UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS, TECNOLOGIAS E SAÚDE
TECNOLOGIAS DA INFORMAÇÃO E COMUNICAÇÃO

Guilherme Bandeira Dolácio

MITIGAÇÃO DE ATAQUES EM REDES DEFINIDAS POR SOFTWARE UTILIZANDO O SISTEMA DE DETECÇÃO DE INTRUSÃO BASEADO EM REDE SNORT

Araranguá
2021

Guilherme Bandeira Dolácio

MITIGAÇÃO DE ATAQUES EM REDES DEFINIDAS POR SOFTWARE UTILIZANDO O SISTEMA DE DETECÇÃO DE INTRUSÃO BASEADO EM REDE SNORT

Trabalho de Conclusão do Curso de Graduação em Tecnologias da Informação e Comunicação do Centro de Ciências, Tecnologia e Saúde da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Tecnologias da Informação e Comunicação.
Orientador: Prof. Vilson Gruber, Dr.

Araranguá
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Dolácio, Guilherme Bandeira
Mitigação de ataques em redes definidas por software utilizando o sistema de detecção de intrusão baseado em rede Snort / Guilherme Bandeira Dolácio ; orientador, Vilson Gruber, 2021.
63 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Araranguá, Graduação em Tecnologias da Informação e Comunicação, Araranguá, 2021.

Inclui referências.

1. Tecnologias da Informação e Comunicação. 2. Redes Definidas por Software. 3. Mitigação de ataques. 4. Sistemas de Detecção de Intrusão. I. Gruber, Vilson. II. Universidade Federal de Santa Catarina. Graduação em Tecnologias da Informação e Comunicação. III. Título.

Guilherme Bandeira Dolácio

MITIGAÇÃO DE ATAQUES EM REDES DEFINIDAS POR SOFTWARE UTILIZANDO O SISTEMA DE DETECÇÃO DE INTRUSÃO BASEADO EM REDE SNORT

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Tecnologias da Informação e Comunicação" e aprovado em sua forma final pelo Curso de Graduação em Tecnologias da Informação e Comunicação.

Araranguá, 17 de maio de 2021.

Documento assinado digitalmente
Vilson Gruber
Data: 19/05/2021 10:46:50-0300
CPF: 175.317.788-07
Verifique as assinaturas em https://v.ufsc.br

Prof. Vilson Gruber, Dr.
Coordenador do Curso

Banca Examinadora:

Documento assinado digitalmente
Vilson Gruber
Data: 19/05/2021 10:47:08-0300
CPF: 175.317.788-07
Verifique as assinaturas em https://v.ufsc.br

Prof. Vilson Gruber, Dr.
Orientador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Giovani Mendonca Lunardi
Data: 19/05/2021 11:17:38-0300
CPF: 520.394.559-49
Verifique as assinaturas em https://v.ufsc.br

Prof. Giovani Mendonça Lunardi, Dr.
Avaliador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
Alix Ribeiro da Silva
Data: 19/05/2021 11:47:25-0300
CPF: 004.838.842-43
Verifique as assinaturas em https://v.ufsc.br

Alix Ribeiro da Silva, MSc.
Avaliador
Universidade Federal de Santa Catarina

Este trabalho é dedicado aos meus queridos pais, familiares e amigos.

AGRADECIMENTOS

Agradeço meus pais, familiares e amigos pelo apoio incondicional durante esta jornada.

Agradeço meu orientador e professor Dr. Vilson Gruber pelo grande apoio que me forneceu durante a graduação e na realização deste trabalho.

Por fim, agradeço a Universidade Federal de Santa Catarina pela oportunidade fornecida e meus professores pelo ensino de qualidade proporcionado.

RESUMO

As Redes Definidas por Software (do inglês SDN) surgiram como um novo modelo de operações de rede de computadores para atender as crescentes demandas atuais das redes, que requerem rapidez na implementação e resposta a mudanças. Em vista disso, o trabalho tem como objetivo geral investigar o uso do sistema de detecção de intrusão baseado em rede Snort em um ambiente de rede SDN para verificar sua eficácia no auxílio à mitigação de ataques destinados à rede. Para tal, inicialmente foi efetuada uma pesquisa bibliográfica e documental para estudo sobre as áreas envolvidas e para buscar as possíveis ferramentas capazes de realizar os testes executados no trabalho. Diante desta pesquisa, foi realizada a simulação de um ambiente virtualizado de rede SDN com o controlador da rede integrado ao Snort, onde foram executados ataques cibernéticos para a coleta e análise quali-quantitativa de resultados sobre a habilidade desta ferramenta de segurança em contribuir na detecção e mitigação dos ataques emitidos à rede. Diante disso, os resultados demonstraram que o Snort conseguiu detectar os ataques executados no ambiente e enviar alertas ao controlador da rede, portanto, concluiu-se que a ferramenta Snort conseguiu efetivamente desempenhar o seu papel na segurança da rede SDN.

Palavras-chave: Redes Definidas por Software. SDN. Sistema de detecção de intrusão baseado em rede. Snort. Mitigação de ataques de rede.

ABSTRACT

Software-defined Networking (SDN) emerged as a new model to operate computer networks to meet the growing current demands of networks, which require speedy implementation and response to changes. In view of this, this work aims to investigate the use of the Snort network-based intrusion detection system in an SDN network environment to verify its effectiveness in helping to mitigate attacks aimed at the network. To this end, a bibliographic and documentary research was initially carried out to study the areas involved and to search for possible tools capable of carrying out the tests performed in this work. On account of this research, a simulation of a virtualized SDN network environment was performed with the network controller integrated to Snort, where cyberattacks were performed for the collection and qualitative and quantitative analysis of results on the ability of this security tool to contribute to the detection and mitigation of the attacks emitted to the network. The results showed that Snort was able to detect attacks performed in the environment and send alerts to the network controller, therefore, it was concluded that the Snort tool was able to effectively play its role in the security of the SDN network.

Keywords: Software-defined Networking. SDN. Network-based Intrusion Detection System. Snort. Mitigation of network attacks.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ACK	*Acknowledgement*
API	*Application Programming Interface*
CPU	*Central Processing Unit*
DPID	*Datapath Identifier*
FIN	*Finish*
GB	*Gigabyte*
HTTP	*Hypertext Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IDS	*Intrusion Detection System*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
LTS	*Long-term Support*
MAC	*Media Access Control*
NIDS	*Network-based Intrusion Detection System*
RAM	*Random-access Memory*
REST	*Representational State Transfer*
RST	*Reset*
SDN	*Software-defined Networking*
SSD	*Solid-state Drive*
SSH	*Secure Shell Protocol*
SYN	*Synchronization*
TCP	*Transmission Control Protocol*
TI	Tecnologia da Informação
UDP	*User Datagram Protocol*
URI	*Uniform Resource Identifier*
VM	*Virtual Machine*

SUMÁRIO

1 INTRODUÇÃO

Com o constante avanço das tecnologias computacionais e da Internet, as infraestruturas de rede se encontram cada vez maiores. Essa expansão se dá devido a atual necessidade de elas suportarem uma crescente quantidade de tráfego originando de diversos dispositivos acessando cada vez mais diferentes tipos de serviços, como: redes sociais, *streaming* de vídeo, comércio eletrônico, transações bancárias, dentre outros (HUMAYUN *et al.*, 2020). Deste modo, as redes tradicionais tornam-se bastante complexas e difíceis de gerenciar, onde a configuração deve ser feita individualmente em cada dispositivo de rede, necessitando de um elevado período de tempo para planejar, implementar e reagir a mudanças (KREUTZ *et al.*, 2015).

Diante disso, um novo modelo de operações de rede foi introduzido para responder às limitações do modelo tradicional, nomeado Redes Definidas por Software, do inglês *Software-defined Networking* (SDN). Na arquitetura SDN, um controlador é utilizado para centralizar algumas funções da rede, o que habilita aplicações de software a dinamicamente configurar e operar redes, provendo uma maior consistência e rapidez na implementação e reação a mudanças e a adoção de melhores práticas operacionais (ODOM, 2020).

Com o aumento do uso das redes de computadores, a segurança desses dispositivos também necessita de atenção e evolução. Portanto, hoje em dia, a segurança cibernética tornou-se uma questão importante no mundo todo (VON SOLMS; VAN NIEKERK, 2013).

Este trabalho tem como objetivo utilizar o sistema de detecção de intrusão baseado em rede Snort em uma rede definida por software com o intuito de avaliar se a utilização desta ferramenta de segurança em um ambiente SDN é eficaz em ajudar a combater potenciais ataques destinados à rede.

Este trabalho justifica-se no atual cenário tecnológico, onde, segundo Cisco (2019), 25% dos líderes de empresas do segmento de TI afirmam que automação de rede e SDN estão entre as tecnologias que terão o maior impacto nas redes de computadores nos próximos anos, e, com a crescente utilização das redes de computadores, também cresceram os números e a variedade de ataques cibernéticos executados, fazendo com que um dos componentes mais importantes de uma rede seja a sua segurança (VERIZON, 2021). Portanto, com as novas oportunidades que a tecnologia SDN oferece, se amplia a possibilidade da investigação de novas maneiras de proteger as redes neste novo modelo.

O trabalho visa uma contribuição acadêmica de forma que amplie as formulações teóricas a respeito do tema, podendo utilizar as bases desenvolvidas neste trabalho para a possível solução de problemas relacionados e novos problemas.

A pesquisa analisará se o método de defesa aplicado nos experimentos é eficiente para mitigar ataques cibernéticos executados em uma rede definida por software. Desse modo, o problema de pesquisa refere-se a seguinte pergunta: a utilização do sistema de detecção de intrusão baseado em rede Snort em um ambiente de rede definida por software é eficaz no auxílio à mitigação de ataques destinados à rede?

Para tal, inicialmente foi efetuada uma pesquisa bibliográfica e documental para estudar as áreas envolvidas e buscar as possíveis ferramentas capazes de realizar os testes para atingir o objetivo do trabalho. Esta pesquisa baseou-se principalmente em livros, artigos científicos e *websites* das áreas de redes de definidas por software, sistemas de detecção de intrusão baseados em rede e ataques cibernéticos. Após esta pesquisa, foi realizada a simulação de um ambiente virtualizado de rede definida por software com o controlador da rede integrado ao sistema de detecção de intrusão baseado em rede Snort, onde foram feitos testes para a obtenção e análise de resultados quanto a habilidade desta ferramenta em auxiliar na detecção e mitigação de ataques emitidos à rede SDN. Uma abordagem quali-quantitativa foi utilizada para a análise dos resultados obtidos no trabalho. O método de análise escolhido para a solução do problema foi o hipotético-dedutivo.

1.1 OBJETIVOS

Nesta seção são descritos o objetivo geral e os objetivos específicos deste trabalho.

1.1.1 Objetivo Geral

O trabalho tem como objetivo geral investigar o uso do sistema de detecção de intrusão baseado em rede Snort em um ambiente simulado de rede definida por software para verificar a eficácia que a integração entre estas duas tecnologias proporciona para o reforço da segurança da rede.

1.1.2 Objetivos Específicos

Para o alcance do objetivo geral, o trabalho tem como objetivos específicos:

- Realizar a simulação de um ambiente de rede definida por software e executar ataques cibernéticos no ambiente para capturar e analisar os padrões de tráfego gerados pelos ataques;

- Configurar e integrar o sistema de detecção de intrusão baseado em rede Snort no ambiente SDN e elaborar regras de detecção no Snort para os ataques executados baseadas nos padrões detectados durante a análise do tráfego gerado pelos ataques;
- Executar novamente os ataques no ambiente, desta vez com o Snort integrado e as regras de detecção elaboradas ativas e bloquear a conectividade da máquina originadora dos ataques detectada pelo Snort;
- Coletar e analisar os dados obtidos na simulação e verificar a eficácia da utilização do Snort em uma rede SDN para o auxílio na mitigação de ataques destinados à rede.

1.2 ORGANIZAÇÃO DO TRABALHO

Além do presente capítulo de introdução, o trabalho é formado por outros quatro capítulos.

No segundo capítulo, é elaborada a fundamentação teórica, onde são abordados os principais conceitos essenciais para a compreensão do trabalho, como redes definidas por software, sistemas de detecção de intrusão baseados em rede e os ataques cibernéticos testados.

No terceiro capítulo, se encontra o desenvolvimento do trabalho, onde são relatados os procedimentos realizados para a obtenção dos resultados do mesmo.

No quarto capítulo, são apresentados os resultados obtidos através dos testes realizados no trabalho, onde são evidenciadas a detecção e mitigação dos ataques testados.

Por fim, no quinto capítulo, são realizadas as considerações finais do trabalho e feitas sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são introduzidos os principais conceitos necessários para o entendimento deste trabalho. Na seção 2.1, são apresentados conceitos sobre redes definidas por software, o protocolo OpenFlow e feita uma introdução ao controlador Ryu e o simulador de redes Mininet. Na seção 2.2, é abordado o conceito de sistemas de detecção de intrusão baseados em rede e realizada a introdução ao Snort e a sintaxe de suas regras de detecção. Por fim, na seção 2.3, os ataques cibernéticos selecionados para a execução dos testes são descritos, conjunto ao seu funcionamento.

2.1 REDES DEFINIDAS POR SOFTWARE

Conforme Kreutz *et al.* (2015), redes definidas por software é um emergente paradigma de redes de computadores que possui o potencial de superar as limitações das atuais infraestruturas de rede. A tecnologia SDN habilita aplicações de software a programar dinamicamente os dispositivos de rede, como roteadores e *switches*, viabilizando controlar o comportamento da rede como um todo (HALEPLIDIS *et al.*, 2015).

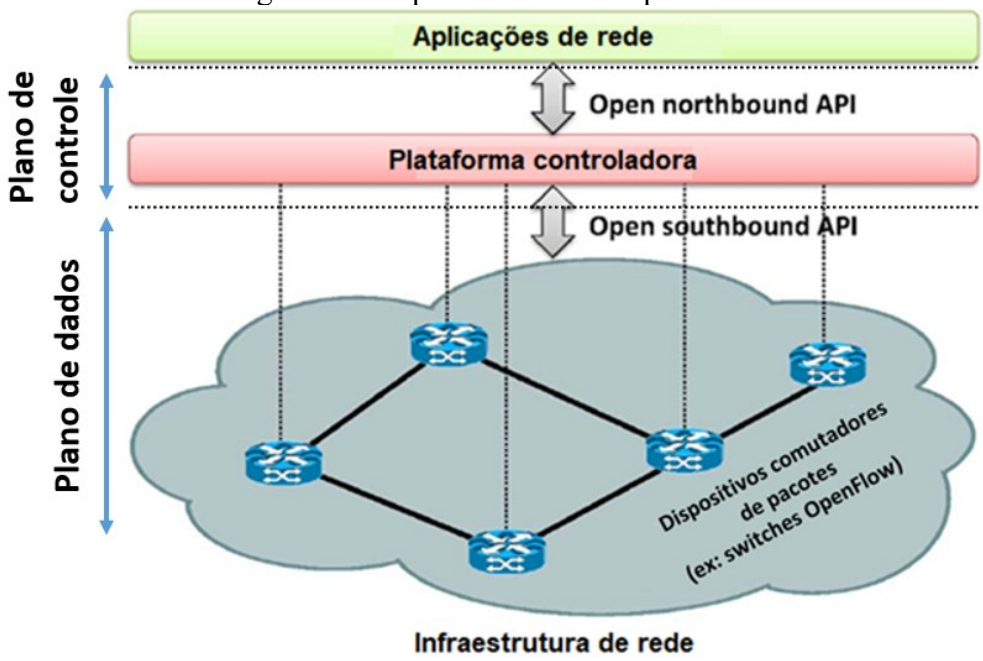
Isso é possibilitado através da separação, no dispositivo de rede, do seu plano de controle, que é a lógica que decide como tratar o tráfego, do seu plano de dados, que encaminha o tráfego de acordo com as decisões tomadas no plano de controle (KREUTZ *et al.*, 2015).

Nas redes tradicionais, ambos os planos são embutidos no mesmo dispositivo para a comutação de pacotes, em uma arquitetura de camada de software fechada, que é executada em hardware proprietário. Esse modelo acaba trazendo implementações de alto custo, onde a introdução de novas funcionalidades e experimentação de novas ideias são altamente dificultadas (ROTHENBERG *et al.*, 2011).

No modelo das redes definidas por software, o plano de controle passa a ser implementado em um ou mais controladores separados e logicamente centralizados. Assim, os roteadores e *switches* passam a ser apenas dispositivos encaminhadores de pacotes (KREUTZ *et al.*, 2015).

Essa separação entre o plano de controle e o plano de dados pode ser realizada através de uma Interface de Programação de Aplicações, do inglês *Application Programming Interface* (API), entre os dispositivos de rede e o controlador, onde o controlador gerencia o estado dos elementos no plano de dados por meio de suas APIs (KREUTZ *et al.*, 2015). Uma visualização simplificada da arquitetura SDN é exibida na Figura 1.

Figura 1 – Arquitetura SDN simplificada



Fonte: Adaptado de Kreutz *et al.* (2015).

Na arquitetura SDN, a rede é programável através de aplicações de rede executando conjuntamente ao controlador, que por sua vez interage com os dispositivos do plano de dados para programá-los (KREUTZ, *et al.*, 2015). O controlador interage com as aplicações e os dispositivos de rede através das APIs nomeadas *Southbound* e *Northbound* (ODOM, 2020).

A API *Northbound* é utilizada para possibilitar a comunicação entre o controlador e as aplicações de rede, onde os dados e funções do controlador possam ser utilizados pelas aplicações, que podem extrair informações do controlador ou configurar variáveis, como programar as tabelas de fluxos de rede dos dispositivos (ODOM, 2020).

Já API *Southbound* é a interface que habilita a comunicação entre o controlador e os dispositivos de rede, permitindo o controlador a programar diversas variáveis nos dispositivos (ODOM, 2020).

O paradigma SDN traz a facilidade que a programação oferece para a mudança das características de uma rede como um todo. A separação do plano de dados do plano de controle simplifica o gerenciamento, configuração e otimização dos recursos de rede, pois, como o plano de controle da rede é logicamente centralizado, os controladores possuem uma visão global da rede. Logo, através de programas dinâmicos e automatizados, os administradores de rede podem otimizar as configurações e aumentar a performance da rede (KARAKUS; DURRESI, 2017). Outro benefício da adoção do SDN é a facilitação da evolução e inovação nas redes. Estes benefícios aumentam a eficiência operacional de uma organização, potencialmente reduzindo custos (KIM; FEAMSTER, 2013).

A arquitetura SDN veio como resposta às limitações do modelo de operação das redes tradicionais, onde os administradores precisam configurar cada dispositivo de rede individualmente, utilizando comandos específicos para os dispositivos de cada fabricante. Desse modo, fica desafiador atuar em ambientes de rede altamente dinâmicos, onde responder a falhas e se adaptar a mudanças é constantemente necessário. Essa dificuldade aumenta o gasto capital e operacional de gerenciar uma rede (KREUTZ *et al.*, 2015).

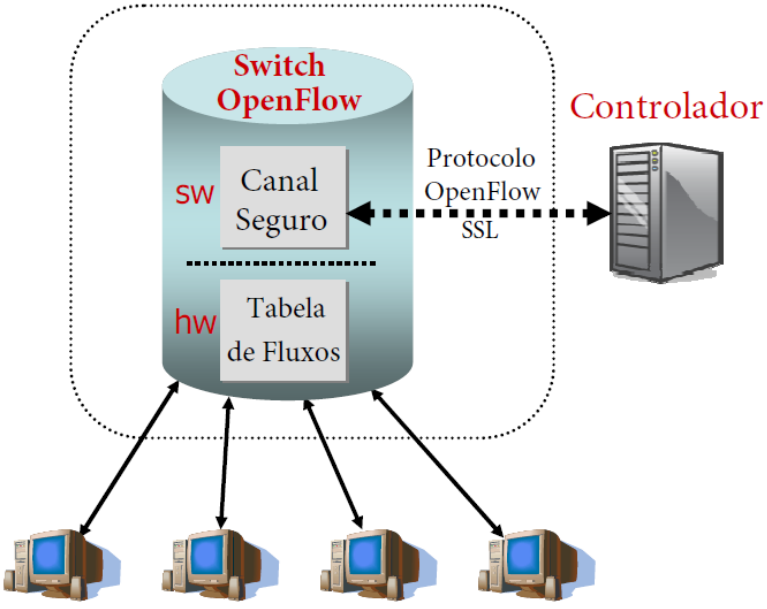
2.1.1 Protocolo OpenFlow

De acordo com McKeown (2008), o OpenFlow é um protocolo de código aberto que possibilita controladores a se comunicarem com os dispositivos de rede e programar suas tabelas de fluxos. Um fluxo consiste na combinação de campos do cabeçalho de um pacote, conjunto a uma ação associada que determina seu processamento pelo dispositivo de rede (ROTHENBERG *et al.*, 2011).

Segundo Rothenberg *et al.* (2011), um *switch* OpenFlow, ilustrado na Figura 2, é constituído de três partes:

- **Tabela de fluxos:** onde ficam armazenados os fluxos.
- **Canal seguro:** via de comunicação segura entre o *switch* e controlador, criptografada pelo protocolo SSL (*Secure Socket Layer*).
- **Protocolo OpenFlow:** o padrão de comunicação entre o controlador e o *switch*.

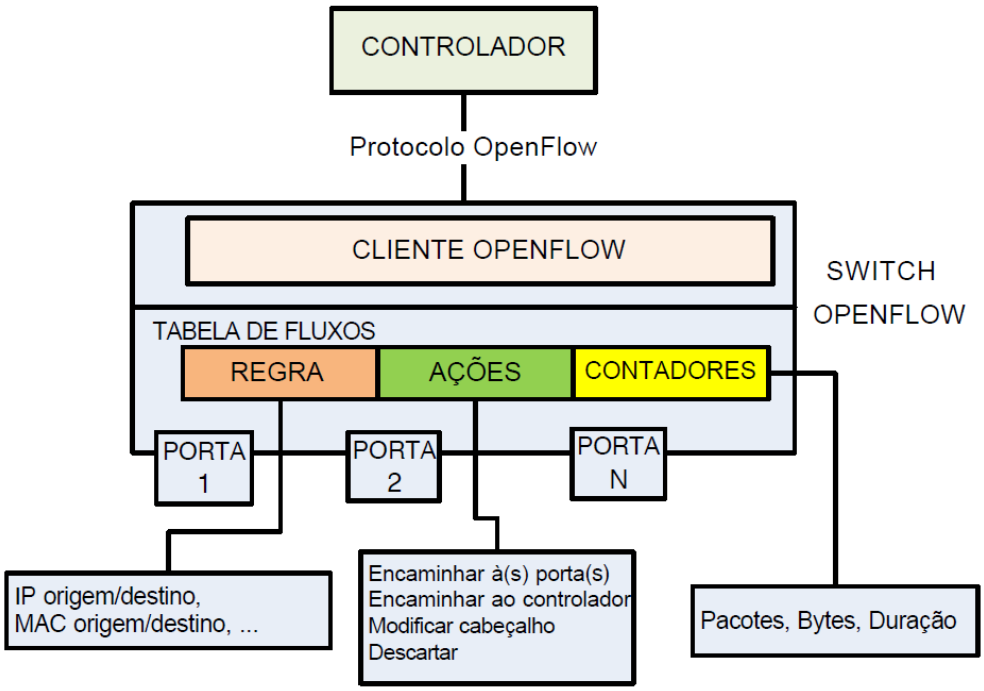
Figura 2 – Especificação do *switch* OpenFlow



Fonte: Adaptado de McKeown (2008).

Na arquitetura OpenFlow, ilustrada na Figura 3, o *switch* contém uma ou mais tabelas de fluxos, que consistem em entradas de fluxo, onde cada fluxo determina como os pacotes pertencentes a ele serão processados (NUNES *et al.*, 2014).

Figura 3 – Arquitetura OpenFlow



Fonte: Adaptado de Nunes *et al.* (2014).

Para isso, os fluxos possuem regras, contadores e ações. As regras são constituídas por um ou mais campos do cabeçalho do pacote, como o seu endereço de IP de origem e destino. Os contadores são utilizados para coletar estatísticas de utilização do fluxo, como o número de pacotes recebidos e duração do fluxo, que podem servir para a remoção de fluxos inativos da tabela. Por fim, as ações definem como os pacotes que correspondem a regra serão tratados, podendo encaminhar, modificar ou descartar os pacotes (ROTHENBERG *et al.*, 2011). A ação de descartar os pacotes pode ser utilizada para mitigar ataques de rede (MCKEOWN, 2008). Utilizando o protocolo OpenFlow, o controlador pode adicionar, atualizar ou deletar as entradas das tabelas de fluxos do *switch* (NUNES *et al.*, 2014).

2.1.2 Controlador Ryu

O Ryu é um controlador desenvolvido para redes definidas por software de código aberto, implementado na linguagem de programação Python, que suporta o protocolo

OpenFlow para o gerenciamento dos dispositivos de rede. Ele fornece os componentes para o desenvolvimento de aplicações de rede através de sua API (RYU, 2020).

O Ryu foi o controlador escolhido para a realização dos testes neste trabalho, visto que possui o suporte ao protocolo OpenFlow, que é utilizado pelo simulador de redes Mininet, e, por possuir um módulo de integração com o sistema de detecção de intrusão baseado em rede Snort.

2.1.3 Mininet

O Mininet é um emulador de rede de código aberto que habilita a simulação de uma rede com *hosts*, *switches* e controladores e cria *links* virtuais entre esses elementos. Ele habilita a experimentação com um sistema de rede definida por software utilizando o protocolo OpenFlow. O Mininet é utilizado para pesquisa, ensino e desenvolvimento (MININET, 2020).

Essa ferramenta possibilita a criação de uma rede virtual que replica uma rede com hardware real com alta precisão, onde os *hosts* virtualizados se comportam como máquinas reais, podendo executar programas que estão instalados no sistema rodando o Mininet (MININET, 2021).

Uma das principais propriedades do Mininet é o uso de *switches* OpenFlow virtualizados que fornecem a mesma semântica dos *switches* OpenFlow baseados em hardware. Desse modo, controladores ou aplicativos desenvolvidos e testados no ambiente emulado podem, em teoria, ser implementados sem modificações em ambientes de rede que utilizam o protocolo OpenFlow (KREUTZ *et al.*, 2015).

Com o suporte do protocolo OpenFlow que o Mininet fornece, é possível a programação dos *switches* virtuais a agirem de diversas maneiras sob os pacotes comutados. A ferramenta também proporciona a programação de topologias personalizadas com código na linguagem Python através de sua API (MININET, 2021).

2.2 SISTEMAS DE DETECÇÃO DE INTRUSÃO BASEADOS EM REDE

Segundo Scarfone e Mell (2007), Sistema de Detecção de Intrusão, do inglês *Intrusion Detection System* (IDS), é o programa de *software* que automatiza o processo de detecção de intrusão. Um Sistema de Detecção de Intrusão Baseado em Rede, do inglês *Network-based Intrusion Detection System* (NIDS), monitora o tráfego de rede em tempo real ou próximo ao tempo real, em determinados pontos de uma rede através de sensores, e, analisando o cabeçalho

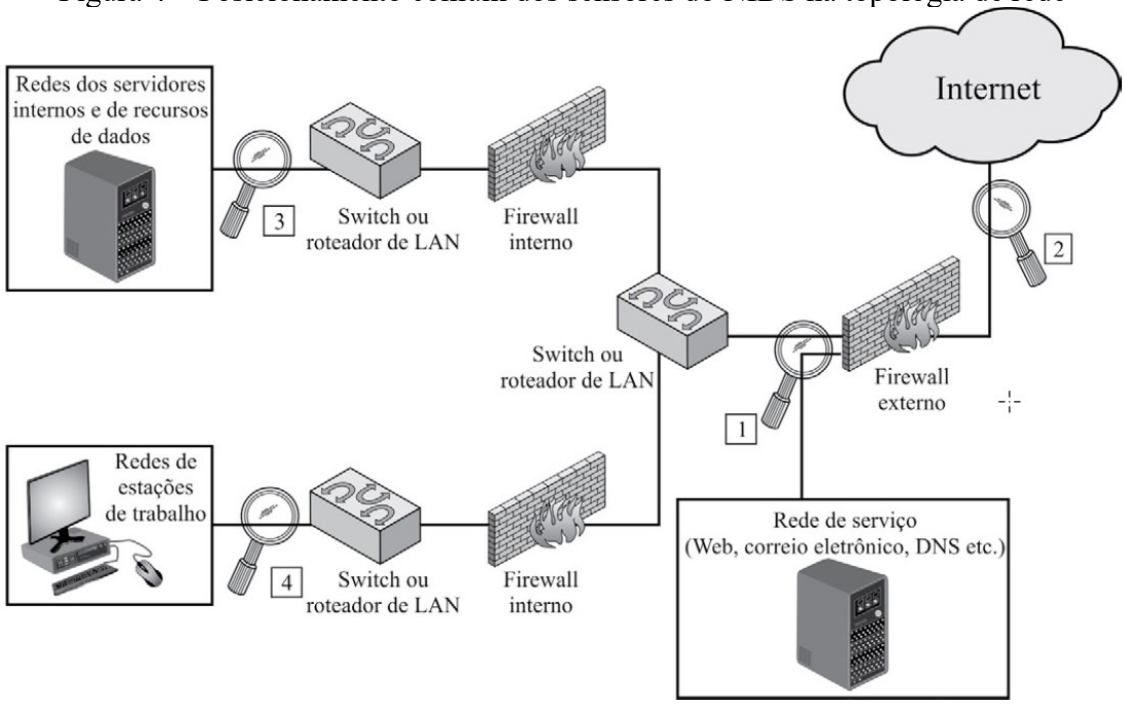
ou o conteúdo dos pacotes, é capaz de detectar anomalias de tráfego baseado em padrões de intrusão configurados (LIAO *et al.*, 2013).

Os sistemas de detecção de intrusão baseados em rede são componentes vitais da segurança da infraestrutura de rede atual (SALAH; KATANI, 2010). Ao detectar um potencial ataque, o NIDS envia um alerta e registra informações relacionadas ao evento. As informações registradas comumente são: carimbo de tempo (data e hora), tipo de evento, classificação (prioridade, gravidade), protocolos de camada de rede, transporte e aplicação utilizados, endereços IP de origem e destino e portas de origem e destino. O administrador de redes pode usar essas informações coletadas para fortalecer as medidas de detecção de intrusão (STALLINGS, 2014).

Segundo Pathan (2014), alguns dos benefícios da utilização de um NIDS como parte da solução de segurança de redes são: não haver impacto na performance da rede, a habilidade de detectar ataques que o *firewall* pode não proteger, a possibilidade de identificar ataques em tempo real, habilitando o administrador de redes a responder rapidamente e, a vantagem de ser invisível a atacantes, não deixando traços das suas atividades de monitoramento. Algumas das desvantagens são a incapacidade de inspecionar tráfego criptografado, a possibilidade da geração de falsos positivos e falsos negativos e a necessidade de alto poder computacional para análise de tráfego em segmentos de rede com grande volume de tráfego.

Para uma solução de segurança abrangente, um ou mais sensores do NIDS têm de ser instalados na infraestrutura. Na topologia de rede, o administrador terá de escolher os pontos em que os sensores serão instalados para sua monitoração (STALLINGS, 2014). Alguns dos segmentos de rede comumente monitorados pelos NIDS são apresentados na Figura 4, simbolizados pelas lupas.

Figura 4 – Posicionamento comum dos sensores do NIDS na topologia de rede



Fonte: Stallings (2014).

Deste modo, um NIDS é considerado um importante componente dentro de uma solução de defesa da infraestrutura de rede de uma organização, visto que possui características próprias as quais podem ser essenciais para uma segurança robusta, auxiliando outros componentes do arsenal de defesa como o *firewall* (NAKAMURA, 2007).

2.2.1 Snort

Snort é um sistema de detecção de intrusão baseado em rede de código aberto capaz de analisar tráfego de rede em tempo real, tendo como base um conjunto de regras de detecção para indicar o tráfego potencialmente malicioso na rede através da geração de alertas. Ele também é capaz de desempenhar o papel de um sistema de prevenção de intrusão, podendo rejeitar os pacotes originados de um ataque identificado (SNORT, 2021). Entretanto, este modo de operação pode potencialmente afetar usuários legítimos, negando a eles acesso à rede, na ocasião em que o sistema detecta um ataque equivocadamente (GUIMARAES; MURRAY, 2008).

Segundo Stallings (2014), a arquitetura do Snort é composta por quatro componentes lógicos:

- **Decodificador de pacotes:** é responsável por processar os pacotes capturados para identificar, isolar e extrair os cabeçalhos de protocolo contidos nos pacotes.

- **Motor de detecção:** é encarregado de analisar os pacotes baseando-se nas regras de detecção configuradas, e, quando um pacote corresponde a uma das regras, é executada a ação especificada pela regra.
- **Registrador:** lida com o armazenamento dos pacotes detectados pelas regras que especificam que o pacote há de ser registrado.
- **Alertador:** envia alertas sobre pacotes detectados pelas regras que determinam que um alerta tem de ser enviado. A notificação pode ser enviada a um arquivo, um banco de dados ou, a um *socket* UNIX, onde o alerta é encaminhado a uma outra máquina ou aplicação situada na rede.

O administrador de rede tem a opção de configurar o Snort para operar em um dos três diferentes modos disponíveis: o modo *Sniffer*, onde faz a leitura dos pacotes capturados e os exibe na tela, o modo *Packet Logger*, em que apenas faz o registro dos pacotes capturados em um arquivo, e o modo NIDS, onde é capaz de efetuar a análise de tráfego de rede e detecção de intrusões (SNORT, 2020).

2.2.1.1 Regras

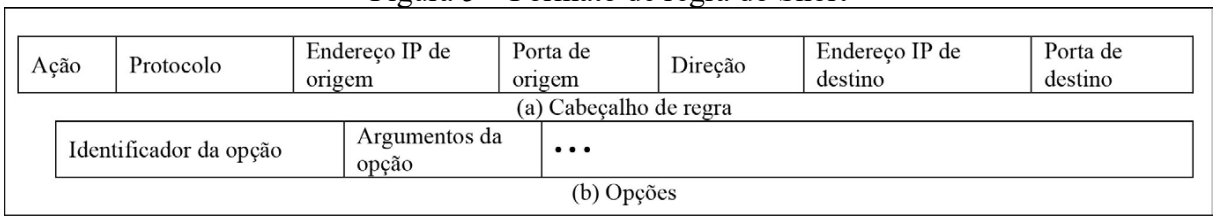
A ferramenta por padrão vem instalada contendo centenas de regras baseadas em ataques previamente observados e uma lista adicional de regras é disponibilizada no site do Snort (SNORT, 2021). Essa lista adicional é atualizada constantemente e cabe ao administrador de rede atualizar sua lista de regras ao mesmo passo em que novos métodos de ataque são descobertos. Adicionalmente, as regras em utilização precisam ser periodicamente otimizadas para diminuir alertas falsos (COX; GREG, 2004).

Além das regras incluídas na instalação padrão do Snort, a ferramenta fornece ao administrador de rede a habilidade de criar suas próprias regras ou de modificar regras já existentes, podendo adaptá-las para o seu próprio ambiente de rede.

Uma das desvantagens dos NIDS é sua propensão de emitir falsos positivos e falsos negativos. Falsos positivos são alertas apontando que um ataque ocorreu ou está ocorrendo, quando na realidade, nada de anormal está acontecendo. De maneira oposta, falsos negativos são ocorrências em que um ataque de fato ocorreu ou está ocorrendo, mas a ferramenta não os detecta, consequentemente, não emitindo nenhum alerta. Regras precisam ser corretamente configuradas para diminuir o número de falsos positivos e falsos negativos (COX; GREG, 2004).

O Snort possui uma sintaxe simples e flexível para a definição das regras utilizadas pelo motor de detecção. Apesar da simplicidade para a escrita das regras, elas possuem a capacidade de detectar uma grande variedade de tráfego potencialmente hostil (STALLINGS, 2014). Os elementos contidos em uma regra do Snort são apresentados na Figura 5.

Figura 5 – Formato de regra do Snort



Fonte: Stallings (2014).

Toda regra é composta por um cabeçalho fixo e zero ou mais opções. O cabeçalho define quem precisa estar envolvido na conexão para então as opções serem consideradas. As opções definem o que precisa estar contido no pacote, isso inclui informações que estão no cabeçalho do pacote, como os *bits* das *flags* TCP ativos ou o conteúdo da carga útil do pacote (NORTHCUTT; NOVAK, 2002).

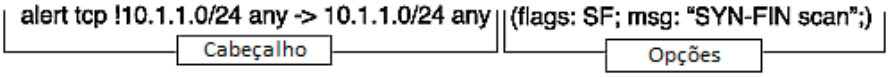
O cabeçalho constitui a primeira seção da regra, que descreve a conexão de rede do pacote inspecionado e qual ação irá tomar quando encontrar este pacote que está de acordo com os critérios da regra (STALLINGS, 2014). Quatro parâmetros definem uma conexão de rede específica: os endereços de IP de origem e destino e as portas de origem e destino. O cabeçalho também analisa o protocolo de rede utilizado na conexão e a direção que o pacote está trafegando, definido pelo símbolo -> para unidirecional ou <> para bidirecional. Os protocolos TCP, UDP, ICMP e IP são suportados na versão atual do Snort (SNORT, 2020).

Segundo Snort (2020), as ações que a ferramenta pode tomar diante um pacote detectado são:

- **alert:** gera um alerta utilizando o método de alerta escolhido e registra o pacote.
- **log:** registra o pacote.
- **pass:** ignora o pacote.
- **drop:** bloqueia e registra o pacote.
- **sdrop:** bloqueia o pacote sem registrá-lo.
- **reject:** bloqueia e registra o pacote e envia um pacote para o *host* de origem rejeitando a conexão.

As opções compõem a segunda parte da regra, onde cada opção é representada por um identificador de opção seguido pelos argumentos que especificam os detalhes da opção. Na forma escrita, os identificadores são separados de seus argumentos pelo caractere de dois-pontos (:) e separados entre si através do caractere de ponto e vírgula (;). As opções da regra ficam entre parênteses para se separar do cabeçalho. (STALLINGS, 2014). A sintaxe de uma regra do Snort é exemplificada na Figura 6.

Figura 6 – Exemplo de regra no Snort



Fonte: Adaptado de Northcutt e Novak (2002).

Apesar de não serem necessárias, as opções são utilizadas na maioria das regras, pois analisam mais detalhes sobre os atributos do pacote, o que pode ajudar na diminuição de alertas falsos (NORTHCUTT; NOVAK, 2002). Segundo Stallings (2014), existem quatro categorias principais de opções de regra:

- **Metadados:** não afetam no processo de detecção, apenas fornecendo informações sobre a regra, como a mensagem exibida nos alertas enviados.
- **Carga útil:** busca dados dentro da carga útil do pacote.
- **Não carga útil:** busca dados em outros campos do pacote além da sua carga útil, como o *bit* das *flags* TCP ativos.
- **Pós-detecção:** adicionadores específicos que ocorrem após a detecção.

Estas opções são verificadas somente se o cabeçalho da regra corresponder ao conteúdo do pacote. Se as opções também corresponderem com o conteúdo do pacote, o Snort irá agir de acordo com a ação determinada no cabeçalho da regra. Geralmente, a ferramenta grava uma mensagem de alerta e os dados do pacote no seu arquivo de registro, para que o administrador de rede possa revisar o pacote e analisar se houve uma tentativa de intrusão (COX; GREG, 2004).

2.3 ATAQUES CIBERNÉTICOS

Segundo Nieles, Dempsey e Pillitteri (2017), ataques cibernéticos são qualquer tipo de atividade maliciosa que tente coletar, perturbar, negar, degradar ou destruir informações ou os recursos de um sistema de informação.

Nas últimas décadas, devido ao crescente uso de redes e da Internet, sistemas de computadores apresentaram vários problemas de segurança, onde a quantidade de intrusões tem aumentado excessivamente ano a ano. Qualquer invasão ou ataque malicioso às vulnerabilidades das redes, computadores ou sistemas de informação podem originar graves desastres (LIAO, *et al.* 2013).

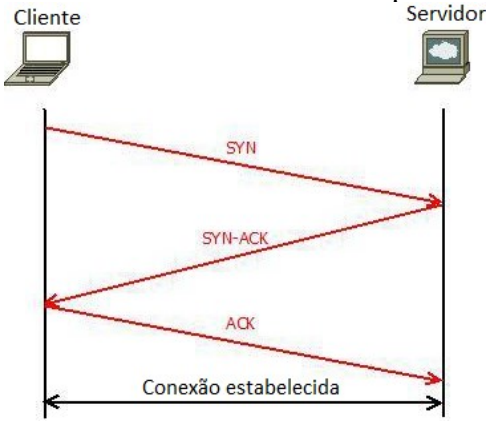
Nesta seção são apresentados os dois ataques cibernéticos selecionados para a execução dos testes neste trabalho, conjuntamente com o funcionamento de ambos e o mecanismo de estabelecimento e fechamento de conexões no protocolo TCP.

2.3.1 Estabelecimento e fechamento de conexões no protocolo TCP

Os dois ataques selecionados para a realização dos testes neste trabalho utilizam o Protocolo de Controle de Transmissão, do inglês *Transmission Control Protocol* (TCP), como seu protocolo de transporte. O processo de criação das regras para a detecção desses ataques se baseou na compreensão do funcionamento de como uma conexão é estabelecida e terminada pelo protocolo TCP.

O estabelecimento de uma conexão entre dois computadores pelo protocolo TCP é feito através do *handshake* de três vias (FALL; STEVENS, 2012). O funcionamento do *handshake* é demonstrado na Figura 7.

Figura 7 – *Handshake* de três vias no protocolo TCP



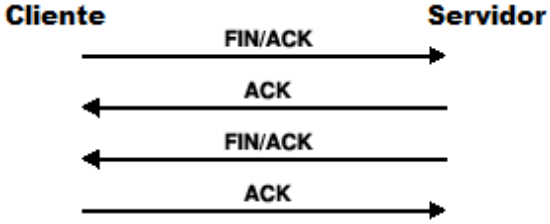
Fonte: Adaptado de Mašetić *et al.* (2017).

O *handshake* é iniciado por um cliente, que envia um pacote com o *bit* da *flag* de sincronização (SYN) ativado para um servidor. O servidor responde com um pacote com os *bit*s da *flag* SYN e a *flag* de reconhecimento (ACK) ativados, reconhecendo o recebimento do pacote SYN do cliente. Por fim, para estabelecer a conexão, o cliente responde com um pacote

com o *bit* da *flag* ACK ativado para o servidor, reconhecendo o recebimento do seu pacote SYN/ACK. Se essa troca de pacotes suceder, a conexão entre cliente e servidor é estabelecida (FALL; STEVENS, 2012).

Para o fechamento de uma conexão, um procedimento semelhante é realizado, porém com a utilização de outros tipos de pacotes. Este processo é apresentado na Figura 8.

Figura 8 – Fechamento de conexão no protocolo TCP



Fonte: Adaptado de IBM (2021).

A parte que pretende fechar a conexão – no exemplo da Figura 8, o cliente – inicia o processo enviando um pacote com o *bit* das *flags* de finalização (FIN) e ACK ativados. O servidor responde com um pacote ACK para reconhecer o recebimento do pacote FIN/ACK do cliente, e outro pacote FIN/ACK para fechar a conexão. O cliente então envia um pacote ACK, reconhecendo o recebimento do pacote FIN/ACK enviado pelo servidor. A conexão também pode ser terminada imediatamente se a parte que pretende fechá-la enviar um pacote com a *flag* de *reset* (RST) ativada (FALL; STEVENS, 2012).

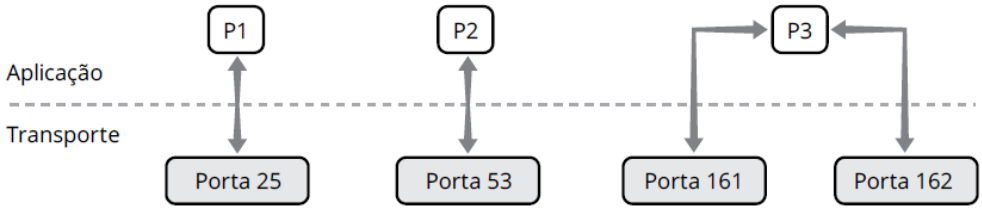
2.3.2 Ataque de varredura de portas TCP SYN

O reconhecimento e coleta de informações sobre uma rede e seus *hosts* é a primeira fase de um ataque em um sistema. Dados como endereços de IP, sistemas operacionais e serviços e aplicações em utilização podem ser essenciais para ajudar um invasor a decidir como irá atacar um sistema. A varredura é o processo de localizar estes sistemas ativos na rede (GRAVES, 2007).

A técnica de varredura de portas é um dos mais comuns ataques de rede. O objetivo deste ataque é de obter informações sobre a rede alvo, procurando potenciais vulnerabilidades que podem ser exploradas. Algumas das informações comumente coletadas neste tipo de ataque são os *hosts* ativos da rede, seus endereços de IP e quais portas estão abertas em cada *host* (GADGE; PATIL, 2008). Esta técnica pode também ser muito útil para um administrador de rede testar a vulnerabilidade de seu ambiente (MELO, 2017).

Computadores que estão conectados a uma rede geralmente executam diferentes aplicações que recebem conexões em portas específicas. Essas conexões são geralmente feitas através dos protocolos de transporte TCP e UDP. Estes protocolos utilizam o conceito de "portas" como entidades lógicas para comunicação na rede. As portas servem para fornecer um número identificador que diferencia as sessões de transporte entre um par de *host*s, assim como para identificar o protocolo da aplicação e o serviço associado a qual um processo se conecta. A combinação de números de portas de origem e destino, juntamente com os endereços de IP dos *host*s identifica exclusivamente uma sessão de um determinado protocolo de transporte (COTTON *et al.*, 2011). Adicionalmente, as portas são utilizadas para permitir que vários processos executando em um mesmo computador possam simultaneamente receber e transmitir dados de forma independente (ELIAS; LOBATO, 2013). Essa identificação dos processos feita pelas portas é ilustrada na Figura 9.

Figura 9 – Identificação de processos através de portas

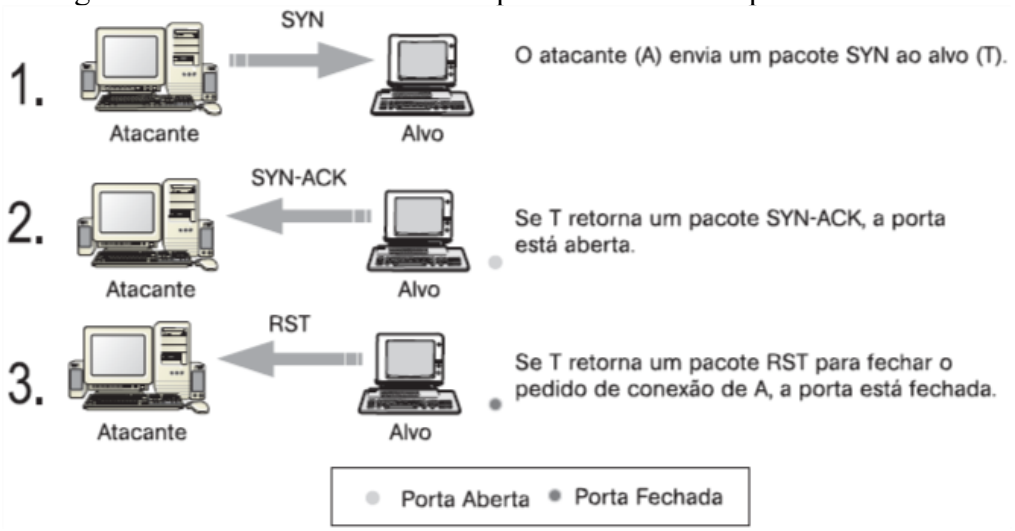


Fonte: Elias e Lobato (2013).

Quando identificadas as portas abertas de um servidor, as vulnerabilidades dos serviços disponíveis podem ser testadas através dessas portas, onde podem ser enviados comandos e realizados ataques (NAKAMURA, 2016).

A varredura de portas TCP SYN é um dos métodos mais populares de varredura por ser relativamente camuflado, pois o *host* atacante nunca abre uma conexão TCP completa com o alvo através do *handshake* de três vias. O atacante envia um pacote SYN ao alvo e, baseado na sua resposta, pode-se deduzir se a porta está aberta ou não (LYON, 2008). O funcionamento do ataque é demonstrado na Figura 10.

Figura 10 – Funcionamento do ataque de varredura de portas TCP SYN



Fonte: Nakamura e Geus (2007).

O atacante inicia enviando um pacote SYN para o alvo em uma determinada porta e espera uma resposta, o que é o primeiro passo no *handshake* de três vias que acontece em qualquer tentativa de conexão pelo protocolo TCP. Se a porta do alvo estiver aberta, ele responderá com um pacote SYN/ACK, assinalando que a porta está aberta para conexões e deseja estabelecer uma conexão com o atacante, mas, para não estabelecer essa conexão, rejeitando o *handshake*, o atacante responde com um pacote RST. Se a porta estiver fechada, o alvo responderá com um pacote RST, assinalando que a porta está fechada e rejeitando a conexão, logo, o atacante não precisará responder pois o alvo já fecha a conexão com a sua resposta (LYON, 2008).

Pelo fato deste tipo de varredura não estabelecer conexões completas, esta técnica é também conhecida como "varredura semiaberta". Este método é mais difícil de detectar, dado que tentativas de conexões através de pacotes SYN são registradas menos frequentemente que conexões estabelecidas (VIVO *et al.*, 1999).

A ferramenta utilizada neste trabalho para executar o ataque de varredura de portas TCP SYN foi o escaneador de redes Nmap. O Nmap, também conhecido como "*Network Mapper*" é uma ferramenta de código aberto para exploração de rede e auditoria de segurança. Ele é utilizado para rapidamente escanear redes e *host*s individuais. O Nmap utiliza pacotes IP para determinar quais *host*s estão disponíveis na rede, quais serviços os *host*s oferecem, quais sistemas operacionais estão executando, entre outras funcionalidades (LYON, 2008).

2.3.3 Ataque de dicionário no protocolo SSH

O método mais comum de autenticação de usuários é a utilização de senhas, devido sua conveniência e praticidade para ambos provedores de serviços e usuários finais. Porém, esses benefícios têm um custo, visto que senhas escolhidas por humanos são geralmente inseguras, pois uma grande parte dos usuários escolhem senhas simples, que são fáceis de serem descobertas, como seu próprio nome, o nome da rua onde moram, sua data de nascimento, uma palavra comum do dicionário, entre outros. Isso abre uma brecha para invasores tentar obter acesso a contas utilizando uma lista de senhas amplamente conhecidas, até encontrar a senha correta. Este ataque é chamado de ataque de dicionário (PINKAS; SANDER, 2002).

O ataque de dicionário é um ataque comum de descobrimento de senhas por ser simples de executar. Ele é utilizado para ganhar acesso à conta de usuários e administradores de um sistema. O programa utiliza arquivos contendo *logins* e senhas amplamente conhecidas e utilizadas, muitas vezes extraídas de listas de vazamento de dados, e tenta realizar o *login* em um sistema com todas as variações do seu dicionário de *logins* e senhas. Este tipo de ataque é capaz de ser muito perigoso, uma vez que um único acerto pode ser suficiente para obter uma amplitude de privilégios em um sistema (STALLINGS, 2014).

Segundo Ylonen *et al.* (2006), o protocolo SSH, do inglês *Secure Shell Protocol*, é um protocolo para *login* remoto e oferecimento de outros serviços seguros de rede sobre uma rede insegura. Ele é tipicamente utilizado para fazer conexões remotas a outros computadores, com o intuito de gerenciá-los através de uma linha de comando. Ele foi criado para substituir o protocolo Telnet e outros protocolos afins que não oferecem segurança e privacidade, visto que esses protocolos enviam todos os dados – principalmente *logins* e senhas – em texto simples, onde alguém malicioso pode capturar e analisar os pacotes e ler todas as informações enviadas durante a sessão. O SSH utiliza criptografia para prover confidencialidade e integridade aos dados que trafegam durante as sessões. Ele utiliza uma arquitetura cliente-servidor, onde um servidor SSH fornece conexões a clientes SSH (BARRETT; SILVERMAN; BYRNES, 2005).

Este protocolo é bastante popular, sendo executado em muitos computadores no mundo todo, o que torna o ataque de dicionário contra sistemas utilizando o SSH uma ameaça de segurança comum (SATOH *et al.*, 2015).

No protocolo SSH, esse tipo de ataque pode ser reproduzido em sistemas pouco seguros e mal configurados, onde são utilizados métodos de autenticação inseguros. Owens e Matthews (2008) apontam que, no SSH, a utilização de senhas para *login* pode ser um problema de segurança, portanto recomenda-se que esse tipo de autenticação seja trocado pela

autenticação de chave pública para a realização de *login* dos usuários no servidor SSH, o que elimina a ameaça de ataques de dicionário a este protocolo.

O protocolo SSH utiliza o TCP como protocolo de transporte de dados, portanto, ele segue as propriedades do TCP para o estabelecimento e fechamento de conexões.

O ataque foi executado utilizando a ferramenta THC Hydra, um quebrador de senhas de código aberto que pode realizar ataques de dicionário em diversos protocolos, inclusive o SSH. A ferramenta inclusive pode realizar ataques utilizando várias tarefas executando em paralelo (KAKARLA; MAIRAJ; JAVAID, 2018).

3 DESENVOLVIMENTO

Este capítulo descreve os procedimentos realizados para o alcance dos resultados obtidos neste trabalho. A seção 3.1 apresenta o ambiente de testes utilizado para o desenvolvimento do trabalho. A seção 3.2 relata a integração feita entre o NIDS Snort, o controlador Ryu e o *switch* OpenFlow, integral para o funcionamento da solução de mitigação. Por fim, a seção 3.3 trata da criação das regras no Snort para a detecção dos ataques de rede executados.

3.1 AMBIENTE DE TESTES

O ambiente de experimentação foi composto por um *notebook* com as seguintes especificações: processador Intel i7 8750H, 8GB de memória RAM e um SSD de 500GB. O sistema operacional utilizado foi o Linux Manjaro e o software de virtualização utilizado foi o Oracle VM VirtualBox (em sua versão 6.1) para a criação e utilização da máquina virtual.

A máquina virtual, do inglês *Virtual Machine* (VM), foi criada com a seguinte configuração: 2 CPUs virtuais, 3GB de memória RAM e 15GB de disco virtual. O sistema operacional utilizado foi o Linux Xubuntu na sua versão 20.04.2 LTS. Essa distribuição do Linux foi escolhida por ser uma versão estável e leve, o que auxilia na rapidez e conservação de recursos computacionais requeridos pela VM.

Na máquina virtual, foram executados todos os programas necessários para a realização dos testes neste trabalho. Os programas utilizados e suas respectivas versões são exibidos no Quadro 1.

Quadro 1 – Programas utilizados no ambiente de testes

Tipo de programa	**Nome**	**Versão**
Emulador de rede	Mininet	2.3
Controlador	Ryu	4.34
Sistema de detecção de intrusão baseado em rede	Snort	2.9.7
Analisador de tráfego	Wireshark	3.2.3
Cliente API	Postman	8.1
Escaneador de rede	Nmap	7.8
Quebrador de senhas	THC Hydra	9

Fonte: Elaborado pelo autor.

O ambiente de rede simulado através do Mininet, utilizado para os testes realizados neste trabalho, constituiu-se de uma rede com o controlador Ryu controlando o *switch*

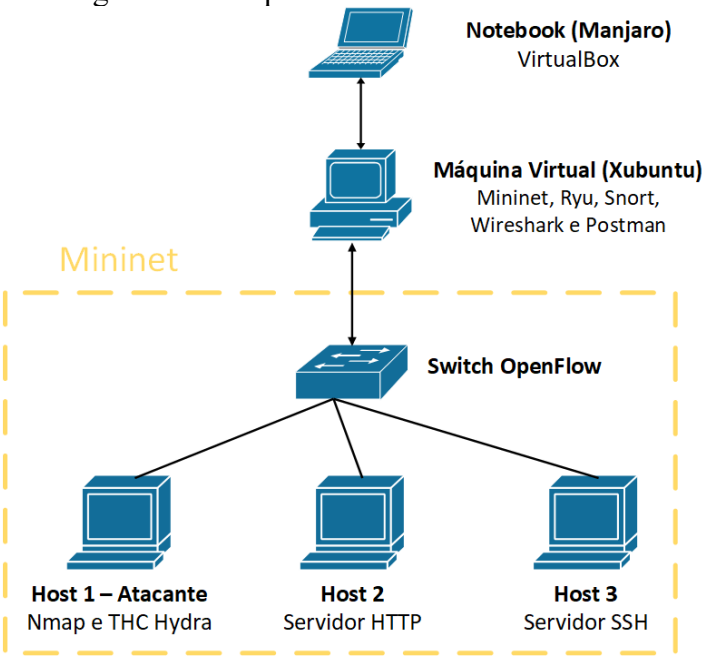
OpenFlow, que conecta três *hosts*. O "*Host* 1" será o atacante e, os demais *host*s, "*Host* 2" e "*Host* 3" foram as vítimas dos ataques. A inicialização desta topologia no Mininet é exibida na Figura 11. A arquitetura do ambiente de testes é apresentada na Figura 12.

Figura 11 – Inicialização do ambiente de rede no Mininet

```
gd@sdn-vm:~$ sudo mn --controller remote --switch ovsk,protocols=OpenFlow13 --mac
--ipbase=10.1.1.0/24 --topo single,3
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Fonte: Elaborado pelo autor.

Figura 12 – Arquitetura do ambiente de testes



Fonte: Elaborado pelo autor.

No "*Host* 1" (*h1* no Mininet) foram executados os ataques de varredura de portas TCP SYN pela ferramenta Nmap e o ataque de dicionário no protocolo SSH pela ferramenta THC Hydra. No "*Host* 2" (*h2* no Mininet) foi aberto um servidor *web* através de um módulo do

Python para a criação de servidores HTTP, utilizando a porta 80 para aceitar conexões. No "*Host* 3" (*h3* no Mininet) foi aberto um servidor SSH, utilizando a porta 22 para aceitar conexões, através da execução do programa *sshd*, já instalado no Linux Manjaro.

3.2 INTEGRAÇÃO ENTRE SNORT, RYU E *SWITCH* OPENFLOW

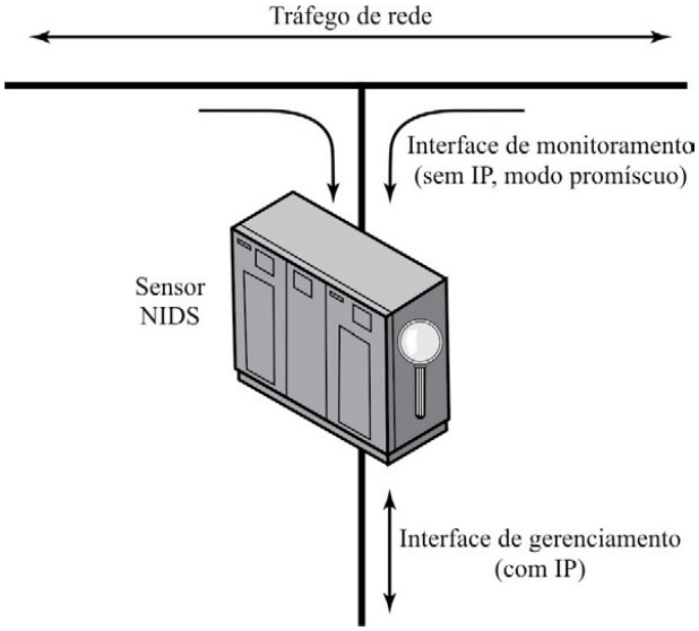
Essa seção apresenta os procedimentos realizados para integrar o NIDS Snort, o controlador Ryu e o *switch* OpenFlow, a fim de que as ferramentas trabalhem em conjunto para possibilitar a mitigação dos ataques.

3.2.1 Porta de espelhamento

Para o NIDS Snort receber o tráfego da rede para análise, é necessário ele estar conectado a um *switch*, que então irá encaminhar todos os pacotes que ele comuta para a interface do Snort. Essa interface normalmente é conectada em uma porta do *switch* que esteja funcionando em modo de espelhamento.

O espelhamento de porta é um método de fazer com que um *switch* envie uma cópia dos seus pacotes comutados para uma porta específica, que geralmente é conectada a um analisador de tráfego ou sistema de detecção de intrusão (CISCO, 2019). O efeito desejado com esse tipo de configuração é ilustrado na Figura 13.

Figura 13 – Sensor NIDS conectado à porta de espelhamento



Fonte: Stallings (2014).

No ambiente de testes, a porta de espelhamento foi criada no *switch* OpenFlow com o nome de "*snort0*" e o Snort foi executado passando um parâmetro para analisar os pacotes recebidos nesta porta. É importante notar que esta porta é configurada para rodar em modo promíscuo, pois nesse modo, todos os pacotes recebidos são enviados para o *kernel* para processamento, incluindo pacotes não destinados ao endereço MAC da placa de rede, o que se torna essencial para inspecionar todo o tráfego passando pela rede (CHUVAKIN *et al.*, 2013).

3.2.2 Envio de alertas do Snort ao controlador Ryu

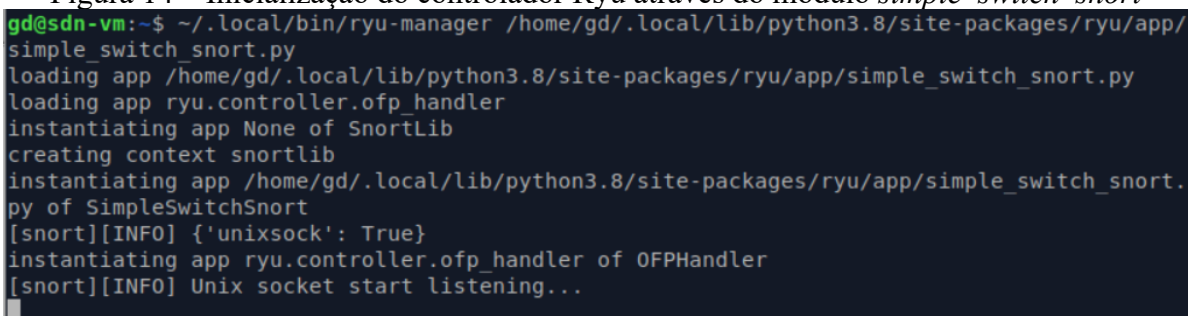
A integração entre o NIDS Snort e o controlador Ryu é feita através da execução de um módulo do Ryu para a integração com o Snort, chamado *simple_switch_snort*. Essa integração possibilita o Snort a receber os pacotes do *switch* OpenFlow e, quando encontrar anomalias na rede, enviar seus alertas para o Ryu, que por sua vez exibe na tela os alertas recebidos.

Com a utilização deste módulo, o envio de alertas do Snort para o Ryu pode ser feito de duas maneiras, dependendo se as duas ferramentas estão sendo executadas na mesma máquina ou em máquinas diferentes (RYU, 2021). Visto que neste ambiente foi utilizado apenas uma máquina virtual para a realização dos testes, a primeira maneira foi utilizada.

Nesta topologia, o Snort recebe os pacotes do *switch* através da porta de espelhamento, e, quando dispara um alarme, envia para o controlador Ryu através do *UNIX domain socket*, que é um mecanismo utilizado para habilitar a comunicação entre dois processos que estão executando na mesma máquina (IBM, 2021). Na inicialização do Snort, é passado a ele um parâmetro para utilização deste *socket* como o mecanismo de envio de alertas selecionado para o alertador.

No Ryu, o recebimento e exibição dos alarmes enviados pelo Snort é realizado através do seu módulo *snortlib*, que é executado junto com o módulo *simple_switch_snort*. O módulo *snortlib* recebe os alertas "escutando" o *UNIX domain socket* e, quando recebidos, os alertas são exibidos na tela. A execução do controlador Ryu, através do seu módulo para integração com o Snort é demonstrado na Figura 14.

Figura 14 – Inicialização do controlador Ryu através do módulo *simple_switch_snort*

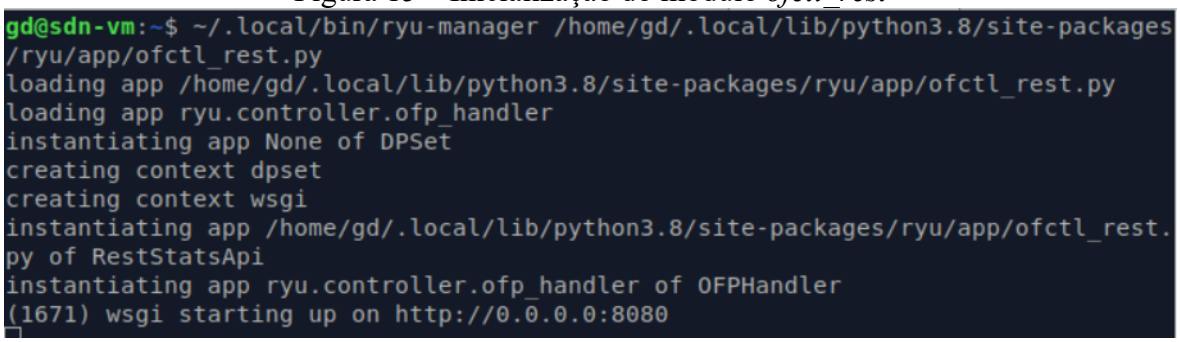


Fonte: Elaborado pelo autor.

3.2.3 Comunicação entre controlador Ryu e *switch* OpenFlow

Para o controlador Ryu poder alterar as entradas de fluxos nas tabelas de fluxos do *switch* OpenFlow, o módulo do Ryu chamado *ofctl_rest* foi utilizado. Essa aplicação provém APIs REST para a extração de informações e atualização de elementos do *switch* (RYU, 2021). REST, do inglês *Representational State Transfer*, é um tipo de API que habilita a comunicação entre aplicações através de mensagens HTTP (ODOM, 2020). A inicialização da aplicação *ofctl_rest* é demonstrada na Figura 15.

Figura 15 – Inicialização do módulo *ofctl_rest*



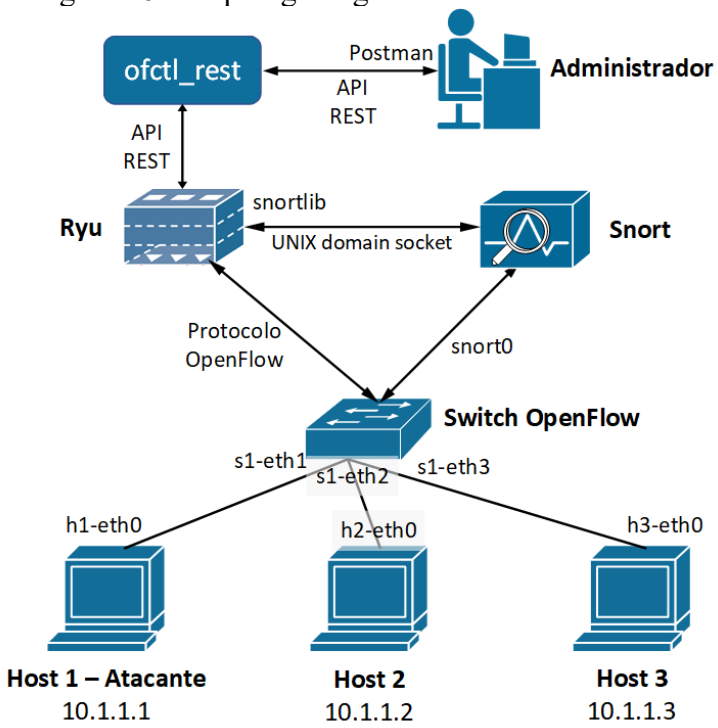
Fonte: Elaborado pelo autor.

Através das APIs REST, o administrador de rede pode enviar comandos ao controlador para adicionar, atualizar ou deletar as entradas de fluxos das tabelas de fluxos do *switch*. O controlador Ryu, por sua vez, envia as mensagens ao *switch* pelo protocolo OpenFlow por meio do canal seguro, numa conexão local estabelecida na porta 6653, como demonstrado na Figura 11.

Para o envio das mensagens HTTP ao *ofctl_rest* por meio da API REST, o programa Postman foi utilizado. O Postman é um cliente API que facilita a criação, teste e documentação de APIs, permitindo criar e enviar solicitações HTTP e ler suas respostas (CIELO, 2021).

Deste modo, a topologia lógica do ambiente de testes simulado para a execução deste trabalho é apresentada na Figura 16. Os *links* virtuais criados no Mininet entre os *hosts* e o *switch* OpenFlow são evidenciados na Figura 17.

Figura 16 – Topologia lógica do ambiente de testes



Fonte: Elaborado pelo autor.

Figura 17 – *Links* dos *hosts* com o *switch* OpenFlow

```
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
h3-eth0<->s1-eth3 (OK OK)
```

Fonte: Elaborado pelo autor.

3.3 ELABORAÇÃO DE REGRAS DE DETECÇÃO NO SNORT

Segundo Scarfone e Mell (2007), a detecção de intrusão é o processo de monitoramento dos eventos que ocorrem num sistema de computador ou rede e a realização de sua análise para encontrar sinais de violações ou ameaças iminentes de violação das políticas de segurança do sistema. Ela é baseada na premissa de que o comportamento do atacante difere do comportamento de um usuário normal de maneiras que podem ser quantificadas (STALLINGS, 2014).

O desenvolvimento das regras no NIDS Snort foi baseado na captura e posterior análise do tráfego feita no programa Wireshark durante a execução dos ataques no ambiente de rede simulado com o Mininet. O Wireshark é um analisador de tráfego de rede de código aberto que apresenta detalhadamente os pacotes capturados (WIRESHARK, 2021).

No Wireshark, a captura de pacotes foi feita na interface *snort0*, visto que essa interface está conectada a uma porta de espelhamento e está rodando em modo promíscuo, consequentemente capturando e processando todos os pacotes que passam pelo *switch*.

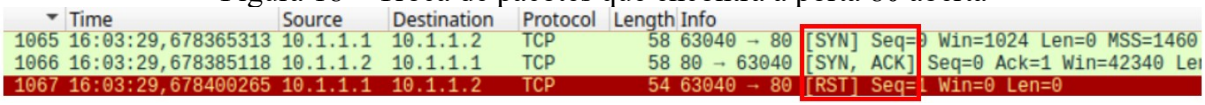
Segundo Stallings (2014), a abordagem mais efetiva para o desenvolvimento das regras de detecção em um sistema de detecção de intrusão baseado em rede é a realização da análise do tráfego gerado pelas ferramentas de ataque. Essa análise possibilita a detecção de padrões atípicos de tráfego gerado pelos ataques, onde, baseando-se neles, as regras para a detecção dos ataques podem ser desenvolvidas no NIDS pelo administrador de redes.

3.3.1 Regra de detecção do ataque de varredura de portas TCP SYN

Segundo Nakamura (2007), os NIDS são eficientes na detecção de ataques de varredura de portas. Este ataque, por padrão, é executado rapidamente, podendo escanear milhares de portas por segundo em uma rede de alta velocidade (LYON, 2008).

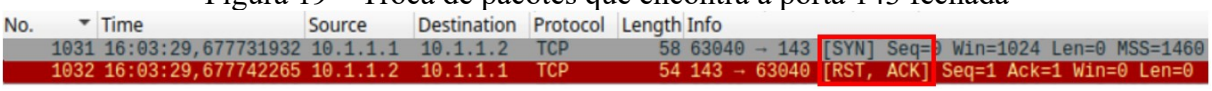
Como visto na seção 2.3.2, o funcionamento do ataque constitui-se no ínicio do *handshake* de três vias com diversas portas do *host* vítima, e, baseado na sua resposta, a ferramenta consegue detectar se uma determinada porta está aberta ou não. Esse comportamento é evidenciado nas Figuras 18 e 19, onde o atacante é o *host* com o endereço IP 10.1.1.1, que começa enviando o pacote SYN ao *host* vítima, de endereço IP 10.1.1.2.

Figura 18 – Troca de pacotes que encontra a porta 80 aberta



Fonte: Elaborado pelo autor.

Figura 19 – Troca de pacotes que encontra a porta 143 fechada

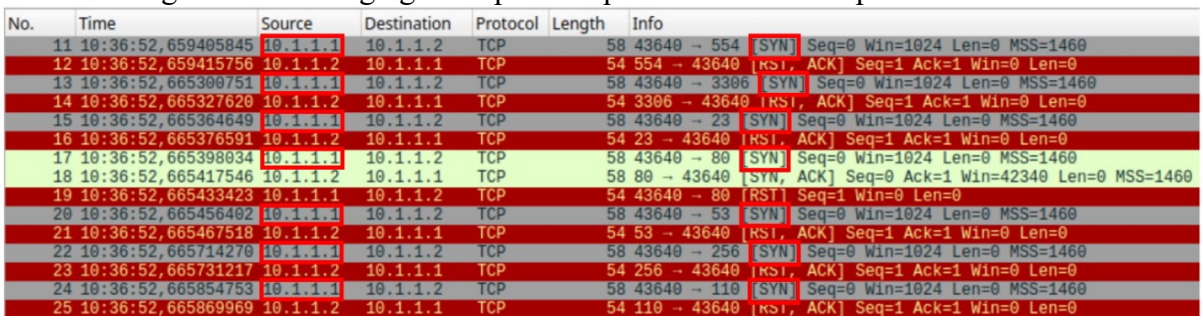


Fonte: Elaborado pelo autor.

Em cada conexão pelo protocolo TCP, o cliente e o servidor fazem o *handshake* de três vias para estabelecer a conexão. No ataque, o *handshake* não é completo pelo atacante,

apenas enviando os pacotes SYN ou RST. Porém, o atacante sempre enviará pacotes SYN, pois é o pacote que inicia o *handshake* de três vias para o estabelecimento da conexão. Ele só enviará o pacote RST se receber um pacote SYN/ACK de resposta da vítima, apontando que a porta está aberta. Isso indica que o pacote constantemente enviado pelo ataque é o pacote SYN. Um exemplo de parte do tráfego gerado pelo ataque de varredura de portas TCP SYN é mostrado na Figura 20.

Figura 20 – Tráfego gerado pelo ataque de varredura de portas TCP SYN



Fonte: Elaborado pelo autor.

Normalmente, esse ínicio de *handshake* não é realizado centenas de vezes por segundo entre apenas um cliente e o servidor, como acontece durante o ataque, evidenciado na Figura 20, onde, no mesmo segundo foram feitas diversas tentativas de estabelecimento de conexão pelo mesmo *host* (10.1.1.1) em diversas portas. Essa anormalidade de tráfego sendo enviado pelo *host* atacante foi a base para a criação da regra de detecção do ataque de varredura de portas TCP SYN. No Snort, a seguinte regra foi estabelecida para a detecção deste ataque:

alert tcp any any -> $HOME_NET any (msg:"* Possível Ataque Varredura TCP SYN ***"; flow:to_server, not_established; detection_filter:track by_src, count 50, seconds 10; flags:S; classtype:attempted-recon; sid:10000001; rev:5;)**

A regra estabelece que o Snort deverá enviar um alerta caso qualquer *host*, em um intervalo de 10 segundos, envie cinquenta pacotes SYN utilizando o protocolo TCP, de qualquer porta de origem para qualquer porta de destino, para os *hosts* que compõe a rede interna, configurado pela variável "*$HOME_NET*" no arquivo de configuração do Snort. Neste trabalho, esta variável foi configurada como 10.1.1.0/24, pois foi a subrede utilizada na rede simulada no Mininet. Para melhorar a filtragem, a regra apenas analisa pacotes enviados em conexões ainda não estabelecidas. É atribuído ao ataque uma classificação de *attempted-recon*, que significa uma tentativa de coleta de informações sobre o ambiente de rede, o que agrega no detalhamento das informações contidas nos alertas. Por fim, ela obrigatoriamente possui um

número único de identificação e um número opcional de revisões, neste caso, a regra foi revisada cinco vezes, fazendo essa a quinta versão da regra.

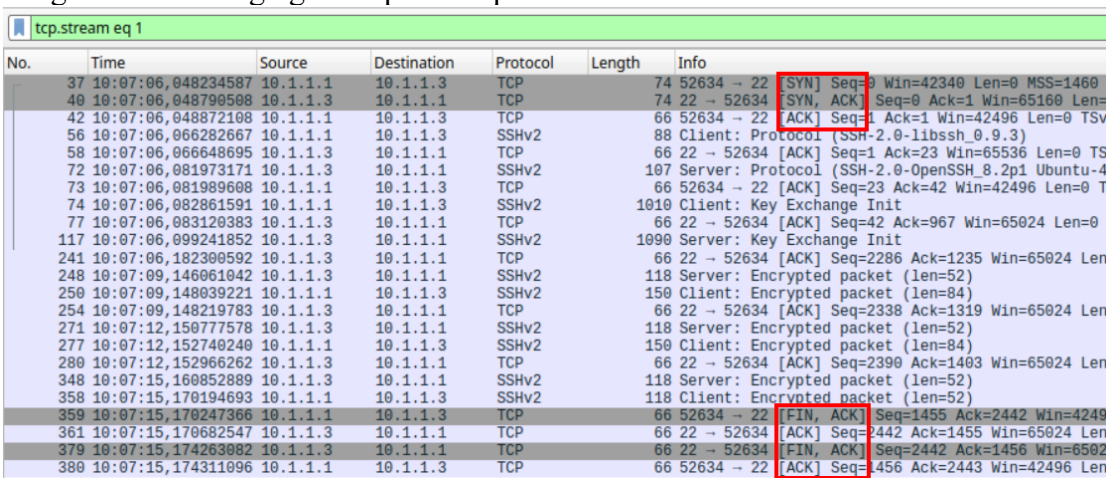
Nos testes efetuados, o ataque disparou em apenas um segundo todos os pacotes SYN ao servidor. São enviados exatamente 1000 pacotes SYN ao servidor, pois o ataque, por padrão, faz a varredura das 1000 portas mais populares utilizadas por servidores. Isso resulta em uma média de 1000 pacotes por segundo, o que justifica a taxa de pacotes na regra, que foi de 10 pacotes por segundo, fazendo com que seja uma regra conservadora, visto que a taxa de pacotes enviados pelo ataque é muito maior. Isso ajuda a detectar não só este ataque de varredura que é rápido, como ataques propositalmente lentos, com o intuito de não serem detectados por regras menos conservadoras.

É relevante notar que o administrador de rede, em caso da realização de uma análise de vulnerabilidades no seu próprio ambiente de rede, pode configurar a regra para ignorar o endereço de IP de origem do computador que realizará a varredura de portas.

3.3.2 Regra de detecção do ataque de dicionário no protocolo SSH

Visto que o ataque de dicionário também utiliza o protocolo TCP como protocolo de transporte, o estabelecimento da conexão entre cliente e servidor é feito através do *handshake* de três vias. Com a conexão TCP estabelecida, é feita a troca de pacotes entre ambas as partes para a abertura de sessão e troca de informações no protocolo SSH. Quando o cliente deseja fechar a sessão, ele realiza o fechamento da conexão através do protocolo TCP. Essa sequência de pacotes é demonstrada na Figura 21, onde o *host* atacante com o endereço de IP 10.1.1.1 realiza uma tentativa de conexão ao servidor SSH de endereço IP 10.1.1.3.

Figura 21 – Tráfego gerado pelo ataque de dicionário em uma tentativa de conexão



Fonte: Elaborado pelo autor.

Esse tráfego é gerado para cada tentativa de conexão que o ataque realiza, onde, nos testes realizados, foi observado que a ferramenta utiliza duas ou três combinações de *login* e senha para tentar obter acesso ao *host* vítima por tentativa de conexão. Como os pacotes enviados no protocolo SSH são criptografados, foi inviável obter um padrão de anomalias na carga útil dos pacotes enviados por meio do SSH. Porém, como o ataque de varredura de portas, as anomalias foram encontradas durante o processo de estabelecimento e fechamento de conexões pelo protocolo TCP. No caso deste ataque de dicionário, o padrão foi encontrado no fechamento das conexões.

Para realizar o ataque, foram utilizados dois arquivos de texto com a ferramenta THC Hydra, um com *logins* bem conhecidos e outro com senhas geralmente utilizadas. A ferramenta então faz uma tentativa de *login* no *host* vítima com todas as combinações de *logins* e senhas contidas nos arquivos de texto. A lista de *logins* foi formada por 18 itens e incluiu-se 23 itens na lista de senhas. Deste modo, o ataque realizou ao total 414 tentativas de *login*.

Com a análise do tráfego de rede gerado pelo ataque de dicionário no protocolo SSH, o padrão de anomalias foi encontrado no fechamento da conexão TCP de cada tentativa de *login* que o THC Hydra realizou. Visto que foram realizadas várias tentativas de *login* por segundo, a ferramenta gerou um número anormal de fechamento de conexões TCP originando de apenas um *host* num curto período de tempo. Um exemplo do tráfego gerado pelo ataque, filtrado por pacotes FIN/ACK originando do servidor para aceitar o término das conexões TCP vindos do *host* atacante é demonstrado na Figura 22.

Figura 22 – Tráfego gerado pelo ataque de dicionário filtrado por fechamento de conexões

(tcp.flags.fin == 1) and (tcp.flags.ack == 1) and (ip.src == 10.1.1.3) and (ip.dst == 10.1.1.1)

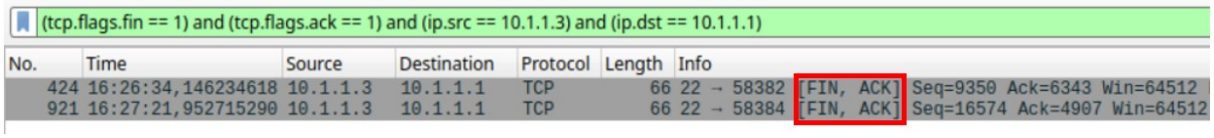
No.	Time	Source	Destination	Protocol	Length	Info
27	11:29:00,442008979	10.1.1.3	10.1.1.1	TCP	66	22 → 45696 [FIN, ACK] Seq=2286 Ack=1204 Win=65024
295	11:29:05,306225561	10.1.1.3	10.1.1.1	TCP	66	22 → 45712 [FIN, ACK] Seq=2390 Ack=1372 Win=65024
336	11:29:08,282423566	10.1.1.3	10.1.1.1	TCP	66	22 → 45704 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
338	11:29:08,283598391	10.1.1.3	10.1.1.1	TCP	66	22 → 45702 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
383	11:29:08,294228880	10.1.1.3	10.1.1.1	TCP	66	22 → 45698 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
385	11:29:08,294331772	10.1.1.3	10.1.1.1	TCP	66	22 → 45706 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
397	11:29:08,295116141	10.1.1.3	10.1.1.1	TCP	66	22 → 45710 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
440	11:29:08,319825670	10.1.1.3	10.1.1.1	TCP	66	22 → 45708 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
500	11:29:08,336521218	10.1.1.3	10.1.1.1	TCP	66	22 → 45700 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
602	11:29:13,838829899	10.1.1.3	10.1.1.1	TCP	66	22 → 45726 [FIN, ACK] Seq=2390 Ack=1372 Win=65024
637	11:29:16,265886506	10.1.1.3	10.1.1.1	TCP	66	22 → 45714 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
667	11:29:16,545314364	10.1.1.3	10.1.1.1	TCP	66	22 → 45716 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
679	11:29:16,553235408	10.1.1.3	10.1.1.1	TCP	66	22 → 45720 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
684	11:29:16,556209236	10.1.1.3	10.1.1.1	TCP	66	22 → 45724 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
708	11:29:16,580635119	10.1.1.3	10.1.1.1	TCP	66	22 → 45728 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
734	11:29:16,591062478	10.1.1.3	10.1.1.1	TCP	66	22 → 45722 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
737	11:29:16,595894636	10.1.1.3	10.1.1.1	TCP	66	22 → 45718 [FIN, ACK] Seq=2442 Ack=1456 Win=65024
901	11:29:22,098101266	10.1.1.3	10.1.1.1	TCP	66	22 → 45736 [FIN, ACK] Seq=2390 Ack=1372 Win=65024
935	11:29:23,841242361	10.1.1.3	10.1.1.1	TCP	66	22 → 45738 [FIN, ACK] Seq=2442 Ack=1456 Win=65024

Fonte: Elaborado pelo autor.

No mesmo minuto foram feitos diversos fechamentos de conexões TCP na porta 22, originando do mesmo *host* (10.1.1.1). Um usuário normal também estabelece e fecha a conexão

TCP a cada sessão SSH que realiza, a diferença se encontra na frequência desses eventos. Um exemplo do tráfego gerado pela utilização do SSH por um cliente normal, filtrado por fechamento de conexões, é exibido na Figura 23.

Figura 23 – Tráfego gerado em uso normal do SSH filtrado por fechamento de conexões



Fonte: Elaborado pelo autor.

Analisando o tráfego durante uma utilização normal do SSH, nota-se que apenas duas sessões, e, portanto, conexões foram fechadas originando do mesmo *host*, cada uma com um minuto de diferença, ao contrário do tráfego analisado durante o ataque, onde várias conexões foram fechadas no mesmo minuto originadas pelo mesmo *host*.

Baseado nesta anomalia de tráfego encontrada, a regra estabelecida no NIDS Snort para a detecção do ataque de dicionário no protocolo SSH realizada através da ferramenta THC Hydra foi a seguinte:

alert tcp $SSH_SERVERS 22 -> any any (msg:"* Possível Ataque Dicionario SSH ***"; flow:to_client; detection_filter:track by_dst, count 40, seconds 60; flags:FA; sid:10000002; rev:4;)**

A regra estabelece que o Snort deverá enviar um alerta caso qualquer servidor SSH, representado pela variável "*$SSH_SERVERS*", em um intervalo de um minuto, envie quarenta pacotes FIN/ACK utilizando o protocolo TCP originando da porta 22 para qualquer *host* em qualquer porta de destino. Ela obrigatoriamente possui um número único de identificação e um número opcional de revisões, neste caso, a regra foi revisada quatro vezes, fazendo esta a quarta versão da regra.

Assim como a variável "*$HOME_NET*", a variável "*$SSH_SERVERS*" pode ser configurada no arquivo de configuração do Snort. Nesse ambiente, essa variável foi definida como 10.1.1.3, que é o endereço de IP do servidor SSH na rede simulada no Mininet.

Comparada a regra criada para detecção do ataque de varredura de portas, esta regra detecta anomalias no caminho contrário dos pacotes, originando do *host* vítima para o *host* atacante.

Durante os testes efetuados, o ataque, por padrão, durou em média 142 segundos, com o servidor SSH disparando, em média, 141 pacotes FIN/ACK ao *host* atacante, o que resulta

em uma média aproximada de um pacote por segundo, o que justifica a taxa de pacotes imposta na regra.

3.3.3 Filtragem de eventos

Dado seus parâmetros, potencialmente muitos alertas seriam gerados pelas regras criadas para a detecção dos ataques. Portanto, uma filtragem de eventos teve de ser utilizada em conjunto com o filtro de detecção de taxa de pacotes imposto nas regras, com o objetivo de reduzir o número de alertas enviados pelo Snort.

A filtragem de eventos do Snort pode ser utilizada para reduzir o número de alertas disparados por regras "barulhentas", limitando o número de vezes que um alerta é enviado durante um período de tempo especificado (SNORT, 2020).

Os filtros de eventos foram configurados no arquivo de configuração nomeado *threshold.conf*, contido na pasta de configurações do Snort. A configuração aplicada no ambiente de testes é apresentada na Figura 24.

Figura 24 – Filtragem de eventos aplicada no arquivo de configuração *threshold.conf*

```
# Filtragem de Eventos
# ====================
#
# Varredura de Portas TCP SYN Nmap
event_filter gen_id 1, sig_id 10000001, type both, track by_src, count 50, seconds 10
#
# Ataque de Dicionário SSH THC Hydra
event_filter gen_id 1, sig_id 10000002, type both, track by_dst, count 40, seconds 60
```

Fonte: Elaborado pelo autor.

A sintaxe utilizada na filtragem de eventos é similar à da especificação de opções para as regras do Snort. Para cada filtro, foi dado o seu número de identificação, o número de identificação único da regra que está filtrando, o tipo de filtragem e o filtro de detecção de taxa de pacotes correspondente a regra.

O tipo de filtragem de evento *both* foi utilizado para ambas as regras. Este tipo de filtragem é aplicado para enviar somente um alerta durante o intervalo de tempo estipulado pela regra, ignorando alertas adicionais que seriam enviados durante o intervalo (SNORT, 2020).

O impacto da utilização da filtragem de eventos no número de alertas disparados pelo Snort é demonstrado na Figura 25, onde, na imagem do lado esquerdo, a filtragem está em utilização, disparando apenas um alarme no ataque detectado, e, ao lado direito, a filtragem foi desativada, disparando vários alarmes durante o mesmo ataque.

Figura 25 – Impacto da utilização da filtragem de eventos

Fonte: Elaborado pelo autor.

4 RESULTADOS

Este capítulo apresenta os resultados obtidos através dos testes conduzidos no ambiente de rede SDN simulado. Na seção 4.1 é evidenciada a detecção dos ataques executados. Já na seção 4.2 é demonstrado o método de mitigação dos ataques detectados.

4.1 DETECÇÃO DOS ATAQUES

Após a configuração do ambiente de testes e a elaboração das regras de detecção no Snort, os ataques cibernéticos selecionados foram executados no ambiente para avaliar a eficácia das regras na detecção dos ataques.

A detecção dos ataques é feita através do NIDS Snort efetuando o processamento dos pacotes sendo comutados pelo *switch* e, quando encontrado um padrão de tráfego que corresponde com uma das regras estabelecidas, um alerta é enviado ao controlador Ryu, que o exibe na tela.

Todas as regras de detecção já contidas no Snort foram desativadas para a validação deste trabalho, deixando ativadas apenas as regras elaboradas para a detecção dos ataques simulados e a filtragem de eventos.

4.1.1 Detecção do ataque varredura de portas TCP SYN

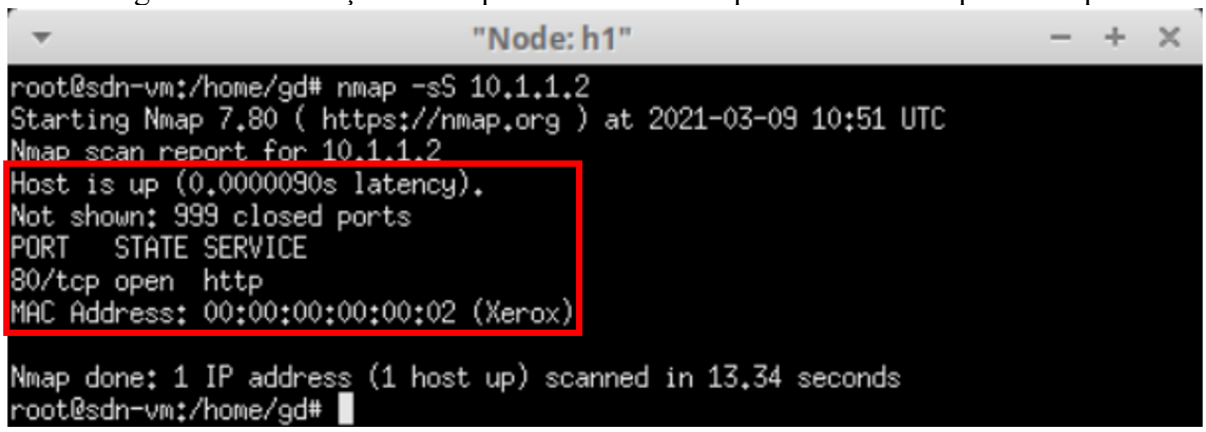
Para testar o funcionamento do ataque de varredura de portas TCP SYN, foi aberto no *host h2* um servidor web utilizando o protocolo HTTP aceitando requisições na porta 80 através do módulo do Python *SimpleHTTPServer*, demonstrado na Figura 26.

Figura 26 – Abertura de servidor web no *host h2*

Fonte: Elaborado pelo autor.

O ataque foi realizado através da ferramenta Nmap, onde foi utilizado o parâmetro de ataque de varredura de portas TCP SYN direcionado à um endereço de IP de uma máquina específica, neste caso o 10.1.1.2, que foi o endereço utilizado para o *host h2* no ambiente simulado. A execução e resultado do ataque são exibidos na Figura 27.

Figura 27 – Execução do ataque de varredura de portas TCP SYN pelo Nmap

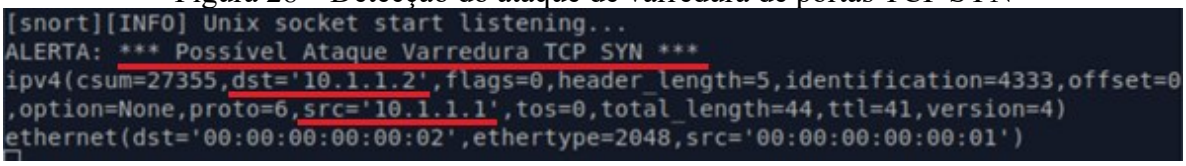


Fonte: Elaborado pelo autor.

Nota-se que o ataque foi bem sucedido, revelando que o *host h2* está ativo, e, entre as 1000 portas testadas, 999 dessas portas estão fechadas e uma porta está aberta, a porta 80. É também afirmado que essa porta está oferecendo o serviço HTTP. Outros dados apresentados são o endereço de MAC do *host* vítima e o resumo do resultado do ataque, evidenciando o número de *hosts* escaneados e a duração total do ataque.

Quando o ataque é efetuado, o *switch* OpenFlow, além de enviar os pacotes para o *host* vítima, também os envia à interface *snort0*, onde o NIDS Snort está analisando o tráfego e, como o padrão de tráfego gerado pelo ataque corresponde a regra criada para o mesmo, o Snort o detecta. Com a detecção do ataque, a ação estipulada na regra é acionada, que, neste caso, é a geração de um alerta. Como configurado neste ambiente, os alertas do Snort são enviados para o controlador Ryu através do *UNIX domain socket*. Por fim, o alerta enviado pelo Snort é recebido e exibido no terminal do Ryu, como é demonstrado na Figura 28.

Figura 28 – Detecção do ataque de varredura de portas TCP SYN



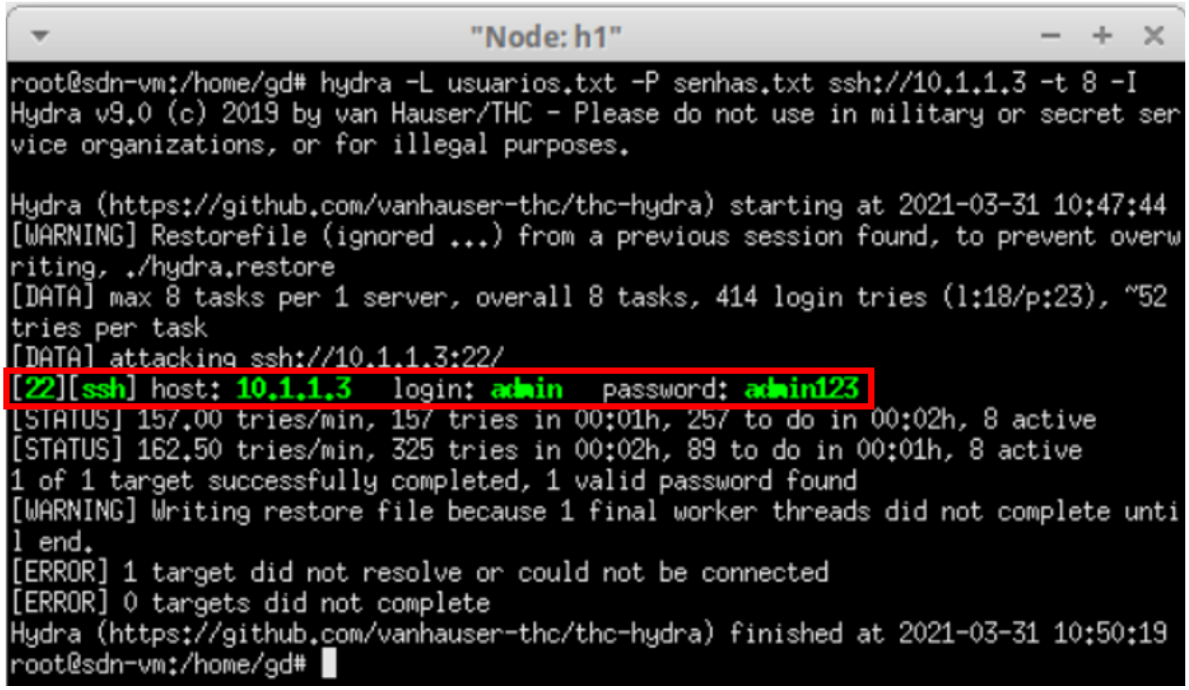
Fonte: Elaborado pelo autor.

Na mensagem contida no alerta, é evidenciado o tipo de ataque detectado, estipulado na opção *msg* da regra, e algumas informações sobre o pacote, com maior ênfase o seu endereço de IP de origem, representado pelo campo *src*, e o endereço IP de destino, representado pelo campo *dst*. Essas informações, sublinhadas em vermelho na Figura 28, são utilizadas para a mitigação dos ataques através da alteração da tabela de fluxos de rede no *switch* OpenFlow pelo controlador Ryu e seu módulo *ofctl_rest*.

4.1.2 Detecção do ataque de dicionário no protocolo SSH

A execução do ataque de dicionário no protocolo SSH foi feita através da ferramenta THC Hydra. Na ferramenta, foram utilizados os seguintes parâmetros: para indicar a localização dos arquivos de texto contendo os *logins* e senhas para realizar as tentativas de *login*, para determinar o protocolo e endereço de IP para ataque, para realizar o ataque com oito tarefas em paralelo e, para ignorar sessões anteriores, iniciando um novo ataque. O protocolo escolhido de ataque foi o SSH e o endereço de IP foi o 10.1.1.3, que foi o IP utilizado no ambiente simulado para o *host h3*. A execução e resultado do ataque são exibidos na Figura 29.

Figura 29 – Execução do ataque de dicionário no protocolo SSH pelo THC Hydra

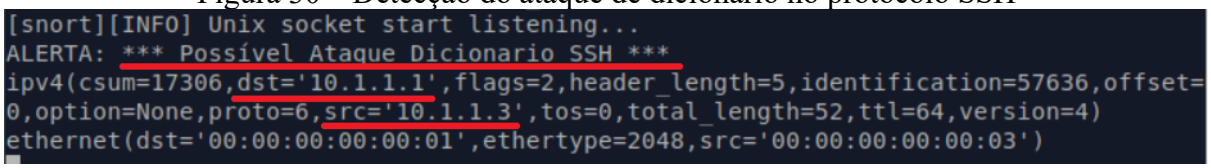


Fonte: Elaborado pelo autor.

Observa-se que o ataque é realizado com êxito, encontrando uma combinação válida de *login* "*admin*" e senha "*admin123*" presentes na máquina do *host h3*. Essa combinação pode ser utilizada para obter acesso a essa máquina através do protocolo SSH. Adicionalmente, apresenta estatísticas sobre a duração do ataque, o número de alvos atacados e de combinações de *login* encontrados.

No momento da realização deste ataque, a mesma sequência de eventos relatada na seção anterior se sucede para que o Snort detecte o ataque e o alerta disparado seja exibido no terminal do Ryu. A detecção do ataque é demonstrada na Figura 30.

Figura 30 – Detecção do ataque de dicionário no protocolo SSH



Fonte: Elaborado pelo autor.

As informações exibidas na mensagem do alerta, exceto a mensagem que relata que tipo de ataque foi detectado, são idênticas as descritas na seção anterior. Da mesma maneira, as informações de maior relevância são os endereços IP de origem e destino do pacote pertencente ao ataque, sublinhados em vermelho na Figura 30.

4.2 MITIGAÇÃO DOS ATAQUES

O administrador de rede deve estar atento ao terminal do controlador Ryu onde os alertas serão exibidos para rapidamente detectar os potenciais ataques. Se uma intrusão for detectada com rapidez, o atacante pode ser identificado e expulsado do sistema antes de causar danos graves. Quanto mais cedo o ataque for detectado, menor será o dano causado e o tempo de recuperação necessário (STALLINGS, 2014).

Após a detecção dos ataques cibernéticos pelas regras criadas no Snort e o envio dos alertas ao Ryu, a mitigação dos ataques foi realizada através de mudanças na tabela de fluxos de rede do *switch* OpenFlow para impedir a comunicação entre o *host* atacante e o restante da rede. As mudanças foram feitas através do Postman, enviando mensagens HTTP para a aplicação *ofctl_rest* através da sua API REST, a qual envia as requisições para o controlador Ryu, que por sua vez envia os comandos de alteração da tabela de fluxos para o *switch* através do protocolo OpenFlow.

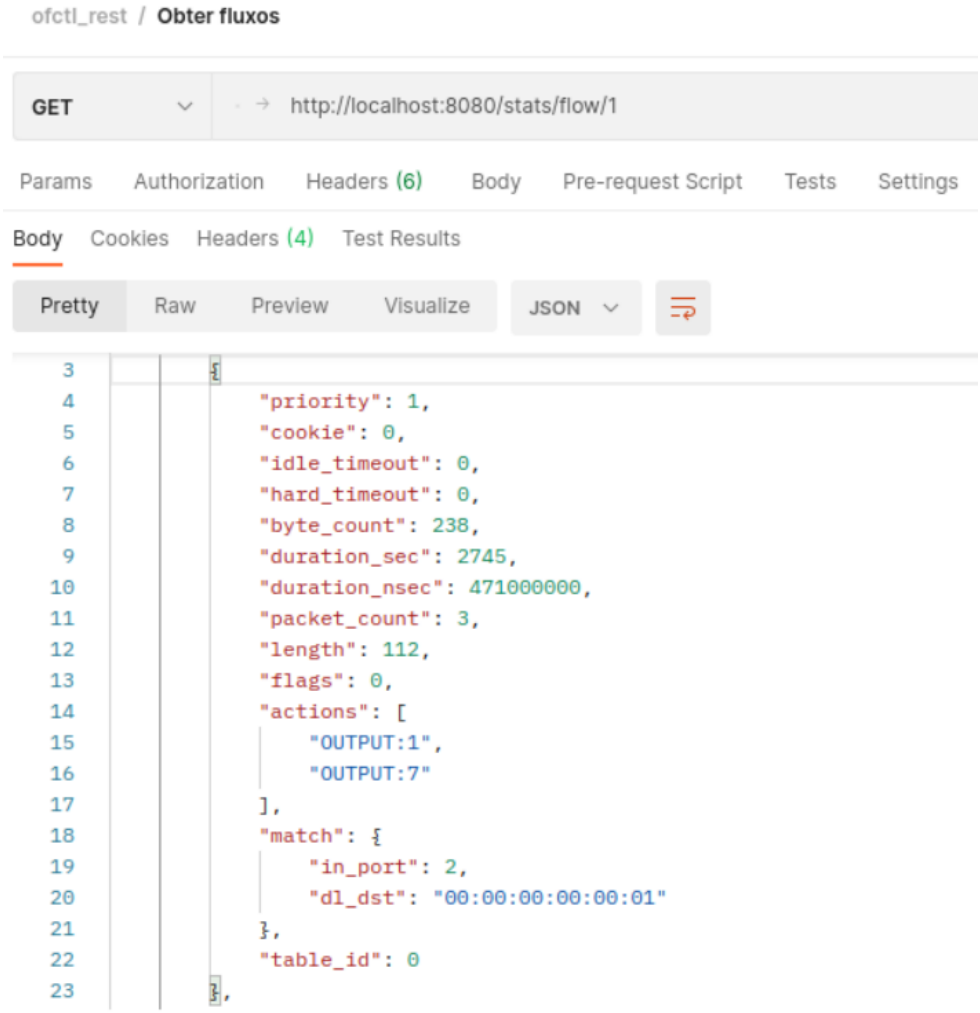
Através do Postman, é possível enviar comandos ao controlador Ryu por meio da API REST fornecida pelo módulo *ofctl_rest* para extrair informações ou modificar elementos do *switch* OpenFlow, como a sua tabela de fluxos.

No Postman, para visualizar a tabela de fluxos do *switch* e suas entradas, a requisição HTTP do tipo GET foi utilizada, com o URI "*http://localhost:8080/stats/flow/1*", onde está sendo utilizado o protocolo HTTP para enviar uma requisição para o serviço do *ofctl_rest* sendo executado na mesma máquina, na porta 8080. As requisições GET são geralmente utilizadas para solicitar a um servidor HTTP o envio de um recurso (GOURLEY *et al.*, 2002).

Os recursos buscados por esta requisição feita são as estatísticas da tabela de fluxos do *switch* com o DPID de número "1". O DPID, do inglês *Datapath Identifier*, é o número

identificador de uma instância OpenFlow executando em um *switch* que suporte o protocolo OpenFlow (BOMBAL, 2021). Um exemplo de um fluxo na tabela de fluxos obtido através da resposta desta requisição é exibido na Figura 31, onde a resposta é formatada na linguagem JSON, que é um formato de texto para a troca de dados entre plataformas, com o intuito de ser fácil tanto para humanos lerem e escreverem quanto para máquinas analisarem e gerarem (BASSETT, 2015).

Figura 31 – Exemplo de fluxo na tabela de fluxos



Fonte: Elaborado pelo autor.

A entrada de fluxo é composta de suas propriedades distribuídas em campos, como o seu valor de prioridade, número de *bytes* trafegados, duração, pacotes enviados, entre outros. O campo "*match*" determina quais pacotes fazem parte do fluxo e o campo "*actions*" instrui o *switch* como deverá tratar o tráfego contido no fluxo (RYU, 2021).

No exemplo da Figura 31, o fluxo corresponde a pacotes que chegam na porta de número 2 do *switch* e possuem o endereço MAC de destino "00:00:00:00:00:01". As ações

determinadas no fluxo estabelecem que os pacotes correspondentes ao fluxo serão enviados às portas 1 e 7. No ambiente de testes deste trabalho, as portas 1 e 7 correspondem a porta conectada ao *host h1* e a porta *snort0*, respectivamente.

Antes de efetuar a mitigação, o *host* atacante possui conectividade normal com a rede, onde é capaz de realizar os ataques. Isso é evidenciado na Figura 32, onde é executado o comando *pingall* no Mininet, que é utilizado para testar a conectividade entre os *hosts* da rede simulada através da ferramenta *ping*. O resultado aponta que nenhum pacote foi perdido, evidenciando que todos os *hosts* possuem conectividade entre si.

Figura 32 – Conectividade normal entre todos os *hosts*

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
```

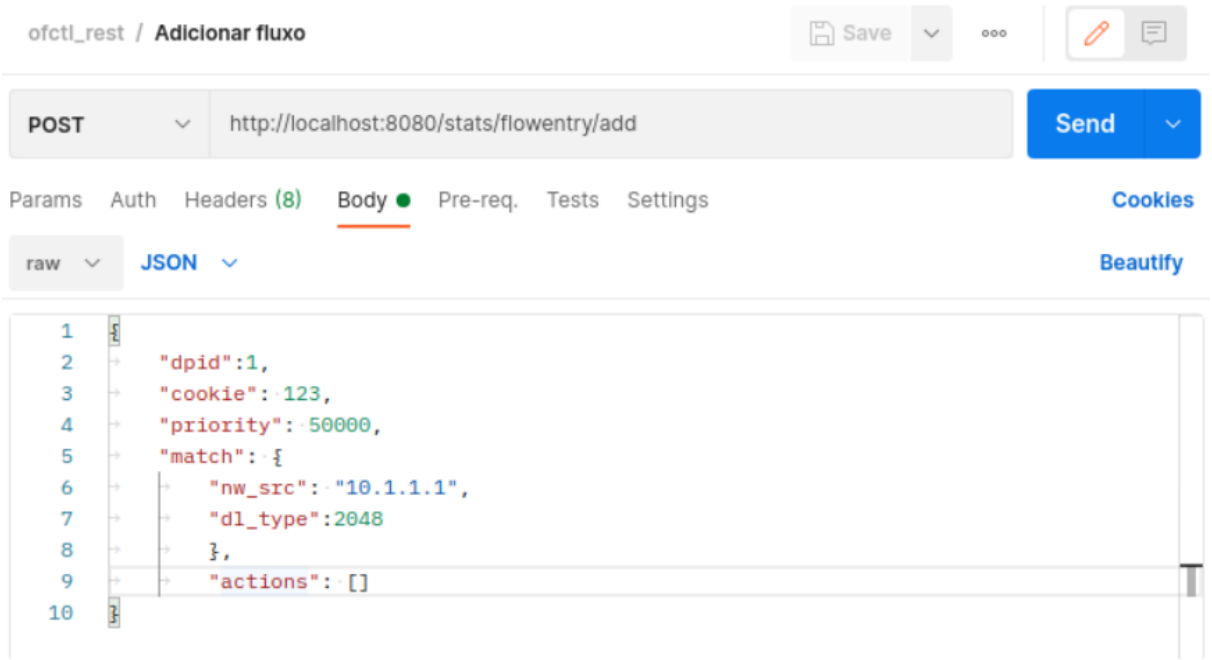
Fonte: Elaborado pelo autor.

A mitigação dos ataques visa impedir a comunicação entre o *host* atacante e o restante da rede. Para isso, foram inseridos dois fluxos na tabela do *switch* para bloquear a conectividade do *host* atacante (*h1*). O primeiro fluxo corresponde a pacotes que contenham o endereço IP de origem do *host h1* e o segundo fluxo corresponde a pacotes que possuem o endereço IP de destino do *host h1*. Ambos os fluxos determinam que o *switch* deverá descartar o pacote. Assim, essas entradas de fluxo inseridas servem para bloquear o atacante de enviar ou receber pacotes do restante da rede.

Para adicionar as novas entradas na tabela de fluxos, a requisição HTTP do tipo POST foi utilizada, com o URI "*http://localhost:8080/stats/flowentry/add*". As requisições POST podem ser utilizadas para enviar dados de inserção ao servidor HTTP (GOURLEY *et al.*, 2002). Nesta requisição, a entrada de fluxo é especificada no corpo da mensagem HTTP, na linguagem JSON.

A primeira entrada adicionada na tabela de fluxos através do Postman é apresentada na Figura 33, onde é estabelecido o fluxo que descarta todos os pacotes que se originam do *host* atacante, bloqueando uma das vias de comunicação do atacante com o restante da rede. A requisição é enviada através do botão "*Send*".

Figura 33 – Fluxo inserido para bloquear pacotes originados do atacante

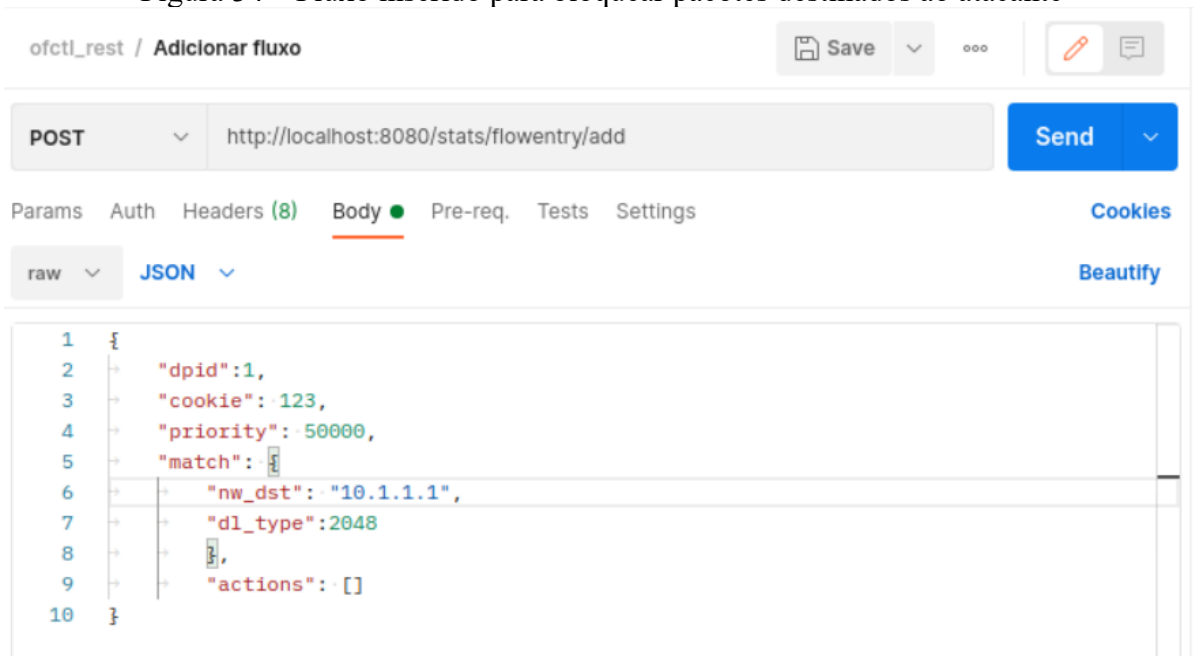


Fonte: Elaborado pelo autor.

Nesta inserção de entrada de fluxo, é necessária a indicação do DPID para que o controlador saiba para qual *switch* deve enviar a alteração, que neste caso foi o *switch* de DPID de número "1". O campo *match*, através do subcampo "*nw_src*", determina que os pacotes pertencentes a este fluxo devem possuir o endereço IP de origem "10.1.1.1", que é o endereço IP do *host* atacante, e determina, através do campo "*dl_type*", que deve ser um pacote IPv4. O campo *actions* se encontra vazio, o que significa que o *switch* irá descartar o tráfego correspondente a este fluxo. O campo *priority* foi utilizado para que o fluxo possuísse uma maior prioridade na tomada de decisão, onde, se um pacote corresponder a mais de um fluxo, o fluxo com o maior valor de prioridade será selecionado para tratar o pacote. Por fim, o campo *cookie* é utilizado para filtragem, onde, por exemplo, podem ser consultadas apenas as entradas de fluxo que possuem o valor "123" no campo *cookie* (ONF, 2012).

A segunda entrada adicionada na tabela de fluxos é apresentada na Figura 34, onde é estabelecido o fluxo que descarta todos os pacotes destinados ao *host* atacante, bloqueando a segunda via de comunicação do atacante com o restante da rede.

Figura 34 – Fluxo inserido para bloquear pacotes destinados ao atacante

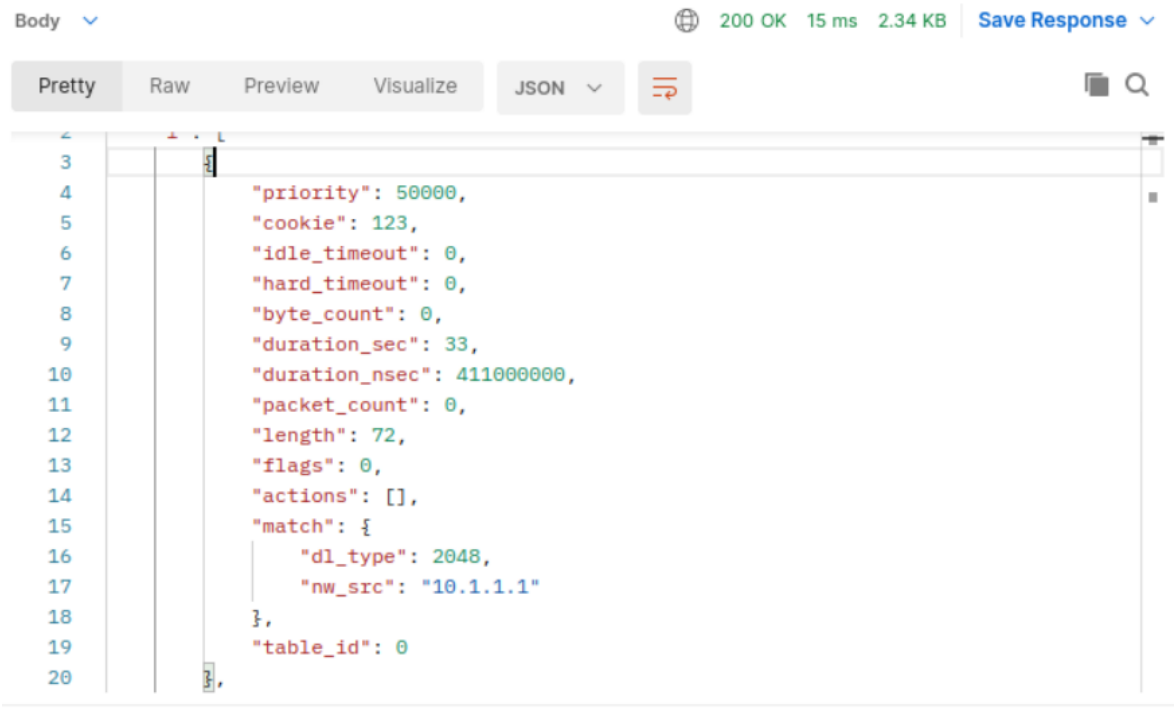


Fonte: Elaborado pelo autor.

Esta inserção de entrada de fluxo utiliza as mesmas propriedades do fluxo anterior, apenas se diferenciando no campo *match*, trocando o seu subcampo *nw_src* pelo subcampo *nw_dst*, o qual determina que os pacotes pertencentes a este fluxo devem possuir o endereço IP de destino “10.1.1.1”.

Uma destas entradas inseridas na tabela de fluxos para a mitigação dos ataques é apresentada na Figura 35, com as suas estabelecidas propriedades, como o campo *priority* com o valor “50000” para ser o fluxo de maior prioridade e o campo *cookie* de valor “123” para poder ser filtrado dos demais fluxos. A mesma requisição HTTP GET apresentada na Figura 31 foi utilizada para obter os dados do fluxo.

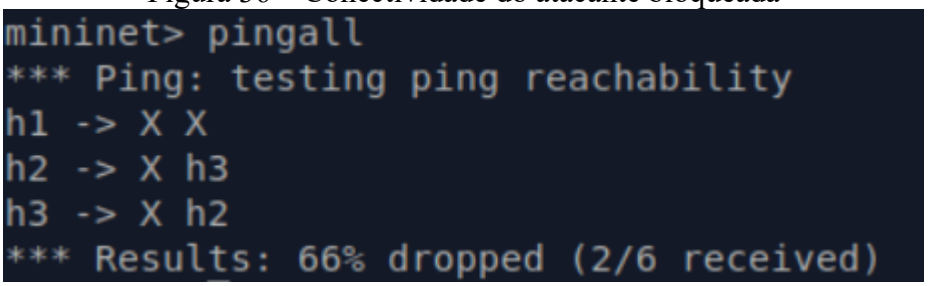
Figura 35 – Fluxo inserido para mitigação dos ataques



Fonte: Elaborado pelo autor.

Com a inserção das duas entradas de fluxo na tabela, o *host* atacante é impossibilitado de enviar ou receber qualquer pacote IPv4 do restante da rede, deste modo, mitigando sua possível danosa influência na rede. Isso é evidenciado na Figura 36, onde a execução do comando *pingall* resulta na falha da conexão entre o *host* atacante (*h1*) e os demais *hosts* da rede, enquanto os *hosts h2* e *h3* possuem conectividade normal entre si.

Figura 36 – Conectividade do atacante bloqueada

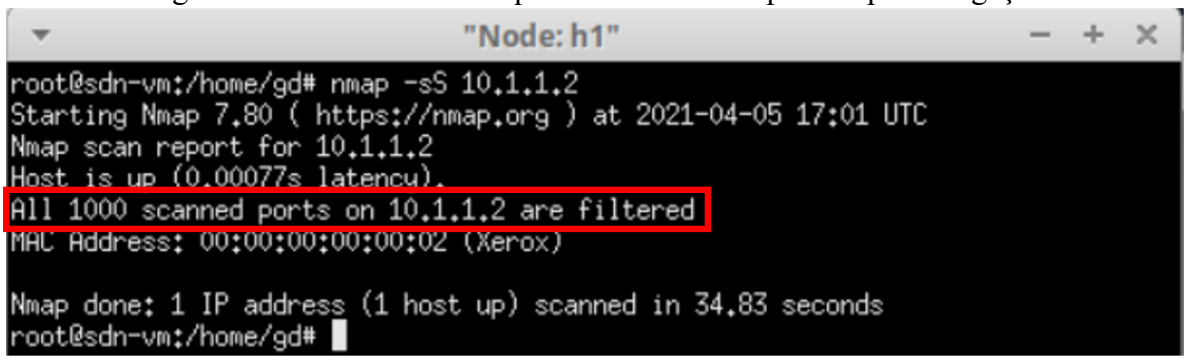


Fonte: Elaborado pelo autor.

A efetividade da mitigação de um potencial futuro ataque de varredura de portas TCP SYN é evidenciada na Figura 37, onde a ferramenta falha no descobrimento de portas abertas, apontando que todas as portas do *host* vítima estão “filtradas”, devido ao *switch* descartando todos os pacotes IPv4 sendo enviados pelo *host* atacante. No Nmap, portas que aparecem como

“filtradas” são portas que a ferramenta não consegue deduzir se estão abertas ou não (LYON, 2008).

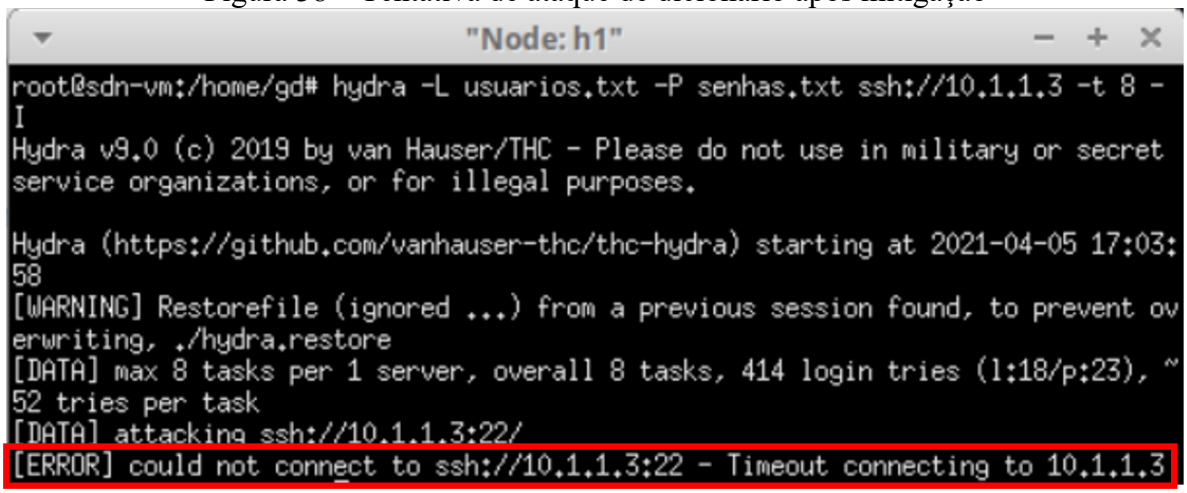
Figura 37 – Tentativa de ataque de varredura de portas após mitigação



Fonte: Elaborado pelo autor.

A efetividade da mitigação de um potencial futuro ataque de dicionário no protocolo SSH é evidenciada na Figura 38, onde a ferramenta falha na realização do ataque, apontando que não consegue conectar ao servidor SSH, devido ao *switch* descartando todos os pacotes IPv4 sendo enviados pelo *host* atacante.

Figura 38 – Tentativa de ataque de dicionário após mitigação



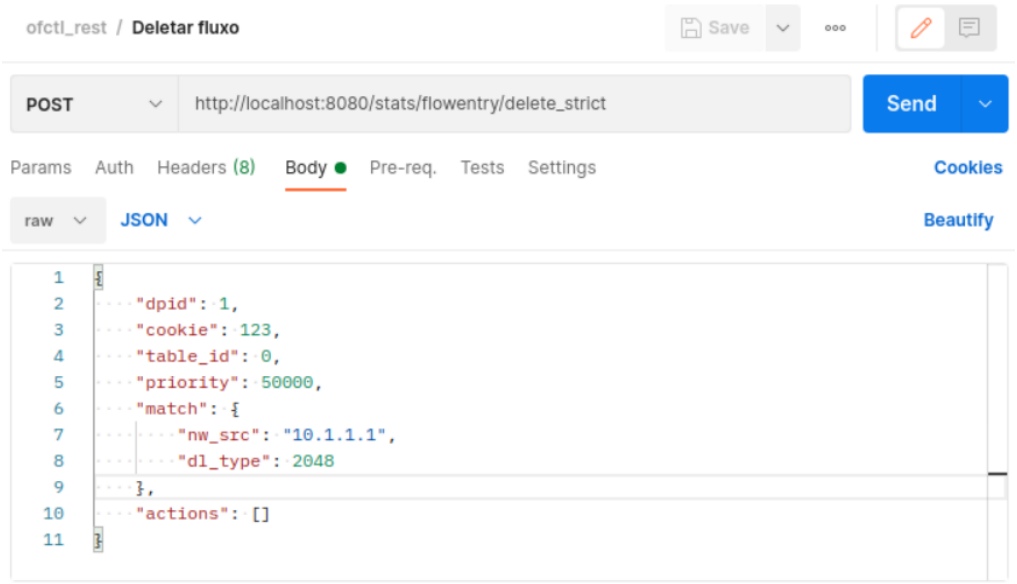
Fonte: Elaborado pelo autor.

Com a detecção e mitigação dos ataques, assume-se que a máquina do atacante seja então desativada da rede fisicamente. Após a resolução do problema, os fluxos adicionados para a mitigação dos ataques podem ser deletados da tabela de fluxos do *switch* para liberar o endereço de IP do antigo atacante para ser utilizado novamente.

A exclusão das entradas inseridas na tabela de fluxos, ilustradas nas Figuras 39 e 40, foram feitas através da requisição HTTP POST utilizando o URI

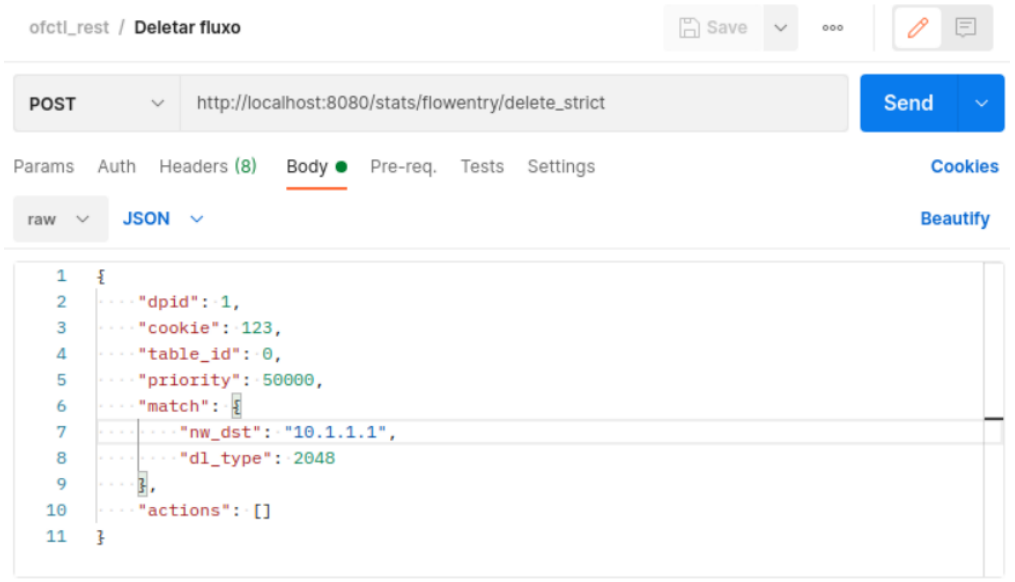
"*http://localhost:8080/stats/flowentry/delete_strict*", com o corpo da mensagem HTTP sendo idêntico ao da inserção da entrada de fluxo correspondente.

Figura 39 – Primeiro fluxo deletado



Fonte: Elaborado pelo autor.

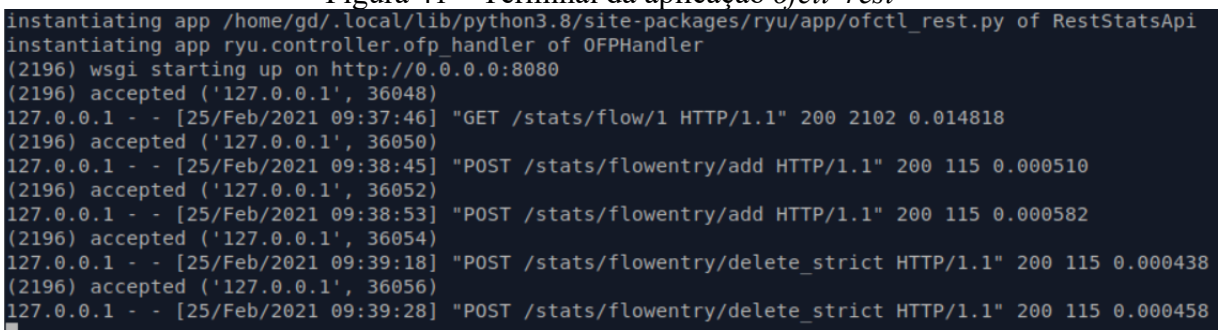
Figura 40 – Segundo fluxo deletado



Fonte: Elaborado pelo autor.

O terminal da aplicação *ofctl_rest*, que exibe o recebimento e a resposta às requisições HTTP feitas através do Postman para a visualização, inserção e exclusão das entradas da tabela de fluxos do *switch*, é demonstrado na Figura 41.

Figura 41 – Terminal da aplicação *ofctl_rest*



Fonte: Elaborado pelo autor.

No ambiente de testes, após a remoção das entradas inseridas na tabela de fluxos, a tabela volta a ficar inalterada, como estava exibida na Figura 31, e o resultado do comando *pingall* volta a mostrar que a conexão entre todos os *hosts* é normal, como anteriormente evidenciado na Figura 32.

Diante disso, pode-se afirmar que a utilização do NIDS Snort, com estas particulares regras elaboradas, neste ambiente de rede SDN simulado, foi eficaz na detecção dos ataques executados e no envio dos alertas gerados para o controlador Ryu. Com os dados fornecidos pela detecção, foi possível realizar a mitigação dos ataques através da utilização da aplicação *ofctl_rest* rodando conjuntamente ao controlador Ryu, utilizando APIs para a comunicação entre o administrador de rede e o controlador para a alteração da tabela de fluxos do *switch* OpenFlow.

5 CONSIDERAÇÕES FINAIS

Ao mesmo passo do aumento da utilização e do avanço tecnológico das redes de computadores, onde novos modelos de operação estão sendo adotados, como o SDN, a segurança é um requerimento que não pode ser ignorado. Administradores de rede não devem apenas considerar, mas priorizar a segurança enquanto planejam, implementam e operam suas redes. Isso torna-se essencial para um design de rede bem-sucedido, onde os controles de segurança implementados protegem os bens de uma organização, como também sua reputação.

Visto que os sistemas de detecção de intrusão baseados em rede são uma ferramenta importante no arsenal de segurança de uma rede, este trabalho testou a utilização do sistema de detecção de intrusão baseado em rede Snort em um ambiente simulado de rede definida por software para o auxílio na mitigação de ataques.

Diante disso, o trabalho teve como objetivo geral averiguar a habilidade do NIDS Snort de auxiliar na segurança de um ambiente simulado de rede definida por software, onde ficou encarregado de identificar os ataques executados no ambiente e enviar os alertas disparados ao controlador da rede. O trabalho demonstrou que o Snort, conjunto as regras de detecção elaboradas, conseguiu efetivamente desempenhar o seu papel na segurança da rede SDN, auxiliando na mitigação dos ataques executados no ambiente através da detecção dos ataques e dos alertas enviados ao controlador contendo importantes dados sobre os ataques. Portanto, constata-se que o objetivo geral do trabalho foi atingido.

Deste modo, o problema indagado no trabalho foi respondido, visto que foi identificado que a utilização do sistema de detecção de intrusão baseado em rede Snort em um ambiente de rede definida por software foi de fato eficaz no auxílio à mitigação de ataques destinados à rede.

Como sugestões para trabalhos futuros, propõe-se verificar a efetividade do método de mitigação apresentado com outros tipos de ataques cibernéticos e experimentar a utilização de outro sistema de detecção de intrusão baseado em rede e outros controladores para a mitigação de ataques em uma arquitetura de rede definida por software.

REFERÊNCIAS

BARRETT, Daniel J.; SILVERMAN, Richard E.; BYRNES, Robert G.. **SSH, the Secure Shell**: The Definitive Guide. 2. ed. O' Reilly, 2005.

BASSETT, Lindsay. **Introduction to JavaScript Object Notation**. O' Reilly, 2015.

BOMBAL, David. **Datapath IDs**. 2021. Disponível em: https://pakiti.com/datapath-ids/. Acesso em: 17 fev. 2021.

CHUVAKIN, Dr. Anton A. *et al.* **Logging and Log Management**: the authoritative guide to understanding the concepts surrounding logging and log management. Elsevier, 2013.

CIELO. **Postman Cielo**: documentações e tutoriais. 2021. Disponível em: https://developercielo.github.io/tutorial/postman. Acesso em: 17 fev. 2021.

CISCO. **Catalyst Switched Port Analyzer (SPAN) Configuration Example**. 2019. Disponível em: https://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/10570-41.html. Acesso em: 27 fev. 2021.

CISCO. **Relatório Global de Tendências de Redes de 2020**. 2019.

COTTON, Michelle *et al.* Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. **RFC 6335**, 2011.

COX, Kerry; GREG, Christopher. **Managing Security with Snort and IDS Tools**. O' Reilly, 2004.

ELIAS, Glêdson; LOBATO, Luiz Carlos. **Arquitetura e Protocolos de Rede TCP-IP**. 2. ed. Rio de Janeiro: RNP, 2013. 414 p.

FALL, Kevin R.; STEVENS, W. Richard. **TCP/IP Illustrated, Volume 1**: The Protocols. 2. ed. Addison-Wesley, 2012.

GADGE, Jayant; PATIL, Anish Anand. Port scan detection. **2008 16th IEEE International Conference on Networks**. IEEE, 2008. p. 1-6.

GOURLEY, David *et al.* **HTTP**: The Definitive Guide. O' Reilly, 2002.

GRAVES, Kimberly. **Official Certified Ethical Hacker Review Guide**. Wiley, 2007.

GUIMARAES, Mario; MURRAY, Meg. Overview of intrusion detection and intrusion prevention. In: **Proceedings of the 5th annual conference on Information security curriculum development**. 2008. p. 44-46.

HALEPLIDIS, Evangelos *et al.* Software-defined networking (SDN): Layers and architecture terminology. **RFC 7426**, 2015.

HUMAYUN, Mamoona *et al.* Cyber Security Threats and Vulnerabilities: a systematic mapping study. **Arabian Journal For Science And Engineering**, v. 45, n. 4, p. 3171-3189, 6 jan. 2020. Springer Science and Business Media LLC. http://dx.doi.org/10.1007/s13369-019-04319-2.

IBM. **TCP connection flow**. 2021. Disponível em: https://www.ibm.com/support/knowledgecenter/SSB23S_1.1.0.2020/gtps7/s5tcpcf.html. Acesso em: 25 fev. 2021.

IBM. **UNIX domain sockets**. 2021. Disponível em: https://www.ibm.com/support/knowledgecenter/en/SSB23S_1.1.0.14/gtpc1/unixsock.html. Acesso em: 27 fev. 2021.

KAKARLA, Tejaswi; MAIRAJ, Aakif; JAVAID, Ahmad Y. A Real-World Password Cracking Demonstration Using Open Source Tools for Instructional Use. **2018 IEEE International Conference on Electro/Information Technology (EIT)**. IEEE, 2018. p. 0387-0391.

KARAKUS, Murat; DURRESI, Arjan. A survey: control plane scalability issues and approaches in software-defined networking (SDN). **Computer Networks**, v. 112, p. 279-293, jan. 2017. Elsevier BV. http://dx.doi.org/10.1016/j.comnet.2016.11.017.

KIM, Hyojoon; FEAMSTER, Nick. Improving network management with software defined networking. **IEEE Communications Magazine**, v. 51, n. 2, p. 114-119, fev. 2013. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/mcom.2013.6461195.

KREUTZ, Diego *et al.* Software-Defined Networking: a comprehensive survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14-76, jan. 2015. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/jproc.2014.2371999.

LIAO, Hung-Jen *et al.* Intrusion detection system: a comprehensive review. **Journal of Network and Computer Applications**, v. 36, n. 1, p. 16-24, jan. 2013. Elsevier BV. http://dx.doi.org/10.1016/j.jnca.2012.09.004.

LIU, Simon; CHENG, Bruce. Cyberattacks: why, what, who, and how. **It Professional**, v. 11, n. 3, p. 14-21, maio 2009. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/mitp.2009.46.

LYON, Gordon. **Nmap Network Scanning**: Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure.Com, 2008.

MAŠETIĆ, Zerina *et al.* SYN flood attack detection in cloud computing using support vector machine. **TEM Journal**, v. 6, n. 4, p. 752, 2017.

MCKEOWN, Nick *et al.* OpenFlow: enabling innovation in campus networks. **ACM SIGCOMM Computer Communication Review**, v. 38, n. 2, p. 69-74, 31 mar. 2008. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/1355734.1355746.

MELO, Sandro. **Exploração de Vulnerabilidades em Redes TCP/IP**. 3. ed. Rio de Janeiro: Alta Books, 2017.

MININET. **Introduction to Mininet**. 2021. Disponível em: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet. Acesso em: 26 jan. 2021.

MININET. **Mininet**: An Instant Virtual Network on Your Laptop (or Other PC). 2020. Disponível em: http://mininet.org/. Acesso em: 10 out. 2020.

NAKAMURA, Emílio Tissato. **Segurança da informação e de redes**. Londrina: Educacional, 2016. 224 p.

NAKAMURA, Emílio Tissato; GEUS, Paulo Lício de. **Segurança de redes em ambientes cooperativos**. São Paulo: Novatec, 2007.

NIELES, Michael; DEMPSEY, Kelley; PILLITTERI, Victoria Yan. An Introduction to Information Security. **NIST special publication**, p. 12, jun. 2017. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-12r1

NORTHCUTT, Stephen; NOVAK, Judy. **Network Intrusion Detection**. 3. ed. New Riders, 2002.

NUNES, Bruno Astuto A. *et al.* A Survey of Software-Defined Networking: past, present, and future of programmable networks. **IEEE Communications Surveys & Tutorials**, v. 16, n. 3, p. 1617-1634, 2014. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/surv.2014.012214.00180.

ODOM, Wendell. **Introduction to Controller-Based Networking**. 2020. Disponível em: https://www.ciscopress.com/articles/article.asp?p=2995354&seqNum=2. Acesso em: 08 jan. 2021.

OPEN NETWORKING FOUNDATION. **VERSION 1.3.1**: OpenFlow Switch Specification. 2012.

OWENS, Jim; MATTHEWS, Jeanna. A study of passwords and methods used in brute-force SSH attacks. In: **USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)**. 2008.

PATHAN, Al-Sakib Khan (ed.). **The State of the Art in Intrusion Prevention and Detection**. Boca Raton: Crc Press, 2014.

PINKAS, Benny; SANDER, Tomas. Securing passwords against dictionary attacks. In: **Proceedings of the 9th ACM Conference on Computer and Communications Security**. 2002. p. 161-170.

ROTHENBERG, Christian Esteve *et al.* OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. **Cad. CPqD Tecnologia, Campinas**, v. 7, n. 1, p. 65-76, 2010.

RYU. **Ryu SDN Framework**. 2020. Disponível em: https://ryu-sdn.org/index.html. Acesso em: 14 out. 2020.

RYU. **Ryu.app.ofctl_rest**. 2021. Disponível em: https://ryu.readthedocs.io/en/latest/app/ofctl_rest.html. Acesso em: 23 jan. 2021.

RYU. **Snort Integration**. 2021. Disponível em: https://ryu.readthedocs.io/en/latest/snort_integrate.html. Acesso em: 20 jan. 2021.

SALAH, K.; KAHTANI, A.. Performance evaluation comparison of Snort NIDS under Linux and Windows Server. **Journal of Network and Computer Applications**, v. 33, n. 1, p. 6-15, jan. 2010.

SATOH, Akihiro *et al.* A flow-based detection method for stealthy dictionary attacks against Secure Shell. **Journal of Information Security and Applications**, v. 21, p. 31-41, abr. 2015. Elsevier BV. http://dx.doi.org/10.1016/j.jisa.2014.08.003.

SCARFONE, Karen; MELL, Peter. Guide to Intrusion Detection and Prevention Systems (IDPS). **NIST special publication**, p. 94, 2007. National Institute of Standards and Technology.

SNORT. **Snort Users Manual**. 2020. Disponível em: https://snort.org/documents/1. Acesso em: 10 jan. 2021.

SNORT. **Snort - Network Intrusion Detection & Prevention System**. 2021. Disponível em: https://snort.org/. Acesso em: 08 jan. 2021.

STALLINGS, William. **Segurança de computadores**. 2. ed. Rio de Janeiro: Elsevier, 2014. Stallings (2014) – (STALLINGS, 2014)

VERIZON. **2021 Data Breach Investigations Report**. 2021.

VIVO, Marco de *et al.* A review of port scanning techniques. **ACM SIGCOMM Computer Communication Review**, v. 29, n. 2, p. 41-48, abr. 1999. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/505733.505737.

VON SOLMS, Rossouw; VAN NIEKERK, Johan. From information security to cyber security. **Computers & Security**, v. 38, p. 97-102, out. 2013. Elsevier BV. http://dx.doi.org/10.1016/j.cose.2013.04.004.

WIRESHARK. **Chapter 1**: introduction. Disponível em: https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html. Acesso em: 27 fev. 2021.

YLONEN, Tatu *et al.* The Secure Shell (SSH) Protocol Architecture. **RFC 4251**, 2006.