

Figura 15: Arquivo fast.log mostrando o bloqueio de um ataque de SQL Injection

```
root@oisf:~#
root@oisf:~# cat /var/log/suricata/fast.log |grep Drop|grep SQL|more
03/01/2013-12:10:21.712994 [Drop] [**] [1:2006446:11] ET WEB SERVER Possible SQL Injection Attempt
UNION SELECT [**] [Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.56.210:5471
2 -> 172.16.20.10:80
03/01/2013-12:10:21.712994 [Drop] [**] [1:2011037:3] ET WEB SERVER Possible Attempt to Get SQL Ser
ver Version in URI using SELECT VERSION [**] [Classification: Web Application Attack] [Priority: 1]
{TCP} 192.168.56.210:54712 -> 172.16.20.10:80
root@oisf:~#
```

Seguindo o mesmo raciocínio foram detectados e bloqueados alguns ataques de shellcodes mostrados na figura 16:

Figura 16: Ataques de shellcodes bloqueados pelo IPS

```
root@oisf:~# cat /var/log/suricata/fast.log|grep SHELLCODE |tail
03/01/2013-12:15:25.215357 [Drop] [**] [1:2010396:3] ET SHELLCODE METASPLOIT BSD Bind
shell (Pex Alphanumeric Encoded 1) [**] [Classification: Executable Code was Detected]
[Priority: 1] {TCP} 192.168.56.210:51756 -> 172.16.20.10:21
03/01/2013-12:15:25.215357 [Drop] [**] [1:2002904:5] ET SHELLCODE x86 Alpha2 GetEIPs
Encoder [**] [Classification: Executable Code was Detected] [Priority: 1] {TCP} 192.16
8.56.210:51756 -> 172.16.20.10:21
03/01/2013-12:15:30.730210 [Drop] [**] [1:2011803:5] ET SHELLCODE Possible TCP x86 JM
P to CALL Shellcode Detected [**] [Classification: Executable Code was Detected] [Prio
rity: 1] {TCP} 192.168.56.210:51759 -> 172.16.20.10:21
03/01/2013-12:15:41.719429 [Drop] [**] [1:2010422:3] ET SHELLCODE METASPLOIT BSD Reve
rse shell (PexFnstenvMov Encoded 2) [**] [Classification: Executable Code was Detected]
[Priority: 1] {TCP} 192.168.56.210:51765 -> 172.16.20.10:21
03/01/2013-12:15:41.719429 [Drop] [**] [1:2010408:3] ET SHELLCODE METASPLOIT BSD Reve
rse shell (PexFnstenvSub Encoded 2) [**] [Classification: Executable Code was Detected]
[Priority: 1] {TCP} 192.168.56.210:51765 -> 172.16.20.10:21
03/01/2013-12:15:41.719429 [Drop] [**] [1:2010402:3] ET SHELLCODE METASPLOIT BSD Bind
shell (PexFstEnvMov Encoded 2) [**] [Classification: Executable Code was Detected] [P
riority: 1] {TCP} 192.168.56.210:51765 -> 172.16.20.10:21
03/01/2013-12:15:41.719429 [Drop] [**] [1:2002903:5] ET SHELLCODE x86 PexFnstenvMov/S
ub Encoder [**] [Classification: Executable Code was Detected] [Priority: 1] {TCP} 192
.168.56.210:51765 -> 172.16.20.10:21
03/01/2013-12:15:41.719429 [Drop] [**] [1:2009247:3] ET SHELLCODE Rothenburg Shellcod
e [**] [Classification: Executable Code was Detected] [Priority: 1] {TCP} 192.168.56.2
10:51765 -> 172.16.20.10:21
03/01/2013-12:15:47.093544 [Drop] [**] [1:2011803:5] ET SHELLCODE Possible TCP x86 JM
P to CALL Shellcode Detected [**] [Classification: Executable Code was Detected] [Prio
rity: 1] {TCP} 192.168.56.210:51768 -> 172.16.20.10:21
03/01/2013-12:15:47.093544 [Drop] [**] [1:2009250:3] ET SHELLCODE Mainz/Bielefeld She
llcode [**] [Classification: Executable Code was Detected] [Priority: 1] {TCP} 192.168
.56.210:51768 -> 172.16.20.10:21
root@oisf:~#
```

4 CONCLUSÕES E TRABALHOS FUTUROS

O uso de ferramentas de detecção e prevenção de incidentes é fundamental em ambientes corporativos pois, desde que bem configurados, diminuem bastante o tempo de indisponibilidade dos serviços.

Este trabalho teve o objetivo de demonstrar o funcionamento de um Sistema IPS utilizando do software livre Suricata que demonstrou ser bastante robusto, flexível e com desenvolvimento bastante ativo.

Normalmente o uso de *appliances* pagos tem vantagens como robustez e confiabilidade, porém existem amarras como a incompatibilidade com outras plataformas, pouca liberdade de customização e o pagamento por atualizações de software e assinaturas pode ter um custo bastante elevado no longo prazo.

No uso do Suricata como IPS é preciso um hardware compatível com o link a ser monitorado. Notou-se que nas mesmas condições, a aquisição de dados via AF_PACKET foi mais rápida do que via NFQ, no entanto em uma situação de roteamento e um pouco menos recurso de hardware o uso do NFQ pode ser bem interessante.

No caso apresentado, optou-se por colocar todas as regras para somente alertar e aos poucos trocando para bloqueio aquelas mais críticas, dessa forma foi sacrificada a segurança em prol de um baixo número de falsos positivos. O conjunto de regras default do Suricata foi capaz de alertar ou bloquear a maioria dos ataques feitos pelo Pytbull. Ainda é necessário a criação de novas regras bem como maiores ajustes nas existentes, mas de maneira geral o IPS se mostrou bastante confiável, sem apresentar travamentos ou maiores problemas durante o seu funcionamento.

Pela flexibilidade da solução é possível pensar em montar uma zona de monitoramento com sistemas IPS/IDS instalados nos próprios hosts, inline ou em modo de escuta colocados em pontos estratégicos da rede. Os logs poderiam ser centralizados, através de uma rede isolada, em um ou mais front-ends, o que facilitaria muito para o profissional de segurança na identificação de problemas e tomada de ações corretivas.

REFERÊNCIAS BIBLIOGRÁFICAS

Boyles, T., CCNA Security Study Guide: Exam 640-553. 1º ed.: Sybex, 2010.

Branco, R.R; "IPS na intimidade", Linux Magazine, Número 56, Julho 2009.

Comunidade de desenvolvedores do PF_RING. Disponível em:
(http://www.ntop.org/products/pf_ring/).

He, K. K.; "Kernel Korner - Why and How to Use Netlink Socket", Linux Journal, Volume 130, Fevereiro 2002. Disponível em
<http://www.linuxjournal.com/node/7356>. Acessado em Agosto de 2012.

Insolubile, G.; "Linux socket filter: sniffing bytes over the network", Linux Journal, Volume 2001, Issue 86, Junho 2001. Disponível em
<http://www.linuxjournal.com/article/4659>. Acessado em Março de 2012.

Insolubile, G.; "Inside the Linux Packet Filter", Linux Journal, Volume 94, Fevereiro 2002. Disponível em <http://www.linuxjournal.com/article/4852>. Acessado em Agosto de 2012.

Insolubile, G.; "Inside the Linux Packet Filter, part II", Linux Journal, Volume 95, Março 2002. Disponível em <http://www.linuxjournal.com/article/5617>. Acessado em Agosto de 2012.

NTOP; "PF_RING USER GUIDE", Versão 5.5.2, Outubro 2012. Disponível em:
<https://svn.ntop.org/svn/ntop/trunk/PF_RING/doc/UsersGuide.pdf>. Acessado em Dezembro de 2012.

OISF - Open Information Security Foundation; Desenvolvedor do Suricata. Disponível em: (<http://www.openinfosecfoundation.org/>).

Scarfone, K.; Mel, P.; *Guide to Intrusion Detection and Prevention Systems (IDPS)*. NIST – National Institute of Standards and Technology, Fevereiro 2007. Disponível em <<http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf>>. Acessado em Março de 2012.

APÊNDICE A – ARQUIVO SURICATA.YAML

```
%YAML 1.1
---
max-pending-packets: 512
pid-file: /var/run/suricata.pid
default-packet-size: 1514
default-log-dir: /var/log/suricata/

outputs:
- fast:
  enabled: yes
  filename: fast.log
  append: yes
- unified2-alert:
  enabled: no
  filename: unified2.alert
- http-log:
  enabled: no
  filename: http.log
  append: yes
- drop:
  enabled: yes
  filename: drop.log
  append: yes

nfq:
  mode: accept

af-packet:
- interface: eth0
  threads: 1
  defrag: yes
  cluster-type: cluster_flow
  cluster-id: 98
  copy-mode: ips
  copy-iface: eth1
  buffer-size: 64535
  use-mmap: yes
- interface: eth1
```

```
threads: 1
cluster-id: 97
defrag: yes
cluster-type: cluster_flow
copy-mode: ips
copy-iface: eth0
buffer-size: 64535
use-mmap: yes
threshold-file: /etc/suricata/threshold.config
```

```
detect-engine:
- profile: medium
- sgh-mpm-context: auto
- inspection-recursion-limit: 3000
```

```
threading:
set-cpu-affinity: no
detect-thread-ratio: 1.5
```

```
mpm-algo: ac
```

```
pattern-matcher:
```

```
- b2gc:
  search-algo: B2gSearchBNDMq
  hash-size: low
  bf-size: medium
- b2gm:
  search-algo: B2gSearchBNDMq
  hash-size: low
  bf-size: medium
- b2g:
  search-algo: B2gSearchBNDMq
  hash-size: low
  bf-size: medium
- b3g:
  search-algo: B3gSearchBNDMq
  hash-size: low
  bf-size: medium
- wumanber:
  hash-size: low
  bf-size: medium
```

```
defrag:
memcap: 32mb
```

hash-size: 65536

trackers: 65535

max-frags: 65535

prealloc: yes

timeout: 60

flow:

memcap: 32mb

hash-size: 65536

prealloc: 10000

emergency-recovery: 30

flow-timeouts:

default:

new: 30

established: 300

closed: 0

emergency-new: 10

emergency-established: 100

emergency-closed: 0

tcp:

new: 60

established: 3600

closed: 120

emergency-new: 10

emergency-established: 300

emergency-closed: 20

udp:

new: 30

established: 300

emergency-new: 10

emergency-established: 100

icmp:

new: 30

established: 300

emergency-new: 10

emergency-established: 100

stream:

memcap: 32mb

checksum-validation: yes

inline: yes

reassembly:

memcap: 64mb

depth: 1mb

toserver-chunk-size: 2560

toclient-chunk-size: 2560

host:

hash-size: 4096

prealloc: 1000

memcap: 16777216

default-rule-path: /etc/suricata/rules

rule-files:

- botcc.rules
- ciarmy.rules
- compromised.rules
- drop.rules
- dshield.rules
- emerging-activex.rules
- emerging-attack_response.rules
- emerging-chat.rules
- emerging-current_events.rules
- emerging-dns.rules
- emerging-dos.rules
- emerging-exploit.rules
- emerging-ftp.rules
- emerging-games.rules
- emerging-icmp_info.rules
- emerging-icmp.rules
- emerging-imap.rules
- emerging-inappropriate.rules
- emerging-malware.rules
- emerging-misc.rules
- emerging-mobile_malware.rules
- emerging-netbios.rules
- emerging-p2p.rules
- emerging-policy.rules
- emerging-pop3.rules
- emerging-rpc.rules
- emerging-scada.rules
- emerging-scan.rules
- emerging-shellcode.rules

- emerging-smtp.rules
- emerging-snmp.rules
- emerging-sql.rules
- emerging-telnet.rules
- emerging-tftp.rules
- emerging-trojan.rules
- emerging-user_agents.rules
- emerging-voip.rules
- emerging-web_client.rules
- emerging-web_server.rules
- emerging-web_specific_apps.rules
- emerging-worm.rules
- rbn-malvertisers.rules
- rbn.rules
- tor.rules
- decoder-events.rules
- stream-events.rules
- http-events.rules
- smtp-events.rules

classification-file: /etc/suricata/rules/classification.config

reference-config-file: /etc/suricata/rules/reference.config

vars:

address-groups:

HOME_NET: "[172.16.20.0/24]"
EXTERNAL_NET: "!\$HOME_NET"
HTTP_SERVERS: "\$HOME_NET"
SMTP_SERVERS: "\$HOME_NET"
SQL_SERVERS: "\$HOME_NET"
DNS_SERVERS: "\$HOME_NET"
TELNET_SERVERS: "\$HOME_NET"
AIM_SERVERS: "\$EXTERNAL_NET"
DNP3_SERVER: "\$HOME_NET"
DNP3_CLIENT: "\$HOME_NET"
MODBUS_CLIENT: "\$HOME_NET"
MODBUS_SERVER: "\$HOME_NET"
ENIP_CLIENT: "\$HOME_NET"
ENIP_SERVER: "\$HOME_NET"

port-groups:

HTTP_PORTS: "80"
SHELLCODE_PORTS: "!80"
ORACLE_PORTS: 1521
SSH_PORTS: 22
DNP3_PORTS: 20000

action-order:

- pass
- drop
- reject
- alert

asn1-max-frames: 256

engine-analysis:

rules-fast-pattern: yes

rules: yes

pcre:

match-limit: 3500

match-limit-recursion: 1500

libhttp:

default-config:

personality: IDS

request-body-limit: 3072

response-body-limit: 3072

request-body-minimal-inspect-size: 32kb

request-body-inspect-window: 4kb

response-body-minimal-inspect-size: 32kb

response-body-inspect-window: 4kb

double-decode-path: no

double-decode-query: no

server-config:

- apache:

address: [172.16.20.10]

personality: Apache_2_2

request-body-limit: 4096

response-body-limit: 4096

double-decode-path: no

double-decode-query: no

```
coredump:
max-dump: unlimited
```

APÊNDICE B – SCRIPT DE INICIALIZAÇÃO DO FIREWALL

```
#!/bin/sh
### BEGIN INIT INFO
# Provides: firewall
# Required-Start:  $network $remote_fs $syslog $time
# Required-Stop:   $network $remote_fs $syslog $time
# Default-Start:   2 3 4 5
# Default-Stop:    0 1 6
# Short-Description: Carrega o firewall
# Description: Carrega as regras do firewall que estão em /etc/firewall/rules
### END INIT INFO

# Este arquivo deve ser armazenado no diretório /etc/init.d/
# Os atributos devem ser 700
# Devem ser inicializados junto com a máquina:
#   Debian: insserv /etc/init.d/firewall

# biblioteca Source function
. /lib/lsb/init-functions

# Menu de opções:
case "$1" in
start)
    echo "\nIniciando o serviço..."
    /etc/firewall/rules
    echo "Serviços de firewall e roteamento operacionais conforme as regras
estabelecidas.\n"
    ;;
stop)
    echo "\nInterrompendo o serviço..."
    /etc/firewall/close
    echo "Serviços de firewall e roteamento interrompidos, com todas as portas
fechadas.\n"
    ;;
```

```
openall)
    echo "\nAbrindo todas as portas..."
    /etc/firewall/open
    echo "Serviço de roteamento operacional, com todas as portas abertas.\n"
    ;;
closeall)
    echo "\nFechando todas as portas..."
    /etc/firewall/close
    echo "Serviços de firewall e roteamento interrompidos, com todas as portas
    fechadas.\n"
    ;;
restart)
    $0 stop
    $0 start
    ;;
status)
    $0 cards
    $0 routes
    $0 rules
    ;;
cards)
    echo "\nApresentação das configurações das placas [ifconfig]:\n"
    ifconfig
    ;;
rules)
    echo "\nApresentação das regras de firewall [iptables]:\n"
    iptables -L
    ;;
routes)
    echo "\nApresentação das rotas [iproute]:\n"
    ip route list
    ;;
*)
    echo "\nSintaxe:"
    echo "firewall {start|stop|restart|openall|closeall|status|cards|rules|routes}\n"
    exit 1
esac
exit 0
```

APÊNDICE C – ARQUIVO RULES

```
#!/bin/bash
#Aqui foram colocadas regras apenas para testar o Suricata no modo NFQ e não
#para um cenário real em produção.

iptables -P INPUT DROP
iptables -F INPUT
iptables -P OUTPUT DROP
iptables -F OUTPUT
iptables -P FORWARD DROP
iptables -F FORWARD
iptables -t nat -F

modprobe ip_tables
modprobe iptable_nat
modprobe iptable_mangle
modprobe iptable_filter
modprobe ip_conntrack
modprobe ipt_LOG
modprobe ipt_limit
modprobe ipt_state

# habilita o funcionamento das conexoes internas: tcp, udp e icmp
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# habilita resolução de DNS e acesso aos repositórios pelo Suricata
iptables -A OUTPUT -o eth0 -p udp --dport 53 -j ACCEPT
iptables -A INPUT -i eth0 -p udp --sport 53 -j ACCEPT
iptables -A OUTPUT -o eth0 -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -i eth0 -p tcp --sport 80 -m state --state ESTABLISHED,RELATED -j
ACCEPT

#libera o compartilhamento NFS para o servidor
iptables -A INPUT -s 172.16.20.10 --dport 2049 -j ACCEPT
iptables -A OUTPUT -d 172.16.20.10 --sport 2049 -j ACCEPT

#libera SSH e SNORBY para máquina do administrador
iptables -A INPUT -s 192.168.40.35 -i eth0 -m multiport -p tcp --dports 22,80 -j
ACCEPT
```

```
iptables -A OUTPUT -d 192.168.40.35 -o eth0 -m state --state  
ESTABLISHED,RELATED -j ACCEPT
```

#Nat para a saída das redes DMZ e do Pytbull

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

#NFQ

```
iptables -A FORWARD -j NFQUEUE
```

APÊNDICE D – ARQUIVO OPEN

```
iptables -P INPUT ACCEPT  
iptables -F INPUT  
iptables -P OUTPUT ACCEPT  
iptables -F OUTPUT  
iptables -P FORWARD ACCEPT  
iptables -F FORWARD  
iptables -t nat -F  
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

APÊNDICE E – ARQUIVO CLOSE

```
iptables -P INPUT DROP  
iptables -F INPUT  
iptables -P OUTPUT DROP  
iptables -F OUTPUT  
iptables -P FORWARD DROP  
iptables -F FORWARD  
iptables -t nat -F
```

APÊNDICE F – ARQUIVO DATABASE.YML

```
snorby: &snorby  
  adapter: mysql  
  username: fulano  
  password: "password"
```

```

host: localhost

development:
  database: snorby
  <<: *snorby

test:
  database: snorby
  <<: *snorby

production:
  database: snorby
  <<: *snorby

```

APÊNDICE G – ARQUIVO SNORBY_CONFIG.YML

```

development:
  domain: localhost:3000
  wkhtmltopdf: /usr/bin/wkhtmltopdf

test:
  domain: localhost:3000
  wkhtmltopdf: /usr/bin/wkhtmltopdf

production:
  domain: localhost:3000
  wkhtmltopdf: /usr/bin/wkhtmltopdf

```

APÊNDICE H – ARQUIVO DEFAULT

```

<VirtualHost *:80>
  ServerAdmin webmaster@localhost
  ServerName ips.dominio.local
  DocumentRoot /var/www/snorby/public
  <Directory "/var/www/snorby/public">
    AllowOverride all
    Order deny,allow
    Allow from all
  </Directory>
</VirtualHost>

```

```
Options -MultiViews
</Directory>
</VirtualHost>
```

APÊNDICE I – ARQUIVO BARNYARD2.CONF

```
config reference_file:      /etc/suricata/reference.config
config classification_file: /etc/suricata/classification.config
config gen_file:           /etc/suricata/rules/gen-msg.map
config sid_file:           /etc/suricata/rules/sid-msg.map
input unified2
output alert_fast: stdout
output database: log, mysql, user=fulano password=password dbname=snorby
host=127.0.0.1 sensor_name=suricata-dmz
```

APÊNDICE J – ARQUIVO IPS

```
#!/bin/sh
### BEGIN INIT INFO
# Provides: ips
# Required-Start:    $network $remote_fs $syslog $time
# Required-Stop:     $network $remote_fs $syslog $time
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Inicializa o Suricata
# Description: Carrega o Suricata, Barnyard2 e atualiza as regras via oinkmaster
### END INIT INFO
case $1 in
  start)
    echo "starting $0..."
    suricata -c /etc/suricata/suricata.yaml -q 0 -D
    barnyard2 -c /etc/suricata/barnyard2.conf -d /var/log/suricata -f unified2.alert
    -w /var/log/suricata/suricata.waldo -D
    echo "done.\n"
    ;;
  stop)
    echo "stopping $0..."
    killall barnyard2
    killall Suricata-Main
```

```

    echo "done.\n"
    ;;
restart)
    $0 stop
    $0 start
    ;;
sincroniza)
    echo "sincronizando regras via oinkmaster $0..."
    oinkmaster -C /etc/oinkmaster.conf -o /etc/suricata/rules
    echo "done.\n"
    ;;
*)
    echo "usage: $0 (start|stop|restart|sincroniza)"
    ;;
esac

```

APÊNDICE K – ARQUIVO CONFIG.CFG

```

[CLIENT]
ipaddr          = 172.16.20.210
iface           = eth0
useproxy        = 0
proxyhost       =
proxyport       =
proxyuser       =
proxypass       =

[PATHS]
db              = data/pytbull.db
urlpdf          = http://dl.dropbox.com/u/30211728/
pdfdir          = pdf/malicious
pcapdir         = pcap
tempfile        = /tmp/pytbull.tmp
alertsfile      = /home/testeftp/suricata/fast.log

[ENV]
sudo            = /usr/bin/sudo
nmap            = /usr/local/bin/nmap
nikto           = /pentest/web/nikto/nikto.pl

```

```
niktoconf          = /pentest/web/nikto/nikto.conf
hping3             = /usr/sbin/hping3
tcpreplay          = /pentest/sniffers/tcpreplay/tcpreplay
ab                 = /usr/sbin/ab
ping               = /bin/ping
ncrack             = /usr/local/bin/ncrack
ncrackusers        = data/ncrack-users.txt
ncrackpasswords    = data/ncrack-passwords.txt
localhost          = 127.0.0.1
```

[FTP]

```
ftpproto          = ftp
ftpport           = 21
ftpuser           = testeftp
ftppasswd         = testeftp
```

[TIMING]

```
sleepbeforegetalerts = 2
sleepbeforenexttest  = 2
sleepbeforetwoftp    = 2
urltimeout           = 10
```

[SERVER]

```
reverseshellport    = 12345
```

[TESTS]

```
clientSideAttacks    = 0
testRules             = 1
badTraffic           = 1
fragmentedPackets    = 1
bruteForce           = 1
evasionTechniques    = 1
shellCodes            = 1
denialOfService      = 0
pcapReplay           = 1
normalUsage          = 1
ipReputation         = 1
```

[TESTS_PARAMS]

```
ipreputationnbtests = 10
```