Gerenciamento de Firewalls em Redes Híbridas

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

[1]Laboratório de Estudos Avançados (LEA)
Programa de Pós-Graduação em Engenharia de Software (PPGES)
Universidade Federal do Pampa (UNIPAMPA)

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

Resumo. *O gerenciamento de firewalls é um processo desafiador em redes híbridas, pois envolve aplicar políticas de segurança em soluções tradicionais (e.g., Cisco NGFW, IPTables) e emergentes (e.g., OpenFlow, P4, e POF). Neste trabalho é proposta e discutida uma arquitetura, organizada em uma pilha de camadas, para gerência integrada de firewalls em redes híbridas. Como prova de conceito, foi implementado um protótipo que gera a configuração de regras para diferentes soluções de firewall. O protótipo foi avaliado experimentalmente através de testes de continuidade de tráfego (bloqueio e liberação) e limite de tráfego (traffic shaping).*

1. Introdução

As redes híbridas (*i.e.*, redes com equipamentos tradicionais e SDN) representam o cenário de adoção progressiva de equipamentos baseados nos padrões e protocolos SDN como OpenFlow, P4, ForCES e POF [McKeown et al., 2008, Bosshart et al., 2014, Yang et al., 2004, Song, 2013, Kreutz et al., 2014]. As redes híbridas são atrativas devido a fatores como investimento de capital (CAPEX) e formação de recursos humanos/operação (OPEX) gradual [Sinha et al., 2017, Abdallah et al., 2017]. Entretanto, elas suscitam desafios, como o gerenciamento integrado de soluções de redes tradicionais e redes SDN.

Os desafios de integração e interoperabilidade de soluções também ocorrem no âmbito de firewalls. Uma rede híbrida pode incorporar diversas soluções de firewall, tanto tradicionais (*e.g.*, IPTables, pfSense, Cisco NGFW), como SDN (*i.e.*, baseadas em OpenFlow, P4, etc.). As soluções de firewall existentes, como FireFlow [Fiessler et al., 2018], Fortress [Caprolu et al., 2019], Gherkin NBI [Esposito et al., 2018] e Cisco NGFW[1], focam no gerenciamento de firewalls tradicionais ou SDN específicos (*e.g.*, de uma única família de equipamentos, como é o caso das soluções da Cisco). No contexto de SDN, as soluções de firewall utilizam essencialmente os recursos providos por padrões ou protocolos como o OpenFlow e P4 [Fiessler et al., 2018, Caprolu et al., 2019, Hu et al., 2014, Zerkane et al., 2016, Morzhov et al., 2016, Othman et al., 2017, Vörös and Kiss, 2016, Datta et al., 2018, Zkik et al., 2019, Esposito et al., 2018]. Até onde se sabe, até então não existe uma solução para unificar o gerenciamento de soluções heterogêneas de firewalls em redes híbridas.

Como um primeiro passo para preencher essa lacuna, este trabalho propõe uma arquitetura (Seção 2) em camadas e modular capaz de unificar o gerenciamento das mais diversas soluções de firewall, tradicionais ou SDN, existentes. A partir de uma interface comum, o administrador da rede deve ser capaz de aplicar e remover políticas de segurança

[1]https://www.cisco.com/

de todas as diferentes soluções de firewall da rede. Como forma de validar a arquitetura proposta, foi implementado um protótipo (Seção 3) que permite traduzir políticas de redes declaradas por meio de *Intents* (ou intenções) [Behringer et al., 2015] para comandos de configuração de diferentes soluções de firewall. Os resultados de uma avaliação experimental preliminar mostram o funcionamento e a viabilidade técnica/operacional da solução proposta.

2. Arquitetura Proposta

O objetivo principal da arquitetura proposta é balizar o desenvolvimento de soluções de gerenciamento integrado de firewalls em redes híbridas. Para isso, a arquitetura precisa ser flexível e modularizada a ponto de suportar os mais diferentes tipos de soluções de firewall, baseadas em tecnologias e protocolos como ASA 5500-X (Cisco), IPTables, pfSense, OpenFlow e P4.

A proposta é uma arquitetura de cinco camadas, como mostra a Figura 1. As colunas da esquerda e da direita contém, respectivamente, as camadas e exemplos de tecnologias viabilizadoras da respectiva camada. Claramente, 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, módulos podem ser acrescentados motivados pela aquisição de tecnologias emergentes ou removidos para economia de recursos computacionais e energéticos. A seguir, cada camada da arquitetura proposta é explicada seguindo uma ordem de cima para baixo.

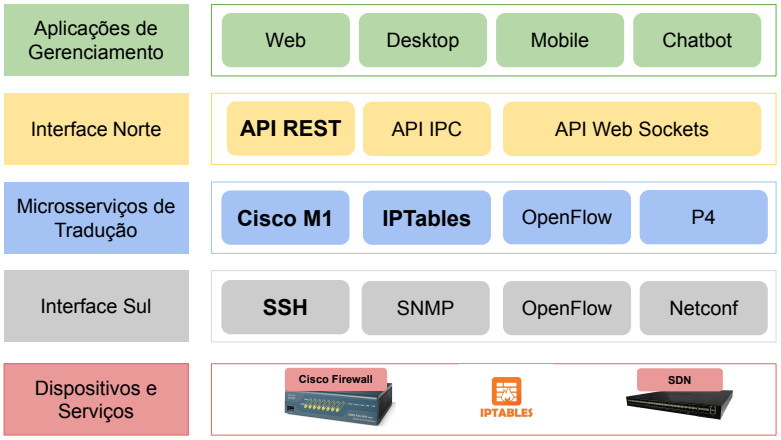


Figura 1. Arquitetura Proposta

As *aplicações de gerenciamento* 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 tecnologias, como *web*, *desktop*, *mobile* e *chatbots*. É importante que as interfaces representem as políticas de segurança em um nível alto de abstração e de forma inteligível e não ambígua para agilizar o processo de gerenciamento e evitar erros de configuração.

A *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.

Os *microsserviços de tradução* recebem as políticas de segurança através da interface norte e as traduzem para comandos específicos a cada tipo de solução de firewall. Na prática, o microsserviço Cisco M1 irá traduzir as políticas de segurança para comandos específico de um firewall ASA 5500-X, por exemplo. No caso de padrões como o OpenFlow, pode haver um serviço de tradução para cada versão (*e.g.*, v1.0, v1.1, v1.2, v1.3, v1.4, v1.5). Supondo que uma rede de grande porte possua sete soluções diferentes de firewall, serão necessários até sete microsserviços de tradução, um para cada linha distinta de firewalls.

A *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. Já um firewall SDN pode ser configurado utilizando o próprio padrão do OpenFlow (*e.g.*, conexão TLS), por exemplo.

Finalmente, os *dispositivos e serviços* correspondem às 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.

3. Resultados Preliminares

3.1. Implementação do protótipo

Como forma de validar a arquitetura, foi desenvolvido um protótipo, utilizando a linguagem de programação Python versão 3.7, que implementa cada uma das cinco camadas propostas. A implementação do protótipo atende a definição de intenções para o gerenciamento de redes, demonstrando que a arquitetura proposta pode ser utilizada para representar (a) o recebimento da intenção, (b) a tradução da intenção e (c) a aplicação das configurações, conforme definido no padrão em proposição no IETF [Sun et al., 2019].

A versão atual do protótipo[2] contempla (destacado em negrito na Figura 1) uma API REST (interface norte), dois microsserviços de tradução para firewalls Cisco ASA 5505 e GNU/Linux IPTables, um conector SSH (interface sul). Para os testes, na camada de dispositivos e serviços foram utilizados um firewall Cisco ASA 5505 e um firewall IPTables em um servidor rodando Ubuntu Server 18.04.

Na aplicação de gerenciamento, representada por uma aplicação Python, são suportados três tipos de intenção: ACL (*Access Control List*), NAT (*Network Address Translation*) do tipo 1:1, e *traffic shaping*. As intenções são representadas utilizando a linguagem de definição de intenções NILE [Jacobs et al., 2018]. Acreditamos que essa linguagem atenda os requisitos de abstração e inteligibilidade para administradores de redes. Entretanto, para a implementação do protótipo foi necessário adicionar duas novas tags à NILE: *range*, utilizada em conjunto com as tags *"from"* e *"to"* para representar casos onde a origem ou o destino do tráfego seja uma rede e não um único nó; e *apply*, necessária para indicar a inserção ou remoção de uma regra, que é sempre sucedida pelos comandos *insert* ou *remove*, indicando a operação a ser realizada.

A API REST recebe as intenções definidas pelo usuário, através do método POST, e extrai as informações (*e.g.* origem, destino, porta, protocolo) que irão compor a sintaxe

[2] `https://github.com/mmfiorenza/intents_hnfw`

nos tradutores especializados. Esses dados são adicionados em um dicionário Python que é enviado aos serviços de tradução. Além disso, a API também é responsável pela checagem de sintaxe da linguagem de definição de intenções.

Os dois tradutores, para Cisco ASA e IPTables, foram implementados utilizando a linguagem de modelagem de templates Jinja2[3]. Esta linguagem é utilizada para criar templates contendo partes fixas e dinâmicas das sintaxes dos comandos dos diferentes firewalls. As partes dinâmicas (*e.g.* endereços IP, largura de banda) são preenchidas automaticamente pela Jinja2 a partir do dicionário de dados enviado pela API REST. O serviço de tradução recebe este dicionário, realiza as conversões necessárias (*e.g.* conversão da largura de banda de Mbps para Kbps), e utiliza a linguagem Jinja2 para gerar a sequência de comandos a serem aplicados nos dispositivos de firewall.

Finalmente, os microsserviços utilizam um conector SSH para efetivar a aplicação das regras nas respectivas soluções de firewall Cisco e Linux/IPTables. O conector foi implementado utilizando a biblioteca Netmiko[4] e recebe como parâmetros endereço IP, porta e usuário a serem utilizados para estabelecer a conexão SSH. O conector retorna para o microsserviço informações de estado (*e.g.*, sucesso ou falha) da aplicação da política na solução de firewalls.

3.2. Avaliação inicial

Para validar o funcionamento do protótipo, foram definidas duas políticas, no formato de intenções, uma de ACL e outra de *traffic shapping*. O ambiente utilizado para testar ambas as intenções foi: (a) um firewall Cisco ASA-5505, controlando o tráfego entre as duas redes, uma interna e outra externa; (b) um servidor Linux Ubuntu Server 18.04, conectado à rede externa; (c) um desktop Linux Mint 19.1, conectado à rede interna; (d) um servidor Linux Ubuntu Server 18.04 executando o protótipo, conectado ao firewall por uma interface específica para gerenciamento. Como interface sul (ou conector), entre os microsserviços e o firewall, foi utilizado o protocolo SSH.

A intenção da ACL foi utilizada para interromper temporariamente o tráfego ICMP (gerado no Linux utilizando o comando `ping`) entre os *hosts* desktop (c) e servidor (b). A Figura 2(a) apresenta a intenção, em linguagem NILE, enviada aos microsserviços utilizando a API REST. A Figura 2(b) apresenta a saída do comando `ping`, isto é, as respostas das requisições ICMP antes, durante e depois da aplicação da política.

```
define intent acl:
  from        endpoint('192.168.1.2')
  to          endpoint('200.19.30.2')
  rule        deny('icmp')
  apply       insert/remove
```

(a) Intenção para ACL

```
usuario@xyz:~$ ping -i4 -O 200.19.30.2
PING 200.19.30.2 (200.19.30.2) 56(84) bytes of data.
64 bytes from 200.19.30.2: icmp_seq=1 ttl=64 time=0.700 ms
64 bytes from 200.19.30.2: icmp_seq=2 ttl=64 time=0.837 ms
64 bytes from 200.19.30.2: icmp_seq=3 ttl=64 time=0.893 ms
no answer yet for icmp_seq=4
no answer yet for icmp_seq=5
64 bytes from 200.19.30.2: icmp_seq=6 ttl=64 time=0.779 ms
64 bytes from 200.19.30.2: icmp_seq=7 ttl=64 time=0.844 ms
64 bytes from 200.19.30.2: icmp_seq=8 ttl=64 time=0.805 ms
```

(b) Resposta ICMP do comando `ping`

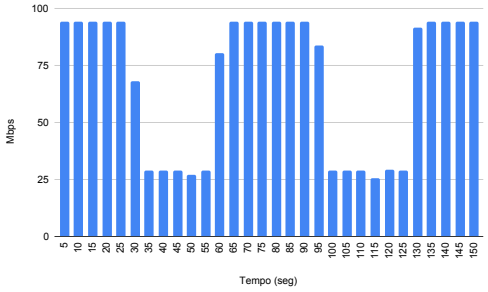
Figura 2. Prova de conceito para aplicação e remoção da ACL

[3] `https://jinja.palletsprojects.com/en/2.11.x/`
[4] `https://github.com/ktbyers/netmiko`

No segundo teste, foi implementada e aplicada, entre os *hosts* desktop (c) e servidor (b), a política de *traffic shaping* apresentada na intenção da Figura 3(a). Como pode ser observado, foi aplicado um limite de tráfego de 30 Mbps (`throughput('30Mbps')`). A ferramenta *iperf*[5] foi utilizada para gerar o tráfego entre os *hosts* e coletar as informações de vazão. O gráfico da Figura 3(b) apresenta o comportamento da vazão da conexão antes, durante e depois de duas aplicações e remoções das regras de *traffic shaping*.

```
define intent traffic_shaping:
  from        endpoint('192.168.1.2')
  to          endpoint('200.19.30.2')
  for         traffic('udp/5555')
  with        throughput('30Mbps')
  apply       insert/remove
```

(a) Intenção para *Traffic Shaping*



(b) Recorte de 150 segundos da vazão de tráfego

Figura 3. Prova de conceito para aplicação e remoção de *Traffic Shaping*

Durante as avaliações realizadas foram identificados alguns aspectos que podem ser melhor investigados e aperfeiçoados nas futuras versões do protótipo. O tempo de aplicação da intenção chegou a aproximadamente oito segundos, o que pode ser crítico no caso de aplicação de políticas de segurança em tempo real para bloquear ataques em andamento, por exemplo. É interessante observar que a maior parte desse tempo foi consumida pelo conector SSH. Um dos trabalhos futuros será avaliar as questões de desempenho e segurança relacionadas ao uso de interfaces sul como esta, baseada no SSH.

4. Conclusão

Neste trabalho, nós apresentamos e discutimos uma arquitetura multicamada e modular que permite a adição, sob-demanda, de novos componentes (*e.g.*, microsserviços de tradução de regras) para gerenciamento unificado de soluções de firewall tradicionais (*e.g.*, Cisco ASA e Linux/IPTables) e emergentes (*e.g.*, como os baseados em SDN/OpenFlow, SDN/P4), que precisam co-existir em uma rede híbrida. Como prova de conceito, foi implementado um protótipo, cuja versão atual permite gerenciar simultaneamente firewalls Cisco ASA-5505 e Linux/IPTables. Os resultados de uma avaliação experimental demonstram o funcionamento e a viabilidade técnica da arquitetura proposta.

Como trabalhos em andamento e futuros, podemos citar: (a) analisar questões de desempenho da solução sob diferentes cenários; (b) implementar microsserviços de tradução e conectores para soluções baseados em OpenFlow e P4; (c) realizar testes de funcionamento e operação do sistema como ferramenta para mitigação de incidentes; e (d) analisar a representação das especificidades de cada firewall através de uma linguagem universal baseada em *intents*.

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

Referências

Abdallah, S., Elhajj, I. H., Chehab, A., and Kayssi, A. (2017). Fuzzy decision system for technology choice in hybrid networks. In *Fourth International Conference on Software Defined Systems (SDS)*, pages 106–111. IEEE.

Behringer, M. H., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B. E., Jiang, S., and Ciavaglia, L. (2015). Autonomic Networking: Definitions and Design Goals. RFC 7575.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al. (2014). P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95.

Caprolu, M., Raponi, S., and Di Pietro, R. (2019). Fortress: an efficient and distributed firewall for stateful data plane sdn. *Security and Communication Networks*, 2019.

Datta, R., Choi, S., Chowdhary, A., and Park, Y. (2018). P4Guard: Designing P4 based firewall. In *IEEE Military Communications Conference (MILCOM)*, pages 1–6. IEEE.

Esposito, F., Wang, J., Contoli, C., Davoli, G., Cerroni, W., and Callegati, F. (2018). A behavior-driven approach to intent specification for software-defined infrastructure management. In *IEEE Conference on NFV-SDN*, pages 1–6. IEEE.

Fiessler, A., Lorenz, C., Hager, S., and Scheuermann, B. (2018). FireFlow-high performance hybrid SDN-firewalls with OpenFlow. In *IEEE 43rd Conference on Local Computer Networks (LCN)*, pages 267–270. IEEE.

Hu, H., Han, W., Ahn, G.-J., and Zhao, Z. (2014). FLOWGUARD: building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 97–102.

Jacobs, A. S., Pfitscher, R. J., Ferreira, R. A., and Granville, L. Z. (2018). Refining network intents for self-driving networks. In *Proceedings of the Afternoon Workshop on Self-Driving Networks*, pages 15–21.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2014). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

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.

Othman, W. M., Chen, H., Al-Moalmi, A., and Hadi, A. N. (2017). Implementation and performance analysis of SDN firewall on pox controller. In *IEEE 9th International Conference on Communication Software and Networks (ICCSN)*, pages 1461–1466. IEEE.

Sinha, Y., Haribabu, K., et al. (2017). A survey: Hybrid SDN. *Journal of Network and Computer Applications*, 100:35–55.

Song, H. (2013). Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane. In *Proceedings of the 2nd ACM SIGCOMM workshop on HotSDN*, pages 127–132.

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

Vörös, P. and Kiss, A. (2016). Security middleware programming using p4. In *Human Aspects of Information Sec., Privacy, and Trust*, pages 277–287. Springer.

Yang, L., Anderson, T. A., Gopal, R., and Dantu, R. (2004). Forwarding and Control Element Separation (ForCES) Framework. RFC 3746.

Zerkane, S., Espes, D., Le Parc, P., and Cuppens, F. (2016). A proactive stateful firewall for software defined networking. In *International Conference on Risks and Security of Internet and Systems*, pages 123–138. Springer.

Zkik, K., El Hajji, S., and Orhanou, G. (2019). Design and implementation of a new security planee for hybrid distributed sdns. *Journal of communications.*, 14(1):26–32.