UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MATHEUS SAPIA GUERRA

FERRAMENTA PARA ANÁLISE DE REGRAS DE FIREWALL

MONOGRAFIA

CAMPO MOURÃO
2019

MATHEUS SAPIA GUERRA

FERRAMENTA PARA ANÁLISE DE REGRAS DE FIREWALL

Trabalho de Conclusão de Curso de Graduação apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Bacharelado em Ciência da Computação do Departamento Acadêmico de Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Luiz Arthur Feitosa dos Santos

CAMPO MOURÃO
2019

ATA DE DEFESA DO TRABALHO DE CONCLUSÃO DE CURSO

Às **15:30** do dia **27 de novembro de 2019** foi realizada na sala **E101** da UTFPR-CM a sessão pública da defesa do Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação do(a) acadêmico(a) **Matheus Sapia Guerra** com o título **Ferramenta para análise de regras de firewall**. Estavam presentes, além do(a) acadêmico(a), os membros da banca examinadora composta por: **Prof. Dr. Luiz Arthur Feitosa dos Santos** (orientador(a)), **Prof. Dr. Rodrigo Campiolo e Prof. Dr. Rogério Aparecido Gonçalves**. Inicialmente, o(a) acadêmico(a) fez a apresentação do seu trabalho, sendo, em seguida, arguido(a) pela banca examinadora. Após as arguições, sem a presença do(a) acadêmico(a), a banca examinadora o(a) considerou _____ na disciplina de Trabalho de Conclusão de Curso 2 e atribuiu, em consenso, a nota _____ (_____). Esse resultado foi comunicado ao (à) acadêmico(a) e aos presentes na sessão pública. A banca examinadora também comunicou ao (à) acadêmico(a) que este resultado fica condicionado à entrega da versão final dentro dos padrões e da documentação exigida pela UTFPR ao professor responsável do TCC no prazo de **onze dias**. Em seguida foi encerrada a sessão e, para constar, foi lavrada a presente Ata que segue assinada pelos membros da banca examinadora, após lida e considerada conforme.

Observações:

_____

_____

_____

_____

Campo Mourão, **27 de novembro de 2019**

_____
Prof. Dr. Rodrigo Campiolo
Membro 1

_____
Prof. Dr. Rogério Aparecido Gonçalves
Membro 2

_____
Prof. Dr. Luiz Arthur Feitosa dos Santos
Orientador

A ata de defesa assinada encontra-se na coordenação do curso.

Agradecimentos

Agradeço a todos que de certa forma contribuíram de forma direta ou indireta no processo de desenvolvimento deste trabalho. Gostaria de agradecer em especial:

Ao meu orientador Luiz Arthur Feitosa dos Santos, que pacientemente me guiou durante a minha graduação e elaboração deste trabalho, me transmitiu seu conhecimento com amor e boa vontade.

Aos membros da banca, professores Rodrigo Campiolo e Rogério Aparecido Gonçalves, que aceitaram o meu convite e contribuíram diretamente para a evolução do meu trabalho.

A minha família, em especial aos meus pais, Aroldo Guerra e Gisele Sapia, que lutaram com todas as forças para que eu possa estar realizando este trabalho, e ao meu tio, Helton Sapia, que em um dos momentos mais difíceis, me adotou como filho e aluno.

Aos meus amigos, Vítor Yudi e Nathalia Hammoud por sempre me incentivarem, e a cada dia me provarem que posso aprender mais.

Resumo

Guerra, Matheus Sapia. Ferramenta Para Análise de Regras de Firewall. 2019. 51. f. Monografia (Curso de Bacharelado em Ciência da Computação), Universidade Tecnológica Federal do Paraná. Campo Mourão, 2019.

Ter um *firewall* na rede não é necessariamente sinônimo de segurança. A implementação e manutenção de *firewalls* é uma tarefa complexa, que exige no mínimo conhecimento de conceitos de rede de computadores e cibersegurança. Além disso, o processo de configuração de *firewalls* normalmente não fornece *feedback* a respeito da corretude das regras criadas e implantadas, que pode tornar a rede suscetível à ciberataques. Assim, para saber acerca da eficácia das regras são necessários inúmeros testes com as regras implantadas nos *firewalls*. Todavia, para a realização destes testes, existe a necessidade de criar diferentes cenários de rede para a realização de experimentos. Por conta disso também, atividades práticas elaboradas com o intuito de ensinar a respeito do funcionamento e da configuração de *firewalls*, acabam sendo complexas de serem realizadas, por falta de recursos computacionais e consequentemente, a realização destas atividades é comprometida. Nesse contexto, o objetivo deste trabalho foi desenvolver uma ferramenta que auxilie administradores de redes, estudantes ou entusiastas a testar e configurar *firewalls* de maneira efetiva e prática. Além de ajudar a detectar possíveis erros de configuração, apresentando ao usuário uma interface simples, intuitiva e informativa. Com o intuito de verificar a eficiência da ferramenta proposta, foram realizados experimentos de carga para comprovar que o uso da mesma não é limitada, permitindo aos usuários da ferramenta desenvolver diferentes cenários de redes, juntamente com a configuração de regras de *firewall* em um tempo de execução que torna possível o uso da ferramenta. O diferencial deste trabalho em relação a outros trabalhos, é o fato de utilizar uma abordagem de simular serviços de rede e testar transmissões efetivas de pacotes de rede. Sendo assim, é possível não só administrar, mas também aprender a utilizar *firewalls*, configurando os mesmos com eficiência e exatidão. Provendo redes de computadores com segurança e usabilidade.

Palavras-chaves: Ensino de *firewall*. Cibersegurança. Redes de Computadores. *Iptables*

Abstract

Guerra, Matheus Sapia. A Tool For Firewall Rules Analysis. 2019. 51. f. Monograph (Undergradute Program in Computer Science), Federal University of Technology – Paraná. Campo Mourão, PR, Brazil, 2019.

Having a firewall on your network is not necessarily synonymous with security. Implementing and maintaining firewalls is a complex task that requires minimal knowledge of computer networking and cybersecurity concepts. In addition, the firewall configuration process typically does not provide administrators with feedback about the correctness of created and deployed rules that can make the network susceptible to cyber attacks. Thus, to know about the effectiveness of the rules, many tests with the rules implemented in the firewalls are necessary. However, to perform these tests there is a need to create different network scenarios for conducting experiments. Because of this also, practical activities designed to teach about the operation and configuration of firewalls, are complex to be performed, due to lack of computational resources and consequently, the performance of these activities is compromised. In this context, the objective of this work was to develop a tool that helps network administrators, students or enthusiasts to test and configure firewalls effectively and practically. In addition to helping to detect potential configuration errors, it presents the user with a simple, intuitive and informative interface. In order to verify the efficiency of the proposed tool, load experiments were carried out to prove that its use will not be limited, allowing the user to develop different network scenarios, together with the configuration of firewall rules at a runtime that allows them to be used. makes it possible to use the tool. The difference between this work and other tools is that it uses the approach of simulating network services and testing firewalls. Thus, it is possible not only to administer, but also to learn how to use firewalls, configuring them efficiently and accurately. Providing computer networks with security and usability.

Keywords: Firewall Teaching. Cybersecurity. Computer Networks. Iptables

Lista de figuras

Lista de tabelas

Siglas

API: *Application Programming Interface*
ARP: *Address Resolution Protocol*
ARPANET: *Advanced Research Projects Agency Network*
CSV: *Comma-Separated Values*
DDoS: *Distributed Denial of Service*
DMZ: *Demilitarized Zone*
DNS: *Domain Name System*
DoD: *United States Department of Defense*
FTP: *File Transfer Protocol*
HTTP: *Hypertext Transfer Protocol*
HTTPS: *Hyper Text Transfer Protocol Secure*
ICMP: *Internet Control Message Protocol*
IP: *Internet Protocol*
ISO: *International Organization for Standardization*
JSON: *JavaScript Object Notation*
LAN: *Local Area Network*
MAN: *Metropolitan Area Network*
NAT: *Network Address Translation*
OSI: *Open Systems Interconnection*
PAN: *Personal Area Network*
RARP: *Reverse Address Resolution Protocol*
SMTP: *Simple Mail Transfer Protocol*
SSH: *Secure Shell*
TCP: *Transmission Control Protocol*
TOS: *Type of Service*
TTL: *Time to live*
UDP: *User Datagram Protocol*
UTFPR: Universidade Tecnológica Federal do Paraná
VPN: *Virtual Private Network*
WAN: *Wide Area Network*
WiFi: *Wireless Fidelity*

Sumário

Capítulo 1

Introdução

Assim como na vida real, na qual casas e organizações possuem muros e portões para proteger-se de eventuais intrusos, redes de computadores também necessitam de proteções para prover segurança a seus usuários (ARTYMIAK, 2003). Empresas e pessoas normalmente possuem dados sensíveis armazenados em computadores. Desta forma, tanto dados de empresas quanto de pessoas, podem ser comprometidos por softwares maliciosos (TANENBAUM; WETHERALL, 2010). Neste contexto de cibersegurança, uma das formas de proteger redes de computadores é a utilização de *firewall*.

O uso do *firewall* para proteger redes de computadores é importante, pois *firewalls* atuam como a primeira linha de defesa. Uma das funcionalidades dos *firewalls* são a filtragem de pacotes, permitindo ou bloqueando o tráfego de pacotes na rede de acordo com regras inseridas pelo administrador da rede (ARTYMIAK, 2003).

Todavia, administradores de redes podem enfrentar problemas ao manipular regras de *firewalls*, isso ocorre devido à complexidade de criação e manutenção das regras. Por possuírem grande número de possibilidades de configurações para realizar a filtragem de pacotes e somado ao conhecimento exigido a respeito de redes de computadores e cibersegurança, erros de configurações podem acontecer ocasionando falhas de segurança (STALLINGS, 2008).

Desta forma, o gerenciamento de *firewalls* requer conhecimento e domínio nas áreas de redes e cibersegurança, pois o administradores terão que lidar com vários conceitos importantes, tais como a pilha de protocolos, roteamento, sintaxe e semântica de regras de *firewall*, vulnerabilidades e ciberataques, para então configurar os *firewalls* de maneira correta e efetiva.

Aliado à complexidade de criação, manutenção de regras de *firewall* e com o conhecimento exigido em redes de computadores, existe a dificuldade em criar diferentes cenários de redes com a finalidade de realizar testes práticos em *firewalls*, dificultando também atividades práticas em ambientes de ensino. Tal fato se deve a limitações de recursos e tempo,

que muitas vezes impossibilita a aplicação prática do conteúdo teórico, pois é necessário configurar o cenário de rede, configuração de endereçamentos, rotas, *hosts*, *firewall*, serviços de rede e por fim a realização dos testes com as regras de *firewall*.

Portanto, foi realizado neste trabalho o desenvolvimento de uma ferramenta que auxilie administradores em testar e identificar se regras em *firewalls* estão de fato bloqueando os pacotes indesejados e permitindo os pacotes desejados. Na nossa abordagem, os testes são realizados de forma automatizada com o auxílio de uma interface *web*, na qual é possível criar cenários de redes, configurar endereçamentos, rotas, roteadores e *firewalls*. Além disso, a ferramenta torna possível a verificação do funcionamento das regras de *firewall* através de testes.

Durante experimentos realizados com a ferramenta implementada, foram alcançados dois resultados, o primeiro resultado é referente aos testes com as regras de *firewalls*. O segundo resultado informará o caminho percorrido pelos pacotes de cada teste, visando proporcionar uma melhor visão a respeito do teste realizado e da eficácia das regras implementadas nos *firewalls*.

Como contribuição, esta ferramenta de análise de regras em *firewalls* pode prover maior segurança em redes de computadores, facilitando para o administrador de redes, a identificação de possíveis falhas de segurança ou vulnerabilidades, ou na constatação da eficiência das regras testadas. Outra contribuição é na área de ensino, na qual é possível aprender a configurar e testar regras de *firewall* com a possibilidade de utilização de diferentes cenários de redes. Já que ferramenta permite observar o comportamento do *firewall*, diferentemente do que ocorre atualmente, pois *firewalls* não retornam *feedback* ao usuário indicando se as regras configuradas são válidas.

Diferentemente das demais ferramentas implementadas até o momento, o grande diferencial deste trabalho, é a utilização de simulação de serviços de redes para a realização de testes nas regras de *firewall*, a grande vantagem desta abordagem é a possibilidade de testar os *firewalls* em diferentes cenários de redes, sem a necessidade de testar *firewall* no ambiente de produção, evitando a rede e dispositivos nessa rede a exposição à ataques ou negação de serviços.

Com isso, foi possível concluir que a ferramenta desenvolvida neste trabalho alcançou os objetivos, utilizando uma abordagem diferente. Sendo assim, facilitando a administração de *firewalls*, e também facilitou no aprendizado a respeito de *firewalls*.

Além deste capítulo introdutório, este trabalho dispõe de outros 4 capítulos. O Capítulo 2 aborda conceitos importantes, tais como redes, *firewall*, *iptables*, além de um trabalho relacionado, no qual esses conceitos são necessários para o entendimento deste trabalho. No Capítulo 3 é apresentada a metodologia utilizada para o desenvolvimento da ferramenta de análise de regras de *firewall*. Para validação da ferramenta desenvolvida, foram realizados alguns experimentos, os experimentos assim como os seus resultados é apresentado

no Capítulo 4. Por fim, no Capítulo 5 as conclusões são apresentadas.

Capítulo 2

Conceitos

Este Capítulo tem como objetivo apresentar conceitos que são relevantes para o entendimento do trabalho.

2.1. Redes de Computadores

As Redes de Computadores são conjuntos de nós interligados, nos quais os nós podem ser computadores, impressoras, celulares, ou qualquer outro dispositivo capaz de receber e transmitir informações com outros nós (FOROUZAN, 2008).

Figura 2.1. Nós interligados trocando informações.

As redes podem ser classificadas de várias formas, sendo uma dessas a classificação

por tamanho ou abrangência. E estas classificações podem ser *Personal Area Network* (PAN), *Local Area Network* (LAN), *Metropolitan Area Network* (MAN) ou *Wide Area Network* (WAN). Rede de área pessoa ou PAN são redes de abrangência limitada, pode-se citar como exemplo o uso de *Bluetooth*. Rede local ou LAN são redes privadas, encontradas com frequência em casas, prédios e empresas, muito utilizadas para realizar compartilhamento de recursos e trocas de informações. O tamanho de uma LAN está limitado a aproximadamente 1 Km de distância (TANENBAUM; WETHERALL, 2010). Redes metropolitanas ou MAN contemplam grupos de casas, prédios, empresas ou até mesmo cidades inteiras. O tamanho de abrangência de uma MAN é de aproximadamente 10 Km(TANENBAUM; WETHERALL, 2010). Também, existem as redes geograficamente distribuídas ou WAN, estas contemplam grandes áreas, podendo ser um estado, país ou até mesmo continentes, o seu tamanho está na faixa de 100 a 1000 km (TANENBAUM; WETHERALL, 2010).

O funcionamento da troca de informações em redes de computadores se deve ao fato de existir uma pilha de protocolos. Cada camada desta pilha de protocolos possuí uma função e fornece serviços para as camadas adjacentes. Esta pilha de protocolos é conhecida também como arquitetura de rede e é utilizada para que empresas distintas possam desenvolver *hardware* e *software* que mesmo diferentes consigam comunicar entre si (TANENBAUM; WETHERALL, 2010).

Dentre as arquiteturas de redes, existem dois importantes modelos de referências, o modelo de referência *International Organization for Standardization* (ISO)/*Open Systems Interconnection* (OSI) e o modelo de referência *Transmission Control Protocol* (TCP)/*Internet Protocol* (IP) (TANENBAUM; WETHERALL, 2010). O mais aplicado é o TCP/IP, este detalhado na Seção a seguir.

2.1.1. Modelo TCP/IP

O modelo mais utilizado no mundo é o TCP/IP, isso graças a *Advanced Research Projects Agency Network* (ARPANET) que era uma rede usada para fins militares, financiada pelo *United States Department of Defense* (DoD). Anos depois a ARPANET passou a ser utilizada também para conectar universidades e repartições públicas (TANENBAUM; WETHERALL, 2010).

Com o surgimento de outros meios de transmissão, houveram problemas com os protocolos da ARPANET, forçando então a criação de um novo modelo de arquitetura de referência, conhecido como TCP/IP (TANENBAUM; WETHERALL, 2010).

O modelo de referência TCP/IP possui quatro camadas, sendo essas: Aplicação, Transporte, Inter-rede e Enlace (TANENBAUM; WETHERALL, 2010).

A Figura 2.2 ilustra resumidamente como funciona o modelo TCP/IP. A seguir é explicado, de maneira resumida, cada camada do modelo TCP/IP, tal como as características da camada e alguns protocolos.

Camada de Aplicação

A camada de Aplicação é responsável por realizar a comunicação entre os aplicativos e a camada de transporte, ou seja, nela estão contidas todos os protocolos de alto nível, *File Transfer Protocol* (FTP), *Simple Mail Transfer Protocol* (SMTP), *Hypertext Transfer Protocol* (HTTP), *Domain Name System* (DNS), Telnet e outros.

Camada de Transporte

A camada de Transporte tem a finalidade de gerenciar o transporte dos dados. Se necessário é nessa camada que os pacotes enviados pela camada de aplicação são fragmentados, segmentados e enviados para a camada abaixo. No destino o processo inverso é realizado. (TANENBAUM; WETHERALL, 2010).

Nessa camada os principais protocolos são o TCP e o *User Datagram Protocol* (UDP). O primeiro garante a entrega dos pacotes na ordem e sem erros, ou seja, é um protocolo confiável (TANENBAUM; WETHERALL, 2010), enquanto o segundo não garante que os pacotes irão de fato chegar ao destino, muito menos a ordem correta de chegada.

O protocolo TCP, diferentemente do UDP possui estados. São 11 estados que representam as etapas necessárias para estabelecer e encerrar conexões, os estados podem ser representados por uma máquina de estados finitos. Para cada estado existem ações que acontecem de acordo com algum evento válido, e caso o evento não seja válido, um erro é reportado.

Camada de Inter-Rede

A camada Inter-Rede, é a camada responsável por garantir que os pacotes recebido das camadas adjacentes possam trafegar até o destino correto. O processo realizado nesta camada é similar ao sistema de correio, no qual uma carta pode ser enviada para outro país, passando por vários centros de distribuição até chegar ao destino (TANENBAUM; WETHERALL, 2010).

Alguns protocolos existente nesta camada são o IP, *Internet Control Message Protocol* (ICMP), *Address Resolution Protocol* (ARP) e o *Reverse Address Resolution Protocol* (RARP).

Camada Enlace

A camada Enlace é a representação das camadas de enlace e física do modelo ISO/OSI, e é responsável por transmitir bits e garantir que o outro lado receba esses bits (TANENBAUM; WETHERALL, 2010). Os padrões mais comuns desta camada são o Ethernet (IEEE 802.3) e o *Wireless Fidelity* (WiFi) (IEEE 802.1).

A Figura 2.2 mostra como a pilha de protocolos funciona. Por exemplo, ao acessar um *site* (conexão HTTP), o cliente faz uma requisição para o servidor, o navegador cria um

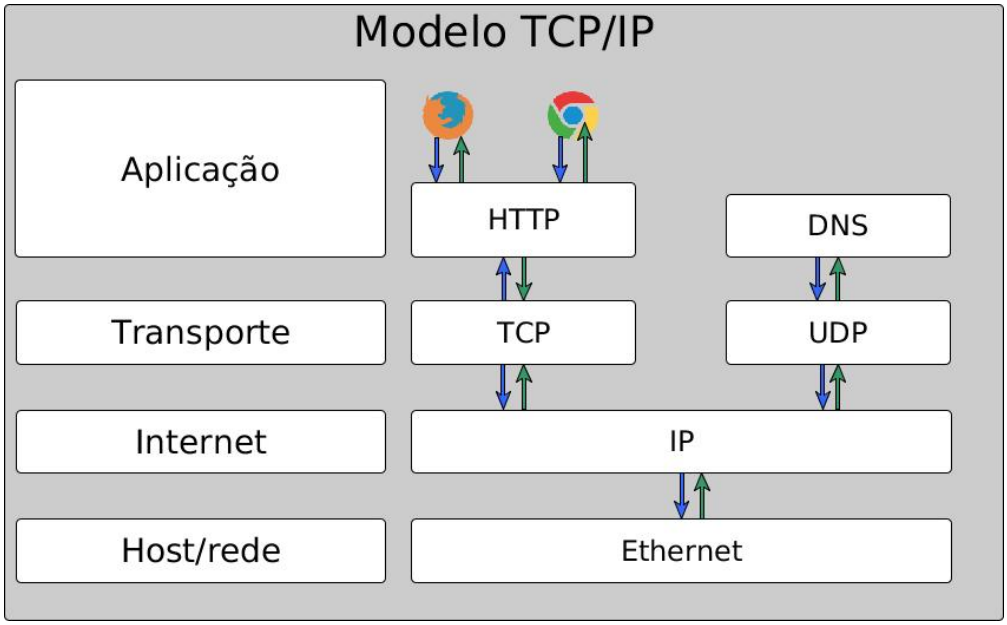

Figura 2.2. Camadas e alguns protocolos do modelo TCP/IP.

pacote, o pacote por sua vez possui a informação (conteúdo) e um cabeçalho, o cabeçalho nesta camada contém dados do tipo: método (*GET* ou *POST*), (*User-Agent*), *cookies* e outras informações. Sendo assim, o mesmo é enviado para a camada de Transporte. Assim que o pacote entra na camada seguinte, é adicionado um novo cabeçalho, que possui informações para criar ou manter conexões cliente-servidor. A próxima camada, recebe o pacote da camada Transporte e adiciona um novo cabeçalho ao pacote, o pacote nesta camada recebe o nome de datagrama IP, o qual possui cabeçalho que é responsável por garantir que os pacotes possam ser roteados e chegar ao destino correto. Na última camada (Rede/*Host*) o pacote (ou datagrama IP) recebe um novo cabeçalho, e neste exemplo, o pacote passaria a se chamar quadro Ethernet, este possui campos para endereços físicos. Todo esse processo chama-se empacotamento e desempacotamento.

A Seção a seguir apresenta o conceito de pacotes nas camadas de Transporte e Internet para melhor entendimento do trabalho.

2.1.2. Pacotes

Quando há a troca de informação entre *hosts*, os dados enviados podem ser subdivididos em pacotes. E estes pacotes são chamados de datagrama IP ou pacotes IP (STALLINGS, 2007). Os pacotes possuem um cabeçalho e a informação ou *payload*, este cabeçalho é diferente para cada camada do modelo TCP/IP (STALLINGS, 2007). Por conter um cabeçalho, é possível rotear os pacotes por inúmeros roteadores até que o mesmo chegue ao seu destino (STALLINGS, 2007).

O conceito de pacotes neste trabalho é importante, pois o *firewall* será testado utilizando pacotes, sendo assim, saber diferenciar os tipos de pacotes é relevante.

2.1.3. Segurança de redes

O uso da Internet em organizações passou a ser fundamental para alavancar os processos de produção, e a tendência é cada vez mais pessoas e empresas dependerem da Internet (MORAES, 2010), pois serviços feitos na vida real (operações bancárias, compras/vendas e outros) passaram a ser realizados também no mundo virtual.

A segurança, no contexto de redes, é garantir que informações não sejam acessadas ou modificadas por pessoas não autorizadas (TANENBAUM; WETHERALL, 2010).

No modelo TCP/IP cada camada contribui para a segurança das redes (TANENBAUM; WETHERALL, 2010). Na camada física é possível garantir segurança mantendo o cabeamento em tubulações lacradas com gás, na camada de enlace é possível usar criptografia de enlace, na camada de rede pode ser utilizado *firewall* com a finalidade de realizar filtragem de pacotes, na camada de transporte pode-se criptografar conexões e, na camada de aplicação realizar autenticação de usuários (TANENBAUM; WETHERALL, 2010).

A Seção a seguir discorre acerca de ataques conhecidos, que em alguns casos são problemas no qual o *firewall* terá que lidar.

2.1.4. Ataques

O *firewall* atua realizando filtragem de pacotes na camada de rede e na camada de transporte do modelo TCP/IP (NAKAMURA; GEUS, 2010). Sendo assim, esta seção tem por finalidade expor alguns ataques conhecidos que utilizam essas camadas.

Ataques na camada de rede são pacotes ou série de pacotes que alteram o cabeçalho na camada de rede, com a intenção de explorar vulnerabilidades (RASH, 2007). O *Distributed Denial of Service* (DDoS) é um ataque cujo o objetivo é inundar *hosts* até que a comunicação seja interrompida, é um ataque difícil de se combater e o fato de ser distribuído potencializa o mesmo (RASH, 2007).

Os ataques na camada de transporte estão na sua maioria classificados em ataques de reconhecimento. Um exemplo de ataque é o escaneamento de porta, este consiste em interrogar um ou mais *hosts* com a finalidade de saber quais serviços estão acessíveis. O escaneamento de portas por si só não é um ataque, mas é um passo inicial muito importante para quem deseja atacar (RASH, 2007).

Estes são somente dois exemplos de ataques que o *firewall* terá que lidar, a configuração correta do *firewall* é importante para prevenir que ataques ocorram na rede.

2.2. Firewall

Bishop (2005) define *firewall* como: “Um dispositivo que realiza a mediação para o acesso a uma rede, permitindo ou não determinados tipos de acessos baseado na política de segurança

configurada".

Os *firewalls* são similares a castelos medievais, no qual há um poço profundo ao redor do castelo e a única forma de entrar ou sair do mesmo, é passando por uma ponte levadiça (TANENBAUM; WETHERALL, 2010).

Scambray et al. (2000) faz a seguinte afirmação: "Um *firewall* bem projetado, bem configurado e bem mantido é praticamente impenetrável".

O *firewall* é a primeira linha de defesa em redes de computadores, o tráfego de rede obrigatoriamente tem que passar através do *firewall*. Sendo assim, é possível realizar filtragem de pacotes, selecionando quais pacotes podem trafegar pela rede e quais pacotes devem ser descartados.

Segundo Moraes (2010) as principais funções do *firewall* são:

- Estabelecer um perímetro de segurança;
- Separar as redes e controlar os acessos;
- Ser um elemento central de controle e aplicação de políticas de segurança;
- Proteger sistemas vulneráveis na rede;
- Aumentar a privacidade.
- Gerar *logs* e estatísticas do uso da rede e acessos indevidos.

Atualmente, *firewalls* possuem diversas tecnologias agregadas e pode-se citar a filtragem de pacotes, *Network Address Translation* (NAT) e *Virtual Private Network* (VPN). Ao utilizá-lo como filtro, o *firewall* analisa o cabeçalho dos pacotes e baseado em regras descarta ou realiza o roteamento do pacote. O filtro de pacotes atua na camada de rede do modelo TCP/IP apresentado na Seção 2.1.1, a Seção 2.2.2 aborda com mais detalhes sobre tecnologias de *firewall*.

Existem também projetos ou arquiteturas de *firewall*, que são estratégias de padrão de configuração de *firewall*. A Seção 2.2.1 entra em detalhes deste assunto.

2.2.1. Projetos de Firewall

Existem vários projetos de *firewall* que podem ser utilizados independentemente do tamanho da rede (ARTYMIAK, 2003). Os projetos de *firewall* mais comuns são: *Screened Host*, *Screened LAN*, *Bastion Host* e *Demilitarized Zone* (DMZ).

Screened Host

A ideia principal deste projeto é posicionar o *firewall* entre a Internet e o *host* a ser protegido, e configurá-lo de tal modo que o *host* nunca seja servidor, ou seja, o *host* pode acessar servidores na Internet, mas alguém da Internet não pode acessar o *host*, esse projeto é de simples implementação (ARTYMIAK, 2003). A Figura 2.3 ilustra este projeto de *firewall*.

Figura 2.3. *Firewall* protegendo um *host* de ataques externos.

Screened LAN

Quando existe a necessidade de conectar dois ou mais computadores em LAN, é possível utilizar o mesmo princípio do *Screened Host* entretanto, conectando o *firewall* em *switch* ou *hub*. Este projeto de *firewall* pode ter problemas de vulnerabilidade pois não protege a LAN de ataques internos, sendo assim é possível combinar *Screened Host* com o *Screened* LAN (ARTYMIAK, 2003). A Figura 2.4 demonstra a configuração de um projeto de *firewall Screened* LAN.

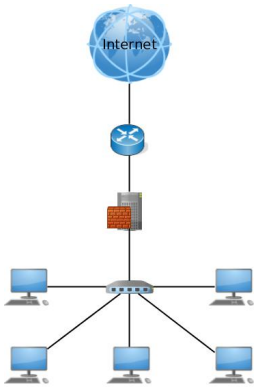

Figura 2.4. *Firewall* protegendo uma *LAN* de ataques externos.

Bastion Host

O projeto de *firewall Bastion host* é similar ao *Screened host*, contudo, a diferença está na configuração do *firewall*, permitindo que o *host* possa fornecer serviços para a Internet, contudo de forma dosada, ou seja, se for um servidor HTTP, só deve ser liberado serviços vitais para o funcionamento e gerenciamento deste tipo de servidor e nada mais (ARTYMIAK, 2003).

DMZ

DMZ é utilizada quando há necessidade de fornecer serviços para a Internet e ao mesmo tempo ter *hosts* que não podem ser servidores. Uma DMZ geralmente é uma LAN de servidores que podem ser por exemplo, HTTP, SMTP, FTP e entre outros. Sendo assim é possível garantir a segurança de outros *hosts* utilizando uma combinação com *Screened* LAN (ARTYMIAK, 2003).

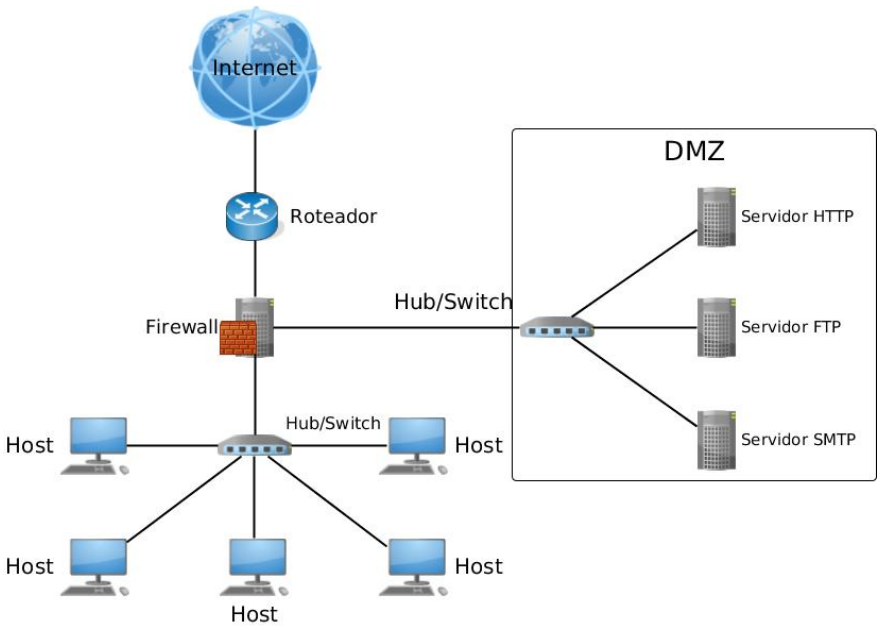

Figura 2.5. *Firewall* protegendo uma *LAN* de ataques externos, enquanto permite que servidores possam prover serviços.

2.2.2. Tecnologias de Firewall

Atualmente os *firewalls* englobam várias tecnologias, dentre essas tecnologias pode-se citar filtro de pacotes, NAT, *proxy*, VPN e outros. A ênfase deste trabalho está na filtragem de pacotes.

Os filtros de pacotes são responsáveis por analisar o cabeçalho dos pacotes, e a partir das regras contidas no *firewall* tomar decisões. As decisões são de descartar o pacote ou aceitar o pacote (NAKAMURA; GEUS, 2010).

A Figura 2.6 representa um cenário no qual o *host* 1 só deve realizar conexões HTTP na porta 80 com o *host* 2. A Tabela 2.1 representa as regras contidas no *firewall*.

Figura 2.6. Cenário mínimo contendo dois *hosts*.

Tabela 2.1. Regras no *firewall* - permitindo somente conexão HTTP

Regra	IP de origem	IP de dest.	Protocolo	Porta origem	Porta dest.	Action
1	10.0.0.2	192.168.0.2	TCP	*	80	ACCEPT
2	192.168.0.2	10.0.0.2	TCP	80	*	ACCEPT
3	*	*	*	*	*	DROP

É válido ressaltar que dependendo da tecnologia do *firewall*, a ordem das regras inseridas tem importância, o *firewall* vai analisar da primeira até a última. A regra número 3 na Tabela 2.1 significa que qualquer pacote que não tenha coincidido com as regras anteriores, será descartado pelo *firewall*.

2.3. Iptables

O `Iptables` é um *firewall* de pacotes e desde 2001 está contido no Kernel do Linux. Através dos anos, o `Iptables` evoluiu ao ponto de ter tantas funcionalidades quanto um *firewall* comercial proprietário (RASH, 2007). O `Iptables` é constituído por tabelas, no qual as 3 principais são: *filter*, NAT e mangle (RASH, 2007).

2.3.1. Tabelas

Cada tabela representa uma funcionalidade do `Iptables`. A tabela *filter* é responsável por realizar a filtragem de pacotes, a NAT realiza conversões de endereços reservados para endereços que possam ser roteados na rede externa. Por fim, a *mangle* é a tabela que contém regras especializadas que alteram os dados específicos dos pacotes, tal como *Type of Service* (TOS), sendo assim é possível melhorar a qualidade de tráfego de rede(RASH, 2007).

Filter

A tabela *filter* possui 3 *chains*, que significa a qual grupo de pacotes as regras serão aplicadas, podendo ser: *Input*, *Output* e *Forward*. A *chain Input* é analisada quando os pacotes tem como endereço de destino o próprio *firewall*. A *chain Output* é utilizada quando o pacote tem como endereço de origem o *firewall*, ou seja, quando o *firewall* envia algum pacote. Por fim a *chain Forward* é analisada quando o pacote será roteado pelo *firewall* (RASH, 2007).

NAT

A tabela NAT, assim como a *filter*, possui 3 *chains*. A *chain Prerouting* realiza alteração no endereço de destino dos pacotes antes de serem roteados. A *chain Output* é similar a *chain Prerouting* contudo é apenas para pacotes originados do próprio *firewall*. A *chain Postrouting* realiza alterações nos endereços de origem após serem roteados (SUEHRING; ZIEGLER, 2005).

Mangle

A tabela mangle possui 5 *chains*. Suas *chains* são a combinação das *chains* das tabelas anteriores, com a diferença de realizar alterações no campo de TOS e *Time to live* (TTL) dos pacotes, podendo melhorar a qualidade do tráfego de rede (SUEHRING; ZIEGLER, 2005).

2.3.2. Política e Regras

No `Iptables` é possível definir políticas nas *chains*. Existem duas opções de política, podendo ser *ACCEPT* ou *DROP*. A política é utilizada quando um pacote é analisado e não coincide com nenhuma regra no *firewall*, neste caso se a política for *DROP* o pacote será descartado, do contrário o pacote será processado.

As regras são constituídas por um conjunto de variáveis. Dentre essas variáveis pode-se citar o IP de origem e destino, porta de origem e destino, protocolo utilizado e entre outros (MORAES, 2010). No exemplo ilustrado pela Figura 2.6, é possível notar que a última regra da Tabela 2.1 indica que a política adotada é *DROP*.

2.4. Mininet

Neste trabalho foi utilizado o `Mininet` (LANTZ et al., 2010) para simular as redes de computadores. Assim, as configurações a respeito da criação dos nós, enlaces de rede e configurações serão concretizadas via `Mininet`.

A ferramenta `Mininet` foi escolhida pois possibilita a criação de redes virtuais com o objetivo de simular um ambiente real. A mesma tem grande importância para realização de testes e experimentos, pois dispensa a necessidade de possuir ou manusear equipamentos de rede, além da facilidade de seu uso.

É recomendável que o `Mininet` seja executado em uma máquina virtual que utiliza o sistema operacional Ubuntu (UBUNTU, 2019), o mesmo possuí uma *Application Programming Interface* (API)(MININET, 2019) em Python (ROSSUM, 1995) que permite criar cenários de redes personalizados. Esta API fornece interfaces para criar, configurar e remover *hosts*, enlaces e outros elementos utilizados em redes de computadores. Esta `API` é utilizada na implementação do Núcleo de gerenciamento da ferramenta de testes de *firewall*.

2.5. TCPDump

O `TCPDump` é uma ferramenta que realiza a captura de pacotes que trafegam através de uma interface de rede. Analogamente à ferramenta Wireshark, a qual é é amplamente utilizado na área de redes de computadores (TCPDUMP, 2019), o `TCPDump` fornece filtros de análise de pacotes, como capturar pacotes destinados à uma dada porta, ou apenas pacotes destinados à determinada interface de rede. A preferência pelo uso do `TCPDump` se dá pelo suporte em realizar a captura de pacotes e aplicar filtros através da linha de comando.

Através dos pacotes de rede capturados pela ferramenta, é possível observar o endereço de destino e origem do mesmo, horário e data do recebimento e envio do pacote além de *flags* no caso de pacotes TCP, como a *flag* ACK.

Devido a informações geradas por esta ferramenta, a mesma é bastante utilizada para depuração em redes problemáticas, uma vez que é possível obter informações do sucesso ou falha de envio de um pacote a um *host* qualquer.

Um exemplo de uso do `TCPDump` é testar a conexão entre duas máquinas que se conectam através de roteadores com um *firewall* configurado. É possível monitorar todo o tráfego de rede utilizando o `TCPDump`, e com a análise dos registros, identificar possíveis falhas na rede.

2.6. Trabalhos Relacionados

O texto a seguir é referente a trabalhos relacionados, que implementa, uma ferramenta para análise de regras de *firewall* porém com uma abordagem diferente.

2.6.1. *A Tool for Automated iptables Firewall Analysis*

Marmorstein e Kearns (2005) contextualizam que os *firewalls* se tornaram a base de maioria das políticas de segurança implementadas em redes de computadores. Contudo a sua complexidade está diretamente relacionada ao seu poder de segurança, no qual um erro sútil pode ser muito difícil de detectar e abrir portas para invasores ou ataques de negação de serviço.

Sendo assim, Marmorstein e Kearns (2005) apresentam uma ferramenta para análise de regras de *firewall*, especificamente para o Iptables, denominada ITval. O funcionamento dessa ferramenta é realizado a partir da saída do comando **iptables -L -n** e através de um arquivo de perguntas configurado previamente, a ferramenta retorna quais pacotes serão aceitos pelo *firewall*.

Referente a implementação da ferramenta, Marmorstein e Kearns (2005) discutem a respeito dos arquivos de consultas e da metodologia adotada. O arquivo de perguntas é configurado utilizando linguagem de consulta, com este arquivo o usuário pode realizar

perguntas do tipo: “Quais máquinas podem ser alcançadas com SSH?”. Este arquivo de consulta consiste de um conjunto de definições para grupos e serviços, e logo em seguida por instruções de consulta. Já para a implementação do mecanismo do `ITval`, é implementado um diagrama de decisão multidirecional, que pode ser representado por um digrafo acíclico. Cada nível não terminal deste diagrama representa um atributo de um pacote, sendo assim, através de decisões este diagrama é preenchido até chegar ao nível zero, que será um estado terminal que representará se o pacote é aceito ou não pelo *firewall*.

Por fim, Marmorstein e Kearns (2005) concluem, que a ferramenta é capaz de auxiliar administradores de sistemas a verificar de forma rápida e fácil se o *firewall* está funcionando corretamente, e que a vantagem de se utilizar a linguagem por consulta facilita na modificação para outros sistemas de *firewall*. Marmorstein e Kearns (2005) concluem também que a ferramenta pode apresentar uma saída confusa para consultas que geram muitos resultados e propõe uma interface interativa.

Este trabalho foi feito com a finalidade de realizar testes em *firewall* utilizando uma abordagem diferente da apresentada por Marmorstein e Kearns (2005), utilizando a construção de pacotes e testando em redes virtualizadas, gerados através de uma interface simples e intuitiva, ao invés de usar diagramas de decisão e linguagem de consulta.

O trabalho de (MARMORSTEIN; KEARNS, 2005) enfatizou a necessidade de uma ferramenta que ajude administradores de sistemas a testar *firewalls*. A abordagem adotada por Marmorstein e Kearns (2005) é válida, contudo ainda precisa de mais funcionalidades, como por exemplo uma interface interativa e a capacidade de suportar outros *firewalls* que não seja o `Iptables`.

2.6.2. *Systematic Literature Review on Usability of Firewall Configuration*

O trabalho de Voronkov et al. (2017) cita que os *firewalls* consistem em *software/hardware* e foram projetados para impedir acessos não autorizados a redes ou dispositivos, são empregados para regular o tráfego da rede entre computadores ou outros dispositivos de rede, aceitando ou descartando pacotes de acordo com a política adotada.

Voronkov et al. (2017) contextualizou que os administradores de redes normalmente utilizam uma abordagem de fases para projetar e implementar o *firewall*. As fases são:

1. Planejar;
2. Configurar;
3. Implementar e testar;
4. Implantar;
5. Gerenciar.

Então Voronkov et al. (2017) foca no processo de configuração correta de *firewalls*.

Os autores citam também que a fase de configuração é um dos fatores mais importantes para a segurança de redes utilizando *firewall*. E então citam um estudo onde revelou-se que todos os conjuntos de regras de *firewall* onde há 12 ou mais regras, possuem erros de configuração.

Voronkov et al. (2017) enfatizam a dificuldade em encontrar artigos a respeito de detecção de erros em *firewalls*, onde a ferramenta transmita de maneira eficaz informações geradas a respeito de configurações corretas em *firewalls* para o usuário.

O trabalho de Voronkov et al. (2017) apresentou a dificuldade em configurar *firewalls* de maneira eficaz, assim como a necessidade de uma ferramenta que aponte de maneira compreensível os erros de configurações em *firewalls*. A implementação deste trabalho foi feita pensando na usabilidade, justamente para identificar com facilidade os erros cometidos durante a configuração do *firewall*, simulando serviços de rede e então testando o *firewall*, aliado a uma interface *web*, provendo usabilidade para quem utilizar a ferramenta.

2.7. Considerações Finais

Os conceitos apresentados neste capítulo são importantes para o entendimento e como a ferramenta foi desenvolvida. Além disso, os trabalhos relacionados enfatizam que, apesar de existirem ferramentas que realizam testes em *firewalls*, a ferramenta desenvolvida nesse trabalho utiliza uma abordagem diferente dos demais trabalhos relacionados, que é simular o cenário de rede, tal como os serviços de rede e submeter o *firewall* a testes.

Capítulo 3

Metodologia

Neste capítulo é apresentada a abordagem utilizada para implementar a ferramenta para realizar testes de *firewall*. A ferramenta possibilita criar cenários de redes virtuais, contendo máquinas e roteadores, os quais são especificadas pelo usuário através de uma interface. Em seguida, as regras de *firewall* são testadas utilizando tráfego de pacotes e por fim, o caminho percorrido pelos pacotes na rede é analisado para se chegar a conclusão se os testes realizados são bem sucedidos ou não, o que ajuda a mostrar para os administradores de redes se as regras utilizadas em *firewalls* são ou não eficazes.

A Figura 3.1 ilustra a sequência de passos para realizar a análise de regras de *firewalls*. Para tal, são utilizados tecnologias e ferramentas já existentes, combinadas com a implementação de um Núcleo de Gerenciamento, cujo objetivo é integrar e gerenciar todas as ferramentas utilizadas, assim como realizar os testes. Estes elementos são abordados ao longo deste capítulo.

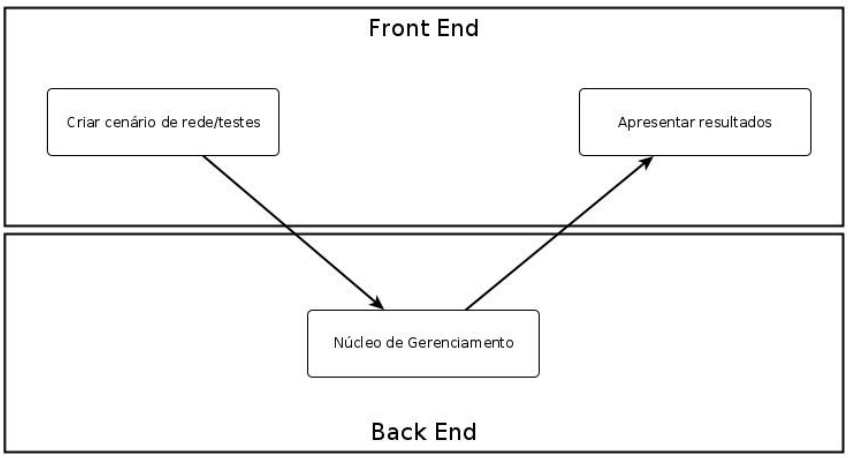

Figura 3.1. Etapas para realizar a análise de regras de *firewall*.

A Figura 3.1 representa, de maneira genérica, as etapas para realizar a análise de regras de *firewalls*. Tais etapas consistem em um *Front End* responsável por obter os dados inseridos pelo usuário, que são a topologia da rede, as regras de *firewalls* e os testes as serem realizados, e um *Back End* responsável por processar as informações, virtualizar cenários de rede, testar as regras de *firewalls* e apresentar os resultados dos testes para os administradores de *firewall*.

3.1. *Front End*

O *front end* é responsável por receber os dados de entrada e apresentar os resultados. A entrada de dados é feita através de uma interface *web*, na qual os administradores de *firewalls* poderão adicionar *hosts*, configurar *firewalls*, enlace de redes, configurar interfaces de redes e criar testes.

A interface contém um menu que oferece opções para adicionar nós que representam computadores ou roteadores, cada nó possui seus próprios atributos que são:

- Tipo: Que pode ser *host* (equipamento comum - cliente/servidor - que não é um roteador) ou *router*;
- DNS: Utilizado para atribuir endereço do servidor DNS nos nós;
- Regras de *firewall*: Campo de texto para inserir as regras de *firewall* a serem implantadas posteriormente nos nós, são essas regras que serão testadas pela ferramentas através dos testes propostos pelos administradores de *firewalls*.

Ao criar um *host* ou *roteador*, é possível adicionar uma ou mais interface de rede, cada interface de rede contém os seguintes atributos:

- IP: Atributo para definir endereço IP da interface de rede.
- Máscara: Necessário para realizar a configuração da interface de rede.
- *Gateway*: Utilizado para definir endereço de rota padrão.

Além do menu para criar nós e interface de redes, existe uma área para visualizar o cenário de rede a ser criado. Ao criar um nó, o mesmo será apresentado graficamente, sendo possível arrastá-los dentro da área para visualizar melhor o cenário de rede criado.

Também é possível criar *links* (enlaces) de rede, tal como cabos que conectam os *hots*/roteadores, para isso é necessário informar a origem e o destino do *link*, e assim como acontece com os nós, é apresentado o *link* graficamente.

A interface *web* conta também com uma lista de casos de testes a serem realizados. Ou seja, no contexto deste trabalho, um experimento com o *firewall* possui vários testes, para um experimento ter sucesso é necessário que todos os testes sejam realizados com êxito, caso contrário o experimento terá falhado, o que indica que o *firewall* possui algum problema de configuração em suas regras. Esses casos de testes são fornecidos pelo usuário com a finalidade de testar o *firewall*. Para isso, o usuário terá que simular requisições, ou seja, terá

que informar os endereços e portas de origem e destino, tal como o protocolo de comunicação utilizado. Além das informações citadas anteriormente, é necessário indicar se os pacotes de rede emitidos pelo dispositivo de origem devem chegar ou não chegar até o dispositivo de destino. Esta última informação será utilizada para definir o resultado do teste.

Ao iniciar o experimento com o *firewall*, todas as informações inseridas na interface serão escritas em arquivo no formato *JavaScript Object Notation* (JSON). O arquivo JSON é processado posteriormente pelo *Back end*.

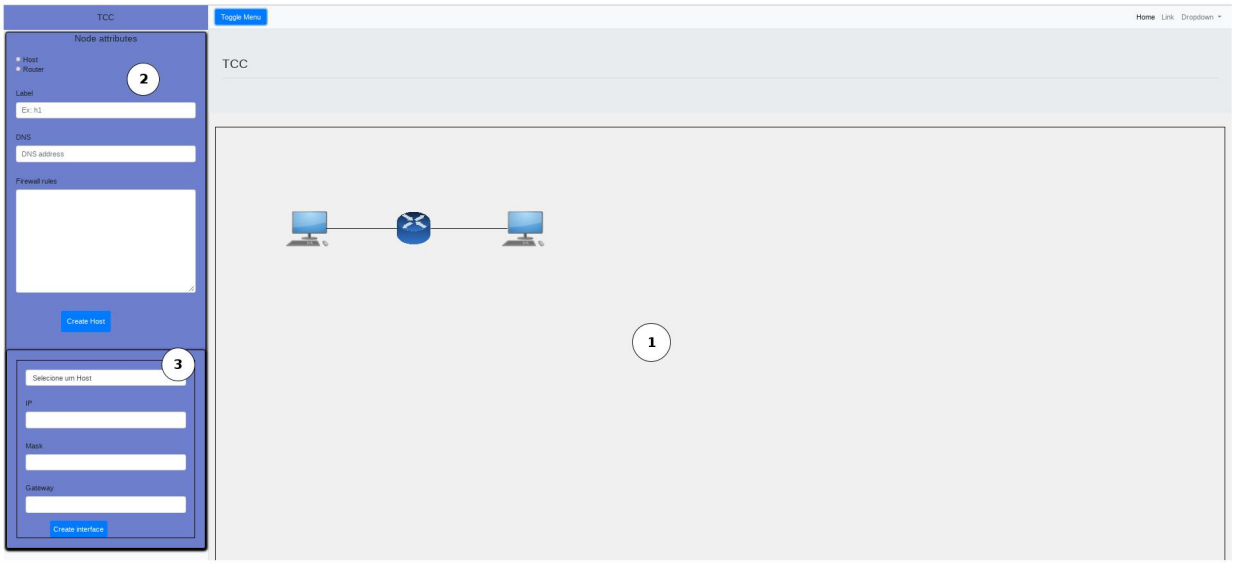

Figura 3.2. Protótipo de interface.

A Figura 3.2 apresenta a concepção da interface *web*. A área marcada com o número 1 representa a parte gráfica da interface, onde os *hosts*, roteadores são apresentados, é possível arrastá-los dentro dessa área. A área marcada com o número 2 representa os atributos de um nó. Nela é possível definir entre *host* ou *router*, atribuir um endereço DNS e as regras de *firewall*. Por fim, a área marcada com o número 3 representa a interface de rede que cada nó pode receber, vale ressaltar que um nó pode receber mais de uma interface de rede.

3.2. Back End

Uma vez que já tenha o arquivo JSON preenchido com os dados inseridos no *Front end*, o próximo passo é processar os dados obtidos do arquivo JSON e com isso virtualizar o cenário de rede, realizar os experimentos de *firewall*, análise dos testes e dar o resultado dos experimentos. Este conjunto de processos inseridos no *Back-end* é denominado de Núcleo de Gerenciamento.

A Figura 3.3 aborda e relaciona cada processo do experimento de *firewall* individualmente. A interface *web* fornece informações para criar nós, *links*, configuração de rede e *firewall* e criar os casos de testes. Estes processos são gerenciados pelo Núcleo de

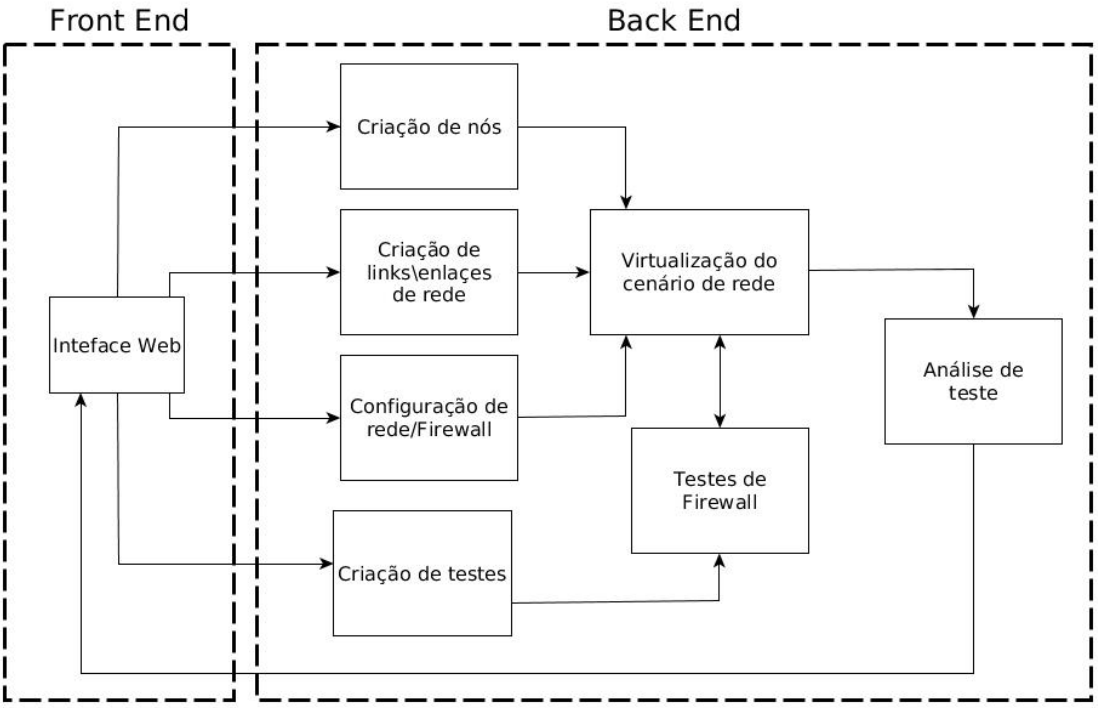

Figura 3.3. Fluxo de processamento dos testes de *firewall*.

Gerenciamento, que também é chamado de *Back End*.

3.2.1. Núcleo de Gerenciamento

O Núcleo de Gerenciamento é implementado em Python (ROSSUM, 1995) e consiste de um conjunto de programas que tem por finalidade gerenciar o cenário de rede no `Mininet` (LANTZ et al., 2010). O Núcleo têm funcionamento sequencial, ou seja, a partir dos dados de entrada fornecidos no *Front end*, o Núcleo de Gerenciamento, situado no *Back end*, realiza a leitura desses dados, armazena-os em memória utilizando estruturas de dados, virtualiza o cenário de rede, realiza os testes, analisa os registros e retorna os resultados obtidos dos experimentos.

3.2.2. Entrada de dados

Os dados de entrada inseridos pelo usuário no *Front end* chegam até o *Back end* no formato JSON. Esse formato foi escolhido, pois é de fácil interpretação para humanos e máquinas, além disso, o JSON representa os dados como objeto, facilitando a leitura e armazenamento. O Núcleo de Gerenciamento realiza a leitura do arquivo de entrada e armazena estes dados como objetos em estruturas que serão utilizadas no decorrer da execução. Foram implementadas algumas classes abstraindo os elementos utilizados pelo Núcleo de Gerenciamento, pensando em tornar a implementação genérica e, assim, possibilitando simular diferentes cenários de redes. Tais classes são explicadas com mais detalhes a seguir.

Criação de nós

Esta etapa do processo é responsável por ler o arquivo JSON, criar os nós e armazená-los em uma estrutura de lista que é utilizada para virtualizar e manipular o cenário de rede. As classes atribuídas a esta etapa são `Host` e `Interface`, estas classes tem por objetivo abstrair o conceito de *host*, contendo atributos necessários para criá-los no `Mininet`, os atributos da classe *host* são:

- **type**: Este atributo é usado para especificar o tipo do nó, podendo ser *host* ou roteador.
- **label**: É utilizado para dar nome ao nó, esse atributo é importante para diferenciar cada nó individualmente e manipulá-lo dentro da estrutura do `Mininet`.
- **dns**: Atributo responsável por definir o endereço do servidor DNS do nó.
- **fwCommand**: Uma lista contendo comandos de *firewall* a serem adicionados no *firewall* do nó. Esses comados são regras reais de *firewall*, mas especificamente do `Iptables`.
- **iface**: É uma lista de interface de redes, a interface de rede possui uma classe própria que será explicado a seguir.

A classe `Interface` foi implementada para representar interfaces de redes, a finalidade dessa classe é simular placas de rede, na qual estas placas serão atribuídas a determinados nós, esta classe possui os seguintes atributos:

- **ip**: Atributo para definir um endereço de IP para a interface de rede.
- **mask**: Utilizado para configurar a rede.
- **gateway**: Endereço IP do roteador/*gateway* padrão.
- **name**: Identificar cada interface, o nome da interface é sempre o nome do nó seguido de um hífen e o nome da interface.

Criação de enlaces

Para representar os enlaces entre *hosts*/roteadores e *switchs* temos a implementação da classe `Link`, essa classe possui três atributos, que são:

- **label**: O atributo label tem por função nomear as arestas e esse nome é constituído pelos nomes dos *hosts* que estão conectados por esse *link*.
- **source**: Nome do nó de origem.
- **dest**: Nome do nó de destino.

Vale ressaltar que a conexão é bi-direcional, ou seja, não há problemas em inverter os atributos de origem e destino.

Configuração de rede/*firewall*

Essa classe foi desenvolvida com a finalidade de organizar e facilitar a inserção de comandos em *hosts*/roteadores e interfaces de rede, para instanciar essa classe é necessário passar uma

interface por parâmetro, com isso é possível chamar métodos com comandos Linux específicos para a interface que foi passada por parâmetro. Os métodos dessa classe são:

- **addGateway**: Comando Linux para adicionar rotas em um determinado *host*. O comando utilizado para realizar essa operação é: *route add default gw* <atributo *gateway* da interface>. Um exemplo de configuração de *gateway* é: *route add default gw* 192.168.0.1
- **configMask**: Utiliza o *ifconfig* para atribuir máscara a interface de rede. O comando é: *ifconfig* <nome da interface de rede> *netmask* <valor da máscara>. Para demonstração, pode-se citar o seguinte exemplo: *ifconfig h1-eth0 netmask 255.255.255.0*.
- **configRouter**: Método responsável por habilitar o encaminhamento de pacotes nos roteadores, o comando que esse método executa é: *echo* 1 >/proc/sys/net/ipv4/ip_forward.
- **configIface**: Sempre que se faz necessário atribuir endereço de IP para interface de rede, é necessário utilizar esse método. O comando Linux executado é: *ifconfig* <nome da interface de rede> <ip a ser definido>. Exemplificando este comando, temos: *ifconfig h1-eth0 192.168.0.2*.
- **start_tcpdump**: Comando necessário para instanciar o `Tcpdump` com a finalidade de monitorar interface de rede, o comando é: *sudo tcpdump* –tt -n -i <nome da interface de rede> -w <nome do arquivo em que os registros serão salvos> *not arp*. Um exemplo aplicado deste comando é: *sudo tcpdump -tt -n -i h1-eth0 -w h1-eth0.log not arp*. A opção *-tt* tem por finalidade armazenar o tempo em *timestamp*, a opção *-n* tem por objetivo não resolver nomes de domínios e portas, *-i* serve para especificar uma interface de rede, -w definir arquivo onde os registros serão armazenados.
- **stop_tcpdump**: Comando utilizado para parar o `Tcpdump`, o comando Linux responsável por executar esse comando é: *killall tcpdump*
- **convertLogTcpdump**: Comando utilizado para converter os registros do `Tcpdump`, que são armazenados em binário, para extensão de texto. O comando que realiza essa ação é: *sudo tcpdump -tt -n -r h1-eth0.log > h1-eth0.txt*.

Criação de testes

Assim como foi feito para nós e *links*, é necessário armazenar os testes em estrutura de dados para usá-lo posteriormente no `Mininet`. Para isso foi desenvolvida a classe `Tests`, essa classe representa um caso de teste para o *firewall* definido pelo usuário utilizando o *Front End*. Os atributos dessa classe são:

- **sourceIP**: Atributo para especificar de onde (IP de origem) a comunicação será realizada.
- **destIP**: Tem por finalidade especificar para onde (IP de destino) a comunicação será realizada.

- **protocol**: Este atributo serve para definir qual protocolo (campo *protocol* do datagrama IP) será utilizado.
- **sourcePort**: Porta de origem, esse atributo pode ser vazio para representar valores aleatórios.
- **destPort**: Porta de destino, atributo necessário para realizar a comunicação.
- **expected**: Este atributo possui dois valores possíveis, *allow* (permitir) ou *deny* (negar), e é utilizado para determinar se as regras inseridas no *firewall* estão corretas ou não, para o caso de teste em específico. Caso seja definido como *allow*, significa que o usuário informou no *Front end* que é esperado que os pacotes de rede cheguem até o destino, caso esteja definido como *deny*, significa que o usuário informou que é esperado que os pacotes não cheguem até o destino.

3.2.3. Virtualização do cenário de rede

Com os dados obtidos da interface *web* e armazenados em memória com o auxílio de estruturas de dados (listas), é possível virtualizar o cenário de rede utilizando `API` do `Mininet`. Primeiramente os nós são adicionados, após os *hosts*/roteadores e *switchs* estarem virtualizados, os enlaces são feitos. Com o cenário virtualizado, então são feitas as configurações de endereçamento e *firewall*, ressaltando que o *firewall* pode ser configurado em qualquer *host*. Sendo assim, utilizando os comandos apresentados anteriormente, o *firewall* é configurado nos *hosts*, tal como o endereçamento. Feito isso, o cenário está pronto e configurado para a realização dos testes.

3.2.4. Testando o *firewall*

Com o cenário de rede funcional, tudo está pronto para realização dos testes, sendo assim, a próxima etapa consiste em abrir instâncias do `Tcpdump` em todas as interfaces de rede, ou seja, todas as interfaces serão monitoradas. O intuito desse passo é descobrir por quais interfaces de rede os pacotes passaram e consequentemente descobrir se os pacotes chegaram ou não ao destino. Esse passo é realizado utilizando os comandos definidos anteriormente, implementados utilizando a classe `Commands`.

Um dado experimento é realizado executando-se cada um dos testes definidos na lista de testes. Baseado nos atributos de cada teste, pacotes são gerados e transmitidos na rede simulando serviços, como: HTTP, DNS, *Secure Shell* (SSH), ICMP e outros, ou seja, *hosts* podem simular servidores enquanto outros simulam clientes. Para gerar os pacotes que simulam serviços TCP e UDP, foram implementados neste trabalho códigos (GUERRA, 2019c) em Python utilizando a biblioteca *Socket* (PYTHON, 2019). A motivação para desenvolver esses códigos em Python, vem da necessidade de simular serviços de rede que

sejam executados de forma simples, ou seja, não sejam gerados pacotes de rede desnecessários, facilitando então a análise do tráfego de rede.

Ao término da conexão, as instâncias do `Tcpdump` são finalizadas e com isso os registros de tráfego em cada interface de rede são gerados.

3.2.5. Análise dos registros

O último passo do Núcleo de Gerenciamento é extrair os resultados dos registros gerados pelo `Tcpdump` e posteriormente gerar respostas ao usuário que criou os casos de testes. O Núcleo retorna duas respostas. A primeira é informar se um teste foi aprovado ou reprovado, em outras palavras, é verificar se o *firewall* está configurado de acordo com requisitos esperados do usuário. A segunda resposta retorna por quais interfaces de rede os pacotes passaram, facilitando análise por parte do usuário, principalmente quando testes são reprovados, pois é possível verificar o caminho que os pacotes passaram e identificar possíveis falhas de implementação do *firewall*.

A primeira análise consiste em verificar quatro possibilidades, que são:

1. Os pacotes chegaram ao destino e deveriam chegar (teste aprovado);
2. Os pacotes chegaram ao destino e não deveriam chegar (teste reprovado);
3. Os pacotes não chegaram ao destino e deveriam chegar (teste reprovado);
4. Os pacotes não chegaram ao destino e não deveriam chegar (teste aprovado).

Ou seja, é necessário realizar duas etapas de verificação. A primeira etapa consiste em verificar se os pacotes chegaram ao *host* de destino, para isso, é necessário verificar se os pacotes enviados pelo *host* de origem estão presentes no registro gerado pelo `Tcpdump` na interface de rede do *host* de destino. A segunda etapa é verificar se os pacotes deveriam chegar ou não. Essa resposta a ferramenta produz de acordo com o comportamento dos pacotes na rede e no valor do atributo *expected* (*allow* ou *deny*) da classe *Tests*, apresentado anteriormente. Uma vez que essas duas etapas forem realizadas, é possível determinar se os testes dos experimentos foram aprovados ou reprovados.

A segunda análise consiste em verificar por quais interfaces de rede os pacotes passaram. Esse passo é feito analisando todos os registros gerados pelo `Tcpdump`. Para isso é realizado a leitura de cada registro do `Tcpdump` individualmente quebrando a tupla do registro e extraindo algumas informações importantes, tal como o horário que o pacote passou pela interface de rede, endereços de origem e destino, portas de origem e destino, protocolo utilizado e no caso do TCP, as *flags* utilizadas. Para obter o trajeto percorrido pelos pacotes, foi utilizado o horário em *timestamp* com precisão em nanossegundos dos pacotes que passaram pela interface, ressaltando que todos os *hosts* do `Mininet` tem o mesmo horário, dessa forma é possível ordena-los de forma crescente e construir o trajeto dos pacotes. Essa análise é importante, pois ajuda apresentar de forma mais clara o caminho que os pacotes

fizeram, possibilitando identificar equívocos cometidos pelos administradores de *firewalls* ao criar as regras de *firewall* que estão sendo testadas.

Essas duas análises são cruciais para o cumprimento do objetivo proposto desta ferramenta e são utilizadas para gerar a resposta de cada teste.

3.2.6. Apresentação do resultado de um caso de teste dentro de um experimento

Ao finalizar um teste, o Núcleo realizará o processamento dos registros do `Tcpdump`. De forma resumida, para analisar um caso de teste, considerando que o cenário de rede já esteja pronto, é necessário realizar a seguinte sequência de passos:

- Criar pacotes referente ao teste;
- Verificar se os pacotes chegaram ao *host* de destino;
- Verificar por quais interfaces de rede os pacotes passaram;
- Verificar se os pacotes deveriam ou não chegar ao *host* de destino;

Ao realizar essas verificações, é apresentado em interface o resultado do teste, se foi aprovado ou não, o caminho percorrido pelos pacotes e quanto tempo demorou cada caso de teste. Vale ressaltar que essa operação é repetida até que todos os testes do experimento sejam concluídos.

A Figura 3.4 apresenta de maneira interligada o Núcleo de Gerenciamento e as ferramentas utilizadas. O *Front End* é responsável por coletar os dados e criar um arquivo de entrada no formato JSON. Já no *Back End*, o Núcleo de Gerenciamento juntamente com a API, cria o cenário de rede virtualizado, configura os *firewalls*, instância o `Tcpdump` nas interfaces de redes e cria uma fila de casos de testes. Ao final de cada caso de teste o Núcleo de Gerenciamento realiza a análise dos registros coletados pelo `Tcpdump` e retorna se o caso de teste foi aprovado ou reprovado, tal como o caminho que os pacotes percorreram.

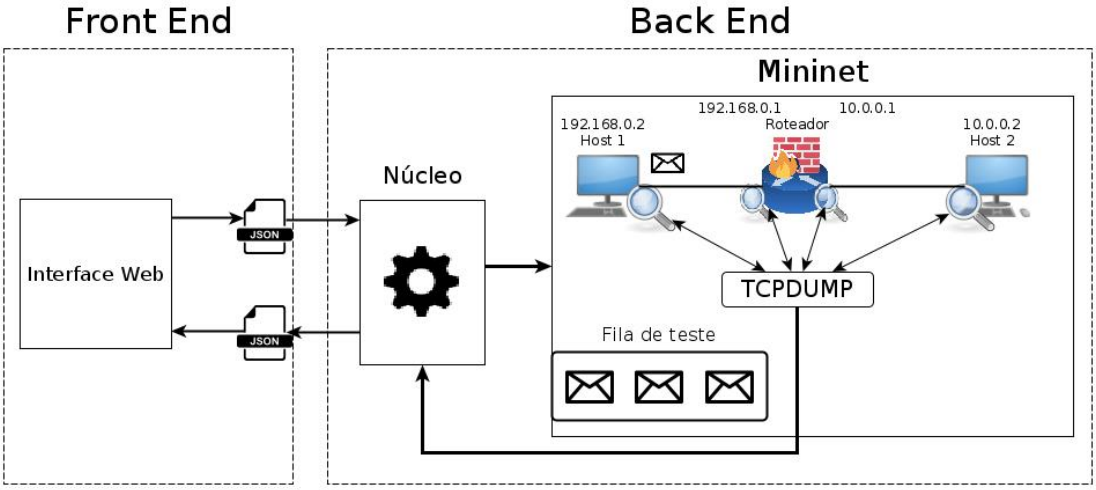

Figura 3.4. Utilização das ferramentas na arquitetura do sistema.

3.3. Considerações Finais

A abordagem apresentada evidenciou que é possível criar e virtualizar cenários de redes, configurar *firewalls* e simular serviços de redes. Com isso, é possível executar testes dentro destes cenários de redes virtualizados e verificar o comportamento dos pacotes na rede, tal como os efeitos das regras do *firewall*.

Com essa implementação é possível garantir que *firewalls* cumpram de fato com os objetivos propostos, garantindo segurança em redes de computadores. Vale ressaltar que a solução implementada, simular serviços de rede e subter *firewalls* a testes, é o diferencial do que existe na literatura.

Vale ressaltar que a interface *web* (GUERRA, 2019b) está em fase de desenvolvimento. É necessário desenvolver elementos onde possibilita a criação de *switchs*, enlaces e criação de casos de testes. Além disso, é necessário converter os dados obtidos em JSON, assim como receber um arquivo JSON e apresentar os resultados na interface *web*.

Capítulo 4

Experimentos e Resultados

Este capítulo apresenta os experimentos que foram realizados para certificar que a ferramenta proposta, de fato cumpre com o objetivo. Os experimentos elaborados visam virtualizar diferentes cenários de redes em intervalo de tempo aceitável, e comprovar que o seu funcionamento é similar ao funcionamento de redes e *firewalls* em ambientes reais.

É importante ressaltar que todos os experimentos foram realizados em um *desktop* com processador Intel Core I7-8700K 4.7Ghz (6 núcleos e 12 *threads*), contendo 32GB de memória RAM DDR4 a 3200Mhz, utilizando um SSD de 480GB para armazenamento.

4.1. Teste de carga: Quantidade de *hosts*

Por se tratar de uma ferramenta que simula cenários de redes, há necessidade de saber quais as limitações da ferramenta desenvolvida. Uma possível limitação é a quantidade de *hosts* que a ferramenta consegue virtualizar simultaneamente. Além da quantidade de *hosts*, é necessário verificar o tempo que os mesmos levam para serem virtualizados e configurados. A limitação na quantidade de *hosts*, assim como altos tempos para virtualizá-los, podem inviabilizar o uso da ferramenta.

A proposta do experimento, foi testar a ferramenta criando e configurando determinadas quantidades de *hosts* e verificar o tempo gasto para completar tal ação. Sendo assim, a ferramenta foi testada com 10, 50, 100, 500, 1000, 5000 e 8180 *hosts*. É importante destacar que a quantidade máxima de *hosts* suportados na máquina a qual os testes foram realizados foi 8180. A partir deste valor, a memória RAM da máquina utilizada no teste se tornou-se insuficiente.

Para realizar esse experimento, foi desenvolvido no presente trabalho um gerador de entradas (GUERRA, 2019a) para facilitar a execução dos testes. O gerador foi implementado para criar *hosts* com informações aleatórias, porém válidas, para que possam ser configurados

posteriormente no `Mininet`.

O gerador de *hosts* permite a configuração da quantidade de *hosts* a serem criados. É importante ressaltar que os *hosts* podem desempenhar o papel de roteador, além de um simples cliente conectado na rede. Além disso, o gerador tem a capacidade de atribuir até quatro interfaces de rede para cada máquina (*hosts*). Ademais, são gerados endereços IPs, máscara de rede, endereço do *gateway* e DNS válidos de forma aleatória, permitindo a configuração dos *hosts* que compõem a rede virtual no `Mininet`.

A partir disto, com o auxílio do gerador de *hosts*, foram realizados os experimentos com variação da quantidade de máquinas. Para cada variação na quantidade de *hosts*, o mesmo experimento foi repetido 5 vezes, para garantir tempos mais precisos. A Tabela 4.1 descreve a quantidade de máquinas presentes na rede, juntamente com os respectivos tempos médios em segundos de virtualização e configuração.

Tabela 4.1. Tempo médio em segundos para a ferramenta criar e configurar *hosts*.

Teste\Hosts	10	50	100	500	1000	5000	8180
Teste 1	0.30	1.39	2.57	13.89	36.52	220.67	443.26
Teste 2	0.29	1.96	2.88	12.60	29.46	210.47	364.33
Teste 3	0.30	1.27	2.37	13.25	29.92	210.47	392.00
Teste 4	0.30	1.27	2.40	13.35	26.03	219.65	382.09
Teste 5	0.31	1.77	2.38	13.19	27.84	228.15	396.60
Média(s)	**0.30**	**1.53**	**2.52**	**13.26**	**29.95**	**217.88**	**395.66**
Desvio padrão(s)	**0.01**	**0.32**	**0.22**	**0.46**	**3.98**	**7.52**	**29.35**

Os resultados obtidos após a realização deste experimento foram satisfatórios, pois os tempos obtidos para criar e configurar 500 *hosts* demorá-se pouco menos de 14 segundos e mesmo dobrando a quantidade de *hosts* (1000) o tempo é inferior a 30 segundos, o que consideramos um tempo aceitável. Sendo assim, demonstra-se que a ferramenta apresentada pode ser utilizada para criar cenários de redes de diversos tamanhos, dos mais simples aos mais complexos.

4.2. Teste de tempo: Caso de teste

Por utilizar ambiente virtual, com liberdade para simular cenários complexos de rede, um fator que pode impactar na utilização da ferramenta é o tempo de execução de cada caso de teste.

A ideia do experimento é criar um cenário de rede e medir o tempo que cada caso de teste consome. Para isso, é levado em consideração o protocolo da comunicação e as regras inseridas no *firewall*.

O experimento foi realizado utilizando um cenário de rede contendo duas redes distintas, e cada rede contendo um *host*. O *firewall* foi configurado no roteador, interligando

as duas redes. Os casos de testes foram pensados para gerar conexões que sejam impedidas pelo *firewall* e conexões que passem pelo *firewall*, ou seja, os casos de testes são feitos com a finalidade de simular diversas situações dentro do cenário de rede.

O experimento consistiu em realizar 4 casos de teste, onde 2 testes são realizados utilizando o protocolo TCP e 2 testes referente ao protocolo UDP. Referente aos 2 casos de teste utilizando o protocolo TCP, um dos casos de teste é bloqueado pelo *firewall* e o outro não, o processo é análogo com o protocolo UDP. Os 4 casos de teste foram repetidos 5 vezes a fim de garantir a confiabilidade no tempo de execução dos mesmos. A Tabela 4.2 apresenta os resultados obtidos após a execução do experimento.

A Figura 4.1 representa o cenário de rede descrito anteriormente, as regras de *firewall* e os testes deste experimento, são abordados a seguir.

Figura 4.1. Cenário de rede para os testes.

Regras de *firewall*

As regras de *firewall* implementadas nesse experimento foram escolhidas para contemplarem os protocolos TCP e UDP, as regras são:

- iptables -A FORWARD -s 192.168.0.2 -d 10.0.0.2 -p tcp - -dport 80 -j DROP
- iptables -A FORWARD -s 10.0.0.2 -d 192.168.0.2 -p udp - -dport 53 -j DROP

Casos de testes a serem realizados

Os casos de testes foram elaborados com a finalidade de contemplar todas as opções possíveis dentro do cenário de rede, sendo assim, foram criados 2 casos de testes para cada protocolo, na qual um dos testes o *firewall* deve descartar os pacotes e no outro o *firewall* deve liberar o encaminhamento dos pacotes de rede. Os testes são:

- O *host* 192.168.0.2 NÃO deve conectar ao *host* 10.0.0.2 utilizando protocolo TCP na porta 80.
- O *host* 192.168.0.2 DEVE conectar ao *host* 10.0.0.2 utilizando protocolo TCP na porta 53.

- O *host* 10.0.0.2 NÃO deve conectar ao *host* 192.168.0.2 utilizando protocolo UDP na porta 53.
- O *host* 10.0.0.2 DEVE deve conectar ao *host* 192.168.0.2 utilizando protocolo TCP na porta 80.

Desta forma, foi calculado a média dos tempos obtidos para cada protocolo. O experimento foi repetido 5 vezes e a média dos tempos obtidos dos 5 experimentos foi calculada.

Tabela 4.2. Tempo médio gasto para a ferramenta executar as repetições dos 4 casos de teste.

	Bloqueando		Não Bloqueando	
Teste\Protocolo	UDP	TCP	UDP	TCP
Primeira execução dos testes	3.348	3.358	3.347	3.363
Segunda execução dos testes	3.357	3.378	3.362	3.361
Terceira execução dos testes	3.350	3.375	3.353	3.362
Quarta execução dos testes	3.357	3.378	3.366	3.351
Quinta execução dos testes	3.470	3.344	3.357	3.352
Média (s)	**3.376**	**3.367**	**3.357**	**3.358**
Desvio padrão (s)	0.052	0.015	0.007	0.006

Através da Tabela 4.2, pode-se observar que os testes realizados para constatar o funcionamento das regras de *firewall* em redes simples podem ser feitas de maneira rápida (em média 3,36 segundos), facilitando experimentos com o objetivo de entender o funcionamento de diferentes regras, além de tornar viável a execução de grande quantidade de testes em um tempo aceitável.

É importante enfatizar que a quantidade de casos de testes não interfere no tempo de execução de cada caso de teste, somente no tempo final de execução experimento.

4.3. Utilizando a ferramenta para atividade da disciplina de segurança

Um dos objetivos da ferramenta é auxiliar no entendimento a respeito de configurações de *firewalls*. Para validar esse objetivo é necessário que indivíduos com conhecimento básico em redes de computadores e segurança da informação usem a ferramenta para implementar regras de *firewall*.

O experimento consistiu em implementar regras a partir de um enunciado, disponível em Apêndice A, que foi aplicado em sala de aula durante a disciplina de Segurança e Auditoria de Sistemas da Universidade Tecnológica Federal do Paraná (UTFPR) de Campo Mourão. No qual um usuário configuraria o cenário de rede, bem como o *firewall* e define testes para verificar a eficiência das regras do *firewall*.

O cenário consiste em duas redes distintas interligadas por um roteador (*firewall)*,

cada rede possui um *switch* interligando dois computadores. Uma das redes deve ser *Lan Screened* (não pode prover serviços).

A Figura 4.2 representa o cenário de rede a ser utilizado durante o experimento. Vale ressaltar que as regras de *firewall* devem ser inseridas pelo individuo que for realizar o experimento. Assim como as regras de *firewall*, os casos de testes devem ser elaborados a partir de enunciados. O enunciado é apresentado a seguir.

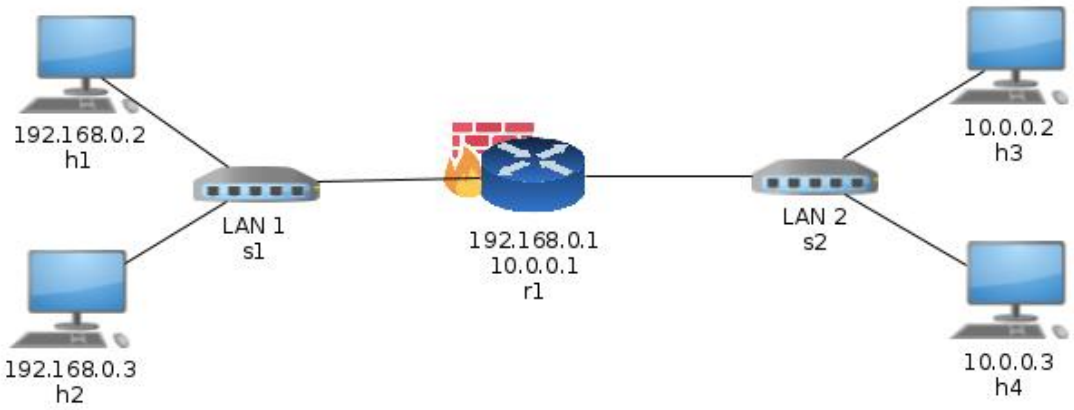

Figura 4.2. Cenário de rede com duas redes distintas interligadas por um roteador, cada rede possui dois *hosts* conectados a um *switch*.

Enunciado para elaboração de regras de *firewall* e casos de testes

- LAN 1 deve ser uma LAN *Screened*, ou seja, não pode ser servidor.
- Testar se mesmo executando serviços HTTP e SSH em h1 e h2, se é possível acessá-los de h3 e h4.
- Verificar se LAN 1 pode acessar normalmente serviços de HTTP e *Hyper Text Transfer Protocol Secure* (HTTPS) em h3.
- Verificar se LAN 1 pode acessar normalmente serviços de SSH e DNS em h3.
- O *firewall* só pode ser acessado via SSH pelos *hosts* da LAN 1.
- O *firewall* pode acessar servidores na LAN 2, mas não na LAN 1.

O resultado obtido a partir do enunciado foi satisfatório, uma vez que a solução apresentada pelo usuário foi a solução correta.

Solução apresentada pelo usuário

As regras de *firewall* apresentadas foram:

- iptables -F
- iptables -t nat -F
- iptables -A FORWARD -d 192.168.0.0/24 -m state - -state NEW,INVALID -j DROP
- iptables -A INPUT ! -s 192.168.0.2 -p tcp - -dport 22 -j DROP

- iptables -A OUTPUT ! -d 10.0.0.0/24 -m state - -state NEW,INVALID -j DROP

Referente ao desenvolvimento dos casos de testes, a solução apresentada pelo usuário atende aos requisitos propostos pelo enunciado apresentado. A Tabela 4.3 apresenta a solução correta do teste e a solução apresentada pelo usuário, cada linha da tabela representa um caso de teste.

Tabela 4.3. Tabela referente aos casos de testes do experimento.

Teste	IP		Protocolo	Porta		*Expected*
	Origem	Destino		Origem	Destino	
0	10.0.0.2	192.168.0.2	TCP	*	80	*deny*
1	10.0.0.3	192.168.0.3	TCP	*	22	*deny*
2	192.168.0.2	10.0.0.2	TCP	*	80	*accept*
3	192.168.0.3	10.0.0.2	TCP	*	443	*accept*
4	192.168.0.2	10.0.0.3	TCP	*	22	*accept*
5	192.168.0.3	10.0.0.3	UDP	*	53	*accept*
6	192.168.0.2	192.168.0.1	TCP	*	22	*accept*
7	10.0.0.3	10.0.0.1	TCP	*	22	*deny*
8	10.0.0.1	10.0.0.3	TCP	*	80	*accept*
9	192.168.0.1	192.168.0.2	TCP	*	22	*deny*

A finalidade deste experimento foi comprovar o funcionamento da ferramenta para aprendizado a respeito de *firewalls*, onde um usuário, a partir de um enunciado, desenvolveu regras de *firewall* e testes. Como resultado, pode-se notar que o *firewall* configurado ficou condizente com o enunciado apresentado.

4.4. Dificuldades enfrentadas

Para chegar até a abordagem atual, vários problemas foram superados. Esta seção apresentará as principais dificuldades encontradas ao decorrer da pesquisa e implementação da ferramenta.

4.4.1. Leitura de dados

Inicialmente, o formato do arquivo de entrada utilizado era o *Comma-Separated Values* (CSV). Apesar de funcionar, o mesmo limitava a ferramenta e dificultava análise humana. A limitação do formato CSV neste contexto, era a falta de representação dos dados como objetos, tornando necessário a criação de vários arquivos no formato CSV. Para o Núcleo de Gerenciamento obter todos os dados necessários, eram necessários pelo menos 4 arquivos de entrada no formato CSV. Atualmente apenas um arquivo no formato JSON é o suficiente. Além da limitação do CSV, existia também a dificuldade em se entender os dados contidos no arquivo de entrada, pois as informações ficavam dispersas, causando confusão e erros ao criar ou alterar tais arquivos.

4.4.2. Geração de tráfego

Como a ideia é realizar testes nas regras de *firewall* utilizando a simulação de serviços, foi pensado em utilizar algumas ferramentas já existentes para simular serviços de rede. O `Nping` é uma ferramenta para geração de pacotes de rede, muito utilizada para testes, pois permite ao usuário que manipule o cabeçalho dos pacotes. Contudo, ao utilizar o `Nping`, foi detectado que o mesmo cria uma conexão própria, para então testar a conexão desejada pelo usuário. Apesar de funcionar, os registros do `Tcpdump` ficavam bastante poluídos, dificultando a análise dos mesmos. A primeira solução pensada para resolver esse problema, foi usar o `Netcat`. O `Netcat`, apesar de funcionar corretamente para a proposta da ferramenta desenvolvida, o mesmo não oferece suporte a ICMP. Além disso, o mesmo gerava alguns pacotes extras na rede, o que dificultava a análise dos registros.

A solução encontrada para este problema em questão foi criar *software* próprio utilizando *Socket* Python, que simule os serviços de rede. Foram desenvolvidos códigos de cliente (GUERRA, 2019d) e servidor (GUERRA, 2019e) para conexões TCP, assim como cliente (GUERRA, 2019f) e servidor (GUERRA, 2019g) para conexões UDP. Por ser implementado para uso exclusivo da ferramenta desenvolvida, o mesmo sanou os problemas enfrentados ao utilizar as ferramentas citadas anteriormente.

4.4.3. Tempo de resposta

Ao realizar os primeiros testes com a ferramenta, foi constatado que o tempo de resposta da ferramenta estava relativamente alto, aproximadamente 15 segundos para cada teste dentro de um experimento, ou seja, um experimento com apenas 4 testes levariam aproximadamente 60 segundos para ser concretizado. Com valores de tempo fora do esperado, a demora para realizar os experimentos estavam inviabilizando o uso da ferramenta, cada teste demorava aproximadamente 5 vezes mais do que os tempos apresentados na Tabela 4.2. Para descobrir, foram colocados cronômetros em cada operação do Núcleo de Gerenciamento, assim foi possível observar quais operações estavam tomando mais tempo. As operações que demandavam mais tempo eram operações de leitura e escrita em arquivos. Sendo assim, foi aplicado refatoração de código, trabalhando com os dados em memória, e utilizando leitura e escrita de arquivos somente quando necessário.

4.4.4. Sincronismo

Com a necessidade de simular serviços de rede, foram implementados códigos utilizando *Socket* Python para cliente/servidor. Contudo, houveram problemas de sincronismo na execução dos códigos de cliente/servidor, causando instabilidade na ferramenta. Isso se deu ao fato do Núcleo não conseguir executar os programas de cliente e servidor simultaneamente. Por

vezes o cliente tentava conectar em um servidor que já estava fechado. A solução para este problema, foi adotar o uso de *threads*, garantindo a execução dos códigos de cliente/servidor simultaneamente, evitando então inconsistências na ferramenta.

4.5. Considerações Finais

A finalidade dos experimentos é validar o uso da ferramenta, independente do cenário de rede a ser utilizado e das regras inseridas nos *firewalls*. Os resultados obtidos comprovam o funcionamento da ferramenta, onde a mesma se comporta de forma similar a ambientes reais.

É importante considerar a necessidade de se realizar mais experimentos, tal como aplicar a ferramenta desenvolvida durante as aulas da disciplina de segurança. O intuito é separar a turma em questão em dois grupos de alunos, onde uma parte usaria métodos convencionais de se testar as regras de *firewall* e a outra metade usaria a ferramenta desenvolvida, a partir disso, verificar qual grupo obteve o melhor desempenho. Este experimento e outros são abortados no Capítulo 5.

Capítulo 5

Conclusões

A execução dos experimentos constatou que o objetivo deste trabalho foi alcançado, a proposta se mostrou viável no auxílio aos administradores de rede, uma vez que permite detectar o pacote que está sendo bloqueado indevidamente ou ainda o pacote que deveria ser bloqueado e está alcançando o seu destino. Apesar da ferramenta não ter sido utilizada por alunos em disciplinas de segurança, por fornecer informações a respeito do caminho dos pacotes de rede e das regras de *firewall* de forma intuitiva, fica evidente que sua utilização da ferramenta colabora para o usuário entender o que está acontecendo com o tráfego de rede e consequentemente facilitando o entendimento do funcionamento dos serviços de rede e da configuração de *firewalls*.

Foi desenvolvida uma ferramenta para testar as regras de *firewalls*, possibilitando a configuração de cenário de rede, o qual é especificado pelo usuário, e posteriormente, os testes das regras de *firewall* através de uma interface *web*. Sendo assim, foi possível identificar falhas e configurar *firewalls* mais efetivos.

Além disso, incentivar o aprendizado prático a respeito de configuração de *firewall* uma vez que seja possível criar diferentes cenários de rede, do mais simples até o mais complexo, e testar as regras do *firewall* de forma econômica em termos de tempo e recursos físicos.

5.1. Trabalhos Futuros

5.1.1. Término de implementação da interface

A interface *web*, necessita ser terminada. Para obter uma interface consistente, é necessário realizar implementações onde possibilita a criação de *switchs* e enlaces, assim como a

necessidade de elementos para criar os casos de testes a serem realizados durante um experimento.

Além disso, é necessário finalizar a implementação da lógica por trás da interface *web*, tal como converter os dados inseridos na interface *web* para o formato JSON, com a finalidade de ser utilizado pelo Núcleo de Gerenciamento. É necessário também informar o resultado do experimento para o usuário, realizando a leitura do arquivo JSON e apresentando os dados na interface *web*.

5.1.2. Protocolo ICMP

A adição de possibilidade de realizar testes utilizando o protocolo ICMP é uma das propostas de melhoria da ferramenta. Atualmente, abrangendo os protocolos UDP e TCP, a implementação de testes para o protocolo ICMP enriqueceria ainda mais a ferramenta, tornando-a mais completa e abrangente.

A implementação de testes para comunicações que utilizam o protocolo ICMP se daria da mesma forma como acontece ao utilizar TCP e UDP.

5.1.3. Sugestão de casos de teste

Futuramente, poderão ser sugeridos casos de testes automaticamente. Ou seja, sugerir casos de testes na qual são testados com mais frequência, como por exemplo, testes de HTTP, SSH, DNS e entre outros.

As sugestões desses casos de teste ajudariam a otimizar tempo de configuração do cenário de rede, uma vez que os casos de testes já estariam previamente preparados, sendo utilizados de acordo com a necessidade do usuário.

5.1.4. Implementar automaticamente a volta dos pacotes

A ferramenta poderá testar de forma automática um caso de teste para o usuário. Quando um caso de teste é executado, um *host* simula por servidor e outro *host* simula um cliente, a ideia principal deste trabalho é sugerir o teste o inverso também, ou seja, testar a ida e a volta dos pacotes.

Este trabalho poderá economizar tempo na hora de definir casos de testes dentro de um experimento. Além disso, poderá evitar erros onde o usuário esqueça de testar a volta dos pacotes.

5.2. Experimentos Futuros

5.2.1. Comparação entre ambiente virtual e real: Cenário mínimo

Um dos passos para validar a ferramenta desenvolvida, é garantir que o ambiente virtual produza exatamente o mesmo resultado do ambiente real.

Este experimento, por exemplo, consiste em montar um cenário real mínimo, contendo dois *hosts* interligados por um roteador, onde o *firewall* estará configurado no roteador. O mesmo cenário será criado e virtualizado utilizando a ferramenta. Serão criados casos de testes com o intuito de aplicá-los aos dois cenários.

A Figura 5.1 ilustra como o cenário de rede a ser utilizado nos testes de ambiente real e simulado.

Figura 5.1. Cenário de rede mínimo.

5.2.2. Utilizar a ferramenta em sala de aula

Para garantir a validade da ferramenta em ambiente acadêmico, foi pensado um experimento onde a ferramenta seja aplicada em uma turma da disciplina de segurança. O experimento consiste em separar a turma em dois grupos, o primeiro grupo utilizaria métodos de testes em regras de *firewall* convencionais, enquanto o segundo grupo usaria a ferramenta desenvolvida. Com isso, obter dados qualitativos para avaliar o desempenho da ferramenta dentro da sala de aula, tais como, qual grupo obteve o *firewall* mais correto para um determinado problema e qual grupo realizou o experimento em menor tempo, e outros dados.

5.3. Conclusão Final

A ferramenta desenvolvida tem muito potencial de trabalhos futuros, ou seja, este trabalho mostrou que a ferramenta é efetiva, porém não houve tempo hábil para desenvolver mais funcionalidades e realizar outros experimentos com a ferramenta implementada.

A grande vantagem dessa ferramenta de testes em regras de *firewalls*, se comparado com outros trabalhos, é o fato de simular os serviços de rede e testar os *firewalls* como se

estivesse em ambiente de rede real. Com isso, é possível configurar *firewalls* menos suscetíveis a erros, de forma que possam cumprir com a política de segurança adotada.

Com isso, podemos concluir que a ferramenta desenvolvida neste trabalho utiliza uma abordagem diferente do que foi apresentada por outros trabalhos, simulando serviços de rede e aplicando testes práticos nos *firewalls*, através de testes e análise do comportamento dos pacotes de rede, a ferramenta colabora para prover maior segurança em redes de computadores e auxilia no ensino e aprendizado a respeito de *firewalls*.

Apêndices

Apêndice

A

Enunciado de atividade utilizada em experimento

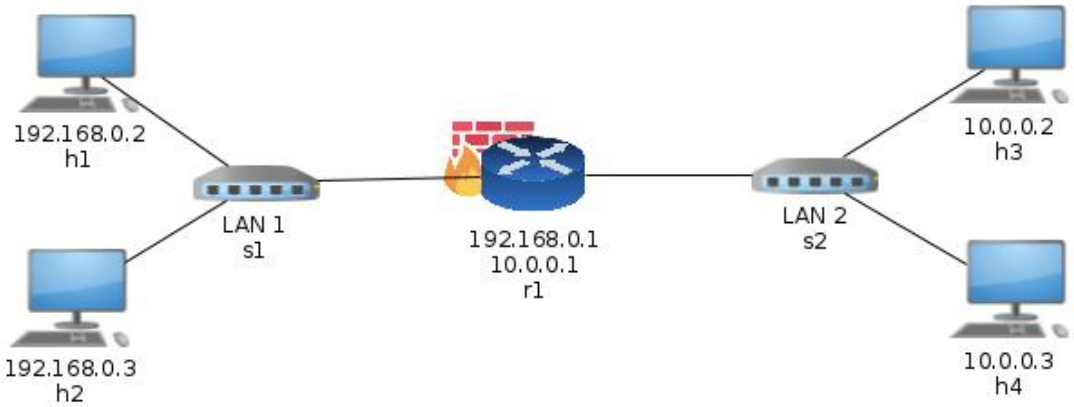

Figura A.1. Cenário de rede

Teste 1 (utilizando política de acessar tudo):

1. LAN 1 deve ser uma LAN *Screened*, ou seja, não pode ser servidor.
   - Testar se mesmo executando serviços HTTP e SSH em h1 e h2, se é possível acessá-los de h3 e h4.
2. Verificar se LAN 1 pode acessar normalmente.
   - serviços de HTTP e HTTPS em h3;
   - serviços de SSH e DNS em h4.
3. O *firewall* só pode ser acessado via SSH (ser servidor de SSH) pelos *hosts* da rede 1 (não por outras redes – ex, LAN2).
4. O *firewall* pode acessar servidores na LAN2, mas não na LAN1.

Referências

ARTYMIAK, Jacek. *Building Firewalls with OpenBSD and PF*. [S.l.]: Lublin, 2003. Second Edition.

BISHOP, Matt. *Introduction to Computer Security*. [S.l.]: Pearson Education, 2005.

FOROUZAN, Behrouz A. *Comunicação de Dados e Redes de Computadores*. fourth. [S.l.]: McGraw Hill, 2008.

GUERRA, Matheus Sapia. *Código gerador de teste*. 2019. Disponível em: <https://github.com/GuerraUTFPR/Tool4AnalysisFwRules/blob/master/src/createTests.py>. Acesso em: 25 nov. 2019.

GUERRA, Matheus Sapia. *Interface Web*. 2019. Disponível em: <https://github.com/GuerraUTFPR/Tool4AnalysisFwRules/blob/master/interface/index2.html>. Acesso em: 25 nov. 2019.

GUERRA, Matheus Sapia. *Repositorio*. 2019. Disponível em: <https://github.com/GuerraUTFPR/Tool4AnalysisFwRules/tree/master/src>. Acesso em: 25 nov. 2019.

GUERRA, Matheus Sapia. *Socket TCP (Client)*. 2019. Disponível em: <https://github.com/GuerraUTFPR/Tool4AnalysisFwRules/blob/master/src/tcpClient.py>. Acesso em: 25 nov. 2019.

GUERRA, Matheus Sapia. *Socket TCP (Server)*. 2019. Disponível em: <https://github.com/GuerraUTFPR/Tool4AnalysisFwRules/blob/master/src/tcpServer.py>. Acesso em: 25 nov. 2019.

GUERRA, Matheus Sapia. *Socket UDP (Client)*. 2019. Disponível em: <https://github.com/GuerraUTFPR/Tool4AnalysisFwRules/blob/master/src/udpClient.py>. Acesso em: 25 nov. 2019.

GUERRA, Matheus Sapia. *Socket UDP (Server)*. 2019. Disponível em: <https://github.com/GuerraUTFPR/Tool4AnalysisFwRules/blob/master/src/udpServer.py>. Acesso em: 25 nov. 2019.

LANTZ, Bob; HELLER, Brandon; MCKEOWN, Nick. A network in a laptop: Rapid prototyping for software-defined networks. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. New York, NY, USA: ACM, 2010. (Hotnets-IX), p. 19:1–19:6. ISBN 978-1-4503-0409-2. Disponível em: <http://doi.acm.org/10.1145/1868447.1868466>.

MARMORSTEIN, Robert; KEARNS, Phil. A tool for automated iptables firewall analysis. In: *Usenix annual technical conference Freenix Track*. [S.l.: s.n.], 2005.

MININET. *Mininet Python API*. 2019. Disponível em: <http://mininet.org/api/hierarchy.html>. Acesso em: 18 nov. 2019.

MORAES, Alexandre Fernandes de. *Segurança em redes: fundamentos*. first. [S.l.]: Editora Errica, 2010.

NAKAMURA, Emilio Tissato; GEUS, Paulo Lício de. *Segurança de redes em ambientes cooperativos*. [S.l.]: Novatec, 2010.

PYTHON. *Python Socket*. 2019. Disponível em: <https://www.tcpdump.org/>. Acesso em: 22 nov. 2019.

RASH, Michael. *Linux firewalls : attack detection and response with iptables, psad, and fwsnort*. [S.l.]: William Pollock, 2007.

ROSSUM, Guido. *Python Reference Manual*. Amsterdam, The Netherlands, The Netherlands, 1995.

SCAMBRAY, Joel; MCCLURE, Stuart; KURTZ, George. *Hackers Expostos*. [S.l.]: Makron Books, 2000.

STALLINGS, William. *Data and Computer Communications*. eighth. [S.l.]: Pearson Prentice Hall, 2007.

STALLINGS, Willian. *Criptografia e segurança de redes*. fourth. [S.l.]: Pearson, 2008.

SUEHRING, Steve; ZIEGLER, Robert. *Linux Firewalls*. third. [S.l.]: Novell Press, 2005.

TANENBAUM, Andrew S.; WETHERALL, David J. *Computer Networks*. 5th. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010. ISBN 0132126958, 9780132126953.

TCPDUMP. *Tcpdump*. 2019. Disponível em: <https://www.tcpdump.org/>. Acesso em: 22 nov. 2019.

UBUNTU. *Ubuntu*. 2019. Disponível em: <https://ubuntu.com/>. Acesso em: 18 nov. 2019.

VORONKOV, Artem; IWAYA, Leonardo Horn; MARTUCCI, Leonardo A.; LINDSKOG, Stefan. Systematic literature review on usability of firewall configuration. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 50, n. 6, p. 87:1–87:35, dez. 2017. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/3130876>.