FWunify: uma Ferramenta para Simplificar a Configuração de Múltiplos Firewalls

Maurício Fiorenza[1], **Diego Kreutz**[1], **Rodrigo B. Mansilha**[1]

[1]Universidade Federal do Pampa (Unipampa)

{mauriciofiorenza,diegokreutz,rodrigomansilha}@unipampa.edu.br

Resumo. *A configuração de múltiplos firewalls é um processo desafiador. As soluções existentes são especializadas e requerem o domínio prévio ou aprendizado de uma variedade de sintaxes e métodos de configuração para implementar corretamente as políticas de segurança desejadas. Para reduzir a curva de aprendizagem e mitigar erros operacionais, propomos a ferramenta FWunify – uma solução para configuração integrada e automática de firewalls. Através de uma arquitetura composta por múltiplas camadas e módulos fracamente acoplados, a FWunify permite que novas soluções de firewall sejam incorporadas a ferramenta impactando minimamente nas camadas adjacentes. Um protótipo funcional da FWunify foi implementado e utilizado para demonstrar a viabilidade técnica e aplicabilidade da proposta.*

1. Introdução

Diferentes fatores, como custos de aquisição ou a evolução dos equipamentos, podem provocar a instalação de firewalls de linhas ou fabricantes diversos em uma rede. Intuitivamente, quanto mais heterogêneo for o conjunto de soluções de firewalls utilizado em uma rede, maior tende a ser o seu custo operacional. Os administradores precisam conhecer e traduzir regras para sintaxes específicas das soluções de firewall instaladas [Fiorenza et al., 2021]. Consequentemente, a especificação e aplicação das regras em diferentes sintaxes potencializa erros de configuração que podem impactar a aplicação e a concretização da política de segurança na instituição [Voronkov et al., 2017]. De fato, estudos e previsões recentes apontam que 99% das violações em firewalls serão causadas por configurações incorretas e não por falha das soluções até 2023 [Gartner, 2019].

Equipamentos de firewalls podem variar em termos de fabricante, linha e modelo, e essas variações tendem a refletir nas linguagens ou versões de linguagens de configuração. Normalmente, as linguagens de configuração variam significativamente entre fabricantes [Fiorenza et al., 2021]. Além disso, frequentemente, as linguagens de configuração podem variar para um mesmo fabricante, tanto entre modelos (*e.g.*, Cisco PIX vs ASA) quanto entre versões de um mesmo modelo (*e.g.*, Cisco ASA 5500 versão 8.2 vs 8.4) de firewall. Na prática, instituições como a Unipampa e provedores de Internet regionais sofrem com os problemas e custos operacionais resultantes da configuração de diferentes tipos e versões de firewalls. Apesar de existirem soluções como a IFCL [Bodei et al., 2018], que viabilizam a migração de regras de um tipo de firewall para outro, elas não oferecem recursos para operacionalizar o gerenciamento simultâneo de diferentes soluções.

Diante do problema exposto, temos empregado esforços de pesquisa, desenvolvimento e inovação para integrar e automatizar o gerenciamento de diferentes soluções

de firewall, sejam elas tradicionais (*e.g.*, Cisco NGFW, Palo Alto, pfSense, IPTables) ou SDNs (*e.g.*, baseadas em OpenFlow ou P4). Inicialmente, propomos uma arquitetura em camadas para o gerenciamento de firewalls em redes híbridas [Fiorenza et al., 2020]. Recentemente, propomos uma linguagem estruturada de definição de intenções, denominada FWlang [Fiorenza et al., 2021], desenvolvida e validada para a representação de políticas de segurança utilizadas nos principais tipos de firewall modernos, ou de próxima geração.

Baseados nos estudos prévios, concebemos e implementamos a FWunify: uma ferramenta para integrar e automatizar o gerenciamento de múltiplas soluções de firewall em redes corporativas. A FWunify incorpora a linguagem FWlang para *simplificar*, seguindo o conceito de abstração em nível de intenção, e *unificar* a descrição de políticas de segurança. A FWunify traduz e implanta *automaticamente* políticas de segurança nas diferentes soluções de firewall da rede. A FWunify é *flexível* pois dispõe de uma arquitetura em camadas, modular e extensível. A organização em camadas favorece a separação de conceitos (*e.g.*, aplicações de gestão e serviços de tradução) e a estruturação da solução. Dentro de cada camada há módulos independentes entre si. Essa modularização permite alteração (adição, remoção ou manutenção) de recursos de maneira simples e rápida – sem impactar nos outros módulos e refletindo minimamente nas camadas adjacentes apenas. Por exemplo, módulos de tradução de regras podem ser facilmente acrescentados na solução para atender o gerenciamento de novos tipos ou famílias de firewall.

As contribuições técnicas deste trabalho são: (a) especificação de uma arquitetura (detalhada) em camadas para permitir a configuração integrada de firewalls em redes corporativas; (b) implementação de uma instância da arquitetura, denominada FWunify, organizada em camadas, modular e extensível; e (c) avaliação empírica da FWunify realizada com alunos dos cursos de computação da Unipampa e demonstração da eficácia das regras geradas em um ambiente real.

Nas seções 2 e 3 apresentamos, respectivamente, a arquitetura e a implementação da FWunify. Na Seção 4 apresentamos alguns resultados de avaliações realizadas com a ferramenta. Por fim, detalhamos a demonstração planejada para o Salão de Ferramentas e as considerações finais do trabalho na Seção 5.

2. Arquitetura

A arquitetura FWunify é organizada em sete camadas, como detalhado na Figura 1. As colunas da esquerda e da direita contém, respectivamente, as camadas e exemplos de módulos para tecnologias viabilizadoras da respectiva camada. A arquitetura em camadas permite que módulos sejam adicionados ou removidos conforme a necessidade da rede gerenciada, impactando apenas e minimamente nas camadas adjacentes. Por exemplo, um novo módulo que gere políticas para um determinado tipo de firewall pode ser adicionado à camada de microsserviços de tradução. Para isso, as camadas superiores (*e.g*, uma API (*Application Programming Interface*) REST (*Representational State Transfer*)) precisam ser informadas do novo módulo, para que elas possam enviar as intenções para a tradução. Dependendo do método utilizado para aplicar as políticas no firewall, esse novo microsserviço de tradução poderá: (a) utilizar um dos módulos disponíveis na interface sul, ou (b) demandar a criação ou instalação de um módulo que suporte a forma de acesso do novo firewall. A seguir, cada camada da arquitetura é detalhada em ordem de cima para baixo.

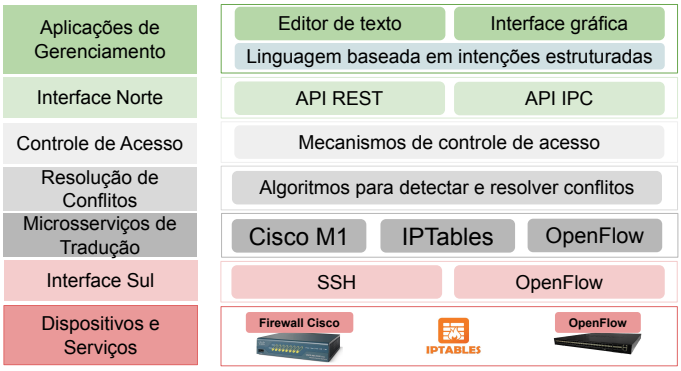


Figura 1. Arquitetura proposta detalhada

Na camada superior, *Aplicações de Gerenciamento*, os módulos representam as interfaces (*i.e.*, pontos de interação com o usuário final) utilizadas para a definição, controle e ativação das políticas de segurança na rede. As interfaces podem ser implementadas utilizando diferentes recursos e tecnologias. A alternativa mais simples pode ser considerada um *editor de texto*, utilizado para a edição das intenções de segurança descritas em uma linguagem baseada em intenções estruturadas. A partir de uma ferramenta simples, como um comando `curl`[1], os operadores da rede podem enviar a intenção para tradução através de uma API REST, por exemplo.

O tipo de interface considerado ideal varia conforme o usuário e as suas atividades realizadas. Por exemplo, administradores rotineiros, como operadores de rede, podem preferir interfaces de linha de comando considerando aspectos como agilidade e facilidade de automação [Botta et al., 2007, Voronkov et al., 2019]. Por outro lado, administradores eventuais (ou menos experientes) podem preferir interfaces gráficas para gerenciar a criação, organização e visualização das intenções de segurança [Voronkov et al., 2019]. De todo modo, para simplificar, padronizar e agilizar o desenvolvimento de aplicações de gerenciamento, é necessário que as interfaces utilizem uma mesma *linguagem universal*.

A camada *Interface Norte* é responsável pela comunicação entre as aplicações de gerenciamento e a camada de tradutores. A Interface Norte pode ser implementada como uma API do tipo REST, que é utilizada para receber e enviar a descrição das políticas de segurança entre as camadas superior e inferior da arquitetura.

A camada *Controle de Acesso* é responsável pela autenticação, autorização e responsabilização (*accountability*) dos usuários. Para isso, a FWunify permite personalizar níveis de privilégios, escopos de ação e granularidades. Os níveis de privilégios possibilitam a criação de perfis de operadores com diferentes permissões. Os níveis de escopos de ação e granularidade no contexto de um escopo específico permitem, por exemplo, adicionar intenções ACL apenas em uma sub-rede do domínio.

O objetivo da camada de *Resolução de Conflitos* é realizar a verificação das intenções de segurança definidas pelo usuário. Existem duas principais abordagens para detecção e resolução de conflitos em firewalls: a verificação das novas políticas adicionadas ao firewall para evitar colisões, redundâncias e problemas de sobreposição em regras

[1] `https://curl.se`

(*e.g.*, PreFirewall [Morzhov et al., 2016]); e o monitoramento de fluxos e caminhos de dados na rede em busca de tráfegos que indiquem violações nas políticas de segurança em uso (*e.g.*, FlowMon [Hu et al., 2019]). Soluções utilizadas na camada de resolução de conflitos podem, por exemplo, evitar a inserção de políticas conflitantes que liberem um tráfego que deveria ser bloqueado segundo a política de segurança de uma instituição.

Os *Microsserviços de Tradução* recebem as intenções de segurança através da interface norte e as traduzem para comandos específicos a cada tipo de solução de firewall. Para facilitar a tradução, os microsserviços são construídos utilizando uma linguagem de tradução baseadas em *templates*, que permite a rápida composição das políticas de segurança na sintaxe específica de cada firewall. Na prática, o microsserviço Cisco M1 irá traduzir as intenções de segurança para comandos específico de um firewall ASA 5500-X, cuja versão do *firmware* é 8.2 ou 8.4, por exemplo. Supondo que uma rede de grande porte possui sete soluções (e/ou versões) diferentes de firewall, serão necessários até sete microsserviços de tradução, um para cada linha (ou versão) distinta de firewalls. Os microsserviços desta camada podem ser desenvolvidos e disponibilizados tanto pelos fabricantes quanto por empresas especializadas ou pelos próprios usuários das soluções de firewall.

A camada *Interface Sul* é responsável pela ligação entre as camadas de tradução e de dispositivos e serviços. As regras geradas pelos microsserviços de tradução são enviadas para as diferentes soluções de firewall utilizando os respectivos protocolos. Por exemplo, um firewall Linux (IPTables) ou Cisco (ASA 5500-X) pode ser configurado utilizando o protocolo SSH (*Secure Socket Shell*). Já um firewall SDN pode ser configurado utilizando o próprio padrão do OpenFlow (*e.g.*, conexão TLS (*Transport Layer Security*)), por exemplo.

Na camada *Dispositivos e Serviços* encontram-se as soluções de firewall utilizadas na rede. Para cada nova solução adicionada na rede, será necessário associar um microsserviço de tradução e um protocolo da interface sul que possibilitem a tradução e aplicação das políticas.

3. Implementação

Uma implementação funcional da ferramenta FWunify está disponível em `https://github.com/mmfiorenza/fwunify`. A solução foi desenvolvida utilizando a linguagem de programação Python versão 3.7. Além de instanciar a arquitetura de software FWunify, a implementação atende também o conceito de intenções para o gerenciamento de redes, isto é, segue os princípios e o fluxo essencial de (a) submissão e (b) tradução das intenções e (c) aplicação das configurações (regras geradas) nos dispositivos da rede, conforme especificado no padrão em proposição no IETF (*Internet Engineering Task Force*) [Sun et al., 2019].

A versão atual da FWunify contempla uma API REST (interface norte), três microsserviços de tradução para firewalls Cisco ASA 5505, GNU/Linux IPTables e Open vSwitch, além de dois conectores SSH (interface sul), um específico para Cisco e outro para equipamentos Linux. Na camada de dispositivos e serviços foram utilizados firewalls Cisco ASA 5520, Linux IPTables e OpenFlow, este último através do Open vSwitch. A Figura 2 resume as tecnologias utilizadas para viabilizar cada camada da arquitetura no FWunify.

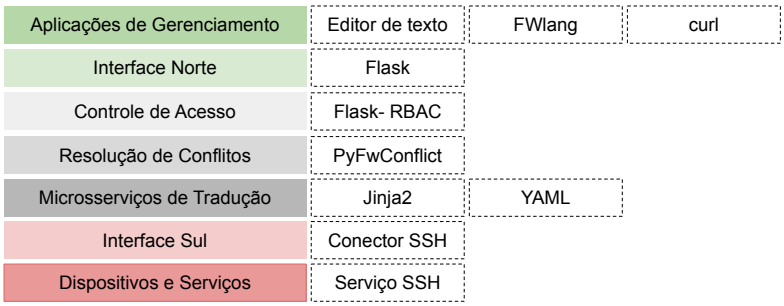


Figura 2. Tecnologias viabilizadoras de cada camada

Na camada aplicações de gerenciamento é utilizado um editor de texto para definir as intenções em arquivos, seguindo a sintaxe da FWlang. Os arquivos das intenções são enviados para tradução através da interface norte utilizando o comando *curl* via interface de linha de comando (CLI). Além dos arquivos das intenções, o administrador do sistema precisa informar também as suas credenciais de acesso para autenticação, autorização e registro de atividades. É importante ressaltar que CLI é um dos métodos mais utilizados por administradores de sistemas [Voronkov et al., 2019].

A interface norte implementa uma API e um serviço REST utilizando framework Flask[2]. As intenções são enviadas através do método POST e do endereço "/" da API. Para enviar uma intenção são necessárias: (a) as informações de autenticação do operador da rede (usuário e senha); e (b) a intenção descrita na sintaxe da FWlang, que deve ser enviada em formato binário (*e.g*, no caso do *curl* é utilizado o parâmetro "*–data-binary*"). O serviço REST recebe e processa a intenção. Na sequência, ele envia os dados da intenção para os microsserviços. Finalizada a tradução e aplicação da intenção, os microsserviços retornam uma resposta ao serviço REST, indicando o sucesso (código de status 200) ou a falha no processo. O serviço REST é responsável pela verificação e validação da sintaxe e dos valores atribuídos aos marcadores da FWlang. Como exemplos de valores de marcadores podemos citar endereços IP, nomes de *hosts*, definição de sub-redes (*e.g*, 10.0.0.0/23) no padrão CIDR (*Classless Inter-Domain Routing*) e definição de largura de banda em "Mbps" para políticas de *traffic shaping* ou filtros de URL. Esse serviço REST também extrai os dados das intenções (*e.g.*, origem, destino, porta, protocolo e ações) necessários para compor a sintaxe nos tradutores especializados. Essas informações são adicionadas em um dicionário de dados Python, que é enviado aos microsserviços de tradução.

A camada controle de acesso é baseada em papéis para administradores com diferentes níveis de acesso aos recursos da FWunify. Essa camada foi implementada utilizando o módulo de controle de acesso do Flask, chamado Flask-RBAC[3] (*Role-based Access Control*), que permite a definição de diferentes funções dentro da aplicação. Para fins de testes, foram definidos três níveis de acesso. Administradores do primeiro nível conseguem realizar somente a tradução e aplicação de políticas de *traffic shaping*. Os administradores do nível intermediário podem traduzir e aplicar políticas de *traffic shaping* e NAT 1to1. Por fim, os administradores do terceiro nível podem realizar qualquer

[2]`https://flask.palletsprojects.com/en/1.1.x//`
[3]`https://flask-rbac.readthedocs.io/en/latest/`

operação suportada pela ferramenta, como aplicar políticas de roteamento estático, NAT Nto1 e ACLs.

Uma vez liberadas pela camada de controle de acesso, as políticas de segurança seguem para a camada de resolução de conflitos. Essa camada realiza verificações de duplicidade e sobreposição de políticas com base em um histórico de políticas. Isso evita que duas políticas diferentes tratem um mesmo tipo de tráfego. Por exemplo, adicionar uma regra que libere o tráfego FTP para o IP 10.0.0.5, sendo que existe outra regra que libera o tráfego para o bloco de IPs 10.0.0.0/24. Caso as novas políticas dupliquem ou sobreponham políticas existentes, o processo de tradução é interrompido e uma notificação é enviada ao administrador do sistema.

Os três tradutores, para Cisco ASA, IPTables e OpenFlow (Open vSwitch), representam a camada microsserviços de tradução. A comunicação das camadas superiores com os microsserviços é realizada através de chamadas remotas de procedimento. Para realizar as chamadas remotas, a API da camada interface norte deve conhecer os microsserviços de tradução disponíveis e habilitados no sistema, bem como as funções disponíveis em cada microsserviço. As chamadas remotas são definidas no padrão URI_AMPQ[4], incluindo detalhes de autenticação, IP e porta do servidor que disponibiliza o microsserviço. Para o processo de tradução, cada microsserviço deve receber um dicionário de dados com as informações extraídas da intenção. Por exemplo, para uma política do tipo ACL são obrigatórias informações como origem e destino dos pacotes, ação a ser tomada, e a ordem de prioridade da regra no respectivo firewall.

O tradutor então recebe o dicionário de dados e realiza a tradução para a sintaxe de comandos específicos do firewall. O processo de tradução pode incluir, inclusive, a conversão de unidades, como a conversão da largura de banda de Mbps para Kbps. O processo de tradução é realizado com o suporte da linguagem de modelagem de *templates* Jinja2[5]. Essa linguagem permite criar *templates* contendo partes fixas e dinâmicas das sintaxes dos comandos para os diferentes firewalls. As partes dinâmicas (*e.g.*, IP, largura de banda) são preenchidas automaticamente pela Jinja2 a partir dos dados recebidos da camada subjacente, gerando os comandos a serem aplicados nos dispositivos de firewall.

Por fim, os microsserviços de tradução utilizam a interface sul, como os conectores SSH, para realizar a aplicação das regras nas respectivas soluções de firewall. Os tradutores realizam a chamada remota de procedimento para o conector definido na implementação, utilizando o padrão URI_AMPQ, assim como utilizado na camada anterior. Os conectores recebem como parâmetro o endereço IP de gerência para acesso ao dispositivo, credenciais de acesso (*i.e.*, usuário e senha) e a sequência dos comandos a serem aplicados ao firewall. Na implementação atual, há dois conectores SSH, um específico para equipamentos Cisco da série ASA 5020 e outro para os sistemas Linux que implementam os firewalls IPTables e OpenFlow. Os conectores SSH foram implementados através da biblioteca Netmiko[6], que utiliza parâmetros como endereço IP, porta e usuário para estabelecer as conexões SSH. Os conectores retornam as informações de estado (*e.g.*, sucesso ou falha) da aplicação da política na solução de firewalls.

[4] `https://www.rabbitmq.com/uri-spec.html`
[5] `https://jinja.palletsprojects.com/en/2.11.x/`
[6] `https://github.com/ktbyers/netmiko`

4. Avaliação

Avaliação com usuários

A ferramenta foi submetida a uma atividade prática de instalação, configuração, utilização e avaliação com alunos de três disciplinas dos cursos de Engenharia de Software, Ciência da Computação e Mestrado Profissional em Engenharia de Software da Unipampa. Na atividade, os alunos foram instruídos a instalar e configurar a ferramenta e compor e aplicar políticas de segurança em um firewall IPTables.

Ao final da atividade, os alunos elaboraram relatórios técnicos de avaliação e sugestões de melhoria da FWunify. Após compilados e analisados, os *feedbacks* dos alunos levaram a um ciclo adicional de atualização e evolução do protótipo, melhorando aspectos de usabilidade, funcionalidade e implementação da ferramenta. Como dois exemplos pontuais de melhoria, podemos citar a inclusão do suporte a nomes de domínio para identificar os pontos de origem e destino (*i.e.*, resolução de nomes via DNS para identificar os IPs) e a utilização do arquivo padrão `/etc/services` do GNU/Linux para identificar os protocolos e portas dos serviços indicados nas intenções de segurança.

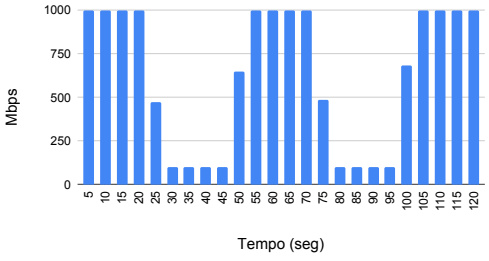
Eficácia das políticas traduzidas e aplicadas

Para realizar a avaliação da eficácia da tradução e aplicação das intenções de segurança utilizando a FWlang e a FWunify, foi especificada e aplicada uma intenção de *traffic shaping* apresentada na Figura 3(a). Essa intenção limita o tráfego da porta 5555/UDP a 100Mbps (`throughput(100Mbps)`) entre a rede 10.0.0.0/24 e o servidor 200.19.0.100, cujo link é de 1Gbps. A conexão entre o *host* na rede interna e o servidor é estabelecida através de um switch virtual Open vSwitch, com OpenFlow na versão 1.3. Neste cenário, o Open vSwitch atua também como firewall SDN da rede.

A medição da vazão entre o *host* e o servidor foi realizada com o auxílio da ferramenta *iperf*[7]. A Figura 3(b) apresenta o comportamento da vazão da conexão antes, durante e depois de duas aplicações e remoções consecutivas da intenção de *traffic shaping*. Como pode ser observado, a vazão reduz para 100Mbps quando a intenção é aplicada ao firewall e retorna a 1Gbps após a remoção da intenção.

```
define intent traffic_shaping:
name        text('limit-net-h100-100m')
from        range('10.0.0.0/24')
to          endpoint('200.19.0.100')
order       before('all')
for         traffic('udp/5555')
with        throughput('100Mbps')
add/del     firewall('openflow-1')
```

(a) Intenção para *Traffic Shaping*



(b) Recorte da vazão de tráfego

Figura 3. Aplicação e remoção de *Traffic Shaping*

Outros resultados e discussões acerda da corretude do processo de tradução e da eficácia de políticas geradas, para firewalls Cisco, IPTables e OpenFlow, estão disponíveis em [Fiorenza et al., 2021].

[7] `https://iperf.fr/`

5. Considerações Finais

Demonstração

A demonstração da ferramenta será realizada através de um ambiente virtual de redes controladas por firewalls IPTables e OpenFlow. O ambiente estará hospedado em um dispositivo próprio dos autores. As funcionalidades da ferramenta serão apresentadas através dos seguintes passos: (a) apresentação de duas políticas de segurança, dos tipos ACL e NAT 1to1, descritas utilizando a linguagem FWlang; (b) demonstração do processo de tradução e aplicação das duas políticas utilizando o FWunify; (c) apresentação dos comandos gerados automaticamente pela ferramenta para cada um dos tipos de firewall; e (d) demonstração do funcionamento das regras aplicadas nos firewalls.

Conclusão

Nós apresentamos a ferramenta FWunify, concebida a partir de uma arquitetura multicamada e modular que permite a adição, sob-demanda, de novos componentes (*e.g.*, microsserviços de tradução de regras) para a configuração unificada e automatizada de soluções de firewall atuais e futuros em redes corporativas.

A versão atual da ferramenta permite configurar, de maneira simples, automática e integrada, firewalls Cisco ASA-5520, Linux/IPTables, e OpenFlow/Open vSwitch. Os resultados das avaliações demonstram a aplicabilidade e o funcionamento da ferramenta para aplicação de políticas de segurança em redes corporativas. Informações sobre a inclusão de novos tradutores, para diferentes tipos e versões de firewalls, podem ser encontradas na documentação da ferramenta.

Referências

Bodei, C., Degano, P., Focardi, R., Galletta, L., and Tempesta, M. (2018). Transcompiling firewalls. In *POST*, pages 303–324.

Botta, D., Werlinger, R., Gagné, A., Beznosov, K., Iverson, L., Fels, S., and Fisher, B. (2007). Towards understanding it security professionals and their tools. In *USENIX SOUPS*, page 100–111. ACM.

Fiorenza, M., Kreutz, D., Mansilha, R., Macedo, D., Feitosa, E., and Immich, R. (2021). Representação e aplicação de políticas de segurança em firewall de redes híbridas. In *XXXIX SBRC*. SBC.

Fiorenza, M. M., Kreutz, D., and Mansilha, R. (2020). Gerenciamento de firewalls em redes híbridas. In *XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg)*. SBC.

Gartner (2019). Technology Insight for Network Security Policy Management. `https://www.gartner.com/en/documents/3902564`.

Hu, H., Han, W., Kyung, S., Wang, J., Ahn, G.-J., Zhao, Z., and Li, H. (2019). Towards a reliable firewall for software-defined networks. *Computers & Security*, 87:101597.

Morzhov, S., Alekseev, I., and Nikitinskiy, M. (2016). Firewall application for Floodlight SDN controller. In *International Siberian Conference on Control and Communications*, pages 1–5. IEEE.

Sun, Q., LIU, W. S., and Xie, K. (2019). An Intent-driven Management Framework. Internet-draft, Internet Engineering Task Force. Work in Progress.

Voronkov, A., Iwaya, L. H., Martucci, L. A., and Lindskog, S. (2017). Systematic literature review on usability of firewall configuration. *ACM Comput. Surv.*, 50(6).

Voronkov, A., Martucci, L. A., and Lindskog, S. (2019). System administrators prefer command line interfaces, don't they? an exploratory study of firewall interfaces. In *USENIX SOUPS*. USENIX Association.