UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

José Deivison Vieira Xavier

NEXT-GENERATION FIREWALL: implementando funcionalidades utilizando Snort e OpenAppID.

Belém
2018

José Deivison Vieira Xavier

NEXT-GENERATION FIREWALL: implementando funcionalidades utilizando Snort e OpenAppID.

Trabalho de Conclusão de Curso apresentado para obtenção do título de Bacharel em Sistemas de Informação pela Faculdade de Computação do Instituto de Ciências Exatas e Naturais da Universidade Federal do Pará.

Orientadora: Prof.ª.Dr ª Cássia Maria Carneiro Kahwage.

Belém
2018

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da Universidade Federal do Pará

X3n Xavier, José Deivison Vieira

Next-Generation Firewall: Implementando funcionalidades utilizando Snort e OpenAppID / José Deivison Vieira Xavier; orientadora, Cássia Maria Carneiro Kahwage - 2018.

77 p. il.

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Pará, Instituto de Ciências Exatas e Naturais, Faculdade de Computação, Curso de Sistemas de Informação, Belém, 2018.

1. Next-Generation Firewall. 2. Snort. 3. OpenAppID.

CDD 005.8

José Deivison Vieira Xavier

NEXT-GENERATION FIREWALL: implementando funcionalidades utilizando Snort e OpenAppID.

Trabalho de Conclusão de Curso apresentado para obtenção do título de Bacharel em Sistemas de Informação pela Faculdade de Computação do Instituto de Ciências Exatas e Naturais da Universidade Federal do Pará.

Data da aprovação: Belém-PA. _____/____/_____

Conceito:

Banca Examinadora

_____

Prof.ª.Dr ª Cássia Maria Carneiro Kahwage

Faculdade de Computação/ICEN/UFPA – Orientadora

_____

Prof.ª Danielle Costa Carrara Couto

Faculdade de Computação/UFPA – Membro

_____

Rômulo Pinto de Albuquerque

CTIC/UFPA – Membro

AGRADECIMENTOS

Agradeço minha esposa Josy, que ao longo desses meses me deu não só força, mas apoio para vencer essa etapa da vida acadêmica.

Aos meus ídolos, meus pais José Francisco e Maria Socorro, obrigado pelo amor incondicional e pelo exemplo de vida. Também sou grato aos meus avós, que me ensinaram valores importantes e contribuíram com a minha educação.

Ao meu irmão Denison, obrigado pelo apoio e torcida.

Ao meu filho Matheus (*in memoriam*), que mesmo com o pouco tempo que passou entre nós, marcou a vida de todos que souberam de sua existência.

A professora Cássia Kahwage que me ajudou muito nesse trabalho sendo minha orientadora, obrigado!

Agradeço à toda minha família, amigos, professores e pessoas que ajudaram em minha jornada acadêmica. Um agradecimento especial aos colegas Maykon, Marcos Paulo e Júlio. Sou imensamente grato pela paciência e incentivo.

RESUMO

A identificação do tráfego de aplicações é uma das principais funcionalidades presente em um Next-*Generation Firewall*. Este trabalho implementa a identificação do tráfego de aplicações catalogadas utilizando as ferramentas de código aberto *Snort* e *OpenAppID*, também utiliza a linguagem “Lua” para a identificação de aplicações não catalogadas. Foram realizados testes para demonstrar o funcionamento e limitações. Os resultados mostraram que é possível a identificação de aplicações utilizando ferramentas as citadas, com a possibilidade de incluir novas aplicações.

Palavras-Chaves: *Next-Generation Firewall*. *Snort*. *OpenAppID*.

ABSTRACT

Identify application traffic is one of the main features present in a Next-Generation Firewall. This work implements the identification of the traffic of applications cataloged using the tools open source Snort and OpenAppID, also uses the language "Lua" for the identification of applications not cataloged. Tests were performed to demonstrate the operation and limitations. The results showed that it is possible to identify applications using the aforementioned tools, with the possibility of including new applications.

Keywords: *Next-Generation Firewall*. *Snort*. *OpenAppID*.

LISTA DE SIGLAS

AD	Serviço da Microsoft que implementa o protocolo LDAP (*Active Directory*)
AppId	Identificadores de aplicativos (*Application identifiers*)
CIO	Diretor de TI (*chief information officer*)
CPU	Unidade Central de Processamento (*Central Process Unit*)
DARP	Agência de projeto de pesquisa avançada de defesa (*Defense Advance Research Project Agency*)
DDoS	Ataques Distribuído de Negação de Serviços (*Distributed Denial of Service*)
DLP	Prevenção contra perda de dados (*Data Loss Prevention*);
DOS	Ataques de Negação de Serviços (*Denial of Service*)
FTP	Protocolo de Transferência de Arquivos (*File Transfer Protocol*)
GPL	Licença Pública Geral (*General Public LicenseGeneral Public License*)
HD	Disco Rígido (*Hard Disk*)
HIDS	Sistema de Detecção de Intrusão Baseado em Host (*Host-based Intrusion Detection System)*
HTTP	Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*)
ICMP	Protocolo de Mensagens de Controle da Internet (*Internet Control Message Protocol*)
IDS	Sistema de Detecção de Intrusão (*Intrusion Detection System)*
IP	Protocolo da Internet (Internet *Protocol*)
IPS	Sistema de Prevenção de Intrusão (*Intrusion Prevention System)*
ISO	Organização Internacional para Padronização (*International Organization for Standardization*)
LAN	Redes Locais (*Local Area Networks*)
LDAP	Protocolo de acesso de diretório leve (Lightweight Directory Access Protocol)
LUA	Linguagem de script de multiparadigma
MTU	Unidade Máxima de Transmissão (*Maximum Transmission Unit*)
NAT	Tradução de Endereços de Rede (*Network Address Translation*)
NGFW	Firewall*Firewall* de próxima geração (*Next-Generation FirewallFirewall*)
NIDS	Sistema de Detecção de Intrusão Baseado em Rede (*Network-based Intrusion Detection System)*
OS	Sistema Operacional (*Operational System*)
PING	Utilitário que usa o protocolo ICMP para testar a conectividade entre equipamentos

SGSI	Sistema de Gestão de Segurança da Informação
SSL	Segurança da camada de Transporte (*Transport Layer Security*)
TCP	Protocolo de Controle de Transmissão (*Transmission Control Protocol*)
UDP	Protocolo de Datagrama de Usuário (*User Datagram Protocol*)
VM	Máquina Virtual (*Virtual Machine)*
VPN	Rede Privada Virtual (*Virtual Private Network*)
WAN	Rede de longa distância (W*ide Area Network)*

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

A segurança de rede é um dos aspectos decisivos na Gestão de Segurança da Informação, muitas organizações gastam milhões para protegerem seus ativos de informação. O Gartner estima que o investimento global em Segurança da Informação previsto para 2018 será de 93 bilhões de dólares, o que representa um aumento de 12% em comparação com 2017 (SEGINFO, 2017).

De acordo com Morais (2015, p. 14) “[...] As empresas estão a cada dia mais dependentes de processos de tecnologia da informação, e esse caminho não tem volta”. É evidente o alto nível de dependência da tecnologia da informação, onde as empresas são expostas a um elevado número de produtos e serviços na rede. A Internet é utilizada amplamente por diversos profissionais durante a jornada de trabalho, cada vez mais as tarefas são realizadas utilizando sistemas de informação.

Aplicações online que antes só eram utilizadas exclusivamente para interações pessoais, como Youtube e Facebook tornaram-se ferramentas essências paras a empresas. Canais de atendimentos foram migrados para o Twitter, bem como serviços de e-mail para o Gmail ou Office365. Essa é a nova realidade tanto em empresas de grande, médio e pequeno porte, as quais buscam através da alta conectividade acelerar as atividades principalmente com aplicações com interações com redes sociais e nuvens.

Este novo cenário tem o objetivo de contribuir com os processos de negócios das empresas, mas seu mau uso pode gerar inúmeros problemas e até mesmo prejuízos, as ferramentas de segurança devem estar preparadas para migração de acesso e compartilhamento de informação em rede sociais e nuvens de serviços, ferramentas de segurança inadequadas as quais podem ocasionar a perda de informações valiosas, desperdício de tempo, falta de foco e a baixa produtividade de funcionários.

Essa mudança dos processos de negócios gerou novos requisitos para a segurança da rede. O aumento das demandas de largura de banda e novas arquiteturas de aplicativos mudaram a forma como os protocolos são usados e como os dados são transferidos. As aplicações como de fluxo de vídeo ou que utilizam grande largura de banda devem ser controladas pelos administradores da rede como

medida de controle no uso democrático da largura de banda da rede, para evitar lentidão nas aplicações mais importantes para a empresa.

Visando realizar esse controle e gerenciamento geralmente utiliza-se um *Firewall*. Este é um dos elementos de grande utilização em um Sistema de Gestão da Segurança da Informação (SGSI), o qual é empregado para aumentar o nível de segurança e gerenciar a utilização dos recursos de uma rede de computadores, exercendo um papel fundamental no nível de isolamento nas redes existentes, realizando a filtragem e inspeção do tráfego de dados de uma rede pública para uma rede privada e de uma rede privada para uma rede pública.

E para enfrentar os desafios nesse novo cenário de redes, os *firewalls* precisaram evoluir para o que o Gartner denomina de "*firewalls* de próxima geração" (NGFW - Next Generation *Firewall*), também conhecidos como *firewall* de camada 7, onde a sua principal característica é o controle do fluxo de dados baseado em aplicação.

Segundo Miller (2016, p. 39, tradução livre):

> Os NGFWs classificam o tráfego pela identidade do aplicativo para permitir visibilidade e controle de todos os tipos de aplicativos - incluindo aplicativos da Web, SaaS e legados - em execução em redes organizacionais.

Mas estes tipos de *firewalls* geralmente possuem valores elevados e necessitam de investimento constante para manter a atualização de suas licenças e a necessidade de mão de obra treinada.

Este trabalho pretende exibir uma alternativa para solução desta problemática do controle do fluxo de aplicações que não seja baseada em solução proprietária e sim em uma solução livre, utilizando um conjunto de ferramentas onde se destaca o *OpenAppID*.

A ferramenta *OpenAppID* utiliza um módulo de processamento e uma linguagem de detecção aberta e focada em aplicativos para o *Snort*. Com o objetivo de identificar aplicativos e serviços com base em padrões ou regras predefinidas, armazenadas em um arquivo detector em linguagem Lua. A hipótese é que esta ferramenta possa ser utilizada para alertar, bloquear e executar análises contextuais de aplicativos utilizados na rede, permitindo a utilização ou não de um determinado aplicativo.

1.1. Motivação e Justificativa

A diversidade de aplicações e a adoção massiva da conectividade pelos usuários finais, por um lado, e a necessidade de gerenciamento otimizado de largura de banda e segurança por outro, alimentam a crescente busca de soluções de segurança. E um dos requisitos fundamentais em uma solução de segurança atualmente é a identificação de aplicações na rede, fornecendo informações importantes a nível de gerenciamento.

Diante dessa necessidade de ferramentas de segurança em rede, que fossem capazes de fazer a detecção de aplicações em uma rede de computadores, surgiram os *Next-Generation Firewalls* (NGFW). Essa nova geração de *firewall* tem como umas de suas principais funcionalidades a identificação de aplicações em uso na rede em tempo real. Produtos desta categoria são disponibilizados através de fabricantes de Appliance comerciais, que possuem valores elevados de investimento como a *Palo Alto*, *Checkpoint*, *Fortinet* e outros.

Este trabalho consistiu na implementação e utilização do pré-processador *OpenAppID*. Essa ferramenta de código aberto facilita o gerenciamento e controle de aplicações em uma rede, possibilitando a utilização de funcionalidades de um *firewall* de nova geração, sendo possível usufruir desta tecnologia sem custos elevados, pretendendo o estudo de alternativa de implementação de um NGFW baseado em ferramentas de GPL (*General Public License* ou Licença Pública Geral).

1.2. Objetivos

1.2.1. Geral

Identificar o fluxo de dados de uma aplicação utilizando a linguagem Lua do pré-processador *OpenAppID* do *Snort*.

1.2.2. Específicos

- Realizar estudos de revisão bibliográfica sobre as gerações de *firewall* e NGFW;
- Estudar e implantar o *Snort*;
- Estudar e implantar o pré-processador *OpenAppID*;
- Identificar padrões de tráfego da aplicação escolhida;

- Descrever fluxo da aplicação escolhida na linguagem do *OpenAppID*;
- Demostrar de forma prática, utilizando código aberto, as funcionalidades de um *firewall* de nova geração (NGFW). Criando regras orientadas a aplicação utilizando o *Snort* com pré-processador *OpenAppID*, para controle de aplicativos e como adicionar suporte a aplicações não catalogadas.

1.3. Metodologia

O método escolhido foi o de pesquisa experimental em ambiente parcialmente simulado e pesquisa bibliográfica, pois a pesquisa pretende descrever os tipos de *firewalls*, o sistema de detecção de intrusão e de prevenção de intrusão (IPS/IDS) *Snort*, o módulo de processamento aberto e focado no aplicativo para o *Snort* (*OpenAppID*) e a elaboração de regras orientadas a aplicação. A técnica de coleta de dados foi através da observação sistemática.

A pesquisa experimental em ambiente parcialmente simulado, teve como base à implantação de maquinas com o *firewall Netfilter/Iptables*, IPS/IDS *Snort* e o pré-processador *OpenAppID*, utilizando sistema operacional Linux. E *Wireshark* e aplicação “Mira”, utilizando sistema operacional Windows, ambas virtualizadas. Onde foi identificado padrões de tráfego da aplicação, elaboração de um detector na linguagem “Lua” para aplicação e a criação de regras no *Snort* para identificação da aplicação na rede virtualizada.

1.3.1. Delimitação do tema

O propósito deste trabalho não é desenvolver um *Next-Generation Firewall* (NGFW), mas sim propor a implementação de funcionalidades de um NGFW utilizando ferramentas de código aberto (*Open Source*), mais especificamente a identificação e o gerenciamento na utilização de aplicações em uma rede local.

1.4. Organização do Trabalho

Este trabalho está estruturado da seguinte maneira: o Capítulo 2 apresenta fundamentação teórica sobre Redes, *Firewall*, IPS/IDS *Snort* e *OpenAppID*. O Capítulo 3 apresenta o cenário de implantação, ferramentas e tecnologias utilizadas. O Capítulo 4 apresenta a detecção de aplicações não catalogadas. O trabalho é finalizado com o Capítulo 5, considerações finais.

2. FUNDAMENTAÇÃO TEÓRICA

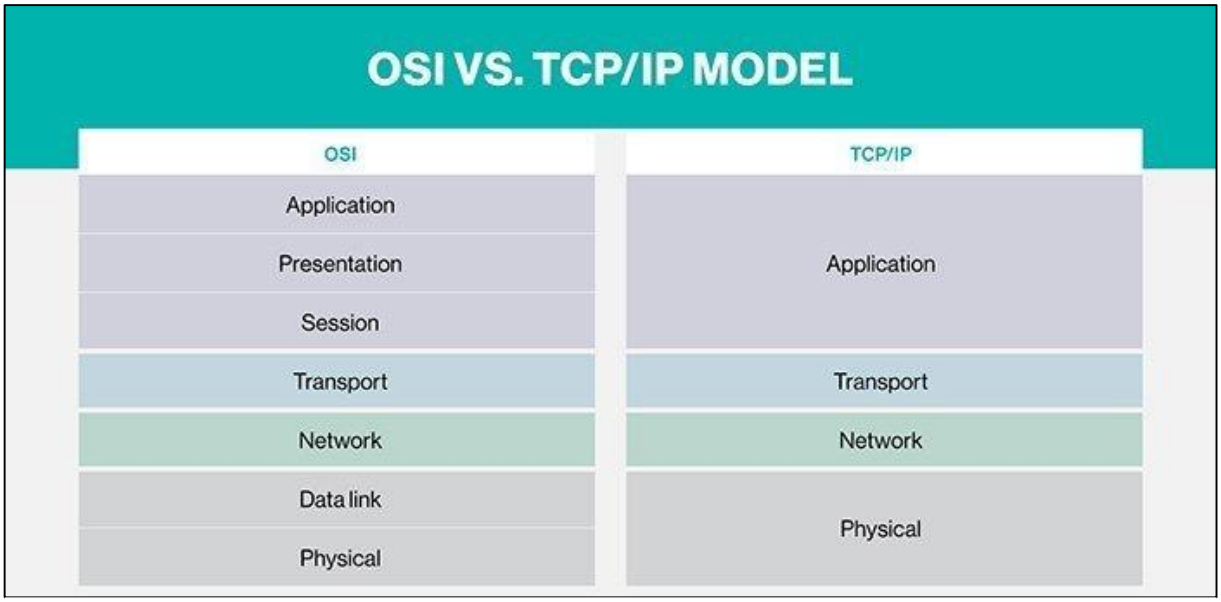
Este capítulo tem por objetivo descrever os principais conceitos, referentes à segurança da informação, redes, *firewall* e a identificação de tráfego que serão utilizados no ambiente experimental, isto é o conhecimento comum a qualquer administrador de rede que se aventure a implantar um NGFW baseado em soluções - GPL - *General Public License* (Licença Pública Geral) e software de código aberto (*Open Source Software*), este deverá conhecer as ferramentas aqui descritas para o melhor entendimento dos assuntos que serão abordados.

2.1. Arquitetura TCP/IP

O modelo TCP/IP é conjunto de protocolos usados nas redes para conectividades na troca de informações e compartilhamento de recursos mundialmente. Foi criado na década de 1970 pela DARPA (*Defense Advance Research Project Agency*) como um modelo de rede pública aberto, neutro para fornecedores. Assim como o modelo de referência OSI, o modelo TCP/IP fornece diretrizes gerais para projetar e implementar os protocolos de rede.

O modelo TCP/IP tem menos camadas do que o modelo OSI, apenas quatro. Essas camadas descrevem diferentes funções de rede e possuem seus próprios padrões e protocolos. As camadas são: Aplicação, Transporte, Internet e Rede, como podem ser observadas na figura 1 em comparação com o modelo OSI.

Figura 1 - Comparação entre modelos OSI e TCP/IP



Fonte: Techtarget (2018)

Miller (2014, tradução livre) explica que “os modelos OSI e o TCP/IP foram criados de forma independente. O modelo de rede TCP/IP representa a realidade no mundo, enquanto o modelo OSI representa um ideal”.

2.1.1. Pacotes

Toda comunicação em uma rede de computadores é segmentada em pequenos pacotes de acordo com a unidade máxima de transferência entre as redes (MTU), geralmente de 1500 *bytes*. Em cada uma das camadas, existem informações de cabeçalho, úteis ao processamento, além da parte de dados (*Payload*), onde a informação é de fato transportada.

Um pacote é uma unidade básica de comunicação em uma rede. Quando os dados precisam ser transmitidos, eles são divididos em estruturas semelhantes de dados antes da transmissão, chamados de pacotes, que são remontados ao bloco de dados original quando chegam ao seu destino. Um pacote contém uma origem, destino, dados, tamanho e outras informações úteis que ajudam o pacote chegar ao destino e a ser remontado apropriadamente.

Segundo Kurose e Ross (2014, p.3):

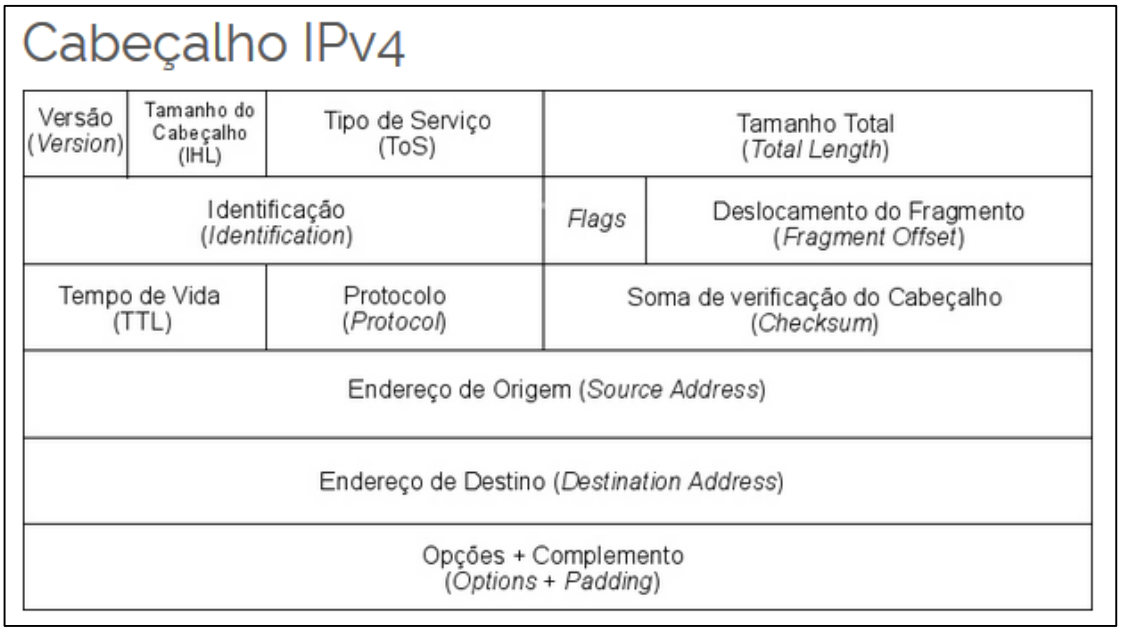
> Quando um sistema final possui dados para enviar a outro sistema final, o sistema emissor segmenta esses dados e adiciona bytes de cabeçalho a cada segmento. Os pacotes de informações resultantes, conhecidos como pacotes no jargão de rede de computadores, são enviados através da rede ao sistema final de destino, onde são remontados para os dados originais.

Um pacote IPv4 consiste em um cabeçalho de pacote IP (*Internet Protocol*), seguido por um cabeçalho de camada de transporte, seguido pela carga útil (*Payload)* ou dados do pacote.

2.1.2. Cabeçalho IP

O cabeçalho IP identifica as características do pacote na camada de rede, que implementa o protocolo IP. Contém informações de endereçamento lógico, como os endereços IP de origem e destino e a seção de carga útil, como pode ser observado na figura 2.

Figura 2 - Cabeçalho IPv4



Fonte: IPV6.BR (2018)

O cabeçalho IP contém também outras informações muito relevantes sobre o pacote, o quais são detalhados a seguir, baseado na figura 2 e na RFC 791:

- **Versão (*Version*):** Campo de 4 bits que indica a versão do cabeçalho utilizado;
- **Tamanho de Cabeçalho (*IHL*):** Campo de 4 bits que indica o comprimento do cabeçalho de Internet e todo o cabeçalho IP, aponta para o início dos dados;
- **Tipo de Serviço (*ToS*):** Campo de 8 bits usado para determinar a prioridade de cada pacote;
- **Tamanho Total (*Total Length*):** Campo de 16 bits que indica o comprimento total do pacote IP (incluindo cabeçalho IP e carga útil (*Payload*)).
- **Identificação (*Identification*):** Campo de 16 bits. Quando o pacote é fragmentado durante a transmissão, todos os fragmentos incluem o mesmo número de identificação para identificar o pacote IP original ao qual pertencem ao ser remontado.
- **Flags**: Campo de 3 bits. *Flags* são sinalizações, caso o pacote seja muito grande para manipular, as *flags* irão sinalizar se pode ser fragmentado ou não.

- **Deslocamentos do Fragmento (*Fragment Offset*):** Campo de 13 bits. O deslocamento irá dizer a posição exata do fragmento no pacote IP original.
- **Tempo de Vida (*TTL*):** Campo de 8 bits. Para evitar o loop na rede, todos os pacotes serão enviados com algum conjunto de valores TTL, que informa à rede quantos roteadores (saltos) esse pacote pode atravessar. Em cada salto, seu valor será decrementado por um e quando o valor chegar a zero, o pacote será descartado.
- **Protocolo (*Protocol*):** Campo de 8 bits. Informa a camada de rede no host de destino, qual protocolo este pacote pertence, ou seja, o protocolo de nível seguinte;
- **Soma de verificação do Cabeçalho (*Checksum*):** Campo de 16 bits usado para manter o valor da soma de verificação do cabeçalho, que é então usado para verificar se o pacote é recebido sem erros.
- **Endereço de Origem (*Source Address*):** Endereço de 32 bits do remetente do pacote.
- **Endereço de Destino (*Destination Address*):** Endereço de 32 bits do receptor do pacote.
- **Opções + Complemento (*Options + Padding*):** Campo opcional. Essas opções podem consistir em valores para opções como segurança, rota de registro, registro de data e hora, etc.

2.1.3. *Payload*

No contexto de rede de computadores*, payload* é a seção de carga útil de um pacote, ou seja, são os dados transportados dentro de um pacote. Quando os dados são enviados pela Internet, cada unidade transmitida inclui as informações do cabeçalho e os dados reais que estão sendo enviados. O cabeçalho identifica a origem e o destino do pacote, dentre outras informações, enquanto os dados reais são chamados de *payload*. Como as informações de cabeçalho são usadas apenas no processo de transmissão, elas são removidas do pacote quando chegam ao seu destino.

2.2. Aplicações de Rede

Aplicações de rede de computadores são softwares que utilizam a Internet ou outra infraestrutura de *hardware* de rede para executar tarefas. Estas utilizam uma arquitetura cliente-servidor, onde o cliente e o servidor são dois ativos conectados à rede, não necessariamente na mesma rede. O servidor está programado para fornecer algum serviço ao cliente. O cliente é tipicamente um *desktop*, *notebook* ou dispositivo portátil como *smartphones*. O servidor normalmente é um computador em um *data center*.

2.3. Segurança de Rede

A Internet evoluiu a partir das redes de organizações educacionais e governamentais, fortemente controladas, para um meio acessível a todos para a transmissão de comunicações pessoais e comerciais. Como resultado, os requisitos de segurança da rede foram alterados. Para essa nova realidade, dois tipos de problemas de segurança de rede devem ser resolvidos: a segurança da infraestrutura de rede e a segurança das informações (CISCO NETWORKING ACADEMY, 2016).

A segurança da infraestrutura de rede inclui a proteção física de dispositivos que fornecem a conectividade de rede e impedem o acesso não autorizado aos softwares de gerenciamento dos mesmos. A segurança da informação refere-se à proteção das informações contidas nos pacotes transmitidos pela rede e das informações armazenadas nos ativos conectados à mesma.

Para alcançar os objetivos de segurança de rede, existem três requisitos fundamentais, de acordo com os três pilares da segurança da informação, correspondente a sigla CID (Confidencialidade, Integridade e Disponibilidade), que é derivada do inglês CIA - *Confidentiality, Integriy and Availability*) - (CISCO NETWORKING ACADEMY, 2016):

- Garantir a confidencialidade: a confidencialidade dos dados significa que apenas os destinatários autorizados (pessoas, processos ou dispositivos) podem acessar e ler os dados. Isto é possível através da implementação de um sistema robusto de autenticação de usuários e criptografia dos dados para que apenas os destinatários possam ter acesso.

- Manter a integridade da comunicação: a integridade dos dados está relacionada a ter a segurança de que a informação não foi alterada durante a transmissão da origem para o destino. A integridade dos dados pode ser colocada em risco se a informação estiver alterada, seja voluntária ou acidentalmente.
- Garantir a disponibilidade: a disponibilidade está relacionada com a garantia de que os usuários autorizados terão acesso aos serviços de dados de maneira confiável e oportuna quando necessário. A disponibilidade está relacionada com a infraestrutura tecnológica, como a manutenção *hardwares*, *softwares* e a utilização da largura banda compatível com as necessidades, visando à manutenção e preservação do acesso aos dados.

2.4. Ataques de Redes

Um ataque é qualquer tentativa de acesso ou alteração de dados armazenados dentro de uma rede. Geralmente categorizados como ataques passivos e ativos, no entanto, em uma organização há três tipos de ataques que podem causar problemas de confidencialidade, integridade e disponibilidade (STALLINGS, 2015):

- 1 - Roubo de informações: É um tipo de ataque passivo que envolve roubar dados confidenciais da organização.
- 2 - Alteração de Informação: É um ataque ativo, o invasor modifica os registros da organização ou cria registros falsos que podem causar danos no futuro.
- 3 - Negação de Serviço (DOS): É um ataque cibernético, o atacante procura tornar um recurso de rede indisponível para o usuário legítimo. No ataque de negação de serviço, o invasor inunda o servidor de rede com o tráfego, que trava o servidor de rede. DOS (*Deny of Service*) não é malicioso, o único objetivo deste tipo de ataque é sufocar a rede para que ninguém possa acessá-la.

2.5. *Firewall*

Sobre *firewall*, Kurose e Ross (2014, p.538) afirmam que:

> Um *firewall* é uma combinação de hardware e software que isola a rede interna de uma organização da Internet em geral, permitindo que alguns pacotes passem e bloqueando outros. O *firewall* permite a um administrador de rede controlar o acesso entre o mundo externo e os recursos da rede que ele administra, gerenciando o tráfego de rede para esses recursos.

Um *firewall* pode ser um dispositivo ou sistema projetado para bloquear o acesso não autorizado de dentro ou de fora de uma rede privada. A maior funcionalidade do *firewall* é a filtragem, em outras palavras, o *firewall* tem a função de desviar o tráfego em relação a políticas pré-definidas e, por isso, pode proteger o sistema ou uma rede de ataques.

Para Morais (2015, p.29), o *firewall* atua de seguinte maneira:

> O *firewall* é um sistema que atua como ponto único de defesa entre a rede privada e a rede pública. Ele pode ainda controlar o tráfego entre as sub-redes de uma rede privada. Basicamente, todo o tráfego de entrada e saída da rede deve passar obrigatoriamente por esse sistema de segurança. O *firewall* pode autorizar, negar e registrar tudo que passa por ele.

Geralmente aplica-se no *firewall* o modelo de controle passivo (“negar por padrão”), o qual estabelece que somente o tráfego definido na política de *firewall* seja permitido na rede, qualquer outro tráfego é negado. O *firewall* também pode fornecer detalhes do tráfego de rede e ajudar a identificar ameaças de segurança. Os recursos típicos dos *firewalls* tradicionais incluem filtragem de pacotes, conversão de endereços de rede e de porta, inspeção com informações de estado e suporte à rede virtual privada (VPN). Os recursos típicos de prevenção contra intrusões incluem assinaturas e heurísticas de vulnerabilidade e ameaças. Abaixo são detalhadas as funções de um *firewall* de acordo com Stalling e Brown (2014):

- Gerenciar e controlar o tráfego da rede, seja ele de entrada ou de saída, concedendo ou negando acesso com base em regras pré-estabelecidas;
- Autenticar acesso, não permite que usuários não autorizados obtenham acesso a uma rede;
- Proteger os recursos de redes, como: servidores Web, servidores de email ou ativos que contenham dados confidenciais que podem causar danos à organização, caso sejam comprometidos.
- Gerenciamento de Rede Privada Virtual (VPN- *Virtual Private Network*). Uma VPN é uma tecnologia usada para criar uma conexão segura (rede privada) através da Internet pública. A implementação da VPN requer

criptografia para manter a segurança da rede e todos os dispositivos de rede devem suportar o mesmo nível de criptografia para se comunicar uns com os outros.

- Tradutor de endereços de rede (NAT - *Network Address Translator*). A conversão de endereços de rede (NAT) permite conectar vários computadores à Internet usando um único endereço IP público. Os *firewalls* geralmente têm a funcionalidade de conversão de endereços de rede (NAT) e os ativos de rede protegidos atrás de um *firewall* normalmente têm endereços no intervalo de endereços privados. Essa funcionalidade dos *firewalls* permite ocultar os verdadeiros endereços dos ativos de rede protegidos.

Os tipos de *firewall* e seu uso dependem do tamanho da rede e dos requisitos funcionais. A Techtarget (2017) classifica o *firewall* em quatro categorias: *Packet-Filtering* (Filtros de pacotes), *Stateful Inspection* (Filtros de estado), *Proxy Firewall* e *Next Generation Firewalls* (NGFW).

2.5.1. Filtros de pacotes (*Packet-Filtering*)

Esses tipos de *firewalls* sequenciam os pacotes relativos às regras permitir ou negar. Isso é feito por meio de informações de campos no cabeçalho do pacote, como: tipo de protocolo, endereço de host origem e destino ou número de porta de origem ou destino, etc. Então a análise não é profunda, ou seja, a detecção de código mal-intencionado não é realizada, e cada pacote é examinado como uma única entidade.

Morais (2015, p. 33), afirma que:

> Os filtros de pacotes atuam verificando apenas os endereços IP e as portas TCP/UDP. Esses *firewalls* trabalham com uma lista de controle de acesso, também conhecida como “Access List”, que é verificada antes que um pacote seja encaminhado para a rede interna. A lista relaciona o tráfego que é permitido e o que deve ser bloqueado.

2.5.2. Filtros de estado (*Stateful Inspection*)

Dentre os vários tipos de *firewall* no mercado, o *firewall* com inspeção de estado é o mais tradicional, no qual as regras são baseadas principalmente nos parâmetros do TCP/IP: endereço de origem e destino, tipo de protocolo, porta, etc. Esses tipos de *firewalls* são uma forma mais avançada dos *firewalls* de filtragem de pacotes, também conhecidos como *firewalls Stateful*, pois são acrescidos da funcionalidade de controle

de estados, o que possibilita rastrear o estado da rede e as conexões ativas que ela possui, como fluxos TCP ou comunicação UDP.

O *firewall Stateful* guarda o estado das conexões ativas em uma tabela, permitindo apenas a passagem de pacotes que pertençam a essas conexões, implementando uma rastreabilidade de parâmetros utilizados para validar uma conexão ativa que está sendo utilizada, monitorando os endereços IP de origem e destino, portas e outras informações da conexão. A tabela de estados sempre é atualizada para garantir a continuidade de segurança e integridade.

Portanto pode-se considerar que “os filtros de estado rastreiam conexões TCP e usam esse conhecimento para tomar decisões sobre filtragem [...]” (Kurose e Ross, 2014, p. 540).

2.5.3. Proxy *Firewall*

Um *Proxy Firewall* também conhecido como *gateway* de aplicação opera na camada de aplicação inspecionando os quadros em um protocolo de aplicação (*Telnet*, FTP, HTTP, etc.), onde ambas as extremidades de uma conexão são forçadas a conduzir a sessão por meio do *Proxy*. É criado e executado um processo no *firewall* que espelha um serviço como se estivesse sendo executado no host final. Desta forma, são impedidas as conexões diretas entre os sistemas dos dois lados do *firewall*. O *Proxy Firewall* centraliza toda a atividade de um aplicativo em um único servidor.

Kurose e Ross (2014, p. 542), explicam:

> Um gateway de aplicação é um servidor específico de aplicação, através do qual todos os dados da aplicação (que entram e que saem) devem passar. Vários gateways de aplicação podem executar no mesmo hospedeiro, mas cada gateway é um servidor separado, com seus próprios processos.

2.5.4. Next Generation *Firewall* (NGFW)

O NGFW é um ativo que além de realizar os controles de filtragem do fluxo de informações também possui função de controle do fluxo de informações por tipo de aplicações e aplicações propriamente ditas.

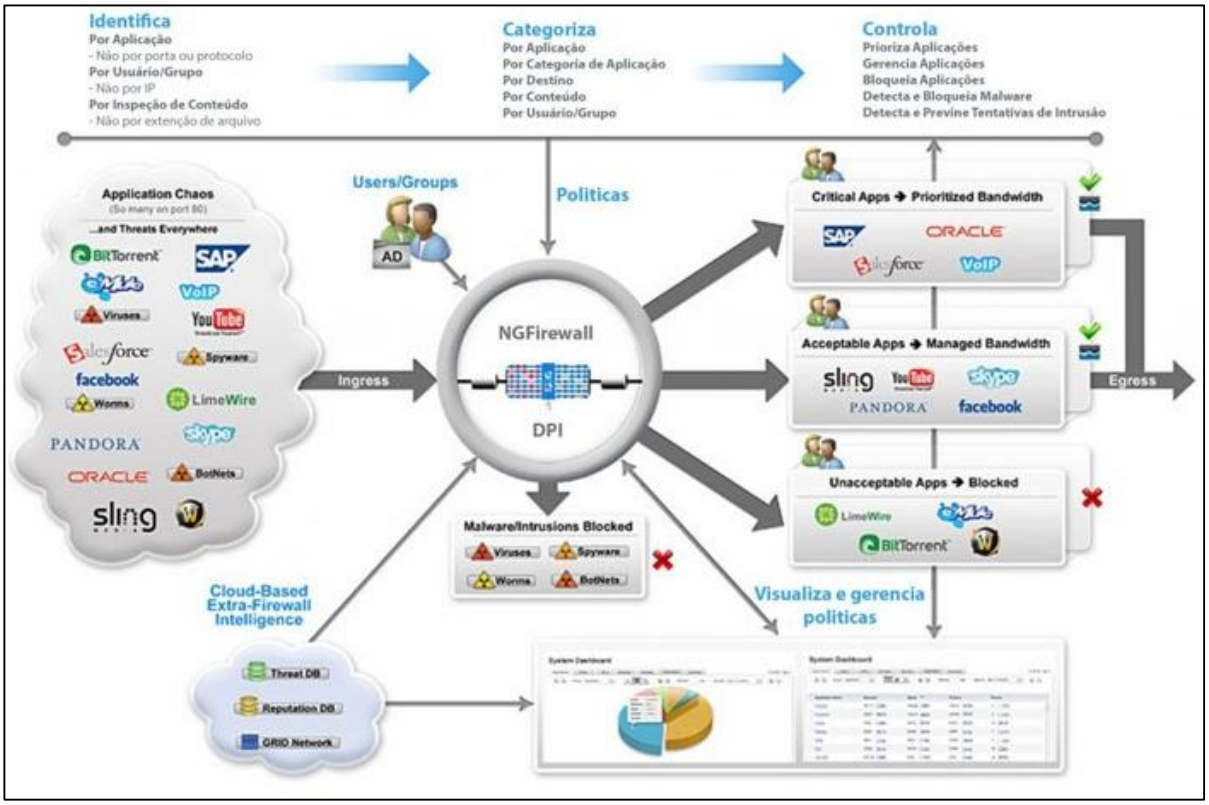
Os *firewalls* tradicionais mais utilizados(*Stateful*), limitados à inspeção de pacotes por estado e às regras de controle de acesso, deixaram de ser eficazes à medida que os ataques tornaram-se cada vez mais sofisticados e as ameaças mais avançadas, assim surgiram os *Next Generation Firewall* (NGFW). Sendo este capaz

de oferecer um nível mais profundo de segurança de rede, a fim de proteger uma empresa contra ameaças em constante evolução.

> “Não se trata de uma nova tecnologia, mas sim do aprimoramento do *firewall* tradicional, ou seja, um *Firewall* de Nova Geração vai realizar todas as tarefas de um *firewall* tradicional mais uma série de novos recursos que o equipamento padrão não executa.” (MORAES, 2015, p. 51).

Ainda de acordo com Moraes (2015), o NGFW é um dispositivo de segurança de rede de alto desempenho que adiciona prevenção de intrusão, visibilidade de aplicativos e usuários, inspeção de SSL (*Secure Socket Layer*) e detecção de ameaças desconhecidas para o *firewall* tradicional. Também possui integração de antivírus e capacidade de comunicação com serviços de terceiros, como o AD (Diretório Ativo da *Microsof*t) ou o LDAP (*Lightweight Directory Access Protocol*) usado em sistemas *Linux/Unix*. A figura 3 mostra o fluxo de operação em um NGFW, onde ilustra diversas aplicações solicitadas, as quais são inspecionadas e aplicadas políticas por usuários ou grupos de usuários, por prioridade de tráfego, por grupo de aplicações e políticas para barrar tráfego não autorizado ou malicioso. Também é possível visualizar a permissão, bloqueio a categorização de aplicações de acordo com políticas preestabelecidas.

Figura 3 - Visão geral do NGFW



Fonte: COSTELLO (2018)

Um NGFW em essência é um dispositivo cuja função é gerenciar a segurança entre redes LAN permitindo ou negando conexões, mas vai além, com funcionalidades avançadas, como “[...] recursos de *firewall* corporativo, um sistema de prevenção de intrusões (IPS) e controle de aplicativos.” (ROUSE, 2014), bloqueando o tráfego que pode explorar vulnerabilidades.

Principais funcionalidades presentes em um NGFW:

- Sistema de Prevenção de Intrusão (IPS);
- Controle de Aplicações;
- Prevenção contra perda de dados (DLP - *Data Loss Prevention*);
- Autenticação de Usuários;
- Gerenciador VPN;
- Antivírus;
- Filtrador Web.

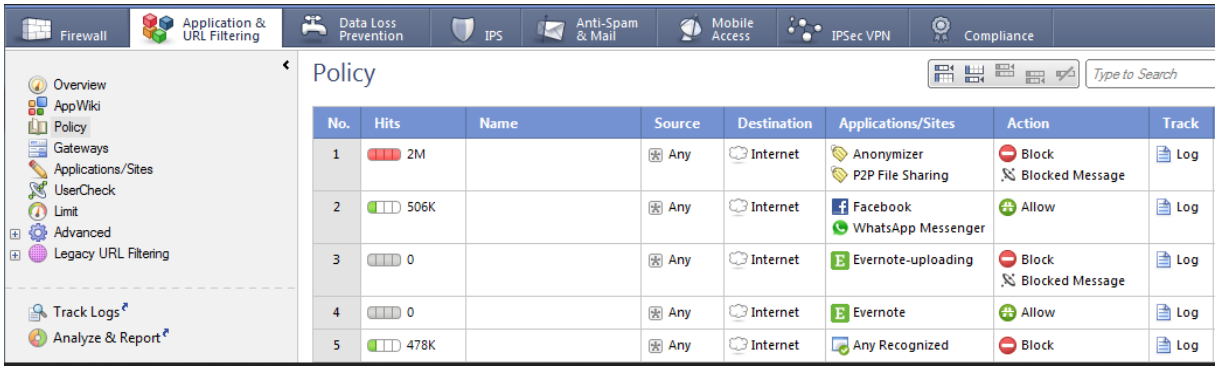
Miller (2016, p. 34), aponta os requisitos funcionais para um NGFW eficaz, que inclue a capacidade de:

- Identificar aplicações independentemente da porta, protocolo, técnicas evasivas ou criptografia SSL;
- Permitir visibilidade e controle granular baseado em políticas sobre aplicativos;
- Identificar com precisão os usuários e, subsequentemente usar sua identidade como atributo para o controle de políticas;
- Prover proteção em tempo real contra uma ampla gama de ameaças, incluindo aquelas que operam na camada de aplicação;
- Integrar, não apenas combine *firewall* tradicional de rede e capacidade de prevenção de intrusão;

Mudanças recentes no comportamento dos aplicativos e nos padrões de uso reduziram continuamente a proteção fornecida pelos *firewalls* tradicionais. Os usuários acessam qualquer aplicativo, de qualquer local, em muitas ocasiões, para fazer seu trabalho. Muitos desses aplicativos usam portas não padrão, portas intermediárias ou usam criptografia para simplificar e facilitar o acesso do usuário e evitar o *firewall*. O resultado é que os *firewalls* tradicionais, que se baseiam em portas e protocolos, não conseguem identificar ou controlar com eficiência o fluxo de aplicativos e ameaças que circulam pela rede.

A figura 4 mostra a tela de gerenciamto de um NGFW da fabricante Chekpoint, onde é possviel observar regras orientadas a aplicações de redes, como Facebook e WhastApp.

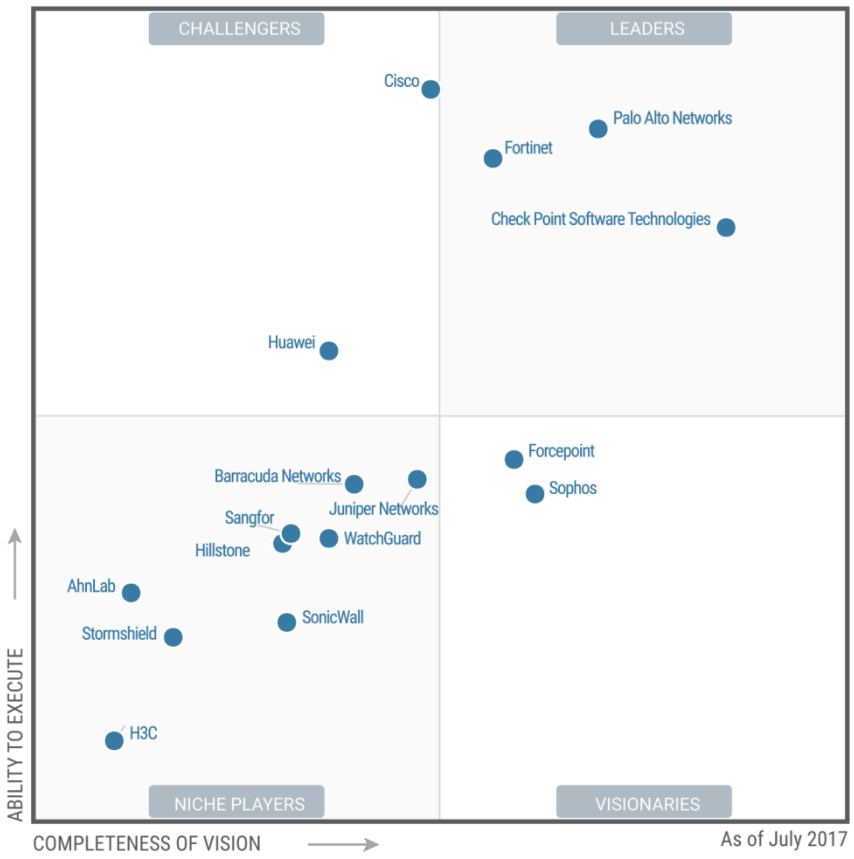
Figura 4 - Tela de gerenciamento NGFW Checkpoint



Fonte: Checkpoint (2018)

As fabricantes de NGFW líderes do mercado são *Palo Alto* Networks, *Fortinet* e *Checkpoint*, como mostra o quadrado mágico do Gartner em 2017 na figura 5:

Figura 5 – Quadro mágico do Gartner em 2017



Fonte: Forcepoint (2017)

2.6. Netfilter/Iptables

Nos sistemas operacionais *Linux*, o *Iptables* é uma ferramenta de *firewall* amplamente utilizada que faz interface com a estrutura de filtragem de pacotes do *kernel*, *Netfilter*. O *Netfilter* é um utilitário no *Kernel Linux* 2.4 (e versões posteriores) que facilita a conversão de endereços de rede (NAT), a filtragem e a manipulação de pacotes. O *Netfilter* e o *Iptables* são frequentemente relacionados na expressão única *Netfilter/Iptables* como uma maneira de se referir ao subsistema *Linux* para NAT, *firewall* e processamento avançado de pacotes. Essa estrutura de filtragem de pacotes é a base para a maioria das soluções de *firewall* nos servidores *Linux,* como nas distribuições *Ubuntu, Debain*, *Slackware* entre outros.

O *Netifilter/Iptables* tem como principais características a realização de filtragem de pacotes sem estado e com estado (IPv4 e IPv6), tradução de rede e porta, infraestrutura flexível e extensível através de APIs para extensão de terceiros. Assim é possível criar *firewalls* de Internet baseados em filtragem de pacotes com

estado(*statefull*), usar NAT e mascaramento para compartilhar o acesso à *Internet* e realizar a manipulação de pacotes (NETFILTER.ORG, 2018).

2.7. IDS (Intrusion Detection System)

Segundo Stalling e Brown (2014, p. 278) apud RFC 2828 (*Internet Security Glossary*) define:

> "**Intrusão de segurança:** Um evento de segurança ou uma combinação de vários eventos de segurança, que constitui um incidente de segurança no qual um intruso obtém ou tenta obter acesso a um sistema (ou recurso de sistema) sem ter a devida autorização.
>
> **Detecção de intrusão:** Um serviço de segurança que monitora e analisa eventos de sistema com a finalidade de descobrir e avisar em tempo real ou quase em tempo real que estão ocorrendo tentativas de acesso a recursos de sistema de modo não autorizado."

Um Sistema de Detecção de Intrusão (IDS) é um dispositivo ou aplicativo de software que monitora uma rede em busca de atividades de tráfego malicioso ou violações de políticas de segurança e produz relatórios para um centro de gerenciamento, alertando o administrador da rede sempre que houver uma tentativa invasão ou quando um ataque conseguiu passar pela segurança da rede.

2.7.1. Categorias de IDS

O IDS pode ser classificado em dois tipos principais, em relação à sua funcionalidade os baseados em Host (*Host-based Intrusion Detection Systems* - HIDS) e o baseados em redes (*Network-based Intrusion Detection System* -NIDS):

- HIDS: O Sistema de Detecção de Intrusão baseado em *host* usa informações de eventos de máquinas locais para detectar a intrusão. Verificando e analisando os arquivos de logs do sistema operacional. Se encontrar qualquer atividade suspeita, o IDS tratará como um ataque. O HIDS pode fornecer informações precisas sobre quais operações são executadas em um host na rede.
- NIDS: O Sistema de Detecção de Intrusão baseado em *Netrwork* analisa o tráfego de rede para detectar ameaças. Verificando todos os pacotes recebidos e procurando por padrões suspeitos. Quando uma ameaça é detectada, notifica-se o administrador.

2.7.2. Métodos de detecção

O IDS usa várias tecnologias diferentes para detectar atividades mal-intencionadas. Em relação à tecnologia utilizada para detecção de ameaças, duas são mais amplamente utilizadas, a detecção baseada em assinaturas (conhecimento) e a detecção baseada em anomalia (comportamento):

- Conhecimento: O IDS baseado em assinaturas se baseia em um banco de dados que reconhece a assinatura de vulnerabilidades já identificadas anteriormente. Uma assinatura é uma regra que examina um pacote ou uma série de pacotes para determinados conteúdos, como correspondências no cabeçalho do pacote ou informações de carga útil de dados.
- Comportamento: o IDS baseado em anomalias detecta o comportamento anormal nos sistemas de computadores e redes, deste modo são criados perfis usando métricas que podem incluir taxa de tráfego, número de pacotes para cada protocolo, caso o tráfego seja diferente das métricas do perfil traçado a alerta e direcionado para o administrador da rede, etc.

Para Kurose e Ross (2014, p. 544):

> Um IDS pode ser usado para detectar uma série de tipos de ataques, incluindo mapeamento de rede (provindo, por exemplo, de nmap), varreduras de porta, varreduras de pilha TCP, ataques de DoS, (Deny of Service) ataques de inundação de largura de banda, worms e vírus, ataques de vulnerabilidade de Sistemas Operacionais e ataques de vulnerabilidade de aplicações.

2.7.3. Componentes do IDS

Segundo Stalling e Brown (2014), a arquitetura geral de um IDS pode ser vista como conjunto de componentes, que são descritos a seguir:

- Fonte de dados: Dados recebidos pelos IDS para detectar atividades, como pacotes de redes;
- Sensor e Analisador: Monitoram e analisam a atividades na rede;
- Administrador: Ser humano responsável pela configuração das políticas de segurança;
- Gerente: É um dispositivo que recebe informações de sensores e os gerencia, realizando análises nas informações recebidas e podem identificar incidentes;
- Operador: O ser humano que é o usuário primário do gerenciador. Monitora a saída do IDS e inicializa ou recomenda ações adicionais.

O IDS é focado principalmente na identificação de possíveis incidentes e registro de informações. Este também é utilizado para outros fins, como identificar problemas com políticas de segurança, documentar ameaças existentes e determinar quais usuários violam as políticas de segurança, tornando-se uma adição necessária à infraestrutura de segurança de rede de uma organização.

2.8. IPS

O Sistema de Prevenção de Intrusão (*Intrusion prevention system* - IPS) é uma tecnologia de prevenção que detecta a intrusão e toma medidas para evitar que o intruso explore fraquezas na rede, um aplicativo ou o software de um sistema operacional. “[...] um IPS incorpora as funcionalidades de um IDS, mas também inclui mecanismos capazes de bloquear o tráfego de intrusos”. (STALLINGS, 2015, p. 407).

Kurose e Ross (2014, p. 544), explicam a diferença:

> Um dispositivo que gera alerta quando observa tráfegos potencialmente mal-intencionados é chamado de sistema de detecção de invasão (IDS, do inglês Intrusion Detection System). Um dispositivo que filtra o tráfego suspeito é chamado de sistema de prevenção de invasão (IPS, do inglês Intrusion Prevention System).

Tanto o IDS como o IPS capturam e analisam pacotes de rede e comparam o conteúdo a um banco de dados de ameaças conhecidas. A principal diferença entre eles é o que acontece após a detecção. O IDS é uma ferramenta de detecção e monitoramento que não age por conta própria. O IPS é um sistema de controle que aceita ou rejeita um pacote baseado no conjunto de regras.

2.9. *SNORT*

O *Snort* é uma ferramenta que implementa a detecção de intrusão (IDS) ou Prevenção de Intrusão (IPS) de código aberto, criado em 1998 por Martin Roesch, fundador e diretor de tecnologia da Sourcefire[1]. Atualmente o *Snort* é desenvolvido pela Cisco, que comprou a Sourcefire por US $ 2,7 bilhões em julho de 2013. E Martin Roesch passou a ser arquiteto chefe de segurança na Cisco.

O *Snort* é um mecanismo que inspeciona o tráfego de redes em tempo real e registra pacotes para executar análises, usando uma linguagem de regras flexível

[1] Foi uma empresa de tecnologia que desenvolveu hardware e software de segurança de rede.

para determinar os tipos de tráfego de rede que devem ser coletados. “O *Snort* pode detectar uma variedade de ataques e sondagens, tendo como base um conjunto de regras configurado por um administrador de sistema”. (STALLING; BROWN, 2014, p. 302).

O *Snort* está disponível sob a *General Public License* (GNU) e é gratuito para uso em qualquer ambiente de trabalho, tornando-se um sistema IDS/IPS mais acessível em comparação com as opções comerciais. O *Snort* oferece regras VRT[2], regras da Comunidade GPLv2, regra abertas de Ameaças Emergentes, detectores *OpenAppID* e regras para detecção de aplicações. O *Snort* VRT Rules requer assinatura paga, mas o usuário também pode se inscrever por 30 dias de teste. As regras da Comunidade da GPLv2 e a regras abertas de Ameaças Emergentes estão disponíveis gratuitamente.

2.9.1. Modos de Operação

O modo de operação definirá se o *Snort* atuará com um IDS ou IPS:

- Modo Passivo: Quando o *Snort* está no modo Passivo, ele aceitara pacotes utilizando a biblioteca *Libpcap*[3] e atuará como um IDS, realizando analise e detecção no tráfego de rede.
- Modo *Inline*: Quando o *Snort* está no modo *Inline*, atuará como um IPS permitindo que as regras de negação sejam acionadas. Nesse modo, o *Snort* aceita pacotes do *Iptables* via *Libipq* (*Linux*) ao invés do *Libpcap*. Assim é possível utilizar os tipos de regras (*drop, sdrop, reject)* para informar ao *Iptables* se o pacote deve ser descartado, rejeitado, modificado ou permitido passar com base em um conjunto de regras.
- Modo *Inline-Test*: Esse modo simula o modo *Inline* de *Snort*, permitindo a avaliação do comportamento Inline sem afetar o tráfego.

2.9.2. Arquitetura do *Snort*

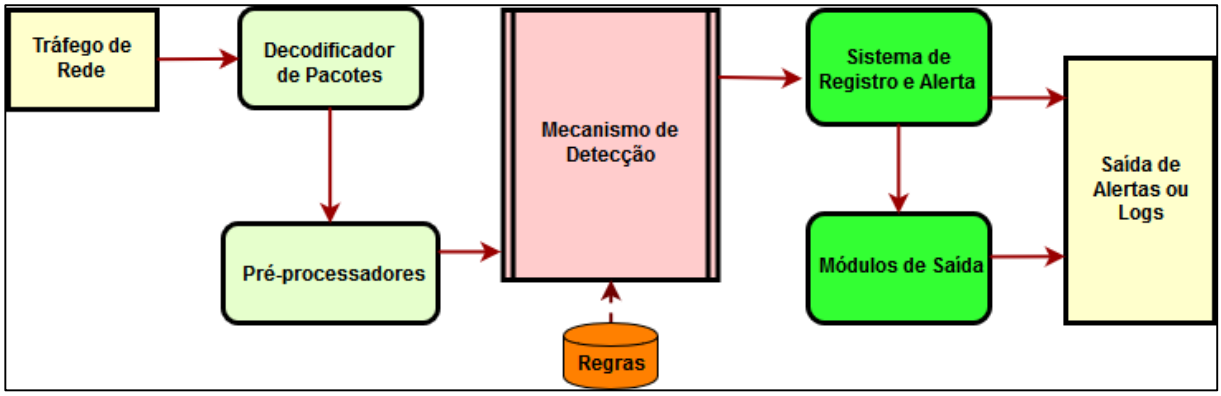
O *Snort* utiliza um mecanismo de captura e inspeção que funciona como um comutador de pacotes. Seu mecanismo de inferência é baseado em regras de

[2] Talos Vulnerability Report. Grupo de especialistas em segurança de rede que trabalham para descobrir, avaliar e responder proativamente às mais recentes tendências em atividades de hackers, tentativas de invasão, malware e vulnerabilidades.
[3] Biblioteca para capturar o tráfego de rede.

assinatura. As assinaturas são características de incidentes já ocorridos e coletados, então para a verificação, o tráfego é comparado com conjunto de assinaturas, os que são identificados são prováveis ameaças. O *Snort* busca por padrões de correspondência com a base de assinaturas com objetivo de detectar uma variedade de ataques.

A arquitetura de *Snort* está focada no desempenho, simplicidade e flexibilidade. O decodificador de pacotes, pré-processadores, mecanismo de detecção, sistema de registro e alerta e o modulo de saída são os cinco subsistemas primários do *Snort* (WANG, 2017). A figura 6 representa a arquitetura do *Snort*:

Figura 6 - Fluxo de processamento da arquitetura Snort



Fonte: Adaptado de Wang (2017)

Detalhando a arquitetura de processamento do *Snort* segundo Wang (2017), temos:

- **1 - Decodificador de pacotes:** recebe os pacotes da interface de rede e realiza análise inicial, decodificando os pacotes para determinar as características básicas de rede, como endereços e portas de origem e destino.
- **2 - Pré-processadores:** são elementos instalados como *plug-ins*[4] que podem ser utilizados para organizar ou modificar pacotes de dados antes que o mecanismo de detecção faça qualquer operação. Essa etapa pode envolver reordenar e remontar fragmentos de IP e segmentos TCP. O código do pré-processador é executado antes que o mecanismo de detecção seja chamado, mas após o pacote ter sido decodificado.

[4] Um plug-in é um complemento de software instalado para aprimorar seus recursos.

- **3 - Mecanismo de detecção:** é o principal elemento do *Snort*, este componente realiza a busca de características no tráfego usando um banco de assinaturas. Todas as regras são comparadas com um pacote antes de gerar um alerta. Se as regras corresponderem aos pacotes de dados, a regra de prioridade mais alta é selecionada para gerar o alerta.
- **4 - Registro e Alerta:** a etapa de registro é onde o *Snort* salva qualquer informação pertinente resultante das etapas anteriores.
- **5 - Módulos de saída:** são utilizados para enviar alertas via *pop up* para sistemas Windows, via *socket* para *UNIX* ou para uma base de dados.

2.9.3. Regras do *Snort*

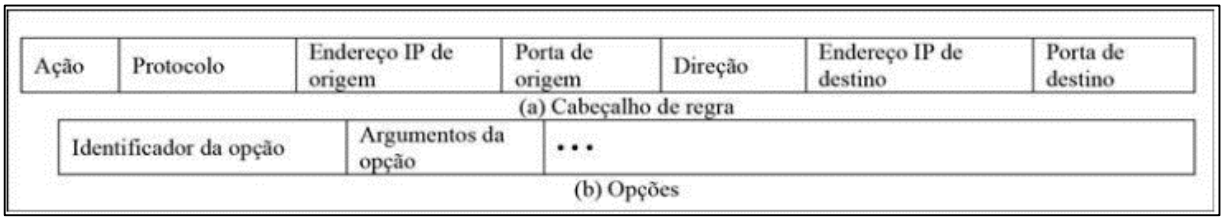
O *Snort* usa uma linguagem de descrição de regras simples, flexível e eficiente. Uma regra é um conjunto específico de palavras-chave e argumentos usados como critérios de correspondência para identificar violações de políticas de segurança. O mecanismo de detecção compara os pacotes com as condições especificadas em cada regra, se os dados do pacote corresponderem as condições especificadas em uma regra, esta regra será acionada.

A seguir é mostrado um exemplo de uma regra básica do *Snort*, onde será gerado um alerta sempre que for detectada uma solicitação ICMP *Echo request (ping)* ou uma mensagem *Echo reply*:

```
alert icmp any any -> any any (msg:"ICMP
teste";sid:10000001;rev:1;classetype:icmp-event;)
```

Uma regra do *Snort* é dividida em duas seções lógicas: o cabeçalho da regra (*rule header*) e as opções da regra (*rule option*), como ilustrado na figura 7. O cabeçalho da regra contém a ação, o protocolo, os endereços IP e as máscaras de rede da origem e do destino e as informações sobre as portas de origem e de destino.

Figura 7 - Formato de regra do Snort



Fonte: STALLING e BROWN, (2018)

A seção de opção de regra (*Rule Option*) contém mensagens de alerta e informações sobre quais partes do pacote devem ser inspecionadas para determinar se a ação da regra deve ser executada.

2.9.3.1. Cabeçalho da regra (*Rule header*)

O cabeçalho da regra contém a ação da regra, o protocolo, os endereços IP e as máscaras de rede da origem e de destino, a direção do tráfego e as informações das portas de origem e de destino. Conforme detalhados a seguir:

Os simbolos utlizados foram defindos por *Snort*.org.

- **Ação da regra:** A ação da regra diz ao *Snort* o que fazer quando encontrar um pacote que corresponda aos critérios da regra. Existem cinco ações padrão disponíveis no *Snort*, *alert, log, pass, activate e dynamic*. Se o *Snort* estiver sendo executado no modo *inline*(IPS), três opções adicionais (*drop, reject e sdrop*) podem ser incluídas.
    1. ***alert:*** gera um alerta e registra o pacote no log;
    2. ***log:*** somente registra o pacote no log ;
    3. ***pass:*** ignora o pacote;
    4. ***activate*****:** gera um alerta, em seguida inicia outra regra dinamica;
    5. ***dynamic:*** ativada somente após uma regra de ativação, após ativada atua como uma regra de log;
    6. ***drop:*** bloqueia e registra um pacote no log ;
    7. ***reject:*** bloqueia e registra o pacote no *log*, e envia *reset* TCP se for protocolo TCP ou envia uma mensagem ICMP de porta inacessivel ser for UDP;
    8. ***sdrop*****:** bloqueia o pacote e não registra em log.
- **Protocolo:** Este campo diz ao *Snort* qual protocolo deve ser analisado no tráfego. Na versão utilizada neste trabalho (2.9.12), quatro protocolos podem ser analisados - IP, TCP, UDP e ICMP.
- **Endereço IP de origem e destino:** O sistema reconhece apenas endereços IP e não aceita nomes de host. Podendo ser especificado:
    - Qualquer endereço IP, usando a palavra-chave "any";
    - Um único endereço IP;
    - Uma lista de endereços IP [192.168.0.1, 192.168.0.10];

  - Um intervalo de endereços IP 192.168.0.0/24;
  - A negação de endereços IP! 192.168.0.10;
  - Uma variável definida no arquivo de configuração .
- **Direção:** Para indicar em qual direção do tráfego a regra será acionada:
  - Do IP de origem para o IP de destino, usa-se o operador direcional '->'.
  - Entre o IP de origem e o IP de destino, use o operador bidirecional '<>'.
- **Portas de origem e destino:** Podem ser especificadas:
  - Qualquer porta, usando a palavra-chave "any";
  - Uma única porta;
  - Uma lista de portas [21, 22, 80-85]
  - Um intervalo de portas 80-587.
  - A negação de portas (!3386);
  - Uma variável definida no arquivo de configuração .

2.9.3.2. Opções da regra (*Rule option*)

A seção de opção da regra contém mensagens de alerta e informações sobre quais partes do pacote devem ser inspecionadas para determinar se a ação da regra deverá ser executada. Todas as opções de regras do *Snort* são separadas umas das outras usando o caractere ponto e vírgula (;). Palavras-chave de opção da regra são separadas de seus argumentos com um caractere de dois-pontos (:).

Existem quatro categorias principais de opções de regras:

- **Geral (*Metadados*):** essas opções fornecem informações sobre a regra, mas não afetam a detecção;
- **Carga útil (*Payload*):** essas opções procuram dados dentro da carga útil do pacote e podem ser inter-relacionadas;
- **Carga não útil (*non-payload*):** essas opções procuram dados em locais difrentes da carga útil;
- **Pós-detecção (*post-detection*):** essas opções são acionadores específicos de regras que acontecem depois que uma regra é disparada.

Na versão 2.9.12 do *Snort*, há noventa palavras-chave de opção de regra disponíveis. Dentre as opções serão destacadas as seguintes: *msg, flow, appid, sid e ver*, as quais foram utilizadas nas regras implementadas neste trabalho.

- Msg: a opção de regra *msg* informa o mecanismo de criação de log e alerta a mensagem para imprimir junto com um despejo de pacote ou para um alerta.
  - Formato: `msg:"<texto da messagem>";`
- Flow: a opção de regra *flow* é usada em conjunto com o rastreamento de sessão. Permite que as regras se apliquem apenas a determinadas direções do fluxo de tráfego.
  - Formato:
    flow:[(established|not_established|stateless)]
    [,(to_client|to_server|from_client|from_server)]
    [,(no_stream|only_stream)]
    [,(no_frag|only_frag)];
- Appid: a opção permite a aplicação da regra por nome de aplicativo.
  - Formato: **appid:<nome de aplicativos>;**
- Sid: A palavra-chave *sid* é usada para identificar de forma exclusiva as regras do *Snort*. E possuir as seguintes restrições referente a numero identificador:
    - <100 - reservado para uso futuro;
    - 100-999.999 - regras incluídas na distribuição do *Snort*;
    - > = 1.000.000 - usado para regras locais.
  - Formato: **sid:<*Snort rules id*>;**
- Rev: A palavra-chave *rev* é usada para identificar de forma exclusiva as revisões das regras do *Snort*.
  - Formato: **rev:<*revision integer*>;**

2.9.4. Pré-processadores

Os pré-processadores ou plug-ins foram introduzidos a partir da versão 1.5 do *Snort*. Estes permitem que o *Snort* estenda suas funcionalidades, possibilitando aos usuários e programadores uma modularidade ao configurar o sistema. O código do pré-processador é executado antes que o mecanismo de detecção seja chamado,

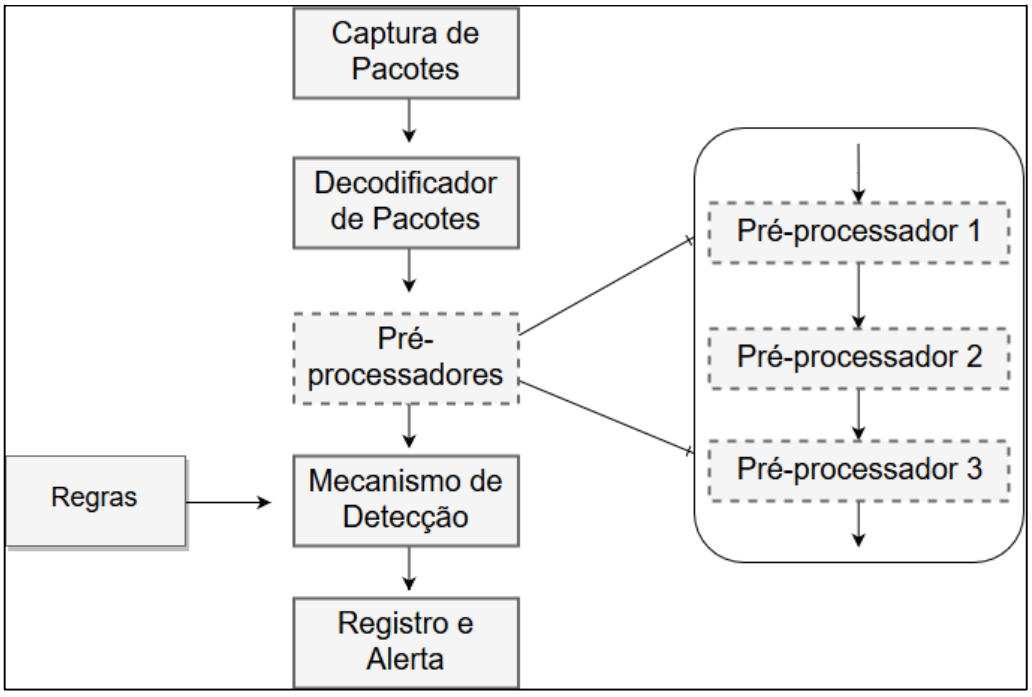
mas após o pacote ter sido decodificado. O pacote pode ser modificado ou analisado de uma maneira fora de banda usando este mecanismo.

Ghafir et al. (2016, p. 4, tradução livre) explica o funcionamento dos pré-processadores:

> A arquitetura do *Snort* permite a implementação de pré-processadores. Os pré-processadores leem o pacote antes da avaliação da regra, serialmente na ordem especificada pela configuração do *Snort*. Isso permite a implementação de palavras-chave de regras adicionais. Além disso, os pré-processadores permitem a implementação de funcionalidades mais complicadas do que a correspondência de padrões, como a decodificação de dados e a detecção de anomalias.

Ainda de acordo com Ghafir et al. (2016), a arquitetura o *Snort* com pré-processadores implementado pode ser visualizada na figura 8:

Figura 8 - Arquitetura do Snort com pré-processadores



Fonte: Ghafir et al. (2016)

Os pré-processadores são carregados e configurados usando a palavra-chave *preprocessor*. O formato da diretiva de pré-processamento no arquivo de configuração do *Snort* é:

```
preprocessor <name>: <options>
```

A versão 2.9.12 do *Snort* utilizada neste trabalho possui suporte a 24 pré-*processadores: Frag3, Session, Stream, sfPortscan, RPC Decode, Performance*

Monitor, HTTP Inspect, SMTP, POP, IMAP Preprocessor, FTP/Telnet Preprocessor, SSH, DNS, SSL/TLS, ARP Spoof Preprocessor, DCE/RPC 2 Preprocessor, Sensitive Data Preprocessor, *Normalizer*, SIP *Preprocessor*, *Reputation Preprocessor*, GTP *Decoder and Preprocessor*, *Modbus Preprocessor*, *DNP3* e *AppId*. Neste trabalho será dado ênfase ao estudo do pré-processador AppId.

2.10. *OpenAppID* (AppID)

Introduzido em 2014 pelo autor do *Snort* e fundador da Sourcefire, Martin Roesch, o *OpenAppID* é uma linguagem de detecção e um módulo de processamento que pode ser adicionado ao *Snort*, possibilitando detectar tráfego de rede de certos aplicativos, pode ser usado para identificar o uso de aplicativos desonestos, detectar aplicativos maliciosos e implementar várias políticas de aplicativos, como a lista negra de aplicativos , limitar o uso de aplicativos e impor controles condicionais (por exemplo, permitir o acesso do Gmail somente se a autenticação de dois fatores for aproveitada).

Segundo Margaret Rouse 2014, o módulo permite detectar aplicativos e também que admistradores criem suas próprias detecções para atender a necessidades comerciais específicas. As informações de detecção também podem ser exportadas do *Snort* para uso por sistemas de análise de segurança ou informações de segurança e gerenciamento de eventos.

Para um melhor entendimento deste trabalho deve-se atentar para as nomenclaturas utilizadas. De acordo com o sitio do *Snort* e o manual versão 2.9.12, o termo *OpenAppID* refere-se a linguagem de detecção e ao pré processador AppId (modulo que realiza a identificação de aplicações).

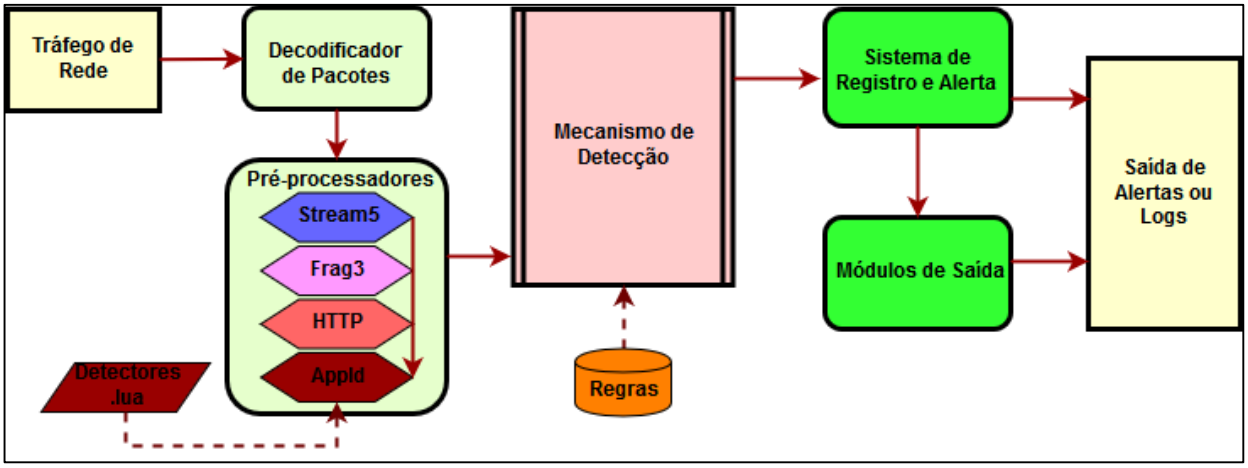
O *OpenAppID* consiste em um conjunto de bibliotecas na linguagem Lua (detectores), usados para detectar aplicativos. Para utilizar esses detectores, é necessário criar regras de texto semelhantes a qualquer outra regra personalizada do *Snort*, com a diferença sendo a palavra-chave "appid" na regra para corresponder apenas ao tráfego já identificado de um aplicativo específico. Os detectores interagem com o *Snort* usando uma API[5] C-Lua.

[5] APIs (interfaces de programação de aplicativos) fornecem uma maneira de conectar componentes de software de computador, facilitando interações entre módulos de código, aplicativos e sistemas de TI.

O método utilizado pelo pré-processador AppId é inspecionar a sessão para obter dados detalhados da camada de aplicação. Estes dados são passados em um conjunto de bibliotecas Lua para identificação de aplicativos. Para correto funcionamento do AppId outros pré-processadores precisam estar em funcionamento. O pré-processador Stream5 que é um módulo de reagrupamento TCP, o qual é capaz de rastrear sessões para TCP e UDP. O pré-processador Frag3 que é um módulo de desfragmentação IP. E o pré-processador HTTP *Inspector* para identificar aplicativos HTTP. Stream5 e Frag3 já são integrados a distribuição oficial do *Snort*.

De acordo com as referências de Ghafir et al. (2016) e Wang (2017). A arquitetura do *Snort* com a implementação do pré-processador AppId pode ser visualizada como na figura 9. Onde o tráfego de rede passará pelo “Decodificador de Pacotes”, cujo objetivo é preparar os pacotes para os pré-processadores. Os “Pré-processadores” recebem o tráfego, permitindo que esses executem funções complexas, incluindo a remontagem do fluxo TCP, desfragmentação IP, a coleta de estatísticas ou a normalização de solicitações HTTP. O AppId inspeciona o pacote utilizando os detectores. Desse modo os pré-processadores modificam os pacotes para que possam ser comparados com as regras do “mecanismo de detecção”.

Figura 9 - Arquitetura do Snort e OpenAppId



Fonte: Adaptado de Wang (2017)

O “Mecanismo de Detecção” recebe pacotes dos pré-processadores e compara-os a uma lista de regras do *Snort*. Se um pacote acabar correspondendo a qualquer uma das regras, ele gerará o alerta ou uma ação apropriada. Se nada corresponder, o pacote é ignorado ou descartado. Os pacotes que foram correspondidos a uma regra do *Snort* vão para o “Sistema de Registro e Alerta”, que

envia as informações para o registro em um formato de arquivo, podendo ser para um arquivo de log ou um banco de dados.

2.10.1. Estrutura do código do detector

Conforme a Sourcefire Cisco (2014), o código de um detector escrito em script Lua deve possuir os seguintes componentes:

- *Detector Info Header:* Trata-se de um bloco opcional, o qual descreve informações a respeito do detector como nome, versão, descrição, etc.
- *Include Required Libraries***:** Neste bloco são incluídas bibliotecas necessárias execução. Como boa prática de programação códigos usados com frequência podem ser incluídos em uma biblioteca para manter os códigos de detectores curtos. As bibliotecas oficais da Cisco devem ser colocadas no subdiretório `<Snort_install_dir>/rules/odp/libs`. Bibliotecas cridas por usuarios devem ser colocá-das no subdiretorio `<Snort_install_dir>/rules/odp/custom/libs`.
- *DetectorPackageInfo:* Esse bloco e necessário em cada detector, pois é ele que identifica as funções do cliente e do servidor que serão chamadas para inicializar, validar (processar pacotes) e limpar o detector. O AppId lê esse bloco depois de carregar o código Lua e chamar as funções de inicialização. A estrutura possui os seguintes elementos:
  - *DetectorPackageInfo.name:* Este é um nome para o detector que é usado para fins de registro.
  - *DetectorPackageInfo.proto*: valor do protocolo. Pode ser DC.ipproto.tcp ou DC.ipproto.udp.
  - *DetectorPackageInfo.client:* Se o detector identificar o aplicativo do lado do cliente, essa estrutura será preenchida. Os detectores para *payload* do aplicativo fornecerão apenas a seção do cliente.

    As seguintes funções podem ser fornecidas:
    - *Init:* Nome da função de retorno de chamada que inicializa um detector.
    - *Validate*: Nome da função de retorno de chamada que processa pacotes no detector. A função inspeciona o conteúdo do pacote e pode usar os resultados

armazenados dos pacotes anteriores para detectar a aplicação. Antes de terminar, as funções chamam uma função de API apropriada para indicar os resultados da detecção. Essas funções não são necessárias para alguns aplicativos específicos.

- *Clean*: Nome da função de retorno de chamada que executa a limpeza quando o *Snort* está finalizando a analise do pacote. A função é opcional e pode ser omitida na estrutura DetectorPackageInfo.

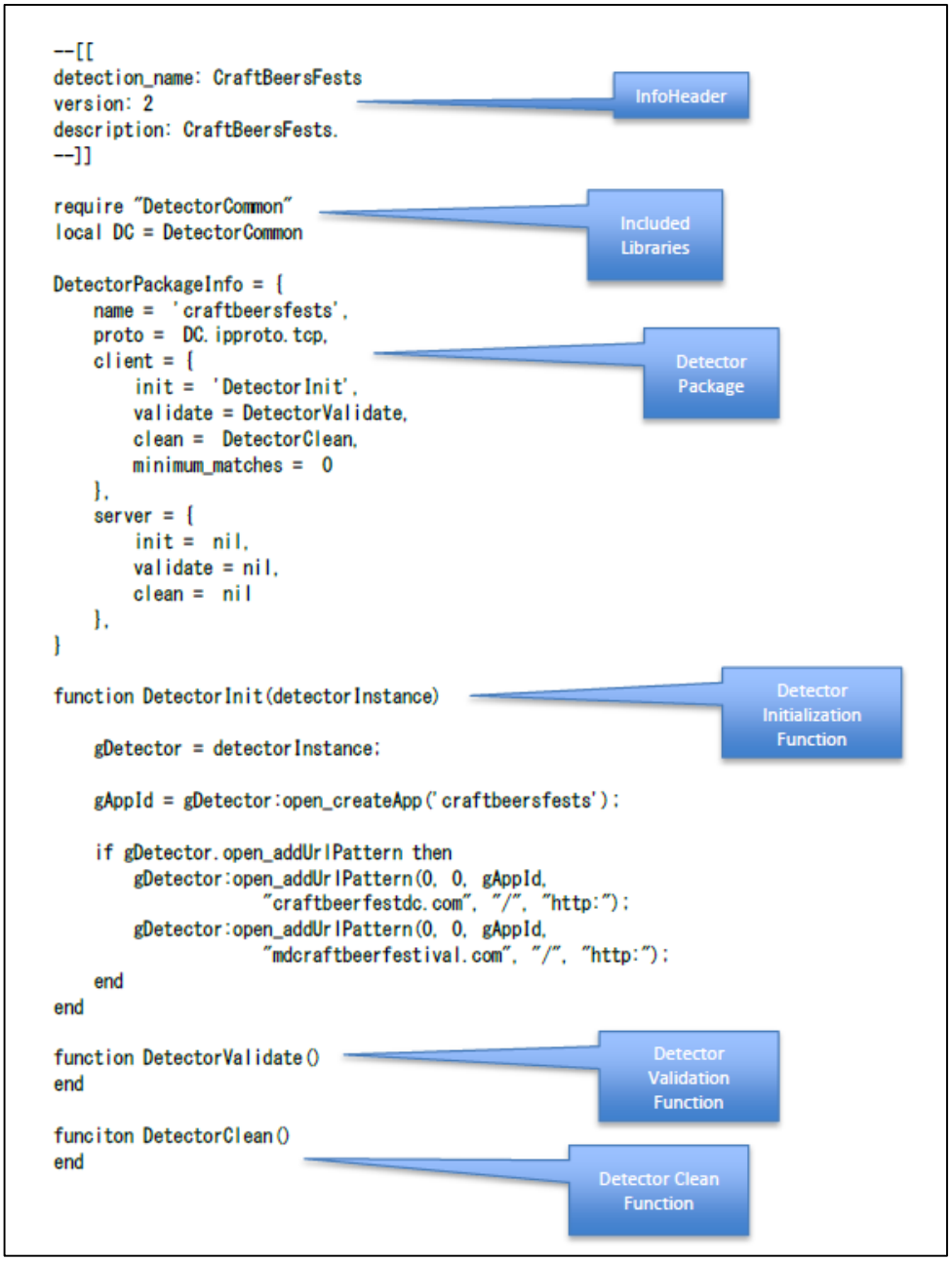
- *DetectorPackageInfo.server*: Se o detector identificar um aplicativo do lado do servidor, essa estrutura será preenchida. A estrutura fornece as funções *init*, *validate* e *clean* que possuem o mesmo propósito do lado do cliente.

- *Detector Initialize*: Cada detector deve ter uma função inicializada na estrutura *DetectorPackageInfo*. O AppId chamará essa função diretamente ao carregar o detector. A função recebe *detectorInstance*, uma instância da classe detector, que deve ser armazenada globalmente e usada para chamar todas as funções da API Lua-C. A função pode executar um ou mais dos seguintes procedimentos:
  - Criar um novo nome de aplicativo chamando open_createApp().
  - Configurar padrões rápidos e a porta para um aplicativo. Estes são usados para selecionar um detector para um fluxo.
  - Adicione padrões para cabeçalhos específicos para HTTP.
- *Detector Validate:* Essa função é chamada quando o pré-processador AppId determina que o detector é viável para uma inspeção mais profunda para detectar um aplicativo. Ele pode ser uma correspondência de padrão acionada pelo estado que abrange pacotes múltiplos em um fluxo ou uma correspondência de padrão simples direta com qualquer pacote. As funções de validação não são chamadas se o aplicativo real for baseado em HTTP, SSL e SIP. Para esses aplicativos, os pré-processadores do *Snort* analisam os cabeçalhos de protocolo e os disponibilizam para a correspondência de padrões através das funções da API C-Lua.

- *Detector Fini:* Esta função é chamada quando um detector é destruído durante a saída do *Snort*. A linguagem Lua realiza coleta de lixo automaticamente quando um objeto não é mais referenciado, então esta função libera memória.

A figura 10 abaixo inlustra a estrutura do detector com a funções descritas acima:

Figura 10 - Estrutura de um detector



Fonte: Sourcefire Cisco (2014)

2.10.2. Funções API Lua

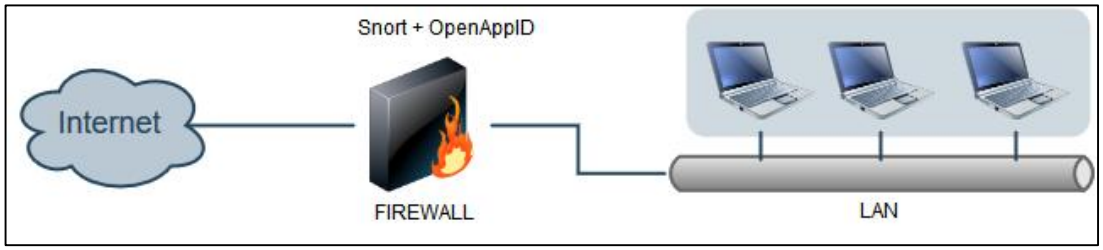
Os detectores do *OpenAppID* interagem com o *Snort* usando uma API C-Lua. Nesta API estão presentes funções para realizar a identificação de aplicações. A API é escrita na linguagem C e recebe o codigo em linguagem Lua dos detectores. Dentre as várias funções presentes, destaca-se a função *addPortPatternService*, utilizada neste trabalho, a qual raliza uma detecção baseada em porta e padrão para o aplicativo. Os critérios de porta e padrão devem ser atendidos antes que o aplicativo

seja considerado detectado. A API pode ser consultada antes da compilação do *Snort* no pacote de instalção, neste exemplo em:
`~/Snort-2.9.12/src/dynamic-preprocessors/appid/luaDetectorApi.c`

2.10.3. Técnica de detecção

O *OpenAppID* deve ser implantado no *firewall*, devendo ficar entre os clientes (hosts da LAN) e a Internet, conforme ilustrado na figura 11. Desta forma, cada pacote é processador através do *firewall* será analisado pelo *Snort* com *OpenAppID* e este pode aplicar regras para correspondência de aplicativos que estão predefinidas na forma de arquivos na linguagem Lua. Os resultados são encaminhados para *Snort*, o qual executa ações necessárias sobre esses pacotes.

Figura 11 - Posicionamento OpenAppID



Fonte: Próprio autor (2018)

O *OpenAppID* tem duas partes principais. A primeira é o pré-processador AppId propiamente dito, o qual é executado pelo *Snort* juntamente com outros pré-processadores, onde o *Snort* envia dados de pacote e sua responsabilidade é determinar quais aplicativos foram identificados em conjunto de pacotes. A segunda são os detectores escritos na linguagem Lua, usados para detectar vários aplicativos.

A detecção de aplicativos pode ser dividida em duas partes:

- 1- O AppId decide quais detectores inspecionarão o tráfego. Isso pode ser feito combinando padrões simples nos dados do *payload* ou combinando a porta de destino.
- 2- O AppId usa o validador do detector para fazer uma inspeção adicional do tráfego. Diferentes detectores precisam ver diferentes quantidades de tráfego antes de poderem fazer uma resposta definitiva, se o tráfego é ou não da aplicação especificada.

O AppId realiza em uma sessão a identificação dos pacotes do lado do cliente e do lado do servidor, analisando os dados do *payload* dos pacotes. O *Snort* pode

acessar diretamente os pacotes identificados através do campo "AppId" da estrutura de pacotes. As regras do *Snort* possuem a opção de detecção "appid" que permite aos usuários incluir o nome do aplicativo como um aprimoramento e adição às declarações de regras. Desse modo, uma regra é aceita como uma correspondência se um dos "appid" em uma regra corresponder a um Identificador de Aplicações (AppId) em uma sessão.

Geralmente quatro Identificadores de Aplicações são armazenados em uma sessão conforme definido abaixo:

- *serviceAppId:* AppId associado à sessão do lado do servidor.
  Exemplo: servidor http.
- *clientAppId:* AppId associado ao aplicativo no lado do cliente de uma sessão.
  Exemplo: navegadores como o Google Chrome e Firefox.
- *payloadAppId:* Para serviços como http, este é um AppId associado ao host do servidor web.
  Exemplo: Office 365, Gmail, etc.
- *miscAppId:* Protocolos encapsulados.

O pré-processador imprime o uso de rede do aplicativo periodicamente no diretório de log do *Snort* no formato *unified2*. As ferramentas *u2spewfoo*, *u2OpenAppID*, *u2streamer* podem ser usadas para imprimir o conteúdo desses arquivos.

2.10.4. Detectores não catalogados

Os usuários podem detectar novos aplicativos adicionando detectores na linguagem Lua, copiando os detectores fornecidos pela equipe do *Snort* e modificando-os ou usando um *script shell* para criação de detectores (**appid_detector_builder.sh**), disponibilizado pela comunidade. Os detectores Lua criados pelos usuários devem ser armazenados no diretório **`/etc/Snort/rules/odp/lua/.`**

A ferramenta chamada **appid_detector_builder.sh** é um *script bash* simples, orientado por *menus*, que cria arquivos “.lua” no diretório atual, com base em escolhas

e padrões fornecidos. Nesse trabalho os detectores personalizados serão elaborados de acordo com o guia “*Open Source Detectors Developers Guide*”.

2.11. Linguagem Lua

Os detectores *OpenAppID* estão escritos na linguagem Lua. De acordo com Ierusalimschy, Celes e Figueiredo (2017), esta é uma linguagem de *script*, rápida e leve, sendo suas principais propriedades a portabilidade, simplicidade e tamanho do código-fonte de apenas alguns Kbytes, o que permite que os programadores incorporem a linguagem Lua em diversas plataformas, que possuem um compilador C.

“Lua é usada em muitas aplicações industriais (ex., *Adobe's Photoshop Lightroom*), com ênfase em sistemas embutidos (exe., o *middleware* Ginga para TV digital) e jogos (ex., *World of Warcraft e Angry Birds*).” (IERUSALIMSCHY; CELES; FIGUEIREDO, 2017). Além do *Snort* e do *OpenAppID* ela também é usada em outros aplicativos de rede como o analisador de protocolo de rede *Wireshark* e o scanner de segurança gratuito Nmap.

2.12. Outras ferramentas utilizadas no trabalho

2.12.1. Wireshark

O *Wireshark* é um *sniffer* de rede. Um *sniffer* é um software que captura os bits que chegam a uma interface de rede, interpretando-os e convertendo-os para uma linguagem amigável para futura análise do tráfego. Ele será utilizado na captura dos pacotes para elaboração de detector de aplicativo não catalogado pelo *OpenAppID*.

3. CENARIO DE IMPLANTAÇÃO

Para este trabalho foi elaborada uma rede virtualizada com máquinas virtuais (VMs), a qual simula uma estrutura de rede padrão, onde existe um servidor exposto à Internet, e este fornece serviços e acesso à Internet para uma Intranet (rede de computadores privada). Foi utilizado o *software* de virtualização *Oracle VM VirtualBox*[6], no qual foram criadas duas VMs para simulação dos testes, em uma máquina hospedeira(*VirtualServer*) com as seguintes configurações de hardware e software: Processador Intel(R) Core(TM) i3-3217U CPU @ 1.80 GHz (4 núcleos), Memória RAM de 8 GB, 700 GB de *Hard Disk*(HD), adaptador de Rede *Wireless* 802.11b|g|n (2.4 GHz) e Sistema Operacional(OS) *Microsoft Windows 10 Enterprise* LTSB 64-bit. Duas sub-redes foram utilizadas no ambiente, a sub-rede da máquina hospedeira 192.168.0.0/24, a qual será utilizada como *Wide Area Network*(WAN) no ambiente virtual e uma rede 10.0.0.0/24, que será utilizada como *Local Area Network*(LAN).

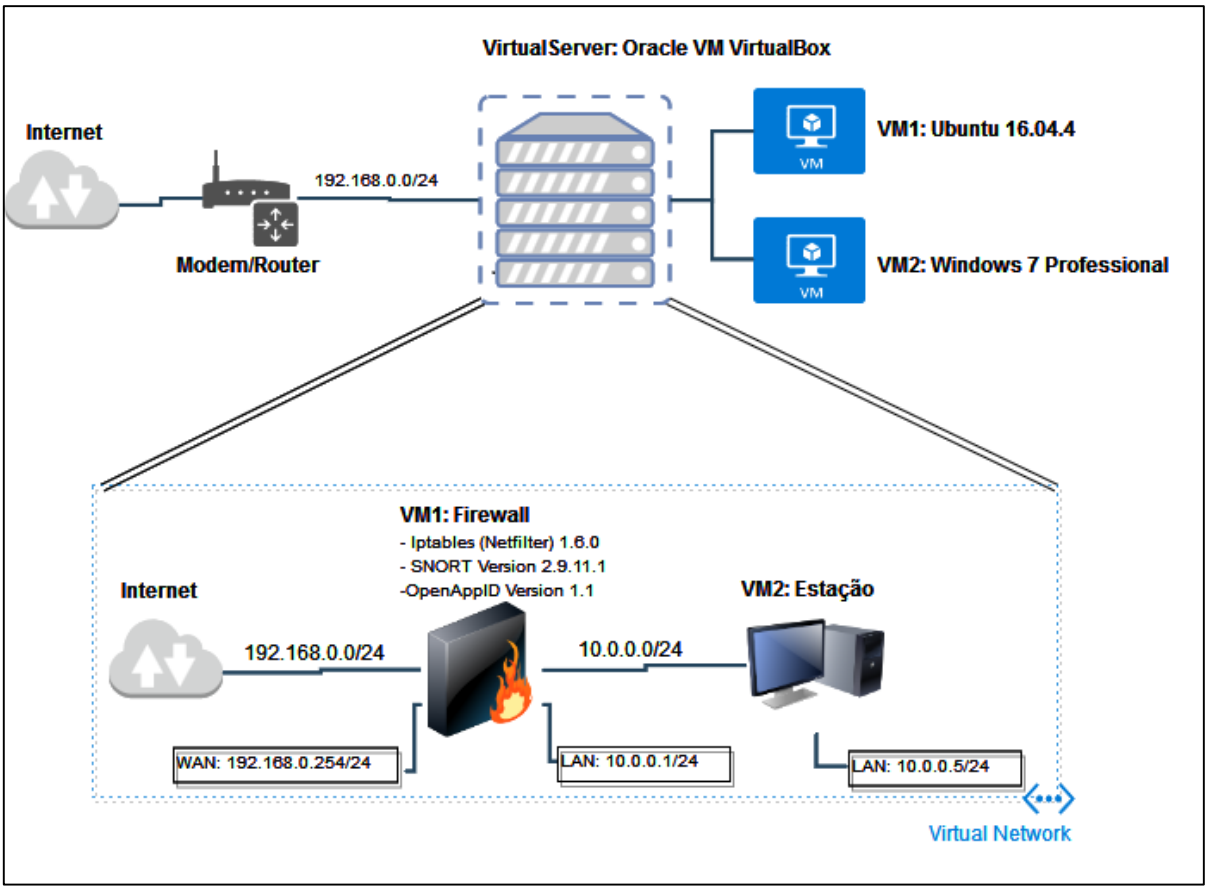
Na primeira máquina virtual(VM1), utilizada como *firewall* nesse ambiente, foram utilizadas as seguintes configurações de hardware e softwares: Processador Intel(R) Core(TM) i3-3217U CPU @ 1.80 GHz (2 núcleos), Memória RAM de 2 GB, 38 GB de *Hard Disk*(HD), 2 adaptadores de Rede *Ethernet* (WAN IP: 192.168.0.254/24 – LAN IP: 10.0.0.1/24) e Sistema Operacional(OS) *Ubuntu* 16.04.4 LTS 64-bit (*Kernel*: 4.4.0-117-*generic*).

A segunda máquina virtual(VM2), utilizada como estação cliente nesse ambiente, foram utilizadas as seguintes configurações de hardware e softwares: Processador Intel(R) Core(TM) i3-3217U CPU @ 1.80 GHz (2 núcleos), Memória RAM de 2 GB, 39,5 GB de *Hard Disk*(HD), adaptador de Rede *Ethernet* (LAN IP 10.0.0.5/24: e Sistema Operacional(OS) *Microsoft Windows 7 Professional 32-bit (SP1)*.

[6] https://www.virtualbox.org/

A figura 12 mostra a arquitetura do cenário virtual (*Virtual Network*) implementado utilizando a ferramenta *Oracle VM VirtualBox* com as máquinas virtuais **"VM1:*Firewall*"** e **"VM2:Estação"** para a realização dos testes do *Snort* com *OpenAppID*.

Figura 12 - Arquitetura do cenário



Fonte: Próprio autor (2018)

3.1. Instalação *Snort* e *OpenAppID*

Para a execução dos testes no cenário, foi realizada a instalação de bibliotecas e ferramentas que irão trabalhar em conjunto com o *Snort* e o *OpenAppID*. Os passos realizados para a instalação foram baseados na documentação disponibilizada no site oficial do *Snort*. Foram realizadas modificações para que os processos se adequassem as versões atualizadas das ferramentas utilizadas. Todo o roteiro de instalação é apresentado no Apêndice A. Os processos foram realizados na máquina virtual **"VM1:*Firewall*".**

Devido a limitações de hardware e software do ambiente de teste, o *Snort* foi instalado e configurado para operar no modo passivo como um Sistema de Detecção de Intrusão de Rede (NIDS).

3.2. Ferramentas Adicionais

Para finalizar a configuração do cenário de teste, foi necessário a instalação ou configuração das fermentas adicionais descritas baixo:

- ***Netfilter/Iptables***

Na máquina virtual **"VM1:*Firewall*"** usada neste cenário foi utilizado o *Iptables* versão 1.6.0 no modo *Statefull*, com regras *FORWARD* e *NAT*, permitindo a conexão da máquina **"VM2:Estação"** com a Internet. O *Iptables* fara a função de *firewall***.**

O *Iptables* já vem pré-instalado no Ubuntu 16. Para realizar o compartilhamento de acesso à *Internet* é necessário primeiramente habilitar o roteamento de pacotes IPv4 no arquivo **`/etc/sysctl.conf`.** Para isso retira-se o comentário (#) da linha 60 e alterasse o valor do parâmetro para 1. Assim, o Ubuntu passa a rotear os pacotes entre as interfaces presentes no sistema.

```
# Linha 60 do arquivo /etc/sysctl.conf
net.ipv4.ip_forward=1
```

Para utilização do *Netifilter/Iptables* com um *Firewall Statefull* é necessário carregar o modulo *Conntrack* no *Kernel* do Linux com os comandos a seguir:

```
$ sudo modprobe ip_conntrack
$ sudo modprobe ip_conntrack_ftp
```

Para que os computadores da LAN consigam navegar na Internet utilizando o mesmo IP público válido, e necessário a criação de uma regra na tabela *nat*, cadeia *POSTROUTING* para habilitar o mascaramento de IP (IP *masquerading*) na interface WAN.

```
iptables -t nat -A POSTROUTING -o $INTERFACE_EXTERNA -j MASQUERADE
```

No Apêndice B encontra-se o código do *Shell script*[7] com a regras que foram implementados no *firewall* do experimento.

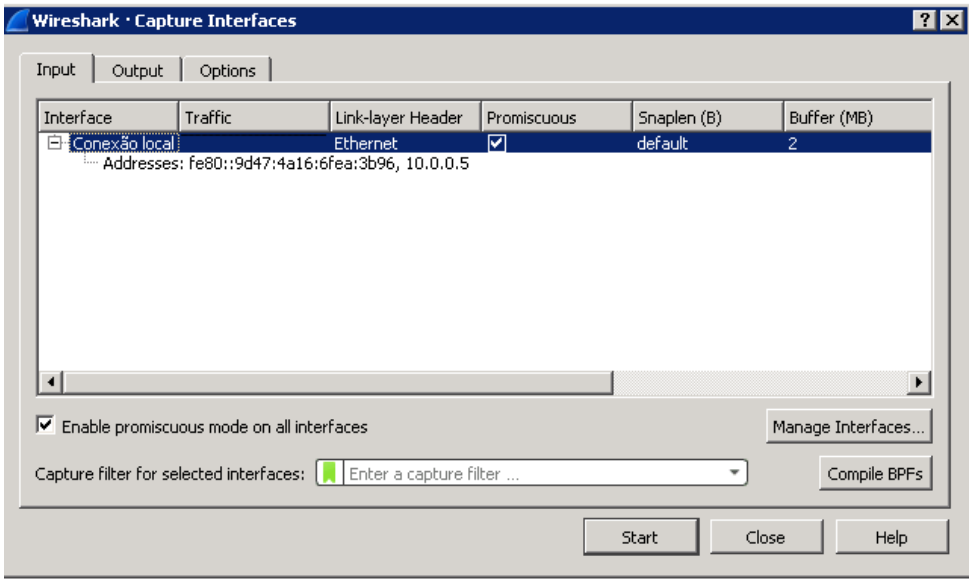
[7] Um *Shell script* é um arquivo que guarda vários comandos e pode ser executado sempre que necessário.

❖ ***Wireshark***

O *Wireshark* foi instalado na máquina virtual **"VM2-Estação"** para capturar informações de tráfego na mesma.

Após a instalação a interface de rede monitorada foi configurada em modo promíscuo, no *"menu" Capture>Options*, conforme figura 13:

Figura 13 - Habilitar modo promíscuo Wireshark



Fonte: Print Wireshark (2018)

❖ **Mira**

É um sistema que atende aos diversos serviços relacionados à área educacional no Modelo Pedagógico do SENAC[8]. O acesso ao sistema é realizado através de uma aplicação desenvolvida para Sistema Operacional Windows. Está será aplicação quer será estudada para realizar a identificação do tráfego de rede. A mesma foi instalada na máquina virtual **"VM2:Estação"**.

[8] SENAC - Serviço Nacional de Aprendizagem Comercial: Instituição brasileira de educação profissional aberta a toda a sociedade.

4. DETECÇÃO DE APLICAÇÕES NÃO CATALOGADAS

O snort.org disponibiliza um pacote chamado *Application Detector Package,* o qual contém os detectores para identificar o tráfego de 2.828 aplicações até a última atualização em 16 de agosto de 2018. Os detectores foram desenvolvidos por *Snort*.org e inclui também detectores adicionais que vieram da comunidade de código aberto.

Descrever-se as etapas necessárias para elaborar a detecção de uma aplicação não catalogada pelo *Snort*.org ou a comunidade. Foi escolhida a aplicação “Mira”, citada na seção 3.2. Para elaborar um detector é necessário identificar padrões ou comportamentos únicos em cada aplicação, esse processo foi dividido em cinco etapas.

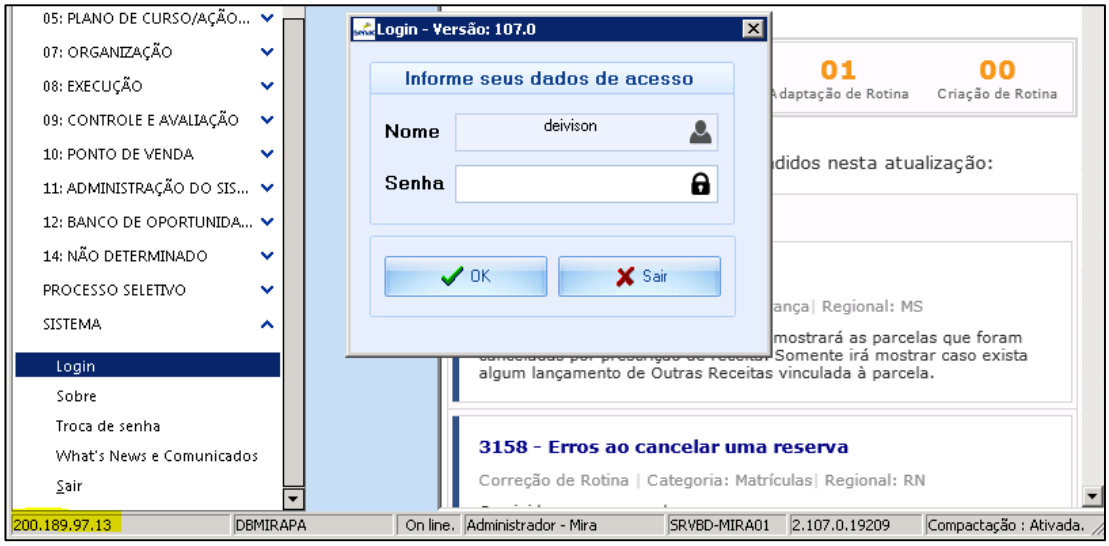
A primeira etapa refere-se à obtenção de informações da aplicação que ajudariam a identificá-la na etapa seguinte. A segunda etapa refere-se à captura dos pacotes de rede. A terceira refere-se a análise dos pacotes capturados, com a finalidade de extrair as informações e utilizá-las na elaboração do detector na linguagem lua na quarta etapa. E por fim, na quinta etapa serão realizados os testes de aplicação em uma regra no *Snort* para realizar a identificação da aplicação “Mira”.

4.1. Obtendo informações da aplicação (Etapa 1)

Antes de elaborar o detector deve-se obter detalhes da aplicação, com a finalidade de obter informações para identificar o tráfego da mesma na etapa seguinte. A figura 14 mostra a tela inicial da aplicação “Mira” onde é possível identificar no rodapé da tela um IP (grifado em amarelo) utilizado pela aplicação. Essa informação

foi utilizada na etapa de captura de pacotes, na utilização da ferramenta Wireshark.

Figura 14 - Tela inicial Mira



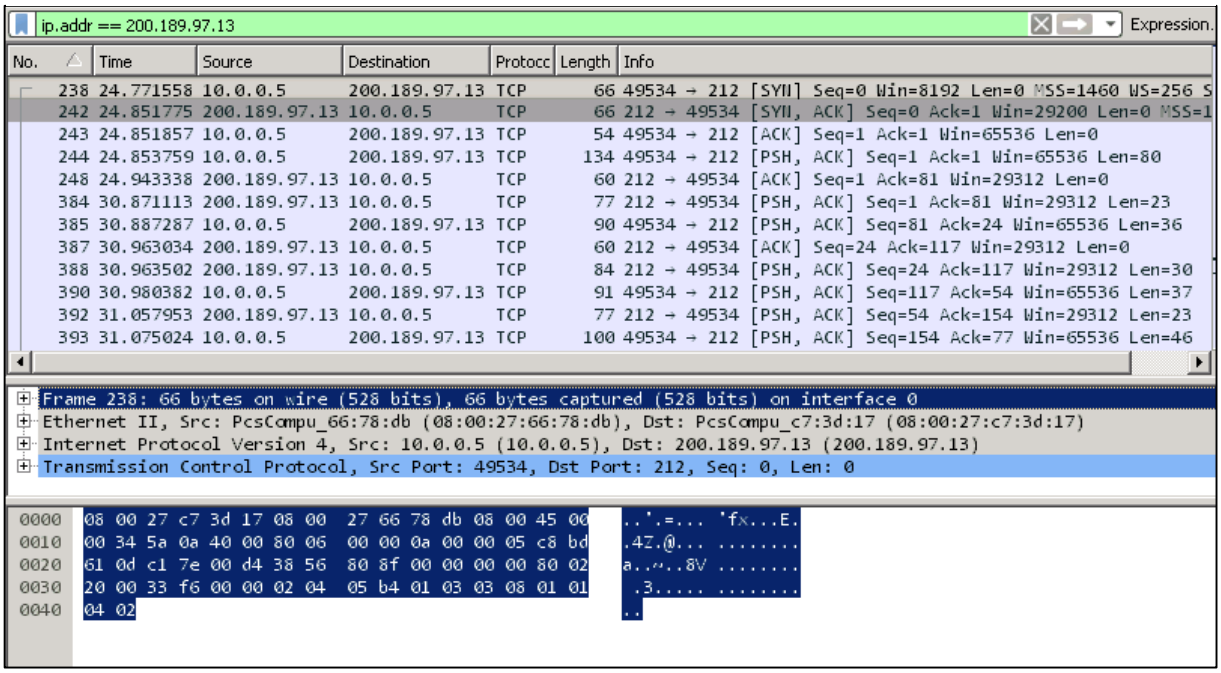
Fonte: Print Sistema Mira (2018)

4.2. Captura de pacotes (Etapa 2)

A fase de captura de pacotes foi realizada pela ferramenta Wireshark, onde as informações capturadas são interpretadas e apresentadas na forma de pacotes individuais no contexto de endereçamento, protocolo e dados. Neste trabalho foi utilizado o método de “captura em máquina local”, na máquina virtual **“VM2:Estação”**.

Após executar a ferramenta Wireshark, foi iniciada a aplicação “Mira” e finalizada após alguns minutos de execução, tempo suficiente para o Wirehsark capturar uma quantidade grande de informações, tanto da aplicação alvo quanto de outros tráfegos simultâneos na máquina. Para isolar o tráfego da aplicação escolhida para análise, realizou-se uma filtragem por IP (ip.addr == 200.189.97.13), obtido na etapa anterior, ficando na tela somente informações trocadas entre a máquina virtual **“VM2:Estação”** (IP: 10.0.0.5) e a máquina servidora da aplicação, conforme figura 15.

Figura 15 - Filtragem por IP no Wireshark



Fonte: Print Wireshark (2018)

4.3. Extração das informações (Etapa 3)

Para a elaboração do detector será utilizada a função “*addPortPatternService*”, esta função está presente na API C-Lua, mencionada no capitulo 2.10 (*OpenAppID*). A função realiza a detecção baseada em porta e padrão para o aplicativo. Abaixo segue detalhemento da função utilizada.

Função:

- *addPortPatternService(proto,port,pattern,offset,appId)*

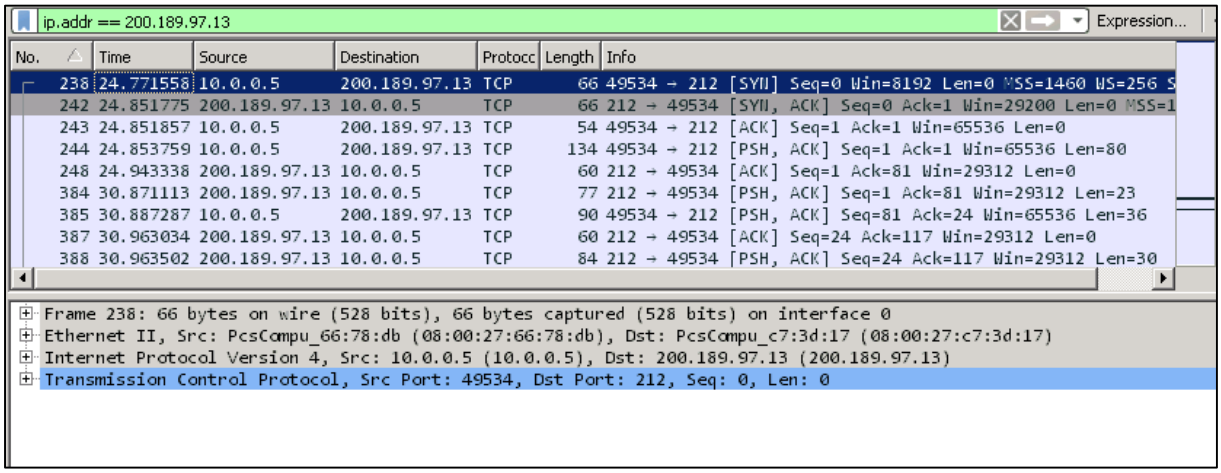
Parâmetros:

- *proto*: Protocolo (TCP = DC.ipproto.tcp ou UDP = DC.ipproto.udp);
- *port:* número da porta utilizada;
- *pattern*: padrão a ser correspondido (padrões ou comportamentos únicos da aplicação);
- *offset*: deslocamento na carga útil do pacote onde a correspondência deve começar;
- *appId*: ID do aplicativo a ser usado para este detector.

Nessa terceira etapa realiza-se a leitura dos registros capturados pelo Wireshark para o preenchimento dos parâmetros necessários da função.

A figura 16 apresenta a estrutura de detalhes do primeiro pacote do fluxo entre o cliente e o servidor, onde já é possível obter dois parâmetros para a função. O tipo de protocolo utilizado (*proto*: TCP) e a porta utilizada pelo servidor da aplicação (*port*: 212).

Figura 16 - Detalhes do pacote selecionado

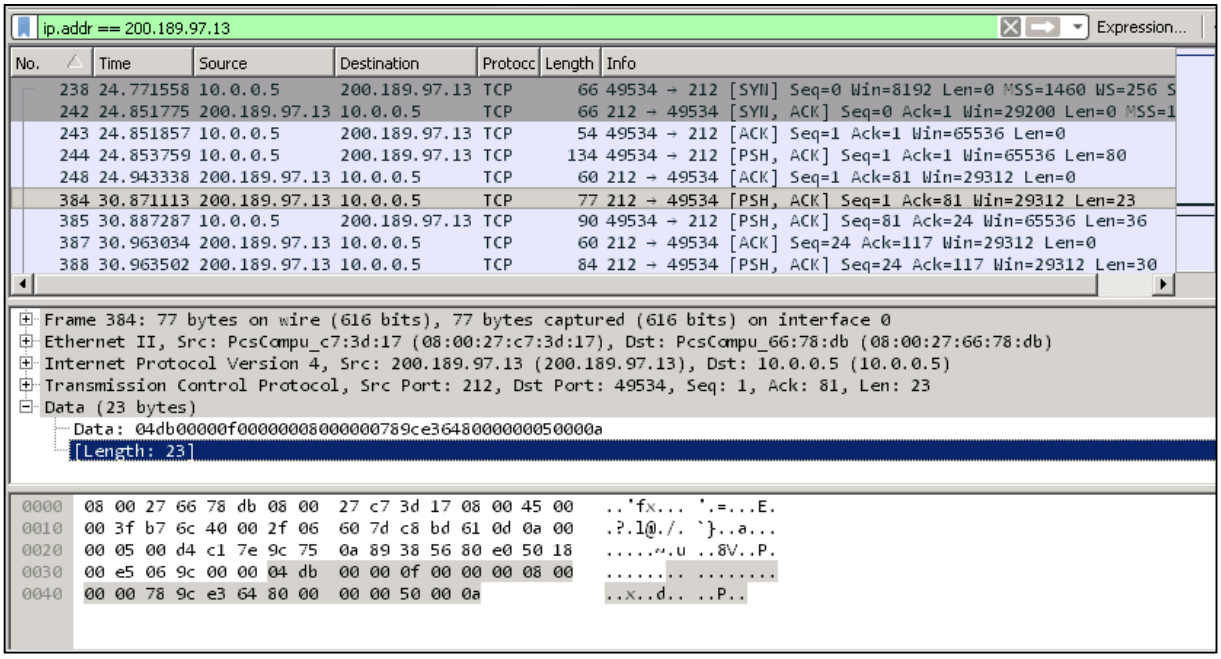


Fonte: Print Wireshark (2018)

O próximo parâmetro a ser preenchido é o *pattern* (padrões ou comportamentos únicos da aplicação). Segundo Kleopa (2015), na realização do "*Three-way Handshak*[9]" entre uma aplicação e um servidor, geralmente é mantido um padrão no primeiro pacote com dados do lado do servidor. Após análise no Wireshark foi identificado a primeira troca de dados do lado do servidor (pacote número 384) com um tamanho de 23 bytes, conforme pode ser visto na figura 17.

[9] Processo utilizado para o estabelecimento de conexões TCP.

Figura 17 - Detalhe de pacote capturado pelo Wireshark



Fonte: Print Wireshark (2018)

Assim pode-se preencher o valor do parâmetro *pattern*, valor em hexadecimal copiado diretamente do Wireshark*: 04 db 00 00 0f 00 00 00 08 00 00 00 78 9c e3 64 0010 80 00 00 00 50 00 0a.*

O parâmetro *offset* refere-se onde inicia o valor dos dados no deslocamento. O primeiro valor em hexadecimal (*04*), inicia no *offset* 0036.

O último parâmetro e ser preenchido é o *appId*, no caso é o nome da aplicação a ser detectada (*appId*: Mira).

Com todos os parâmetros coletados pode iniciar a próxima etapa.

4.4. Elaboração do detector (Etapa 4)

Com a informações obtidas na etapa anterior foram preechidos todos os parametros necessarios para elaboração do detector:

- *proto:* TCP;
- *port*: 212;
- *pattern:* no Wireshark este valor está em hexadecimal. Para utilização no detector, o valor deve ser convertido para decimal, substituído os espaços por “\”. Foram utilizadas as informações do offset 0036 até o offset 003F,

que são os primeiros 9 bytes dos dados do lado do servidor no deslocamento 0.

HEX: *04 db 00 00 0f 00 00 00 08*

DEC: *004\219\000\000\015\000\000\000\008*

- offset: *0*;

Assim foi eladorado o detector, como poder ser observado no código presente no apêndice D, neste código foi adicionada a linha que acrescenta padrões da aplicação Mira no arquivo chamado de “client_Mira.lua” e armazenado no diretorio `/etc/snort/rules/custom.`

4.5. Testes e aplicação do detector (Etapa 5)

Quando o *Snort* for executado, serão criados arquivos no diretório de log (**`/var/log/snort`**), nomeados como “**`appstats-u2.log.xxxxxxxxxx`**”, onde **`x`** são números. O *OpenAppID* disponibiliza algumas ferramentas para visualização de informações contidas nesses arquivos, são elas:

- *u2OpenAppID*: com essa ferramenta e possível obter uma lista das aplicações detectadas;
- *u2spewfoo:* com essa ferramenta é possível visualizar estatísticas de utilização das aplicações detectadas;

O detector criado na etapa 4 ainda não está sendo utilizado pelo pré-processador AppId. Inicializando o *Snort* em modo IDS, conforme o comando a seguir, é possivel verificar a detecção de aplicações já catalogadas com a ferramenta *u2OpenAppID,* conforme figura 18.

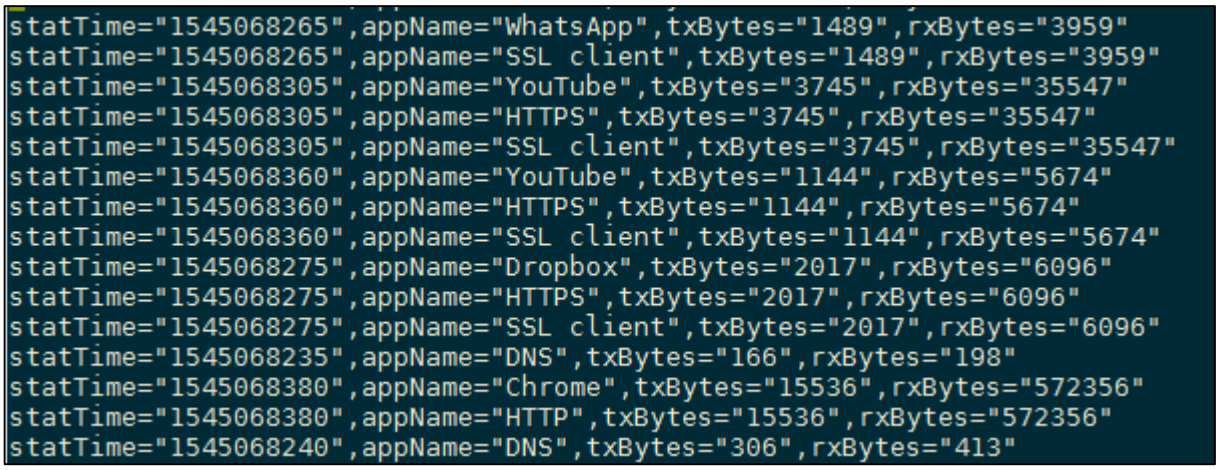
```
$ sudo snort -A console -q -c /etc/snort/snort.conf -i enp0s8
-N
```

Detalhes do comando:

- **(-A) -> A opção faz o console imprimir alertas para a saída padrão na tela;**
- **(-q) -> executa em modo “silencioso” (não mostrando o relatório de banner e status)**
- **(-c) -> especifica o arquivo de configuração**

- **(-i enp0s8) -> especifica qual interface ira monitorar.**

Figura 18 - Saída ferramenta u2openappid



Fonte: Print bash Ubuntu (2018)

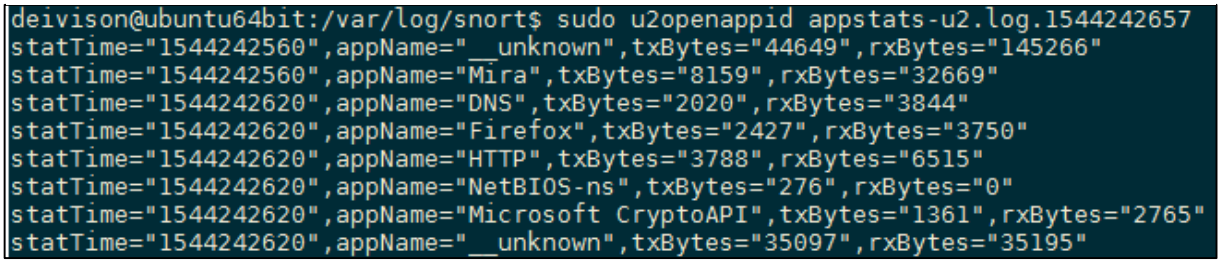
As aplicações foram acessadas na máquina virtual **“VM2:Estação”.**

Para que o AppId possa utilizar o detector da aplicação “Mira” foi criado um *link* do detector para o diretório **odp.**

**$ sudo ln -s /etc/*Snort*/rules/custom/client_Mira.lua **
/etc/*Snort*/rules/odp/lua

Executando-se novamente a ferramenta *u2OpenAppID* é possível detectar a utilização da aplicação “Mira” (figura 19), a qual foi executada após criação do *link* para o detector. Os aplicativos listados com o mesmo ***statTime*** são da mesma solicitação.

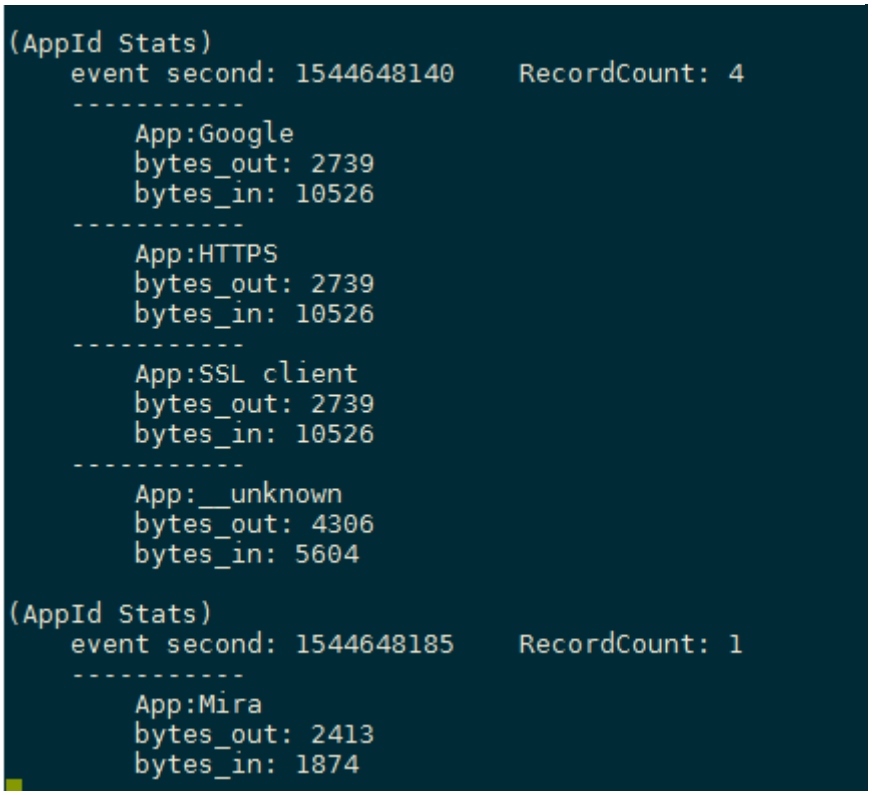
Figura 19 - Detecção da aplicação “Mira”



Fonte: Print bash Ubuntu (2018)

Com a ferramenta *u2spewfoo* é possível visualizar estatísticas de utilização das aplicações detectadas, conforme exibido na figura 20:

Figura 20 - Saída ferramenta u2spewfoo



Fonte: Print bash Ubuntu (2018)

Observando a figura 20, cada bloco “AppId Stats” corresponde a detecção de uma aplicação em uma sessão, sendo que uma aplicação pode utilizar outras aplicações como é visto ao acessar o Google. Também são visualizadas informações de tráfego, bytes de entrada e bytes de saída utilizando pela aplicação.

Após as configurações realizadas nas etapas anteriores, obter informações, capturar pacotes, extrair informações e elaborar detector e realizadas os testes já é possível aplicar regras no *Snort* orientadas a aplicações.

Para que o *Snort* emita alerta em tempo real sobre o uso da aplicação “Mira”, foi necessário à adição de uma regra de alerta para a detecção do tráfego da utilização da aplicação. A regra foi adicionada no arquivo de regras locais, localizado no caminho **`/etc/snort/rules/local.rules`** com a seguinte sintaxe**:**

```
alert tcp any any -> $EXTERNAL_NET any
(msg:"Mira";flow:from_client;appid:Mira;sid:74781;rev:1;)
```

Assim quando da utilização da aplicação “Mira” em máquinas clientes, o *Snort* + AppId identifica e emite alertas em tempo real conforme pode ser observado na figura 21:

Figura 21 - Snort detectando aplicação em uso

```
12/08-01:16:58.419670  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
12/08-01:16:58.504115  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
12/08-01:16:58.598722  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
12/08-01:16:58.703754  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
12/08-01:16:58.806861  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
12/08-01:16:58.911385  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
12/08-01:16:58.991591  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
12/08-01:16:59.079327  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
12/08-01:16:59.185823  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
12/08-01:16:59.193997  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
12/08-01:16:59.328747  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
12/08-01:16:59.707769  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
12/08-01:17:00.100193  [**] [1:74781:1] Mira [**] [Priority: 0] {TCP} 10.0.0.5:49591 -> 200.189.97.13:212
```

Fonte: Print bash Ubuntu (2018)

A sintaxe da regra pode ser modificada de acordo com a necessidade desejada, por exemplo, podem ser adicionados vários *appid* em uma regra para detectar mais de uma aplicação com uma única regra.

Neste capítulo foram apresentadas as etapas necessárias para elaboração de um detector, bem como a aplicação do mesmo em uma regra do *Snort* para identificação de tráfego de uma aplicação.

5. CONSIDERAÇÕES FINAIS

As soluções de TI são crucias para eficácia das organizações. E as redes de computadores são umas das soluções de TI mais importantes para empresas, independente do seu porte. Em contrapartida o uso das redes e principalmente a Internet pode pôr em risco a segurança da informação e proporcionar uso indevido dos recursos disponibilizados. Nessa realidade a identificação de aplicações utilizadas em uma rede torna o gerenciamento mais simplificado e eficaz.

O objetivo deste trabalho foi demonstrar uma alternativa para identificar o fluxo de dados de uma aplicação em uma rede local, utilizando tecnologia de código aberto, sendo possível a inclusão de novas regras para identificação de tráfego de aplicações não catalogadas. O software escolhido foi o IDS/IPS Snort implementado com o pré-processador OpenAppID.

O desenvolvimento desse trabalho contribuiu significativamente para o entendimento do processo de detecção de aplicações em uma rede. Através do estudo das alternativas utilizados, foi possível investigar as características relevantes para identificação aplicações com a utilização de software livre, sem custos de aquisições de licenças. Sendo assim uma alternativa para um cliente que não dispõe de capital para investimento em soluções proprietárias.

No que diz respeito ao objetivo central do estudo, acreditasse que o objetivo foi alcançado e, pois, foi possível a identificação de aplicações catalogadas pelo IDS Snort integrado com o pré-processador OpenAppID, bem como a inclusão de uma nova aplicação não presente no referido catalogo.

Na realização do estudo houve dificuldade para encontrar referências teóricos sobre o pré-processador OpenAppID. Outro ponto de dificuldade foi a escolha de qual função utilizar para identificar uma nova aplicação, pois existem poucas informações das funções, as quais só podem ser visualizadas antes da compilação do pré-processador.

5.1. Trabalhos futuros

Em pesquisas futuras, propõem-se um estudo mais aprofundado sobre a funcionalidade de identificação de aplicações, bem como de outras funcionalidades

presentes em um NGFW, como o desenvolvimento de uma interface gráfica visando um melhor gerenciamento das ferramentas utilizadas e tratamento dos logs, para facilitar a utilização do conjunto proposto. E testar outras aplicações mais complexas, utilizando as funções presentes na API Lua.

REFERÊNCIAS BIBLIOGRÁFICAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT NBR ISO/IEC 27001:2013:** Tecnologia da informação - Técnicas de segurança - Sistemas de gestão de segurança da informação - Requisito. Rio de Janeiro, 2013.

CISCO NETWORKING ACADEMY (Org.). **Introduction to Networks:** Companion Guide v5.1. 2016. Disponível em: http://ptgmedia.pearsoncmg.com/images/9781587133572/samplepages/9781587133572.pdf>. Acesso em: 26 maio 2018.

CISCO. **What Is Network Security?**. Disponível em: <https://www.cisco.com/c/en/us/products/security/what-is-network-security.html>. Acesso em: 03 fev. 2018.

COSTELLO, Hector. **2018 Next Generation *Firewall* Market Report By Applications (Network, Database, Application Layer *Firewalls*) Types, Segmentation & Forecast By 2025.** 2018. Disponível em: <https://marketersmedia.com/2018-next-generation-*firewall*-market-report-by applications-network-database-application-layer-*firewalls*-types-segmentation forecast-by-2025/301765>. Acesso em: 07 abr. 2018.

FORCEPOINT. **Gartner 2017 Magic Quadrant for Enterprise Network *Firewalls***. Disponível em: <https://www.forcepoint.com/resources/industry-analyst reports/gartner-2017-magic-quadrant-enterprise-network-*firewalls*>. Acesso em: 13 jan. 2018.

GETCARD. **O que é um Next Generation *Firewall* (NGFW)?**. Disponível em: <www.getcard.com.br/novo/o-que-e-um-next-generation-*firewall*-ngfw/>. Acesso em: 07 abr. 2017.

GHAFIR, Ibrahim et al. **A Survey on Network Security Monitoring Implementations.** 2016. Disponível em: <https://www.researchgate.net/profile/Vaclav_Prenosil2/publication/309229246_A_Survey_on_Network_Security_Monitoring_Systems/links/58a5ad4b4585150402d2918d/A-Survey-on-Network-Security-Monitoring-Systems.pdf>. Acesso em: 13 dez. 2018.

IERUSALIMSCHY, Roberto; CELES, Waldemar; FIGUEIREDO, Luiz Henrique de. **A Linguagem de Programação Lua.** 2017. Disponível em: <https://www.lua.org/portugues.html>. Acesso em: 25 nov. 2017.

IPV6.BR. **Cabeçalho**. Disponível em: <http://ipv6.br/post/cabecalho/>. Acesso em: 03 fev. 2018.

KLEOPA, Costas; JUDGE, Cliff. ***OpenAppID*:** Open Source Community Webinar. 2015. Disponível em: <https://*Snort*.org/documents/*OpenAppID*-detection-webinar>. Acesso em: 02 dez. 2017.

KUROSE, James; ROSS, Keith. **Redes de Computadores e a Internet**: uma abordagem top-down. 6ª ed. São Paulo: Pearson Education do Brasil, 2014. 658 p.

MILLER, Lawrence C.. **Next-Generation *Firewalls* For Dummies**: *Palo Alto* Networks 2nd Edition. 2 ed. New Jersey: John Wiley & Sons, Inc., 2016. 77 p.

MORAES, Alexandre Fernandes De. ***Firewalls***: segurança no controle de acesso. 1ª ed. São Paulo: Érica, 2015. 120 p.

NETFILTER.ORG. **What is netfilter.org?** Disponível em: <https://www.netfilter.org/>. Acesso em: 01 dez. 2018.

PALO ALTO NETWORKS. ***PALO ALTO* NETWORKS:** App-ID Technology Brief. 2011. Disponível em: <https://www.hitachi solutions.co.jp/paloalto/sp/download/download_files/data_sheet/en/41/App_ID_tech pdf>. Acesso em: 12 mai. 2018.

RFC 791: **Internet Protocol Darpa Internet Program Protocol Specification Set 1981**. Disponível em: <www.ietf.org/rfc/rfc791.txt>. Acesso em 11 de novembro de 2018.

ROUSE, Margaret. ***OpenAppID*.** 2014. Disponível em: <https://searchsecurity.techtarget.com/definition/*OpenAppID*>. Acesso em: 15 dez. 2018.
ROUSE, Margaret. **next-generation *firewall* (NGFW)**. 2014. Disponível em: <http://searchsecurity.techtarget.com/definition/next-generation-*firewall*-ngfw>. Acesso em: 03 abr. 2017.

SEGINFO. **Gartner estima que investimento em Segurança pode chegar a 93 bilhões de dólares em 2018, mas o que está faltando?**. Disponível em: <https://seginfo.com.br/2017/08/30/gartner-estima-que-investimento-em-seguranca pode-chegar-a-93-bilhoes-de-dolares-em-2018-mas-o-que-esta-faltando/>. Acesso em: 15 jan. 2018.

SNORT. ***Snort* 3 User Manual**. Disponível em: <https://www.*Snort*.org/downloads/snortplus/*Snort*_manual.pdf>. Acesso em: 26 mai. 2018.

SOURCEFIRE CISCO. **Open Source Detectors:** Developers Guide. 2014. Disponível em: <*https://Snort.org/downloads/OpenAppID/7610*>. Acesso em: 02 dez. 2017.

STALLING, William; BROWN, Lawrie. **Segurança de computadores:** princípios e práticas. 2. ed. Rio de Janeiro: Elsevier, 2014. Tradução de: Arlete Simille Marques.

STALLINGS, William. **Criptografia e segurança de redes:** princípios e práticas. 6.ed. São Paulo: Pearson Education do Brasil, 2015. Tradução de: Daniel Vieira.

TECHTARGET. **TCP/IP (Transmission Control Protocol/Internet Protocol)**. Disponível em: <https://searchnetworking.techtarget.com/definition/TCP-IP>. Acesso em: 03 fev. 2018.

TECHTARGET. **The five different types of *firewalls***. Disponível em: <https://searchsecurity.techtarget.com/feature/The-five-different-types-of-*firewalls*>. Acesso em: 03 fev. 2018.

WANG, Xiang. **Hyperscan and *Snort** Integration.** 2017. Disponível em: <https://software.intel.com/en-us/articles/hyperscan-and-*Snort*-integration>. Acesso em: 13 dez. 2018.

APÊNDICE A - Instalação Snort e OpenAppID

1. Introdução

Este documento descreve a instalação do *Snort* e *OpenAppID*. Abrange a descrição e as compilações das bibliotecas[10] necessárias.

2. Pré-requisitos

Na lista a seguir estão descritos os principais requisitos necessários para instalação do *Snort* e *OpenAppID*:

- *build-essential*: fornece ferramentas de compilação (GCC e similares) para compilar software.
- *autotools* ou *cmake*: para construir pacote de software a partir da fonte;
- *libdumbnet-dev*: a biblioteca *libdumbnet* fornece uma interface portátil simplificada para várias rotinas de rede de baixo nível.
- *LuaJIT(libluajit-5.1-dev):* configuração e scripts;
- *Libpcap-dev*: Biblioteca para captura de tráfego de rede requerida pelo *Snort*.
- *libpcre3-dev*: Biblioteca de funções para suportar expressões regulares exigidas pelo *Snort*.
- *zlib1g-dev*: uma biblioteca de compressão requerida pelo *Snort*.
- *liblzma-dev*: fornece descompressão de arquivos swf (adobe flash)
- *openssl* e *libssl-dev*: assinaturas de arquivos SHA e MD5 e detecção de serviço SSL;
- *bison*, *flex*: analisadores requeridos pelo DAQ (o DAQ está instalado mais adiante).
- daq: biblioteca de aquisição de dados, para pacotes de E/S;
- g++ >= 4.8: compilador C++11;

[10] Em Sistemas operacionais Linux Bibliotecas são arquivos terminados em ".s" ou ".a" e tem como função auxiliar programadores, fazendo com que partes nativas do Sistema Operacional Linux sejam integrados aos seus programas.

Abaixo seguem comandos para instalar todas as dependências, requeridas no manual do *Snort*, disponíveis nos repositórios do Ubuntu 16:

```
$ sudo apt update && sudo apt install -y build-essential autotools-dev libdumbnet-dev libluajit-5.1-dev libpcap-dev libpcre3-dev zlib1g-dev pkg-config libhwloc-dev cmake liblzma-dev openssl libssl-dev cpputest libsqlite3-dev uuid-dev bison flex libtool git autoconf asciidoc dblatex source-highlight w3m libnetfilter-queue-dev libnghttp2-dev
```

3. Compilação das bibliotecas

Para o funcionamento eficiente do *Snort* é necessário a compilação das seguintes bibliotecas: *daq, libsafec, ragel, boost, hyperscan e flatbuffers.*

A biblioteca de Aquisição de Dados(DAQ) substitui as chamadas diretas para as funções *Libpcap* por uma camada de abstração que facilita a operação em uma variedade de interfaces de hardware e software sem exigir alterações no *Snort*. Foi realizada a instalação da versão daq-2.06, baixada do site do *Snort*:

```
$ cd ~/snort_src
$ wget https://www.Snort.org/downloads/Snort/daq-2.0.6.tar.gz
$ tar -xvzf daq-2.0.6.tar.gz
$ cd daq-2.0.6.tar.gz
$ ./configure && make && sudo make install
```

A Biblioteca Safe C fornece funções para verificação adicional de erros de tempo de execução de algumas operações de cópia de memória em determinadas chamadas legadas da biblioteca C:

```
$ cd ~/snort_src
$ wget
http://downloads.sourceforge.net/project/safeclib/libsafec-10052013.tar.gz
$ tar -xzvf libsafec-10052013.tar.gz
$ cd libsafec-10052013
$ ./configure && make && sudo make install
```

O *Snort* usará o *Hyperscan* e este requer os cabeçalhos *Ragel* e *Boost*. Deve-se baixar e compilar o *Ragel* e baixar o *Boost*, mas não realizar a compilação:

```
$ cd ~/snort_src
$ wget http://www.colm.net/files/ragel/ragel-6.10.tar.gz
$ tar -xzvf ragel-6.10.tar.gz
$ cd ragel-6.10
```

```
$ ./configure && make && sudo make install

$cd ~/snort_src
$ wget
https://dl.bintray.com/boostorg/release/1.65.1/source/boost_1_
65_1.tar.gz
$ tar -xvzf boost_1_65_1.tar.gz
```

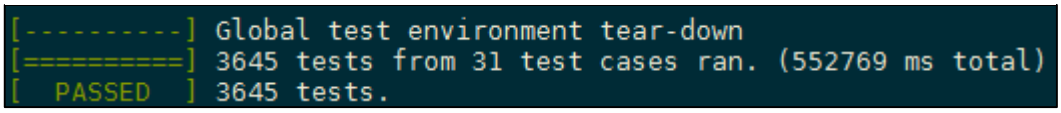
O *Hyperscan* é um mecanismo de expressão regular projetado para oferecer alto desempenho, capacidade de combinar várias expressões simultaneamente e flexibilidade na operação de varredura. Os padrões são fornecidos para uma interface de compilação que gera um banco de dados padrão imutável. A interface de varredura pode então, ser usada para varrer um buffer de dados de destino para os padrões fornecidos, retornando quaisquer resultados correspondentes desse buffer de dados. Será instalado o *Hyperscan* 4.6.0, referenciando a localização do diretório de origem dos cabeçalhos do *Boost*:

Testar o funcionamento do *Hyperscan*, no diretório de compilação executar:

```
$ cd ~/snort_src/hyperscan-4.6.0-build/
$ ./bin/unit-hyperscan
```

A saída deverá ser parecida coma figura abaixo:

Figura 22 - Saída Hyperscan



Fonte: Print bash Ubuntu (2018)

O *FlatBuffers* é uma eficiente biblioteca de serialização em memória entre plataformas para C ++, C #, C, Go, *Java*, *JavaScript*, PHP e *Python*.

```
$ cd ~/snort_src
$ wget
https://github.com/google/flatbuffers/archive/master.tar.gz -O
flatbuffers-master.tar.gz
$ tar -xvzf flatbuffers-master.tar.gz
$ mkdir flatbuffers-build
$ cd flatbuffers-build
$ cmake ../flatbuffers-master && make && sudo make install
```

4. Instalação do *Snort* com suporte ao *OpenAppID*

Compiladas as bibliotecas requeridas, executa-se o comando *ldconfig*. Esse comando cria os links e o caches necessários para as bibliotecas compartilhadas mais recentes encontradas nos diretórios confiáveis (**`/usr/lib`** e **`/lib`**). O *ldconfig* verifica os nomes de cabeçalho e arquivos das bibliotecas que encontra ao determinar quais versões devem ter seus links atualizados.

```
$ sudo ldconfig
```

Após a instalação dos pré-requisitos e compilação das bibliotecas, foi realizada a instalação do *Snort*. Nesse trabalho foi utilizada uma versão disponível no site oficial (versão 2.9.12). Na compilação foi usada a opção "**--enable-open-appid"**, a qual prepara o *Snort* para ser construído com suporte ao *OpenAppID*.

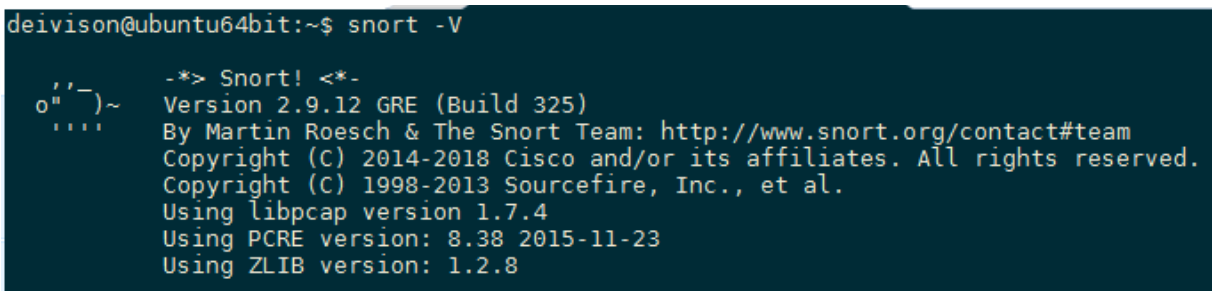
```
$ cd ~snort_src
$ wget https://Snort.org/downloads/Snort/Snort-2.9.12.tar.gz
$ tar -xvzf Snort-2.9.12.tar.gz
$ cd Snort-2.9.12
$ ./configure --enable-sourcefire --enable-open-appid
$ make && sudo make install
$ sudo ldconfig
```

Finalizada a compilação foi criado um link simbólico do binário do *Snort* em **`/usr/sbin/snort`**, através do comando:

```
$ sudo ln -s /usr/local/bin/sSnort /usr/sbin/snort
```

Após as etapas acima, verificou-se a compilação, executando o binário do *Snort* com a opção **–V**, assim o *Snort* imprime na tela a versão instalada conforme a figura 23:

Figura 23 - Versão instalada do Snort



Fonte: Print bash Ubuntu (2018)

5. Ajustes e Configurações no *Snort*

Por razões de segurança, o *Snort* deve ser executado como um usuário sem privilégios. Então foi criado um usuário e grupo *Snort* para este propósito:

```
$ sudo groupadd snort
$ sudo useradd snort -r -s /sbin/nologin -c SNORT_IDS -g snort
```

É necessário a criação de alguns arquivos, diretórios e alterar permissões para o usuário criado acima. Um *script* chamado **basic_config_dir_*Snort*.sh** foi criado para agilizar a configuração, o *script* encontrasse no Apêndice B.

O arquivo de configuração do *Snort* 2.9.12 está armazenado em **/etc/snort/snort.conf**. Este arquivo possui mais de 500 linhas e várias opções para a configuração. É necessário comentar as linhas que fazem com que o *Snort* importe o conjunto padrão de arquivos de regras. Para comentar as linhas foi executando o seguinte comando:

```
$ sudo sed -i 's/include \$RULE\_PATH/#include \$RULE\_PATH/'
/etc/snort/snort.conf
```

Editando manualmente o arquivo **/etc/snort/snort.conf**, foi alterada a variável **HOME_NET** para corresponder à rede interna(LAN) utilizada neste cenário. No cenário proposto, a **HOME_NET** é **10.0.0.0** com máscara de sub-rede de **24 bits (255.255.255.0)**. A linha deve ficar da seguinte forma:

```
Linha 45 - ipvar HOME_NET 10.0.0.0/24
```

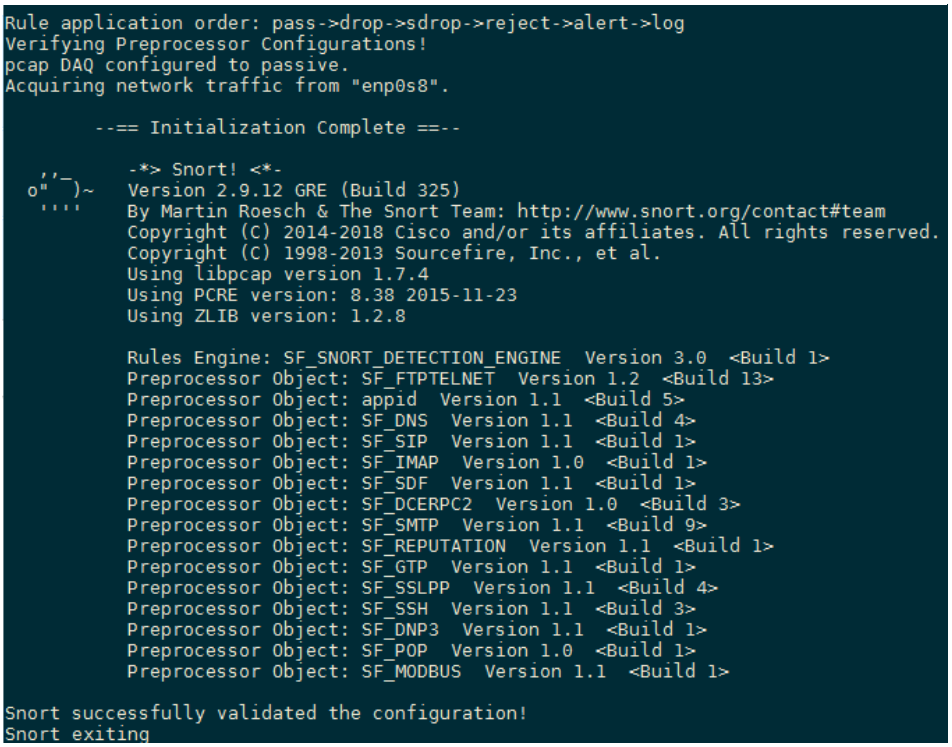
Ainda no arquivo de configuração modificam-se alguns caminhos de arquivos para corresponder com os criados anteriormente:

```
Linha 104 - var RULE_PATH /etc/snort/rules
Linha 105 - var SO_RULE_PATH /etc/snort/so_rules
Linha 106 - var PREPROC_RULE_PATH /etc/snort/preproc_rules
Linha 113 - var WHITE_LIST_PATH /etc/snort/rules/iplists
Linha 114 - var BLACK_LIST_PATH /etc/snort/rules/iplists
Linha 546 - include $RULE_PATH/local.rules
```

Finalizada a edição do arquivo é possível validá-la com o comando a seguir, onde a opção **-T** diz ao *Snort* para testar, **-** indica o caminho do arquivo de configuração e o **-i** específica a interface, que neste cenário é **enp0s8**. Na figura 24 está a saída da validação realizada com o comando a seguir:

```
$ sudo snort -T -i enp0s8 -c /etc/snort/snort.conf
```

Figura 24 - Validação da configuração do Snort



Fonte: Print bash Ubuntu (2018)

6. Habilitando o *OpenAppID*

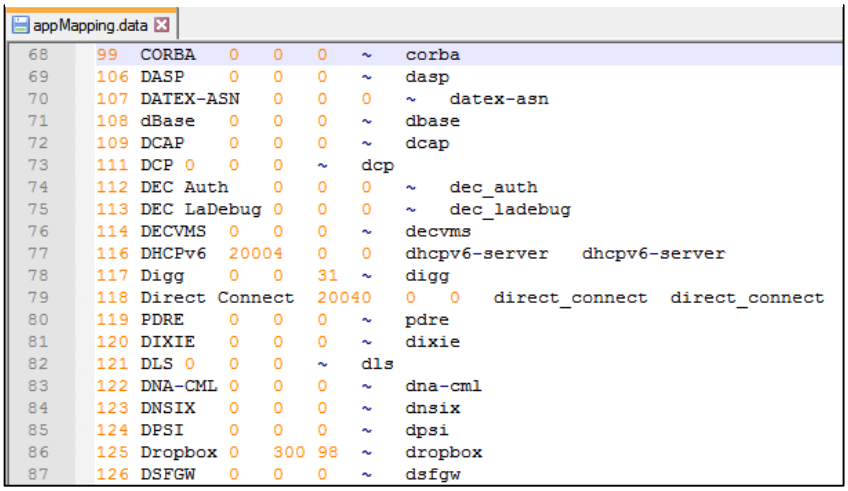
O *OpenAppID* encontra-se para download no sitio do *snort*.org, listada como ***Snort-OpenAppID*.tar.gz**. A versão utilizada neste trabalho é a 303, mas provavelmente mudará, pois, a equipe do *Snort* atualiza o arquivo regularmente. Dentro do pacote existe um diretório chamado *Application Detector Package,* o qual contém as regras para detectar tipos de tráfego de aplicativos catalogados pelo *Snort*.org e a comunidade.

O *Application Detector Package* é um pacote que contém os arquivos detectores na linguagem Lua, aplicativos e *metadado*s, nele estão os seguintes componentes:

- Detectores de aplicativos na linguagem Lua (.lua);

- Detectores de portas, que são detectores de aplicativos somente de porta, em metadados no formato YAML (*Yet Another Markup Language*) [11];
- Arquivo da biblioteca Lua, DetectorCommon.lua;
- O arquivo *appMapping.data* contendo *metadados* de aplicativos. Este arquivo não deve ser modificado. A primeira coluna contém o identificador do aplicativo e a última coluna contém o nome do aplicativo. Outras colunas contêm informações internas, conforme figura abaixo:

Figura 25 - Trecho do conteúdo do arquivo “appMapping.data”



Fonte: Print arquivo appMapping.data (2018)

Os comandos abaixo foram executados para baixar o diretório de detectores do *OpenAppID***:**

```
$ cd ~/snort_src/
$ wget https://www.snort.org/downloads/openappid/8373 -O OpenAppId-8373.tar.gz
$ tar -xzvf OpenAppId-8373.tar.gz
```

Após a execução dos comandos, será criado o diretório **odp**. O mesmo deve ser movido para o diretório de regras do *Snort*:

```
$ sudo cp -r ~/snort_src/odp/ /etc/snort/rules/
```

[11] YAML é uma linguagem de serialização de dados projetada para ser diretamente gravável e legível por seres humanos.

Para habilitar o *OpenAppID*, o *Snort* precisa enviar os dados do AppId da aplicação para o pré-processador. Essa configuração é realizada editando-se o arquivo ***Snort*.conf**, e adicionando as seguintes linhas antes da seção 6:

```
# Linha 518
preprocessor appid: app_stats_filename appstats-u2.log, \
    app_stats_period 10, \
    app_detector_dir /etc/snort/rules
```

As linhas informam ao *Snort* o nome do arquivo de *log* para gerar estatísticas (*appstats-u2.log*), com que frequência gravar no *log* (a cada 10 segundos) e onde encontrar o diretório **odp**.

Ainda no arquivo **snort.conf**, foi adicionada a linha a seguir, abaixo da seção 6 do arquivo de configuração:

```
# Linha 646
output unified2: filename snort.log, limit 128, appid_event_types
```

A diretiva informa ao *Snort* para enviar alertas no formato binário *unified2* para o *Snort*.log, o tamanho do *log* e para enviar dados do AppId para o mesmo local.

Finalizada a edição foi valida-se à configuração:

```
$ sudo snort -T -i enp0s8 -c /etc/snort/snort.conf
```

Para iniciar a coleta de pacotes, usa-se o comando abaixo e ctrl-c para parar a mesma:

```
$ snort -c /etc/snort/snort.conf --daq afpacket -i enp0s8 -k none
```

Os dados devem estar visíveis para a interface que o *Snort* está escutando. Assim que coletado os dados (os dados estão sendo gravados a cada 10 segundos no log conforme configuração), deverá haver em **`/var/log/snort/`** um ou vários arquivos com o nome no seguinte padrão: **`appstats-u2.log.xxxxxxxxxx`** (onde os ***x*** são números). Estes são os arquivos de dados do *OpenAppID*, para poder processa-los utiliza-se o binário u2*OpenAppID*, localizado em **`/usr/local/bin`**.

APÊNDICE B – Shell script firewall Iptables Statefull

Abaixo segue *Sheel script* utilizado no cenário:

```
#!/bin/bash
# Interfaces
WAN="enp0s3"
LAN="enp0s8"
IPWAN="192.168.0.254"

# Limpar Regras
iptables -F
iptables -X
iptables -Z

# Politica Padrao
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

# Regra na tabela nat, cadeia POSTROUTING para habilitar o
mascaramento de IP (IP masquerading) na interface WAN.
iptables -t nat -A POSTROUTING -o $WAN -j MASQUERADE

# Regras INPUT
iptables -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j
ACCEPT
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m conntrack --ctstate INVALID -j DROP
iptables -A INPUT -p icmp -m icmp --icmp-type 8 -m conntrack --
ctstate NEW -j ACCEPT
iptables -A INPUT -p udp -m conntrack --ctstate NEW -j ACCEPT
iptables -A INPUT -p tcp --tcp-flags FIN,SYN,RST,ACK SYN -m
conntrack --ctstate NEW -j ACCEPT
iptables -A INPUT -p udp -j REJECT --reject-with icmp-port-
unreachable
iptables -A INPUT -p tcp -j REJECT --reject-with tcp-reset
iptables -A INPUT -j REJECT --reject-with icmp-proto-unreachable

# Regras FORWARD
iptables -A FORWARD -m conntrack --ctstate NEW,ESTABLISHED -j
ACCEPT

# DNAT RDP
iptables -t nat -A PREROUTING -i $WAN -p tcp -d $IPWAN --dport
3389 -j DNAT --to 10.0.0.5:3389
```

APÊNDICE C – Script de configuração

basic_config_dir_Snort.sh

```
#!/bin/sh
## Cria os diretórios padões de regras do Snort:
sudo mkdir -p /etc/snort/rules/iplists
sudo mkdir /etc/snort/preproc_rules
sudo mkdir /etc/snort/so_rules
sudo mkdir /usr/local/lib/snort_dynamicrules

## Arquivos que armazenam regras e listas de IP:
sudo touch /etc/snort/rules/iplists/black_list.rules
sudo touch /etc/snort/rules/iplists/white_list.rules
sudo touch /etc/snort/rules/local.rules
sudo touch /etc/snort/sid-msg.map

## Criar diretório de log:
sudo mkdir -p /var/log/snort/archived_logs

## Ajustar permissões:
sudo chmod -R 5775 /etc/snort
sudo chmod -R 5775 /var/log/snort
sudo chmod -R 5775 /var/log/snort/archived_logs
sudo chmod -R 5775 /etc/snort/so_rules
sudo chmod -R 5775 /usr/local/lib/snort_dynamicrules

## Alterar proprietários nos diretórios:
sudo chown -R snort:snort /etc/snort
sudo chown -R snort:snort /var/log/snort
sudo chown -R snort:snort /usr/local/lib/snort_dynamicrules

## Mover os arquivos extraídos para a pasta de configuração do
Snort:
cd ~/snort_src/snort-2.9.12/etc/
sudo cp *.conf* /etc/snort
sudo cp *.map /etc/snort
sudo cp *.dtd /etc/snort
cd ~/snort_src/snort-2.9.12/src/dynamic-
preprocessors/build/usr/local/lib/snort_dynamicpreprocessor/
sudo cp * /usr/local/lib/snort_dynamicpreprocessor/
```

APÊNDICE D – Código do detector para aplicação Mira

```
--[[
detection_name: Mira
version: 1
description: Testing
--]]

require "DetectorCommon"
local DC = DetectorCommon

local proto = DC.ipproto.tcp;
DetectorPackageInfo = {
        name = "Mira",
        proto = proto,
        server = {
                init = 'DetectorInit',
                clean = 'DetectorClean',
                minimum_matches = 1
        }
}

function DetectorInit(detectorInstance)

        gDetector = detectorInstance;
        gAppId = gDetector:open_createApp("Mira");

        if gDetector.addPortPatternService then

gDetector:addPortPatternService(proto,212,"\004\219\000\000\01
5\000\000\000\008",0, gAppId);
        end
        return gDetector;
end

function DetectorClean()
end
```