LEANDRO MEIRA MARINHO QUEIROZ

Fireasy: uma ferramenta para auxílio na modelagem de políticas de segurança, tradução e compreensão de configurações de *firewalls*

Presidente Prudente - SP
2021

Fireasy: uma ferramenta para auxílio na modelagem de políticas de segurança, tradução e compreensão de configurações de *firewalls*

LEANDRO MEIRA MARINHO QUEIROZ

Orientador: *Prof. Dr. Rogério Eduardo Garcia*

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UNESP, necessária para a obtenção do título de mestre em Ciências da Computação.

FCT-Unesp – Presidente Prudente
Dezembro/2021

Q3f

Queiróz, Leandro Meira Marinho
Fireasy: uma ferramenta para auxílio na modelagem de políticas de segurança, tradução e compreensão de configurações de firewalls / Leandro Meira Marinho Queiróz. -- Presidente Prudente, 2021
108 f.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Faculdade de Ciências e Tecnologia, Presidente Prudente
Orientador: Rogério Eduardo Garcia
Coorientador: Danilo Medeiros Eler

1. Ciência da Computação. 2. Firewalls (Medidas de segurança para computadores). 3. Visualização de software. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Ciências e Tecnologia, Presidente Prudente. Dados fornecidos pelo autor(a).

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Presidente Prudente

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: Fireasy: uma ferramenta para auxílio na modelagem de políticas de segurança, tradução e compreensão de configurações de firewalls

AUTOR: LEANDRO MEIRA MARINHO QUEIROZ
ORIENTADOR: ROGÉRIO EDUARDO GARCIA

Aprovado como parte das exigências para obtenção do Título de Mestre em Ciência da Computação, área: Computação Aplicada pela Comissão Examinadora:

Prof. Dr. ROGÉRIO EDUARDO GARCIA (Participaçao Virtual)
Departamento de Matemática e Computação / UNESP/Câmpus de Presidente Prudente

Prof. Dr. RONALDO CELSO MESSIAS CORREIA (Participaçao Virtual)
Matemática e Computação / FCT/UNESP - Câmpus Presidente Prudente

Profa. Dra. FERNANDA MADEIRAL DELFIM (Participaçao Virtual)
KTH Royal Institute of Technology

Presidente Prudente, 15 de dezembro de 2021

Resumo

A internet atualmente é utilizada por mais da metade da população mundial. Se por um lado isso facilita a comunicação e a busca por conhecimento, por outro aumenta a quantidade de incidentes de segurança. Empresas e organizações armazenam quantidades cada vez maiores de dados valiosos, sendo necessária a implementação de mecanismos para protegê-los de pessoas mal-intencionadas. Existem técnicas e procedimentos que visam a aumentar a segurança de sistemas computacionais, seja por meio da proteção física dos servidores, por diretivas realizadas nos sistemas operacionais a fim de dificultar invasões, ou por serviços de proteção entre redes. Os *Firewalls* fazem parte deste último, tendo como objetivo filtrar os pacotes que entram e saem da rede. Suas configurações dependem de políticas de segurança, as quais consistem em documentos que descrevem o que é permitido trafegar na rede e o que é proibido. A transcrição das políticas de segurança em regras, escritas em linguagem nativa de *firewall*, que as representam, consiste na principal fonte de erros nas configurações de *firewalls*. Não existem ferramentas acadêmicas ou de código aberto que permitam representar as regras de *firewalls* já implementados por meio de elementos gráficos, e que seja possível editá-las para posteriormente traduzi-las novamente para linguagem nativa de *firewall*. Este trabalho possui o objetivo de apresentar uma ferramenta para modelagem de regras de *firewalls* além de possibilitar a abstração das regras em elementos gráficos, permitindo editá-las. Por fim, foi conduzido um experimento controlado para validação da abordagem, que indicou uma melhora na tradução de políticas de segurança em regras de *firewall* utilizando a ferramenta, quando comparada com a abordagem ad-hoc. Na tarefa de compreensão de regras de *firewall*, houve uma homogeneização do desempenho dos participantes quando eles utilizaram a ferramenta.

Palavras Chave— firewall, visualização de firewall, SPML, packet filter, políticas de segurança

Abstract

The internet is currently used by more than half of the world's population. If, on the one hand, this facilitates communication and the search for knowledge, on the other hand, it increases the number of security incidents. Companies and organizations store increasing amounts of valuable data, requiring the implementation of mechanisms to protect them from malicious people. There are techniques and procedures that aim to increase the security of computer systems, either through physical protection of servers, through directives carried out in operating systems in order to hinder intrusions, or through protection services between networks. Firewalls are part of the last one, with the objective of filtering packets that enter and leave the network. Its settings depend on security policies, which consist of documents that describe what is allowed to travel on the network and what is prohibited. The transcription of security policies into rules, written in native firewall language, that represent them, is the main source of errors in firewall configurations. There are no academic or open source tools that allow representing the rules of already implemented firewalls through graphic elements, and that it is possible to edit them to later translate them back into the firewall's native language. This work has the objective of presenting a tool for modeling firewall rules in addition to allowing the abstraction of rules in graphic elements, allowing them to be edited. Finally, a controlled experiment was conducted to validate the approach, which indicated an improvement in the translation of security policies into firewall rules using the tool, when compared to the ad-hoc approach. In the task of understanding firewall rules, there was a homogenization of the participants' performance when they used the tool.

Keywords— firewall, firewall visualization, SPML, packet filter, security policies

Lista de Figuras

Lista de Tabelas

Índice

CAPÍTULO 1

Introdução

1.1 Contextualização

Com a globalização da internet e com bilhões de usuários possuindo mais de um dispositivo conectado, a segurança se tornou um problema de grandes proporções (Tanenbaum et al., 2011). No gráfico apresentado na Figura 1.1, é ilustrada a quantidade anual de incidentes gerais (os tipos de incidentes são apresentados na Seção 2.3) reportados ao CERT.br - Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (Cert.br, 2019c). É interessante observar que o número de incidentes anuais cresceu aproximadamente duzentas e oitenta vezes entre os anos de 1999 e 2019, passando de 3.107 incidentes para 875.327. Analisando as informações, é iminente a necessidade da implementação de sistemas de segurança para garantir a segurança da informação, a fim de evitar que dados sejam acessados, interceptados ou alterados por pessoas ou organizações não autorizadas.

Quando se considera segurança em sistemas computacionais, existem ferramentas (Talukder et al. (2009), Lara e Ramamurthy (2016), Kaçar e Öztoprak (2017), Symantec (2019), Trendmicro (2019), Fortinet (2019), FireEye (2019), Cisco (2019), Nagios (2019), Nmap (2019), Snort (2019), Wireshark (2019), OSSEC (2019), Metasploit (2019), Fail2Ban (2019)) e técnicas (Banik e Pena (2015), Ford et al. (2016), Sharma e Sharma (2016) Parres-Peredo et al. (2018), Hong et al. (2018), Chandre et al. (2018)) para buscar um alto nível de impenetrabilidade. No entanto, com as tecnologias ligadas à internet sendo constantemente desenvolvidas, surgindo novos serviços e ferramentas, novas vulnerabilidades os acompanham, o que dificulta (ou até mesmo impossibilita) a existência de ambientes totalmente seguros.

O foco deste trabalho está voltado especificamente ao *firewall*, um serviço de proteção para sistemas de segurança. Ele é o responsável por proteger os serviços e os dados de uma rede interna contra ataques oriundos de redes externas (Mosharraf e Forouzan, 2013).

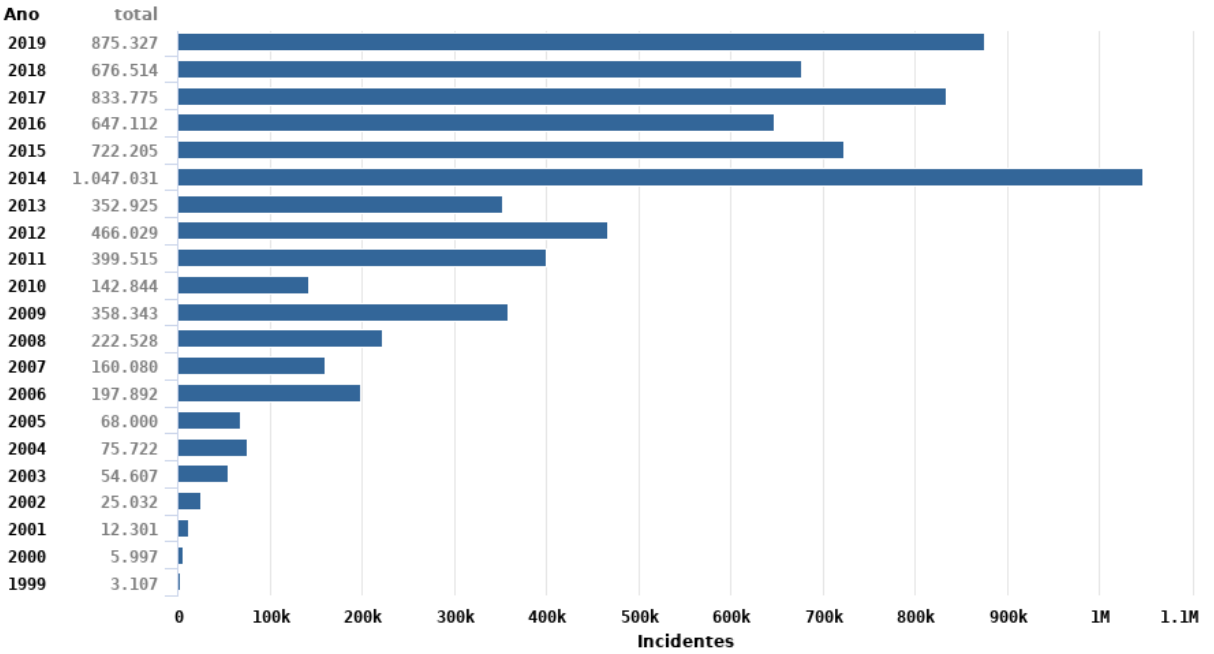


Figura 1.1: Total de Incidentes Reportados ao CERT.br por ano (Cert.br, 2019c).

Empresas e organizações que não possuem um *firewall*, ou o possuem com falhas na configuração, estão sujeitas a ataques como: disseminação de *worms*[1], sequestros de dados e escaneamentos da rede, além de invasões aos computadores com o objetivo de posteriormente utilizá-los para enviar *spam*, minerar criptomoedas, ou fazer parte de ataques DDoS (*Dynamic Denial of Service*).

Contudo, o desafio para realizar uma filtragem eficiente dos pacotes originados de uma rede, ou que a adentram, não consiste na simples instalação do *firewall*. A tarefa mais importante é a sua configuração (Rubin et al., 1997). Realizar a tradução das políticas de segurança em regras de *firewall* (ambos abordados na Seção **??**) é um procedimento complexo e suscetível a erros.

1.2 Motivação e Justificativa

A configuração inicial de um *firewall* usualmente se dá por meio da tradução de políticas de segurança em regras na linguagem de *firewall*. Políticas de segurança podem ser diretivas de segurança gerais (como a negação de tráfego em portas utilizadas por *worms* conhecidos), além de políticas específicas que possam levar em consideração: a produtividade dos usuários (bloqueando acesso a redes sociais por exemplo), ou o consumo de largura de banda (KUROSE e ROSS, 2013). O nível de detalhamento da descrição das políticas de segurança pode variar, cabendo ao administrador de redes realizar a sua tradução em linguagem nativa de *firewall*.

Para auxiliar no processo de tradução da política de segurança em regras de *firewall*, Trevisani e Garcia (2008) propuseram uma notação gráfica chamada SPML (*Security Policy Modeling Language*). Por meio dela, é possível utilizar elementos gráfi-

[1] um programa independente, semelhante a um vírus, porém que se replica com o objetivo de se espalhar para outros computadores

cos para especificar as políticas de segurança, que posteriormente são traduzidas em regras de linguagem nativa de *firewall*.

Em trabalho posterior, Sapia (2016) definiu a evolução da SPML, chamada SPML2. Além de melhorias realizadas, abordadas com detalhes na Seção 4.12, são especificados os formalismos da SPML e SPML2, com definições léxicas, sintáticas e semânticas.

Para validar as contribuições científicas da SPML2, Sapia (2016) apresentou uma ferramenta chamada *SP2Model*, que permite realizar o *design* das políticas de segurança por meio de elementos gráficos, posteriormente traduzidos automaticamente para linguagem nativa de *firewall*.

No entanto, a utilização da *SP2Model* não auxilia os *firewalls* já implementados, pois por meio da ferramenta é possível apenas realizar a implementação inicial das políticas de segurança. Portanto, este trabalho é motivado pelo fato de que milhares de *firewalls* em produção não possuem uma representação gráfica de suas regras. Possui-las permitiria aos administradores de rede realizar o gerenciamento dos *firewalls* utilizando elementos de alto nível, facilitando a adição, edição e remoção de regras.

1.3 Formulação do Problema

O processo que se inicia com a definição das políticas de segurança e a sua posterior tradução em regras em linguagem de *firewalls* é suscetível a erros. Conforme apresentado na Subseção **??**, os erros podem ser de violações de política, de inconsistências, e de ineficiências.

Usualmente, o administrador de redes recebe as políticas de segurança em formato de texto, com frases em linguagem comum, como “apenas o RH pode ter acesso ao servidor de folha de pagamento”. Esse tipo de frase não especifica detalhes técnicos que devem ser levados em consideração ao definir a regra no *firewall*. Para tanto, o administrador de redes deve tomar conhecimento do IP do servidor de folha de pagamento e dos IPs dos computadores do setor de RH (ou a subrede exclusiva deste local). Então, com o conhecimento da linguagem de *firewall*, escrever as regras para permitir o acesso ao servidor em questão apenas pelos IPs do RH.

Esse processo de tradução das políticas de segurança em linguagem nativa de *firewall* é facilitado com a utilização da SPML2. Por meio dela, o administrador de redes pode realizar a modelagem das políticas de segurança (definidas previamente) utilizando elementos gráficos. Posteriormente, com a utilização de uma ferramenta como a *Fireasy* (apresentada no Capítulo 5), é possível traduzir diretamente para linguagem nativa de *firewall*.

No entanto, após a implantação inicial das regras no *firewall*, é necessário haver manutenção constante. Considerando que o administrador de redes possua um *firewall* em funcionamento com diversas regras, mas não tenha a modelagem das regras em SPML2, realizar a manutenção do *firewall* pode ser uma tarefa difícil, pois ela depende da análise das regras diretamente em linguagem nativa de *firewall*.

As manutenções podem ocorrer por meio de inclusões de novas regras, alterações

ou remoções. É possível que estas ações resultem em eventos não previstos. Por exemplo, supondo que um administrador de rede tenha escrito, dentre outras regras, a regra ilustrada na Tabela 1.1. Ainda segundo o administrador de rede, o IP 10.1.1.10 estava sofrendo diversas tentativas de invasão na porta 22 (porta do SSH). Com a definição da regra *r1*, todo o tráfego proveniente de redes externas é negado na porta 22 do IP 10.1.1.10. Em um segundo momento, um outro administrador da rede decide que qualquer tráfego deve ser permitido para a rede 10.1.1.*. O segundo administrador então adiciona a regra *r1* da Tabela 1.2, sendo esta incluída antes da regra que bloqueava o tráfego da porta 22 no IP 10.1.1.10. Pelo fato do *firewall* contar com muitas regras, e obviamente o administrador não se lembrar de todas elas, ele apenas inclui esta nova regra no ínicio. Considerando um *firewall first match*, a regra *r2* da Tabela 1.2, previamente configurada, nunca será executada. Dessa maneira, todo o tráfego malicioso que estava sendo anteriormente bloqueado passa a ser permitido, resultando em uma falha de segurança.

Tabela 1.1: Regra bloqueando o acesso à porta 22 do IP 10.1.1.10.

Rule	Protocol	Source IP	Source Port	Destination IP	Destination Port	Action
r1	*	*	*	10.1.1.10	22	deny

Tabela 1.2: Exemplo de configuração falha nas regras de um *firewall*.

Rule	Protocol	Source IP	Source Port	Destination IP	Destination Port	Action
r1	*	*	*	10.1.1.*	*	allow
r2	*	*	*	10.1.1.10	22	deny

Portanto, realizar uma simples alteração nas configurações de um *firewall* é uma operação delicada, e exige que o administrador de redes possua o domínio das regras definidas. No entanto, até mesmo o administrador de redes que realizou a implementação inicial das regras do *firewall*, após algum tempo, pode não se recordar de exatamente todas as regras e as relações entre elas. A compreensão dos arquivos de configuração do *firewall* em linguagem nativa é complexa. Esta afirmação se baseia nos trabalhos científicos que buscam abstrair estes arquivos de configuração em modelos e linguagens de alto nível (Mayer et al. (2000), Al-Shaer et al. (2005), Lihua Yuan et al. (2006), Bodei et al. (2018), Tran et al. (2007), Kim et al. (2021) e Mansmann et al. (2012)).

O problema do projeto é fundamentado no fato de que não existem ferramentas acadêmicas ou de código aberto que permitam representar as regras de *firewalls* já implementados por meio de elementos gráficos, e que seja possível editá-las para posteriormente traduzi-las novamente para linguagem nativa de *firewall*. Há apenas a ferramenta comercial da CISCO chamada *Zone-Based Policy Firewall* (CISCO, 2010)

com essas características, porém ela possui custo elevado e limita-se a operar apenas com a linguagem nativa de *firewall* do fabricante.

1.4 Objetivo do Projeto

Como demonstrado por Sapia (2016), a possibilidade de criar um modelo a partir de políticas de segurança, gerando de maneira automática as correspondentes regras em linguagem de *firewall*, pode auxiliar os administradores de redes na configuração inicial do *firewall*. Partindo desta premissa, este projeto se baseia na hipótese da melhora na qualidade das configurações de *firewalls*, mitigando erros, por meio da representação das regras em elementos gráficos.

Este projeto tem como objetivo geral realizar a abstração das regras escritas em linguagem nativa de *firewall* em elementos gráficos, buscando uma melhora no processo de manutenção de regras de *firewalls*. Para traduzir as regras em elementos gráficos, são utilizadas a metalinguagem e os elementos definidos na SPML2.

Na Figura 1.2 é apresentado um fluxo ilustrando uma visão geral dos processos descritos. A *SP2Model* já é capaz de realizar os processos descritos pelas setas azuis, ou seja, a transcrição de políticas de segurança em representação gráfica com SPML2, e sua tradução para linguagem nativa de *firewall*. É importante ressaltar que a definição das políticas de segurança não fazem parte do escopo do projeto. A *Fireasy*, ferramenta desenvolvida no projeto, é capaz de realizar os mesmo processos da SP2Model. No entanto, ao contrário da SP2Model, a *Fireasy* permite a realização do processo inverso, ou seja, a tradução de regras de *Firewall* em representação gráfica com SPML2, permitindo sua edição pelo administrador de redes. Por fim, a representação gráfica em SPML2 pode ser traduzida novamente para regras nativa de *firewall*.

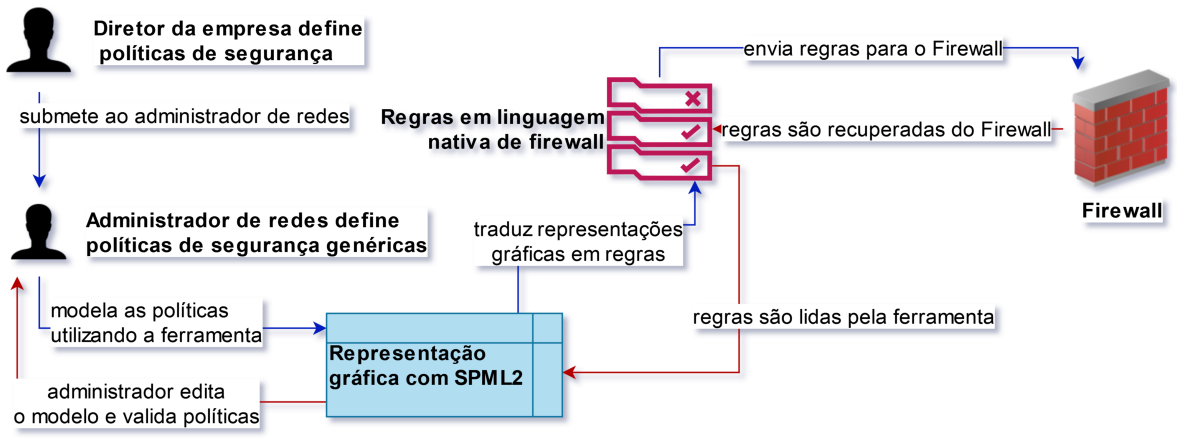


Figura 1.2: Fluxo da visão geral do projeto a ser desenvolvido.

Os objetivos específicos do projeto são:

- Reimplementar a ferramenta SP2Model proposta por Sapia (2016), pois sua arquitetura não está estruturada em módulos.

- Realizar a tradução de regras escritas em linguagem nativa de *firewall* para a metalinguagem da SPML2.
- Representar as regras escritas na metalinguagem em elementos gráficos, conforme o padrão proposto na SPML2.
- Com a representação das regras em elementos gráficos da SPML2, busca-se permitir a análise e manutenção do modelo, por meio de alterações, que posteriormente são novamente transcritas automaticamente para linguagem nativa de *firewall*.
- Implementar um módulo que seja capaz de traduzir a modelagem gráfica em metalinguagem SPML, facilitando a tradução desta para diferentes linguagens de *firewall*.

1.5 Contribuições

Com o desenvolvimento da *Fireasy*, pretende-se oferecer ao administrador de redes uma ferramenta capaz de gerenciar regras de *firewall* de maneira gráfica. Esse gerenciamento inclui a modelagem inicial das regras, além de alterações. A *Fireasy* também pode ser utilizada para fins educacionais, para ensinar conceitos de *firewall* e suas regras para estudantes.

Também foi realizado um experimento controlado para avaliar a eficiência e eficácia da *Fireasy*, apresentado no Capítulo 6.

A ferramenta foi disponibilizada com seu código aberto, sob a licença do MIT (2022), e pode ser baixada no endereço "github.com/leandromeira/Fireasy".

1.6 Organização

Este texto consiste na dissertação de mestrado, uma sequência dos trabalhos realizados anteriormente no grupo de pesquisa, especificamente SPML e SPML2. A solidez destas duas, devido aos elementos gráficos e formalismos definidos, abre a possibilidade de melhorias e novas propostas.

Este texto encontra-se organizado como segue:

- No Capítulo 2 são apresentados conceitos de segurança da informação e de redes.
- No Capítulo **??** são apresentados conceitos de *firewall* e é discorrido sobre as dificuldades e problemas que podem ocorrer.
- No Capítulo 4 são apresentados os trabalhos relacionados disponíveis na literatura e são descritas a SPML e SPML2.
- No Capítulo 5 é apresentada a implementação da ferramenta *Fireasy*.
- No Capítulo 6 é apresentado o experimento conduzido.
- No Capítulo 7 são apresentadas as considerações finais.

CAPÍTULO 2

Segurança entre redes

2.1 Considerações Iniciais

A segurança de, ou entre, redes é um assunto que abrange inúmeros problemas. Nem mesmo grandes empresas, que investem quantidades significativas de recursos em segurança, estão livres de ameaças. Empresas como Facebook (UpGuard, 2019) e Nasa (NASA, 2019) já tiveram problemas de segurança, tendo ocorrido vazamentos de dados. Também vale mencionar os vazamentos realizados por Edward Snowden e Julian Assange, este último o fundador da *WikiLeaks* (WikiLeaks, 2019), que divulgaram documentos sigilosos de agências do governo dos Estados Unidos.

É evidente a necessidade de definição e de aplicação de diretivas que visem à segurança de dados. Na série ISO 27000 (ISO, 2018) são apresentadas normas técnicas elaboradas pela *International Organization for Standardization*, com procedimentos de segurança da informação. Ela define normas com medidas desde a segurança física, como impedir acesso não autorizado a *datacenters*, climatização deles, prevenção de incêndios, até técnicas aplicadas em servidores e ativos de rede, como procedimentos para prevenção a ataques DDoS (*Dynamic Denial of Service*).

Neste capítulo são apresentados brevemente conceitos sobre a segurança na comunicação de dados, identificando problemas relacionados. São apresentados dados recentes sobre os tipos de ataques ocorridos e as portas mais atacadas. Também são abordadas as medidas de segurança relacionadas a cada camada da pilha de protocolos de rede.

2.2 Características da Segurança entre Redes

Segundo Tanenbaum et al. (2011), a segurança de redes visa a impedir que pessoas mal intencionadas sejam capazes de ler ou alterar informações. Também ocupa-se em impedir acessos a serviços não autorizados, e prover mecanismos para verificar a veracidade de uma mensagem.

Segundo KUROSE e ROSS (2013), o objetivo da segurança de redes é cumprir as propriedades de uma comunicação segura, são elas:

- Confidencialidade. É necessário que somente o remetente e o destinatário da mensagem possam ler o seu conteúdo. No entanto, uma mensagem enviada de um remetente A para um destinatário B pode passar por n pontos (nós de rede) até que ela seja entregue a B. A maior preocupação consiste no fato de que os caminhos que a mensagem percorre não fazem parte da rede de A nem de B, sendo necessário garantir que ninguém a leia no caminho. Com o intuito de dificultar a leitura da mensagem por terceiros, o remetente deve cifrá-la de modo que estranhos não possam lê-la. Técnicas de criptografia são utilizadas para prover confidencialidade.

- Integridade de mensagem. Também conhecida como autenticação da mensagem, ela visa a assegurar que a mensagem foi realmente enviada pelo remetente, e que ela não foi alterada no caminho. Isso pode ser garantido utilizando métodos de assinatura digital ou código de autenticação de mensagem.

- Autenticação do ponto final. Ela visa a verificar a identidade do remetente e do destinatário, ou seja, confirmar que eles são realmente quem alegam ser. Isso é realizado utilizando uma lógica similar ao *three-way handshake* presente no protocolo TCP (KUROSE e ROSS, 2013).

- Segurança operacional. Esta propriedade trata da necessidade de empresas e organizações de proteger suas redes contra ataques provenientes da internet. Atacantes podem tentar: invadir servidores e computadores, realizar ataques DoS (*Denial of Service*), mapear a rede interna em busca de vulnerabilidades, instalar *worms*, ou adquirir segredos corporativos. Para realizar a proteção contra estes tipos de ameaças, as duas principais ferramentas utilizadas são os *firewalls*, responsáveis por guardar a fronteira entre a rede interna e a internet pública, e os *IDS* (*Intrusion Detection System*), que realizam uma inspeção profunda dos pacotes em busca de porções maliciosas.

É evidente que tornar uma rede segura não é uma tarefa trivial. Ao contrário, é complexo e custoso implementar meios para garantir cada uma das propriedades de uma comunicação segura. Às técnicas e aos processos que visam a melhorar a segurança de sistemas computacionais, dá-se o nome de *Hardening*. Segundo Terpstra et al. (2004), *Hardening* é o processo pelo qual se identificam falhas gerais no sistema, inclusive no *hardware*, aplicando técnicas para prevenção e correção, além de recuperação de falhas ou invasões de modo rápido e eficiente. Essas técnicas são de naturezas variadas, podendo ser, por exemplo, a instalação de antivírus no computador, a configuração apropriada de um serviço (como *ssh*, *php*, *postfix*, *dns*), ou a configuração de serviços de proteção na rede. Os processos de *Hardening* podem envolver operações que visem a proteger apenas o sistema operacional e serviços de um computador ou uma rede por completo.

2.3 Cenário Atual

Segundo Tanenbaum et al. (2011), grande parte dos problemas de segurança são causados de maneira intencional, e tornar uma rede segura vai muito além de apenas mantê-la livre de erros de programação. Esta afirmação é reforçada observando as Figuras 2.1 e 2.2. Na primeira, é ilustrado um gráfico no qual são apresentados dados coletados pelo CERT.br durante o ano de 2019, especificando os tipos dos ataques reportados (Cert.br, 2019a). Os dados mostram que 46,81% dos ataques reportados são de *scan* [1], seguido por 34,42% de ataques DoS (*Denial of Service*). Na segunda, é ilustrado um gráfico indicando as portas mais mapeadas nesses ataques de *scan* (Cert.br, 2019b). As duas portas com maior número de incidentes reportados são a 22 e 25. A primeira é a porta padrão para o serviço *ssh*, e se um invasor conseguir se autenticar em um computador utilizando esse serviço, ele obtém acesso aos arquivos e permissão para executar comandos. A segunda é a porta padrão para o serviço de envio de e-mail *smtp*. Invasores geralmente utilizam essa porta para o envio de *spam* em massa.

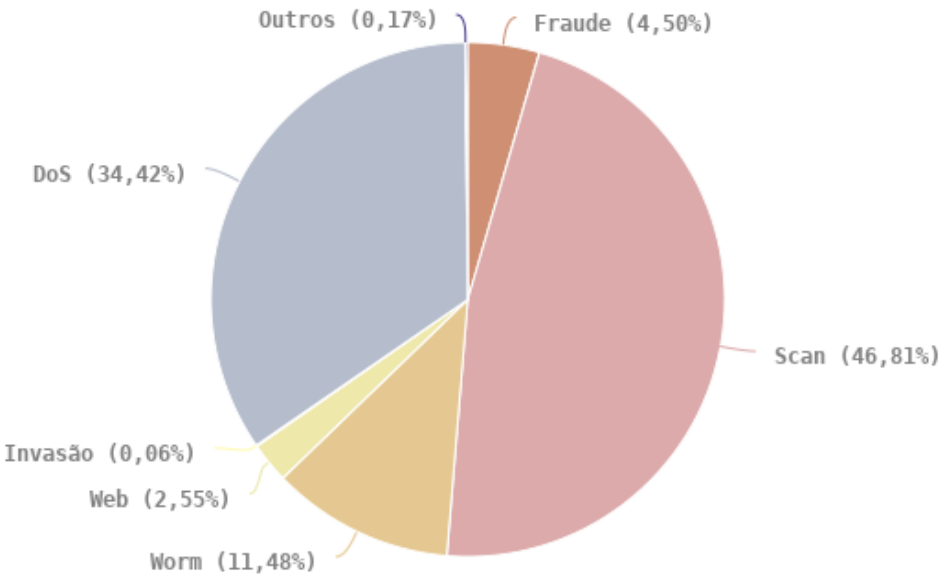


Figura 2.1: Tipos de ataques dos incidentes reportados ao CERT.br no ano de 2019 (Cert.br, 2019a).

O trabalho de Queiroz (2014) discorre sobre diretivas para melhorar a segurança desses dois serviços, além de prover procedimentos de *hardening* em sistemas operacionais Linux. Os trabalhos de Terpstra et al. (2004) e Turnbull (2006), apesar de datados, fornecem técnicas e métodos ainda válidos para melhorar a segurança de sistemas operacionais e de serviços como *ssh*, *dns*, *FTP*, *postfix*, entre outros.

Apesar de ser importante realizar o *hardening* em servidores, é necessário que o *firewall* seja capaz de bloquear a entrada de pacotes indesejados na rede. O bloqueio

[1] Notificações de varreduras em redes, com o intuito de identificar computadores ativos e serviços disponibilizados por eles. É utilizado por atacantes para identificar potenciais alvos, pois permite associar possíveis vulnerabilidades aos serviços habilitados em um computador (Cert.br, 2019a).

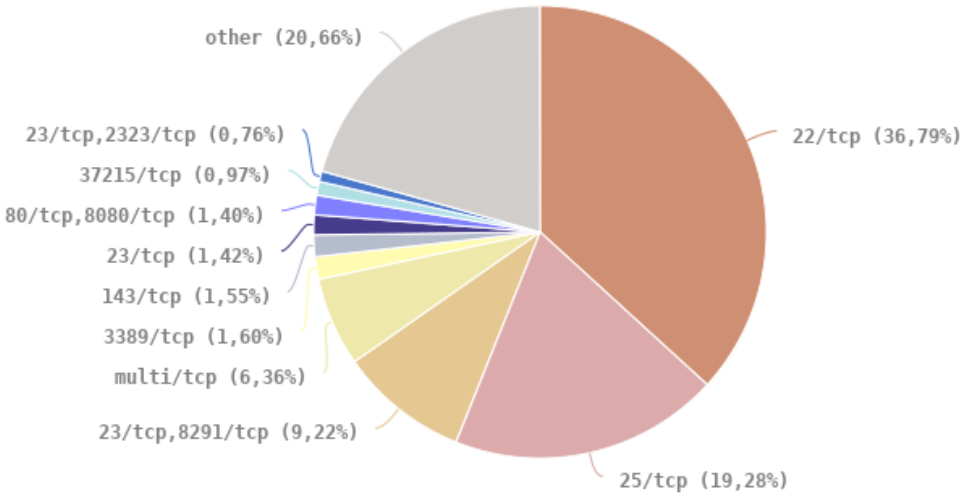


Figura 2.2: Incidentes de *scan* Reportados ao CERT.br - Janeiro a Dezembro de 2019 (Cert.br, 2019b).

consiste em uma barreira adicional que pode proteger servidores configurados de modo não apropriado em relação à segurança. Além disso, realizar o bloqueio de pacotes antes que entrem na rede interna pode melhorar a performance da rede, uma vez que uma menor quantidade de pacotes tendem a trafegar após a filtragem.

2.4 Segurança na Pilha de Protocolos de Rede

A segurança de redes não faz parte de apenas uma camada da pilha de protocolos de rede. Pelo contrário, técnicas de segurança podem ser aplicadas em todas elas. Na Figura 2.3 é apresentada a pilha contendo as camadas utilizadas como referência por Tanenbaum et al. (2011). Essa pilha é um híbrido entre o modelo de referência OSI e o TCP/IP.

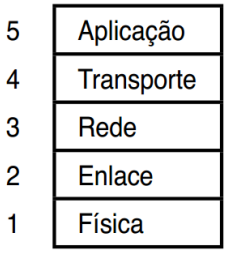


Figura 2.3: Pilha de protocolos de rede utilizada como referência por Tanenbaum et al. (2011).

Iniciando pela camada física, a segurança é realizada buscando meios para evitar "grampos" na transmissão de dados. Um método utilizado por sistemas militares é o envolvimento das fibras ópticas em tubos contendo gás inerte em alta pressão, que se rompidos, resultam na redução da pressão interna, acionando um alarme (Tanenbaum et al., 2011).

Na camada de enlace, é utilizada uma técnica chamada criptografia no nível do enlace de dados, onde os pacotes da transmissão são criptografados ao saírem de

um computador, e descriptografados ao adentrarem outro. No entanto, essa técnica não é eficiente quando os pacotes necessitam passar por diversos roteadores, pois a cada roteador é necessário descriptografar os pacotes, tornando os dados vulneráveis (Tanenbaum et al., 2011).

Na camada de rede, os *firewalls* são utilizados para prover segurança por meio da filtragem dos pacotes (Tanenbaum et al., 2011). Ademais, o protocolo IPsec é utilizado para criptografar datagramas IP buscando garantir, entre outros atributos, a integridade dos dados (KUROSE e ROSS, 2013).

Na camada de transporte, é implementada criptografia de conexões ponto -a-ponto, ou seja, processo a processo (Tanenbaum et al., 2011). Por fim, na camada de aplicação a segurança consiste na tomada de medidas para autenticar usuários e a utilização de assinaturas digitais (Tanenbaum et al., 2011).

2.5 Considerações Finais

Os administradores de rede devem se preocupar muito com segurança de seus sistemas, visto que atacantes representam os mais diversos tipos de ameaças, e novas vulnerabilidades em sistemas de software são descobertas todos os dias.

É possível imaginar um teste no qual é realizada a instalação de um servidor *ssh* em um computador com um IP público atribuído, deixando-o sem qualquer proteção (*firewall*, IDS, ou alterações no arquivo de configuração do *ssh*). Em poucos minutos, a análise dos *logs* mostrará inúmeras tentativas de conexão provenientes de diferentes lugares do mundo. Usualmente estas conexões são feitas por *bots* que "varrem" (*scan*) as redes em busca de portas abertas, e ao encontrá-las, se utilizam de técnicas de força bruta para realizar ataques.

Por fim, com a utilização de *firewalls*, é possível impedir a entrada de pacotes que realizem *scan* da rede interna, além de filtrar o acesso às portas relacionadas a serviços.

CAPÍTULO 3

Firewall

3.1 Considerações Iniciais

É paradoxal analisar a difusão contínua da internet globalmente. Apesar de benéfica, por facilitar a comunicação e disseminar conhecimento, uma rede maior consequentemente gera um aumento na quantidade de problemas de segurança.

Nos dias de hoje, independente do tamanho das empresas, todas possuem dados confidenciais, sejam eles segredos de negócio, planos de desenvolvimento, estratégias de marketing, dados financeiros, dados pessoais de clientes, etc. O vazamento desses dados podem inferir em consequências catastróficas. Portanto, é necessário que existam "barreiras" que visem a impedir o vazamento de informações valiosas, ou que dados indesejados (e potencialmente perigosos) entrem. Os *firewalls* são responsáveis por essa "barreira". Analogamente, podemos compará-los com a forma de segurança realizada em castelos medievais, onde fossos profundos eram escavados em volta, forçando qualquer entrada ou saída por uma única ponte, na qual guardas poderiam revistar quem passasse.

Neste capítulo são discutidos conceitos relacionados aos *firewalls*, e o que os tornam tão importantes e indispensáveis em ambientes de rede. Na Seção **??** são definidos os tipos de *firewalls* e apresentados conceitos de políticas e regras. Na Seção **??** são abordados os principais problemas relacionados a *firewalls*, além de dois estudos realizados com o objetivo de validar configurações de *firewalls* em empresas.

3.2 Conceitos de Firewalls

Como mencionado na Seção 2.2, a segurança operacional da informação é realizada por mecanismos responsáveis pela filtragem dos pacotes que entram e saem da rede, podendo ser tomadas três ações: registrar, aceitar, ou recusar. Esses mecanismos são conhecidos como *firewalls*, IDS (*Intrusion Detection System*) e IPS (*Intrusion Prevention System*). KUROSE e ROSS (2013) definem *firewall* como sendo a combinação de

hardware e *software* localizado na fronteira entre a rede da organização e a internet pública, isolando-os, e assim permitindo que alguns pacotes passem e outros sejam bloqueados. Além disso, ele permite que o administrador de redes possua maior conhecimento sobre o tráfego da rede.

Segundo KUROSE e ROSS (2013), os *firewalls* possuem três objetivos principais. São eles:

- Monitoramento do tráfego. Todo tráfego que entra ou sai passa necessariamente pelo *firewall*. Conforme ilustrado na Figura **??**, o *firewall* delimita a rede interna da internet pública. Embora seja possível a existência de diversos *firewalls* na rede de empresas, centralizar o tráfego em apenas um simplifica o gerenciamento e torna a rede menos suscetível a falhas.
- Filtragem do tráfego. Apenas o tráfego autorizado é aceito pelo *firewall*. Como todo o tráfego passa necessariamente pelo *firewall*, este deve possuir mecanismos que visem a regular o tráfego de entrada e saída.
- Realizar segurança de fronteira. O próprio *firewall* é imune a invasão. Pelo fato do *firewall* estar conectado diretamente à internet, é necessário que sejam aplicadas medidas rígidas de *Hardening*, pois se por ventura ele for comprometido, toda a rede interna é consequentemente exposta.

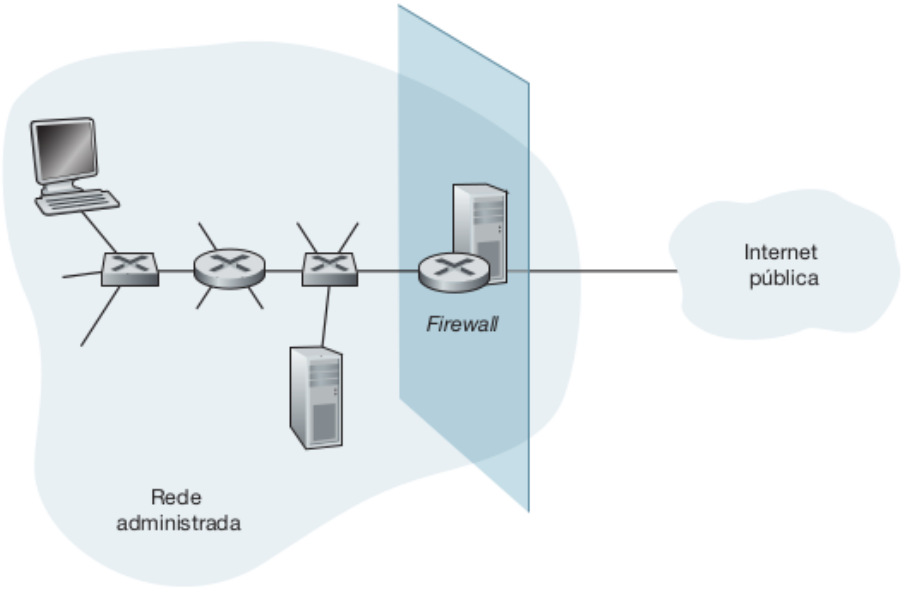


Figura 3.1: *Firewall* posicionado na fronteira entre a rede interna e a internet (KUROSE e ROSS, 2013).

No entanto, a simples instalação de *firewalls* não garante a segurança. O desafio consiste nas suas configurações, que podem ser complexas (Mayer et al., 2000). Para realizar a configuração de um *firewall*, políticas de segurança devem ser seguidas para a definição de regras.

3.2.1 Políticas e Regras

Quando um *firewall* é adquirido, o administrador de rede deve configurá-lo para que as políticas de segurança da organização sejam cumpridas. De modo geral, as políticas de segurança podem incluir diretivas de segurança gerais (como a negação de tráfego em portas utilizadas por *worms* conhecidos), além de políticas específicas que possam levar em consideração: a produtividade dos usuários (bloqueando acesso à redes sociais por exemplo), ou o consumo de largura de banda (KUROSE e ROSS, 2013).

Após a definição da política de segurança, estas devem ser traduzidas em uma lista de regras ordenadas que definem as ações realizadas em pacotes que satisfaçam determinadas condições. No entanto, a tarefa de transcrever a política de segurança em regras de linguagem nativa de *firewall*, ou tentar compreende-las em *firewalls* já configurados, é uma tarefa complexa.

Existem dificuldades que podem influenciar na capacidade do *firewall* de transcrever corretamente a política de segurança definida. Por exemplo, a existência de diversos *firewalls* na rede exige que, para que seja possível afirmar se determinado pacote pode ou não adentrar a rede, é necessário verificar a configuração de todos eles. Além disso, dentre os *firewalls* da organização podem existir modelos de diferentes fabricantes, e como usualmente cada fabricante implementa sua própria linguagem de definição de regras, é necessário que o administrador de rede domine-as. Desta forma, ferramentas que facilitem a análise e testes de configurações de *firewall* podem ajudar no processo de administração de uma rede (Mayer et al., 2000).

Algumas políticas de segurança podem ser consideradas "boas práticas" quando aplicadas em ambientes de rede. No tocante a *firewalls*, por exemplo, recomenda-se que nenhum computador da rede interna hospede servidores web (os servidores ficam situados na DMZ[1]). Esta política é traduzida em uma regra onde o acesso à porta 80 (porta padrão do HTTP) seja bloqueado para todos os IPs da rede interna. Ademais, existem *blacklists* contendo IPs conhecidos por realizarem ataques, sendo considerada boa prática rejeitar qualquer tráfego de, ou para, eles (Lihua Yuan et al., 2006).

De maneira geral, uma regra de *firewall* é composta pelos seguintes atributos: protocolo (TCP/UDP), IP de origem, IP de destino, porta de origem, porta de destino, e ação a ser executada quando um pacote satisfizer os atributos de origem e destino. É possível definir intervalos de IPs ou portas de modo que, por exemplo, seja concebível negar o acesso à porta 80 para todos os IPs da rede interna com a definição de apenas uma regra (Al-Shaer e Hamed, 2004).

Apesar da ordem nas quais os atributos de uma regra são declarados possa variar de acordo com o fabricante do *firewall*, uma regra genérica em um filtro de pacotes possui o seguinte formato: (Al-Shaer e Hamed, 2004)

```
<ordem><protocolo><ip_ori><porta_ori><ip_dest><porta_dest><ação>
```

[1]DMZs são zonas desmilitarizadas de uma rede. Elas podem, por exemplo, abranger os servidores WEB e de e-mail, pois para eles o tráfego para as portas 80 e 25 é permitido

Nesta regra genérica, respectivamente, *ip_ori* e *porta_ori* definem o IP e a porta de origem, e *ip_dest* e *porta_dest* definem o IP e a porta de destino. Na Figura **??** é ilustrado um exemplo de configuração de *firewall* centralizado. O símbolo "*" nas configurações significa "qualquer". Tomando como exemplo a primeira regra, ela indica que todos os pacotes TCP contendo como origem o IP 140.192.37.20 e em qualquer porta (*any*) devem ser bloqueados (*deny*) se eles possuírem como destino a porta 80 de qualquer (*.*.*.*) IP de destino.

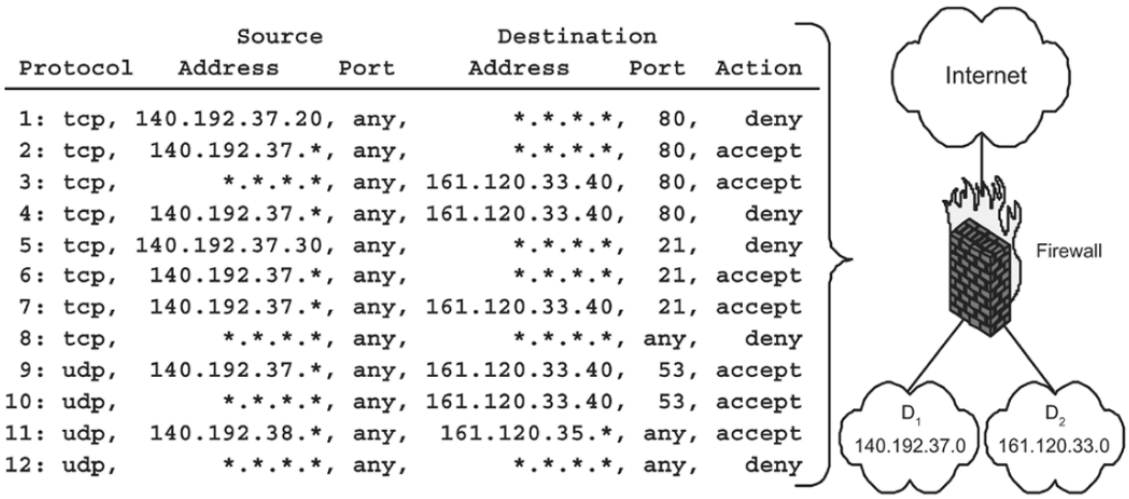


Figura 3.2: Exemplo de configuração de regras de *Firewall* (Al-Shaer et al., 2005).

É importante observar que a ordem na qual as regras são declaradas no *firewall* é relevante. Quando um pacote é recebido, este é verificado com as regras de forma sequencial. Os *firewalls* podem utilizar o processo de *first-matching* ou *last-matching*. No primeiro, o destino do pacote analisado é definido de acordo com a primeira regra que combinar com ele, sendo ignorado o restante. No segundo, o pacote é comparado necessariamente com todas as regras definidas, e a ação a ser realizada é definida pela última regra que combinar com o pacote (Lihua Yuan et al., 2006).

Existem situações nas quais o pacote recebido pelo *firewall* não combina com as regras definidas. Nesses casos, os fabricantes definem regras padrão, podendo permitir ou negar os pacotes. Apesar disso, os fabricantes permitem que uma regra padrão seja definida explicitamente. A prática recomendada é negar os pacotes não pertencentes às regra, pois dessa maneira é possível saber exatamente o que está passando pelo *firewall* (Al-Shaer et al., 2005).

3.2.2 Tipos de *Firewalls*

Os *firewalls* podem ser organizados em três categorias, são elas: filtros de pacotes tradicionais, filtros de estado de conexão, e filtros com *gateways* de aplicação.

Os filtros de pacotes tradicionais tratam cada pacote isoladamente, analisando-o contra cada regra definida, o que exige maior processamento, podendo influenciar negativamente na performance da rede (Lihua Yuan et al., 2006). Nesses filtros, cada datagrama é analisado de acordo com as regras, que podem conter, além dos atributos

genéricos de regras: o tipo de protocolo (TCP, UDP, ICMP, etc.), os bits de *flags* do protocolo TCP (SYN, ACK, etc.), e o tipo de mensagem ICMP (se houver) (KUROSE e ROSS, 2013).

Os filtros de estado de conexão, também chamados de filtro de pacotes com controle de estado, mapeiam os pacotes e criam as tabelas de estados de conexões por meio da análise dos seus cabeçalhos TCP/IP. Esta análise é realizada monitorando o processo conhecido como *three-way handshake* do protocolo TCP. Se o *firewall* receber um pacote de uma conexão que já esteja estabelecida, ele é aceito sem a necessidade de compará-lo com todas as regras. Dependendo da situação, isso pode reduzir significantemente a carga de processamento do *firewall* (Lihua Yuan et al., 2006). A utilização deste tipo de *firewall* permite que, por exemplo, um servidor externo envie pacotes para um *host* interno somente se este estabeleceu previamente uma conexão com aquele (e ela ainda estiver aberta) (Tanenbaum et al., 2011).

Por fim, os *firewalls* podem ser compostos por filtro e *gateways* de aplicação. Além de examinar os cabeçalhos dos pacotes IP, eles também analisam os seus conteúdos (Tanenbaum et al., 2011). Esses *gateways* permitem, por exemplo, autenticar usuários (ao invés de permitir e negar IPs) para utilizarem determinados serviços (como *telnet*, FTP). Normalmente a implementação dos *gateways* são realizadas de forma adjunta aos filtros, pois para cada aplicação é necessário um *gateway* diferente (KUROSE e ROSS, 2013).

3.2.3 Ambientes *Multi-firewall*

Possuir apenas um *firewall* centralizado e com regras simples é normalmente a melhor prática (Wool, 2010). No entanto, não é incomum que empresas possuam mais de um *firewall* em suas redes, seja para filtrar o tráfego de ISPs (*Internet Service Providers*) diferentes, ou para separar redes com demandas distintas de tráfego. O desafio consiste na configuração destes *firewalls*, sendo necessário que as regras sejam consistentes, pois do contrário podem ocorrer problemas no fluxo dos pacotes, além de dificultar o gerenciamento e manutenção (Lihua Yuan et al., 2006).

Na Figura **??** é ilustrada uma rede com quatro *firewalls* *W*, *X*, *Y*, *Z*. É possível observar dois ISPs (ISP A e ISP B), com os *firewalls* *W* e *X* sendo responsáveis por proteger a DMZ da internet pública. Os *firewalls* representados por *Y* e *Z* guardam a rede interna, onde geralmente as regras são mais restritivas. É importante que os pares de *firewalls* *W,X* e *Y,Z* possuam configurações condizentes entre si, de modo que um pacote não seja aceito por um e negado pelo outro, ou vice-versa. (Lihua Yuan et al., 2006)

3.3 Problemas Relacionados a *Firewalls*

Ferramentas para testes de *firewalls* apontam apenas um tipo de erro de configuração: aceitar tráfego que deveria ser rejeitado. No entanto, existe um segundo erro possível: bloquear pacotes que deveriam ser aceitos. Esses erros tipicamente são descobertos apenas quando um usuário adverte o administrador de rede do problema, podendo acarretar em prejuízos para a empresa (Mayer et al., 2000).

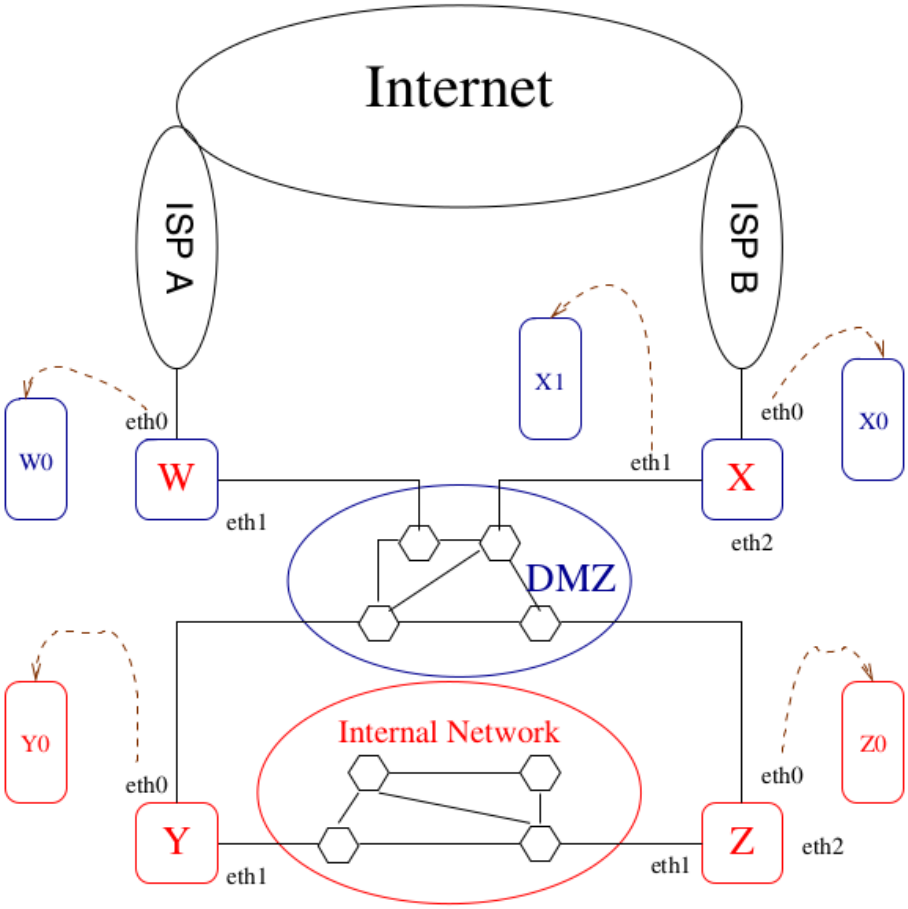


Figura 3.3: Ambiente com múltiplos *firewalls* (Lihua Yuan et al., 2006).

Firewalls mal configurados ou sem configuração alguma abrem portas para que *worms* se espalhem pela internet com facilidade e rapidamente. A vasta maioria dos *firewalls* possuem falhas em suas configurações, como apontado pelo estudo abordado na Seção **??**. Um exemplo de configuração falha é apresentado a seguir:

```
1 tcp 192.168.0.0/16 * * *    accept
2 tcp 192.168.1.0/24 * * 3127 deny
```

Se o *firewall* em análise utilizar o processo de *first-matching*, a regra 2, que rejeita o tráfego da rede 192.168.1.0/24 para qualquer (indicado pelo "*") pacote que tenha como destino a porta 3127, nunca é executada, pois a primeira regra permite a saída de todo o tráfego dos IPs 192.168.0.0/16 de qualquer porta para qualquer IP e porta. O domínio de pacotes possíveis para a regra 2 está contido no domínio da primeira regra.

Em ocasiões onde grandes empresas possuem *firewalls* distribuídos, podem existir centenas de regras escritas por diferentes pessoas em diferentes épocas. É possível inferir que isso aumenta a possibilidade de ocorrência de anomalias na rede da organização. Desta forma, ferramentas que auxiliem administradores de redes a gerenciar regras de *firewall*, analisando, resumindo e verificando-as, são importantes para a obtenção de uma rede mais segura.

3.3.1 Tipos de Erros e Anomalias

Firewalls contendo regras que não traduzam de maneira correta as políticas de segurança podem passar a falsa impressão de segurança. As falhas na tradução podem resultar em erros ou anomalias de três classes distintas, que são aprensetadas nas próximas seções (Lihua Yuan et al., 2006).

3.3.1.1 Violações de Política

Cada empresa possui políticas de segurança particulares às suas demandas, que exigem que o administrador de redes as traduzam em regras de *firewall*. Essas políticas são geralmente entregues ao administrador de redes em linguagem de alto nível, como por exemplo: os usuários não pertencentes à seção de recursos humanos não devem possuir acesso ao servidor responsável por armazenar os salários de todos os funcionários. É papel do administrador de redes transcrever essas regras para linguagens de baixo nível, ou linguagens nativas de *firewall*. Portanto, os erros de violação de política são falhas ocorridas durante o processo de tradução das políticas de segurança em regras de *firewall*, resultando em regras não condizentes com as políticas (Lihua Yuan et al., 2006).

3.3.1.2 Inconsistências

Ao contrário dos erros de violações de política, ao buscar por erros de inconsistência, não é possível validar as configurações comparando-as com um documento externo (políticas escritas ou *blacklists* e *whitelists*). A identificação de erros de inconsistências depende apenas da análise dos arquivos de configuração do(s) *firewall(s)*. Eles

podem ser de três naturezas: *intra-firewall*, *inter-firewall* e inconsistências de caminho cruzado.

Os erros *intra-firewall*, ou seja, erros que ocorrem em um *firewall* isoladamente, podem ser divididos nas seguintes categorias: (Hu et al., 2012)

- *Shadowing*. Uma regra pode sofrer *shadow* de outra quando a segunda permite um determinado tráfego que já foi negado previamente pela primeira. Dessa maneira, a segunda regra nunca será executada em *firewalls* do tipo *first-matching*. Nas configurações ilustradas na Tabela **??**, a regra *r4* sofre *shadow* de *r3*, pois *r3* aceita todo o tráfego TCP vindo dos IPs 10.1.1.* em qualquer porta, para os IPs 192.168.1.* na porta 25, porém parte deste tráfego é negado pela regra *r4*.

- *Generalization*. Uma regra B é generalização de uma regra A se os pacotes de B forem um subconjunto dos pacotes de A, porém com ação diferente. Na Tabela **??**, *r5* é uma generalização de *r4*, pois as duas indicam que os pacotes recebidos da rede 10.1.1.* são permitidos, exceto pacotes TCP com destino a rede 192.168.1.* na porta 25. Generalizações não necessariamente são erros, sendo consideradas *warnings*, pois é possível reescrever o conjunto de regras de maneira a eliminar a generalização.

- *Correlation*. Uma regra A tem correlação com uma regra B se existe um conjunto de pacotes que pertencem à intersecção de A com B, mas com ações distintas. A regra *r2* tem correlação com a *r5* no exemplo da Tabela **??**, pois qualquer pacote UDP originado de qualquer porta da rede 10.1.1.* com destino a porta 53 da rede 172.32.1.* satisfaz as duas regras. Portanto, considerando ainda um *firewall first-matching*, como a *r2* é predecessora de *r5*, todo o tráfego mencionado será bloqueado.

Tabela 3.1: Exemplo de erros de configuração de regras de um *Firewall* (Hu et al., 2012).

Rule	Protocol	Source IP	Source Port	Destination IP	Destination Port	Action
r1	UDP	10.1.2.*	*	172.32.1.*	53	deny
r2	UDP	10.1.*.*	*	172.32.1.*	53	deny
r3	TCP	10.1.*.*	*	192.168.*.*	25	allow
r4	TCP	10.1.1.*	*	192.168.1.*	25	deny
r5	*	10.1.1.*	*	*	*	allow

Com relação às inconsistências *inter-firewall*, elas não necessariamente são erros. Em ambientes com diversos *firewalls* interligados, para que pacotes cheguem aos seus destinos, é necessário que passem pelos filtros de todos os *firewalls* do caminho. Desta forma, *firewalls* mais próximos aos *hosts* podem ter configurações mais permissivas, pois eles contam com os *firewalls* mais próximos à internet pública para

implementar as políticas de segurança. Portanto, *shadowing*, generalizações e correlações não são considerados erros pois é possível definir permissões distintas para redes delimitadas pelos *firewalls* da empresa.

No entanto, existem erros específicos de inconsistência chamados *shadowed accept rules*. Eles consistem na permissão por um *firewall* interno de determinados pacotes que foram previamente bloqueados por um *firewall* externo. Desta forma, a regra do *firewall* interno nunca é executada. Apesar de não representar um erro, manter *shadowed accept rules* dificultam a compreensão das configurações globais dos *firewalls* da empresa (Lihua Yuan et al., 2006).

Por fim, em redes *multi-firewalls*, podem existir diferentes caminhos possíveis para que pacotes cheguem aos seus destinos, passando por mais de um *firewall*. Quando um pacote é bloqueado por um *firewall* através de um caminho, e é aceito por outro, é caracterizado erro de inconsistência de caminho cruzado (Lihua Yuan et al., 2006).

3.3.1.3 Ineficiências

A preocupação com a eficiência de *firewalls* deve ser considerada prioridade, pois a elevada carga de pacotes analisados podem influenciar o desempenho da rede, criando um *bootleneck*. A forma mais simples de melhorar a eficiência de *firewalls* é por meio da otimização das regras (Lihua Yuan et al., 2006).

Ineficiências são consideradas anomalias, e podem ser divididas em duas classes: redundâncias e verbosidades. As redundâncias se referem a cenários nos quais é possível realizar a remoção de determinadas regras sem que as ações tomadas para qualquer pacote sejam alteradas. A redução da redundância resulta em um número menor de regras, em *firewalls* mais simples e de entendimento mais fácil, e consequentemente diminui os recursos computacionais utilizados.

É interessante observar que regras redundantes são consideradas anomalias em redes com um único *firewall*, porém não necessariamente em redes *multi-firewalls*. Nestas, regras redundantes de permissão não são consideradas anomalias, e são necessárias para que o pacote seja entregue. Regras redundantes de negação não são necessárias, porém são consideradas boas práticas, pois uma linha de defesa adicional é constituída, sendo útil caso algum *firewall* seja comprometido.

Por sua vez, as verbosidades são anomalias que abrangem um conjunto de regras declaradas que podem ser resumidas em um conjunto menor. Elas ocorrem com frequência quando os administradores constroem as regras do *firewall* conforme necessário durante um período de tempo (Lihua Yuan et al., 2006).

3.3.2 Estudos Quantitativos de Erros de Configuração

Nos anos de 2004 e 2010, Wool (2004) e Wool (2010) publicou dois estudos com o objetivo de avaliar as configurações de *firewalls* de empresas e quantificar os erros encontrados.

No primeiro estudo (Wool, 2004), foram coletados dados entre os anos 2000 e 2001, de trinta e sete *firewalls* de organizações dos setores de telecomunicações, finanças, energia, mídia, automotivo, saúde, laboratórios de pesquisa, instituições acadêmicas e empresas de consultoria em segurança de redes.

Na Tabela **??** são ilustrados dados estatísticos coletados em relação à quantidade mínima, máxima e média de regras dos *firewalls*, objetos(*hosts* ou *subnets*) e interfaces de rede dos *firewalls*. É interessante observar a diversidade na quantidade de regras, existindo empresas que possuíam apenas 5 (cinco), até um máximo de 2.671 (duas mil seiscentas e setenta e uma). A elevada quantidade de regras pode trazer a falsa impressão de que a rede está mais segura, no entanto, como concluído pelo estudo, quanto mais simples (menor quantidade) forem as regras, as redes estarão menos suscetíveis a erros.

Tabela 3.2: Propriedades estatísticas dos dados coletados por Wool (2004).

Propriedade	Mínimo	Máximo	Média
Número de regras	5	2.671	144
Número de objetos	24	5.847	968
Número de interfaces	2	13	4,1

Para definir quais configurações seriam considerados erros, as regras foram comparadas com doze políticas de configuração de segurança que são consideradas boas práticas. Dentre elas, destacam-se:

- Qualquer tráfego direcionado diretamente ao *firewall* deve ser negado.
- O *firewall* não deve permitir acesso remoto por meios não criptografados, como *telnet*.
- O número de pessoas que possuem acesso ao *firewall* deve restrito. Foi definido um limite de cinco pessoas.
- O *firewall* não deve ser acessado remotamente por meio da rede externa. Qualquer acesso remoto deve ser realizado por meio da rede interna ou utilizando VPN (*Virtual Private Network*).
- Não deve existir uma regra permitindo "*any*" portas no tráfego de entrada.

Os resultados obtidos mostraram que todos os trinta e sete *firewalls* analisados apresentavam algum tipo de erro de configuração. Quase 80% deles permitiam tráfego de entrada para qualquer porta e acesso remoto não criptografado. Através da análise da quantidade de erros em cada *firewall*, e os comparando com a quantidade de regras, Wool (2004) concluiu que os *firewalls* que continham uma menor quantidade de regras apresentaram os menores índices de erros de configuração.

No segundo estudo, Wool (2010) analisou oitenta e quatro *firewalls*, com dados coletados entre os anos de 2003 e 2005, provenientes de empresas do setor de telecomunicações, finanças, energia, mídia, automotivo e saúde.

Desta vez, Wool (2010) considerou trinte e seis erros que representam ameaça à segurança. Os erros foram divididos em quatro classes: tráfego de entrada, tráfego de saída, tráfego interno, e regras inerentemente perigosas. Nelas são descritos erros que

representam potenciais ameaças em portas padrões para serviços como FTP, Telnet, TFTP, HTTP, DNS e SMTP.

Como resultado, na avaliação das regras de entrada, mais de 45% dos *firewalls* permitiam que os serviços de DNS, FTP ou SMTP atingissem mais de 256 IPs da rede, o que não é aconselhável. Com relação ao tráfego de saída, mais de 80% permitiam que pacotes SMTP saíssem da rede por mais de 256 IPs, e mais de 60% permitiam tráfego *peer-to-peer*, o que raramente é utilizado em ambientes corporativos. Por fim, Wool (2010) concluiu que, assim como em seu estudo de 2004, quanto menor a quantidade de regras e menos complexas elas forem, menores são os números de erros de configuração.

3.4 Considerações Finais

Tendo como base as definições relacionadas a *firewalls* apresentadas na Seção **??**, é possível observar que apesar dos *firewalls* comporem apenas uma das vertentes da segurança entre redes, eles podem ser implementados de diferentes formas. O nível de complexidade de ambientes *multi-firewalls* de tipos e fabricantes diferentes é alto, resultando em um crescimento na probabilidade de ocorrências de erros e anomalias.

Além de violações de políticas, isto é, problemas na tradução das políticas de segurança em regras de *firewall*, as inconsistências e ineficiências são muito comuns, principalmente em regras implementadas com o decorrer do tempo. Desta forma, é importante que as configurações sejam realizadas da forma mais sucinta possível, pois como foram mostrados nos estudos de Wool (2004) e Wool (2010), quanto mais simples forem as configurações, menores são as chances de ocorrência de erros e anomalias.

CAPÍTULO 4

Trabalhos Relacionados

4.1 Considerações Iniciais

Neste capítulo são apresentados trabalhos disponíveis na literatura relacionados a políticas de segurança e *firewalls*. Os trabalhos descrevem conceitos de modelagem das políticas de seguranças e propõem ferramentas que auxiliem os administrador de redes no gerenciamento dos *firewalls*. Nas Seções 4.11 e 4.12 são descritas, respectivamente, as linguagens SPML e SPML2, a serem seguidas na linha de pesquisa deste trabalho.

4.2 *Firmato*

No trabalho de Bartal et al. (1999) é apresentada uma plataforma chamada *Firmato*. Os autores a propõem com o objetivo de auxiliar no gerenciamento de *firewalls* por meio de um modelo entidade-relacionamento para representar as políticas de segurança e a topologia da rede.

Na Figura 4.1 é ilustrado o modelo entidade-relacionamento definido. Com relação às políticas de segurança, elas tem como base a entidade *Role*, composta por um conjunto de *Peer Capabilities*. Cada *Capability* define os serviços permitidos, a direção do tráfego, e os *peers* (IPs de origem e destino). A entidade *Service* consiste no protocolo base (TCP, UDP ou ICMP) e as portas de origem e destino. A topologia da rede é definida por *Zones*, conectadas por *Gateways* - nesses *gateways* são instalados os *firewalls*. Estes possuem uma ou mais *Gateways-Interface*, e por meio do atributo *Adjacent Zone* são realizadas as conexãos entre *Zones*. As *Zones* consistem em *Host Groups*, que podem definir outros *Host Groups* ou uma lista de *Hosts*.

No trabalho também é apresentada uma linguagem para definir instâncias dos modelos entidade-relacionamento (*Model Definition Language*), e um compilador (*Model Compiler*) responsável por transcrever as definições da linguagem em código de configuração de *firewall*. Por fim, é apresentada uma ferramenta gráfica que traduz

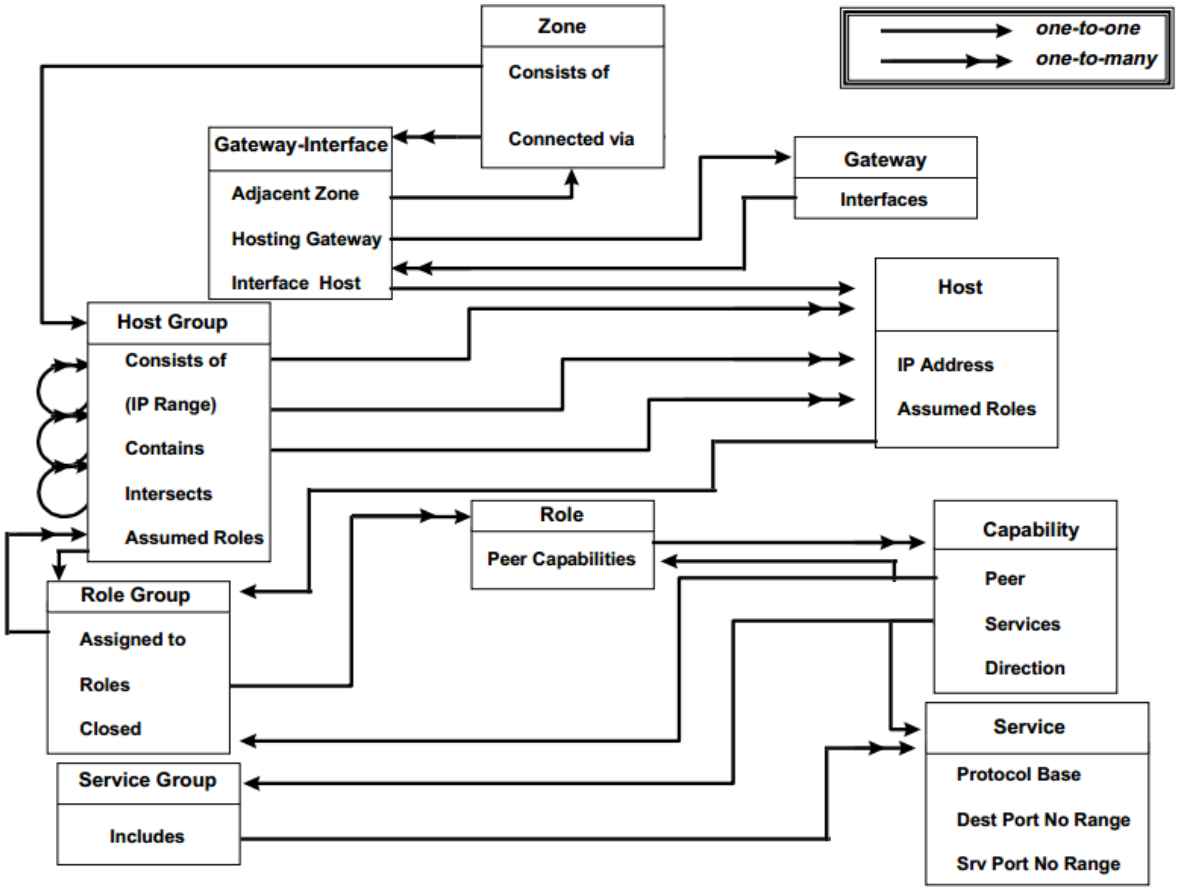


Figura 4.1: Modelo entidade-relacionamento de Bartal et al. (1999).

as saídas do compilador em uma representação visual baseada em grafos das políticas e topologia. O trabalho é concluído validando os processos descritos em um ambiente realista.

4.3 *Fang*

Mayer et al. (2000) apresentam um módulo adicional (porém independente) da *Firmato* (Bartal et al., 1999), chamado de *Fang: A Firewall Analysis Engine*. Ele permite ao administrador de redes compreender e validar as políticas globais de *firewall*, além de ser possível testar as configurações mediante ataques de *spoofing*. Inicialmente, a topologia da rede é instanciada utilizando a linguagem de modelagem da Firmato, e também são carregados os arquivos de configurações escritos em linguagem nativa de *firewall*.

Fang transforma cada arquivo de configuração em uma tabela de regras lógicas, representada pela seguinte estrutura:

```
struct rule {
        struct hostgrp *source;
        struct hostgrp *dest;
        struct servicegrp *service_grp;
        direction_ty direction;
```

```
        action_ty action;
};
```

Quando os pacotes são filtrados, eles são comparados sequencialmente com as regras lógicas da tabela até que ocorra um *match*. Os atributos *source* (origem), *dest* (destino) e *service grp* (serviço) são comparados aos campos correspondentes no pacote filtrado. *Direction* indica se a regra aplica-se a pacotes que entram ou saem do *firewall*. Se um *match* ocorre, a *action* correspondente (aceitar ou bloquear) é executada.

O *software* se baseia em *queries* para realizar a interação com o usuário. Cada *query* é composta por um *host-group* de origem, um *host-group* de destino, e um *service group*. As *queries* construídas podem seguir a semântica genérica: "quais IPs do *host-group* de origem podem enviar pacotes do *service group* para quais IPs do *host-group* de destino?".

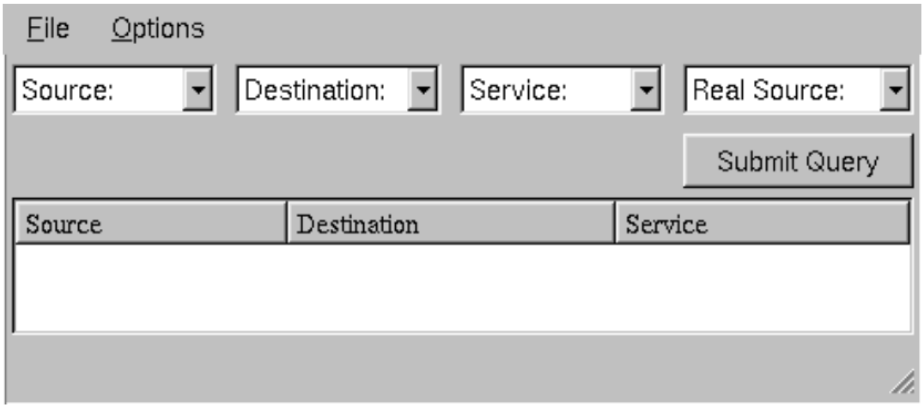


Figura 4.2: Interface da *Fang* (Mayer et al., 2000).

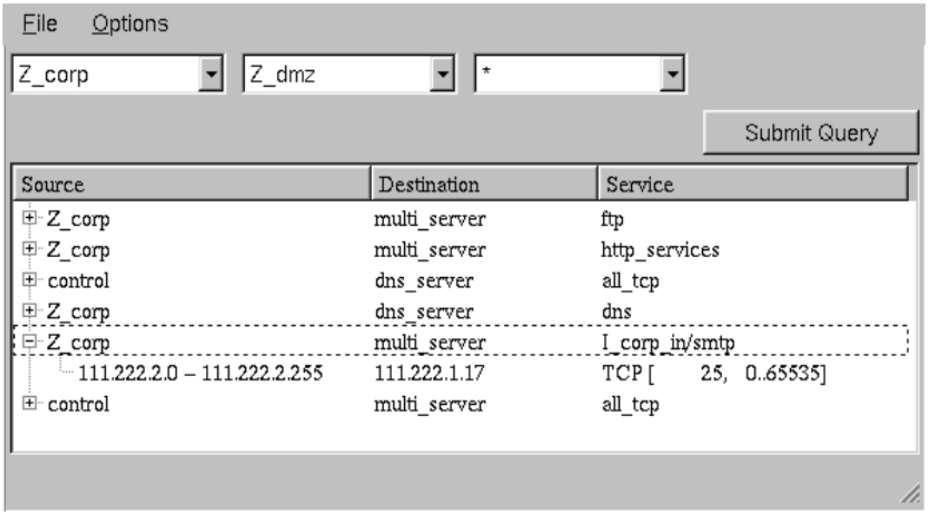


Figura 4.3: *Fang* - Resultado de uma *query* (Mayer et al., 2000).

Nas Figuras 4.2 e 4.3 são ilustrados exemplos da interface da *Fang*. Na primeira são apresentados os campos *Source*, *Destination*, *Service* e *Real Source*. Este último é utilizado para teste de ataques de *spoofing*. Na segunda figura são apresentados

os resultados para uma *query* exemplo. Nela, a seguinte *query* é realizada: "quais serviços são permitidos entre a rede *Z_corp* (um *hostgroup*) e a rede *Z_dmz* (outro *hostgroup*)?". Na saída são listados os serviços disponíveis, sendo interessante observar que na coluna *Source* o *hostgroup control* é um *hostgroup* contido no *hostgroup Z_corp*. O mesmo ocorre com os *hostgroups* da coluna *Destination*, onde todos listados representam *hostgroups* contidos no *hostgroup Z_dmz*.

De forma sucinta, a *Fang* realiza a leitura dos arquivos contendo a topologia da rede e as configurações dos *firewalls* existentes, fornecendo uma visualização textual das interações entre eles. Ela possui relevância para administradores de redes validarem configurações existentes.

4.4 FPA - *Firewall Policy Advisor*

No software chamado *Firewall Policy Advisor*, Al-Shaer e Hamed (2004) apresentam técnicas e algoritmos para identificar, de maneira automática, anomalias em *firewalls* centralizados ou distribuídos. No trabalho é proposto um método de modelagem de regras de *firewall* utilizando árvores, como ilustrado na Figura 4.4. É possível observar a árvore com raiz única, dividindo inicialmente as regras de acordo com o protocolo (TCP ou UDP). Posteriormente as ramificações da árvore dividem as regras, progressivamente, de acordo com o IP de origem (*src_ip*), porta de origem (*src_port*), IP de destino (*dst_ip*), porta de destino (*dst_port*) e ação (*action*).

Para identificar anomalias nas configurações, inicialmente são especificados os tipo de relações entre regras. Elas podem ser: *completely disjoint, exactly matching, inclusively matching, partially disjoint*, ou *correlated*. Por meio destas especificações, e utilizando a árvore de configurações do *firewall*, são apresentados algoritmos para encontrar anomalias intra (*shadowing*, *correlation*, *generalization*, *redunancy*, *irrelevance*) e *inter-firewall*. Tomando como exemplo as regras 2 e 3 da árvore contida na Figura 4.4, os triângulos contendo o número 4 em ambas as regras indicam que existe uma relação de *shadowing* entre elas, pois o conjunto de pacotes abrangentes pela regra 2 é um subconjunto da regra 3.

No software também é apresentado um algoritmo que indica a ordem correta para a inserção ou remoção de uma determinada regra. Apesar de ser possível descobrir anomalias e erros nas configurações, a ferramenta não indica possíveis correções.

4.5 *FIREMAN*

Lihua Yuan et al. (2006) apresentam um *toolkit* chamado *FIREMAN*. No trabalho é descrita uma representação das regras de *firewalls* centralizados por meio de grafos. Essa representação é ilustrada na Figura 4.5, na qual a porção mais à esquerda é mostrado um diagrama de regras em cadeia. Elas são utilizadas pelo *firewall* do *Linux Netfilter*, em que é possível definir conjuntos de regras que funcionam de maneira análoga a funções em linguagens de programação (com chamadas e retornos). Na porção à direita da figura são ilustrados todos os caminhos possíveis de um pacote dentro das regras definidas.

Em ambientes *multi-firewall*, os *firewalls* e suas interfaces são representadas uti-

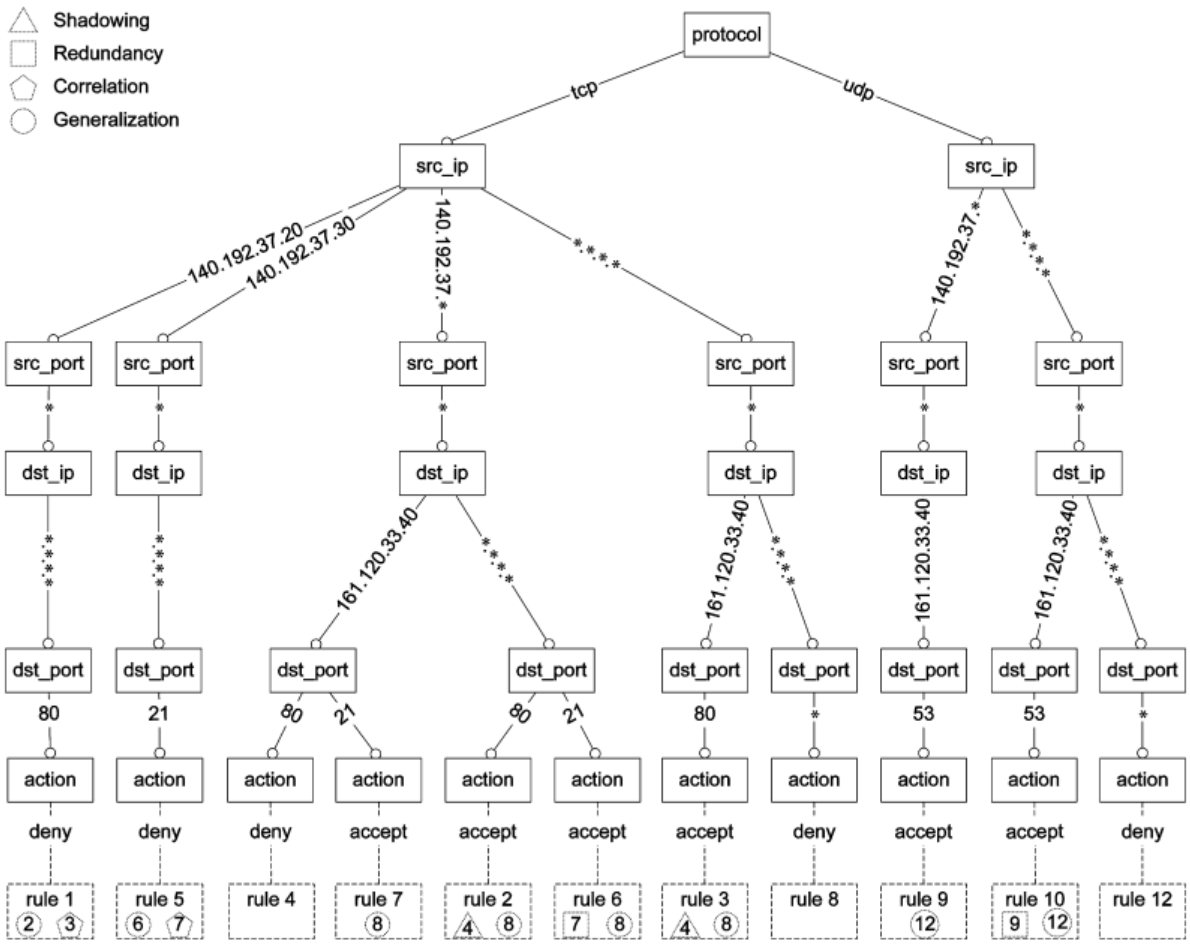


Figura 4.4: *Firewall Policy Advisor* - árvore das regras de um *firewall* (Al-Shaer e Hamed, 2004).

lizando árvores chamadas *ACL-trees*. Na Figura 4.6 são ilustradas, da esquerda para a direita, a arquitetura de um ambiente com quatro *firewalls* (W, X Y e Z), e a respectiva *ACL-tree* representando-os. As regras de cada *firewall* nesses ambientes são modeladas isoladamente, seguindo a representação em grafos mencionada.

As buscas por anomalias e erros em *firewalls* são realizadas dividindo as regras em dois grupos: regras de permissão e de negação. Em ambas, são apresentadas definições de relações entre regras utilizando teoria de conjuntos, classificando-as em: *good rule*, *shadowed rule*, *redundant rule*, *generalized rule*, e *correlated rule*. Utilizando as definições de anomalias e a modelagem das regras, é proposto um algoritmo capaz de apontar os erros e anomalias encontradas. No entanto, o software não realiza ou sugere correções. Além disso, ele possui foco em análise de *firewalls* já implantados, não provendo ferramentas para a modelagem inicial.

4.6 *OpenSec*

Lara e Ramamurthy (2016) apresentam uma ferramenta chamada *OpenSec*. Ela utiliza o conceito de SDN - *Software Definied Networks* (Nadeau e Gray, 2013). As SDNs permitem que controladores centralizados realizem as configurações dos ativos de rede (*switches*, por exemplo) de forma automatizada, não necessitando que cada equi-

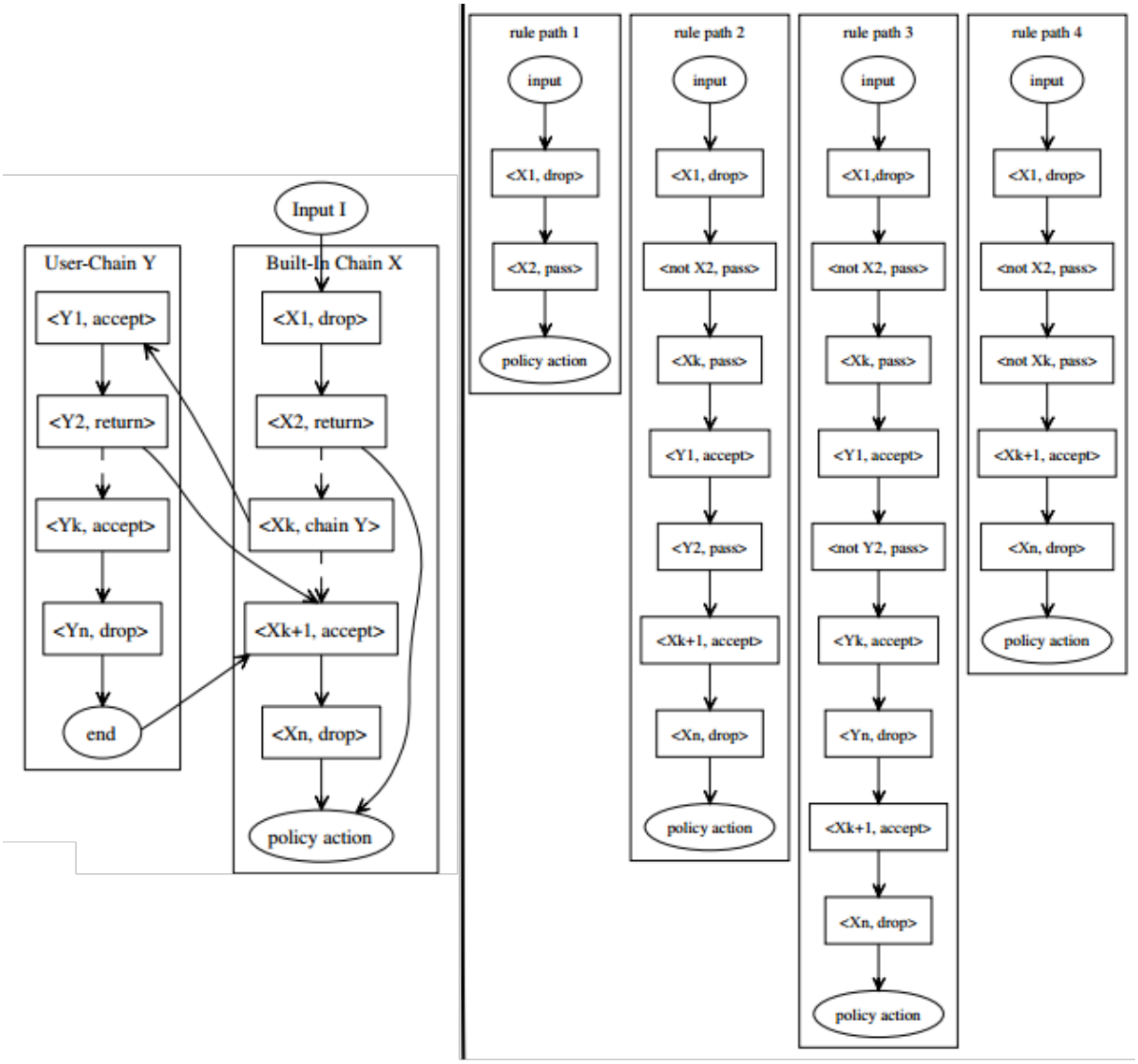


Figura 4.5: *FIREMAN* - modelagem das regras e grafos dos caminhos possíveis (Lihua Yuan et al., 2006).

pamento seja configurado manualmente. A *OpenSec* é proposta como esse controlador centralizado, sendo capaz, entres outras funcionalidades, de direcionar tráfego e enviar configurações de filtragem de pacotes para os ativos da rede.

Por meio da *OpenSec* é possível criar políticas de segurança utilizando uma linguagem de alto nível apresentada no trabalho. Essas políticas são traduzidas pela *OpenSec* em regras e enviadas para a *switch* correta por meio da utilização do protocolo *OpenFlow* (McKeown et al., 2008). O *OpenFlow* permite a comunicação da *OpenSec* com as *switches* da rede, sendo possível enviar configurações e comandos referentes à filtragem de pacotes.

Uma arquitetura genérica de uma rede utilizando a *OpenSec* é ilustrada na Figura 4.7. A *OpenSec* não realiza a análise do tráfego. Ela conta com unidades de processamento externas (*Processing Units* na figura), como *firewalls*, IPS (*Intrusion Prevention System*) e DPI (*Deep Packet Inspection*). A ferramenta implementa um gerenciador de unidades de processamento, mantendo uma lista destas unidades e as

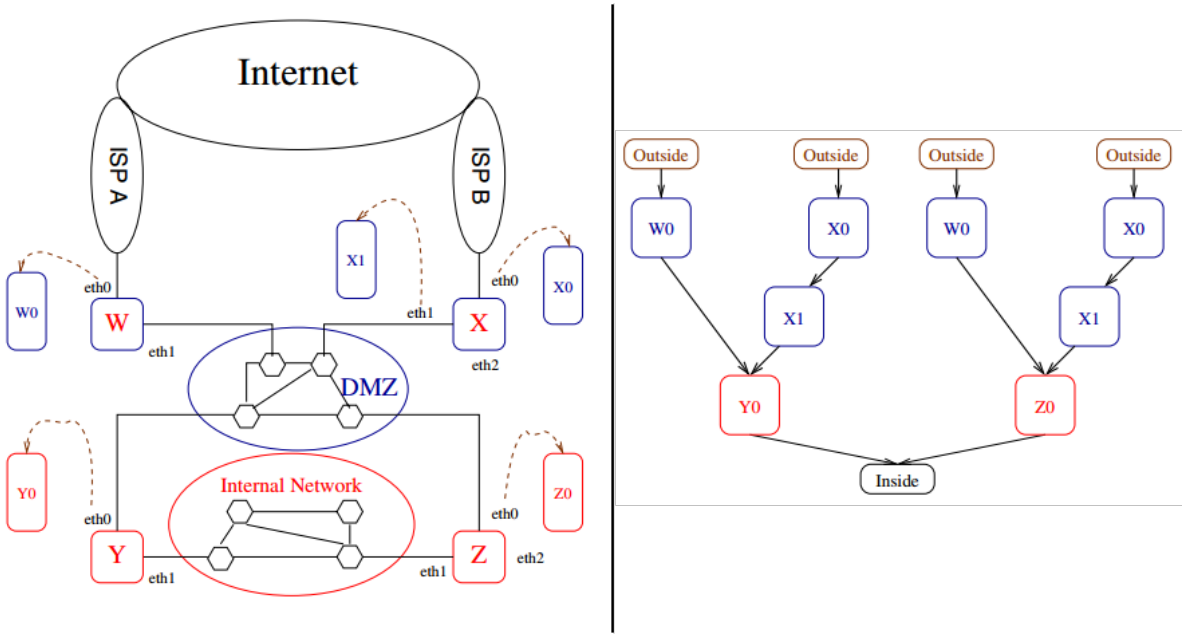


Figura 4.6: *FIREMAN* - ambiente *multi-firewall* e *ACL-tree* correspondente. (Lihua Yuan et al., 2006).

suas localizações na rede. Utilizando *OpenFlow*, a *OpenSec* espelha e encaminha os tráfegos específicos para cada unidade de processamento realizar as análises condizentes. Quando uma unidade de processamento identifica uma anomalia, a *OpenSec* é notificada do problema. Por meio desta notificação, a *OpenSec* é capaz de bloquear automaticamente, por meio do *OpenFlow*, todo o tráfego anômalo notificado enviando comandos para o roteador de entrada da rede, sem a necessidade de intervenção humana.

Com relação a topologia da rede, a ferramenta se situa entre a internet pública e a rede interna, possuindo elementos, ilustrado na Figura 4.7, que são responsáveis por: realizar a comunicação com os ativos de rede (por meio do *OpenFlow*), processar os eventos de segurança reportados pelas *Processing Units* e gerencia-las, e gerenciar as políticas de segurança definidas na linguagem.

Apesar da linguagem para definição de políticas apresentada ser considerada de alto nível pelos autores, ela ainda exige uma curva de aprendizagem, e por se tratar de políticas definidas em texto, pode se tornar difícil obter uma visão geral das políticas quando elas são muito numerosas.

4.7 FWS - *Firewall Synthesizer*

O trabalho apresentado por Bodei et al. (2018) consiste na sintetização de políticas de segurança utilizando uma linguagem independente de plataforma. Por meio da ferramenta proposta, chamada FWS (*FireWall Synthesizer*), as regras de *firewalls* como *iptables* (netfilter, 2019) e *ipfw* (FreeBSD, 2019) são traduzidas em uma especificação abstrata que as representa. Essa tradução é realizada por meio de uma linguagem intermediária proposta.

Segundo os autores, a abstração das regras torna possível a migração de um mo-

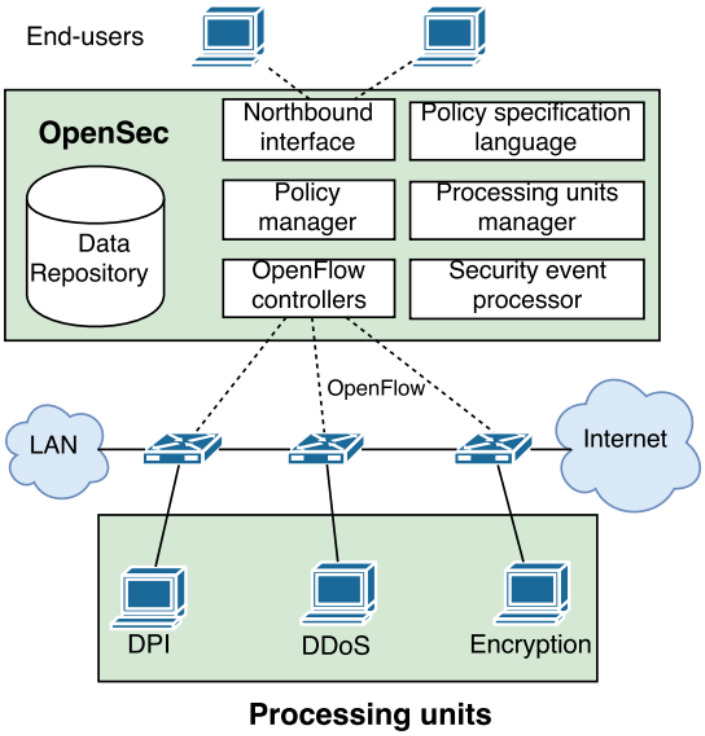


Figura 4.7: Arquitetura de uma rede utilizando *OpenSec* (Lara e Ramamurthy, 2016).

delo de *firewall* para outro, pois a FWS é capaz de traduzir as regras na linguagem intermediária para regras nativas de determinados *firewalls*. O funcionamento dela é parecido à *Fang*, onde a interface com o usuário é realizada por meio de *queries*. Na Figura 4.8 é ilustrada uma *query* e a subsequente resposta do FWS em tabela. Assumindo que as configurações de um *firewall* tenham sido carregadas na ferramenta, na *query* é verificado se a rede 10.0.1.0/24 pode iniciar conexões tendo como destino a rede 10.0.2.0/24, e também o contrário, isto é, se 10.0.2.0/24 pode iniciar conexões com 10.0.1.0/24. Na tabela resultante é informado que as regras satisfazem ambas as condições.

```
( (srcIp == 10.0.1.0/24 && dstIp == 10.0.2.0/24) ||
  (srcIp == 10.0.2.0/24 && dstIp == 10.0.1.0/24) )
&& state == NEW
```

Source IP	Source Port	Destination IP	Destination Port	Protocol	State
10.0.2.0/24	*	10.0.1.0/24	*	*	NEW
10.0.1.0/24	*	10.0.2.0/24	*	*	NEW

Figura 4.8: Exemplo de *query* e saída do FWS (Bodei et al., 2018).

Adicionalmente, a FWS permite comparações entre configurações, indicando equivalência (se uma regra aceitando um pacote é igual a outra ou a um subconjunto dela), diferença (se pacotes são aceitos por uma regra e negados pela outra) e relação (se configurações afetam o processamento dos pacotes identificados por uma *query* informada pelo usuário).

Apesar da FWS ser relevante para visualizar as configurações por meio de consultas, ela requer a aprendizagem da linguagem intermediária para realizar consultas.

Além disso, é possível visualizar apenas quais pacotes são permitidos pelas configurações carregadas, não provendo meios para responder a consultas como: “quais redes não possuem acesso ao servidor web?”. Por fim, a FWS não conta com uma interface gráfica, o que pesa negativamente em relação a parâmetros de facilidade de uso.

4.8 PolicyVis

No trabalho de Tran et al. (2007) é apresentada a ferramenta *PolicyVis*. Seu objetivo é melhorar a compreensão e inspeção de regras de *firewall*. Ao contrários de outros trabalho, ela não utiliza *queries* textuais, mas sim uma abordagem visual, representando as regras por meio de matrizes, e apresentando-as em gráficos ao usuário.

A representação é realizada utilizando filtros para selecionar determinados tipo de tráfegos. Na Figura 4.9 é apresentando um exemplo em que são mostradas as regras permitidas que possuam a porta 22 como destino. No eixo x são representados os IPs destinos que possuem a porta 22 como destino, enquanto no eixo Y são mostrados os IPs de origem.

Figura 4.9: Exemplo de gráfico gerado pela *PolicyVis* (Tran et al., 2007).

Além de permitir a visualização por meio de gráficos, a ferramenta também pode ser utilizada para identificação de anomalias nas regras. Na Figura 4.10 é possível observar um exemplo em que para o IP destino *161.120.33.45* e porta destino 20, existem duas regras que se sobrepõem, o que caracteriza uma generalização. Por meio da ferramenta também é capaz de identificar correlação, redundância e sobreposição.

A *PolicyVis* não informa ao usuário onde está a anomalia, ficando a cargo dele identifica-las por meio das intersecções de regras apresentadas no gráfico. Além disso, a ferramenta não oferece uma visão geral das regras e nem permite edita-las ou traduzi-las de qualquer maneira.

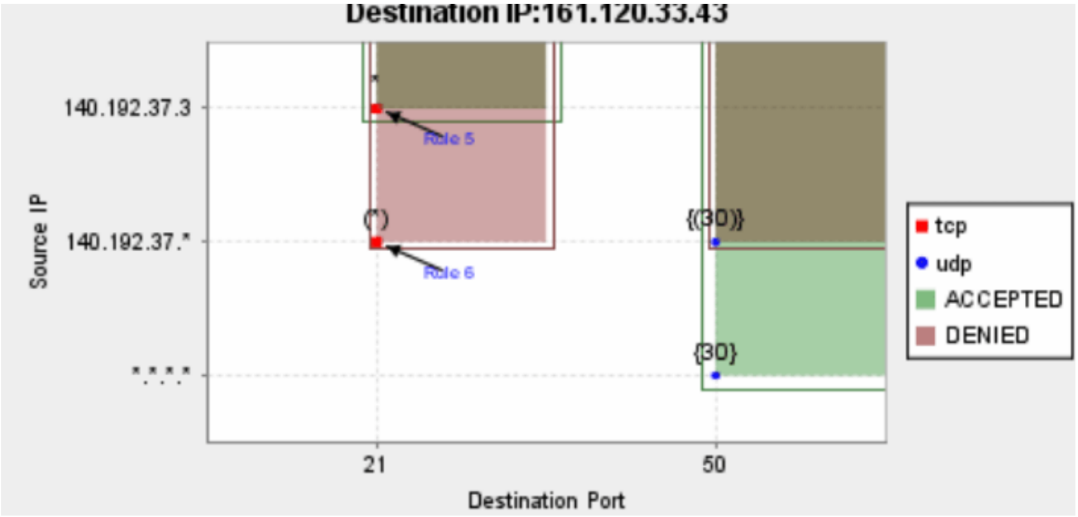


Figura 4.10: Exemplo de anomalia na *PolicyVis* (Tran et al., 2007).

4.9 F/Wvis

No trabalho de Kim et al. (2021) é proposto um método de visualização hierárquico em 3D para gerenciamento e análise das regras de *firewall*. Também é apresentada a ferramenta F/Wvis.

A ferramenta utiliza dois tipos de dados para processamento: as regra do *firewall*, e o tráfego que está passando por ele.

Para representar as regras, é utilizado um modelo tridimensional composto por duas camadas, similar a um cilindro. A superior (a face superior do cilindro) indica as anomalias encontradas, geradas automaticamente pela ferramenta. Na camada inferior (o restante do cilindro) as regras são representadas por pequenas figuras geométricas (esferas e cubos por exemplo), e posicionadas de acordo com uma função desenvolvida que leva em consideração o IP e a porta, e o *hitCount* da regra, ou seja, quantas vezes o tráfego que está passando pelo *firewall* combinou com a regra.

Na Figura 4.11 é ilustrada a principal interface da ferramenta. Ela conta com opções de busca de filtragem (A, B e C na figura), para realizar buscas específicas por IP e porta por exemplo. Na camada superior, indicada por D estão contidas as anomalias encontradas, sendo possível visualizar as regras que a estão causando e altera-las, se necessário. Abaixo da camada superior, está localizado o plano tridimensional onde as regras são representadas (E). No componente indicado por F são apresentados os detalhes das anomalias, e em G são configuradas opções de visualização.

Apesar da ferramenta indicar as anomalias automaticamente e com clareza de detalhes, a representação das regras não é intuitiva, sendo praticamente impossível localizar uma regra na camada sem a utilização do filtro. Além disso, a ferramenta não permite a modelagem inicial das políticas de segurança, sendo focada em *firewalls* que estejam em produção.

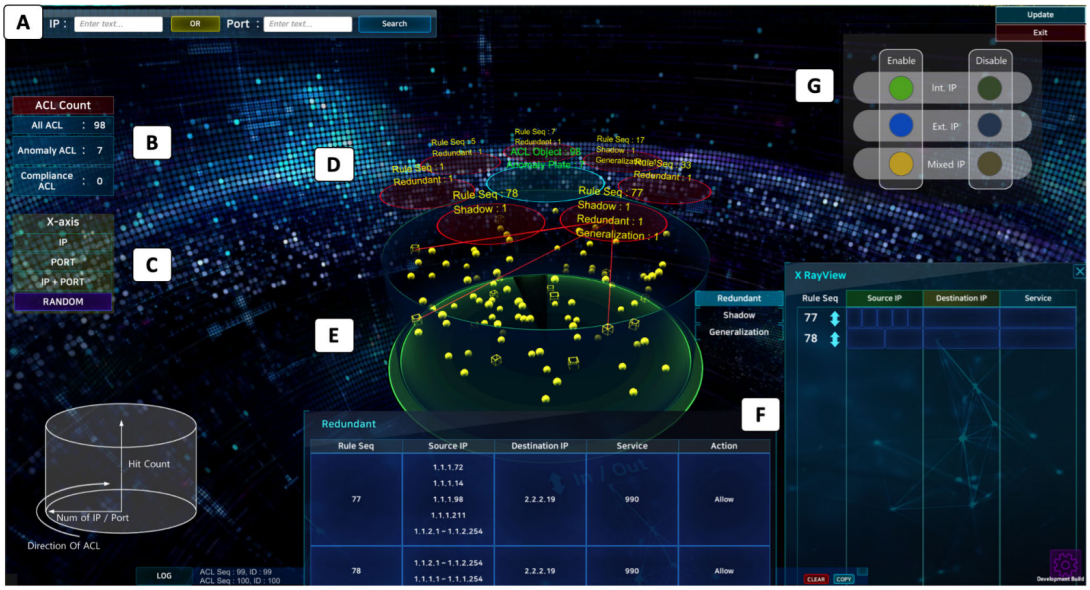


Figura 4.11: Exemplo da interface da F/Wvis (Kim et al., 2021).

4.10 *Visual Firewall Editor*

No trabalho de Mansmann et al. (2012) é apresentado um método de visualização hierárquica utilizando *sunburst* para compreensão das regras. A ferramenta apresentada, chamada *Visual Firewall Editor*, permite a visualização das regras de *firewall*, agrupando-as em ACLs (*Access Control Lists*, listas de controle de acesso), por meio de características comuns entre regras.

A *Visual Firewall Editor* permite a visualização em três componentes: *sunburst* para a visualização da hierarquia, editor com a representação das regras em texto, e visualização da árvore de ACLs criada por meio de *tree map*.

Na Figura 4.12 é apresentada a interface principal da ferramenta. No gráfico *sunburst* é possível observar o agrupamento de regras em ACLs, seguidas pelas ações de *permit* ou *deny*, ramificando até a composição completa da regra. No painel à esquerda podem ser observadas: o editor da ACLs criadas, os *tree maps* criados para representar as regras, sendo possível navegar pelas suas ramificações.

A *Visual Fireall Editor* oferece uma visão geral das regras do *firewall*, no entanto não fornece meios de modelar as regras ou traduzi-las de e para linguagem nativa de *firewall*, sendo seu funcionamento destinado à compreensão das regras.

4.11 SPML e SOH

No trabalho em que a SPML é apresentada, mencionada na Seção 1.1, Trevisani e Garcia (2008) a descrevem como uma notação gráfica para modelagem de políticas de segurança. Por meio dela as políticas são modeladas utilizando elementos gráficos, o que permite representar as regras em alto nível.

Os elementos gráficos podem representar quatro classes de componentes: componentes de *firewall*, sendo possível representar o *firewall* e suas interfaces de rede; componentes de filtragem, utilizados para representar os tráfegos de entrada e saída;

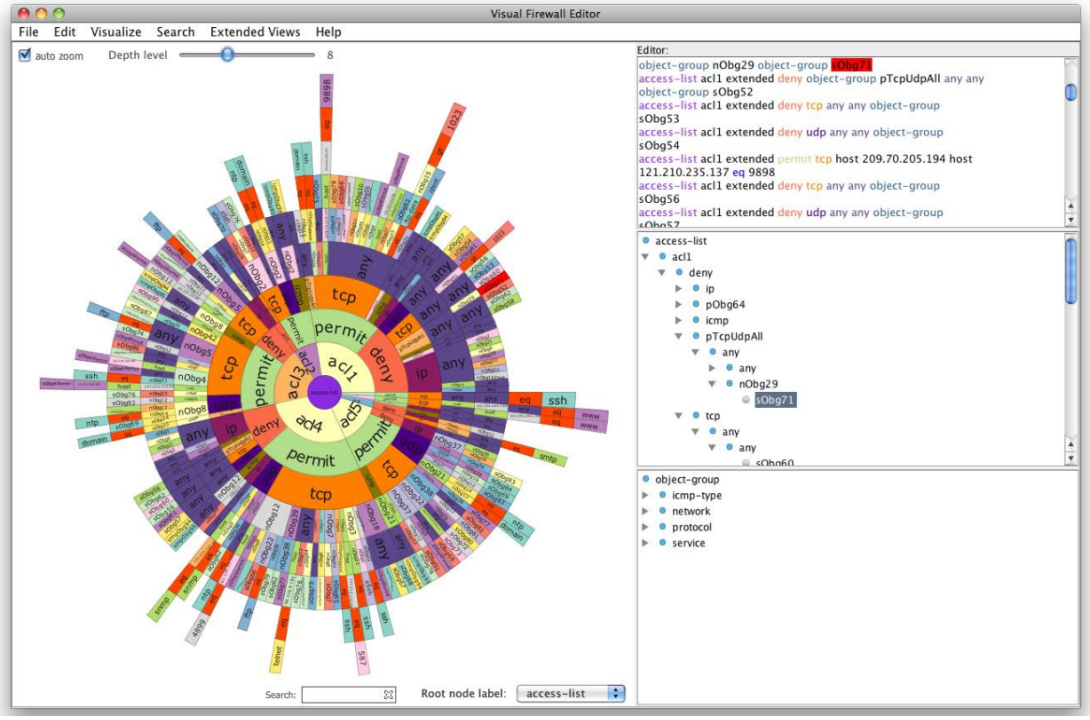


Figura 4.12: Exemplo da interface da *Visual Firewall Editor* (Mansmann et al., 2012).

componentes de tradução, para representar NAT (*Network Address Translation*) e RDR (*Service Redirecting*); e entidades externas, para representar uma rede, um *host* ou uma lista deles, e a internet pública.

Na Figura 4.13 é apresentado um exemplo da modelagem utilizando os componentes citados. O círculo interno representa o *Firewall* e os quatro quadrados que o intersectam representam suas interfaces de rede (ext, adm, dmz e acd). As linhas com setas que iniciam a partir das interfaces de rede representam tráfegos de saída, e as com destino nas interfaces de rede representam tráfegos de entrada. Os elementos mais externos são entidades externas, as quais representam: *Inet* - internet pública, *Adm Net* - uma rede, *squid* - um *host*, *Profs* - uma lista de *hosts*, e *Academic Net* - um rede.

Esta modelagem abstrai as políticas de segurança de uma rede fictícia de uma universidade, na qual é definido que a rede acadêmica deve estar isolada da rede da administração, e todos os usuários devem possuir acesso limitado à internet (por conteúdo, por exemplo), exceto os professores. Tomando como exemplo o fluxo necessário para que a rede acadêmica (*Academic Net*) acesse a internet, por meio da figura é possível observar que o fluxo chamado *WebStudent* com origem na entidade externa *Academic Net* possui um *Flow ID* (*FID*, utilizado para identicar o fluxo) de "1", e tem como destino a interface de rede *acd*. Esse fluxo é então encaminhado para a interface de rede *dmz*, da qual origina-se o fluxo *WebProxyIn* com os *FIDs* "1,4" atribuídos, onde "1" representa o fluxo proveniente da rede acadêmica (fluxo *WebStudent*) e "4" o da rede administrativa (fluxo *WebAdm*). O *WebProxyIn* tem como destino o *host squid*, responsável por realizar as limitações na internet definidas pela política. O fluxo *Web-*

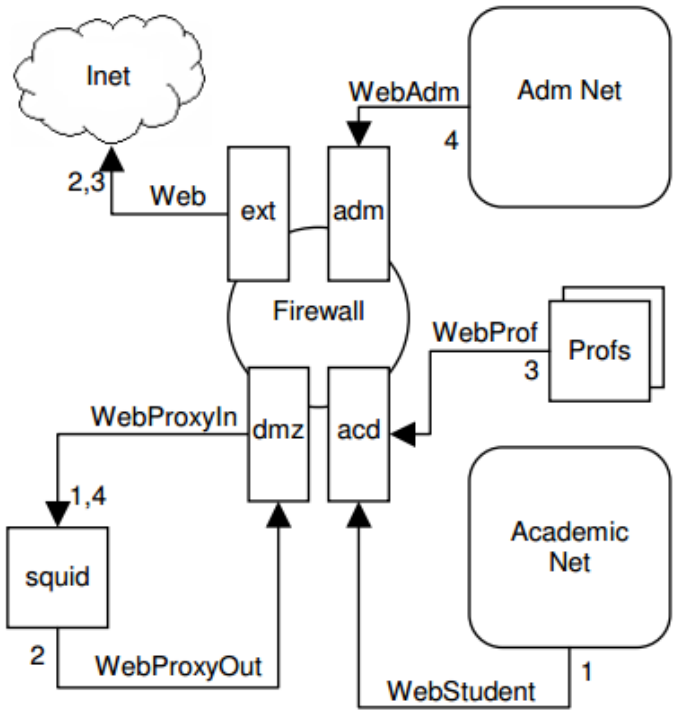


Figura 4.13: Exemplo de diagrama SPML (Trevisani e Garcia, 2008).

ProxyOut, com origem no *squid* e *FID* "2", retorna à interface de rede *dmz*, e por fim esta o encaminha para a interface de rede *ext*, da qual origina-se um fluxo de saída com destino à internet que possui o fluxo de *FID* "2" atribuído.

Além de permitir que as políticas de segurança sejam modeladas de forma gráfica e amigável por meio da SPML, os autores também apresentam uma ferramenta chamada SOH (*Security on Hands*) capaz de editar diagramas SPML, traduzi-los para documentos XML, e por fim traduzir estes documentos em configurações em linguagem nativa de *firewall*.

A utilização da SPML e da SOH permitem, de maneira simples, representar as políticas de *firewall* sem que seja necessária a aprendizagem das linguagens de *firewalls* de diferentes fabricantes. Além disso, por utilizar diagramas nas definições de políticas, é possível obter uma visão geral delas sem a necessidade de analisar textos de configurações.

4.12 SPML2 e *SP2Model*

Em continuação do trabalho de Trevisani e Garcia (2008), Sapia (2016) descreve o formalismo da SPML e apresenta a SPML2, propondo-a como uma melhoria da primeira.

Segundo o autor, a SPML2 é "...uma evolução da SPML que elimina a ambiguidade de símbolos, possibilita uma representação mais clara da política de segurança utilizando apenas um diagrama, e permite representar o bloqueio de tráfego de pacotes, além de outras melhorias.".

Para formalizar a SPML, com definições léxicas, sintáticas e semânticas, é definida uma metalinguagem que permite descrever os elementos gráficos em tokens (forma-

lismo léxico), tornando possível o formalismo sintático e semântico.

4.12.1 Formalismo Léxico

Como parte de seu trabalho, o autor define o formalismo léxico da SPML, utilizado também na SPML2. Esse formalismo é a base da metalinguagem, pois nele são definidos os *tokens* que identificam os lexemas da linguagem. Os *tokens* apresentados, juntamente com os respectivos significados, são listados a seguir:

- *program*: conjunto de linhas que definirão o *firewall*;
- *line*: definição de algum componente do *firewall*;
- *fw*: definição de um *firewall*;
- *if* : definição de uma interface;
- *extent*: definição de uma entidade externa;
- *ht*: definição de um *host*;
- *htl*: definição de uma lista de *hosts*;
- *net*: definição de uma rede de equipamentos;
- *un*: definição de uma rede cujo endereço é desconhecido;
- *flw*: definição de regras de filtragem de fluxos;
- *ifl*: definição de um fluxo de entrada;
- *af* : definição da versão do endereço ip;
- *state*: definição do tipo da conexão;
- *ofl*: definição de fluxo de saída;
- *napt*: definição de regras de tradução e redirecionamento;
- *srcnat*: definição de fluxo de entrada para tradução;
- *dstnat*: definição de fluxo de saída já traduzido;
- *srcrdr*: definição de fluxo de entrada a ser redirecionado;
- *dstrdr*: definição de fluxo de saída já redirecionado;
- *napt*: definição de regras de tradução e redirecionamento;
- *device*: definição do dispositivo da interface;
- *netmask*: definição da máscara de subrede;
- *antispoof* : definição da utilização do recurso *antispoof*;
- *domain*: definição do domínio;

- *gateway*: definição do endereço ip do *gateway*;
- *dns*: definição do endereço IP do servidor de DNS;
- *scrub*: definição da utilização do recurso de remontagem de pacotes;
- *proto*: definição do protocolo da camada de transporte;
- *srcport*: definição da porta de origem;
- *dstport*: definição da porta de destino;
- *fwname*: definição do nome do *firewall*;
- *ifname*: definição do nome da interface;
- *htname*: definição do nome do *host*;
- *htlname*: definição do nome da lista de *hosts*;
- *netname*: definição do nome da rede;
- *uname*: definição do nome da rede sem endereço ip;
- *iflname*: definição do nome de um fluxo de entrada de dados;
- *ofname*: definição do nome de um fluxo de saída de dados;
- *ip*: definição do IP de uma interface ou *host*.

Alguns *tokens* da SPML são removidos na SPML2, como *domain*, *gateway* e instruções que indicam tradução ou redirecionamento de tráfego. Também é criado um novo *token blk*, utilizado para representar bloqueio de tráfego.

4.12.2 Formalismo Sintático

As definições sintáticas, necessárias para a análise computacional de sentenças formadas por lexemas, são descritas utilizando a notação EBNF - *Extended Backus-Naur Form*) (Wirth, 1996). Na Definição 4.1 é apresentada a gramática utilizada para definir a SPML2.

Definição 4.1: Gramática da SPML2 expressa em EBNF (Sapia, 2016)

```
1  <program> ::= <line>{<line>}EOF
2  <line> ::= <fw> | <if> | <extent> | <tf>
3  <fw> ::= fw(<fwname>,<defaultpolicy>)
4  <defaultpolicy> ::= block | pass
5  <if> ::= if(<ifname>,<device>,<ip>,<netmask>,<fwname>)
6  <extent> ::= <ht> | <htl> | <net> | <un>
7  <ht> ::= ht(<htname>,<ip>,<netmask>)
8  <htl> ::= htl(<htlname>,<htname>{,<htname>})
9  <nat> ::= nat |
10 <net> ::= net(<netname>,<prefix>,<netmask>)
11 <un> ::= un(<uname>)
12 <tf> ::= <it> | <ot> | <blk>
13 <it> ::= it(<tid>,<itname>,<af>,<ifname>,<srcname>,
```

```
14              <srcport>,<rdrport>,<proto>{,<proto>})
15 <ot> ::= ot(<otname>,<ifname>,<dstname>,<dstport>,
16              <nat>,<tid>{,<tid>})
17 <blk> ::= blk(<tid>,<blkname>,<af>,<ifname>,<srcextent>,
18                <srcport>,<dstextent>,<dstport>,<proto>{,<proto>})
19 <af> ::= inet | inet6
20 <proto> ::= tcp | udp | icmp
21 <srcport> ::= 0..65535
22 <dstport> ::= 0..65535
23 <rdrport> ::= 0..65535
```

Por meio dos tokens e das regras gramaticais apresentadas, as sentenças da Definição 4.2 representam o modelo em SPML2 da Figura 4.15.

Definição 4.2: Tradução do modelo em SPML2 para a metalinguagem (Sapia, 2016)

```
 1 fw( Firewall, block)
 2 if( ifdmz, em0, 10.0.0.10, 24, Firewall)
 3 if( iflan, em2, 172.16.0.10, 24, Firewall)
 4 if( ifnet, em3, 10.10.10.10, 30, Firewall)
 5 if( ifint, em1, 192.168.0.10, 24, Firewall)
 6 ht( web, 10.0.0.1, 24)
 7 ht( dns, 10.0.0.2, 24)
 8 ht( ssh, 10.0.0.3, 24)
 9 un( Internet)
10 ht( adm1, 192.168.1.1, 24)
11 net( lan, 172.16.0.0, 24)
12 net( fin, 192.168.2.0, 24)
13 net( sub, 192.168.0.0, 16)
14 it( 1, subTcpIn, inet, ifint, sub, 0, 0, tcp)
15 it( 2, subUdpIn, inet, ifint, sub, 0, 0, udp)
16 it( 3, admTcpIn, inet, ifint, adm1, 0, 0, tcp)
17 it( 4, lanTcpIn, inet, iflan, lan, 0, 0, tcp)
18 it( 5, lanUdpIn, inet, iflan, lan, 0, 0, udp)
19 ot( webOut, ifdmz, web, 80, , 1)
20 ot( dnsOut, ifdmz, dns, 53, , 2)
21 ot( sshOut, ifdmz, ssh, 22, , 3)
22 ot( netWebOut, ifnet, Internet, 80, nat, 4)
23 ot( netDnsOut, ifnet, Internet, 53, nat, 5)
24 blk( 6, blkFin, inet, ifint, fin, 0, web, 80, tcp)
```

4.12.3 Formalismo Semântico

Por fim, a análise semântica, necessária para avaliar as restrições e condições das sentenças da metalinguagem, é formalmente definida utilizando a notação **Z** (Spivey e Abrial, 1992), que se baseia na teoria de conjuntos e cálculo de predicados. Para especificar um sistema utilizando a notação Z, é necessário primeiramente definir os tipos base ([*Firewall*; *Interface*; *ExtEntity*]) para, posteriormente, definir as operações sobre eles, juntamente às respectivas restrições.

Um exemplo de instanciação de um *firewall* é apresentado na Figura 4.14. Políticas de segurança expressas em SPML podem possuir apenas um *firewall*. Essa pré-condição é validada pelo predicado $fw = \emptyset$. Também é necessário que o *firewall* declarado não tenha sido criado anteriormente, condição garantida pelo predicado $firewall? \notin fw$.

$$
\begin{array}{|l}
\hline
\textit{AddFirewall} \\
\Delta SPML \\
firewall? : Firewall \\
\hline
fw = \emptyset \\
firewall? \notin fw \\
fw' = fw \cup \{firewall?\} \\
\hline
\end{array}
$$

Figura 4.14: Adicionando um *firewall* em notação **Z** (Sapia, 2016).

Por meio dos formalismos da metalinguagem definida, é possível eliminar a necessidade de traduzir os diagramas em documentos XML. A tradução para linguagem nativa de *firewall* é realizada diretamente das políticas escritas na metalinguagem.

4.12.4 *SP2Model*

Por meio da ferramenta apresentada, chamada *SP2Model*, é possível criar modelagens das políticas de segurança utilizando a SPML2. Na Figura 4.15 é ilustrado um exemplo de utilização da ferramenta.

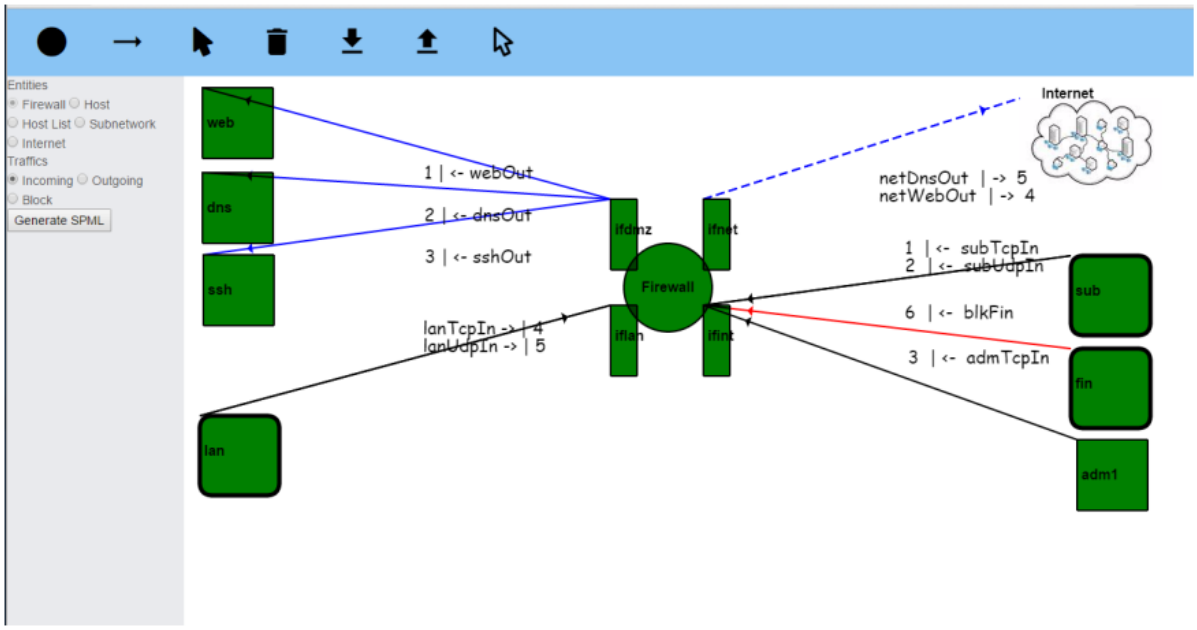


Figura 4.15: Exemplo de política de segurança utilizando a *SP2Model* (Sapia, 2016).

Os elementos gráficos funcionam de forma análoga aos descritos na Figura 4.13. No entanto, a SPML2 introduziu a utilização de cores nos componentes de filtragem e de tradução. Nos primeiros, cores são aplicadas para indicar a direção ou a permissão e bloqueio dos fluxos. Linhas pretas indicam tráfegos de entrada, azuis indicam tráfegos de saída e vermelhas indicam bloqueios dos tráfegos. Nos componentes de tradução, representados por uma reta tracejada com uma seta, a cor preta indica um redirecionamento de porta e a cor azul indica uma tradução de endereços de rede.

Adicionalmente, é possível traduzir as políticas criadas na ferramenta em regras em linguagem nativa de *firewall*, o que elimina a necessidade de escrevê- las manualmente.

Assim como a SOH, apresentada juntamente da SPML, a *SP2Model* realiza uma boa visualização das políticas de segurança. No entanto, em ambas não é possível carregar configurações já existentes a fim de visualizá-las graficamente. Além disso, não é possível testar as políticas na ferramenta, sendo necessário testa-las diretamente no *firewall*.

4.13 Considerações Finais

A literatura conta ainda com trabalhos focados na performance de *firewalls* como Liu et al. (2008), Yoon et al. (2010) e Cheng et al. (2019). Eles apresentam técnicas para compressão do número de regras, sem alterar a semântica do *firewall*. Por este trabalho não visar a performance como ponto central, não há relevância na descrição de tais trabalhos.

Analisando os trabalhos descritos neste capítulo, e o comparativo contido na Tabela 4.1, é possível observar que todas as ferramentas se preocupam com o processo de tradução das políticas e compreensão das regras já configuradas. A *Firmato* permite a representação das políticas por meio de um modelo entidade-relacionamento, e posteriormente traduzindo para linguagem nativa de *firewall*. Na *Fang* o administrador de redes pode validar configurações já realizadas por meio de um sistema de *queries*. A *Firewall Police Advisor* e a *FIREMAN* identificam anomalias nas configurações, que podem ter sido geradas por falha na tradução das políticas de segurança em regras. Por sua vez a *OpenSec* permite descrever as políticas por meio de uma linguagem de alto nível apresentada. Na ferramenta *FWS*, as regras de *firewalls* são abstraídas em tabelas, buscando facilitar a compreensão das configurações. Por fim, a *SPML* e *SPML2* utilizam representação gráfica para definição de políticas.

É evidente que o processo de tradução das políticas de segurança em regras e a compreensão das regras já implementadas são aspectos problemáticos para administradores de redes, visto que todas ferramentas buscam, de alguma forma, realizar abstrações com o objetivo de facilitar a compreensão, evitar erros, e validar configurações.

Tabela 4.1: Comparativo entre as ferramentas apresentadas.

	Abstração das políticas de segurança	**Representação das políticas em linguagem intermediária**	**Tradução para linguagem nativa de firewall**	**Abstração de regras de firewalls implantados**
Firmato	Modelo Entidade-relacioamento e ferramenta gráfica	Sim	Sim	Não
Fang	Com interações baseadas em queries	Não	Não	Sim
FPA	Em árvores	Não	Não	Sim
FIREMAN	Em grafos	Não	Não	Sim
OpenSec	Em linguagem textual definida	Não	Sim	Não
FWS	Com interações baseadas em queries	Sim	Sim	Sim
PolicyVis	Em matrizes	Não	Não	Sim
F/Wvis	Em 3D	Não	Sim	Sim
Visual Firewall Editor	Sunburst Tree map	Não	Não	Sim
SP2Model	Com elementos gráficos	Sim	Sim	Não
Fireasy	**Com elementos gráficos**	**Sim**	**Sim**	**Sim**

CAPÍTULO 5

A ferramenta Fireasy

5.1 Considerações Iniciais

Devido à falta de ferramentas de código aberto que proporcionem ao administrador de redes um ambiente que permita gerenciar as regras de *firewall* com a utilização de elementos gráficos, a *Fireasy* foi implementada. Como mencionado na Seção 1.4, foi necessário realizar a reimplementação da ferramenta proposta juntamente à SPML2, chamada SP2Model. Embora esta cumpra o seu objetivo, a introdução de novas funcionalidades, necessárias para o objetivo deste projeto, não seria viável. Por ser necessário realizar a engenharia reversa das regras em linguagem de *firewall*, traduzindo-as na metalinguagem SPML2, e por fim em elementos gráficos, é interessante que a implementação dos processos seja feita em módulos (ou camadas). Com esta modulação, é possível realizar a tradução de e para linguagens nativas de *firewalls* de diferentes fabricantes. Este processo de tradução da modelagem em elementos gráficos da SPML2 em regras nativas de *firewall* é ilustrado na Figura 5.1.

Em comparação com a *Fireasy*, a SP2Model não é capaz de interpretar regras escritas em linguagem de *firewall* e traduzi-las em diagrama SPML2, permitindo a adição, alteração ou deleção de regras, e posteriormente traduzindo novamente para regras em linguagem de *firewall*. A SP2Model se limita a realizar a modelagem inicial das políticas de segurança utilizando digramas SPML2, e traduzindo-o para linguagem de *firewall*. Ou seja, a SP2Model não é capaz de auxiliar os administradores de redes no gerenciamento de *firewalls* já implantados. Por outro lado, a *Fireasy* também permite realizar a modelagem inicial das politicas de segurança em diagramas SPML2, traduzindo-as para linguagem de *firewall*, no entanto ela realiza a interpretação das regras escritas de linguagem de *firewall*, transcrevendo-as em diagramas SPML2, permitindo a edição das regras e posterior tradução para regras em linguagem de *firewall*.

Neste capítulo a *Fireasy* é apresentada. Na Seção 5.2 estão contidas as tecnologias

utilizadas na implementação. Na Seção 5.3 é apresentada a arquitetura da *Fireasy*, descrevendo as principais funções implementadas, além do detalhamento do processo de tradução das regras de *firewall* em diagramas SPML. Por fim, na Seção 5.4 é exibida a interface da *Fireasy*.

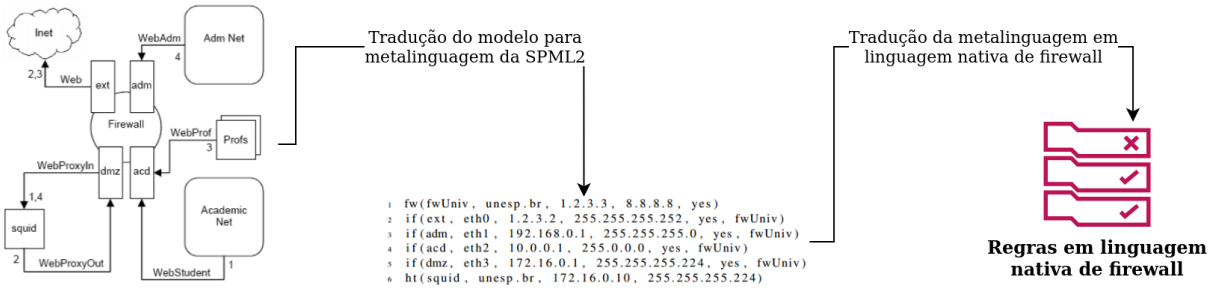


Figura 5.1: Fluxograma do processo de tradução de elementos gráficos da SPML2 em regras na linguagem nativa de *firewall*.

Para que seja possível visualizar as regras de *firewalls* já implantados em elementos gráficos seguindo a SPML2, é necessário que as regras em linguagem nativa de *firewall* sejam traduzidas para a metalinguagem da SPML2, seguindo seus formalismos léxico, sintático e semântico. Posteriormente, todas as configurações escritas na metalinguagem são representadas em elementos gráficos da SPML2. Este processo é ilustrado na Figura 5.2.

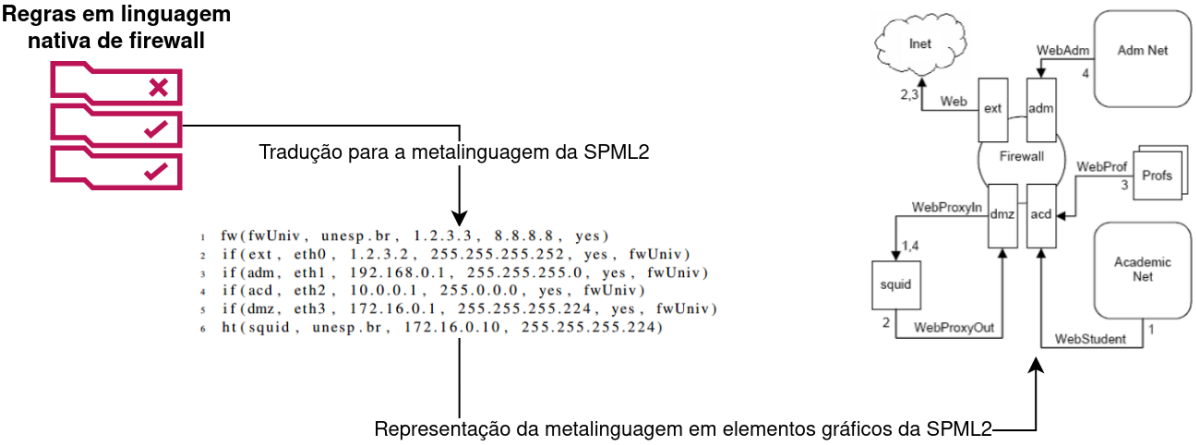


Figura 5.2: Fluxograma do processo de tradução de regras na linguagem de *firewall* em elementos gráficos da SPML2.

Com a visualização em elementos gráficos, é interessante que, por meio da ferramenta a ser desenvolvida, o administrador de redes seja capaz não apenas de visualizar as regras do *firewall* por meio de uma representação gráfica, mas também de criar novas regras, alterar regras existentes, ou remover regras que não sejam mais necessárias. Por fim, elas podem ser novamente traduzidas para linguagem nativa de *firewall*.

5.2 Tecnologias Utilizadas

Para o desenvolvimento da *Fireasy*, as tecnologias utilizadas foram as seguintes:

- Linguagem de programação Javascript;
- Linguagem de marcação de hipertexto HTML;
- Linguagem de estilos CSS;
- Formato compacto de troca de dados JSON
- Biblioteca Javascript GoJS;
- Biblioteca Javascript JQuery;
- Editor de código-fonte Visual Studio Code;
- Versionamento da ferramenta por meio do protocolo *git*, utilizando o *GitHub* como repositório;
- Navegadores Web:
  - Mozilla Firefox;
  - Google Chrome;

A escolha da linguagem *Javascript* se deu pela sua vasta utilização e constante atualização, além de ampla documentação e facilidade de obtenção de suporte por meio da comunidade, além de facilitar a integração com a biblioteca de diagramação utilizada.

Em relação ao ambiente de desenvolvimento, inicialmente foi utilizada a IDE Jetbrains WebStorm. No entanto, o editor Visual Studio Code contém *plugins* que podem melhorar o processo de desenvolvimento, dentre eles: *Live Server* (alterações feitas no código são refletidas imediatamente no navegador web), *JavaScript code snippets* (automatização na escrita de trechos de código genéricos), *Debugger for Chrome*, além de extensões visuais que podem melhorar a legibilidade do código, e autocompletar para funções e variáveis.

Baseada em JavaScript, a GoJs (GoJS, 2020) foi a biblioteca utilizada para implementar a representação gráfica da SPML2. Ela é uma biblioteca de diagramação que provê meios para facilitar a criação de gráficos e diagramas interativos, sendo possível executá-la nativamente na maioria dos navegadores web, sem a necessidade de instalação de *plugins*.

5.3 Arquitetura

A *Fireasy* é executada integralmente no navegador, ou seja, não é necessário um servidor para processar os dados. Uma vez que o navegador do usuário solicita a página, todos os arquivos *HTML*, *Javascript* e *CSS* são enviados ao cliente, e o processamentos dos dados é realizado utilizando os recursos computacionais do usuário.

A implementação da *Fireasy* foi efetuada em camadas. Por meio delas, é possível isolar cada processo que a ferramenta desempenha, o que pode proporcionar um código mais inteligível.

Na Figura 5.3 é ilustrado o diagrama de sequência que representa o processo de tradução do diagrama SPML2 em linguagem do *Packet Filter* que a *Fireasy* desempenha. As camadas presentes são: *UI*, *Utils*, *SPML*, *Estrutura* e *PacketFilter*. As funções de cada camada são apresentadas na Subseção 5.3.1.

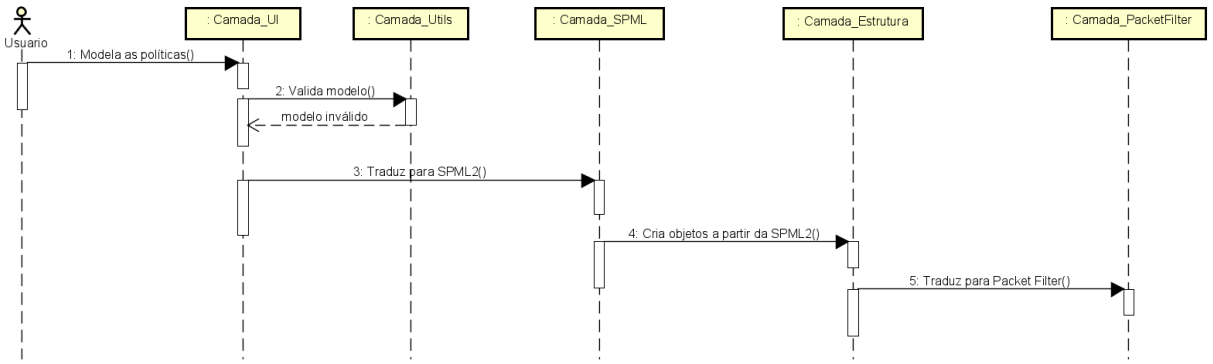


Figura 5.3: Diagrama de sequência da *Fireasy*. Tradução para *Packet Filter*.

O diagrama de sequência é iniciado com o usuário realizando a modelagem das políticas de segurança em diagramas SPML2 por meio da camada *UI*. Quando o usuário termina de modelar as políticas e requisita a tradução do modelo para regras de *firewall*, o diagrama SPML2 é enviado à camada *Utils*, onde é realizada a validação dos dados. Se o modelo for inválido, ou seja, se houver alguma violação da gramática da SPML2, o modelo é enviado novamente à camada *UI*, para que o usuário possa edita-lo. Caso o diagrama modelado seja válido, ele é enviado à camada *SPML*, responsável por traduzir o diagrama em metalinguagem SPML2.

Em seguida, as regras descritas em metalinguagem SPML2 são convertidas em objetos da camada *Estrutura*, que contém classes para os elementos do *firewall* (por exemplo *hosts*, interfaces, tráfegos). Por fim, os objetos, criados a partir das classes da camada *Estrutura*, que contém a representação das regras, são enviados à camada *PacketFilter*, onde é realizada a tradução deles em regras de *firewall* na linguagem do *Packet Filter*.

O diagrama de sequência ilustrado na Figura 5.4 representa a operação inversa, ou seja, o carregamento das regras em linguagem do *Packet Filter* e posterior representação em diagrama SPML2. As regras são inicialmente carregadas e na camada *Packet Filter* é realizado o processo de criação de objetos de acordo com as classes da camada *Estrutura*. A partir daí, os objetos são enviados à camada *SPML*, onde são convertidos em regras escritas em metalinguagem SPML2.

Para que seja possível representar as regras (em metalinguagem SPML2) em diagramas SPML2, elas são traduzidas para o modelo utilizado pela biblioteca GoJS, que necessita de uma representação em arquivo JSON, para por fim apresentá-las para o usuário na camada *UI*.

5.3.1 Estrutura da *Fireasy*

Estruturalmente, a *Fireasy* é dividida nas camadas ilustradas na Figura 5.5.

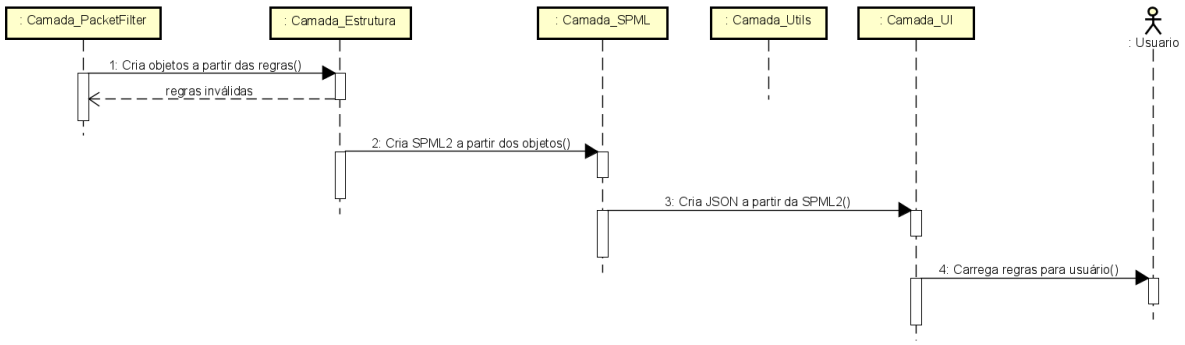


Figura 5.4: Diagrama de sequência da *Fireasy*. Tradução de regras em diagrama SPML2.

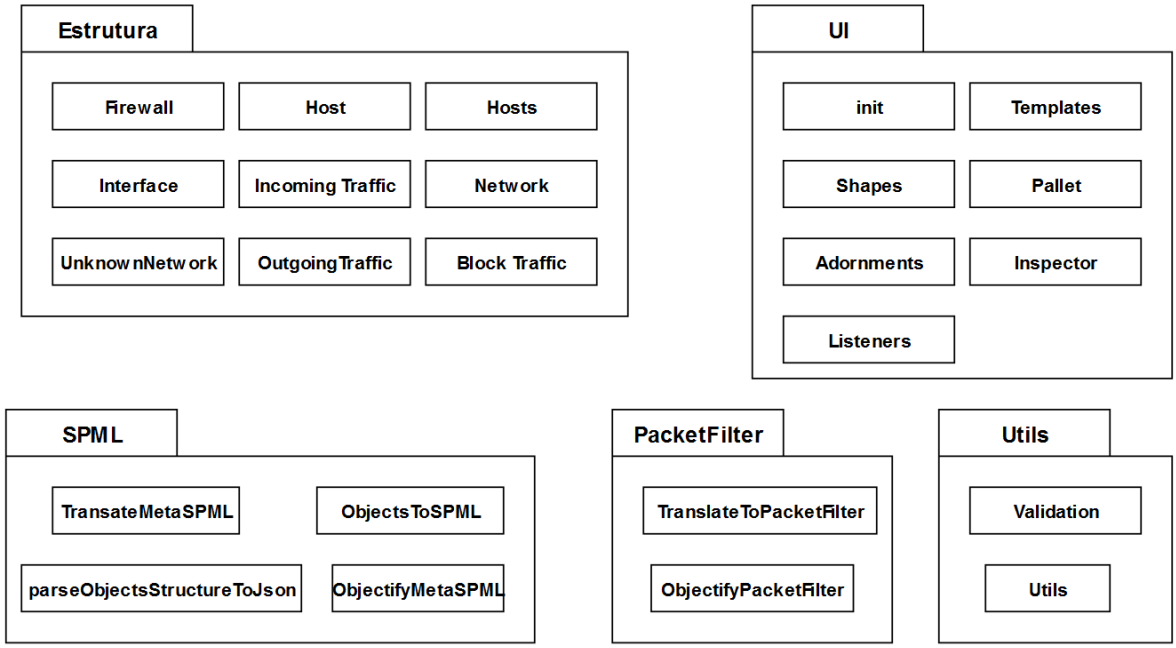


Figura 5.5: Camadas (ou pacotes) da implementação da *Fireasy*

5.3.1.1 Camada *Estrutura*

Na camada *Estrutura* estão contidas classes para representar as entidades (*Firewall*, *Host*, *Hosts*, *Interface*, *Network*, e *Unknown Network*) e tráfegos (*IncomingTraffic*, *OutgoingTraffic*, e *BlockTraffic*) do *firewall*. Os atributos delas seguem a gramática da SPML2, descrita na Definição 4.1.

5.3.1.2 Camada *UI*

Na camada *UI* estão contidas funções de interface com o usuário. A função *init* é a primeira a ser executada quando a ferramenta é iniciada. A partir dela são invocadas as outras funções da camada. A função *Shapes* define formas geométricas customizadas, que não estão presentes nativamente na biblioteca GoJS. Na função *Templates* são definidos os modelos geométricos para os elementos gráficos utilizados no diagrama, como por exemplo qual será a forma geométrica de um *firewall* (círculo), ou qual a cor da linha que representa um tráfego de entrada (preta). Também são definidas restrições para o modelo, como por exemplo verificar se é possível editar ou

redimensionar uma entidade.

A função *Pallet* instancia a paleta que contém os modelos dos elementos gráficos (*firewall*, *host*, *network*, *hostlist* e *unknown network*), exceto interfaces e tráfegos, que possuem outra maneira de serem adicionados, descritas mais adiante nessa seção. Por meio dela, o usuário pode arrastar um modelo para criá-lo na área que contém o diagrama principal.

A função *Inspector* é responsável por gerenciar os atributos dos elementos gráficos. Cada elemento gráfico do diagrama SPML2 contém atributos que seguem a gramática descrita na Definição 4.1. Por exemplo, uma interface possui os atributos: nome, nome do dispositivo, IP, máscara de rede, e nome do *firewall* associado à ela. O diagrama apenas representa a forma dos elementos e as conexões entre eles. Os seus atributos são inseridos por meio do *Inspector*. Além disso, ele realiza validações que verificam quando um campo é preenchido de maneira incorreta.

Na função *Adornments*, são definidos os botões de criação de nova interface e tráfegos. Estes botões são visíveis apenas quando um elemento é selecionado. Nas Figuras 5.17 e 5.18 estão representados os botões de criação de interface e de tráfego, respectivamente. Nesta função também são definidas restrições para criação de link, pois um tráfego de saída só poderá ser criado se houver ao menos um tráfego de entrada existente.

Por fim, na função *Listeners* estão contidos todos os *listeners* da *Fireasy*, que processam os eventos de mouse e teclado. Por exemplo. quando uma interface ou o *firewall* é selecionado, e em seguida é pressionada a tecla *delete*, é apresentada uma caixa de diálogo de confirmação, pois a deleção desses elementos acarreta na remoção dos tráfegos a eles conectados.

5.3.1.3 Camada *Utils*

Na camada *Utils* estão contidas as funções *Validation* e *Utils*. A primeira é responsável por validar o modelo, realizando verificações em relação aos: IPs, portas, máscara de rede, nomes. A validação de nomes é particularmente importante para a tradução do modelo para a metalinguagem SPML2. Considerando um exemplo de definição de uma sub-rede em metalinguagem SPML2, representado por "*net(administrativa,192.168.0.0,24)*", e um tráfego de entrada "*it(2,amd_udp,inet,adm,administrativa,*,,udp)*", a referência da entidade de origem do tráfego neste caso é "*administrativa*". Não é inserido o IP da sub-rede, mas o seu nome, portanto, os nomes das entidades devem ser únicos, para evitar ambiguidade na tradução dos tráfegos. Por fim, a função *Utils* possui métodos diversos que realizam por exemplo: a redefinição de contadores utilizados para nomear as entidades e tráfegos, redefinição dos objetos da Estrutura quando um modelo está sendo carregado na ferramenta, operações de redimensionamento do elemento *firewall* à medida que novas interfaces são adicionadas ou removidas, busca de elementos por id, busca de todas as entidades.

5.3.1.4 Camada *SPML*

Na camada *SPML* estão contidas funções que realizam a tradução do diagrama em metalinguagem SPML2, e quando regras de *firewall* estão sendo carregadas, ela realiza

a tradução de objetos das classes da camada Estrutura em metalinguagem SPML2. Conforme o diagrama é criado, a biblioteca GoJS disponibiliza um arquivo JSON que representa o modelo. A função *TranslateMetaSPML* percorre este arquivo JSON, e realiza a tradução para regras em metalinguagem SPML2. Esta tradução é de certa forma simples, pois os atributos dos elementos do diagrama seguem a descrição da gramática da SPML2, sendo necessário apenas extraí-los e estrutura-los na notação da metalinguagem SPML2.

Na função *ObjectifyMetaSPML* a metalinguagem SPML2 é processada, e a partir dela são criados objetos das classes da camada Estrutura, que as representam. Por sua vez, a função *ObjectsToSPML* é utilizada no processo de tradução das regras escritas em linguagem do *Packet Filter* em diagramas SPML. Inicialmente as regras são processadas e a partir delas são criados os objetos (das classes contidas no pacote *Estrutura*) correspondentes à cada elemento ou tráfego. Esta implementação está detalhada na Subseção 5.3.2. Com os objetos que representam os elementos e tráfegos criados, a função *ObjectsToSPML* os processa e traduz para regras em metalinguagem SPML2. Por fim, a função *parseObjectsStructureToJson* é responsável por traduzir os objetos que contém os elementos e tráfegos em JSON, que será enviado à camada *UI* para carrega-lo e permitir a visualização do modelo.

5.3.1.5 Camada *PacketFilter*

Na camada *PacketFilter*, a função *TranslateToPacketFilter* processa os objetos recebidos da função *ObjectifyMetaSPML (da camada SPML)* e realiza a tradução para regras em linguagem do *Packet Filter*. A função *ObjectifyPacketFilter*, abordada com mais detalhes na Subseção 5.3.2, instancia objetos das classes da camada *Estrutura* a partir das regras em linguagem do *Packet Filter*.

É importante observar que ferramenta foi implementada de forma que a portabilidade para outras linguagens de *firewall* seja realizada de maneira simples. Para tanto, bastaria reescrever as funções da camada *PacketFilter*.

5.3.2 Implementação da tradução de regras *Packet Filter* em diagramas SPML2

A implementação da funcionalidade de traduzir regras de *firewall* (especificamente em linguagem do *Packet Filter*) em diagramas SPML2 é realizada principalmente na função *ObjectifyPacketFilter*. Nesta Subseção são apresentados autômatos finitos utilizados para descrever os processos de tradução das regras de *firewall* em SPML2. O objetivo desta função é preparar objetos estruturados de acordo com a gramática da SPML2, descrita na Definição 4.1, para que posteriormente as regras sejam traduzidas em metalinguagem SPML2, e por fim em diagrama SPML2.

O autômato da Figura 5.6 representa a função principal da tradução, a partir da qual são derivadas as demais funções abordadas adiante. Cada função percorre todas as regras de *firewall*, filtrando a parte pertinente para seu objetivo, e instancia os objetos necessários para representa-las. Esta função principal invoca as demais funções na sequência de acordo com a numeração dos estados do autômato, do *q0* ao

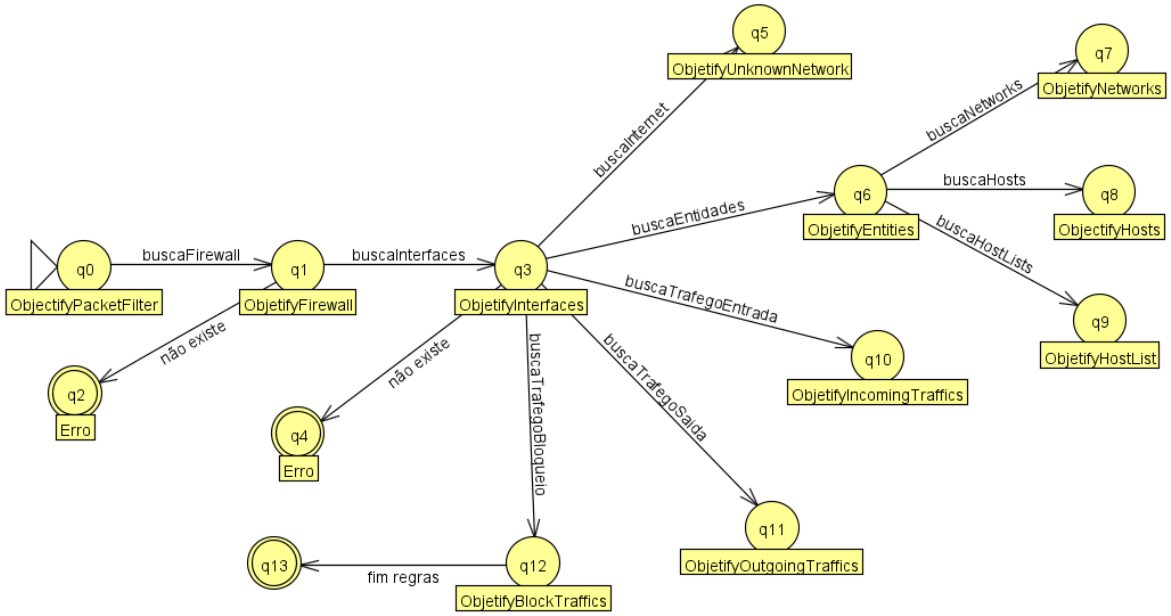


Figura 5.6: Autômato da função *ObjetifyPacketFilter*.

q12.

Para auxiliar o leitor na compreensão das funções descritas nessa Subseção, na Definição 5.1 é ilustrado um exemplo regras de *firewall* na linguagem do *Packet Filter*.

Definição 5.1: Exemplo de regras de *Firewall* na linguagem do *Packet Filter*

```
1  block in all
2  pass in on em2 inet proto { udp } from 192.168.0.0/24 to /
3  8.8.8.8/32 port 53
4  pass in on em3 inet proto { udp } from 172.16.0.0/24 to /
5  8.8.8.8/32 port 53
6  pass in on em2 inet proto { tcp } from 192.168.0.0/24 to /
7  172.16.0.10/24 port 3128
8  pass in on em3 inet proto { tcp } from 172.16.0.10/24 to /
9  any port 80
10 pass in on em3 inet proto { tcp } from 172.16.0.10/24 to /
11 any port 443
12 pass in on em0 inet proto { tcp } from any to 5.5.5.5/30 /
13 port 80 rdr-to 172.16.0.80/24 port 8080
14 pass in on em2 inet proto { tcp } from 192.168.0.0/24 to /
15 { 173.194.205.108/32, 173.194.205.109/32 } port 25
16 pass in on em2 inet proto { tcp } from 192.168.0.0/24 to /
17 { 173.194.205.108/32, 173.194.205.109/32 } port 110
18 pass out on em0 inet proto { udp } from 192.168.0.0/24 to /
19 8.8.8.8/32 port 53 nat-to 5.5.5.5/30
20 pass out on em0 inet proto { udp } from 172.16.0.0/24 to /
21 8.8.8.8/32 port 53 nat-to 5.5.5.5/30
22 pass out on em3 inet proto { tcp } from 192.168.0.0/24 to /
23 172.16.0.10/24 port 3128
24 pass out on em0 inet proto { tcp } from 172.16.0.10/24 to /
25 any port 80 nat-to 5.5.5.5/30
26 pass out on em0 inet proto { tcp } from 172.16.0.10/24 to /
27 any port 443 nat-to 5.5.5.5/30
28 pass out on em3 inet proto { tcp } from any to 172.16.0.80/24 /
29 port 8080
30 pass out on em0 inet proto { tcp } from 192.168.0.0/24 to /
```

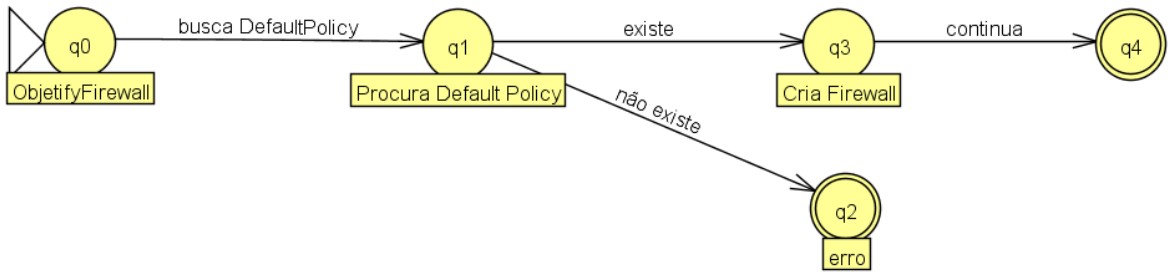


Figura 5.7: Autômato da função *ObjetifyFirewall*.

```
31 { 173.194.205.108/32, 173.194.205.109/32 } port 25 nat-to 5.5.5.5/30
32 pass out on em0 inet proto { tcp } from 192.168.0.0/24 to /
33 { 173.194.205.108/32, 173.194.205.109/32 } port 110 nat-to 5.5.5.5/30
```

5.3.2.1 Firewall

No trabalho de (Sapia, 2016), foram definidas regras semânticas para a SPML, utilizando Notação Z para definir as limitações e propriedades da gramática. A Notação 1 representa a operação de criação do *firewall*, que deve ser único e não pode ter sido criado anteriormente.

$$
\begin{array}{|l}
\hline
AddFirewall \\
\Delta SPML \\
firewall? : Firewall \\
\hline
fw = \emptyset \\
firewall? \notin fw \\
fw' = fw \cup \{firewall?\} \\
\hline
\end{array}
$$

Notação 1: Adiciona um *firewall*.(Sapia, 2016)

No processo de tradução das regras de *firewall*, a função *ObjetifyFirewall* é ilustrada no autômato da Figura 5.7. Segundo a gramática da SPML2, um *firewall* possui como atributos apenas nome e política padrão. O estado *q1* do autômato indica a procura da política padrão nas regras. Se ela não existir, o programa não pode continuar a execução e é apresentado um erro. Ao encontrar a política padrão, a função cria o objeto da classe *Firewall* (camada *Estrutura*), e insere a política padrão.

No exemplo apresentado na Definição 5.1, a linha "*block in all*" é processada e a política padrão "*block*" é encontrada.

5.3.2.2 Interfaces de rede

Uma vez que o *Firewall* foi criado, o próximo passo é adicionar suas interfaces de rede. Este processo é ilustrado no autômato da Figura 5.8. A busca por interfaces de rede é iniciada percorrendo todas as regras de *firewall* carregadas na ferramenta, linha a

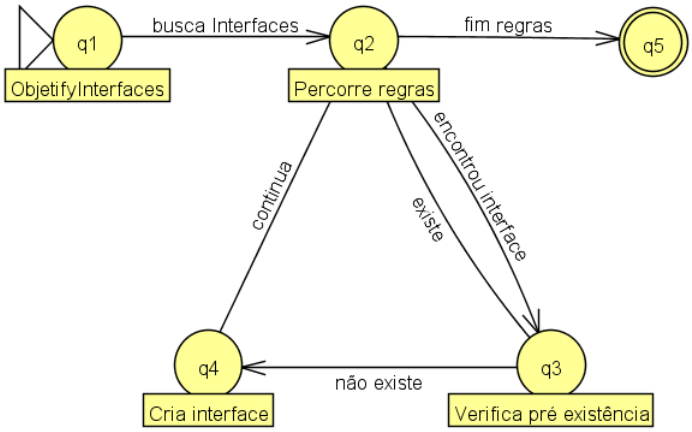


Figura 5.8: Autômato da função *ObjetifyInterfaces*.

linha. Ao encontrar uma interface, ele deve verificar sua pré existência, de acordo com a operação de adicionar uma interface da Notação 2. Se a interface já existir no *firewall*, a linha é ignorada e a busca por interfaces continua na linha seguinte. Se a interface não existir, ela é criada. Ao final das regras, todas as interfaces existentes no *firewall* são devidamente adicionadas.

Tomando a linha "*pass in on em3 inet proto tcp from 172.16.0.10/24 to any port 80*" do exemplo da Definição 5.1, a função encontrará a interface de rede "*em3*".

$$
\begin{array}{|l}
\hline
AddInterface \\
\Delta SPML \\
interface? : Interface \\
firewall? : Firewall \\
\hline
interface? \notin if \\
firewall? \in fw \\
if' = if \cup \{interface?\} \\
ifs' = ifs \cup \{(interface? \mapsto firewall?)\} \\
\hline
\end{array}
$$

Notação 2: Adiciona uma interface.(Sapia, 2016)

5.3.2.3 Redes Desconhecidas (*Unknown Networks*)

A criação de uma rede desconhecida, usualmente representada por *internet* pois é uma rede externa ao *firewall*, é ilustrada no autômato da Figura 5.9. Assim como ocorre nas interfaces, a função percorre as regras em busca da palavra *any*, o que indica a existência de uma rede desconhecida. Seguindo as restrições de operação de adição de uma rede desconhecida, representada na Notação 3, uma rede desconhecida deve ser única. No autômato, é possível observar que quando a palavra *any* é encontrada, a rede desconhecida é criada e a execução da função é finalizada.

Tomando a linha "*pass in on em3 inet proto tcp from 172.16.0.10/24 to any port 80*" do exemplo da Definição 5.1, ao encontrar a palavra *any*, a rede desconhecida é

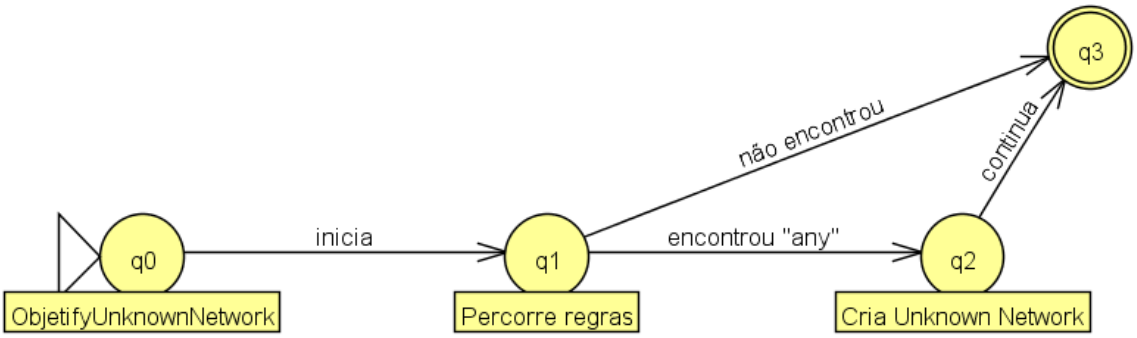


Figura 5.9: Autômato da função *ObjetifyUnknownNetwork*.

adicionada.

AddUnkNet

$\Delta SPML$

$unknet? : ExtEntity$

$unknet? \notin un$

$un' = un \cup \{unknet?\}$

Notação 3: Adiciona uma rede com endereço desconhecido.(Sapia, 2016)

5.3.2.4 Entidades

Os IPs existentes nas regras de *firewall* representam diferentes entidades, podendo ser um *host*, uma sub-rede (*network*), ou uma lista de host (*hostlist*). Para que essa distinção seja feita, primeiramente os IPs são filtrados das regras de *firewall*, para posterior classificação.

O autômato da Figura 5.10 representa o fluxo de execução da função que filtra os IPs das regras. Inicialmente, é criado um *array* onde os IPs serão armazenados, e então as regras são percorridas. Para cada linha, é filtrado o IP de origem (*from*), e se ele for igual a *any* (uma rede desconhecida), ele é ignorado, se não ele é adicionado ao *array* de IPs. A execução continua filtrando o IP de destino (*to*), realizando a mesma verificação de igualdade com *any* para adicionar ou não o IP ao *array*. Por fim, é verificado se a regra possui um IP de redirecionamento (*rdr*), se sim, extrai-o e adiciona ao array de IPs.

Tomando a linha "*pass in on em2 inet proto udp from 192.168.0.0/24 to 8.8.8.8/32 port 53*" do exemplo da Definição 5.1, a função adiciona os IPs "192.168.0.0/24" e "8.8.8.8/32" no *array* de IPs.

Uma vez que todos os IPs estão inseridos no *array* de IPs, a execução continua na função *ObjectifyHosts*, que tem seu funcionamento ilustrado na Figura 5.11. A execução é iniciada com o instanciamento do *array* de *hosts*, e então o *array* de IPs é percorrido. Se o IP analisado possuir os caracteres "{" ou "," ele é ignorado, pois

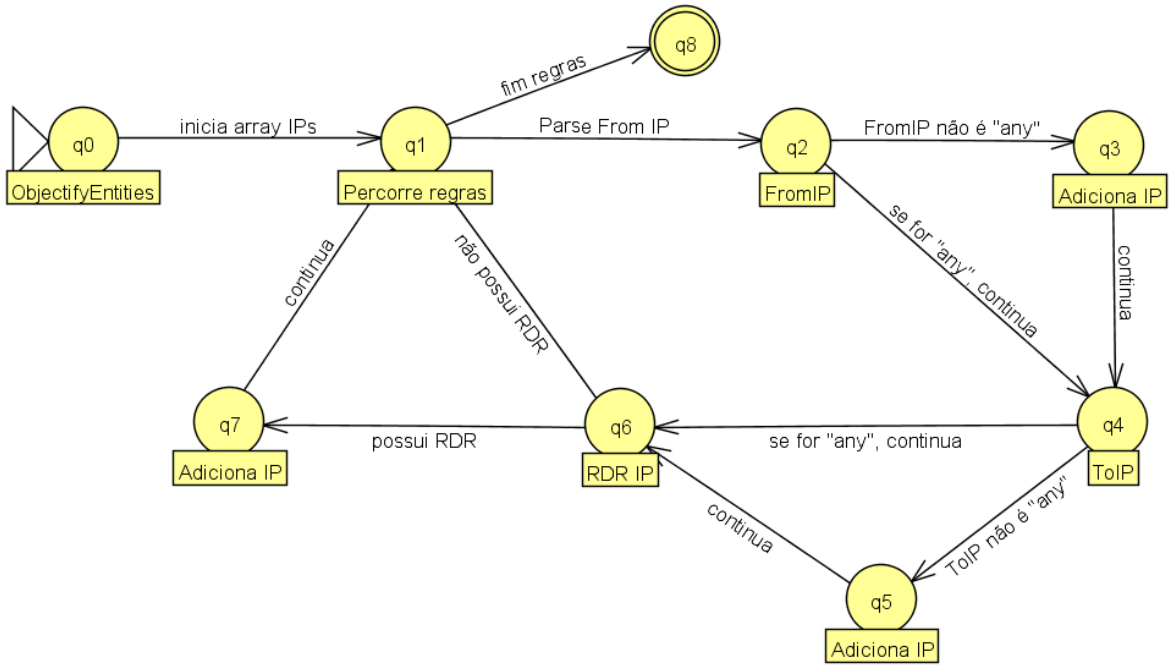


Figura 5.10: Autômato da função *ObjetifyEntities*.

se trata de uma lista de hosts, que será tratada posteriormente. Se o IP não contiver esses caracteres, é verificado se a ultima parte do IP termina em 0, e em caso positivo, se trata de uma sub-rede, e o IP é ignorado. Em caso negativo (a última parte do IP não termina em zero), é verificado se ele já está contido no array de *hosts*, e se não estiver, adiciona-o. Esta verificação de pré existência do *hosts* é a condição única da operação de adição de *hosts*, de acordo com a Notação 4.

AddHost

$\Delta SPML$

$host? : ExtEntity$

$host? \notin ht$

$ht' = ht \cup \{host?\}$

Notação 4: Adiciona um *host*.(Sapia, 2016)

A adição de um *hostlist*, ilustrada no autômato da Figura 5.12, é iniciada pelo instanciamento do *array* de *hostlists*. A seguir, o *array* de IPs é percorrido e, se os caracteres "{" e "," não estiverem presentes, o IP é ignorado, pois não se trata de uma lista de *hosts*. Se estiverem presentes, os IPs separados por vírgulas são extraídos e percorridos. Para cada IP, é verificado se o *host* já existe no *array* de *hosts*, adicionando-o em caso negativo. Por fim, é apurado se o *hostlist* a ser criado já existe, e em caso negativo, ele é instanciado. A verificação de pré existência é a condição única na criação de um *hostlist*, segundo a Notação 5

As últimas entidades a serem criadas são as sub-redes (*networks*). O autômato

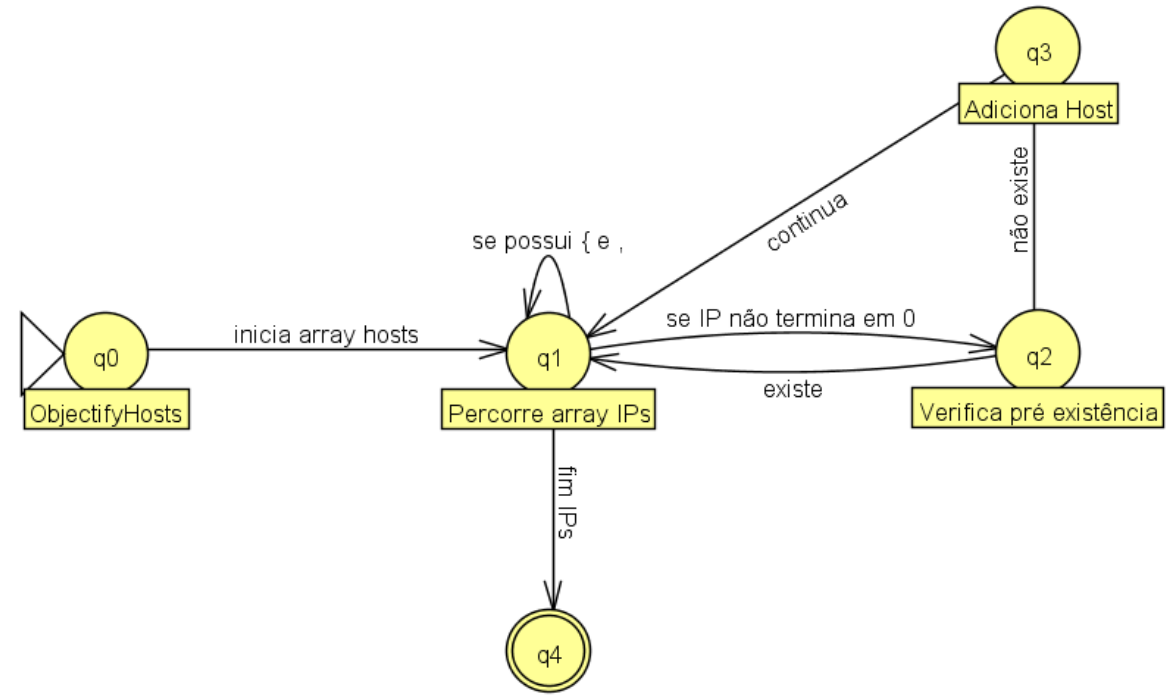


Figura 5.11: Autômato da função *ObjetifyHosts*.

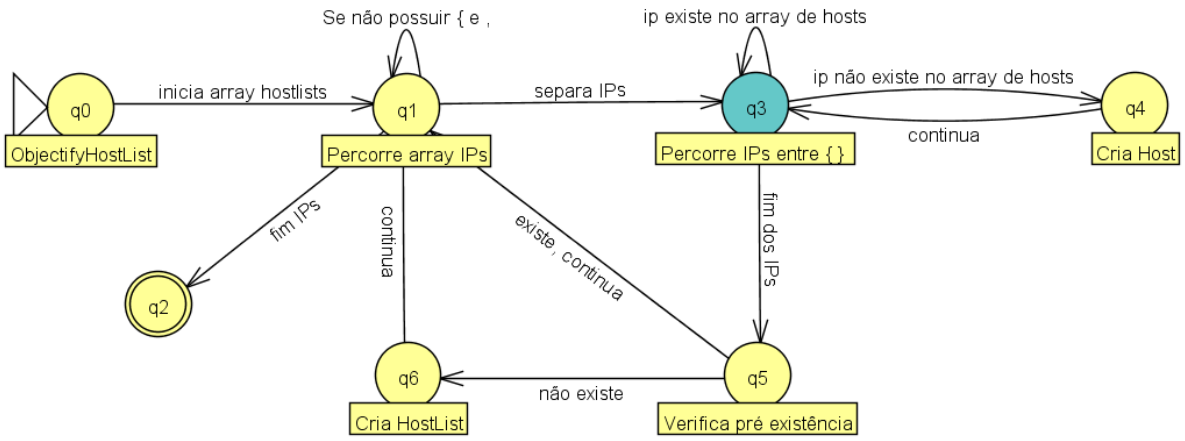


Figura 5.12: Autômato da função *ObjetifyHostList*.

AddHostList

$\Delta SPML$

$hostlist? : ExtEntity$

$hostlist? \notin htl$

$htl' = htl \cup \{hostlist?\}$

Notação 5: Adiciona uma lista de *hosts*.(Sapia, 2016)

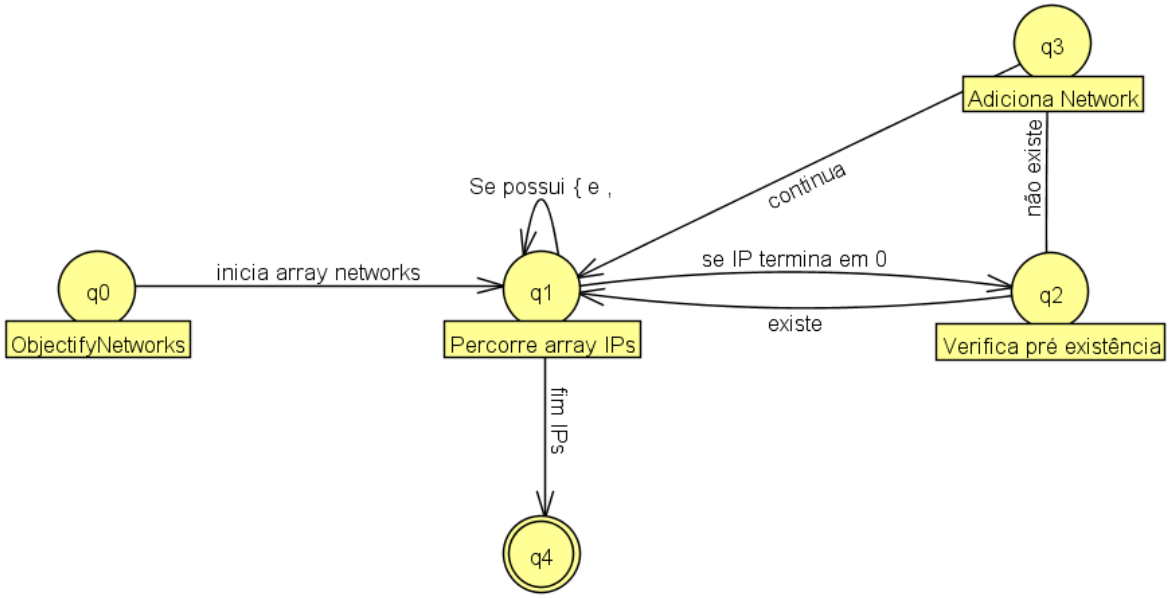


Figura 5.13: Autômato da função *ObjetifyNetwork*.

da Figura 5.13 representa a função *ObjectifyNetworks*, a qual é inicia pelo instanciamento do *array* de *networks*. Em seguida, o *array* de IPs é percorrido, e para cada IP, é verificado se ele contém "{"ou ",", ignorando-o em caso positivo. Se não contiver, é verificado se a última parte do IP é igual a 0. Se sim, trata-se de uma sub-rede e então é verificada sua pré existência. Caso não exista, ela é adicionada. De acordo com a Notação 6, a verificação de pré existência é o único requisito na operação de criação de uma sub-rede.

$$
\begin{array}{|l}
\hline
AddNetwork \\
\Delta SPML \\
network? : ExtEntity \\
\hline
network? \notin net \\
net' = net \cup \{network?\} \\
\hline
\end{array}
$$

Notação 6: Adiciona uma rede.(Sapia, 2016)

5.3.2.5 Tráfego de Entrada

Uma vez que todas as entidades foram criadas, são executadas as funções que definem as relações entre elas, ou seja, os tráfegos.

Na Figura 5.14 é ilustrado o autômato que descreve a execução da função *ObjectifyIncomingTraffics*, responsável por instanciar os tráfegos de entrada. Segundo a semântica da SPML2 com relação à adição de tráfego de entrada, representada na Notação 7, um tráfego só pode ser criado se a entidade externa que origina os pacotes existir. Esta condição é satisfeita pela criação das entidades externas descritas

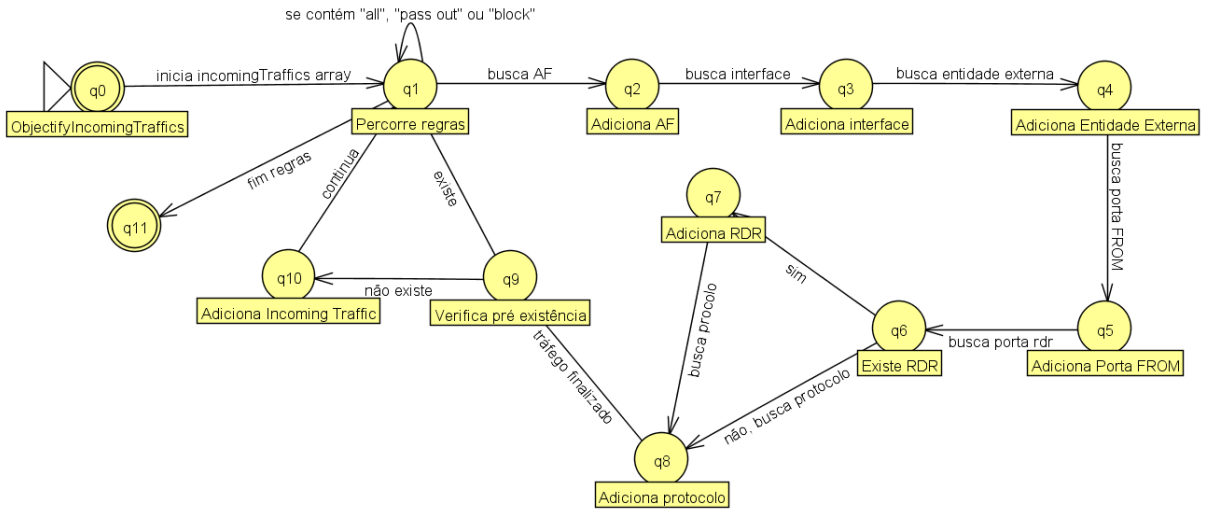


Figura 5.14: Autômato da função *ObjetifyIncomingTraffics*.

nas funções *ObjectifyUnknownNetwork*, *ObjetifyHosts*, *ObjectifyHostList* e *Objectify-Networks*.

AddIncomingTraffic

$\Delta SPML$

$externalentity? : ExtEntity$

$interface? : Interface$

$externalentity? \in ht \cup htl \cup net \cup un$

$interface? \in if$

$ifl' = ifl \cup \{(externalentity? \mapsto interface?)\}$

$fid' = fid + 1$

Notação 7: Adiciona um tráfego de entrada. (Sapia, 2016)

A execução da função *ObjectifyIncomingTraffics* percorre todas as regras, ignorando as que iniciem com "*pass out*" (tráfegos de saída) ou "*block*" (tráfegos de bloqueio), além de também ignorar a regra de política padrão.

Segundo a gramática da SPML2, expressa na Definição 4.1, um tráfego de entrada deve conter os atributos: id, nome, AF, nome da interface, nome da entidade externa de origem, porta de origem, porta de redirecionamento e protocolo.

Os *ids* e nomes dos tráfegos de entrada são criados dinamicamente à medida que os tráfegos são adicionados, com os nomes seguindo o padrão "*Incoming Traffic <num>*", sendo *num* um contador de tráfegos.

Seguindo a execução da função e tomando como exemplo as regras da Definição 5.2, no processamento da primeira regra, é realizada a busca pela AF (*inet*) e interface (*em3*). A seguir é filtrada a entidade externa de origem, o IP 172.16.0.10/24. No entanto, segundo a gramática da SPML2, é necessário recuperar o nome da entidade externa (não o IP), tarefa realizada por uma função auxiliar que percorre as

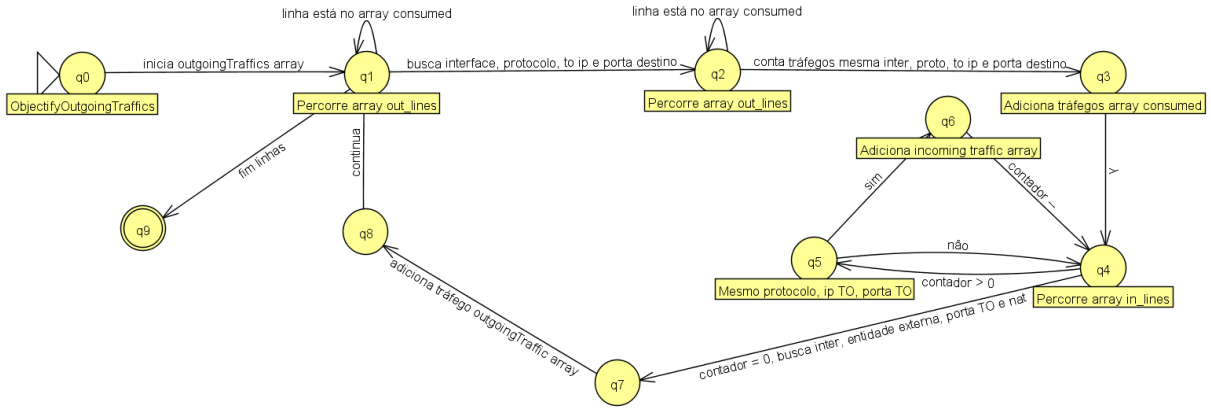


Figura 5.15: Autômato da função *ObjetifyOutgoingTraffics*.

entidades externas criadas anteriormente e encontra a entidade com o IP em questão, recuperando o seu nome.

O próximo passo é realizar a busca pela porta de origem, que no exemplo está omitida, indicando que se trata de qualquer porta, então o campo de porta de origem é preenchido com "*". A seguir é verificado se o tráfego possui porta de redirecionamento, e em caso positivo a adiciona. O tráfego é finalizado com a filtragem do protocolo (tcp no exemplo). É interessante observar que a gramática da SPML2 não utiliza informações sobre destino do tráfego ("any"na porta 80 para o primeiro tráfego, e 443 para o segundo) para definir um tráfego de entrada, essas informações serão utilizadas apenas no tráfego de saída.

Por fim, a função verifica se o tráfego de entrada criado já existe. No exemplo, as duas regras possuem os mesmos atributos para a montagem do tráfego de entrada, diferenciando-se apenas nos destinos, os quais são ignorados por ora. Sendo assim, quando a função processa a segunda linha, é verificado que já existe um tráfego de entrada com os mesmo atributos, não sendo necessário cria-lo novamente.

Definição 5.2: Exemplo de tráfegos de entrada

```
1        pass in on em3 inet proto { tcp } from 172.16.0.10/24 /
2        to any port 80
3        pass in on em3 inet proto { tcp } from 172.16.0.10/24 /
4        to any port 443
```

5.3.2.6 Tráfego de Saída

A operação de criação de um tráfego de saída, segundo a Notação 8, só pode ser realizada se o seu respectivo tráfego de entrada já houver sido criado. Na execução da função *ObjetifyPacketFilter*, os tráfegos de entrada são criados antes dos de saída, e no instanciamento destes é realizada a busca pelo respectivo tráfego de entrada.

A função que cria os tráfegos de saída, chamada *ObjectifyOutgoingTraffics*, tem sua execução ilustrada pelo autômato da Figura 5.15. Algumas variáveis e verificações foram omitidas para simplificar o entendimento.

A função é iniciada com o instanciamento do *array* de tráfegos de saída. Em seguida, são instanciados mais dois *arrays* auxiliares, o *out_lines*, que contém todas

$$
\begin{array}{|l}
\hline
\textit{AddOutgoingTraffic} \\
\Delta SPML \\
externalentity? : ExtEntity \\
interface? : Interface \\
flowid? : \mathbb{N} \\
\hline
externalentity? \in ht \cup htl \cup net \cup un \\
interface? \in if \\
flowid? > 0 \land flowid? \leq fid \\
ofl' = ofl \cup \{(interface? \mapsto externalentity?)\} \\
\hline
\end{array}
$$

Notação 8: Adiciona um tráfego de saída.(Sapia, 2016)

as regras de tráfego de saída, e o *in_lines*, que contém todas as regras de tráfego de entrada.

No estado "*q1*", o *array* que contem as regras de tráfego de saída (*out_lines*) é percorrido, e para cada tráfego, são filtradas a interface, protocolo, IP e porta de destino. Este *array* é então novamente percorrido, contando quantos tráfegos de saída possuem a mesma interface, protocolo, IP e porta de destino do primeiro. Esses tráfegos são então armazenados em um *array* temporário, chamado de *consumed*, para indicar que essas regras não necessitam ser verificadas novamente. Também é criada uma variável chamada *count_outgoing* para contar quantos tráfegos de saída possuem a mesmo interface, protocolo, IP e porta de destino do tráfego inicialmente analisado.

Tomando o exemplo da Definição 5.3, os dois tráfegos de saída (iniciados por "*pass out* ") possuem a mesma interface, protocolo, IP e porta de destino.

Definição 5.3: Exemplo de tráfegos de saída

```
1  pass in on em2 inet proto { udp } from 192.168.0.0/24 to /
2  8.8.8.8/32 port 53
3  pass in on em3 inet proto { udp } from 172.16.0.0/24 to /
4  8.8.8.8/32 port 53
5  pass out on em0 inet proto { udp } from 192.168.0.0/24 /
6  to 8.8.8.8/32 port 53 nat-to 5.5.5.5/30
7  pass out on em0 inet proto { udp } from 172.16.0.0/24 /
8  to 8.8.8.8/32 port 53 nat-to 5.5.5.5/30
```

Segundo a gramática da SPML2, representada na Definição 4.1, um tráfego de saída pode contem um ou mais tráfegos de entrada. Desta forma, é instanciado um *array* chamado *in_traffics*, para armazenar os tráfegos de entrada (objetos da classe IncomingTraffics, instanciados na função *ObjectifyIncomingTraffics*) relacionados ao tráfego de saída sendo processado.

Em seguida, o *array in_lines* (que contém as regras de tráfegos de entrada) é percorrido, e enquanto a variável *count_outgoing* for maior que zero, é verificado para cada linha do *array in_lines* se o protocolo, IP e porta destino são os mesmo do tráfego de saída sendo processado. Em caso positivo, o objeto do tráfego de entrada

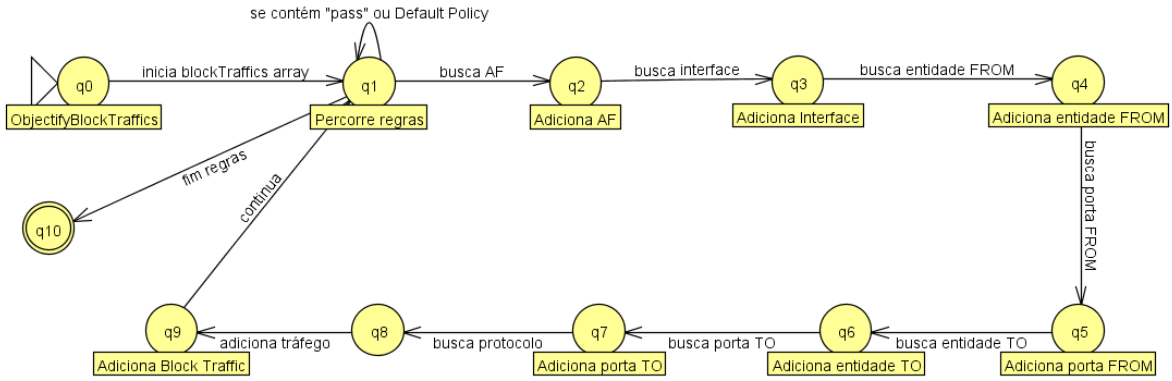


Figura 5.16: Autômato da função *ObjetifyBlockTraffics*.

é recuperado e armazenado no *array in_traffics*. No exemplo da Definição 5.3, os dois tráfegos de entrada são lidos e armazenados no *array in_traffics*, pois possuem mesmo protocolo, IP e porta de destino dos tráfegos de saída.

Quando o contador *count_outgoing* for igual a zero, isto indica que todos os tráfegos de entrada relacionados ao tráfego de saída sendo processado já estão armazenados no *array in_traffics*. Posteriormente são recuperados os objetos da interface, entidade externa destino (ambos instanciados em suas respectivas funções), além da porta destino e da existência ou não de *nat*. Estas informações são inseridas em um objeto que represente o tráfego de saída (da classe *OutgoingTraffic*), e ele é inserido no *array outgoingTraffics*.

5.3.2.7 Tráfego de Bloqueio

A função *ObjetifyBlockTraffics*, tem sua execução representada no autômato da Figura 5.16

De acordo com a gramática da SPML2, descrita na Definição 4.1, um tráfego de bloqueio possui os seguintes atributos: id, nome do tráfego, AF, nome da interface, entidade externa de origem, porta de origem, entidade externa de destino, porta de destino, e protocolo. Um exemplo de tráfego de bloqueio em linguagem do *Packet Filter* é ilustrado na Definição 5.4.

Definição 5.4: Exemplo de tráfegos de bloqueio

```
1 block in on em0 inet proto { tcp } from 12.12.12.12/24 to/
2 172.16.0.0/24 port 22
```

A execução da função é iniciada com o instanciamento do *array blockTraffics*. Em seguida, as regras carregadas são percorridas, ignorando qualquer linha não iniciada com a palavra "*block*". Por fim, são recuperados os atributos: AF, interface, entidade externa origem, porta de origem, entidade externa de destino, porta de destino, e protocolo. Os atributos são inseridos no objeto da classe *BlockTraffic*, que é inserido no *array blockTraffics*.

5.4 Interface

Nesta seção é apresentada a interface da *Fireasy* e como o usuário pode interagir com ela para modelar políticas de segurança e traduzi-las em regras de *firewall*, e também como carregar regras de *firewall* e edita-las em diagrama SPML2.

5.4.1 Tela principal

Na Figura 5.17 é ilustrada a tela inicial da *Fireasy*. Na barra superior existem opções para ocultar determinados tipos de tráfegos. Este recurso é útil quando o *firewall* possui muitas regras, o que pode dificultar a visualização dos tráfegos. Ocultar tráfegos de saída, por exemplo, pode proporcionar uma melhor visão geral dos tráfegos restantes.

A barra na lateral esquerda contém modelos das entidades que são utilizadas para representar, de cima para baixo: um *host*, uma *network* (sub-rede), um *hostlist* (lista de *hosts*) e uma *unknown network* (rede desconhecida, usualmente a internet). Para criar uma entidade na área principal, basta clicar no modelo e arrastar para o centro. No lado direito da tela está localizado o *inspector* (inspetor), onde são inseridos os atributos das entidades e tráfegos, ilustrados mais adiante nessa seção.

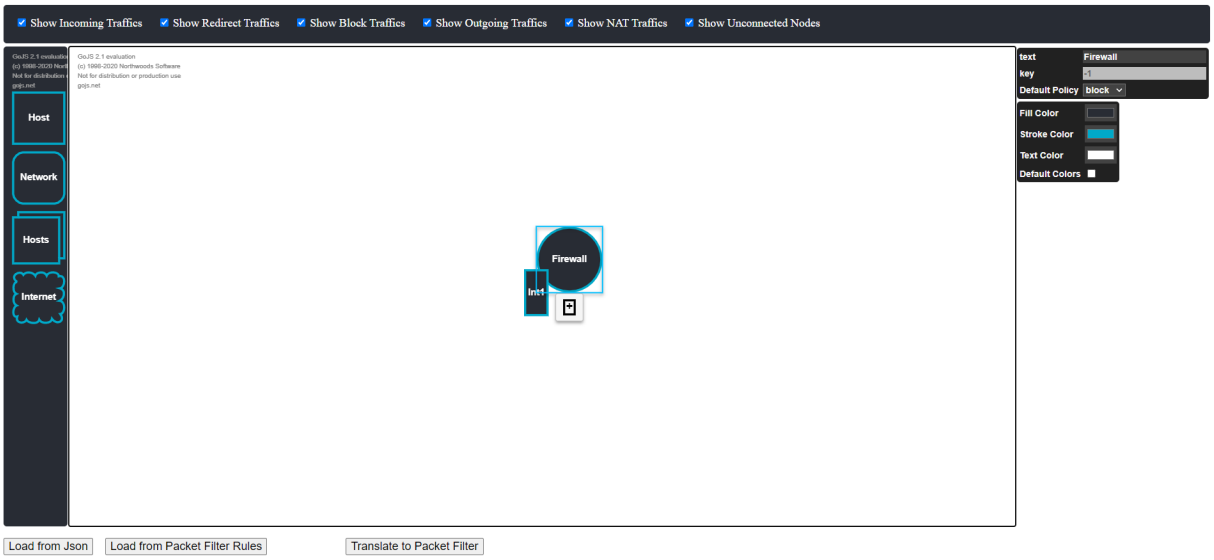


Figura 5.17: Tela inicial da *Fireasy*, com o *firewall* e uma interface criados.

Na parte inferior da tela, existem botões para executar as funções de traduções da ferramenta. O primeiro botão, da esquerda para a direita, chamado "*Load from Json*" é utilizado para carregar um modelo já existente a partir de um arquivo JSON estruturado de acordo com a sintaxe definida pela biblioteca GoJS. Esta função pode ser utilizada para realizar a modelagem inicial das políticas e salvar o JSON gerado para, em outro momento, carrega-lo na ferramenta. A vantagem é que metadados dos tráfegos e entidades são mantidos, como posição, cores e nomes, enquanto carregar regras em linguagem do *Packet Filter* apresenta nomes e cores genéricas e posições dos elementos nem sempre otimizadas.

O segundo botão, rotulado "*Load from Packet Filter Rules*" realiza a tradução das

regras de Packet Filter para diagrama SPML2, além de também apresentar as regras em metalinguagem da SPML2. O último botão, chamado "*Translate to Packet Filter*", realiza a validação do modelo SPML2, a tradução para metalinguagem SPML e por fim a tradução para regras na linguagem do *Packet Filter*. Exemplos das funcionalidades destes dois botões estão inclusos no decorrer desta seção.

Por fim, ao centro está localizada a área onde são criados e carregados os modelos em diagrama SPML. Neste exemplo, existe um *firewall* e uma interface de rede. A adição de outras interfaces é realizar pressionando o botão com símbolo de "+" que é mostrado quando o *firewall* é selecionado. A remoção de qualquer elemento (*firewall*, interface, entidades ou tráfegos) é realizada selecionado os elementos necessários e pressionando a tecla *delete*. Caso o usuário esteja realizando a remoção do *firewall* ou de uma interface de rede, é requisita a confirmação do usuário, pois a remoção de interfaces também remove todos os tráfegos à ela associados, ao passo que a remoção completa do *firewall* remove todas as interfaces de rede e consequentemente todos os tráfegos. Além das funções já descritas, a ferramenta oferece funções de *zoom* no diagrama, mobilidade dos elementos e tráfegos, além de funcionalidade de desfazer ações (utilizando a combinação de teclas *Ctrl+Z*) e de copiar (*Ctrl+c*) e colar (*Ctrl+v*) elementos.

5.4.2 Criação de tráfegos

Para realizar a criação de um tráfego, são utilizados os chamados *adornments* (adornos), assim como na criação de uma interface de rede no *firewall*. Na Figura 5.18 há um *host* chamado *Host1*. Quando o *host* é selecionado, são apresentados dois botões, o que contém uma linha com seta preta representando um tráfego de entrada, e a vermelha um tráfego de bloqueio. Essas opções estão disponíveis apenas quando são selecionadas entidades, não estando presentes no *firewall* ou em sua interfaces, pois tráfegos de entrada não podem ser originados em uma interface, apenas em entidades externas (*hosts*, *networks*, *hostlists* e *internet*). O tráfego é criado clicando no botão desejado e clicando na interface destino.

Figura 5.18: Criação de tráfego de entrada.

Analogamente, a criação de tráfegos de bloqueio e de redirecionamento são realizados da mesma maneira. O tráfego de redirecionamento é representado por uma linha preta tracejada com uma seta indicando o fluxo, e a *Fireasy* altera a notação de um

tráfego de entrada para redirecionamento automaticamente quando é inserida uma porta de redirecionamento no tráfego de entrada. Na Figura 5.19 são ilustrados três entidades, com as notações dos tráfegos de entrada, redirecionamento e bloqueio, de cima para baixo respectivamente.

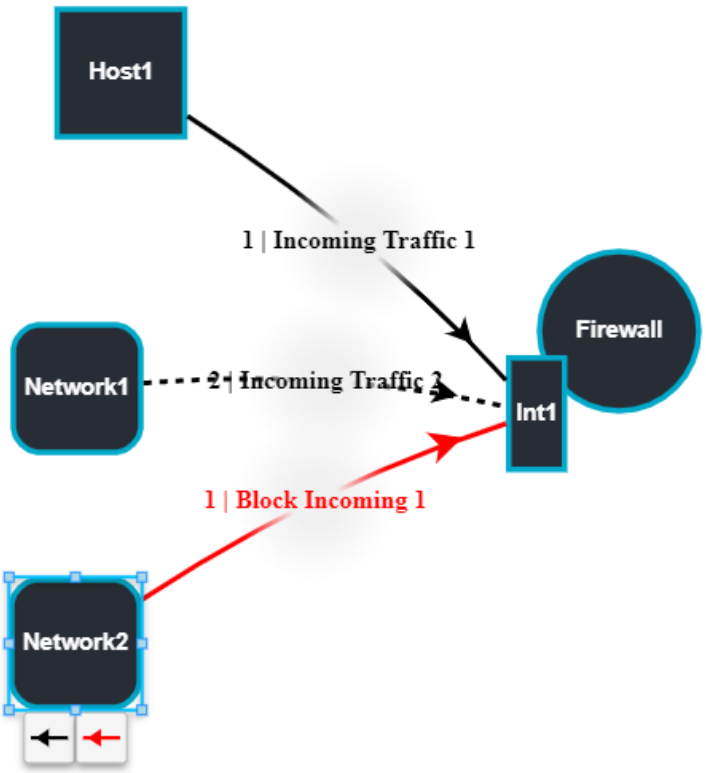


Figura 5.19: Notação de tráfegos de entrada, redirecionamento e bloqueio.

Com relação aos tráfegos de saída, eles só poder ser originados em uma interface. Para que seja possível criar um tráfego de saída, é necessário que exista ao menos um tráfego de entrada adicionado previamente. A inserção do tráfego de saída é feita selecionando uma interface e clicando no botão com uma seta azul, e posteriormente clicando na entidade externa destino do tráfego. Na Figura 5.20 é ilustrada a criação de um tráfego de saída.

5.4.3 Inspetor e atributos de entidades e tráfegos

Os elementos modelados, sejam entidades ou tráfegos, possuem atributos que não são representados no modelo. Informações como IP das entidades, portas e protocolo dos tráfegos, dentre outras, são inseridas utilizando o inspetor localizado na lateral direita ta tela principal da *Fireasy*. Todos os atributos das entidades e tráfegos seguem a gramática da SPML2, apresentada na Definição 4.1.

Na Figura 5.21 são exemplificados os atributos disponíveis no *firewall* e em uma interface. Juntamente aos atributos do *firewall*, é ilustrado um inspetor auxiliar onde é possível alterar as cores do texto, da margem e do preenchimento. Este inspetor está disponível para todas as entidades (*hosts*, *networks*, *hostlist* e *uknown network*),

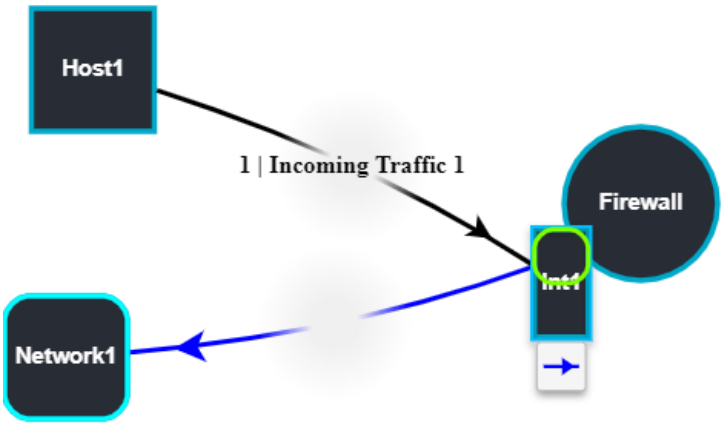
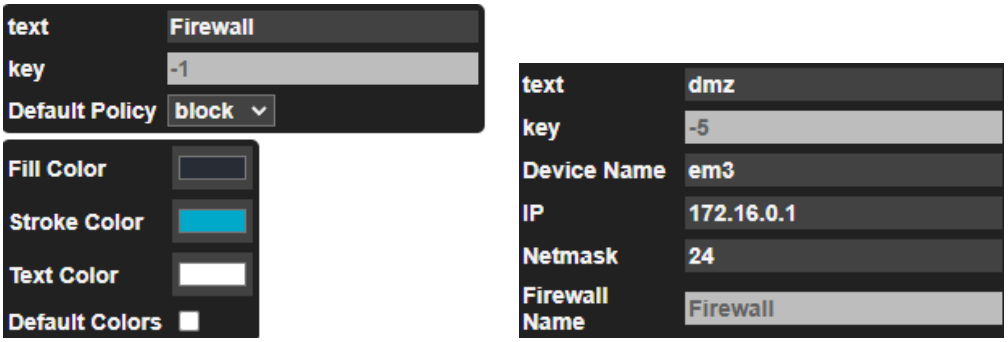


Figura 5.20: Notação de tráfegos de saída.

firewall e interfaces. Ele não é ilustrado no restante dos exemplos, para evitar redundância.

Um *firewall* tem, de acordo com a SPML2, apenas dois atributos, seu nome e sua política padrão, baseado na gramática da SPML2 da Definição 4.1, na linha três. Uma interface possui os atributos: nome, *device*, IP, *netmask* e nome do firewall, conforme linha cinco da Definição 4.1.



(a) (b)

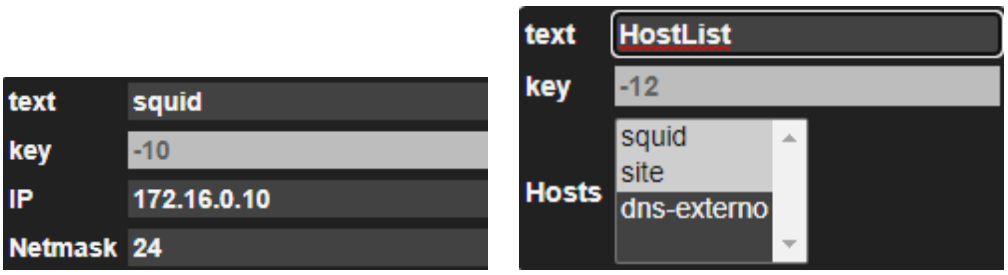
Figura 5.21: (a) Inspetor com atributos de um *firewall* (b) Inspetor com atributos de uma interface.

Na Figura 5.22 são ilustrados os atributos de um *host* (a) e de um *hostlist* (b). Um host, de acordo com a linha sete da Definição 4.1, possui nome, IP e *netmask*. Uma lista de host (*hostlist*) possui nome e os nomes dos hosts contidos na lista, conforme linha 8 da Definição 4.1.

Os atributos de uma sub-rede (*network*) e de uma rede desconhecida (internet ou *unknown network*) são apresentados na Figura 5.23. Uma sub-rede (*network*) possui os atributos nome, prefixo e *netmask*, conforme linha dez da Definição 4.1. Já uma rede desconhecida possui apenas um atributo, o nome, de acordo com a linha onze da Definição 4.1.

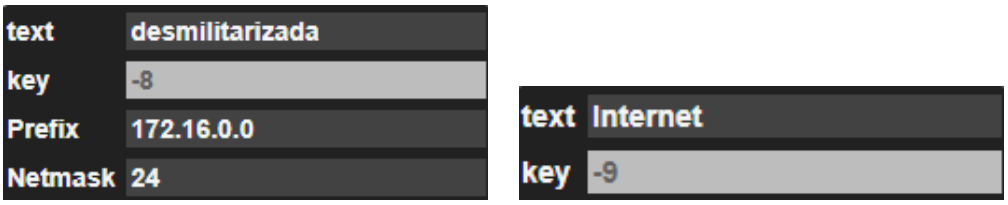
Em relação aos tráfegos de entrada, saída e bloqueio, seus atributos disponíveis são ilustrados na Figura 5.24.

Um tráfego de entrada (a), de acordo com a linha treze da Definição 4.1, possui os



(a) (b)

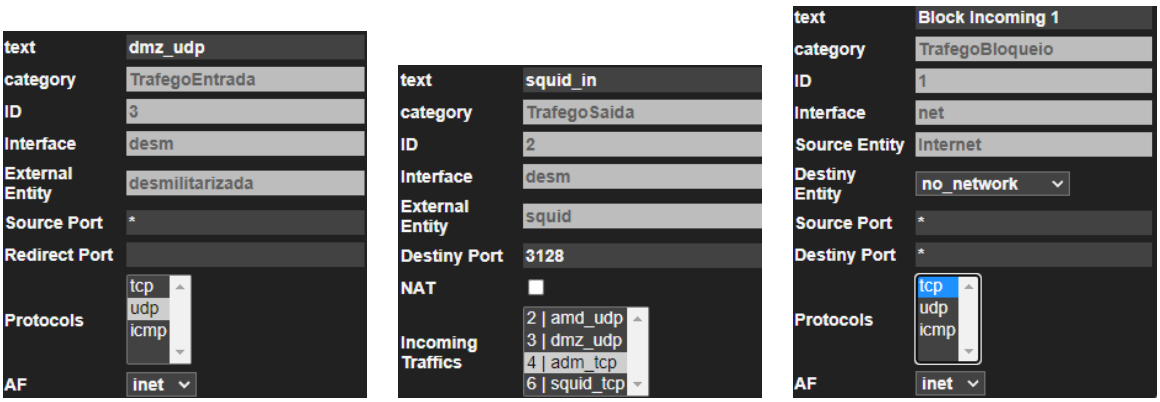
Figura 5.22: (a) Inspetor com atributos de um *host* (b) Inspetor com atributos de uma lista de *hosts*.



(a) (b)

Figura 5.23: (a) Inspetor com atributos de uma *network* (b) Inspetor com atributos de uma *unknown network*.

seguintes atributos: id, nome, nome da interface, AF, entidade externa, porta origem, porta de redirecionamento (opcional) e protocolo. Um tráfego de saída (b), conforme a linha quinze da Definição 4.1, possui: nome, nome da interface, entidade externa, porta de destino, nat (opcional) e um ou mais ids de tráfego de entrada. O tráfego de bloqueio (c) por sua vez, seguindo a gramática da SPML2, linha dezessete da Definição 4.1, possui: id, nome, AF, nome da interface, entidade externa de origem, porta de origem, entidade externa de destino, porta de destino e protocolo.



(a) (b) (c)

Figura 5.24: (a) Inspetor com atributos de um tráfego de entrada (b) Inspetor com atributos de um tráfego de saída (c) Inspetor com atributos de um tráfego de bloqueio.

5.4.4 Traduções

Uma vez que o usuário realiza a modelagem das políticas de segurança em diagrama SPML2, tomando como exemplo o modelo da Figura 5.25, para que o modelo seja

traduzido em regras na linguagem do *Packet Filter*, o usuário deve clicar no botão *Translate to Packet Filter*.

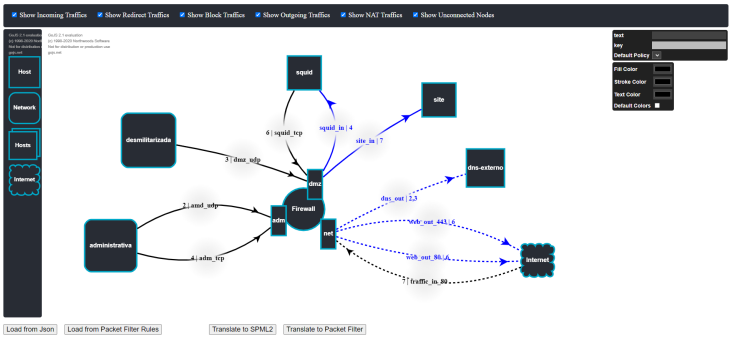


Figura 5.25: Modelo de política de segurança em SPML2.

Conforme explanado na Seção 5.3, quando é requisitada a tradução do diagrama SPML2 em regras de *firewall*, o diagrama é inicialmente traduzido para a metalinguagem SPML2, e posteriormente para regras de *firewall Packet Filter*. A metalinguagem SPML2 gerada de acordo com o modelo da Figura 5.25 é apresentada na *Fireasy* na continuação da tela inicial, realizando uma rolagem na página, ilustrado na Figura 5.26.

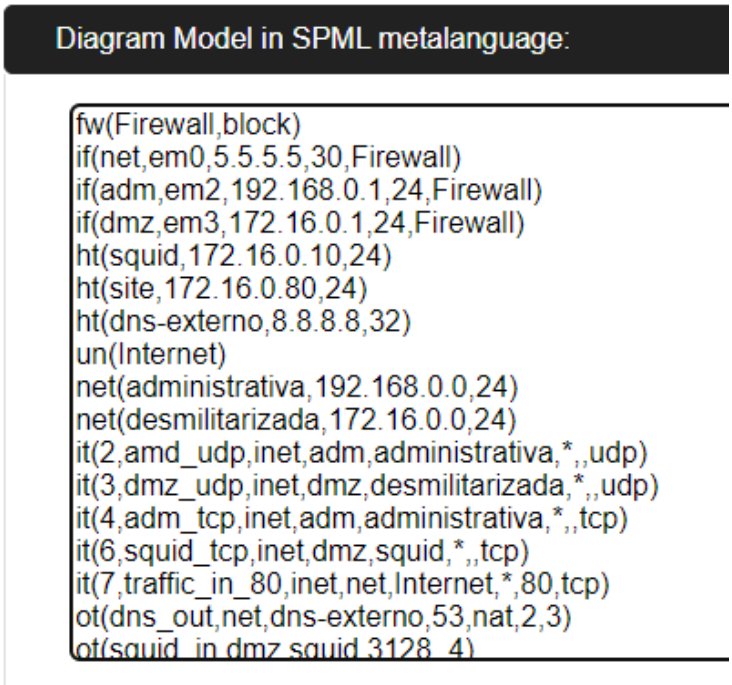


Figura 5.26: Modelo traduzido para metalinguagem SPML2.

Abaixo das regras em metalinguagem SPML2, são mostradas as regras em linguagem do *packet filter*. As regras de *firewall* condizentes com o exemplo da Figura 5.25 são apresentadas na Figura 5.27.

Para realizar o carregamento das regras em linguagem do *Packet Filter*, o usuário deve copiar as regras e colar na área da ferramenta chamada "*Packet Filter Rules*". Posteriormente, clicando no botão *Load from Packet Filter*, as regras são automaticamente traduzidas em metalinguagem SPML2, apresentadas na respectiva área,

Packet-filter rules:

```
# Default Policy
block in all
pass in on em2 inet proto { udp } from 192.168.0.0/24 to 8.8.8.8/32 port 53
pass in on em3 inet proto { udp } from 172.16.0.0/24 to 8.8.8.8/32 port 53
pass in on em2 inet proto { tcp } from 192.168.0.0/24 to 172.16.0.10/24 port 3128
pass in on em3 inet proto { tcp } from 172.16.0.10/24 to any port 80
pass in on em3 inet proto { tcp } from 172.16.0.10/24 to any port 443
pass in on em0 inet proto { tcp } from any to 5.5.5.5/30 port 80 rdr-to 172.16.0.80/24 port 8080
pass out on em0 inet proto { udp } from 192.168.0.0/24 to 8.8.8.8/32 port 53 nat-to 5.5.5.5/30
pass out on em0 inet proto { udp } from 172.16.0.0/24 to 8.8.8.8/32 port 53 nat-to 5.5.5.5/30
pass out on em3 inet proto { tcp } from 192.168.0.0/24 to 172.16.0.10/24 port 3128
pass out on em0 inet proto { tcp } from 172.16.0.10/24 to any port 80 nat-to 5.5.5.5/30
pass out on em0 inet proto { tcp } from 172.16.0.10/24 to any port 443 nat-to 5.5.5.5/30
pass out on em3 inet proto { tcp } from any to 172.16.0.80/24 port 8080
```

Figura 5.27: Regras em linguagem do *Packet Filter*.

e então criado o diagrama. Utilizando as regras da Figura 5.27 como exemplo, e carregando-as na *Fireasy*, o diagrama SPML2 gerado é ilustrado na Figura 5.28.

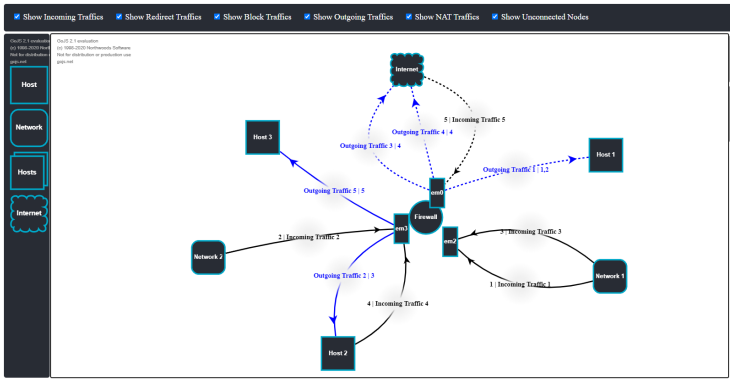


Figura 5.28: Diagrama SPML criado a partir de regras de *firewall*.

5.5 Considerações Finais

A implementação da *Fireasy* conta com, aproximadamente, cinco mil linhas de código, divididas nas camadas apresentadas na Seção 5.3. Os autômatos apresentados neste capítulo representam apenas o fluxo de execução do código da principal funcionalidade da ferramenta, que conta com diversas outras funções que desempenham diferentes papéis.

Além da arquitetura da ferramente, neste capítulo foi apresentada a interface da *Fireasy*. Ela pode ser divida em duas áreas principais: a área de modelagem das regras, em que são instanciadas as entidades do *firewall* e seus tráfegos e atributos; e a área de traduções, que permite realizar a tradução do diagrama SPML2 para linguagem do *Packet Filter*, e também carregar regras escritas em linguagem do *Packer Filter* e apresentá-las em diagrama SPML2.

CAPÍTULO 6

Avaliação da Fireasy: Um Experimento Controlado

6.1 Considerações Iniciais

Para avaliar a *Fireasy*, foi realizado um experimento controlado. Ela foi avaliada para determinar se existe uma melhora nos processos de tradução de políticas de segurança em regras de *firewall*, e na compreensão de regras de *firewalls* já implementados. O experimento foi conduzido de maneira virtual, com quatro alunos voluntários, cursando Ciência da Computação, já tendo cursado a disciplina de Redes de Computadores, e com conhecimentos básicos de *firewalls*. Os participantes realizaram tarefas de modelagem de políticas de segurança e interpretação de regras de *firewall*, utilizando ora a *Fireasy*, ora o método *ad-hoc*, ou seja, sem o auxílio da *Fireasy* ou qualquer outra ferramenta.

Na Seção 6.2 são apresentados os objetivos e métricas do experimento. A Seção 6.3 contém uma descrição dos artefatos utilizados na condução do experimento. Na Seção 6.4 é apresentado o experimento piloto. Na Seção 6.5 é discorrido sobre o experimento principal. Na Seção 6.7 são mostrados e analisados os resultados obtidos. Por fim, na Seção 6.8 são apresentadas as ameaças à validade do experimento.

O experimento seguiu o paradigma *Goal/Question/Metric* (GQM) proposto por Basili (1994), e o Processo de Experimentação apresentado por Wohlin (2000).

6.2 Objetivos e Métricas do Experimento

O experimento realizado teve como objetivo comparar a abordagem apresentada neste projeto (para configuração inicial e edição das regras de *firewalls*) à abordagem tradicional, em que o administrador de redes realiza a configuração inicial e a manutenção do *firewall*, por meio de edições de regras manualmente.

Seguindo o paradigma GQM (*Goal/Question/Metric*), o objetivo do experimento é

descrito da seguinte maneira:

- Objeto de Estudo:
  - Ferramenta *Fireasy*.
- Propósito:
  - Comparar e avaliar os benefícios da utilização da ferramenta desenvolvida no apoio aos administradores de redes em relação à abordagem tradicional, ou seja, sem a utilização da ferramenta, nas seguintes tarefas:
    - Configuração inicial do *firewall* – tradução das políticas de segurança em regras de *firewall*, utilizando a SPML2;
    - Manutenção do *firewall* – inserir, remover ou alterar as regras de *firewall* por meio da representação destas na linguagem SPML2;
- Enfoque de Qualidade:
  - Tempo;
  - Acurácia;
- Perspectiva:
  - Do experimentador.
- Contexto:
  - Laboratório de informática da Universidade Estadual Paulista Júlio de Mesquita Filho, Unesp, campus da Faculdade de Ciência e Tecnologia, Presidente Prudente.

Portanto, seguindo o paradigma GQM, o experimento controlado tem como objetivo analisar a ferramenta *Fireasy* com o propósito de avaliar sua utilização (comparando com a abordagem tradicional) em relação ao tempo e à acurácia das configurações realizadas, da perspectiva do experimentador, no contexto de um laboratório de pesquisa.

Para auxiliar na verificação da validade da abordagem, as seguintes questões (identificadas pela letra *Q*) são levantadas. Para responde-las, as métricas (identificadas pela letra *M*) devem ser aplicadas.

- **Q1**: Utilizar a ferramenta auxilia no processo de tradução das políticas de segurança em regras de *firewall*?
  - **M1.1**: Quantidade de políticas de segurança descritas inicialmente.
  - **M1.2**: Quantidade de políticas de segurança corretamente traduzidas em regras de *firewall*.
- **Q2**: Utilizar a ferramenta auxilia no processo de compreensão de regras de *firewalls* já implantados?

  - **M2.1**: Quantidade de regras presentes no *firewall*.
  - **M2.2**: Quantidade de políticas de segurança corretamente descritas.

- **Q3**: Utilizar a ferramenta tornam os processos descritos em *Q1* e *Q2* mais eficientes (mais rápidos)?
  - **M3.1**: Tempo médio (em minutos) despendido em cada processo.
  - Métricas *M1.2* e *M2.2*.

6.2.0.1 Planejamento

Seguindo o Processo de Experimentação proposto por Wohlin (2000), o experimento controlado foi planejado como se segue.

Para formulação das hipóteses, foi analisada a utilização da ferramenta com relação à eficácia e eficiência nos processos de tradução de políticas de segurança e manutenção de regras de *firewalls*.

As hipóteses relativas à eficácia no processo de tradução de políticas de segurança em regras de *firewall* são as seguintes:

- **Hipótese Nula** ($\mathbf{H}_0^1$): A utilização da ferramenta não proporciona maior eficácia na tradução de políticas de segurança em regras de *firewall* quando comparada à abordagem tradicional (sem a ferramenta).
  (**Eficácia-Tradução**$_{ferramenta}$) $\leq$ (**Eficácia-Tradução**$_{tradicional}$)
  Se os resultados obtidos demonstrarem que a eficácia na tradução das políticas de segurança em regras de *firewall* é menor ou igual à tradução sem a utilização da ferramenta, então a hipótese nula não pode ser rejeitada. Esta hipótese é baseada na quantidade de políticas de segurança corretamente traduzidas em regras de *firewall* pelos participantes do experimento. Para a verificação da hipótese, deve-se utilizar a questão *Q1* e as métricas *M1.1* e *M1.2*.

- **Hipótese Alternativa** ($\mathbf{H}_1^1$): A utilização da ferramenta proporciona maior eficácia na tradução de políticas de segurança em regras de *firewall* quando comparada à abordagem tradicional (sem a ferramenta).
  (**Eficácia-Tradução**$_{ferramenta}$) $>$ (**Eficácia-Tradução**$_{tradicional}$)
  Se os resultados obtidos demonstrarem que a eficácia na tradução das políticas de segurança em regras de *firewall* é maior comparado à tradução sem a utilização da ferramenta, então a hipótese nula ($\mathbf{H}_0^1$) pode ser rejeitada. Esta hipótese é baseada na quantidade de políticas de segurança corretamente traduzidas em regras de *firewall* pelos participantes do experimento. Para a verificação da hipótese, é utilizada a mesma questão e métricas da Hipótese Nula ($\mathbf{H}_0^1$).

As hipóteses relativas à eficácia na compreensão das regras de *firewall* são as seguintes:

- **Hipótese Nula** ($\mathbf{H}_0^2$): A utilização da ferramenta não proporciona maior eficácia na compreensão de regras de *firewalls* implantados, quando comparada à abordagem tradicional (sem a ferramenta).

(**Eficácia-Manutenção**$_{ferramenta}$) $\leq$ (**Eficácia-Manutenção**$_{tradicional}$)
Se os resultados obtidos demonstrarem que a eficácia na compreensão das regras de *firewall* é menor ou igual à compreensão sem a utilização da ferramenta, então a hipótese nula não pode ser rejeitada. Esta hipótese é baseada na quantidade de políticas de segurança corretamente transcritas a partir das regras pelos participantes do experimento. Para a verificação da hipótese, deve-se utilizar a questão *Q2* e as métricas *M2.1* e *M2.2*.

- **Hipótese Alternativa** ($\mathbf{H}_1^2$): A utilização da ferramenta proporciona maior eficácia na compreensão de regras de *firewall* quando comparada à abordagem tradicional (sem a ferramenta).
(**Eficácia-Manutenção**$_{ferramenta}$) $>$ (**Eficácia-Manutenção**$_{tradicional}$)
Se os resultados obtidos demonstrarem que a eficácia na compreensão das regras de *firewall* é maior comparado à compreensão sem a utilização da ferramenta, então a hipótese nula ($\mathbf{H}_0^2$) pode ser rejeitada. Esta hipótese é baseada na quantidade de políticas de segurança corretamente transcritas a partir das regras pelos participantes do experimento. Para a verificação da hipótese, é utilizada a mesma questão e métricas da Hipótese Nula ($\mathbf{H}_0^2$).

As hipóteses relativas à eficiência no processo de tradução de políticas de segurança em regras de *firewall* são as seguintes:

- **Hipótese Nula** ($\mathbf{H}_0^3$): A utilização da ferramenta não proporciona maior eficiência na tradução de políticas de segurança em regras de *firewall* quando comparada à abordagem tradicional (sem a ferramenta).
(**Eficiência-Tradução**$_{ferramenta}$) $\leq$ (**Eficiência-Tradução**$_{tradicional}$)
Se os resultados obtidos demonstrarem que a eficiência na tradução das políticas de segurança em regras de *firewall* é menor ou igual à tradução sem a utilização da ferramenta, então a hipótese nula não pode ser rejeitada. Esta hipótese é baseada na quantidade de políticas de segurança corretamente traduzidas em regras de *firewall* em relação ao tempo despendido pelos participantes do experimento. Para a verificação da hipótese, deve-se utilizar as questões *Q1* e *Q3*, e as métricas *M1.1*, *M1.2* e *M3.1*.

- **Hipótese Alternativa** ($\mathbf{H}_1^3$): A utilização da ferramenta proporciona maior eficiência na tradução de políticas de segurança em regras de *firewall* quando comparada à abordagem tradicional (sem a ferramenta).
(**Eficiência-Tradução**$_{ferramenta}$) $>$ (**Eficiência-Tradução**$_{tradicional}$)
Se os resultados obtidos demonstrarem que a eficiência na tradução das políticas de segurança em regras de *firewall* é maior comparado à tradução sem a utilização da ferramenta, então a hipótese nula ($\mathbf{H}_0^3$) pode ser rejeitada. Esta hipótese é baseada na quantidade de políticas de segurança corretamente traduzidas em regras de *firewall* em relação ao tempo despendido pelos participantes do experimento. Para a verificação da hipótese, é utilizada a mesma questão e métricas da Hipótese Nula ($\mathbf{H}_0^3$).

As hipóteses relativas à eficiência na compreensão das regras de *firewall* são as seguintes:

- **Hipótese Nula** ($\mathbf{H}_0^4$): A utilização da ferramenta não proporciona maior eficiência na compreensão de regras de *firewalls* implantados, quando comparada à abordagem tradicional (sem a ferramenta).
  (**Eficiência-Manutenção**$_{ferramenta}$) $\leq$ (**Eficiência-Manutenção**$_{tradicional}$)
  Se os resultados obtidos demonstrarem que a eficiência na compreensão das regras de *firewall* é menor ou igual à compreensão sem a utilização da ferramenta, então a hipótese nula não pode ser rejeitada. Esta hipótese é baseada na quantidade de políticas de segurança corretamente transcritas a partir das regras em relação ao tempo despendido pelos participantes do experimento. Para a verificação da hipótese, deve-se utilizar as questões *Q2* e *Q3*, e as métricas *M2.1*, *M2.2* e *M3.1*.

- **Hipótese Alternativa** ($\mathbf{H}_1^4$): A utilização da ferramenta proporciona maior eficiência na compreensão de regras de *firewall* quando comparada à abordagem tradicional (sem a ferramenta).
  (**Eficiência-Manutenção**$_{ferramenta}$) $>$ (**Eficiência-Manutenção**$_{tradicional}$)
  Se os resultados obtidos demonstrarem que a eficiência na compreensão das regras de *firewall* é maior comparado à compreensão sem a utilização da ferramenta, então a hipótese nula ($\mathbf{H}_0^4$) pode ser rejeitada. Esta hipótese é baseada na quantidade de políticas de segurança corretamente transcritas a partir das regras em relação ao tempo despendido pelos participantes do experimento. Para a verificação da hipótese, é utilizada a mesma questão e métricas da Hipótese Nula ($\mathbf{H}_0^4$).

As variáveis consideradas no experimento são classificadas da seguinte maneira:

- Variáveis independentes:
  - Quantidade de políticas de segurança previamente definidas que deverão ser traduzidas em regras de *firewall*. Por meio dessa quantidade será possível verificar os índices de eficácia e eficiência na tradução de políticas de segurança em regras de *firewall*.
  - Quantidade de regras de *firewall* que deverão ser interpretadas. Por meio dessa quantidade será possível verificar os índices de eficácia e eficiência na compreensão das regras de *firewall*.
- Variáveis dependentes:
  - Quantidade de políticas de segurança corretamente traduzidas em regras de *firewall*, com a utilização da ferramenta e sem ela.
  - Índice de eficácia na tradução de políticas de segurança em regras de *firewall*.
    $\frac{PolticasTraduzidas}{PolticasDefinidas}$

Tabela 6.1: Ferramentas utilizados na condução do experimento.

Software	Utilização
Google Chrome	Executar a ferramenta Fireasy
Editor de texto	Escrever as regras de firewall na linguagem do Packet Filter
Fireasy	Realizar a modelagem das políticas de segurança e a interpretação de modelo apresentado em SPML2

- Índice de eficiência na tradução de políticas de segurança em regras de *firewall*.
  $\frac{PolticasTraduzidas}{PolticasDefinidas} / TempoMedioEmMinutos$
- Quantidade de políticas de segurança corretamente produzidas a partir das regras, com a utilização da ferramenta e sem ela.
- Índice de eficácia na compreensão das regras de *firewall*.
  $\frac{PolticasProduzidas}{PolticasDefinidas}$
- Índice de eficiência na compreensão das regras de *firewall*.
  $\frac{PolticasProduzidas}{PolticasDefinidas} / TempoMedioEmMinutos$

Com relação à seleção dos indivíduos participantes do experimento, foram escolhidos alunos de graduação em Ciência da Computação (em estágio avançado do curso). Os alunos possuem conhecimentos mínimos de redes de computadores e do funcionamento básico de *firewalls*. Posteriormente à seleção, os indivíduos receberam treinamento específico da linguagem de *firewall* do Packet Filter. Além disso, eles foram apresentados à ferramenta desenvolvida para a obtenção de condição mínima na realização do experimento.

6.3 Artefatos

Para a realização do experimento, foram utilizados artefatos de diferentes tipos, tanto para apoio como para a aplicação do experimento. São eles: materiais de treinamento (descritos na Subseção 6.3.1), formulários (mencionados na Subseção 6.3.2), ferramentas (relacionados na Tabela 6.1), políticas de segurança (descritas na Subseção 6.3.3), além de planilhas para controle do experimento.

6.3.1 Treinamento

Com o objetivo de preparar os participantes com os conceitos necessários para o experimento, foi conduzido um treinamento. Na Tabela 6.2 são apresentados os treinamentos realizados.

6.3.2 Formulários

Para auxiliar na aplicação do experimento, foram criados formulários para captar informações sobre o perfil dos participantes, para o termo de consentimento, e para *feedback* do experimento. Os formulários utilizados são apresentados na Tabela 6.3.

Tabela 6.2: Treinamentos conduzidos para a realização do experimento.

Treinamento	Objetivo
1	Conceitos de política de segurança e seu funcionamento
2	Apresentação da linguagem de *firewall* utilizada no *Packet Filter*
3	Apresentação da Ferramenta *Fireasy* e suas funcionalidades básicas
4	Condução de um exemplo de política de segurança e sua modelagem utilizando a *Fireasy*, juntamente com as respectivas regras em *Packet Filter*

Tabela 6.3: Formulários utilizados na condução do experimento.

Documento	Utilização
1	Termo de consentimento livre e esclarecido
2	Perfil dos participantes
3	Questionário com *feedback* sobre o treinamento e a ferramenta

6.3.3 Políticas de Segurança

Foram elaboradas três políticas de segurança (Política A, Política B e Política C) com diferentes regras de acesso às redes e hosts da rede interna e externa. Elas foram criadas considerando três instituições fictícias, com acessos a serviços de DNS, web, e outros acessos pertinentes à instituição. Além destas três políticas, também foi elaborada uma política utilizada exclusivamente no treinamento do experimento. A quantidade de regras contidas em cada política é apresentada na Tabela 6.4.

6.4 Experimento Piloto

Com o objetivo de validar o experimento controlado realizado, em relação ao treinamento e às tarefas do experimento, foi conduzido um experimento piloto.

Inicialmente os participantes receberam uma breve explanação do experimento, seguido do preenchimento do termo de consentimento e do questionário para coletar os seus perfis. Posteriormente foi conduzido o treinamento para habituá-los à ferramenta *Fireasy* e à linguagem do *Packet Filter*. Os participantes foram então divididos em dois grupos e a eles foi entregue a Política A. O Grupo 1 realizou a modelagem das políticas de segurança utilizando a ferramenta *Fireasy*, enquanto o Grupo 2 realizou a tradução das políticas de segurança diretamente na linguagem de *firewall* do *Packet Filter*, sem o auxílio da ferramenta. Terminada esta etapa, foi-lhes entregue a Política B, e o Grupo 1 realizou a modelagem das políticas de segurança diretamente na linguagem de *firewall*, enquanto o Grupo 2 utilizou a *Fireasy* para realizar a modelagem.

Tabela 6.4: Quantidade de regras das políticas de segurança.

Política	Quantidade de Regras
Treinamento	17
A	23
B	23
C	25

O experimento piloto ocorreu em um intervalo de aproximadamente quatro horas.

Com a realização do experimento piloto, foi observado que os seguintes pontos necessitavam de melhoria:

- A quantidade diretivas de segurança da política de segurança utilizada no treinamento estava alta.
- A quantidade de diretivas de segurança das políticas utilizadas no experimento estava alta.
- Foi necessário realizar ajustes em algumas políticas para eliminar ambiguidades.
- O tempo utilizado para treinamento foi insuficiente, prejudicando a compreensão da *Fireasy* por parte de alguns participantes.

O resultado desse experimento não foi levado em consideração para as conclusões deste projeto, visto que o experimento piloto visa apenas a aprimorar os processos que constituem o experimento principal, para que os resultados do deste sejam mais precisos. Além disso, os participantes do experimento piloto não participaram do experimento principal.

6.5 Experimento Controlado

Com a experiência adquirida na realização do experimento piloto, e com as informações provenientes do questionário respondido pelos participantes deste, os artefatos e cronograma foram ajustados, e o experimento principal foi realizado.

Para participar do experimento, foram selecionados alunos de graduação em ciência da computação, cursando a disciplina de Redes II da Faculdade de Ciências e Tecnologia, campus de Presidente Prudente da UNESP. Os alunos já haviam cursado a disciplina de Redes I e possuem conhecimentos básicos em *firewalls*. O experimento foi realizado com quatro alunos, que concordaram em participar de maneira voluntária. Todas as atividades foram realizadas de forma remota (*online*).

6.6 Perfis dos Participantes

Todos os participantes realizaram a assinatura do Termo de consentimento livre e esclarecido, e responderam um questionários para definição de seus perfis. Os dados obtidos são descritos a seguir:

- Nenhum aluno havia realizado a configuração de um *firewall* anteriormente
- Nenhum aluno havia realizado o mapeamento de regras de *firewall* a partir de políticas de segurança
- Nenhum aluno havia estudado técnicas de modelagem de mapeamento de políticas de acesso à rede

Tabela 6.5: Sessões do experimento controlado

Sessão	Política	Grupo	Método utilizado para tarefa
1	A	Grupo 1	Fireasy
		Grupo 2	Ad-hoc
2	B	Grupo 1	Ad-hoc
		Grupo 2	Fireasy
3	C	Grupo 1	Fireasy
		Grupo 2	Ad-hoc

- Nenhum aluno havia realizado a modelagem de políticas de segurança de *firewall* utilizando uma ferramenta visual, onde os elementos que compõem a configuração do *firewall* são representados de maneira gráfica.

6.6.1 Organização e condução do Experimento

Os participantes inicialmente receberam o termo de consentimento livre e esclarecido para assinatura. Então foi-lhes enviado o questionário para definição de seus perfis. Após estas tarefas, o treinamento foi iniciado, sendo conduzido de acordo com as atividades descritas na Tabela 6.2.

Para garantir a aleatoriedade na divisão de grupos, para cada aluno foi atribuído um *id* de 1 a 4, e posteriormente utilizada uma ferramenta para gerar aleatoriamente dois grupos a partir dos *ids* informados.

As tarefas do experimento foram divididas em três sessões, conforme ilustrado na Tabela 6.5. A primeira sessão foi realizada no primeiro dia, juntamente ao treinamento, enquanto as sessões 2 e 3 foram realizados no segundo dia do experimento.

Para a primeira sessão, foi disponibilizado aos participantes a política da instituição A, descrita no Apêndice A.1. O Grupo 1 realizou a modelagem das políticas de segurança utilizando a *Fireasy* para criar o modelo, ao passo que o Grupo 2 utilizou apenas texto para escrever as regras de *firewall* em linguagem do *Packet Filter*, ou seja, sem a utilização da *Fireasy*.

Na segunda sessão os papéis se inverteram. Os participantes receberam o descritivo da política de segurança da instituição B, apresentada no Apêndice A.3, em que o Grupo 1 realizou a escrita das regras de *firewall* em linguagem do *Packet Filter* utilizando apenas texto, e o Grupo 2 utilizou a *Fireasy* para fazer a modelagem da política de segurança por meio dos diagramas SPML2.

Por fim, na terceira e última sessão, foi realizada uma tarefa para avaliar se a utilização da *Fireasy* poderia auxiliar na compreensão de regras de *firewalls* já configurados. Para tanto, as regras de *firewall* da instituição C, descritas no Apêndice A.5, foram disponibilizadas aos participantes de maneiras distintas. O Grupo 1 realizou o carregamento das regras em linguagem do *Packet Filter* na *Fireasy*, e partir do modelo gerado pela ferramenta (que representa as regras carregadas), deveriam descrever as políticas de segurança da instituição C. O Grupo 2 recebeu as regras apenas em linguagem do *Packet Filter*, e sem a utilização da *Fireasy*, descreveram as políticas de segurança representadas pelas regras.

Tabela 6.6: Cronograma de execução do experimento

Dia	Atividade	Tempo despendido
1	Apresentação do experimento	10 minutos
	Assinatura do termo de consentimento livre e esclarecido	10 minutos
	Preenchimento do questionário de perfil	5 minutos
	Treinamento - Conceitos de política de segurança	10 minutos
	Treinamento - Packet Filter e Fireasy	45 minutos
	Política A - Configuração	30 minutos
2	Política B - Configuração	30 minutos
	Política C - Interpretação	30 minutos
	Preenchimento do questionário de Feedback do Treinamento e Fireasy	10 minutos

O cronograma contido na Tabela 6.6 descreve as atividades conduzidas e seus respectivos tempos despendidos.

6.6.1.1 Obtenção dos resultados

Para verificar se os resultados das atividades realizadas pelos participantes estavam corretos, foram utilizados três métodos, de acordo com a tarefa realizada.

Quando a tarefa do participante foi realizar a modelagem das políticas de segurança utilizando a *Fireasy*, exemplificado pela Figura 6.1, o participante enviou o *Json* gerado automaticamente. Posteriormente, com o arquivo *Json* em posse do experimentador, ele foi carregado na *Fireasy* e traduzido para linguagem de *Packet Filter*. Por fim, as regras geradas, que representam o modelo criado pelo participante, foram comparadas com as regras do oráculo da tarefa realizada, podendo ser da instituição A ou B, listadas nos oráculos, apresentados nos Apêndices A.1 e A.3 respectivamente. A quantidade de regras corretamente realizadas representou a pontuação do participante.

Para a tarefa de transcrição das política de segurança em regras de *firewall* sem a utilização da *Fireasy*, os participantes utilizaram apenas um editor de texto para escrevê-las. Posteriormente, em posse do experimentador, as regras escritas pelos participantes foram comparadas com as regras dos oráculos das instituições A e B. A quantidade de regras corretamente realizadas representou a pontuação do participante. É importante ressaltar que nesse caso, as regras são consideradas em pares, pois um tráfego completo deve possuir uma regra de entrada e saída. Regras designando apenas o tráfego de entrada (*pass in*), sem seu correspondente tráfego de saída (*pass out*) não foram consideradas corretas. O mesmo não acontece com as regras modeladas utilizando a Fireasy, pois a ferramenta gera automaticamente os pares de regras (entrada e saída).

A comparação das regras produzidas por um participante, em relação ao oráculo pode ser observada na Figura 6.2, onde a coluna da esquerda representa as regras produzidas pelo aluno, ou traduzidas por meio da *Fireasy*, e a coluna da direita o oráculo.

Por fim, para a tarefa da sessão 3, em que os participantes produziram políticas

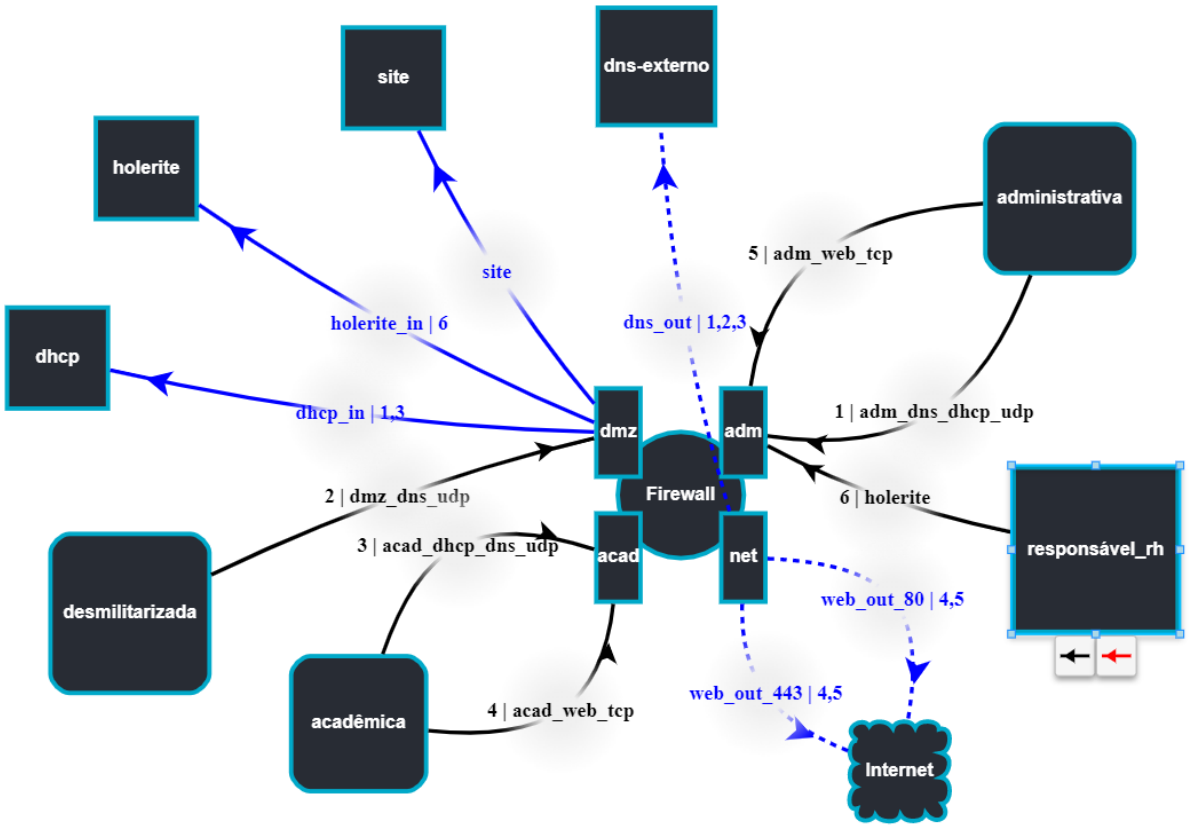


Figura 6.1: Modelo de uma política de segurança realizado por um participante.

	Participante 2	Oráculo
2	pass in on em2 inet proto { udp } from 10.0.0.0/16 to 1.1.1.1/32 port 53	block in all
3	pass in on em3 inet proto { udp } from 192.168.1.0/24 to 1.1.1.1/32 port 53	pass in on em4 inet proto { udp } from 172.16.0.0/24 to 1.1.1.1/32 port 53
4	pass in on em4 inet proto { udp } from 172.16.0.1/24 to 1.1.1.1/32 port 53	pass in on em3 inet proto { udp } from 192.168.1.0/24 to 1.1.1.1/32 port 53
5	pass out on em0 inet proto { udp } from 10.0.0.0/16 to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30	pass in on em2 inet proto { udp } from 10.0.0.0/16 to 1.1.1.1/32 port 53
6	pass out on em0 inet proto { udp } from 192.168.1.0/24 to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30	pass in on em2 inet proto { tcp } from 10.0.0.0/16 to any port 80
7	pass out on em0 inet proto { udp } from 172.16.0.1/24 to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30	pass in on em2 inet proto { tcp } from 10.0.0.0/16 to any port 443
8	pass in on em2 inet proto { tcp } from 10.0.0.0/16 to any port 80	pass in on em2 inet proto { tcp } from 10.0.0.0/16 to 172.16.0.30/24 port 2222
9	pass in on em2 inet proto { tcp } from 10.0.0.0/16 to any port 443	pass in on em2 inet proto { tcp } from 10.0.0.0/16 to 96.127.0.1/32
10	pass in on em3 inet proto { tcp } from 192.168.1.0/24 to any port 80	pass in on em3 inet proto { tcp } from 192.168.1.0/24 to any port 80
11	pass in on em3 inet proto { tcp } from 192.168.1.0/24 to any port 443	pass in on em3 inet proto { tcp } from 192.168.1.0/24 to any port 443
12	pass out on em0 inet proto { tcp } from 10.0.0.0/16 to any port 80 nat-to 5.5.5.5/30	pass in on em3 inet proto { tcp } from { 192.168.1.10/24, 192.168.1.20/24 } to 172.16.0.40/24 port 8080
13	pass out on em0 inet proto { tcp } from 10.0.0.0/16 to any port 443 nat-to 5.5.5.5/30	pass in on em5 inet proto { udp } from 192.168.100.0/24 to 172.16.0.50/24 port 161
14	pass out on em0 inet proto { tcp } from 192.168.1.0/24 to any port 80 nat-to 5.5.5.5/30	pass out on em1 inet proto { udp } from 172.16.0.0/24 to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30
15	pass out on em0 inet proto { tcp } from 192.168.1.0/24 to any port 443 nat-to 5.5.5.5/30	pass out on em1 inet proto { udp } from 192.168.1.0/24 to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30
16	pass in on em4 inet proto { tcp } from 10.0.0.0/16 to 172.16.0.30/24 port 2222	pass out on em1 inet proto { udp } from 10.0.0.0/16 to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30
17	pass out on em2 inet proto { tcp } from 10.0.0.0/16 to 172.16.0.30/24 port 2222	pass out on em1 inet proto { tcp } from 10.0.0.0/16 to any port 80 nat-to 5.5.5.5/30
18	pass in on em3 inet proto { tcp } from 192.168.1.10/24 to 172.16.0.40/24 port 8080	pass out on em1 inet proto { tcp } from 192.168.1.0/24 to any port 80 nat-to 5.5.5.5/30
19	pass in on em3 inet proto { tcp } from 192.168.1.20/24 to 172.16.0.40/24 port 8080	pass out on em1 inet proto { tcp } from 10.0.0.0/16 to any port 443 nat-to 5.5.5.5/30
20	pass out on em4 inet proto { tcp } from 192.168.1.10/24 to 172.16.0.40/24 port 8080	pass out on em1 inet proto { tcp } from 192.168.1.0/24 to any port 443 nat-to 5.5.5.5/30
21	pass out on em4 inet proto { tcp } from 192.168.1.20/24 to 172.16.0.40/24 port 8080	pass out on em4 inet proto { tcp } from 10.0.0.0/16 to 172.16.0.30/24 port 2222
22	pass in on em5 inet proto { udp } from 192.168.100.0/24 to 172.16.0.50/24 port 161	pass out on em4 inet proto { tcp } from { 192.168.1.10/24, 192.168.1.20/24 } to 172.16.0.40/24 port 8080
23	pass out on em4 inet proto { udp } from 192.168.100.0/24 to 172.16.0.50/24 port 161	pass out on em4 inet proto { udp } from 192.168.100.0/24 to 172.16.0.50/24 port 161
24	pass in on em2 inet proto { tcp } from 10.0.0.0/16 to 96.127.0.1/32 port *	pass out on em1 inet proto { tcp } from 10.0.0.0/16 to 96.127.0.1/32 nat-to 5.5.5.5/30
25	pass out on em1 inet proto { tcp } from 10.0.0.0/16 to 96.127.0.1/32 nat-to 5.5.5.5/30	

Figura 6.2: Exemplo de verificação de regras produzidas por participante do experimento.

de segurança, alguns a partir de regras em linguagem do *Packet Filter*, e outros a partir de diagramas da *Fireasy*, as políticas foram comparadas com as descritas no Apêndice A.7. Pelo fato das políticas de segurança realizada pelos participantes terem sido escritas de formas distintas umas das outras, foram consideradas corretas políticas que descreveram o serviço contendo *host*, *hostlist* ou *sub-rede*, porta, protocolo, localização dele (em qual interface de rede ou sub-rede está conectado), e quais *hosts*, *hostlists* ou *sub-rede* possuem acesso a ele.

6.7 Análise dos Resultados

Nesta Seção são apresentados os resultados obtidos com o experimento, sendo divida em duas partes. Na Subseção 6.7.1, são apresentados e analisados os resultados da configuração das políticas de segurança das instituições A e B, em que os participantes receberam o descritivo das políticas e realizar a modelagem das regras ora utilizando a *Fireasy*, ora a abordagem adhoc. Na Subseção 6.7.2 são descritos e analisados os resultados obtidos a partir da compreensão das regras de *firewall* da instituição C, realizada por meio da *Fireasy* por um grupo, e o outro apenas analisando as regras de *firewall* em linguagem do *Packer Filter*.

6.7.1 Configuração das Políticas de Segurança das Instituições A e B

Na Tabela 6.7 são apresentados os resultados obtidos nas configurações das políticas de segurança das instituições A e B. É importante ressaltar que os *ids* dos participantes não são os mesmos utilizados para a aleatorização dos grupos. Na tabela é possível observar que houve um aumento da eficiência quando a *Fireasy* foi utilizada em três dos quatro participantes. Com relação ao participante 2, seu desempenho pode ser afetado pelo fato de que a configuração da política A foi realizada no mesmo dia do treinamento, enquanto a política B foi configurada no segundo dia do experimento, sendo possível que o participante tenha se habituado melhor com a *Fireasy* e a linguagem do *Packet Filter*. Observando o desempenho do participante na compreensão das regras da instituição C, apresentado na Tabela 6.9, houve uma melhora em relação ao desempenho na configuração da Política A, o que reforça a hipótese de que o participante se habituou melhor tanto à *Fireasy* quanto a linguagem do *Packet Filter* no segundo dia do treinamento.

Para avaliar o resultado do experimento aplicado com as políticas de segurança das instituições A e B, foi utilizado o teste de amostras pareadas de Wilcoxon Wilcoxon (1945).

Este teste utiliza diferentes abordagens para o cálculo de acordo com o tamanho da amostragem, ou seja, ele se adapta ao número de casos presentes na amostra. Fundamentalmente, ele busca verificar se a hipótese nula de que não há diferença entre as duas amostrar pareadas seja válida ou não, utilizando o valor do *p-value* (ou significância) calculado para tanto. Se o valor do *p-value* for menor do que 0.05, é possível afirmar com 95% de certeza que existe uma diferença entre as amostras testadas, e que elas não ocorreram por acaso Fávero e Belfiore (2017).

Tabela 6.7: Desempenho dos participantes na configuração das Políticas A e B.

Grupo	**Participante**	**Política A**		**Política B**	
1		**Tempo Fireasy**	**Quantidade regras corretas**	**Tempo Adhoc**	**Quantidade regras corretas**
	1	30 min	21/23	30 min	16/23
	2	30 min	1/23	20 min	6/23
2		**Tempo Adhoc**	**Quantidade regras corretas**	**Tempo Fireasy**	**Quantidade regras corretas**
	3	28 min	14/23	29 min	23/23
	4	27 min	0/23	20 min	1/23

Tabela 6.8: Teste de Wilcoxon aplicado comparando o método utilizado com: Divisão entre eficácia por eficiência dos participantes, e quantidade de regras corretas realizadas pelos participantes.

	Eficácia/Eficiência - Método Utilizado	Quantidade de regras corretas - Metodo Utilizado
Z	-2.521	-1.992
Asymp. Sig. (2-tailed)	0.012	0.046

O resultado do teste de Wilcoxon aplicado às amostras pode ser observado na Tabela 6.8. Foi utilizada a ferramenta de análise estatística IBM SPSS (SPSS (2021)) para realização do teste. Ele foi executando levando em consideração o método utilizado, ou seja, se o participante utilizou a *Fireasy* ou método adhoc, em relação à: quantidade de regras corretas, e também a divisão entre a eficácia (quantidade de regras corretas) pela eficiência (tempo despendido). Em ambos os casos, o valor do *p-value* for inferior a 0.05, sendo possível afirmar com 95% de certeza que existe uma diferença entre as amostras testas, não ocorridas pelo acaso.

Na Figura 6.3 é ilustrado um gráfico apresentando o desempenho médio, utilizando os valores obtidos pela divisão entre a eficácia e eficiência de cada participante, em relação ao método utilizado nas políticas A e B. É possível observar que o desempenho com a utilização da *Fireasy* foi significantemente melhor na Política A, e ligeiramente melhor na Política B. Portanto, as hipóteses nulas apresentadas na Subseção 6.2.0.1 referentes à tradução de políticas de segurança em regras de *firewall* podem ser refutadas.

Portanto, a hipótese nula $\mathbf{H}_0^1$, que indica que a utilização da ferramenta não proporciona maior eficácia na tradução de políticas de segurança em regras de *firewall* quando comparada à abordagem tradicional, pode ser rejeitada. Em relação à eficiência para a mesma tarefa, a hipótese nula $\mathbf{H}_0^3$ não pode ser rejeitada. No entanto, é

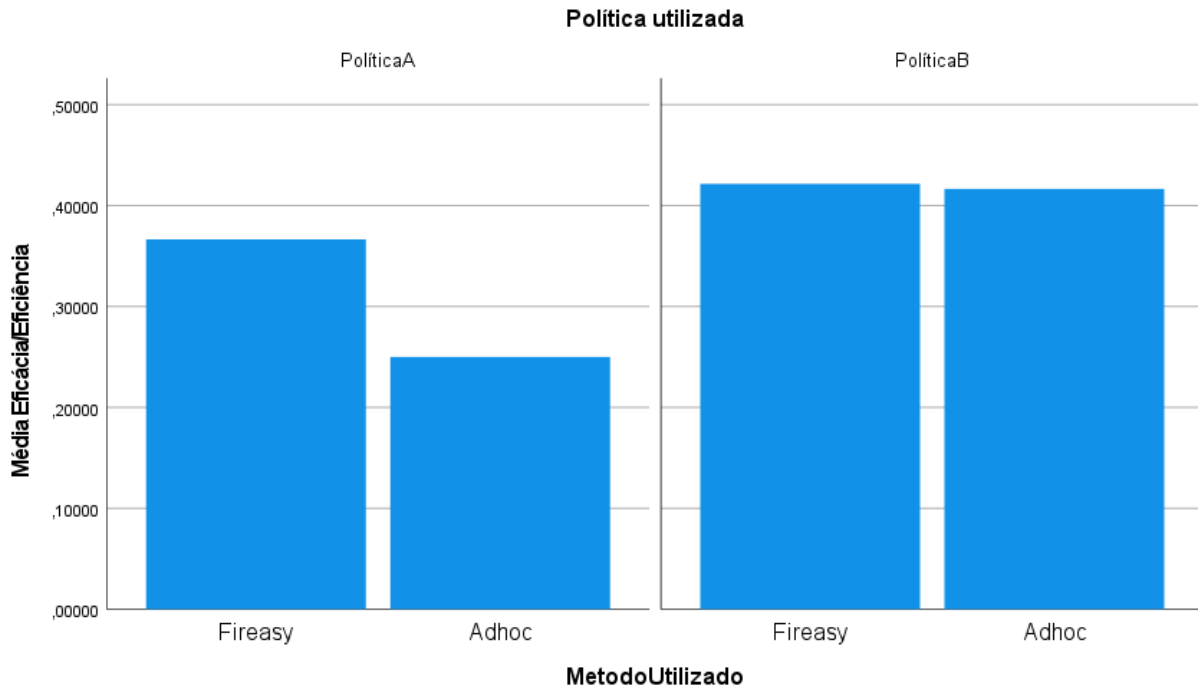


Figura 6.3: Gráfico com o desempenho médio dos participantes nas Políticas A e B, comparando os métodos utilizando a *Fireasy* ou adhoc. Os valores considerados foram obtidos pela divisão da eficácia (quantidade de regras corretas) pela eficiência (tempo despendido para a tarefa).

importante observar que a não melhora da eficiência pode ter sido causada pelo fato dos participantes terem levado algum tempo para se habituar à ferramenta.

6.7.2 Compreensão das regras da instituição C

Com relação à compreensão das regras da instituição C, foi realizada uma análise qualitativa dos resultados. Na Tabela 6.9 estão contidos os dados obtidos, em que o grupo 1 ficou responsável por descrever as políticas de segurança da instituição C a partir de um modelo previamente carregado na *Fireasy*, enquanto ao grupo 2 foi entregue um arquivo com regras de *firewall* na linguagem do *Packet Filter*, para a partir delas descrever as políticas de segurança.

Ao analisar cada participante individualmente, é possível observar que o participante 4 obteve um resultado abaixo dos demais, assim como nas configurações das políticas A e B. Com relação ao participante 3, seu resultado também foi condizente com seu desempenho nas outras tarefas. Esses dois participantes utilizaram o método adhoc para descrever a política de segurança.

No grupo 1, constituído pelos participantes 1 e 2, o resultado foi semelhante para ambos. No entanto, nas outras tarefas, o participante 1 obteve melhores resultados do que o participante 2. Portante é possível concluir que com a utilização da *Fireasy* para a compreensão de regras de *firewall*, houve uma homogeneização do desempenho dos participantes, com o participante 2 obtendo resultado ligeiramente ao participante 1, o qual havia obtido melhor desempenho nas tarefas anteriores.

Com relação à hipótese nula $\mathbf{H}_0^2$, que indica que a utilização da ferramenta não pro-

Tabela 6.9: Desempenho dos participantes no compreendimento das regras da instituição C

Grupo	**Participante**	**Política C**	
1		**Tempo Fireasy**	**Quantidade políticas corretas**
	1	30 min	6/11
	2	28 min	7/11
2		**Tempo Adhoc**	**Quantidade políticas corretas**
	3	19 min	9/11
	4	22 min	2/11

porciona maior eficácia na compreensão de regras de *firewall* implantados, quando comparada à abordagem tradicional, pode ser rejeitada, visto que houve um melhor desempenho dos participantes que utilizaram a *Fireasy* na tarefa. Em relação à eficiência para a mesma tarefa, a hipótese nula $\mathbf{H}_0^4$ não pode ser rejeitada, pois os participantes que utilizaram o método *ad-hoc* levaram menos tempo para desempenhar a tarefa.

6.8 Ameaças à Validade

Nesta Seção são apontadas as medidas tomadas para mitigar as ameaças à validade do experimento, além de descrever as que ainda podem existir.

As seguintes precauções foram utilizadas para diminuir as ameaças à validade do experimento:

- foram definidas métricas seguindo o paradigma *GQM* com o objetivo de analisar os resultados obtidos no experimento
- os grupos participantes foram aleatorizados para evitar ao máximo um desbalanceamento nos desempenhos deles.
- as políticas de seguranças das instituições A, B e C foram elaboradas com cenários distintos, porém condizentes em complexidade, utilizando práticas comumente aplicadas em *firewalls* reais
- o treinamento foi conduzida de maneira igualitária para todos, com espaço para dúvidas ao final de cada sessão
- escolha de um teste apropriado (Wilcoxon) para amostras pareadas e com baixo número de amostras

Apesar das medidas tomadas para diminuir os riscos à validade do experimento, é necessário apontar que o baixo número de participantes no experimento constitui

uma ameaça. Ele foi conduzido de maneira online e inicialmente haviam 23 pessoas presentes. No entanto, no decorrer do treinamento e durante a primeira sessão do experimento, 19 pessoas deixaram a reunião, restando apenas 4 para a continuação do experimento.

6.9 Considerações Finais

Neste capítulo foi apresentada a avaliação da ferramenta desenvolvida por meio de um experimento controlado. Foram definidas métricas e variáveis dependentes e independentes para o experimento, utilizando o paradigma *GQM* e o processo de experimentação de Wohlin (2000). Também foram descritos os artefatos utilizados para suporte à condução do experimento, constituído de ferramentas, planilhas, apresentações e formulários. Foi conduzido o experimento piloto, que permitiu o refinamento dos processos para que o experimento final fosse conduzido.

Por fim, com o experimento final foram descritos os meios de obtenção dos resultados, a aplicação do teste de Wilcoxon para validação e as ameaças à validade. É possível concluir que a *Fireasy* pode proporcionar uma melhora no gerenciamento de regras de *firewalls*, principalmente na tarefa de modelagem inicial das políticas de segurança. Na tarefa de compreensão das regras, a *Fireasy* proporcionou uma homogeneização do desempenho dos participantes, principalmente no Grupo 1.

CAPÍTULO 7

Conclusão

O aumento dos ambientes de redes empresariais e domésticos requer que os esforços para mitigar ameaças virtuais também cresçam. Para isso, é de suma importância que existam ferramentas capazes de auxiliar os administradores de redes a constituir uma segurança de rede sólida. Apesar dos *firewalls* representarem apenas uma das ferramentas necessárias para isso, eles são imprescindíveis, pelo fato deles comporem a principal barreira que delimita os ambientes de redes privados da internet pública.

Ao analisar os aspectos e adversidades relacionados a *firewalls* apresentados neste trabalho, juntamente com os principais trabalhos relacionados da literatura, é evidente a necessidade do desenvolvimento de ferramentas que visem a facilitar o gerenciamento da segurança da rede. Essas facilidades podem ser fornecidas por meio de ferramentas que, de algum modo, façam abstraiam as configurações de *firewalls* em representações amigáveis, tornando possível identificar erros, validar configurações, ou realizar operações de inserção, alteração e remoção de regras.

Os trabalhos de Trevisani e Garcia, 2008 e Sapia (2016) definiram, respectivamente a SPML e a SPML2, sendo esta a evolução da primeira. No entanto, as ferramentas desenvolvidas não realizam a tradução de regras de *firewall* em modelo SPML ou SPML2, o que inviabiliza a sua utilização em *firewalls* que já estejam em produção.

A *Fireasy*, ferramenta apresentada neste projeto, permite: a modelagem inicial das políticas de segurança e posterior tradução automática para regras de *firewall* na linguagem do *Packet Filter*, e também o carregamento de regras escritas em linguagem do *Packet Filter*, traduzindo-as em metalinguagem SPML2, e posteriormente permitindo sua edição por meio de diagrama SPML2.

Com a utilização da *Fireasy*, esperava-se prover um ambiente para administradores de rede que contribuíssem com uma melhoria na eficácia e eficiência nas configurações iniciais de políticas de segurança, além da manutenção de *firewalls* já em produção.

Para avaliar a abordagem proposta, juntamente com a ferramenta desenvolvida,

foram definidas métricas, utilizando o paradigma *GQM* (Basili (1994)), para verificar se houve melhora na eficácia e eficiência das tarefas de modelagem de políticas de segurança e compreensão de regras. Seguindo o processo de experimentação proposto por Wohlin (2000), foram definidas hipóteses com base nas métricas, além de variáveis dependentes e independentes, com o objetivo de melhorar os processos do experimento, bem como sua posterior validação.

Foi conduzido um experimento piloto, posterizado pelo experimento principal. Para avaliar os resultados obtidos, foi aplicado o teste estatístico de amostras pareadas de Wilcoxon (1945), o qual indiciou que, com 95% de certeza, havia uma diferença entre utilizar a *Fireasy* ou o método Adhoc para realizar a modelagem de políticas de segurança, em que o desempenho dos que utilizaram a *Fireasy* foi superior.

Na tarefa de compreensão de regras de *firewall*, foi realizada uma análise qualitativa dos resultados, indicando que os participantes que utilizaram a *Fireasy* obtiveram um resultado médio superior aos que utilizaram o método Adhoc.

Vale ressaltar que, embora não tenha significado estatístico, os alunos que tiveram mais facilidade na execução das tarefas obtiveram resultado melhores quando utilizaram a *Fireasy*.

7.1 Contribuições, Limitações e Dificuldades

Com o desenvolvimento da ferramenta proposta, pretende-se oferecer ao administrador de redes um ambiente em que seja possível gerenciar as regras de *firewall* de maneira gráfica. Com a utilização de elementos gráficos, o processo de modelagem das políticas de segurança e a manutenção das regras de *firewall* são feitas sem a necessidade de que o administrador possua conhecimento avançado da linguagem nativa de *firewall* utilizada.

Além disso, a ferramenta pode ser utilizada para fins educacionais, para o ensinamento de conceitos de *firewall*. Os alunos podem aprender sobre fluxo de tráfego de rede, topologia de rede, filtragem de tráfego, definição de políticas de segurança, dentre outros.

A *Fireasy* está disponível no endereço “github.com/leandromeira/Fireasy”, com o seu código aberto. Alem disso, foi utilizada a licença do MIT (2022), que permite a livre utilização e alteração da ferramenta.

Com relação às limitações, para *firewall* com uma quantidade muito expressiva de regras, a visualização pode se tornar difícil. No entanto, foram implementadas funções para filtragem dos tipos de tráfegos que são apresentados na ferramenta, com o objetivo de permitir que o usuário se concentre no tipo de tráfego necessário no momento. Além disso, a ferramenta é capaz de carregar regras na linguagem do *Packet Filter* utilizando a sintaxe geralmente utilizada. Regras avançadas com utilizações de variáveis não são suportadas.

No tocante às dificuldades encontradas, destaca-se a baixa quantidade de amostrar obtidas para o experimento. Além disso, foi desafiador elaborar políticas de segurança adequadas para alunos de graduação, pois elas não poderiam ser muito simples a ponto de todos conseguirem ter um desempenho alto, e não tão complexas com uti-

lização de conceitos mais avançados (como por exemplo muitos redirecionamentos de porta).

7.2 Trabalhos Futuros

Com relação aos trabalhos futuros, a continuação da implementação da *Fireasy* pode seguir por três caminhos distintos. Por ter sido implementada em módulos independentes, o módulo de tradução de e para linguagem de *Packet Filter* pode ser reimplementado para suportar outras linguagens de *firewall*. Pode ser implementada funcionalidade de visualização do tráfego por meio de animações. Por fim, a *Fireasy* pode ser estendida com recursos focados em educação, como tutoriais, módulos de acesso, e recursos que indiquem ao aluno os erros cometidos e como corrigi-los.

Além dos trabalhos futuros referentes à implementação da *Fireasy*, podem ser realizados outros experimentos em diferentes cenários, para melhor avaliar a ferramenta.

Referências Bibliográficas

AL-SHAER, E.; HAMED, H.; BOUTABA, R.; HASAN, M. Conflict classification and analysis of distributed firewall policies. *IEEE Journal on Selected Areas in Communications*, v. 23, n. 10, p. 2069–2084, 2005.

AL-SHAER, E. S.; HAMED, H. H. Discovery of policy anomalies in distributed firewalls. In: *IEEE INFOCOM 2004*, 2004, p. 2605–2616 vol.4.

BANIK, S. M.; PENA, L. Deploying agents in the network to detect intrusions. In: *2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*, 2015, p. 83–87.

BARTAL, Y.; MAYER, A.; NISSIM, K.; WOOL, A. Firmato: a novel firewall management toolkit. In: *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344)*, 1999, p. 17–31.

BASILI, V. R. Goal question metric paradigm. *Encyclopedia of software engineering*, p. 528–532, 1994.

BODEI, C.; DEGANO, P.; GALLETTA, L.; FOCARDI, R.; TEMPESTA, M.; VERONESE, L. Language-independent synthesis of firewall policies. In: *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, 2018, p. 92–106.

CERT.BR Incidentes reportados ao cert.br - janeiro a dezembro de 20189(tipos de ataques). Disponível em: https://cert.br/stats/incidentes/2019-jan-dec/tipos-ataque.html, acesso em: 23/05/2020, 2019a.

CERT.BR Incidentes reportados ao cert.br - janeiro a dezembro de 2019 (scan reportados, por porta). Disponível em: https://cert.br/stats/incidentes/2019-jan-dec/scan-portas.html, acesso em: 23/05/2020, 2019b.

CERT.BR Total de incidentes reportados ao cert.br por ano. Disponível em: https://www.cert.br/stats/incidentes/, acesso em: 23/05/2020, 2019c.

CHANDRE, P. R.; MAHALLE, P. N.; SHINDE, G. R. Machine learning based novel approach for intrusion detection and prevention system: A tool based verification. In: *2018 IEEE Global Conference on Wireless Computing and Networking (GCWCN)*, 2018, p. 135–140.

CHENG, Y.; WANG, W.; WANG, J.; WANG, H. Fpc: A new approach to firewall policies compression. *Tsinghua Science and Technology*, v. 24, n. 1, p. 65–76, 2019.

CISCO Zone-based policy firewall design and application guide. Disponível em: https://www.cisco.com/c/en/us/support/docs/security/ios-firewall/98628-zone-design-guide.html, acesso em: 28/07/2019, 2010.

CISCO Firewalls de próxima geração (ngfws). Disponível em: https://www.cisco.com/c/pt_br/products/security/firewalls/index.html, acesso em: 25/07/2019, 2019.

FAIL2BAN Fail2ban. Disponível em: https://www.fail2ban.org/, acesso em: 25/07/2019, 2019.

FÁVERO, L.; BELFIORE, P. *Manual de análise de dados: Estatística e modelagem multivariada com excel®, spss® e stata®*. Elsevier Editora Ltda., 2017. Disponível em https://books.google.com.br/books?id=SmlaDwAAQBAJ

FIREEYE *Network Security and Forensics*. Disponível em: https://www.fireeye.com/solutions/nx-network-security-products.html, acesso em: 25/07/2019, 2019.

FORD, M.; MALLERY, C.; PALMASANI, F.; RABB, M.; TURNER, R.; SOLES, L.; SNIDER, D. A process to transfer fail2ban data to an adaptive enterprise intrusion detection and prevention system. In: *SoutheastCon 2016*, 2016, p. 1–4.

FORTINET Fortigate: Firewall de próxima geração (ngfw). Disponível em: https://www.fortinet.com/br/products/next-generation-firewall.html#models-specs, acesso em: 25/07/2019, 2019.

FREEBSD *ipfw*. Disponível em: https://www.freebsd.org/doc/handbook/firewalls-ipfw.html, acesso em: 28/07/2019, 2019.

GOJS *GoJs*. Disponível em: https://gojs.net, acesso em: 09/10/2020, 2020.

HONG, J. B.; YUSUF, S. E.; KIM, D. S.; KHAN, K. M. Stateless security risk assessment for dynamic networks. In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2018, p. 65–66.

HU, H.; AHN, G.; KULKARNI, K. Detecting and resolving firewall policy anomalies. *IEEE Transactions on Dependable and Secure Computing*, v. 9, n. 3, p. 318–331, 2012.

ISO Iso/iec 27000 family - information security management systems. Disponível em: https://www.iso.org/isoiec-27001-information-security.html, acesso em: 26/07/2019, 2018.

KAÇAR, M. S.; ÖZTOPRAK, K. Network security scoring. In: *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*, 2017, p. 477–481.

KIM, T.; KWON, T.; LEE, J.; SONG, J. F/wvis: Hierarchical visual approach for effective optimization of firewall policy. *IEEE Access*, v. 9, p. 105989–106004, 2021.

KUROSE, J. F.; ROSS, K. W. Redes de computadores, uma abordagem top-down. 2013.

LARA, A.; RAMAMURTHY, B. Opensec: Policy-based security using software-defined networking. *IEEE Transactions on Network and Service Management*, v. 13, n. 1, p. 30–42, 2016.

LIHUA YUAN; HAO CHEN; JIANNING MAI; CHEN-NEE CHUAH; ZHENDONG SU; MOHAPATRA, P. Fireman: a toolkit for firewall modeling and analysis. In: *2006 IEEE Symposium on Security and Privacy (S P'06)*, 2006, p. 15 pp.–213.

LIU, A. X.; TORNG, E.; MEINERS, C. R. Firewall compressor: An algorithm for minimizing firewall policies. In: *IEEE INFOCOM 2008 - The 27th Conference on Computer Communications*, 2008, p. 176–180.

MANSMANN, F.; GÖBEL, T.; CHESWICK, W. Visual analysis of complex firewall configurations. In: *Proceedings of the Ninth International Symposium on Visualization for Cyber Security*, VizSec '12, New York, NY, USA: Association for Computing Machinery, 2012, p. 1?8 (*VizSec '12*,).
Disponível em https://doi.org/10.1145/2379690.2379691

MAYER, A.; WOOL, A.; ZISKIND, E. Fang: a firewall analysis engine. In: *Proceeding 2000 IEEE Symposium on Security and Privacy. S P 2000*, 2000, p. 177–187.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, v. 38, n. 2, p. 69–74, 2008.
Disponível em http://doi.acm.org/10.1145/1355734.1355746

METASPLOIT Metasploit. Disponível em: https://www.metasploit.com/, acesso em: 25/07/2019, 2019.

MIT The mit license. Disponível em: https://opensource.org/licenses/MIT, acesso em: 03/02/2022, 2022.

MOSHARRAF, F.; FOROUZAN, B. A. *Redes e computadores: uma abordagem top-down*. AMGH/Porto Alegre, 2013.

NADEAU, T. D.; GRAY, K. *Sdn: Software defined networks: an authoritative review of network programmability technologies*. "O'Reilly Media, Inc.", 2013.

NAGIOS Nagios. Disponível em: https://www.nagios.org/, acesso em: 25/07/2019, 2019.

NASA *CYBERSECURITY MANAGEMENT AND OVERSIGHT AT THE JET PROPULSION LABORATORY*. Disponível em: https://oig.nasa.gov/docs/IG-19-022.pdf, acesso em: 26/07/2019, 2019.

NETFILTER *iptables*. Disponível em: https://netfilter.org, acesso em: 28/07/2019, 2019.

NMAP Nmap. Disponível em: https://nmap.org/, acesso em: 25/07/2019, 2019.

OSSEC Ossec. Disponível em: https://www.ossec.net/, acesso em: 25/07/2019, 2019.

PARRES-PEREDO, A.; PIZA-DAVILA, I.; CERVANTES, F. Mapreduce approach to build network user profiles with top-k rankings for network security. In: *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2018, p. 1–5.

QUEIROZ, L. M. M. Hardening em ambientes linux. 2014.

RUBIN, A. D.; GEER, D.; RANUM, M. J. *Web security sourcebook*. John Wiley & Sons, Inc., 1997.

SAPIA, H. M. Apoio à tradução da política de segurança para regras de *firewall* utilizando uma linguagem de modelagem visual: Spml2. 2016.

SHARMA, A.; SHARMA, D. Big data protection via neural and quantum cryptography. In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, p. 3701–3704.

SNORT Snort. Disponível em: https://snort.org/, acesso em: 25/07/2019, 2019.

SPIVEY, J. M.; ABRIAL, J. *The z notation*. Prentice Hall Hemel Hempstead, 1992.

SPSS, I. Ibm spss statistics. Disponível em: https://www.ibm.com/br-pt/products/spss-statistics, acesso em: 12/11/2021, 2021.

SYMANTEC *Symantec Web & Network Security*. Disponível em: https://www.symantec.com/products/web-and-network-security, acesso em: 25/07/2019, 2019.

TALUKDER, A. K.; MAURYA, V. K.; SANTHOSH, B. G.; JANGAM, E.; MUNI, S. V.; JEVITHA, K. P.; SAURABH, S.; PAIS, A. R. Security-aware software development life cycle (sasdlc) - processes and tools. In: *2009 IFIP International Conference on Wireless and Optical Communications Networks*, 2009, p. 1–5.

TANENBAUM, A.; WETHERALL, D.; TRANSLATIONS, O. *Redes de computadores*. PRENTICE HALL BRASIL, 2011.

TERPSTRA, J. H.; LOVE, P.; RECK, R. P.; SCANLON, T. *Hardening linux*. McGraw-Hill/Osborne, 2004.

TRAN, T.; AL-SHAER, E.; BOUTABA, R. Policyvis: Firewall security policy visualization and inspection. In: *21st Large Installation System Administration Conference (LISA 07)*, Dallas, TX: USENIX Association, 2007.
Disponível em https://www.usenix.org/conference/lisa-07/policyvis-firewall-security-policy-visualization-and-inspection

TRENDMICRO *Network Defense*. Disponível em: https://www.trendmicro.com/pt_br/business/products/network.html, acesso em: 25/07/2019, 2019.

TREVISANI, K. M.; GARCIA, R. E. Spml: A visual approach for modeling firewall configurations. In: *MODSEC@MoDELS*, 2008.

TURNBULL, J. *Hardening linux*. Apress, 2006.

UPGUARD *Losing Face: Two More Cases of Third-Party Facebook App Data Exposure*. Disponível em: https://www.upguard.com/breaches/facebook-user-data-leak, acesso em: 26/07/2019, 2019.

WIKILEAKS Wikileaks. Disponível em: https://wikileaks.org/, acesso em: 26/07/2019, 2019.

WILCOXON, F. *Individual comparisons by ranking methods*. The Bobbs-Merrill reprint series in the Social Sciences. Bobbs-Merrill Company Incorporated, 1945.
Disponível em https://books.google.com.br/books?id=jgzRuQEACAAJ

WIRESHARK Wireshark. Disponível em: https://www.wireshark.org/, acesso em: 25/07/2019, 2019.

WIRTH, N. Extended backus-naur form (ebnf). *Iso/Iec*, v. 14977, p. 2996, 1996.

WOHLIN, C. *Experimentation in software engineering : an introduction*. (The Kluwer international series in software engineerging ; 6). Boston: Kluwer Academic., 2000.

WOOL, A. A quantitative study of firewall configuration errors. *Computer*, v. 37, n. 6, p. 62–67, 2004.

WOOL, A. Trends in firewall configuration errors: Measuring the holes in swiss cheese. *IEEE Internet Computing*, v. 14, n. 4, p. 58–65, 2010.

YOON, M.; CHEN, S.; ZHANG, Z. Minimizing the maximum firewall rule set in a network with multiple firewalls. *IEEE Transactions on Computers*, v. 59, n. 2, p. 218–230, 2010.

APÊNDICE

A

Apêndice A

A.1 Política de segurança da rede da instituição A

Na política de acesso da instituição A, é ilustrado um cenário da rede de uma universidade. A política padrão do *firewall* é bloquear, ou seja, todo tráfego que não for explicitamente definido como permitido deverá ser bloqueado. A rede acadêmica (Acad) deve ser isolada da rede administrativa (Adm) para prevenir que estudantes tenham acesso a informações ou serviços de rede não autorizados. Os servidores da instituição são conectados à rede desmilitarizada (DMZ).

O *firewall* possui quatro interfaces de rede, uma para cada sub-rede cujo o tráfego necessita ser filtrado. As informações necessárias para configuração das interfaces são descritas na Tabela A.1.

Os prefixos e IPs das sub-redes e *hosts* para os quais são definidas políticas de filtragem são listados respectivamente nas Tabelas A.2 e A.3.

As restrições de acesso aos serviços são listadas a seguir.

- Serviço de DNS (porta 53, protocolo UDP) do servidor dns-externo localizado na Internet pode ser acessado por:
  - Todos os *hosts* da sub-rede acadêmica (requer *nat*)
  - Todos os *hosts* da sub-rede administrativa (requer *nat*)

Tabela A.1: Informações das interfaces de rede do *firewall*

Conecta-se a sub-rede	**Interface**	**IP/Máscara**
internet	em1	5.5.5.5/30
acadêmica	em2	10.0.0.1/16
administrativa	em3	192.168.0.1/24
desmilitarizada	em4	172.16.0.1/24

Tabela A.2: Informações das sub-redes da instituição

Sub-rede	**Prefixo/Máscara**
administrativa	192.168.0.0/24
acadêmica	10.0.0.0/16
desmilitarizada	172.16.0.0/16
internet	*

Tabela A.3: Informações dos hosts da instituição

Host	**IP/Máscara**
dns-externo	1.1.1.1/32
holerite	172.16.0.20/24
dhcp	172.16.0.40/24
site	172.16.0.80/24
responsável_rh	192.168.0.2/24

  - Todos os *hosts* da sub-rede desmilitarizada (requer *nat*)

- Serviços de navegação web (porta 80 e 443, protocolo TCP) localizados na Internet podem ser acessado por:
  - Todos os *hosts* da sub-rede acadêmica (requer *nat*)
  - Todos os *hosts* da sub-rede administrativa (requer *nat*)

- Serviço de holerite (porta 8080, protocolo TCP) do servidor holerite, localizado na rede desmilitarizada, pode ser acessado por:
  - O *host responsável_rh*, que está conectado na rede administrativa

- Serviço de dhcp (porta 67, protocolo UDP), localizado na rede desmilitarizada, pode ser acessado por:
  - Todos os *hosts* da sub-rede acadêmica
  - Todos os *hosts* da sub-rede administrativa

- Serviço HTTP no *host* site (porta 8080, protocolo TCP a partir de redirecionamento da porta 80), localizado na rede desmilitarizada, pode ser acessado por:
  - Todos os *hosts* da Internet (requer *rdr*)

A.2 Oráculo da Política de segurança da rede da instituição A

O oráculo com as regras escritas em linguagem de *firewall* do *Packet Filter* condizentes com as políticas de segurança definidas para a instituição A é apresentado na Definição A.1.

Definição A.1: Oráculo das regras da rede da instituição A

```
1  # Default Policy
2  block in all
3  pass in on em3 inet proto { udp } from 192.168.0.0/24 \
4  to 172.16.0.40/24 port 67
5  pass in on em3 inet proto { udp } from 192.168.0.0/24 \
6  to 1.1.1.1/32 port 53
7  pass in on em4 inet proto { udp } from 172.16.0.0/24 \
8  to 1.1.1.1/32 port 53
9  pass in on em2 inet proto { udp } from 10.0.0.0/16 \
10 to 172.16.0.40/24 port 67
11 pass in on em2 inet proto { udp } from 10.0.0.0/16 \
12 to 1.1.1.1/32 port 53
13 pass in on em2 inet proto { tcp } from 10.0.0.0/16 \
14 to any port 80
15 pass in on em2 inet proto { tcp } from 10.0.0.0/16 \
16 to any port 443
17 pass in on em3 inet proto { tcp } from 192.168.0.0/24 \
18 to any port 80
19 pass in on em3 inet proto { tcp } from 192.168.0.0/24 \
20 to any port 443
21 pass in on em3 inet proto { tcp } from 192.168.0.2/24 \
22 to 172.16.0.20/24 port 8080
23 pass in on em1 inet proto { tcp } from any to 5.5.5.5/30 \
24 port 80 rdr-to 172.16.0.80/24 port 8080
25 pass out on em4 inet proto { tcp } from 192.168.0.2/24 \
26 to 172.16.0.20/24 port 8080
27 pass out on em4 inet proto { udp } from 192.168.0.0/24 \
28 to 172.16.0.40/24 port 67
29 pass out on em4 inet proto { udp } from 10.0.0.0/16 \
30 to 172.16.0.40/24 port 67
31 pass out on em4 inet proto { tcp } from any \
32 to 172.16.0.80/24 port 8080
33 pass out on em1 inet proto { udp } from 192.168.0.0/24 \
34 to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30
35 pass out on em1 inet proto { udp } from 172.16.0.0/24 \
36 to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30
37 pass out on em1 inet proto { udp } from 10.0.0.0/16 \
38 to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30
39 pass out on em1 inet proto { tcp } from 10.0.0.0/16 \
40 to any port 80 nat-to 5.5.5.5/30
41 pass out on em1 inet proto { tcp } from 192.168.0.0/24 \
42 to any port 80 nat-to 5.5.5.5/30
43 pass out on em1 inet proto { tcp } from 10.0.0.0/16 \
44 to any port 443 nat-to 5.5.5.5/30
45 pass out on em1 inet proto { tcp } from 192.168.0.0/24 \
46 to any port 443 nat-to 5.5.5.5/30
```

A.3 Política de segurança da rede da instituição B

Na política de acesso da instituição B, é ilustrado um cenário da rede de uma empresa de TI. A política padrão do *firewall* é bloquear, ou seja, todo tráfego que não for explicitamente definido como permitido deverá ser bloqueado. A rede dos devops (devops) deve ser isolada da rede administrativa (Adm). Os servidores da instituição serão instalados na rede desmilitarizada (DMZ). Os ativos de rede (*switchs* e *access points*) serão instalados na rede Infraestrutura (Infra).

Tabela A.4: Informações das interfaces de rede do firewall

Conecta-se a sub-rede	**Interface**	**IP/Máscara**
internet	em1	5.5.5.5/30
devops	em2	10.0.0.1/16
administrativa	em3	192.168.1.1/24
desmilitarizada	em4	172.16.0.1/24
infraestrutura	em5	192.168.100.1/24

Tabela A.5: Informações das sub-redes da instituição

Sub-rede	**Prefixo/Máscara**
administrativa	192.168.1.0/24
devops	10.0.0.0/16
desmilitarizada	172.16.0.0/16
infraestrutura	192.168.100.0/24
internet	*

O *firewall* possui cinco interfaces de rede, uma para cada sub-rede cujo o tráfego necessita ser filtrado. As informações necessárias para configuração das interfaces são descritas na Tabela A.4.

Os prefixos e IPs das sub-redes, *hosts* e *hostlists* para os quais são definidas políticas de filtragem são listados respectivamente nas Tabelas A.5, A.6 e A.7.

As restrições de acesso aos serviços são listadas a seguir.

- Serviço de DNS (porta 53, protocolo UDP) do servidor dns-externo localizado na Internet pode ser acessado por:
  - Todos os *hosts* da sub-rede devops (requer *nat*)
  - Todos os *hosts* da sub-rede administrativa (requer *nat*)
  - Todos os *hosts* da sub-rede desmilitarizada (requer *nat*)
- Serviços de navegação web (porta 80 e 443, protocolo TCP) localizados na Internet podem ser acessado por:
  - Todos os *hosts* da sub-rede devops (requer *nat*)

Tabela A.6: Informações dos hosts da instituição

Host	**IP/Máscara**
dns-externo	1.1.1.1/32
ssh	172.16.0.30/24
sis-rh	172.16.0.40/24
zabbix	172.16.0.50/24
rh1	192.168.1.10/24
rh2	192.168.1.20/24
AWS	96.127.0.1/32

Tabela A.7: Informações do hostlist da instituição

Host List 1	IP/Máscara
rh1	192.168.1.10/24
rh2	192.168.1.20/24

  - Todos os *hosts* da sub-rede administrativa (requer *nat*)

- Serviço de SSH que opera no sevidor ssh (protocolo TCP, porta 2222), localizado na rede desmilitarizada, pode ser acessado por:
  - Todos os *hosts* da sub-rede devops
- Serviço de acesso ao sistema do RH que opera no servidor sis_rh (protocolo TCP, porta 8080), localizado na rede desmilitarizada, pode ser acessado por:
  - *Hosts* rh1 e rh2 (*hostlist*), pertencentes à rede administrativa.
- Serviço de acesso ao Zabbix que opera no servidor *zabbix* (protocolo UDP, porta 161), localizado na rede desmilitarizada, pode ser acessado por:
  - Todos os *hosts* da sub-rede infraestrutura
- Serviço de acesso à plataforma AWS (protocolo TCP), localizado na Internet, pode ser acessado por:
  - Todos os *hosts* da sub-rede devops (requer *nat*)

A.4 Oráculo da Política de segurança da rede da instituição B

O oráculo com as regras escritas em linguagem de *firewall* do *Packet Filter* condizentes com as políticas de segurança definidas para a instituição B é apresentado na Definição A.3.

Definição A.2: Oráculo das regras da rede da instituição B

```
1  # Default Policy
2  block in all
3  pass in on em4 inet proto { udp } from 172.16.0.0/24 \
4  to 1.1.1.1/32 port 53
5  pass in on em3 inet proto { udp } from 192.168.1.0/24 \
6  to 1.1.1.1/32 port 53
7  pass in on em2 inet proto { udp } from 10.0.0.0/16 \
8  to 1.1.1.1/32 port 53
9  pass in on em2 inet proto { tcp } from 10.0.0.0/16 \
10 to any port 80
11 pass in on em2 inet proto { tcp } from 10.0.0.0/16 \
12 to any port 443
13 pass in on em2 inet proto { tcp } from 10.0.0.0/16 \
14 to 172.16.0.30/24 port 2222
15 pass in on em2 inet proto { tcp } from 10.0.0.0/16 \
16 to 96.127.0.1/32
```

```
17          pass in on em3 inet proto { tcp } from 192.168.1.0/24 \
18          to any port 80
19          pass in on em3 inet proto { tcp } from 192.168.1.0/24 \
20          to any port 443
21          pass in on em3 inet proto { tcp } from { 192.168.1.10/24, 192.168.1.20
22          to 172.16.0.40/24 port 8080
23          pass in on em5 inet proto { udp } from 192.168.100.0/24 \
24          to 172.16.0.50/24 port 161
25          pass out on em1 inet proto { udp } from 172.16.0.0/24 \
26          to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30
27          pass out on em1 inet proto { udp } from 192.168.1.0/24 \
28          to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30
29          pass out on em1 inet proto { udp } from 10.0.0.0/16 \
30          to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30
31          pass out on em1 inet proto { tcp } from 10.0.0.0/16 \
32          to any port 80 nat-to 5.5.5.5/30
33          pass out on em1 inet proto { tcp } from 192.168.1.0/24 \
34          to any port 80 nat-to 5.5.5.5/30
35          pass out on em1 inet proto { tcp } from 10.0.0.0/16 \
36          to any port 443 nat-to 5.5.5.5/30
37          pass out on em1 inet proto { tcp } from 192.168.1.0/24 \
38          to any port 443 nat-to 5.5.5.5/30
39          pass out on em4 inet proto { tcp } from 10.0.0.0/16 \
40          to 172.16.0.30/24 port 2222
41          pass out on em4 inet proto { tcp } from { 192.168.1.10/24, 192.168.1.20
42          to 172.16.0.40/24 port 8080
43          pass out on em4 inet proto { udp } from 192.168.100.0/24 \
44          to 172.16.0.50/24 port 161
45          pass out on em1 inet proto { tcp } from 10.0.0.0/16 \
46          to 96.127.0.1/32 nat-to 5.5.5.5/30
```

A.5 Regras de *firewall* da rede da instituição C

Definição A.3: Oráculo das regras da rede da instituição B

```
1           # Default Policy
2           block in all
3           pass in on em2 inet proto { udp } from 10.0.0.0/16 /
4           to 1.1.1.1/32 port 53
5           pass in on em3 inet proto { udp } from 192.168.1.0/24 /
6           to 1.1.1.1/32 port 53
7           pass in on em4 inet proto { udp } from 172.16.0.0/24 /
8           to 1.1.1.1/32 port 53
9           pass out on em1 inet proto { udp } from 10.0.0.0/16 /
10          to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30
11          pass out on em1 inet proto { udp } from 192.168.1.0/24 /
12          to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30
13          pass out on em1 inet proto { udp } from 172.16.0.0/24 /
14          to 1.1.1.1/32 port 53 nat-to 5.5.5.5/30
15          pass in on em2 inet proto { tcp } from 10.0.0.0/16 /
16          to 172.16.0.30/24 port 3128
17          pass in on em3 inet proto { tcp } from 192.168.1.0/24 /
18          to 172.16.0.30/24 port 3128
19          pass out on em3 inet proto { tcp } from 10.0.0.0/16 /
20          to 172.16.0.30/24 port 3128
21          pass out on em3 inet proto { tcp } from 192.168.1.0/24 /
22          to 172.16.0.30/24 port 3128
23          pass in on em3 inet proto { tcp } from 172.16.0.30/24 /
24          to any port 80
```

```
25          pass in on em3 inet proto { tcp } from 172.16.0.30/24 /
26          to any port 443
27          pass in on em2 inet proto { tcp } from 10.0.0.2/16 /
28          to any port 80
29          pass in on em2 inet proto { tcp } from 10.0.0.2/16 /
30          to any port 443
31          pass out on em1 inet proto { tcp } from 172.16.0.30/24 /
32          to any port 80 nat-to 5.5.5.5/30
33          pass out on em1 inet proto { tcp } from 172.16.0.30/24 /
34          to any port 443 nat-to 5.5.5.5/30
35          pass out on em1 inet proto { tcp } from 10.0.0.2/16 /
36          to any port 80 nat-to 5.5.5.5/30
37          pass out on em1 inet proto { tcp } from 10.0.0.2/16 /
38          to any port 443 nat-to 5.5.5.5/30
39          pass in on em1 inet proto { tcp } from any to 5.5.5.5/30 /
40          port 443 rdr-to 172.16.0.80/24 port 4430
41          pass out on em4 inet proto { tcp } from any /
42          to 172.16.0.80/24 port 4430
43          pass in on em3 inet proto { tcp } from 192.168.1.0/24 /
44          to 172.16.0.50/24 port 80
45          pass out on em4 inet proto { tcp } from 192.168.1.0/24 /
46          to 172.16.0.50/24 port 80
47          pass in on em2 inet proto { tcp } from 10.0.0.0/16 /
48          to 172.16.0.60/24 port 80
49          pass out on em4 inet proto { tcp } from 10.0.0.0/16 /
50          to 172.16.0.60/24 port 80
```

A.6 Diagrama SPML2 da rede da instituição C

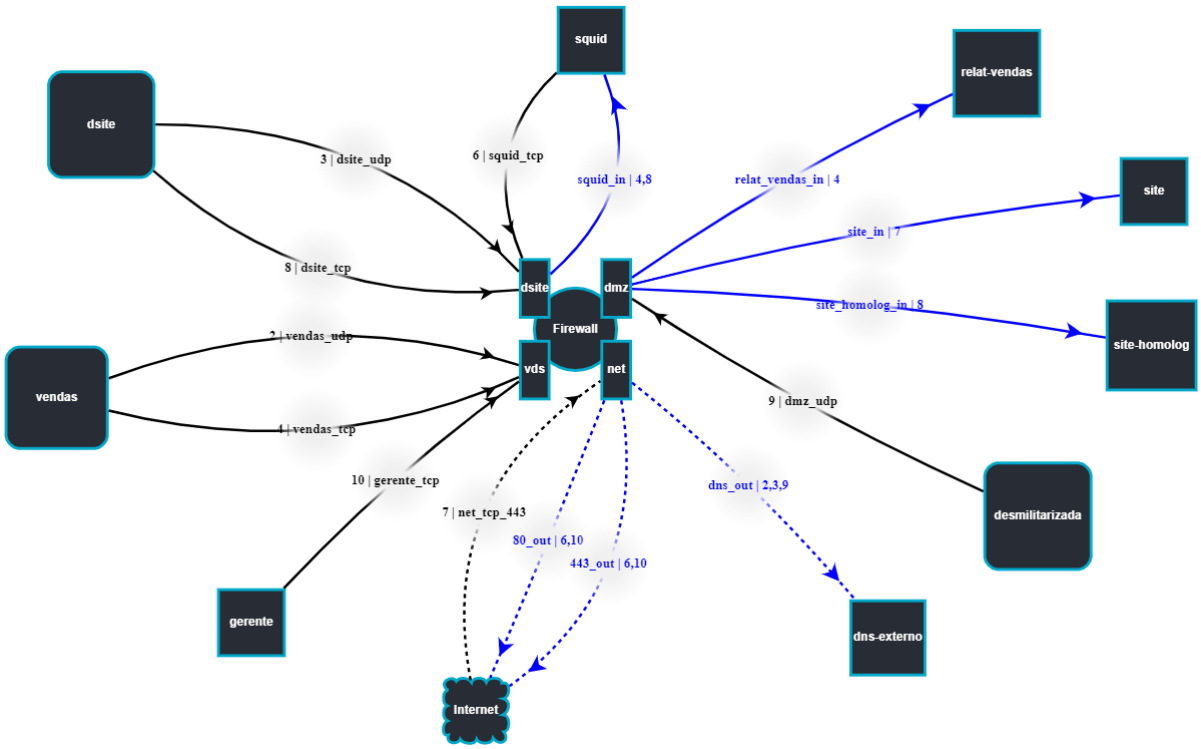


Figura A.1: Diagrama SPML2 da rede da instituição C

Tabela A.8: Informações das interfaces de rede do firewall

Conecta-se a sub-rede	Interface	IP/Máscara
internet	em1	5.5.5.5/30
vendas	em2	10.0.0.1/16
dsite	em3	192.168.1.1/24
desmilitarizada	em4	172.16.0.1/24

Tabela A.9: Informações das sub-redes da instituição

Sub-rede	Prefixo/Máscara
dsite	192.168.1.0/24
vendas	10.0.0.0/16
desmilitarizada	172.16.0.0/16
internet	*

A.7 Oráculo da política de segurança da rede da instituição C

Na política de acesso da instituição C, é ilustrado um cenário da rede de uma empresa de comércio online. A política padrão do *firewall* é bloquear, ou seja, todo tráfego que não for explicitamente definido como permitido deverá ser bloqueado. A rede Vendas (vendas) deve ser isolada da rede Desenvolvimento site (DSite). Os servidores da instituição serão instalados na rede desmilitarizada (DMZ).

O *firewall* possui quatro interfaces de rede, uma para cada sub-rede cujo o tráfego necessita ser filtrado. As informações necessárias para configuração das interfaces são descritas na Tabela A.8.

Os prefixos e IPs das sub-redes e *hosts* para os quais são definidas políticas de filtragem são listados respectivamente nas Tabelas A.5 e A.6.

As restrições de acesso aos serviços são listadas a seguir.

- Serviço de DNS (porta 53, protocolo UDP) do servidor dns-externo localizado na Internet pode ser acessado por:
  - Todos os *hosts* da sub-rede vendas (requer *nat*)
  - Todos os *hosts* da sub-rede dsite (requer *nat*)

Tabela A.10: Informações dos hosts da instituição

Host	IP/Máscara
dns-externo	1.1.1.1/32
squid	172.16.0.30/24
site	172.16.0.80/24
site-homologacao	172.16.0.50/24
relatorio-vendas	172.16.0.60/24
gerente	10.0.0.2/16

  - Todos os *hosts* da sub-rede desmilitarizada (requer *nat*)

- Serviço *proxy-squid* que opera no servidor squid, localizado na sub-rede desmilitarizada, (porta 3128, protocolo TCP) pode ser acessado por:
  - Todos os *hosts* da sub-rede vendas
  - Todos os *hosts* da sub-rede dsite

- Serviços de navegação web (porta 80 e 443, protocolo TCP) localizados na Internet podem ser acessado por:
  - O *host* squid (servidor proxy-squid) (requer nat)
  - O *host* gerente, que se conecta à rede de vendas (requer *nat*)

- Serviço HTTPS no *host site* (porta 4430, protocolo TCP a partir de redirecionamento da porta 443) pode ser acessado por:
  - Todos os *hosts* da Internet (requer rdr)

- Serviço de homologação do site, que opera no servidor *site-homologacao* (porta 80, protocolo TCP), pode ser acessado por:
  - Todos os *hosts* da sub-rede dsite

- Serviço de relatório de vendas, que opera no servidor relatórios-vendas (porta 80, protocolo TCP), pode ser acessado por:
  - Todos os *hosts* da sub-rede vendas