NETWORK FUNCTIONS VIRTUALIZATION-BASED SECURITY PROPOSALS FOR CLOUD COMPUTING ENVIRONMENTS

Leopoldo Alexandre Freitas Mauricio

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia Elétrica.

Orientadores: Otto Carlos Muniz Bandeira Duarte
Marcelo Gonçalves Rubinstein

Rio de Janeiro
Março de 2019

NETWORK FUNCTIONS VIRTUALIZATION-BASED SECURITY PROPOSALS FOR CLOUD COMPUTING ENVIRONMENTS

Leopoldo Alexandre Freitas Mauricio

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Otto Carlos Muniz Bandeira Duarte, Dr.Ing.

Prof. Marcelo Gonçalves Rubinstein, D.Sc.

Prof. Mauro Sergio Pereira Fonseca, Dr.

Prof. Edmundo Roberto Mauro Madeira, D.Sc.

Prof. Luís Henrique Maciel Kosmalski Costa, Dr.

Prof. Pedro Braconnot Velloso, Dr.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2019

Freitas Mauricio, Leopoldo Alexandre

Network Functions Virtualization-Based Security Proposals for Cloud Computing Environments/Leopoldo Alexandre Freitas Mauricio. – Rio de Janeiro: UFRJ/COPPE, 2019.

XII, 76 p.: il.; 29, 7cm.

Orientadores: Otto Carlos Muniz Bandeira Duarte
Marcelo Gonçalves Rubinstein

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2019.

Referências Bibliográficas: p. 65 – 76.

1. Security attacks. 2. Software-Defined Networking. 3. Network Function Virtualization. I. Duarte, Otto Carlos Muniz Bandeira *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Acknowledgments

I thank my mom Solange Mauricio, my father Luiz Octavio Mauricio, my wife Beatriz, and my kids Sophia and Pedro who have always been by my side, for all their love and understanding. In particular, I thank my parents for the support they give me at all times and for always motivating me to move on.

Thanks also to all the friends I made in the Grupo de Teleinformática e Automação (GTA), since they have always contributed positively to the conclusion of this work.

Thanks also to all the teachers who participated in obtaining this degree. In particular, I thank my advisors, Professors Otto Carlos Duarte and Marcelo Rubinstein, for all the advice, dedication and especially patience during the orientation. Also, a special mention to Professor Marcelo Rubinstein, for the discussion and contribution to this work and personal life. I would also like to thank Professors Luís Henrique Maciel Kosmalski Costa, Miguel Elias Mitre Campista, Pedro Braconnot Velloso and Rodrigo de Souza Couto for making the GTA/UFRJ laboratory a pleasant working environment.

I thank Professors Mauro Pereira Fonseca and Edmundo Roberto Mauro Madeira for their participation in the examining jury.

I thank the employees of the Electrical Engineering Department (PEE) of COPPE/UFRJ, Mauricio, Daniele, Roberto and Marcos for prompt service in the program secretariat.

I thank all the people who directly or indirectly collaborated with this stage of my life. Finally, a special thanks to Globo.com, who supported me economically and with free time that I needed for this achievement.

Finally, I thank CAPES, CNPq, FAPERJ and FAPESP (2015/24514-9, 2015/24485-9, and 2014/50937-1) for the funding of this work.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

PROPOSTAS DE SEGURANÇA BASEADAS EM VIRTUALIZAÇÃO DE FUNÇÃO DE REDE PARA AMBIENTES DE COMPUTAÇÃO EM NUVEM

Leopoldo Alexandre Freitas Mauricio

Março/2019

Orientadores: Otto Carlos Muniz Bandeira Duarte
Marcelo Gonçalves Rubinstein

Programa: Engenharia Elétrica

Esta tese implementa e avalia propostas de segurança baseadas em virtualização de funções de rede para ambientes de computação em nuvem. Suas principais contribuições são: (i) transferimos o grande número de regras de segurança tipicamente implementadas nos roteadores de topo de *rack* de um datacenter estudado para funções virtualizadas de segurança *firewall* (FW-VSF), criadas em *hardware* de prateleira. Assim, reduzimos custos e liberamos recursos de TCAM para acelerar as operações de roteamento. Avaliamos o desempenho de uma FW-VSF criada com o Iptables (Iptables FW-VSF) em função das demandas encontradas no datacenter estudado. (ii) propomos e implementamos o *framework* ACLFLOW, que é uma estrutura de segurança NFV/SDN que cria e gerencia *firewalls* OpenFlow (FW-VSFs) distribuídos, como uma alternativa ao uso de TCAMs de roteador ou *middleboxes* de segurança, para controlar o tráfego de máquinas virtuais em um ambiente de computação em nuvem. O ACLFLOW converte regras de segurança regulares (IP de origem/destino, porta de origem/destino e protocolo) em regras de filtragem OpenFlow, cria e gerencia grandes quantidades de regras em OpenFlow FW-VSFs distribuídos, além de orquestrar e acelerar sua implantação em nuvens de produção. Também propomos um algoritmo que adapta oportunamente as regras do FW-VSF às mudanças nas condições de tráfego, priorizando dinamicamente as mais populares para melhoria de desempenho. (iii) propomos e implementamos uma arquitetura de segurança NFV que fornece proteção automática e eficiente contra ataques, encadeando uma VSF ao fluxo de dados para bloquear dinamicamente o tráfego malicioso, sem interromper o benigno. Prototipamos as propostas na plataforma aberta para NFV (*Open Platform for NFV* - OPNFV) e avaliamos seus desempenhos.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

NETWORK FUNCTIONS VIRTUALIZATION-BASED SECURITY PROPOSALS FOR CLOUD COMPUTING ENVIRONMENTS

Leopoldo Alexandre Freitas Mauricio

March/2019

Advisors: Otto Carlos Muniz Bandeira Duarte
Marcelo Gonçalves Rubinstein

Department: Electrical Engineering

This thesis implements and evaluates Network Functions Virtualization-based security proposals for cloud computing environments. The following are the main contributions of this work: (i) We move a large number of security rules implemented in Top-of-Rack routers of a studied virtualized data center to virtual firewalls created in commodity hardware. Thus, we can reduce costs and release TCAM resources for accelerating routing operations. We evaluate an Iptables FireWall Virtual Security Function (FW-VSF) performance against the demands encountered in the production data center studied. (ii) We propose and implement the ACLFLOW framework that is an NFV/SDN security framework that creates and manages distributed OpenFlow FW-VSFs as an alternative to using router TCAMs or specialized security middleboxes to control traffic from virtual machines in a cloud computing environment. ACLFLOW translates regular security rules (source/destination IP, source/destination port, and protocol) into OpenFlow filtering rules. Besides, it creates and manages large amounts of OpenFlow rules on distributed firewall VSFs and implements mechanisms that orchestrate and accelerate the deployment of OpenFlow FW-VSF in production clouds. We also propose an algorithm that timely adapts FW-VSF rules to changes in the traffic conditions by dynamically prioritizing the most popular rules to improve performance. (iii) We propose and implement an NFV security architecture that provides automatic and efficient protection against attacks, by chaining a Virtual Security Function to the data stream to dynamically block malicious traffic without stopping the benign one. We prototype our security proposals into the Open Platform for NFV (OPNFV) and evaluate their performances.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, the Internet-provided communication infrastructure has been transforming the world economy. New markets and services for people and businesses have emerged from the development of virtualization techniques, the growth of cloud computing, and the Internet of Things (IoT), which enables interaction between different types of networks and devices, such as home and business networks, sensor networks, surveillance cameras, TVs, cell phones, cars, etc. [1]. Nevertheless, the Internet has evolved presenting several security issues, such as issues of confidentiality, integrity, and availability. Security attacks are increasingly frequent and cause great harm to people and organizations because the Internet was not originally designed to deal with malicious nodes [2–5].

One of the most popular attacks is malware one. Malware is a malicious code created to damage the victim's resources intentionally or to execute malicious activity against the interest of the computer user. It can be a worm, a Trojan, a virus, a ransomware software, etc. [6]. Malicious codes can exploit vulnerable Web applications by performing Structured Query Language Injection (SQLI) attacks to steal sensitive data, such as financial or personal private information [7]. Likewise, a malware could inject several malicious codes into operating systems to execute Local File Inclusion (LFI) attacks, Web exploitation attacks can redirect users towards phishing sites (cross-Site Scripting - XSS), deface or vandalize websites, etc. [8].

Many companies have been dealing with data breaches and significant financial losses caused by vulnerability exploitations [9–11]. The amount of data breach reported in 2015 is higher than in previous years. In 2011 there were 1281 security incidents, while 3930 were recorded in 2015, exposing 736 million sensitive data [12]. Only one database exploit attack illegally exposed 191 million users' personal information [13]. Akamai security report [3] shows that 88% of Web application exploits in 2016 were related to SQLI and LFI. These problems happen because there are more and more vulnerable systems accessible through the Internet and available attack codes to exploit their weaknesses. Accenture cost of cybercrime study[2] shows

that 98% of the Internet companies had to deal with malware attacks in 2017, 67% with Web application attacks, and the two higher expenses were incurred in SQLI and LFI (about 2.4 million and 2 million US dollars).

Distributed Denial of Service (DDoS) attacks aims to consume the victim's resources to cause unavailability. Some have already reached rates above 1.1 Tb/s [14]. The increase in the intensity of denial of service attacks is related to the low-security level of the IoT devices, which present a progressively more substantial computing power without implementing robust security mechanisms. 70% of the IoT devices evaluated by HP in 2016 [15] presented security vulnerabilities. About 25 vulnerabilities, on average, were found on each device evaluated. In more than 80% of the cases, it was not possible to use passwords with satisfactory complexity and length, approximately 70% of the devices do not encrypt the communication, and 60% have vulnerable user interfaces and firmware. For these reasons, such devices are increasingly being used to augment the volume of DDoS attacks [14].

Many security attack sources are botnet nodes [16], a set of zombie machines with malware that allows an attacker to control them remotely, also known as "infected" devices. In that case, owners of enslaved hosts, which may be computers, TVs, cameras, mobile devices, and so on, do not know that these hosts are generating malicious traffic. Some estimates indicate that there are already botnets with more than one million zombie devices [17]. Moreover, source IPs of security attacks may belong to a Network Address Translation (NAT) gateway that serves many network nodes. Although blocking source IPs should be an effective countermeasure against most of the denial of service attacks [18], efficient network security systems must also be able to prevent vulnerability exploitation, such as Structured Query Language Injection, Local File Inclusion, and Cross-Site Scripting, without affecting the benign traffic of the same source IP. However, most traditional security solutions are usually unable to segregate malicious traffic from the benign one when they are generated from the same source so that they commonly apply filters to block all traffic generated by the attacking node.

Several security mechanisms are used to protect networks and systems. Encryption hinders unauthorized reading of end-to-end communications traffic, and authentication and authorization mechanisms control access to network resources. An Intrusion Prevention System (IPS) is a Network Function (NF) that examines traffic flows, detects and prevents vulnerability exploits, whereas Web Application Firewalls (WAFs) filters traffic based on the analysis of the packet content. Routers traditionally implement Access Control Lists (ACLs) to filter traffic by selectively evaluating the five-tuple source/destination IP, source/destination port, and protocol.

Network Functions generally process security policies at line speed using Ternary

Content Addressable Memories (TCAMs) [19] which are memory tables that enable fast search operations by reading all the rules in parallel when the matching entries for each packet need to be identified. Although NF can use TCAMs to process security policies at wire speed, such hardware solution is power hungry and is not designed to handle large amounts of rules. Many rules may be required, whereas the storage capacity of TCAMs is scarce and expensive [20]. TCAMs limit the number of filters to a few thousand rules, between 2-4k, and are up to 400 times more expensive than RAMs, with power consumption up to 100 times higher [21]. For this reason, security Network Functions usually are expensive specialized middleboxes found in a data center and Internet Service Provider (ISP) networks that require a high deployment time and rely on the manual configuration of network administrators [22, 23].

Network operators typically rely on security middleboxes to protect systems and applications and control traffic between networks. Those middleboxes are hardware Network Functions, such as router Access Control Lists, physical Web Application Firewalls, Intrusion Prevention Systems devices and so on [24]. A typical hardware-based north-south data center traffic created by network administrators to secure a web application, for instance, may traverse a BGP router, a Monitor, an IPS, a Load Balancer, an ACL, and a WAF [25, 26]. Each physical NF of that sequential service chain has a specific position, and it is not easy to change them within the network topology to provide new services, meet new security demands or deal with capacity changes. For these reasons, relying on physical security middleboxes does not meet the requirements of cloud computing where computing resources, development platforms, and software must quickly adapt to variations in user demand and be offered as a service [27, 28].

Network Functions Virtualization (NFV), standardized by the European Telecommunications Standards Institute (ETSI), is a new technology that aims to make networks more agile and flexible and to reduce equipment and operating costs [29]. NFV proposes the deployment of Virtual Network Functions (VNFs) as software packages running on Commercial Off-The-Shelf (COTS) hardware to be an alternative to using specialized network and security devices [30]. Thus, with NFV, the number of middleboxes decreases as well as the costs with heat dissipation, electricity consumption, and maintenance. Then, it is possible to reduce both CAPEX and OPEX. Network Functions Virtualization also enables software-based Service Function Chaining (SFC) in which policy routes can be dynamically created to steer traffic through hardware-agnostic VNFs [31, 32].

Software-based SFCs are not tied to network topologies, and their physical connections since NFV decouples Network Functions from specialized hardware to increase flexibility. As such, Virtual Network Functions can be chained to deliver

services at the edge, closer to end users, and fine-grain service chains can deploy VNFs on-demand to meet security or another specific constraint [33]. The Internet Engineering Task Force (IETF) proposes the creation of service chains using the Network Service Header (NSH) protocol [34]. The NSH protocol encapsulates network traffic that is routed between Virtual Network Functions to implement new services, such as chaining multiple VNFs located on remote Internet sites. Although the NSH enables new services, it increases the NFV overhead [35]. Moreover, NSH has the disadvantage of being a clean-slate [36] network solution due to it demands changes in the Network Function Operating System (OS) kernel to be appropriately used.

Cloud computing providers can offer Virtual Network Functions of different types as a service to their users. So, a client would be able to instantiate an application firewall to protect a Web site or a database service that is hosted in the cloud provider. ISPs can also use NFV to offer new services to their customers. Currently, ISPs install router devices on the users' home/business network, which have software capable of connecting only the ISP client's network to the Internet. For this reason, ISPs always need a technical workforce and new hardware installation in the customer environment to provide new services. With NFV, client middleboxes can offer software that supports the remote creation of different types of Virtual Network Functions. Such devices can also become Virtual Network Functions instantiated within the ISP's network. Therefore, ISPs can use NFV to offer new services to their clients, such as a Deep Packet Inspection VNF, a custom firewall VNF to protect security cameras, an application firewall VNF to secure financial operations made by credit card machines of a network of pharmacies, and so on.

1.1 Objectives

The goal of this thesis is to present proposals regarding the problems of benign traffic blocking and performance of security NFV solutions. We argue that efficient protection systems should be able to dynamically steer malicious traffic through security software-based Service Chains to block exploitation attacks without stopping the benign traffic. Besides, we also argue that each cloud computing physical server can host a firewall Virtual Network Function as an alternative to using TCAMs or specialized security middleboxes. Therefore, the security policies are implemented closest to the object to be protected, which is the Virtual Machine (VMs).

This thesis first evaluates and discusses the NFV overhead of a firewall implemented as a security Virtual Network Function (or Virtual Security Function). Then, we propose the ACLFLOW framework with mechanisms to implement and manage distributed firewalls in cloud computing environments. The ACLFLOW framework

implements OpenFlow firewalls on distributed virtual switches to address the TCAM storage capacity problem. Moreover, this work proposes and implements an algorithm that prioritizes firewall rules to improve NFV performance. This work also proposes and implements an NFV security architecture to provide automatic and efficient protection against attacks by dynamically blocking malware without affecting benign traffic. We build prototypes in Open Platform for NFV (OPNFV), which is an open source Linux Foundation platform developed to accelerate the NFV implementation in real networks [37]. The evaluations measure the efficiency of the NFV security architecture and the performance of the security framework and its firewall Virtual Security Function using commodity machines. The main contributions of this thesis are:

- We evaluate the performance of an Iptables FireWall (FW) implemented as a Virtual Security Function (Iptables FW-VSF) using commodity servers [38]. We propose a case study to evaluate if an FW-VSF can handle 4000 rules, that is, the maximum number of Access Control Lists we can implement in the Top of Rack routers TCAMs found in the production data center studied.

- We propose and implement a cost-effective security framework, named ACLFLOW, that translates regular ACLs (source/destination IP, source/destination port, and protocol) into OpenFlow filtering rules [26] and manages distributed FireWall VSFs (FW-VSFs) that are implemented on each cloud computing physical server, closest to the Virtual Machine (VMs), to protect cloud computing environments by addressing the TCAM limited size problem [39]. ACLFLOW also simplifies orchestration and accelerates the implementation of FW-VSFs in production clouds.

- We propose an algorithm that dynamically prioritizes the rule with the highest traffic volume (the most popular rule) to increase the ACLFLOW FW-VSF performance, because the use of Virtual Security Functions generally presents lower performance [29, 38, 40]. The evaluation of the ACLFLOW algorithm efficiency uses real traces from production OpenFlow FW-VSFs [39, 41].

- We design an NFV security architecture that provides automatic and efficient protection against attacks by dynamically steering traffic through a Web Application Firewall Virtual Security Function[1] (WAF-VSF), which can block only malicious activity without interrupting benign traffic from the same source [42].

[1]Security Virtual Network Functions and Virtual Security Functions are synonyms in this work.

1.2 Publications

The results of this thesis were published in the following events:

- GTA-18-39 - Mauricio, L. A. F., Rubinstein, M. G., and Duarte, O. C. M. B. - "ACLFLOW: An NFV/SDN Security Framework for Provisioning and Managing Access Control Lists", in IEEE 9th International Conference Network of the Future - NoF'2018, Poznan, Poland, November 2018. `http://www.gta.ufrj.br/ftp/gta/TechReports/MRD18.pdf`

- GTA-17-03 - Mauricio, L. A. F., Alvarenga, I. D., Rubinstein, M. G., and Duarte, O. C. M. B. - "Uma Arquitetura de Virtualização de Funções de Rede para Proteção Automática e Eficiente contra Ataques", in XXII Workshop de Gerência e Operação de Redes e Serviços (WGRS'2017) - SBRC'2017, Belém- Pará, PA, Brazil, May 2017. `http://www.gta.ufrj.br/ftp/gta/TechReports/MARD17.pdf`

- GTA-16-48 - Mauricio, L. A. F., Rubinstein, M. G., and Duarte, O. C. M. B. - "Proposing and Evaluating the Performance of a Firewall Implemented as a Virtualized Network Function", in IEEE 7th International Conference Network of the Future - NoF'2016, Buzios-RJ, Brazil, November 2016. `http://www.gta.ufrj.br/ftp/gta/TechReports/MRD16.pdf`

- GTA-17-15 - Sanz, I. J., Alvarenga, I. D., Andreoni Lopez, M. E., Mauricio, L. A. F., Mattos, D. M. F., Rubistein, M. G., and Duarte, O. C. M. B. - "Uma Avaliação de Desempenho de Segurança Definida por Software através de Cadeias de Funções de Rede", in XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg'17, Brasília, DF, Brazil, November 2017. `http://www.gta.ufrj.br/ftp/gta/TechReports/SAA17.pdf`

1.3 Text Organization

The rest of this thesis is organized as follows. In Chapter 2 we introduce Software-Defined Networking and Network Functions Virtualization technologies. Besides, we show that Service Function Chaining, proposed by NFV, can be created with OpenFlow policies on SDN-enabled switches. Chapter 3 discusses related work. In Chapter 4, we present a case study to evaluate if ACLs found in ToRs of the studied data center can be moved to Iptables FW-VSFs implemented on commodity servers to release TCAM resources. Chapter 5 details the proposed ACLFLOW

framework and presents the prioritization algorithm for increasing OpenFlow FW-VSF performance. In Chapter 6 we present the proposed NFV security architecture that automatically chains Web Application Firewall VSFs to protect Web servers against malware without affecting the benign traffic of the same source IP. Chapter 7 presents conclusions and future work.

Chapter 2

Software-Defined Networking and Network Functions Virtualization

Network Functions Virtualization is a new technology aimed at reducing the complexity of operation and management of Network Functions and improving network efficiency and performance [43]. To reduce CAPEX and OPEX, NFV builds Virtual Network Functions on commodity servers.

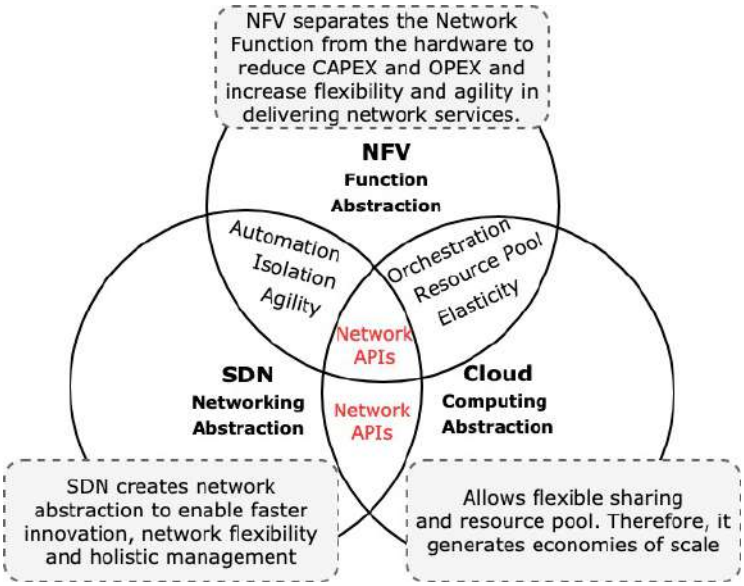

Figure 2.1: Relationship between NFV, SDN and Cloud Computing.

Figure 2.1 shows that there is a relationship between NFV, cloud computing and Software-Defined Networking. Thus, NFV can use the automation and isolation provided by SDN, and the orchestration and on-demand resource allocation capabilities offered by cloud computing to create VNFs and service chains. Accordingly, Mijumbi *et al.* states that the best approach to accelerate the development and maturation of NFV is combining it with SDN technologies and cloud computing capabilities [29, 44]. In this chapter, we present the main features of SDN and NFV.

Cloud computing will not be presented because it is a well-discussed and general topic.

2.1 Software-Defined Networking

Traditional networking equipment has a data plane and a control plane. The control plane supports several protocols and is responsible for routing rules configuration into routing tables. The data plane forwards packets by following the instructions written by the control plane. Despite this, traditional networking equipment is not flexible. Programming new logic of packet routing other than that implemented by the IP protocol, for instance, is not permitted [5]. For this reason, these devices "ossify" the network core [36].

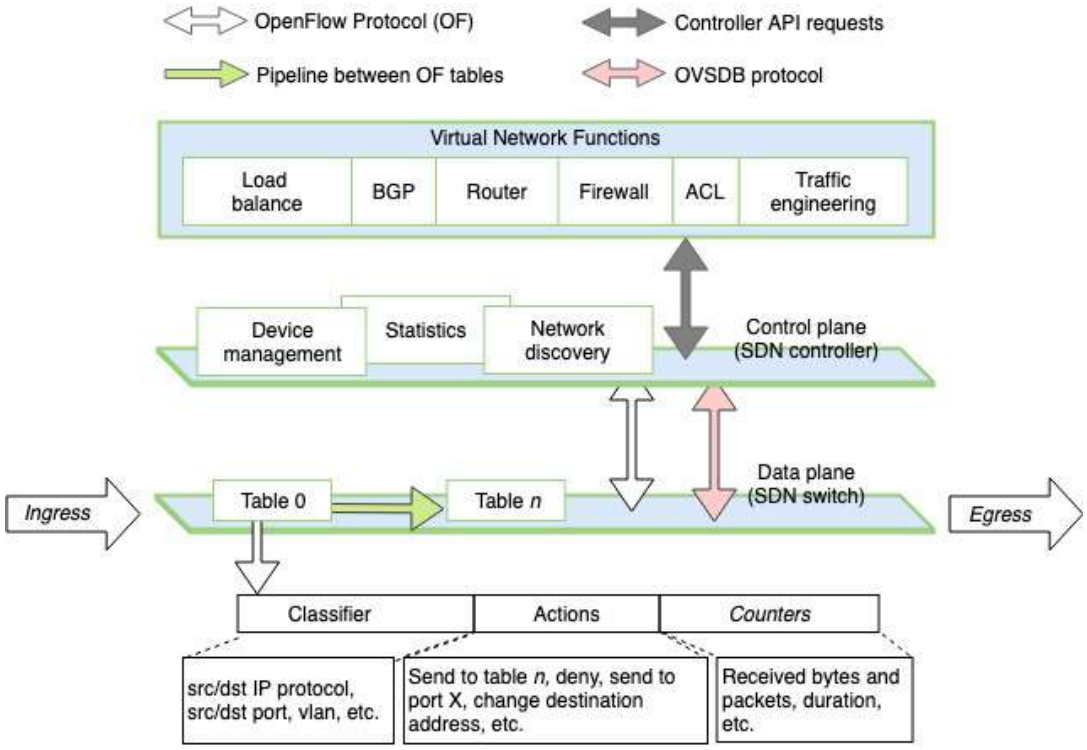

Figure 2.2: Multiple types of Virtual Network Functions created over an SDN architecture.

Software-Defined Networking (SDN) separates the network control function of the forwarding function. Its most popular protocol is OpenFlow (OF) [45, 46]. Through it, the control plane inserts flow rules into the data plane to control its behavior (Figure 2.2). Software-Defined Networking enables centralized control and a global view of the network. SDN controllers manage the network topology, provide an Application Programming Interface (API) to facilitate its operation and many

times use the Open vSwitch Database Management Protocol (OVSDB) to enable the configuration of switch interfaces, VLANs, and so on.

Multiple types of Virtual Network Functions can be created in a data plane by combining NFV with SDN technologies. It is possible to orchestrate multiple VNF types as a distributed data plane, from centralized controllers [40]. Figure 2.2 illustrates that a Virtual Network Function can be a load balancer that increases capacity and reliability by distributing the network traffic across multiple servers, based on source IP, destination IP, and so on [47]. Otherwise, the VNF can be a traffic engineering, when data plane rules specify which paths the data flow should follow [48, 49]. The VNF can also be a firewall, which enforces filtering rules [38]. Therefore, with proper rules, the data plane SDN device can act as a layer-two switch, a router, a firewall, a load balancer, a traffic engineering, and so on [33].

Moreover, an SDN can be implemented in either reactive or proactive mode [44–46]. In the reactive mode, the controller installs flow rules in the data plane according to the request made by the OpenFlow switches. Whenever a switch receives a new flow, it sends an instruction request message named *OFF-packet-in* to the SDN controller, which inserts rules into the data plane to allow the traffic. This approach inserts a network delay because the data plane must wait for controller intervention whenever switches receive a new flow. Moreover, the controller and switch buffers may become overloaded when the number of simultaneous new flows is large [50]. Consequently, an SDN virtual security function configured in the reactive mode may block traffic incorrectly or cause security issues.

All filtering rules are previously installed in the data plane when a proactive SDN is used to provide security. Thus, when the first packets of a new flow reach the OpenFlow switch, those that do not match the previously installed rules are immediately blocked, and the matching packets are forwarded without setup delays [51]. This characteristic is very interesting for security solutions.

Fail secure and fail standalone modes can be set up in an SDN switch. Every time an OpenFlow switch is configured in the fail standalone mode and loses its connection with the controller it removes all flow rules that have been inserted by the controller, and it starts to act as a legacy standalone Ethernet switch. On the other hand, the SDN switch in the fail secure mode persists all rules programmed by the controller. Those rules expire in the fail secure mode only if a timeout value is set.

Software switches are widely used by virtualization tools to interconnect virtual machines and forward their packets. They can store large amounts of rules, and some already reach rates of up to 40 Gb/s although running on machines with only four cores [21]. Open vSwitch (OVS) is an example of a software switch, widely used in cloud computing implementations.

When an Open vSwitch belongs to an SDN data plane, its packet classification evaluates the first packets of a flow in user space (OVS pipeline), through queries executed in several hash tables (Tables 0 to n in Figure 2.3). The number of OVS hash tables queried in the OVS pipeline varies according to the number of OpenFlow rule types. It happens because the OVS packet classification algorithm creates a particular hash table, automatically, in the user space, for each different match field combination being used for the OpenFlow rules it stores. For instance, let P = $\{p_1, ..., p_n\}$ be the OVS OpenFlow rule types. When all OpenFlow rules evaluate only destination IPs (*dst_ip*), there is only one hash table in user space, represented by: ($p_1 \in$ Table 1 | p_1 = [dst_ip, *]), with wildcard (*) representing all undefined fields. Similarly, OVS classifier automatically creates two hash tables in user space when some OpenFlow rules evaluate only destination IPs, while others deal with the source IP (*src_ip*), destination IP (*dst_ip*), and destination port (*dst_port*) fields, such as: ($p_1 \in$ Table 1; $p_2 \in$ Table 2 | p_1 = [dst_ip, *] and p_2 = [src_ip, dst_ip, dst_port, *])

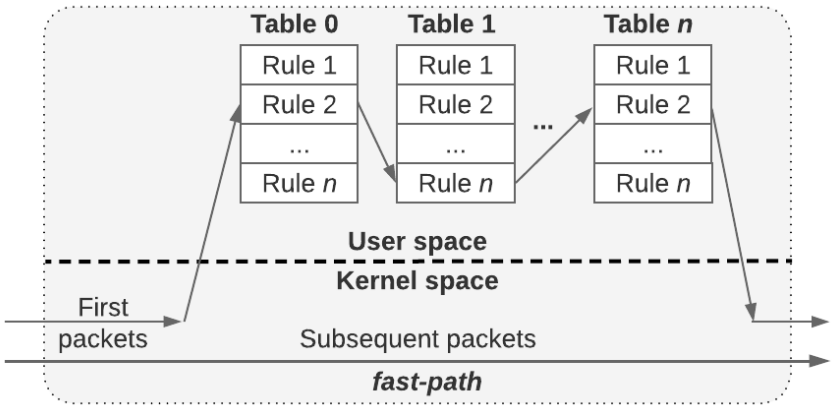

Figure 2.3: Open vSwitch architecture. First packets of a flow are evaluated in the user space and cached decisions are evaluated in kernel.

Therefore, when an OpenFlow firewall has rules created from the five tuples (source IP, destination IP, source port, destination port, and protocol) only up to 32 different hash tables may exist in user space to be queried by OVS packet classifier, no matter how many rules are in the OpenFlow firewall. The OVS pipeline does only one query per hash table at constant execution time, O(1), to find a match. In this case, queries happen at linear time O(n), where n is the number of hash tables that varies between 1 and 32.

When priorities are the same, the match always occurs in the most general OpenFlow rule that has the highest degree of specificity. Table 2.1 illustrates some rules that we have inserted in an OpenFlow firewall to describe its operation. The four OpenFlow rules in Table 2.1 have the same priority (priority = 65000) and allow the Internet Control Message Protocol (ICMP) traffic generated by the host

10.170.29.40/32. The action "NORMAL" means that the OpenFlow firewall should forward the traffic. All matches happen in the OpenFlow rule of row 1 because it is the most generic with the highest degree of specificity among the policies.

All fields of the most generic OpenFlow rule that could be created in a firewall are wildcards (*). That means they are not specified. Therefore, all traffic evaluated by the firewall packet classifier, whatever the protocols, source, and destination IP addresses, source and destination ports, etc. will match this rule. On the other hand, the most specific rule of an OpenFlow firewall instructs the packet classifier to evaluate all bits of all OpenFlow header tuples. In this case, the classifier evaluates whether the particular values indicated in the rule are met by the information included in all the packet header fields by testing all its bits, looking for a match.

The first two rules of Table 2.1 are more generic than rules 3 and four because they evaluate only the header tuples protocol and source IP address, while 3 and 4 assess the tuples protocol, source IP address and destination IP address. Among the more generic rules (rules 1 and 2), which evaluate the same header fields, the first one has a higher degree of specificity. This is because it instructs the packet classifier to evaluate all 32 bits of the source IP address (nw_src=10.170.29.40/32), while in rule 2 the classifier evaluates only 24 bits of the same header tuple. Therefore, although rule 2 can release ICMP traffic generated by any host on the 10.170.29.0/24 network, the match occurs on the first line rule (ID = 1) when the host 10.170.29.40/32 sends ICMP traffic. Rules 3 and four also release ICMP traffic generated by the host 10.170.29.40/32 to the tested destination, which has IP address 10.170.0.170/32. However, they are not used because they are less specific OpenFlow rules than rules 1 and two since they evaluate more header fields. On the other hand, if we remove rules 1 and two from the OpenFlow firewall and keep only rules 3 and 4, traffic will match rule 4 because it will become the rule with the highest degree of specificity between them. This is because all 32 bits of both the source IP address and the destination IP address match the generated ICMP traffic.

Table 2.1: Examples of OpenFlow rules

ID	OpenFlow rules
1	cookie=0x5ba3200000b19, duration=5761.177s, table=0, **n_packets=3112, n_bytes=629614**, idle_age=1, hard_age=65534, priority=65000, icmp, nw_src=10.170.29.40/32 actions=NORMAL
2	cookie=0x5c1a300000b19, duration=5761.100s, table=0, n_packets=0, n_bytes=0, idle_age=65534, hard_age=65534, priority=65000, icmp, nw_src=10.170.29.0/24 actions=NORMAL
3	cookie=0x5a98f00000b19, duration=5761.280s, table=0, n_packets=0, n_bytes=0, idle_age=65534, hard_age=65534, priority=65000, icmp, nw_src=10.170.29.40/32, nw_dst=10.170.0.0/24 actions=NORMAL
4	cookie=0x65d6a00000b19, duration=5761.280s, table=0, n_packets=0, n_bytes=0, idle_age=65534, hard_age=65534, priority=65000, icmp, nw_src=10.170.29.40/32, nw_dst=10.170.0.170/32 actions=NORMAL

OpenFlow rules may include priority. When it happens, regardless of the generality and degree of specificity, the match always happens in the rule with the highest priority that corresponds to the traffic (matches traffic). Besides, the OVS packet classifier reads rule priorities in descending order, from highest to lowest, and caches the forwarding decision in the kernel data-path that speeds up the forwarding of subsequent packets [21, 52]. However, cached rules leave kernel space whenever their cache times expire or every time the SDN controller changes rules in the data plane. When this happens, packets that do not match a rule in the fast-path kernel follow back through the slower user space processing [46]. Every time the matching rule does not have the highest priority, the sequential searching between hash tables goes on until the last hash table. However, when the matching rule already has the highest priority, the linear searching between hash tables is stopped and OVS immediately forwards the packet, without querying other hash tables of the pipeline. Consequently, the number of queries performed in user space is smaller.

Over the years, different SDN controllers have been developed. The first one, known as NOX, was created by Gude *et al.* [53] to act as a network operating system capable of allowing interaction with OpenFlow switches. Subsequently, NOX evolved into POX to allow its execution on Linux, Windows or MacOS machines. Among the different types of SDN drivers developed [54], this thesis highlights OpenDaylight (ODL), as it is implemented in OpenStack, which is used in the Open Platform for NFV [37]. This thesis uses the OPNFV, described next in this chapter and the OpenDaylight SDN controller, to implement some of the proposed security solutions.

2.2 Network Functions Virtualization

The European Telecommunications Standards Institute released Network Functions Virtualization in 2012 proposing solutions to reduce the dependency of Internet Service Providers on the manufacturers of network middleboxes [32, 55]. However, data centers can also benefit from the use of NFV solutions to increase network flexibility. Since 2013, ETSI released other proposals with different test scenarios to be implemented using NFV to solve ISP and data center problems [30, 56, 57].

Figure 2.4 illustrates the ETSI reference architecture for Network Functions Virtualization [55, 57]. The NFV Management and Orchestration (MANO) is a framework with tools to orchestrate and manage all virtual and physical resources of the Network Functions Virtualization cloud. MANO has a Virtualized Infrastructure Manager (VIM), a Virtual Network Function (VNF) Manager, and an NFV Orchestrator. The NFV VIM controls the Network Functions Virtualization Infrastructure (NFVI), which manages physical and software resources. NFVI provides computing (memory and CPU), networking, and storage resources required to deploy and run

VNFs. The VNF Manager deals with VNF instance life cycles. It creates, updates, migrates and removes VNFs. It also captures and manages event reports from the Element Management System (EMS) modules that monitor VNF resource utilization. The NFV Orchestrator globally manages all NFV cloud resources authorizing and validating each NFVI resource request, managing the creation and the life cycle of NFV instances and Network Services and controlling VNF packages. Operation Support System (OSS) and Business Support System (BSS) modules manage network inventory, service provisioning, billing, crash reports, and so on.

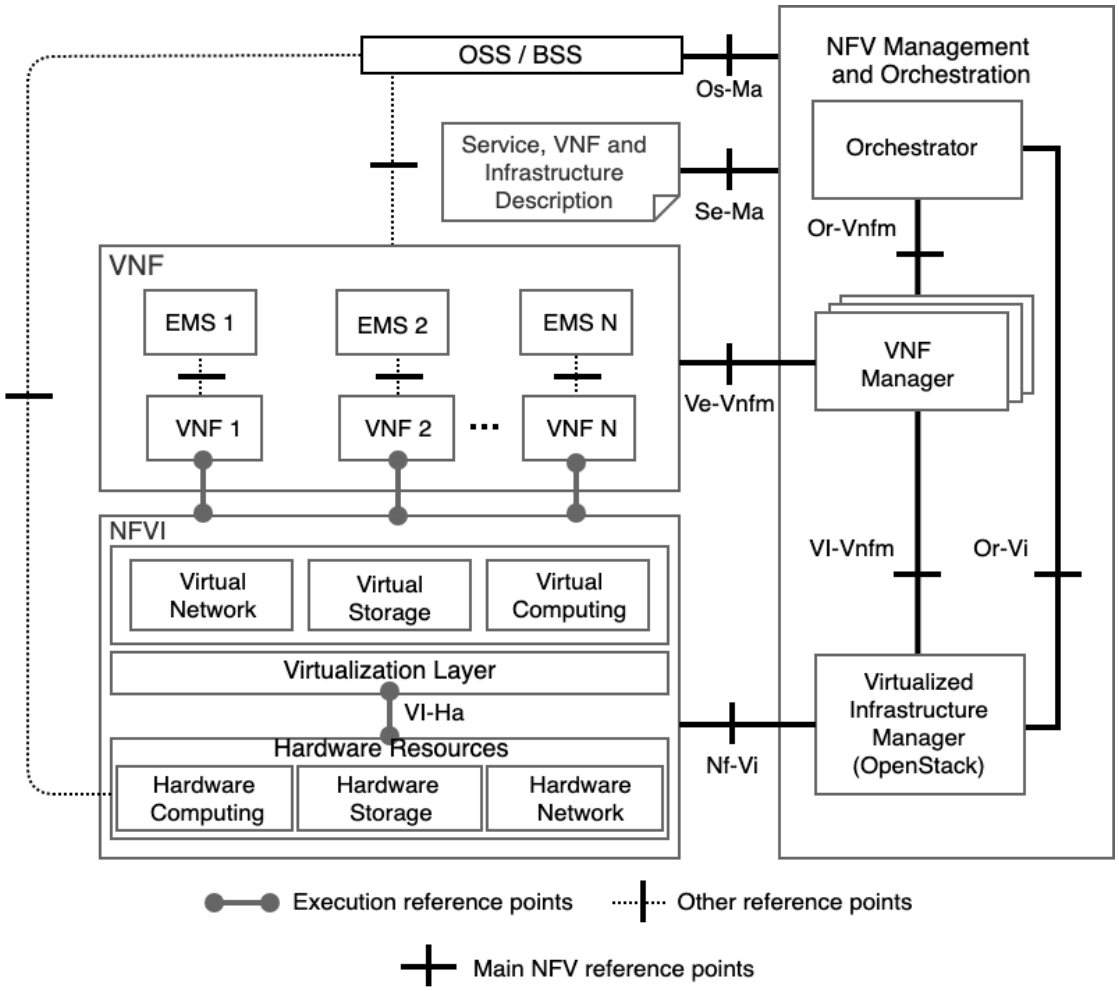

Figure 2.4: The ETSI reference NFV architecture.

Figure 2.4 also shows that ETSI reference NFV architecture proposes interaction interfaces (Nf-Vi, Vi-Ha, Vn-Nf, Ve-Vnfm, Se-Ma, Os-Ma, Or-Vnfm, Or-Vi, and Vi-Vnfm) that correspond to APIs between NFV components [30, 32, 56]. Those interaction interfaces (reference points) are named with the initials of the interconnecting systems. The Or-Vnfm interface, for instance, connects the NFV Orchestrator to the VNF Manager module to allow communication between them. The NFV Orchestrator can also interact directly with the Virtualized Infrastructure Manager through the Or-Vi interface to configure, reserve, and manage hardware resources.

Alternatively, these activities can be performed by the VNF Manager, through the Vi-Vnfm interface. The VNF Manager deploys and allocate resources and configures and monitors VNFs during its life cycle by using the Ve-Vnfm interface.

The virtualization layer serves as an abstraction tool for the physical resources provided by the NFV Infrastructure. Therefore, regardless of the hardware platform, the virtualization layer must be able to use the Vl-Ha interface to instantiate VNFs. The Vn-Nf interface has been defined to support the information traffic required for Virtual Machine creation, execution, portability, monitoring, performance measurements, and resource management. Through the Nf-Vi interface, the Virtualized Infrastructure Manager determines the way requests for configuration and allocation of resources, either sent by the VNF Manager or by the NFV Orchestrator, will be performed. Therefore, it is through the Nf-Vi interface that the MANO module communicates with the NFV Infrastructure to configure and monitor both hardware resources and the virtual resources provided by the virtualization layer.

Finally, the Os-Ma reference point interconnects the OSS and BSS modules to MANO. This happens because the applications developed in these modules need to be able to collect information from the NFV architecture for services such as registration of usage and billing, status information of installed VNFs and available physical resources, among others [32, 55].

2.2.1 The Open Platform for NFV (OPNFV)

The Open Platform for Network Functions Virtualization (OPNFV) is an open platform aimed at accelerating the introduction of new products and services related to NFV [37]. OPNFV aims at the integrated implementation of a Virtualized Infrastructure Manager and an NFV Infrastructure. Using a custom Linux named OpenStack Fuel (OPNFV manager), it is possible to manage the installation of the VIM and the NFVI components on physical servers.

The OPNFV Virtualized Infrastructure Manager (Figure 2.4) is OpenStack, which is a cloud operating system capable of virtualizing and managing resources. The OPNFV NFVI enables the deployment and management of Software-Defined Networks (virtual network in Figure 2.4) using the OpenDaylight driver. OpenStack Tacker [58] is the OPNFV module responsible for managing and orchestrating the life cycle of VNFs, as well as creating and removing Service Function Chains [59]. OpenStack Tacker links the created VNFs with Element Management Systems, which allow it to monitor each Virtual Network Function resource utilization.

Since the latest OPNFV version still does not implement the ETSI Operational Support System and Business Support System, it is not possible to perform network inventory management operations yet. Likewise, the generation of service provision-

ing reports or failure reports and billing are not yet available.

2.2.2 Service Function Chaining

RFC 7665 [34] proposes the Service Function Chaining, also known as Network Function Chaining (NFC), the new logical components illustrated in Figure 2.5, and RFC 8300 [60] proposes the Network Service Header (NSH) protocol. A Service Function Chain is an abstraction that specifies the set of Network Functions in a given chain and in which order they must be traversed by the network traffic [61]. SFC is still one of the major challenges of NFV solutions. The Network Functions positioning, for instance, has been studied by different authors because it is necessary to avoid VNF positions which lead to latencies capable of making the network operation unfeasible [59, 62, 63]. As an example, the wrong placement and ordering of VNFs may result in non-enforcement of security policies, leading to security vulnerabilities and incidents. Various chaining techniques have been implemented in NFV solutions to solve this problem [64].

A service chain can be implemented using the Network Service Header (NSH) protocol described in RFC 8300 [60]. OPNFV has introduced an experimental python plugin to support chaining with NSH encapsulation. The NSH header has a field to allow the exchange of metadata information between Network Functions of an SFC. In this way, it is possible to create classification rules with highly specific routing policies, such as routing rules based on each tenant ID. With NSH it is also possible to classify and reclassify flows that traverse multiple Network Functions.

Whenever traffic satisfies a classification rule, all packets generated by the endpoints are encapsulated with the NSH protocol and inserted into the respective chain of Network Functions. The architecture element responsible for encapsulation and routing the NSH traffic is the Service Function Forwarder (SF Forwarder) (Figure 2.5). The OPNFV SF Forwarder is implemented within the Open vSwitch, which acts as an NSH packet forwarder.

A Network Function is a "SFC aware" when it can understand, decapsulate, and encapsulate the NSH packets it receives from the SF Forwarder. An unaware SFC Network Function needs to use an SF Proxy to insert traffic into a service chain. The SF Proxy is an element capable of decapsulating the data stream before sending it to unaware VNFs and encapsulating the stream to send it to aware VNFs.

With NSH encapsulation, multiple chains can use the same VNF because it adds two new header fields into packets, the Service Path Identifier (SPI) and the Service Index (SI). Figure 2.5 illustrates that incoming packets, sent by clients, first reach the Service Function Classifier (SF Classifier) (1) that encapsulates them with NSH headers, processes the SPI field to correlate each packet to its Service Function Paths

(SFPs), and sends them to the Service Function Forwarder (SF Forwarder) (2). SFPs define the order of the VNFs each packet should traverse. SF Forwarder steers traffic through VNFs that may be aware (3 and 4) or unaware (5 to 8) of the NSH protocol. Aware VNFs decapsulate packets, execute its Network Function, encapsulate packet again decrementing the service index, and then send it back to the SF Forwarder. If the VNF is unaware of the NSH protocol, all NSH encapsulation and decapsulation steps are performed by a Service Function Proxy (SF Proxy), which removes NSH headers from each packet before sending it to unaware VNFs (6). The SF Proxy insert the NSH headers again (7) before sending the packets back to aware VNFs. The communication between physical machines happens through the remote's SF Forwarders. The steps 10 and 11 in Figure 2.5 illustrate an NSH communication between the SF Forwarder and the VNF of the remote host. When the SF forwarder identifies that there are no more VNFs ahead in the Service Chain, it sends the traffic to the SF Classifier (12). Then the SF Classifier identifies the terminator belonging to the Service Function Path (Server in Figure 2.5) and delivers the traffic to it (13). However, NSH encapsulation introduces performance overhead [65].

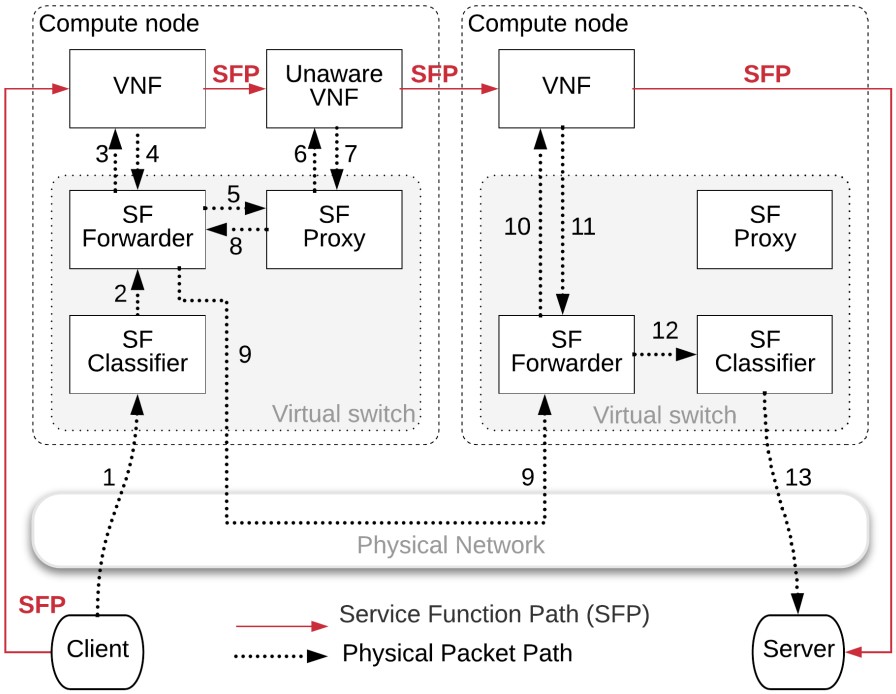

Figure 2.5: Example of an SFC including logical components involved in the execution of the SFC.

Sanz *et al.* [66] design and implement service chains using the Network Service Header protocol and evaluate its performance impact in terms of throughput, round trip time, and HTTP request rate. The evaluation followed the ETSI NFV-MANO reference architecture standard and the RFC 7665 Network Function Chaining architecture and the environmental testing was created in the OPNFV platform. Both

an Intrusion Detection and Prevention System, as well as a Firewall, are built as Virtual Security Functions. We evaluate the overhead added for Virtual Security Functions by varying the chain topology implemented, the number of virtual cores provided to VSFs, and the VSF configurations. Results show that the increase of physical link hops, as well as the sharing of resources on the same node, are factors that compromise the end-to-end delay and increase the NFV overhead. Moreover, the number of cores supplied to the Virtual Security Function affect the number of packets that each VSF is capable of processing. We conclude that the NSH encapsulation using the *vxlan_tool*, that is a python application is the main limitation for throughput because all encapsulation and decapsulation activities happen in the user space of the Operating System (OS).

NFV could use SDN, which separates the control plane of Network Functions from their data plane, to enable Service Function chaining, creating multiple virtual networks with innovative forwarding and routing logic. A service chain can be implemented by creating hierarchical routing rules in OpenFlow devices to properly drive the network traffic through the previously specified Network Functions. Thus, suitable OpenFlow rules in switch tables can create multiple Service Function Paths, enabling refined and detailed control of the routing policies [67]. In this thesis, we use this type of chaining solution.

Therefore, the sequential SFC illustrated in Figure 2.5 can be created with OpenFlow policies in SDN-enabled devices without the NSH encapsulation overhead because flow rules can steer traffic through VNFs [40, 50, 68–70]. Packets from an initial source IP address or source MAC sent to a destination IP and port could match a flow rule that drives them firstly to a VNF. Subsequently, packets from this VNF can match another flow rule that sends them to other VNFs or towards the destination host and port. Besides, an encapsulation protocol, such as MPLS, can be used for Service Function Path identification [71]. Fields of the MPLS packet header could be correlated to specific chains to determine by which VNFs traffic must follow. These solutions are an IETF SFC architectural concept without the NSH encapsulation overhead [39].

Chapter 3

Related Work

This chapter discusses the state-of-the-art and presents related work of this thesis. We divide this chapter into three main topics. Section 3.1 describes related works that implement Virtual Network Functions using Software-Defined Networking technologies. In Section 3.2 we discuss proposals to increase the performance of Virtual Network Functions and solutions designed to address the problem of shortage of TCAM entries for storing security rules. We present Service Function Chaining proposals in Section 3.3.

3.1 Virtual Network Functions Using Software-Defined Networking Technologies

Several researches propose Network Functions Virtualization security solutions using Software-Defined Networking properties [40, 68, 72–77]. Table 3.1 lists the main features of our NFV/SDN proposal versus those implemented by other researches.

Deng *et al.* [40] propose the VNGuard framework, which uses SDN and NFV to provide and manage virtualized firewalls. VNGuard components are implemented in ClickOS [72] and it defines a high-level language to simplify policy management so that users do not need to know low-level information from virtual networks to create filtering rules. However, there is a significant reduction in VNF network performance when the number of rules increases [38].

Porras *et al.* [68] create a programming language and software modules of a solution named FRESCO that simplifies the development and deployment of security services in SDN networks using NOX controllers. Network administrators can program policies to interconnect security modules, such as firewalls, IDSs, and so on. They also can insert instructions for monitoring security alerts, blocking malicious traffic, redirecting them to a dynamic quarantine or a remote reflector scanner, and

so on. FRESCO acts on SDN-enabled switches and NOX controllers to create custom OpenFlow rules but is not able to program application firewalls to block only malicious traffic without disrupting the reliable traffic of the same source IP.

Zanna *et al.* [73] show that it is possible to integrate an Intrusion Detection System with an SDN controller to detect and block Denial-of-Service (DoS) attacks. IDS Bro is configured to send a blocking flow request to the Application Programming Interface (API) of the Ryu controller to filter IPs that generate the malicious traffic. However, besides blocking malicious network traffic, the proposed solution also blocks the benign traffic that is generated by the same source IPs.

Xing *et al.* [74] propose SnortFlow, which is a Snort-based IDS that use the OpenFlow protocol to detect attacks. SnortFlow implements countermeasures against attacks through network reconfiguration. However, the proposal is limited only to the Xen hypervisor virtualization platform and the Snort agent installed in each management domain (Dom 0) can overload Xen Dom 0. Dom 0 overhead can degrade the network performance of all Virtual Machines hosted on the same physical server because they need to use the Dom 0 network drivers to access resources located on other machines in the cloud.

Table 3.1: NFV/SDN features implemented in this thesis versus related work.

Features	[40]	[68]	[73]	[74]	This thesis
Ability to detect malicious traffic		✓	✓	✓	✓
Ability to automatically block Denial-of-Service (DoS) attacks			✓	✓	✓
Ability to automatically block malware attacks			✓	✓	✓
Ability to block malware without affecting benign traffic					✓
Dynamic network reconfiguration to block malicious traffic			✓	✓	✓
Simple OpenFlow Rule Management	✓	✓			✓
API to implement security services in SDN networks (firewalls, IDSs, etc)	✓	✓	✓	✓	✓
API to deploy SDN networks in production cloud environments					✓
High performance NFV/SDN solution					✓

This thesis proposes and implements an NFV/SDN framework to create security environments by implementing OpenFlow rules into distributed FW-VSF to protect and control the network traffic of virtual machines from cloud environments. Furthermore, we also propose and implement an NFV/SDN architecture which automatically detects, and blocks malware sent for web applications. Our NFV/SDN solutions have low overhead and offer APIs and web interfaces that simplify their implementation into computing servers of production cloud environments. It is also easy to manage policies and deploy new virtual security functions because our APIs hide low-level information from the network administrators.

3.2 TCAM limitations and performance of Virtual Network Functions

The efficiency and low-cost processing of policies is a subject that has required the attention of the research community because security rules are usually stored in TCAMs that are expensive, power-hungry and have a scarce rule space [78]. Some proposals address these problems through the optimization of the TCAM rule space while other researches implement and process filtering rules in software [20, 21, 47, 52, 79, 80]. Table 3.2 lists the main characteristics of some researches that deal with the TCAM problem versus the proposals of this thesis.

Kanizo *et al.* [79] implement a framework for decomposing large flow tables into small ones to distribute the rules among various heterogeneous switches with tables of limited size. Using a positioning algorithm, Kang *et al.* [47] optimize the placement of rules into SDN hardware switches by minimizing its number since hardware switches have a limited TCAM rule space. The algorithm prioritizes the implementation of rules into SDN hardware switches according to the network traffic that traverses them. Thus, the rule placement is efficient, taking into account each switching path used by end-to-end communications. Katta *et al.* [21] propose a hybrid hardware-software switch that provides large and low-cost rule tables using a cache algorithm that places the essential rules in a TCAM and redirects the cache missed rules to software switches. The aforementioned proposals do not reduce the costs of OPEX/CAPEX since they still heavily rely on TCAMs processing. In addition, solutions that divert some of the traffic to be handled by software switches rather than specialized hardware do not implement techniques to accelerate the packet classification to compensate for performance losses caused by the non-use of TCAMs.

Maqbool *et al.* [20] develop an application named T-Flex that implements virtualized TCAMs on ToR (Top-of-Rack) switches using their onboard Central Processing

Units (CPUs) and virtual disk memories. T-Flex increases the TCAM size by performing the same way an Operating System (OS) does when it extends the amount of memory available to applications by using the virtual memory disk. Whenever an input packet does not match a rule stored in the TCAM switch, it is redirected to the switch CPU by a default lowest priority rule that redirects all "missed TCAM incoming packets" to T-Flex, which performs a lookup operation to find out a matching rule. After that, a new rule is inserted into the TCAM to accelerate subsequent packet processing. That proposal still uses expensive, power-hungry and limited size TCAMs to store security rules. Therefore, it is not a low-cost security solution and remains linked to the life cycles of network devices from different vendors.

Kourtis *et al.* [80] implement a DPI (Deep Packet Inspection) Virtual Network Function using SR-IOV (Single Root Input/output Virtualization) and DPDK (the Data Plane Development Kit) features to enhance the VNF performance. SR-IOV provides efficient allocation of low-level network resources because it enables the direct access and control of the system hardware devices. The Intel DPDK libraries make it easy to deploy network-intensive applications by implementing a packet-processing engine that uses polling mode instead of the standard interrupt mode of the Linux network stack. All packet classification and forwarding actions happen in the user space, with no copying from the kernel, for efficiently consuming CPU cycles. However, with this solution, all applications running on the operating system need to be changed to allow communication with the DPDK library and specialized Network Interface Controllers (NICs) must be used. For this reason, the implementation cost increases. Moreover, its full implementation in a production cloud would demand hardware changes on all existing physical servers.

Emmerich *et al.* [81] evaluate software switches and conclude that optimization in the kernel Operating System (OS), dedicated CPU utilization for network interfaces or techniques to forward packets without copy are essential to the implementation of high-performance virtual switches. Bonafiglia *et al.* [82] propose to use virtual switches with DPDK to improve the NFV performance. They also compare the performance of Virtual Network Functions implemented in containers versus a fully virtualized VNF when the virtual switch used by them implements DPDK libraries. Results show that Virtual Machines using switches with DPDK support have better performance than containers with dedicated processing cores.

This thesis evaluates the performance of the Iptables firewalls created as Virtual Network Functions (or Virtual Security Functions) to process the number of security policies found in the Top of Rack routers of a studied data center. Small Iptables firewall VSFs with only 4 GB of RAM and two virtual CPUs can store more filtering rules than commercial Top-of-Rack switch TCAMs. However, our analysis shows that although reducing costs and increasing flexibility the Iptables firewall VSF

overhead increases due to the rise in the number of security rules.

This thesis proposes and implements the NFV/SDN ACLFLOW security framework that solves the ToR TCAM storage problem without a significant reduction in network performance when the number of security rules rises. The ACLFLOW implements distributed firewall Virtual Security Functions, one on each physical server of a cloud environment, to protect the networks and virtual machines that those physical node hosts. Our performance evaluation shows that ACLFLOW is an NFV security solution with low overhead that does not significantly affect the network overhead and RTT found when physical servers run without our FW-VSF. We implement an algorithm to increase the performance of the proposed firewall Virtual Security Function, by dynamically prioritizing its most popular rules. The implementation of the ACLFLOW prioritization algorithm neither requires changes in applications running on the same operating system, as occurs when DPDK solutions are used nor does it need specialized hardware.

Table 3.2: Features implemented in this thesis to reduce cost by removing security rules from TCAMs versus related work.

Features	[79]	[47]	[21]	[20]	[80]	[82]	This thesis
Ability to manage ACLs of heterogeneous switches	✓						✓
Ability to prioritize most popular security rules			✓	✓			✓
Ability to deprive unused rules	✓	✓	✓	✓			✓
Dynamic firewall rule reconfiguration	✓	✓	✓	✓			✓
Do not rely on specialized hardware resources to work							✓
TCAM security rules can be moved to software switches			✓		✓	✓	✓
Do not implement security rules in expensive TCAMs					✓	✓	✓
High performance security solution	✓	✓	✓	✓	✓	✓	✓

3.3 Service Function Chaining

Different techniques to implement service chains have been proposed by researches [34, 69, 83–85]. The table 3.3 lists different implementations of SFC and its main features.

Qazi *et al.* [69] propose the policy enforcement middlebox software-defined structure named SIMPLE, which is an SDN-based policy application layer capable of directing packets through a desired sequence of Network Functions without the need to manually plan and configure routes to define by which middleboxes and routers the traffic should flow. However, SIMPLE does not provide tools for dynamically detecting and blocking malicious traffic.

Fayazbakhsh *et al.* [83] proposal implements a Service Function Chaining (SFC) encapsulation so that middleboxes can introduce labels in the packet header and pass context information to the next one in the service chain, enabling the traffic policies enforcement. Qazi *et al.* [86] also implement labels into packets that should follow through the network functions of a service chain. However, their solution increases the overhead by building tunnels between the network functions when they perform the chaining of the service functions.

Zhang *et al.* [84] do not add labels to packet headers to implement service chains. Instead, they use the OpenFlow protocol to create hierarchical routing rules with custom actions to direct packets between the Network Functions of the data plane. [84] and Callegati *et al.* [85] proposals do not employ the IETF Network Function Chaining definition [34] which proposes the packet header encapsulation using the Network Service Header (NSH) protocol, which increases the overhead significantly [35]. Callegati *et al.* [85] implement the Cloud4NFV that is a Network Functions Virtualization architecture based on four planes: infrastructure, virtual infrastructure management, orchestration, and service.

Table 3.3: Service Function Chaining proposals.

Features	[69]	[83]	[86]	[84]	[85]	[35]	This thesis
Ability to automatically block malware by creating service chains						✓	✓
Do not increase network overhead	✓			✓	✓		✓
SFC technique: New Encapsulation (NE) Tunneling (Tn) SDN solution (SDN)	SDN	NE	Tn NE	SDN	SDN	NSH NE	SDN

This thesis proposes and implements an automated and integrated solution for detecting and mitigating security attacks using SDN and NFV technologies. The proposal is compliant with the ETSI Network Functions Virtualization security management and monitoring specification [57]. It includes an NFV security controller module that receives and analyzes alerts sent by an IDS-VSF and dynamically decides whether to chain the data stream to an application firewall Virtual Security Function. This firewall provides suitable policy rules to block only the attack traffic, without harming the benign one of the users that use the same source IP. The NFV security architecture algorithm also automates the vulnerability scanning on Web applications. From scan results, it proactively inserts new policies into a VSF to mitigate and reduce the exposure time to identified threats. Further, by reducing the number of security policies in chained VSFs, the end-to-end latency of the SFC is decreased [38, 65]. Therefore, this proposed algorithm also removes policies that become unnecessary, whenever a vulnerability ceases to exist, to reduce latency and increase software-based SFC capacity.

In the next chapter, we implement and evaluate Virtual Network Functions created to replace security middleboxes. We assess and measure VNFs performance against the demands commonly encountered in production data centers.

Chapter 4

Iptables Firewall Virtual Security Function

Data centers are spread worldwide and continue to evolve nowadays. They can benefit from using virtualization to optimize CAPEX/OPEX [61] and increase network flexibility [70]. It is possible to reduce expenses related to heat dissipation, electricity consumption, and maintenance by using fewer physical machines. There are also advantages of not being tied down to particular vendors since Commercial Off-The-Shelf hardware can be used.

One of the most used topologies for data centers is the Fat-tree one [25]. This topology illustrated in Figure 4.1, has a core and pod elements containing aggregation switches, edge switches (Top-of-Racks) and servers. There are trees of routing and switching elements in that network architecture, with progressively more specialized and expensive equipment moving up the network hierarchy. Thus, the ToR is the minor cost switch/router.

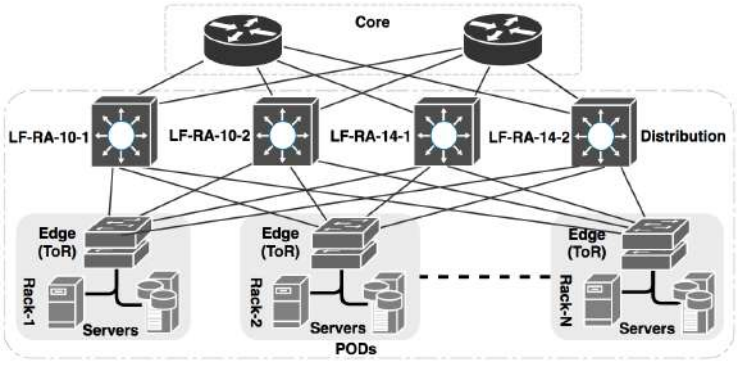

Figure 4.1: Typical topology of a data center that uses the Fat-Tree architecture.

Data center servers host different services, and many users (tenants) can have several Virtual Machines spread over multiple physical hosts and networks that can be in different racks. Moreover, those networks may need security policies to allow access to shared services like a MySQL database or a file system. Therefore, to

control the transfer of data between these data center networks, security policies need to be applied.

We made a case study in a production data center hosting a big Internet Content Provider [41]. It is a virtualized data center with nearly 10,000 VMs and 3 Tb/s dedicated bandwidth. The case study shows that this data center already reaches bandwidth usage higher than 2 Tb/s when providing web content to more than 1.4 million unique users (Figure 4.2). Access Control Lists are still the basic technique for providing security in communications between the tenant networks of this Internet Content Provider. In a significant number of these data centers, ACLs provide fine-grained and good performance security solution by using TCAMs of the ToR routers, that also act as network gateways to every server located inside the respective data center rack. Moreover, our case study reveals that there is a large amount of network traffic between racks that require high link utilization and many ACLs to allow/deny network communication within the data center studied.

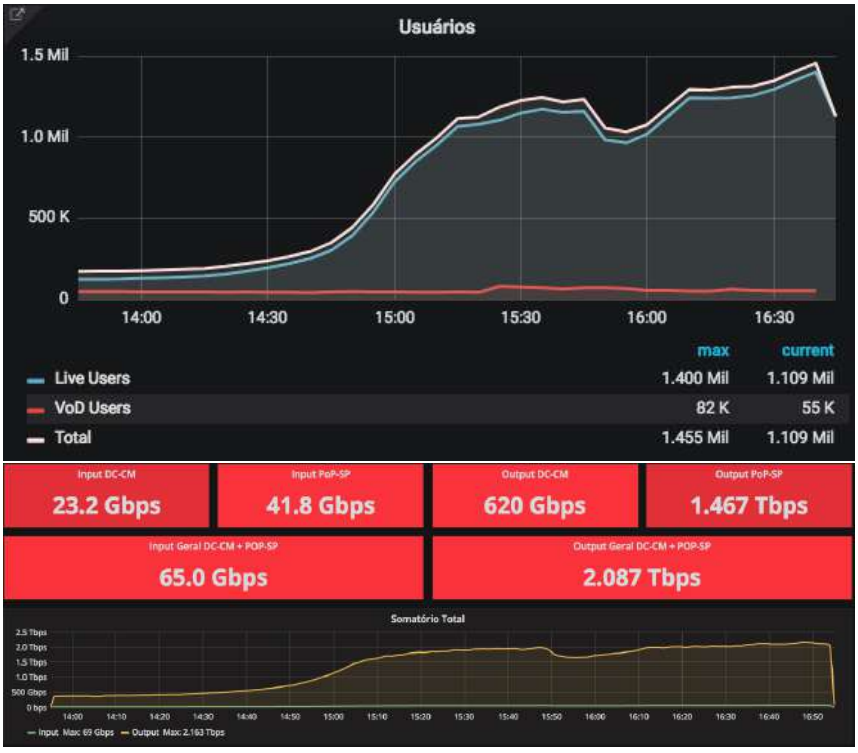

Figure 4.2: Number of unique Internet Content Provider users and the amount of network bandwidth registered during the case study.

ToRs generally use hardware optimization, such as Ternary Content Addressable Memories, to accelerate the ACL packet classification, which selectively filters traffic based on the five-tuple source/destination IP, source/destination port, and protocol [78]. TCAMs are the current state-of-the-art regarding the optimization of

packet classification in ACLs[1] hosted in ToR routers [87, 88]. They are high-speed memory tables that perform only one operation to search all memory. It is like a searching performed on hash tables, which is fast because of its low complexity, equal to O(1) [78, 89, 90]. However, although ToRs can use TCAMs to process ACLs at wire speed, such hardware solution is not designed to handle large amounts of rules. TCAMs are specialized but expensive so that ToRs usually have limited TCAM storage capacity [20, 88, 91]. Those TCAMs have limited storage capacity and are up to 400 times more expensive than RAMs. Besides, they have power consumption up to 100 times higher than RAMs [21].

Figure 4.3 displays the amount of TCAM entries reserved, in Cisco Nexus 6001 and Dell S5048F-ON, for Virtual LAN ACLs (Vacl are the same as IN-L3 ACL in Dell), for Quality of Service (QoS) operations, TCAM entries to accelerate traffic mirroring operations (Span) from one or more source ports to a destination port, and so on. Besides, Figure 4.3 also illustrates the number of TCAM entries reserved for Layer 2 (L2) and Layer 3 (L3), input and output ACLs, as well as the reserved amount to speed up routing operations (IN-L3 FIB). The number of TCAM entries for each activity is configurable by network operators. Therefore, we argue that it is necessary to minimize the number of ACLs stored in ToRs to increase the amount of TCAM resource reserved to routing packets at line speed, which is the main activity of the ToR router [92]. Moreover, Figure 4.3 shows that the first Top-of-Rack switch/router supports only 1024 network ACLs, while the second one is configured to work with up to 2048.

Cisco Nexus 6001 and Dell S5048F-ON are the two top models of ToR routers found in the case study. Although the maximum TCAM capacity of these ToR switch/routers for processing layer 3 ACLs is 4096 TCAM [93], the network operators generally reserve only up to 2048 entries for layer-3 ACLs because ToRs also use TCAMs to accelerate their routing operations, QoS, etc. Table 4.1 illustrates command lines used to set up layer 3 ACL in a Dell S5048F-ON device.

Table 4.1: Command Line to set up entries in TCAM for layer 3 ACLs

```
conf t
feature acloptimized
!
cam-acl l2acl 0 ipv4acl 9 ipv6acl 0 ipv4qos 0 l2qos 0 l2pt 0 ipmacacl 0 vman-qos 0
!
cam-ipv6 extended-prefix 1024
!
cam-acl-vlan vlanopenflow 0 vlaniscsi 0 vlanaclopt 2
end
```

[1]Cisco 6001 is a ToR with more than 60% of market share in the switch and router market according to https://www.forbes.com/sites/greatspeculations/2017/04/12/where-does-cisco-stand-in-the-ethernet-switch-market c4aa0434a

```
LF-CM-AL21-1# sh hardware profile tcam resource template
    Template   Type      State   Vacl  Ifacl   Rbacl  Qos  Span   Sup      TOTAL
-------------------------------------------------------------------------------
     default system Committed   1024   1152  1152    448    64   256      4096
-------------------------------------------------------------------------------
```

(a) Cisco Nexus 6001

```
LF-CM-AQ25-1#sh cam-usage
Stackunit|Portpipe| CAM Partition   | Total CAM   |  Used CAM   |Available CAM
========|========|================|=============|=============|=============
    1   |   0    | IN-L2 ACL      |       512   |         0   |       512
        |        | IN-L3 ACL      |      2048   |      2027*  |        21
        |        | IN-V6 ACL      |       256   |         0   |       256
        |        | IN-NLB ACL     |         0   |         0   |         0
        |        | IPMAC ACL      |         0   |         0   |         0
        |        | OUT-L2 ACL     |       206   |        12   |       194
        |        | OUT-L3 ACL     |       134   |         8   |       126
        |        | OUT-V6 ACL     |       134   |         3   |       131
        |        | IN-L3 FIB      |    161792   |      4006   |    157786
Codes: * - cam usage is above 90%.
```

(b) Dell S5048F-ON

Figure 4.3: TCAM entries reserved for speeding up routing (IN-L3 FIB), to layer 3 ACL processing (IN-L3 ACL in Dell and Vacl in Cisco), and so on of a Cisco Nexus 6001 (a) and a Dell S5048F-ON (b) Top-of-Racks.

Nevertheless, the current demand for Access Control Lists in the data center studied is about 6000 ACLs per rack, much higher than the capacity of the Top-of-Rack devices. To work around this problem, which is common in cloud computing environments, network operators generally insert policies that do not fit into TCAMs in Iptables firewalls installed on Xen Dom 0 or in tenant's Virtual Machines.

Over the years several studies have proposed techniques to reduce the overhead that packet filters impose on network performance. Qiu *et al.* [91] propose algorithms, specialized data structures and heuristics to improve the packet classification by identifying and removing useless policies to reduce the number of rules to be inspected. Thus, the search time of a matching rule decreases. Hamed *et al.* [94] show that traditional stateless Iptables firewalls perform sequentially and ordered matching search in policies so that it has a packet classification complexity equal to O(n), where n is the number of security rules. For this reason, the higher the number of rules the lower the Iptables firewall performance. Moreover, Iptables firewall is vulnerable to the Denial of Service attacks when it hosts a large number of rules [95, 96].

Hamed *et al.* [94] also implement techniques to enhance the packet classification by rejecting unwanted packets as early as possible to avoid using the default deny rule that is at the end of the sequential firewall policy set. Thus, it is possible to reduce the long path cost of matching the default deny rule to drop a packet that does not find a matching rule [95]. Most commercial firewalls implement classification enhancement techniques to speed up packet classification and prevent Denial of Service attacks. However, they are expensive [23, 97]. Table 4.2 reveals how

many security devices the data center studied bought to protect its networks and applications, and how much money was invested. They had to spend about $13 million in Top-of-Rack devices with TCAM capacity to store up to 4096 layer 3 ACLs, more than $1 million in Threat Mitigation Systems, and so on. In this way, cheaper alternative solutions should be sought.

Table 4.2: Expensive security devices for data center and campus networks

Type	Model	Price	Quantity	Network type
ACL Gateway	Dell S5048F-ON	$ 130,000	40-100	Data center network
Threat Mitigation	Arbor	$ 350,000	2-4	Data center network
Firewall	Palo Alto/Fortinet	$ 100,000	15-20	Campus network
Load Balance	F5 i5800	$ 160,000	6-8	Data center network
VPN Gateway	Palo Alto	$ 10,000	2-4	Data center network

It is possible to dynamically manage the Iptables firewalls rules considering traffic characteristics to reduce the number of inspected rules, by rearranging them. Thus, the sequential rule order can be dynamically changed according to the frequency of matching. In this type of approach, the most frequent rule, for example, can be positioned at the beginning of the rule set so that it can be quickly evaluated [87]. El-Atawythey *et al.* [96] state that a significant part of the Internet traffic matches a small subset of firewall rules that are usually in sparse database positions. Moreover, they state that many Internet firewalls still implement orderly and sequential packet classification, without prioritization techniques. Firewalls are generally configured with an explicit default deny rule in the last position of the sequential firewall policy set [96].

When a packet matches the first rule of a given Iptables firewall, the "first-match" happens. The Iptables "middle-match" happens when packets match the security rule in the "central position" of the firewall policy sequential set (the "middle-one rule"). Finally, the Iptables "last-match" occurs when a packet does not match any previous rule and needs to be dropped by the default deny rule.

El-Atawythey *et al.* [96] claim that is possible to use decision (or classification) trees to reorder firewall rule bases in order to reduce the search time when a huge number of packets matches the "middle-one rule". There are also researches proposing mechanisms to remove redundant rules from the rule base before reordering them according to the characteristics of the traffic. Gupta *et al.* [98] propose a set of off-line mechanisms to identify and remove redundant rules before to place the most frequently matched rules at the top of the firewall policy sequential set.

Gupta *et al.* [98] mechanisms also optimize the firewall performance when the number of matches in a default deny rule increases. An adaptive scheme based on traffic builds organization profiles to optimize the firewall policy set. However, the efficiency of this approach decreases when the number of overlapping rules is large, and the adaptability of the solution is low because the mechanism that evaluates most frequently rules is an offline task.

Our case study shows that when a cloud solution is implemented to optimize the data center physical resource utilization, it is expected that the number of Virtual Networks (VNs) and security policies increases. However, it is not possible to increase the TCAM storage capacity of ToRs without replacing equipment. Moreover, each more TCAM entries a ToR router has configured to dealing with ACLs fewer hardware resources (TCAM entries) the device has to accelerate routing activities [88].

On the other hand, it is possible to use the NFV paradigm to move security policies from ToRs to virtual firewalls created as Virtual Machines in commodity hardware, like an x86 server. Thus, the ToR routing operations can be increased while firewall virtual security functions deal with ACLs. From Cloud Computing techniques it is also possible to scale up virtual firewalls according to the network demand inside the Internet Content Provider. In this scenario, the security rule processing capacity ceases to depend on the size of ToR TCAMs and becomes a function of the amount of memory and CPU allocated to Virtual Network Functions. As the processing capability of a group of VMs may be higher than the capacity of a TCAM ToR, such a solution can be employed. Figure 4.4 illustrates this type of solution, where a client VM, located in the server 1, communicates with a server VM through the FW-VSF, which contains the matching rules necessary to release the data traffic. Therefore, in a simple solution, the Access Control Lists of a ToR can be moved to Virtual Security Functions by implementing a firewall such as Iptables [38, 99]. In this context, we use Iptables firewall Virtual Security Functions (Iptables FW-VSF) to process policies for access between networks.

4.1 Implementation

We implement a firewall as a VSF to process security policies inside a virtualized data center network that hosts an Internet Content Provider. The Iptables FW-VSFs are implemented as VMs and the network environment is configured to use VLANs. Three network interfaces were configured in an Iptables FW-VSF in different VLANs: one to allow communications with other networks of the laboratory, the second for allowing access to the Internet, and the third one to allow communications with other VMs created. Iptables is configured to act as a "state-

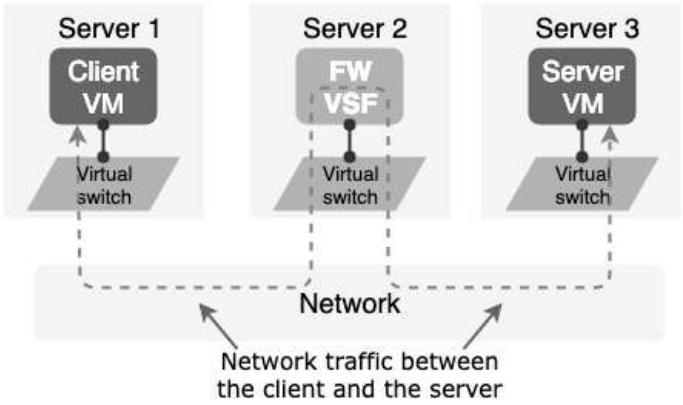

Figure 4.4: The firewall virtual security function implemented in a Virtual Machine controls communication between nodes, that may be in different data center racks.

less" packet filter. Moreover, no NAT is configured in the Iptables FW-VSF. The network default route of VMs involved in the tests is configured to point to the Iptables FW-VSF. Thus, all network traffic generated from VMs to any destination is inspected by the Iptables FW-VSF. The testbed network is also configured to ensure that all traffic return through the Iptables FW-VSF. Therefore, the Iptables FW-VSF is configured to guarantee the security perimeter of all cloud networks.

4.2 Evaluation and Results

We use the Iperf tool to send aggregate traffic from one or more generators to a receiver. To ensure greater control over the experimental scenario and to isolate external factors, all test machines have no Internet access. We use three physical servers and VMs with two virtual CPUs and 4 GB of RAM in the evaluations, to avoid network performance issues, that happen when the amount of memory and CPU allocated to VMs do not fit the size of "NUMA" nodes in the hardware. We discuss this problem in [100]. Thus, the traffic generator is a VM created on the first server, the Iptables FW-VSFs were created on the second, and the VM used as a server was created on the third physical node. Our physical servers are Intel(R) Xeon(R) CPU E5-2630L 2.00 GHz with six cores, 64 GB of RAM, and three 1 Gb/s Ethernet interfaces.

We start the tests with a single Iptables FW-VSF and a single instance of a traffic generator. After that, we need to create more instances to increase the evaluation traffic. The number of generators is equal to the amount of instantiated Iptables FW-VSFs. The receiver in all test scenarios is also a VM with two virtual CPUs and 4 GB of RAM. We use three physical servers.

The aggregate transmission rate[2] is measured by the machine(s) that generate

[2]The transmission rate is defined as the rate effectively injected into the network.

traffic. The reception rate is measured on the receiving machine, where the Iperf server is executed. The transmission rate varies according to the size of the packet Therefore, as the packet size increases, the transmission rate gets closer to the maximum value of 1 Gb/s. All results are means with 95%-confidence intervals. We omit error intervals when they are too small.

In the first experiment, we configure the firewall with a different number of rules and use different packet sizes in a 1 Gb/s UDP flow to traverse the firewall from one generator to the receiver. Since an Iptables firewall handles its rules in a sequential and orderly way, we have made all performance evaluation in the worst case, which occurs when the rule that allows the traffic is in the last position of the rule set [38]. Thus, only the last one of the firewall policy set releases the evaluation traffic.

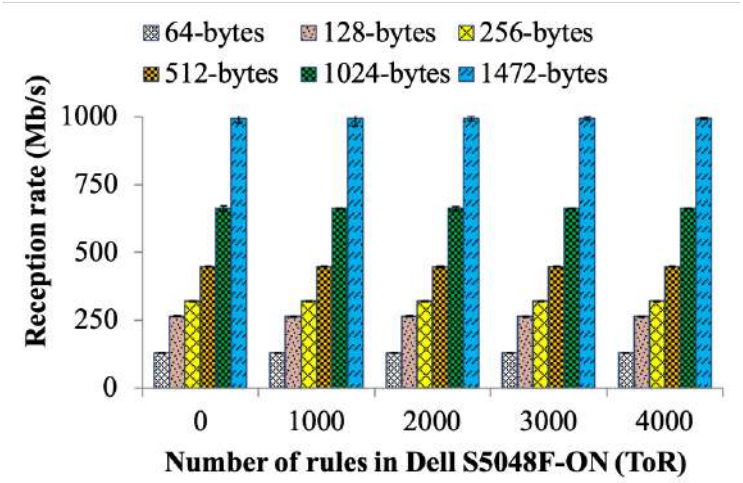

Figure 4.5: Reception rate for different UDP packet sizes and different number of rules in Dell S5048F-ON.

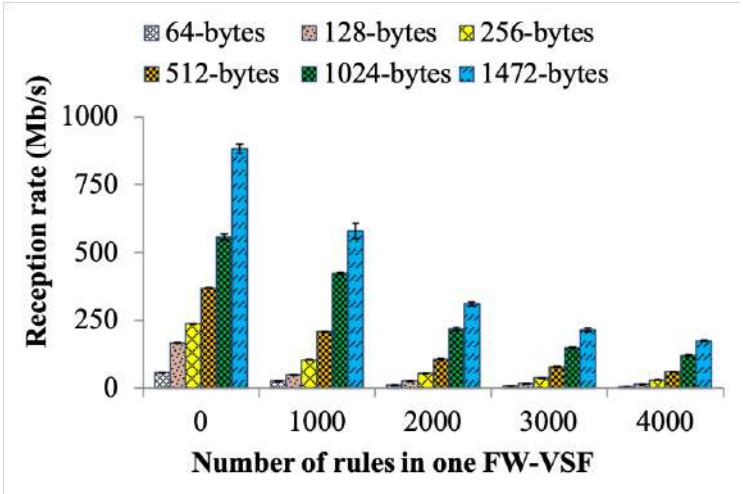

Figure 4.6: Reception rate for different UDP packet sizes and different number of rules in one Iptables FW-VSF.

The transmission rate varies according to the size of the packet. Therefore, as

the packet size increases, the transmission rate gets closer to the maximum value of 1 Gb/s.

We first evaluated the performance of a Dell S5048F-ON Top of Rack by varying the number of ACLs in its TCAM between 0 and 4000 and the packet size between 64 bytes and 1472 bytes. Figure 4.5 shows that the S5048F-ON handles the 4000 ACLs at line speed without reducing the network performance. Regarding IPtables FW-VSF, Figure 4.6 shows that, beyond the effect of the header overhead, there is a significant variation according to the number of rules. Results show that the firewall performance for more than 1000 rules is lower than the transmission rate and the reception rate significantly decreases for more than 2000 rules. The performance becomes worse as the number of rules increases. The higher the number of rules, the higher the processing time of each packet in the firewall and, hence, the lower the reception rate. Besides, the maximum receive rate achieved when using a physical machine to host the Iptables FW-VSF is slightly less than the maximum reception rate achieved by the Dell S5048F-ON.

In a second test, we transmit UDP packets at different rates, ranging from 100 to 900 Mb/s. We use 1472-byte packet size, in order to reduce the effect of the limitations of the transmission capacity and the overhead. The measured transmission rate is constant and corresponds to the rate offered by Iperf and, then, it is not presented in a figure. We omit evaluation results using the Dell S5048F-ON when transmission rates vary from 100 to 900 Mbps, as ToR handles 4000 ACLs at line speed without reducing network performance, as expected.

Figure 4.7 depicts the effect that the number of rules and the transmission rate caused in the reception rate. Iptables FW-VSF performs well for every number of rules at a 100 Mb/s transmission rate. For 300 Mb/s rate, however, the performance is good enough up to 2000 rules.

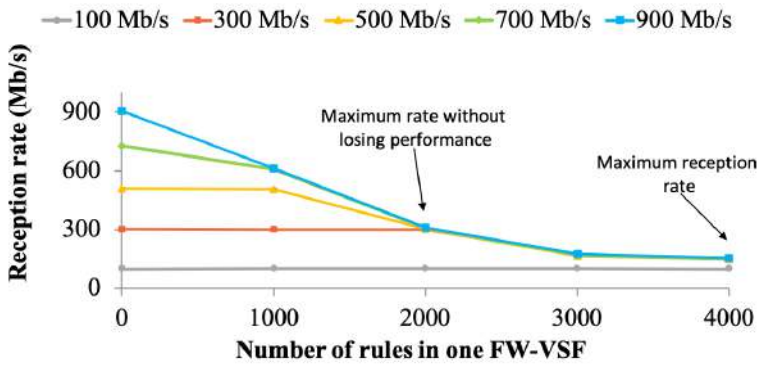

Figure 4.7: Reception rate for different UDP transmission rates and different number of rules in one Iptables FW-VSF.

Figures 4.6 and 4.7 show that a firewall function using a unique Virtual Machine (one Iptables FW-VSF) is not enough to handle the incoming firewall traffic when

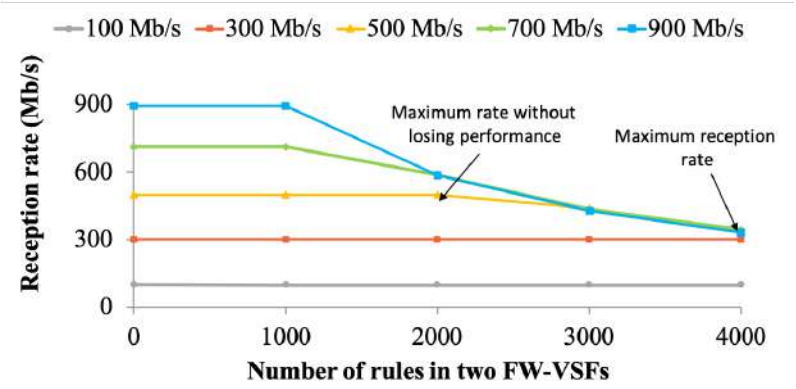

Figure 4.8: Reception rate for different UDP transmission rates and different number of rules using two Iptables FW-VSF.

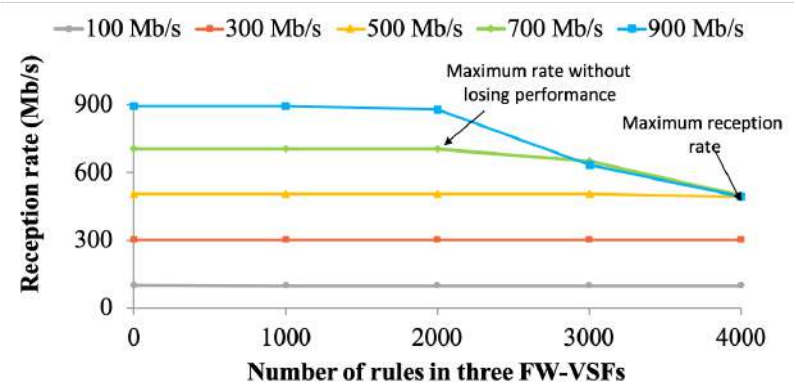

Figure 4.9: Reception rate for different UDP transmission rates and different number of rules using three Iptables FW-VSFs.

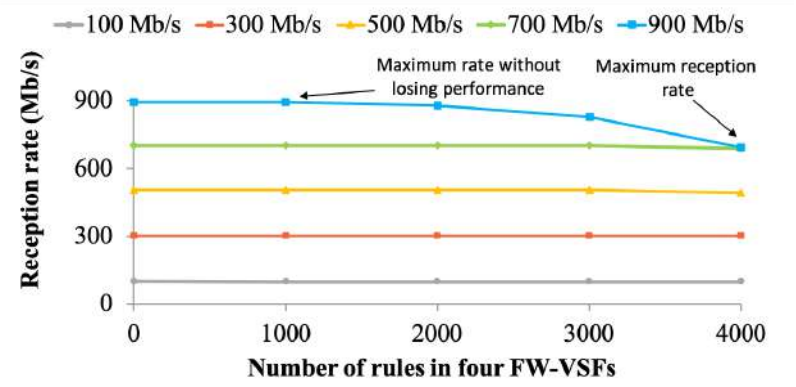

Figure 4.10: Reception rate for different UDP transmission rates and different number of rules using four Iptables FW-VSFs.

the number of security rules is equal to or higher than 1000, and the reception rate is higher than 500 Mb/s. Thus, we can use the elastic property of cloud computing, increasing the number of Iptables FW-VSFs to meet the traffic demand. Therefore, we varied the amount of Iptables FW-VSFs between 2 (Figure 4.8) and 6 (Figure 4.12) with the same specifications of the previous experiment, and we transmit the same

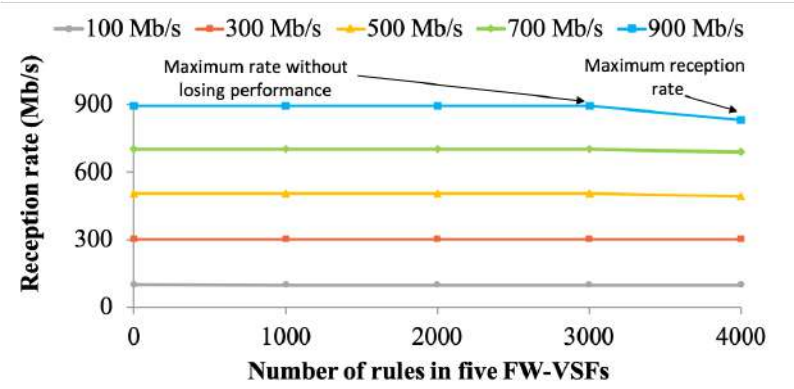

Figure 4.11: Reception rate for different UDP transmission rates and different number of rules using five Iptables FW-VSFs.

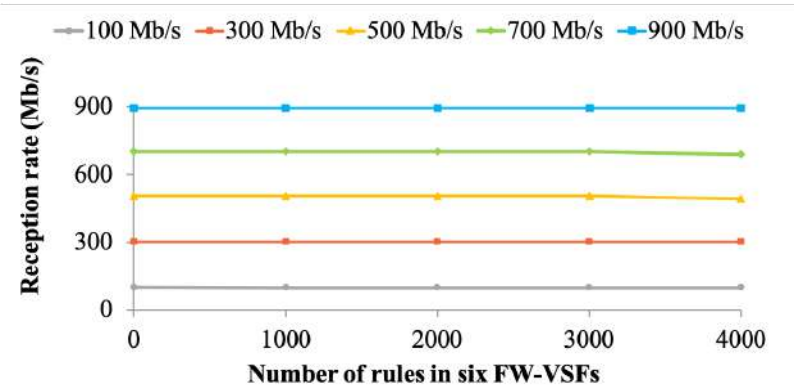

Figure 4.12: Reception rate for different UDP transmission rates and different number of rules using six Iptables FW-VSFs.

UDP packets at different rates, ranging from 100 to 900 Mb/s. In all tests, the aggregate traffic is divided between the two/six generators. Therefore, in the case of a transmission rate of 100 Mb/s and five Iptables FW-VSF, each client transmits 20 Mb/s. Also, each 20 Mb/s traverse a different Iptables FW-VSF instance.

We vary the number of ACL rules between 0[3] and 4000 and the transmission rate between 100 Mb/s and 900 Mb/s using two FW-VSFs. Figure 4.8 shows that the communication between the client and the server is satisfactory when the transmission rate is equal to or less than 300 Mb/s using two Iptables FW-VSFs. Besides, Figure 4.9 illustrates that three Iptables FW-VSFs could satisfactorily handle a 500 Mb/s traffic when the number of rules is equal to or less than 4000 and Figure 4.10 show that four could adequately handle a 700 Mb/s traffic. Figure 4.11 illustrates that five Iptables FW-VSFs could satisfactorily handle a 900 Mb/s traffic when the number of rules is equal to or less than 3000 and six are enough to sufficiently process traffic up to 900 Mb/s.

Results show that it would be necessary to use six Iptables FW-VSFs to process

[3]The Iptables FW-VSF has only one rule that allows any incoming traffic.

900 Mb/s traffic.

The tests performed here always considered the worst case when packet classification time and memory consumption are maximized. Also, virtualization allows simple scaling the number of Iptables FW-VSFs, to meet the number of ACLs and the required baud rate. For this reason, we conclude that it is possible to use firewall Virtual Security Functions to increase the number of ACLs processed in a virtualized data center. However, it is also possible to state that Iptables FW-VSFs created in Virtual Machines are subject to Denial-of-Service (DoS) attacks that exploit their worst matching case whenever the number of rules is large. This is evident in the graphs because network performance is strongly affected by the increase in the number of rules, while the rule that matches the test traffic is the last one in the Iptables FW-VSF. This problem arises from the approach used by the Iptables classification mechanisms that evaluate firewall rules in an orderly and sequential way. As a result, an attacker can overload the FW-VSF by sending several streams that will be discarded only through the standard deny rule, after analyzing a large number of rules.

In the next chapter, we detail the proposed ACLFLOW framework. ACLFLOW translates regular ACLs (source/destination IP, source/destination port, and protocol) into OpenFlow filtering rules that we insert into distributed FireWall VSFs (FW-VSFs). We also present an algorithm to improve the network performance of FW-VSFs, dynamically prioritizing the rules with the highest number of matches.

Chapter 5

ACLFLOW Security Framework

Chapter 4 shows that Iptables firewalls implemented in Virtual Machines as Virtual Security Functions demand mechanisms to improve the packet classification performance. Orderly and sequential search across multiple rules in a stateless firewall policy set is very costly when the rule base is too large. This type of research has an O(n) complexity, where n is the number of firewall rules. For this reason, the higher the number of rules the lower the Iptables FW-VSF performance. Taking advantage of SDN concepts, we propose and implement a security framework named ACLFLOW to deal with this problem by creating distributed OpenFlow firewalls as Virtual Security Functions.

In this thesis, we implement the ACLFLOW framework with OpenFlow FW-VSFs, by creating filtering rules in Open vSwitches (OVSs) configured in the proactive mode. Thus, ACLFLOW previously programs firewall rules in FW-VSFs to define which types of traffic should be allowed and which ones should be blocked. Furthermore, the OpenFlow fail secure mode is configured in ACLFLOW FW-VSFs to persist firewall rules if their communications with the SDN controller fails, preventing inappropriate traffic blocking and security failures. Additionally, ACLFLOW does not use a timeout value in the rules it creates, i.e., the rules never expire. ACLFLOW also implements an algorithm that dynamically prioritizes the most popular rules in user space to accelerate the OpenFlow pipeline and increase our firewall performance.

The algorithm implemented in ACLFLOW (Algorithm 1) dynamically prioritizes the FW-VSF rule with the highest traffic volume (most popular rule) of each algorithm execution round. Thus, it increases FW-VSF performance by reducing unnecessary searches across multiple hash tables in the user space. The algorithm deals with four priority levels to accelerate the packet classification in FW-VSF user space: lowest priority, low priority, standard priority, and high priority (Table 5.1). We create each FW-VSF with a default deny rule and assign it the lowest priority ($Pri.drop = LowestPriority$). ACLFLOW also provides a REST API so that

cloud computing tenants can create firewall rules on distributed FW-VSFs to protect their applications. These rules always receive low priority (*Pri.low*). Critical FW-VSF rules, such as those created by administrators to enable DHCP (Dynamic Host Configuration Protocol), DNS (Domain Name System), and NTP (Network Time Protocol), receive the default priority, which is higher than the low priority (*Pri.standard = Pri.low + 1*). We dynamically set up the high priority (*Pri.high*) to the rule identified as having the highest amount of traffic in each algorithm execution round. It's an online operation.

We implement two thresholds in ACLFLOW prioritization algorithm: an Upper Threshold (UT) and a Lower Threshold (LT). UT is used to identify which rules have enough traffic to be a candidate to be prioritized. LT is used to find out which rules should receive low priority because they now have low traffic volume. Therefore, filtering rules with the amount of traffic higher than UT should be prioritized and those who no longer have the highest traffic volumes can go back to the low priority. ACLFLOW prioritization module executes Algorithm 1 at each time interval *T*. Lines 6–13 and 14-24 of the algorithm show which decisions the algorithm makes based on the traffic volume of each FW-VSF security rule. Decisions also vary depending on the type of each rule.

Table 5.1: Prioritization algorithm input data.

Type	Description
rule.id	Rule id
UT	Upper Threshold
LT	Lower Threshold
R	FW-VSF filtering rule
S	Standard filtering rules created by network administrators
T	Waiting time for re-executing the algorithm
rule.idle	Time without matching in each rule
rule.pk	Last number of matches in each rule
rule.pk.count	Cumulative number of matches in each rule
Pri.drop	Default deny rule priority
Pri.low	Priority assigned to rules created by cloud computing tenants
Pri.standard	Priority assigned to rules that allow essential network services
Pri.high	Priority dynamically assigned to the most popular rule

Algorithm 1: PRIORITIZATION ALGORITHM

input : Table 5.1 values

```
1  begin
2      IdleRules ⟵ 0
3      regular.rule.ids[ ]⟵ 0
4      MisusedRules, CountRules ⟵ 0
5      highest.rule.id, highest.rule.δ ⟵ 0
6      for rule ∈ R do
7          if rule.id ∈ S then
8              CONTINUE;
9          else
10             rule.δ ⟵ (rule.pk.count - rule.pk);
11             rule.pk ⟵ (rule.pk.count);
12             CountRules ⟵ CountRules + 1;
13     end
14     for rule ∈ R do
15         if rule.idle ≤ T AND rule.δ ≥ UT AND rule.δ > highest.rule.δ then
16             highest.rule.id ⟵ rule.id;
17             highest.rule.δ ⟵ rule.δ;
18         if rule.δ ≤ LT AND rule.priority == Pri.high then
19             regular.rule.ids[ ]⟵ rule.id;
20         if rule.δ == 0 then
21             IdleRules ⟵ IdleRules + 1;
22         if rule.pk.count == 0 then
23             MisusedRules ⟵ MisusedRules + 1;
24     end
25     highest.priority ⟵ Pri.high;
26     regular.priority ⟵ Pri.low;
27     UpgradePriority(highest.rule.id, highest.priority);
28     DowngradePriority(regular.rule.ids[ ], regular.priority);
29     IdleRules ⟵ (IdleRules/CountRules);
30     RulesNeverUsed ⟵ (MisusedRules/CountRules);
31 end
```

We have decided not to automatically change the priority of the standard filtering rules created by network administrators. Therefore, the algorithm does not process its statistics (Lines 7-8). Otherwise, the algorithm calculates and records the difference between the cumulative number of matches and the last number of matches in each rule (Line 10). We calculate how many rules the prioritization algorithm processes at each time interval T (Line 12) for generating detailed network statistics.

When the time without new matches for a rule is less than or equal to the time between two executions of the algorithm (T), it evaluates if the variation in the number of packets is greater than or equal to UT (Line 15). If so, the number of packets that have matched the rule since the last analysis is considered significant and this could be the filtering rule with the highest amount of traffic. Then, the algorithm records the firewall rule ID (Line 16) and the number of matching packets since the last algorithm execution (Line 17). The identified most popular rule priority is dynamically set up to the high priority (Line 25) in all OpenFlow

FW-VSFs through an update message sent to the OpenFlow controller(s) (Line 27). Thereby, the ACLFLOW prioritization algorithm dynamically speeds up the packet classification in user space to improve the OpenFlow FW-VSFs performance.

Otherwise, if the number of packets that have matched the rule since the last analysis is less than the lower threshold (Line 18 of Algorithm 1), the traffic variation is considered small. Then, if the filtering rule analyzed still has a $Pri.high$ priority, the algorithm records its rule ID in a list (Line 19) of rules that will have their priority reset to the default value, also through a message sent to the OpenFlow controller(s) (Line 28).

The algorithm also calculates the number of inactive filtering rules since its last execution (Lines 20, 21, and 29) and the ratios of rules that never recorded packet traffic (Lines 22, 23, and 30). With this information, cloud administrators can remove long-term inactive rules, as well as identify which rules are no longer in use. They also can decrease the Upper Threshold to increase the number of candidate rules to be prioritized; so that more rules can be inserted into the most popular rule group, one rule of each algorithm execution round. On the other hand, the algorithm shows that lowering the Lower Threshold reduces the number of rules that remain prioritized in the most popular rules group.

We have developed the ACLFLOW translation module as an API that can translate security policies that evaluate the five-tuple source/destination IP, source/destination port, and protocol into OpenFlow filtering rules (with up to 12 tuples) [26, 45, 46].

Each physical cloud server of the ACLFLOW security framework has an OpenFlow FW-VSF that belongs to a security domain (Figure 5.1). The ACLFLOW security domain is a set of SDN controllers that manage multiple VSFs that can be grouped in the same rack or distributed, even across different data centers. All FW-VSFs belonging to the same security domain have the same set of firewall rules. This feature enables secure migration of Virtual Machines without the risk of traffic blocking or improper security breaches, as illustrated in Figure 5.1 by Vr3 VM. Therefore, when a Virtual Machine migrates from one physical cloud computing machine to another, in the same security domain, all firewall rules required for its operation are already replicated in the remote OpenFlow FW-VSF.

5.1 Implementation

Figure 5.1 illustrates the implemented ACLFLOW translation, management, monitoring, and prioritization modules.

ACLFLOW translation module is a python package make available on GitHub in [26], under the Apache license version 2.0. This module aims to hide of network

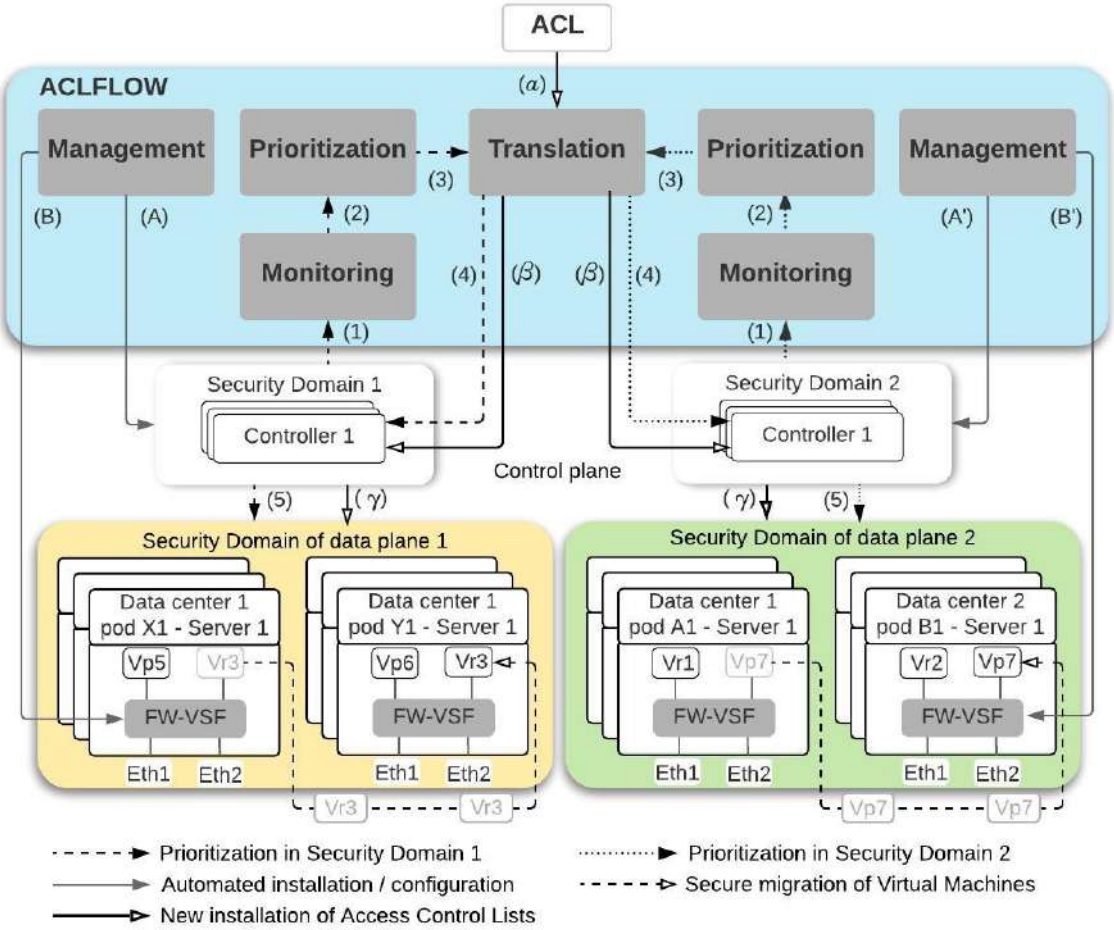

Figure 5.1: ACLFLOW security framework that installs a FireWall Virtual Security Function (FW-VSF) on each cloud computing node (server). Distributed FW-VSFs that belong to the same security group have the same firewall rules. Therefore, Virtual Machines (Vr3 and Vp7) can migrate between servers without the risk of blocking traffic or security breaches. The ACLFLOW prioritization algorithm improves FW-VSF performance. The management module speeds up the deployment of distributed FireWall Virtual Security Functions into production clouds.

administrators the details and complexities of the distribution of OpenFlow firewall rules among various FW-VSFs. Thus, regular ACLs inserted as a JavaScript Object Notation (JSON) document into the RESTful interface of the ACLFLOW translator (Figure 5.1 - (α)) are sent to SDN controller(s) (β), which insert OpenFlow filtering rules into the FW-VSFs (γ). Since only five tuples vary, the ACLFLOW translator inserts wildcard characters into the remaining OpenFlow tuples when translating regular ACLs and each OpenFlow filtering rule receives a unique identifier. That way, we can get usage statistics and manage every filtering rule installed on any FW-VSF in the ACLFLOW security framework. Figure 5.2 illustrates some OpenFlow rules we have created in Open vSwitches from production servers of the data center studied by using the ACLFLOW translation module.

The ACLFLOW management module provides an API and some Web interface

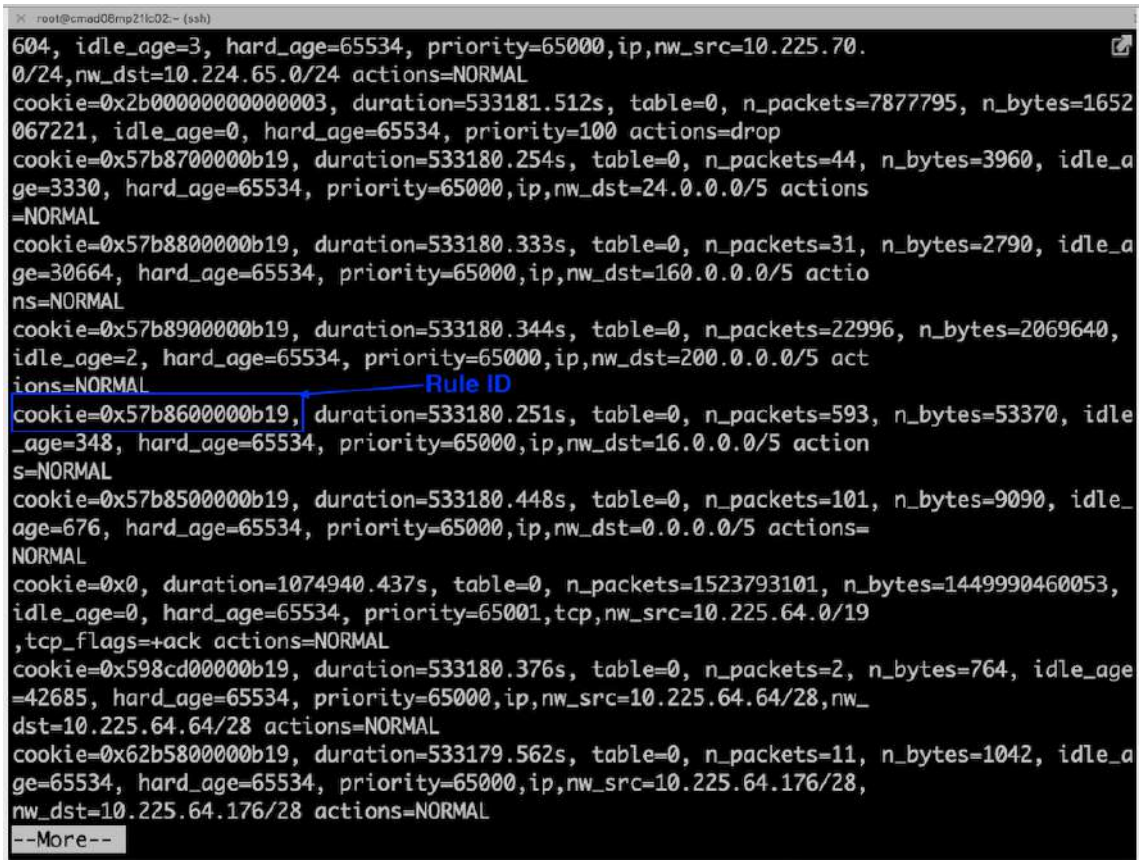

Figure 5.2: OpenFlow rules we have created in Open vSwitches from production servers in the data center studied. The ACLs were removed from the ToRs and written in the OVSs.

developed as python applications to simplifies the implementation of FW-VSFs in a cloud production environment. Figures 5.3, 5.4 and 5.5 respectively illustrate the web interface we have developed to register new FW-VSFs, the developed interface to handle multiple security ambient and its FW-VSFs, and the interface to simplify the FW-VSF operation. Through those interfaces and APIs, we automate the installation of SDN controllers (A and A' of Figure 5.1) and the configuration of OpenFlow FW-VSFs (B and B'), by reducing the setup time from about 2 hours to about 5 minutes. Thereby, cloud administrators can quickly implement OpenFlow FW-VSFs in a production environment.

Network administrators can use the ACLFLOW management northbound REST API to configure new SDN controllers and distribute FW-VSFs efficiently. The northbound API requires the IP address of the machine hosting Open vSwitch; the IP address of the Virtual Machine to install an OpenDaylight (ODL) controller, and the security domain identifier used by ACLFLOW to define which FW-VSFs should have the same rules. Network administrators can also inform the IP addresses of critical services they wish to allow into the distributed FW-VSF rules, such as DHCP, DNS, or NTP. After receiving information through its northbound API or

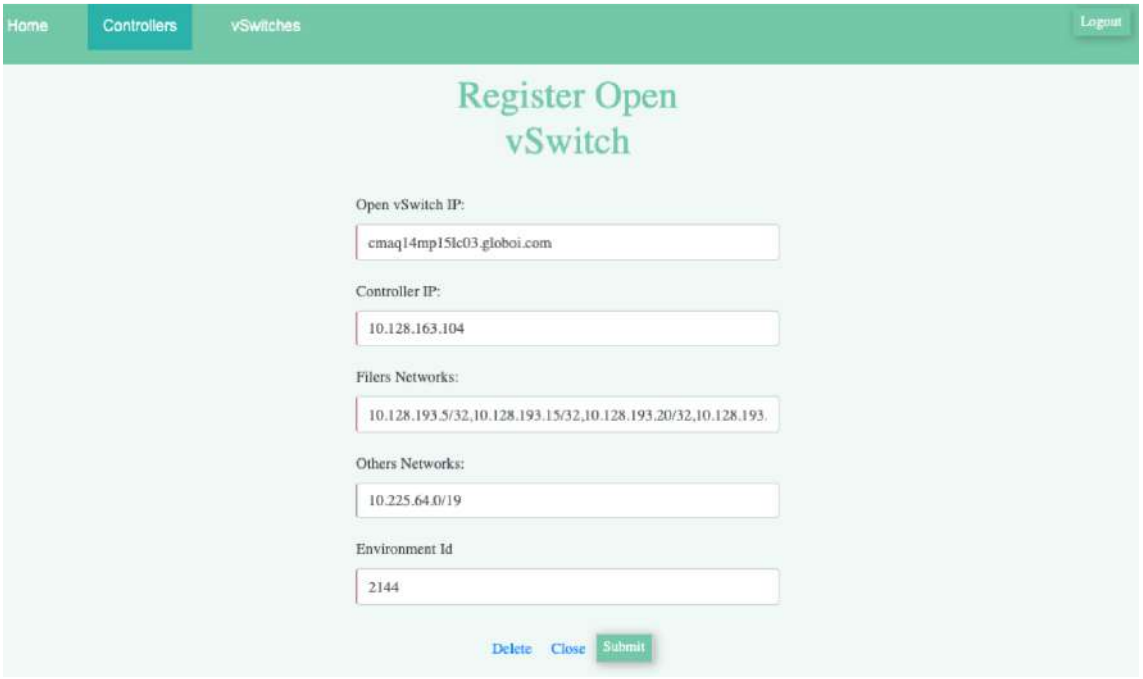

Figure 5.3: ACLFLOW management module web interface developed to register new FW-VSFs.

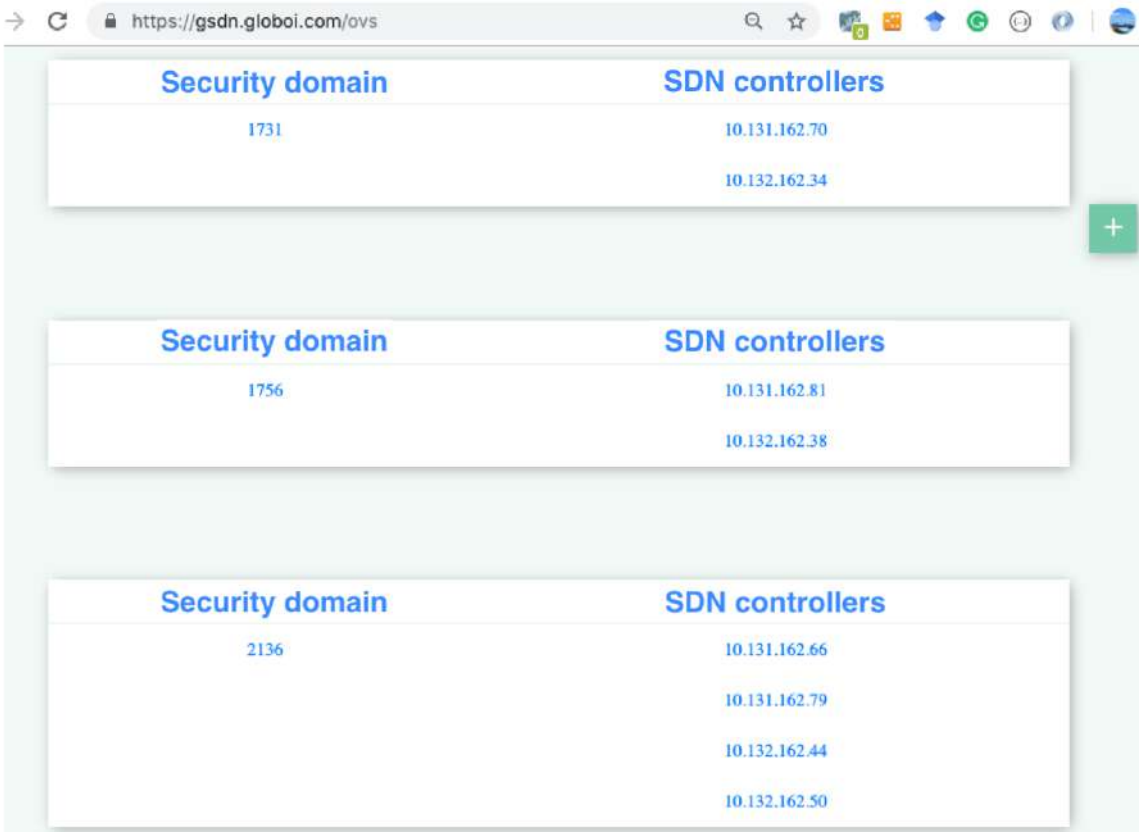

Figure 5.4: ACLFLOW management module web interface developed to handle multiple security ambient and their FW-VSFs.

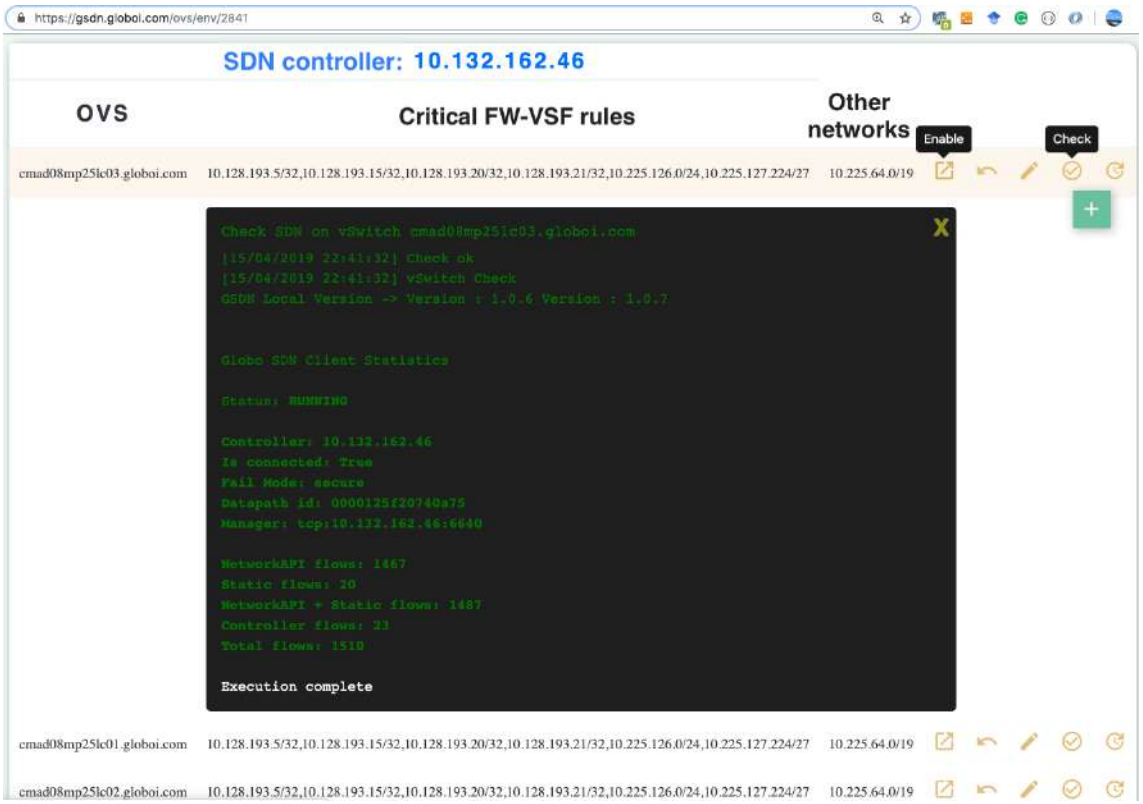

Figure 5.5: ACLFLOW management module web interface to operate each FW-VSF.

Web interface, the management module uses its southbound API to configure FW-VSFs in a proactive mode. We do that to avoid the reactive mode delays which increase and can create security problems when the number of new flows is large (the controller takes a too long time to write the OpenFlow rules in the data plane by causing packet loss) [45, 46]. Also, the management module configures the fail secure mode in FW-VSFs to persist firewall rules previously created in the data plane if communication with its SDN controller fails.

ACLFLOW monitoring module gets the unique identifier of each firewall rule (*rule.id*), the cumulative number of matches in each rule (*rule.pk.count*), and for how long each filtering rule is not used (*rule.idle*). The monitoring module collects all information from the SDN controller API ((1) in Figure 5.1) and sends them to the prioritization module (2) at each time interval T. Upon receiving statistics, ACLFLOW prioritization module performs the algorithm described in Section 5 to accelerate packet classification and improve FW-VSF performance. To update firewall rules (*rule.id*) the prioritization module sends a message to the translation module (3) informing new priorities. To prioritize a rule, it sends the highest one (*highest.priority*) priority. To remove a previously prioritized rule from the most popular rule group, it sends a regular priority (*regular.priority*). Upon receiving messages from the prioritization module, the ACLFLOW translation module up-

dates the priorities in all SDN controllers belonging to the same FW-VSF security domain (4). Then, SDN controllers update distributed FW-VSF firewall rules (5).

5.2 Evaluation and Results

The Iptables Virtual Security Function presented in [38] is a stateless firewall. Iptables classifies packets by performing linear, ordered, and sequential searches on the rule set [95]. For this reason, its performance varies according to the position of the rule that matches the traffic. Therefore, we evaluate the performance of Iptables "first-match", when the matching rule is in the first position of the Iptables policies, Iptables "middle-match", when the matching firewall rule is in the middle position and Iptables "last-match", when the matching rule is in the last position of the Iptables policies and compare their performance with that of OpenFlow FW-VSF.

We implement this proposal in the Open Platform for NFV (OPNFV). The OPNFV installation involves five machines. The OPNFV manager is a desktop, one machine is used to build the OpenStack controller node, and three others are compute nodes, where VSFs are installed. The OPNFV manager and the OPNFV controller nodes are Intel (R) Core (TM) i7-4770 CPU @ 3.40 GHz with four cores, 8 threads, 32 GB of RAM, and three 1 Gb/s Ethernet interfaces. The OpenStack computes are Intel (R) Xeon (R) E5-2650 CPU @ 2.00 GHz with eight cores, 16 threads, 64 GB of RAM, and three 1 Gb/s Ethernet interfaces. Besides, all OpenStack nodes are Ubuntu 14.04.

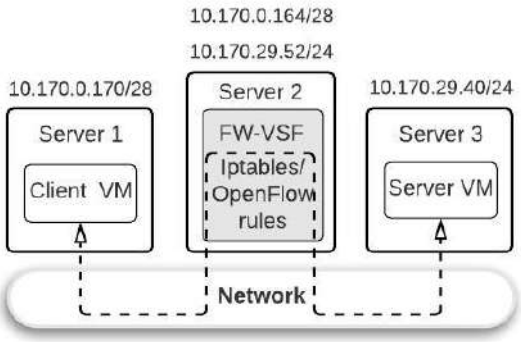

Figure 5.6: Test scenario with Virtual Security Functions that implement ACLs on commercial off-the-shelf servers with Iptables/OpenFlow rules.

We built our OpenFlow FW-VSFs inside Open vSwitches, version 2.5.1, configured to act in the proactive mode. Thus, distributed FW-VSFs receive firewall rules in advance. Only up to 32 hash tables are created inside OpenFlow FW-VSFs by the five tuples: source IP, destination IP, source port, destination port, and protocol [45, 46, 52]. Besides, no matter how many rules exist on each FW-VSF hash table, the OVS packet classifier performs only one query (an O(1) lookup) when it

checks whether the information of the packet header evaluated matches one of the stored rules. Thus, increasing the number of rules does not significantly change the number of performed queries in FW-VSF, which is always less than 32.

In our testbed, communication between a client (IP 10.170.0.170) and a server (IP 10.170.29.40) traverses the FW-VSF as illustrated in Figure 5.6. To measure network performance, we use HTTPerf, Iperf, and Netcat tools. We use HTTPerf to evaluate how much FW-VSF impacts the maximum HTTP request rate transmitted from the client to the server. In another experiment, we use Iperf tool to send UDP packets with 1472 bytes at 1 Gb/s to the server and measure the maximum throughput. We also estimate RTT using Netcat. Results are means of 15 rounds with 95% confidence intervals.

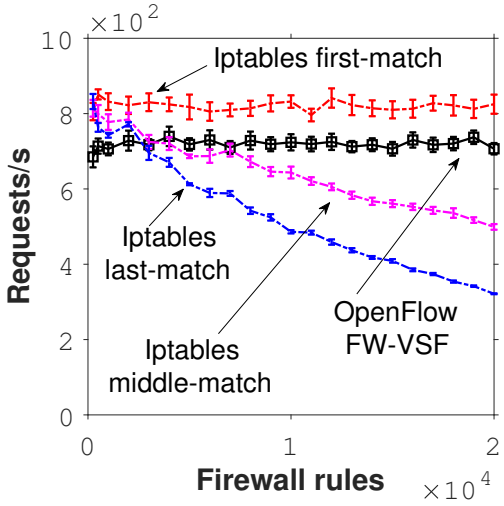

Figure 5.7: Maximum HTTP request rate for different number of rules.

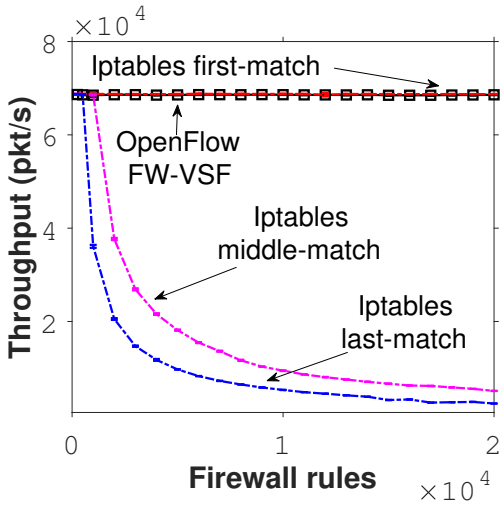

Figure 5.8: Maximum throughput for different number of rules.

We first compare the Iptables performance and OpenFlow FW-VSF without

prioritization rules (OpenFlow FW-VSF). The maximum HTTP request rate (Figure 5.7), the maximum throughput (Figure 5.8) and the Round-Trip Time (Figure 5.9) are evaluated when we vary the number of rules in the firewall from 250 to 20000. Results show better HTTP request rate (about 800 requests/s), higher throughput (about 68k pkt/s), and lower RTT (less than 3 ms) when the matching rule is in the first position of the Iptables policies (Iptables "first-match" FW-VSF). The number of rules does not significantly affect the network throughput and the HTTP request rate, because in this scenario all traffic match the first rule and the others do not need to be evaluated. Besides, RTT variation is up to only 0.8 ms when the Iptables "first-match" FW-VSF is used, and the number of rules varies between 0 and 20000. That small variation in latency is not sufficient to affect a live video application transmitting a football game via an over-the-top (OTT) streaming service, for instance.

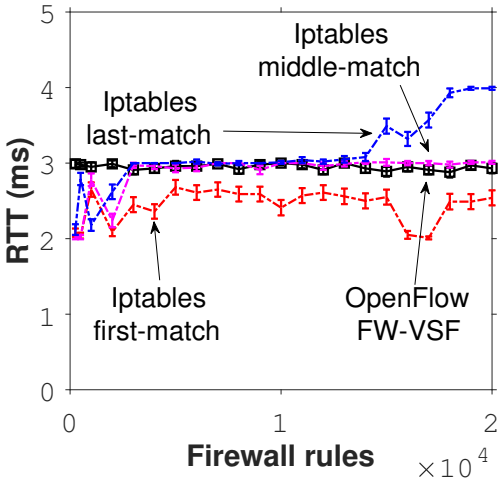

Figure 5.9: Round-trip-time for different number of rules.

RTT increases and the rate of HTTP requests and the maximum throughput decrease when the matching firewall rule is in the middle position of the rule set (Iptables middle-match). Worst results are obtained when the last Iptables firewall rule forwards the traffic (Iptables last-match). The increasing number of filtering rules causes a throughput reduction of more than 90%, the reduction in HTTP request rate is approximately 60%, and RTT increases by about 70% when the number of rules is 20000. That is the worst case for Iptables FW-VSF.

When comparing OpenFlow FW-VSF without the prioritization engine to Iptables first-match, results show similar maximum throughput values (see Figure 5.8), a slightly higher RTT (Figure 5.9) for ACLFLOW, and the maximum HTTP request rate a little bit lower than the best case for Iptables (Figure 5.7). It is essential to highlight the performance of OpenFlow FW-VSF is not affected by the variation of the number of rules.

To evaluate the efficiency of the ACLFLOW prioritization algorithm, we have deployed 22 OpenFlow FW-VSFs on 22 physical servers in a production data center [41]. These servers host more than 145 virtual machines, about 80 distinct applications, and 145 virtual networks. All OpenFlow FW-VSFs are within the same security domain. Therefore, all FW-VSFs have the same set of production filtering rules. However, each OpenFlow FW-VSF has a unique set of "matching statistics" because physical servers host distinct Virtual Machines with different traffic patterns. 2860 firewall rules are inserted on each of 22 production OpenFlow FW-VSFs, but these rules are in different positions for different FW-VSFs. We took these 22 different "traces" of "matching statistics" from those 22 physical production servers to use as a baseline to our prioritization algorithm. Thus, a different trace from the 22 productions FW-VSF is used on each one of the 22 round tests executed. We vary the number of rules in our tests, from 250 to 20000. First round with 250 rules inside the proposed firewall. Second round with 500 rules. The third one with 1000, and so on, 22 rounds until 20000 (250, 500, 1000, 2000, ..., 19000, 20000). Each round was executed for 15 times to get means with 95% confidence intervals.

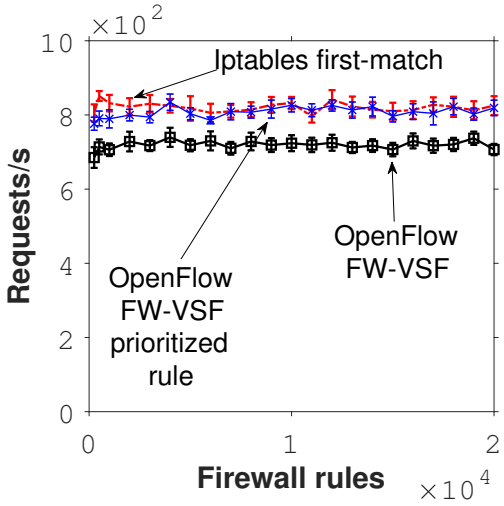

Figure 5.10: Maximum HTTP requests rate comparing the Iptables first-match performance with OpenFlow FW-VSF with and without dynamically prioritized rules.

When evaluation begins, the prioritization algorithm has matching statistics of the production "trace" with traffic records that are larger than that of the firewall rule that matches the test traffics generated by HTTPerf, Iperf, and Netcat. For this reason, the firewall rule that matches the test traffic is not prioritized immediately. After a while, the number of matches registered in the test rule becomes higher than those we have taken from the production trace to use as a baseline in the prioritization algorithm. Then, the proposed algorithm changes the priority of the

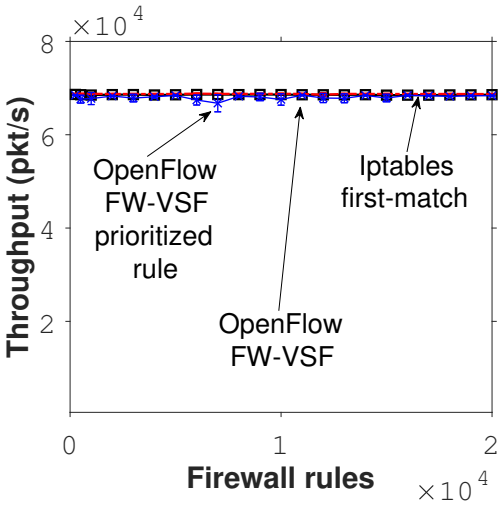

Figure 5.11: Maximum throughput comparing the Iptables first-match performance with OpenFlow FW-VSF with and without dynamically prioritized rules.

test rule because its matching statistic has become the one with the highest volume of traffic (most popular rule).

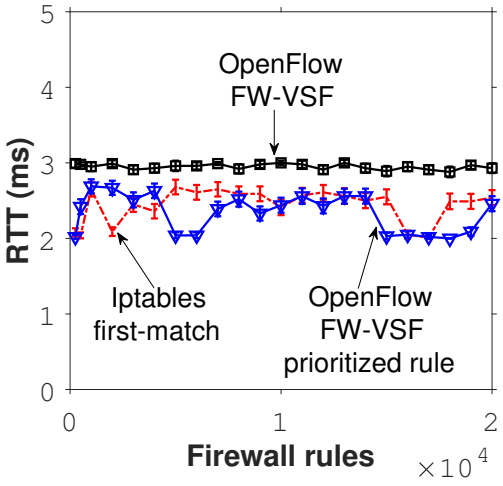

Figure 5.12: Round trip time comparing the Iptables first-match performance with OpenFlow FW-VSF with and without dynamically prioritized rules.

We choose to set up the prioritization algorithm time interval T of ACLFLOW to 5 min. Thus, we make it five times greater than the cache entries in the OpenFlow FW-VSF kernel space that is set up to 60 s, which is the standard time found in Open vSwitches. Thus, the flows prioritized by the algorithm are those that persistently presented higher traffic volume for at least 5 minutes. The Upper Threshold is configured to prioritize the firewall rules which register a variation in the number of matches greater than 2000. The Lower Threshold is used to reset the priority of rules previously prioritized (with *Pri.high*) when they have registered a variation in the number of matches higher than UT. Firewall rules with packet variation less

than 500 have their priorities reset to *Pri.low*. The priority of the default deny rule is 300, the non-privileged rules receive priority equals to 30000, and most popular rules receive the highest one priority, equal to 65001 ($Pri.high = 65001$).

We compare first-match Iptables VSFs and the proposed OpenFlow FW-VSF with and without the interaction of the prioritization algorithm. Figures 5.10, 5.11 and 5.12 show that the ACLFLOW prioritization module presents similar performance when compared with Iptables first-match. Moreover, it improves the HTTP request rate by about 15% and reduces RTT by about 25% when compared to ACLFLOW without prioritization. Performance is enhanced because the algorithm engine speeds packet classification into user space to quickly return the rule with the highest traffic volume to the OpenFlow FW-VSF kernel, where packet forwarding is fast.

Furthermore, Figure 5.12 shows that by using the prioritization algorithm, it is possible to obtain even lower RTT values than those obtained when using the Iptables "first-match". This is because, most of the time, prioritized rules are classified and forwarded into the operating system (OS) kernel. On the other hand, Iptables performs all packet classification in the Operating System user space, which is generally slower to forward traffic, when copies of packets must be sent from user space to kernel space to achieve the machine's network interfaces [101].

Chapter 6

NFV Security Architecture

In this chapter we present the proposed and implemented NFV security architecture that dynamically creates a software-based chain with a WAF Virtual Security Function to block attacks.

Figure 6.1 illustrates the ETSI NFV reference architecture [55, 57], extended with our security components, which aim to provide automatic and efficient protection against Web exploitation attacks. Our three new components are highlighted in Figure 6.1. The central element is the NFV Security Controller module, which interacts with an Intrusion Detection System and a Web Application Firewall. The IDS Virtual Security Function analyzes data traffic, detects threats, and sends alerts to the Security Controller (Figure 6.1 (a)). Upon receiving the alert and identifying the type of attack, the NFV Security Controller decides whether an update of the application firewall rules is required and if WAF-VSF should be chained to filter the detected malicious traffic. If so, the NFV Security Controller writes suitable security rules into WAF Virtual Security Function to block detected attacks (b). Then, the NFV Security Controller sets up OpenFlow rules into virtual switches to steer traffic through WAF-VSF that can block malicious traffic without stopping the benign one. The NFV Security Controller creates OpenFlow Service Function Chains managing an SDN controller that is inside the NFVI Virtual Network module (c). We choose to create OpenFlow SFCs because so far, the SFC with NSH is still an experimental option in the OPNFV that presents a deficient network performance, as we show in [35]. Subsequently, the NFV Security Controller performs a periodic vulnerability scan on the attacked Web application to see if it can reduce the number of security policies implemented in WAF-VSF to increase its performance.

6.1 Implementation

We implement the proposed NFV security architecture in the same Open Platform for NFV (OPNFV) described in Chapter 5. OPNFV virtual network module

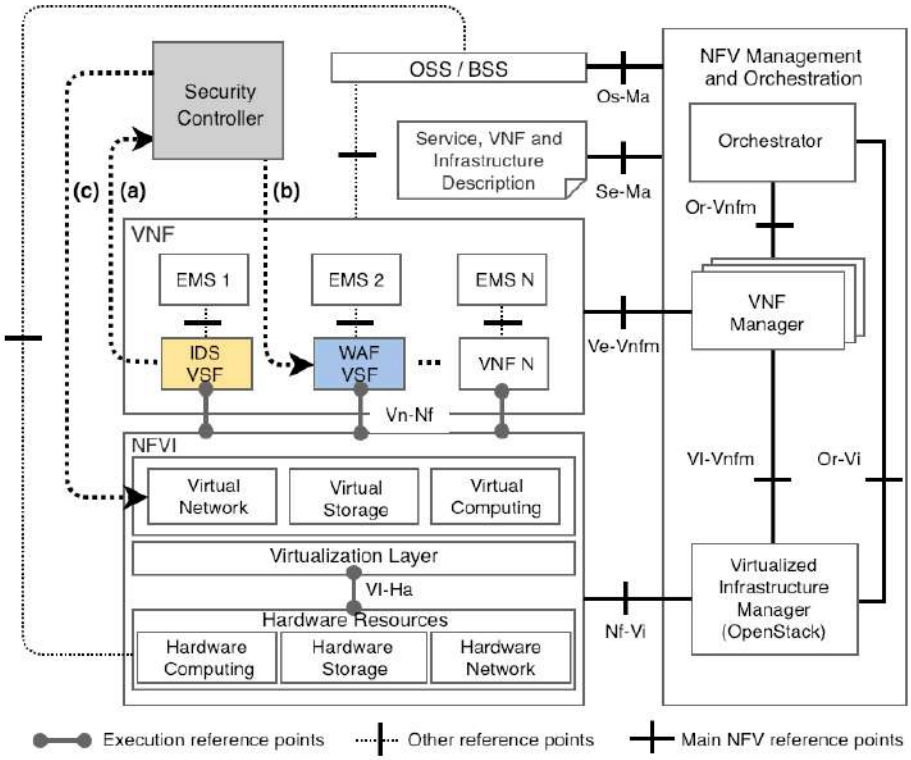

Figure 6.1: Extended Network Functions Virtualization architecture. The proposed security components are colored. Main interactions of the proposed security controller are a) receiving alarms; b) automatic installation of customized security rules in Virtualized Security Function application firewall, and c) firewall chaining to block malicious traffic.

uses OpenDaylight (ODL) [102] as an SDN controller, virtual switches are Open vSwitches [52], and OpenStack [103, 104] acts as NFV VIM, managing all Network Function Virtualization Infrastructure physical resources. We used three OpenStack computing nodes to allow testing with VSFs built on different physical servers and one network node and built the NFV Security Controller. Thus, the WAF Virtual Security Function and the IDS Virtual Security Function run on different computing machines, one component per node. Each compute node has an Open vSwitch configured as an OpenFlow data plane of the SDN controller OpenDaylight, installed in the OpenStack network node. Figure 6.2 details each proposed security module and its interactions when they block malicious traffic sent to a vulnerable Web application.

We implement the proposed Intrusion Detection System Virtual Security Function (IDS-VSF) with threat detection and threat notification mechanisms (Figure 6.2). To build the threat detection IDS-VSF, we use the open source IDS Bro [105], which we configure to monitor network traffic passively and to search for suspicious activity. Thus, it can analyze network traffic to find events with patterns classified as malicious, such as XSS, SQL Injection and LFI. We use the

TAP technique[1] to send a copy of each network packet to the IDS-VSF. IDS-VSF receives copies of each network packet without being chained to the data stream. For this reason, the IDS-VSF does not overhead the end-to-end communication of the software-based security chain.

In Figure 6.2, Interaction (1) shows that the Intrusion Detection System receives the traffic it analyses from the Open vSwitch through its TAP network interface. IDS-VSF threat notification is a Python code that we designed, extending the Bro code, to send HTTP messages with threat alerts to the NFV Security Controller (2). Each message has the IP address that generates the malicious traffic, the destination IP address, the Uniform Resource Locator (URL) related to the attack, the Uniform Resource Identifier (URI) used, and the type of threat detected.

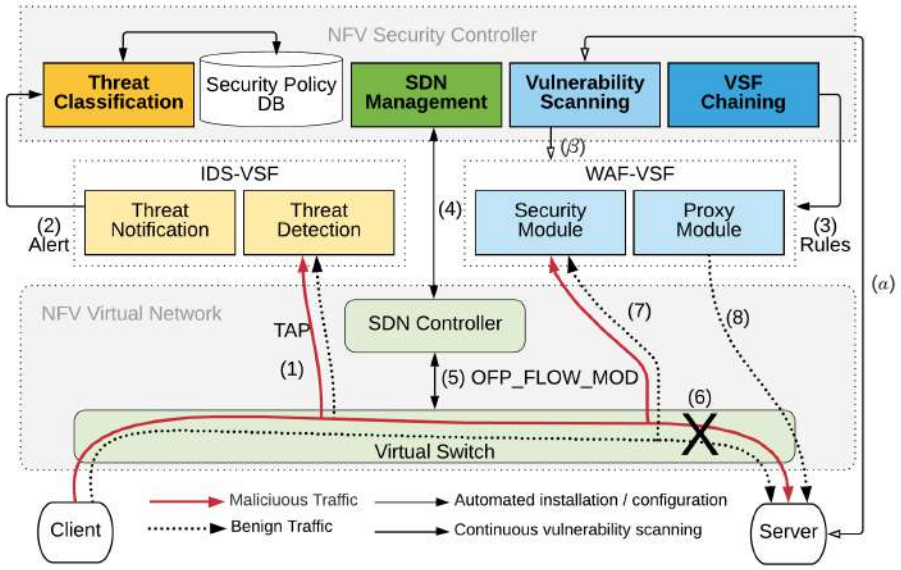

Figure 6.2: NFV Security Controller interactions with Virtual Security Functions and NFV Virtual Network.

We developed the NFV Security Controller with a Threat Classification, a VSF Chaining, a Vulnerability Scanning, and an SDN Management engines. They are all python applications. Besides, it also has a Security Policy Database (DB) that we build to store threat alerts types, in our case the OWASP top 10 most critical Web application security risks [106], and each suitable security rule able to block them. We use MongoDB [107] to create this database and our security rules are a set of generic attack detection rules for use with Mod Security. The NFV Security Controller executes the proposed Algorithm 2 to handle packets when malicious activity is detected. Upon receiving alerts from IDS-VSF Threat Notification, the Threat Classification engine of the NFV security controller validates if malicious

[1]TAP network operation type sends a copy of each network interface event to a configured system, which usually monitors and analyzes the captured data.

traffic is a Denial of Service (Ψ) attack (Algorithm 2, Line 2). If so, the NFV Security Controller automatically sends OpenFlow rules to the OpenDaylight controller ((4) in Figure 6.2) to filter all network traffic generated by the identified source IP address (Line 3).

Unknown vulnerability exploitation is a zero-day attack which has no available fixes. Therefore, every time the NFV Security Controller Threat Classification receives an unknown vulnerability exploitation alert (Line 4) it sends HTTP messages to the OpenDaylight REST API to block all network traffic generated by the identified source IP address to the vulnerable application IP (Line 5). Thus, the NFV security architecture automatically blocks traffic when it is not able to write a countermeasure rule within the WAF Virtual Security Function. This happens when there is no appropriate rule in the Security Policy Database to mitigate malicious traffic. Security network operators can register new security rules on the Security Controller Policy Database at any time.

When IDS-VSF detects known vulnerability exploitation (Line 6), the Threat Classification engine can find suitable rules on the Security Policy Database to block the malicious activity. Then, it triggers the NFV Security Controller VSF Chaining mechanism that automatically configures policies inside the WAF Virtual Security Function (3) to mitigate the detected attack. Then, through the SDN controller OpenDaylight (4) the NFV Security Controller inserts redirection flows into OVS (5), so that all traffic previously chained directly to the vulnerable site (6) is routed first through the WAF Virtual Security Function (7), which filters malicious traffic without blocking benign HTTP requests (Lines 7-9). After that, the NFV Security Controller starts its vulnerability scanning module (Lines 11-15) to periodic assess the application attacked (Interaction (α)). If it founds a new vulnerability, NFV Security Controller adds policies on WAF-VSF Security Module (Line 13). Thus, we can discover vulnerabilities and mitigate the threats before an attacker could exploit the weaknesses. Furthermore, if a known vulnerability once identified no longer exists, the NFV Security Controller removes this unnecessary policy from WAF-VSF Security Module (Line 15). The NFV Security Controller also records all attacks received (Line 10).

We use Mod Security that is a widely deployed open source WAF in the world, used by more than one million websites [108] to create our WAF-VSF Security Module. We implement our Proxy Module by using the NGINX, that is the second most used web server in the Internet [109]. The NFV Security Controller automatically chains and configures the WAF-VSF Security Module to remove malicious traffic and the WAF-VSF Proxy Module to act as a "reverse proxy" that redirects all benign HTTP requests it receives to the vulnerable Web server (8). In this way, the WAF Virtual Security Function removes malicious traffic without blocking benign

HTTP requests generated from the same source address.

Algorithm 2: Security Attack Treatment

```
input : δ={srcIP,dstIP,URL,URI,type}: Detected attack;
        Φ: Known vulnerability exploitation;
        WAF-VSF: The WAF Virtual Security Function;
        Ψ: Denial of service attacks;
1  begin
2      if δ.type ∈ Ψ then
3          Filter security attack(δ.srcIP, ∗);
4      else if δ.type ∉ Φ then
5          Filter security attack(δ.srcIP, δ.dstIP);
6      else
7          edit_waf-vsf_rules(δ.URL, δ.URI, δ.type);
8          create_chain(WAF-VSF);
9          divert_traffic(δ.srcIP, δ.dstIP);
10         Register(δ);
11         while Φ ∈ (URL ‖ URI) do
12             if Φ ∉ WAF-VSF then
13                 write_policies(WAF-VSF(security_module));
14         end
15         remove_policies(WAF-VSF(security_module))
16 end
```

6.2 Evaluation and Results

We install and configure a honeypot, which is a tool able to emulate vulnerable applications and operating systems [110]. The primary purpose of this security tool is to serve as a vulnerable Web server to be exploited and compromised. So, new types or trends of attacks can be discovered through careful analysis of the behavior and actions of the attacker. In our evaluation, a python script sends malicious and benign HTTP requests from a Web client to the vulnerable Web server for a fixed time interval, in which 220 benign and 340 malicious HTTP requests are sent on average. The benign HTTP requests access the main page of the vulnerable Web site[2] whereas the malicious HTTP requests (malware) aim to exploit ten different vulnerabilities in the honeypot. Besides, there are no preconfigured policies on the WAF-VSF Security Module and no previously URL redirection policies on its Proxy Module. Moreover, the Policy Database of the NFV Security Controller stores all suitable rules to prevent the malicious HTTP requests sent to the Web server.

Table 6.1 illustrates some security attack types we use in the evaluation. The HTTP request of Line 1 aims to get the encrypted password, IDs, and names of all registered users of a vulnerable operating system hosting the exposed Web site. Line 2 shows an HTTP request used to insert a file named "9zseqh" on the Web

[2]We use the curl command to send several GET HTTP requests to the vulnerable website.

Table 6.1: Web application attack examples

HTTP Method	Malicious HTTP request	HTTP Response Status Code
GET	http://mysite/?rHPbc8c=../../../../etc/passwd	200 OK
PUT	http://mysite/oRnQL file:9zseqh	201 OK
GET	http://mysite/?XzuFzsw=SELECT TOP 3 * FROM adminusers	200 OK
GET	http://mysite/?rHPbc8c=ps -aux	403 Forbidden

server. This file could be a malware able to execute malicious code to enslave or damage its operating system. Line 3 corresponds to an SQL Injection attack to get all information from three registered users in the "adminuser" table of the vulnerable Web site database. We can also inject a malicious JavaScript to redirect users to phishing sites or to modify the visited HTML. Line 4 describes an attack to list all running processes on the vulnerable Web server operating system. Other attacks from the OWASP top 10 most critical Web application security risks are sent.

Using our architecture, we obtain different responses depending on the attack. Malicious HTTP requests described in Lines 1, 2, and 3 are successful because they receive 2xx (OK) HTTP response status codes. It happens when a network security system does not block malicious traffic sent to a vulnerable application. On the other hand, the HTTP response status code of Line 4 indicates that the malicious request is unsuccessful, which means that our architecture was able to prevent this attack.

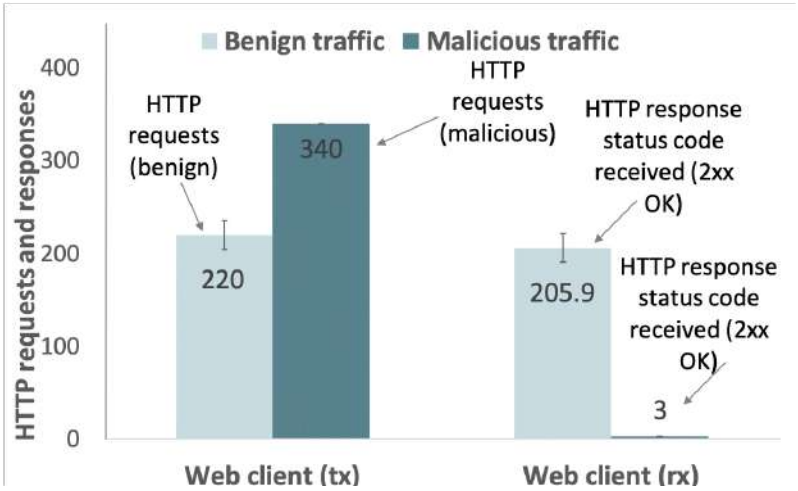

Figure 6.3: Web client benign and malicious HTTP requests (rx) and responses (tx).

We install the Open Platform for NFV on four commodity servers: one network node and three computing nodes. They are Ubuntu 14.04.5 LTS, KVM is the virtualization platform, and OpenStack is the Virtualized Infrastructure Manager. One

computing node and the OpenStack network node are Intel (R) Core i7-4770 CPU @ 3.40 GHz eight-core processors, with 32 GB of RAM and three 1 Gb/s Ethernet interfaces. Another computing node is an Intel (R) Core i7-2660 CPU @ 3.40 GHz eight-core processor, with 32 GB of RAM and three 1 Gb/s Ethernet interfaces. Finally, the third computing node has two Intel (R) Xeon (R) CPUs E5-2650 v2 @ 2.60 GHz sixteen cores, 32 GB of RAM, and two 1 Gb/s interfaces. The proposed NFV Security Controller, the WAF Virtual Security Function, the IDS-VSF, the Security Policy Database, the vulnerable Web server, and the Web client are VMs with two virtual CPUs and 4 GB of RAM. Evaluations are averages of 10 rounds with 95% confidence intervals. Some confidence intervals are not displayed because they are too small.

We first send malicious and benign HTTP requests to the vulnerable Web server without enabling the proposed security modules on the Network Functions Virtualization architecture and, as expected, all malicious HTTP requests successfully receive 2xx (OK) status codes responses. Then we evaluate the efficiency of our architecture.

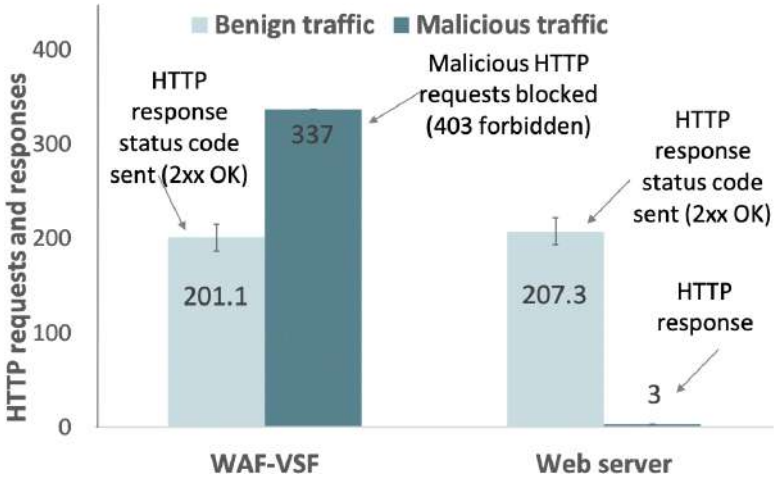

Figure 6.4: Malicious HTTP requests blocked by WAF Virtual Security Function. Number of successful attacks, benign HTTP requests and successful HTTP responses when WAF-VSF is automatically chained.

Figure 6.3 shows that Web client test script generates an average of 340 malicious HTTP requests and 220 benign ones (see HTTP requests (malicious) and HTTP requests (benign) by the Web client (tx)). As soon as the Web client generates the malicious traffic, the IDS-VSF rapidly identifies malicious requests and sends an alert message to NFV Security Controller, which automatically diverts all traffic to the WAF-VSF by applying OpenFlow rules on virtual switches. Moreover, the NFV Security Controller configures security rules into the WAF-VSF to filter the identified malicious traffic and configures the WAF-VSF Proxy Module.

Figure 6.4 shows that the vulnerable Web server sends back three successful HTTP response status codes to malicious requests (there is an average delay of about 6 seconds related to this procedure). Therefore, only three attacks reach the vulnerable Web server and are successful (see HTTP response status code received (2xx OK) by the Web client in Figure 6.3). Evaluation results indicate that there are network outages when the Security Controller makes changes in the OVS rules to redirect the Web client flow packets to chain the WAF Virtual Security Function. OVS starts to redirect all packets with Web client source IP and Web server destination IP to the WAF Virtual Security Function. During this operation, packet losses happen, like those when a router in a packet-switching network fails. For this reason, only 207.3 (see HTTP response status code sent (2xx OK) by the Web server in Figure 6.4) of the 220 benign HTTP requests (see HTTP requests (benign) sent by Web client in Figure 6.3) reach the Web server without problems.

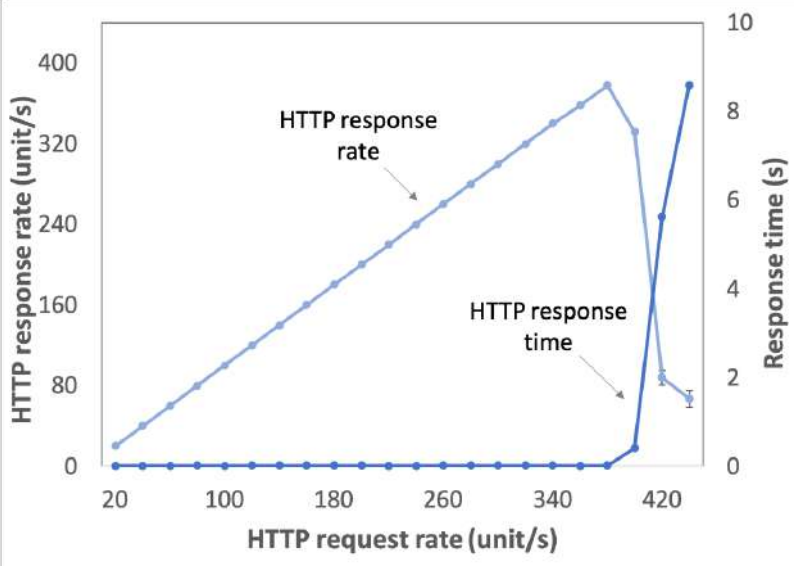

Figure 6.5: Web application performance when WAF-VSF is not chained.

Likewise, 205.9 (see HTTP response status code received (2xx OK) by the Web client in Figure 6.3) of the 207.3 HTTP response status code sent by Web server reach the Web client successfully. Moreover, the WAF Virtual Security Function chained to the data stream blocks 337 malicious HTTP requests (see Malicious HTTP requests blocked (403 forbidden) by WAF-VSF in Figure 6.4) and its reverse proxy handles 201.1 (see HTTP response status code sent (2xx OK) by WAF-VSF in Figure 6.4) benign HTTP requests from the 220 made by the Web client (see HTTP requests (benign) in Figure 6.3). There was an average of 12.7 benign HTTP requests lost between the client and the vulnerable server (220 - 207.3). The WAF-VSF proxy module handles 201.1 of 207.3 benign HTTP responses sent from server to the Web client approximately; so six of these HTTP responses do not go through the WAF-VSF. Besides, the amount of benign HTTP responses lost between the server and

the Web client is on average equal to 1.4 (207.3 - 205.9). Therefore, the results show that the proposed NFV security architecture is efficient since it dynamically blocks 99.12% of the generated attacks and 93.59% of the benign traffic is not affected when it is dynamically chained with the WAF Virtual Security Function.

To evaluate the performance of our architecture, we vary the number of HTTP requests per second sent to the Web server. The purpose is to verify how the WAF Virtual Security Function chaining affects the HTTP response rate and the Web site response time. During the experiment, WAF-VSF dynamically receives ten suitable rules because we generate ten different attack types. We use HTTPerf to generate HTTP requests. This application generates and maintains sustainable rates of HTTP requests to overload a Web server. Thus, it is possible to define the TCP connection rate per second, how many HTTP requests should be performed on each connection and the maximum number of HTTP requests that a Web client must generate. It is also possible to increase the TCP connection rate per second during a test by setting an initial rate, increment value, and a maximum TCP connection rate.

We use Auto bench [111] to automate HTTPerf and it is configured to vary the TCP connection rate between 10 and 220, increasing 10 connections per second. We send two persistent HTTP requests per each TCP connection established with the Web server. Therefore, the HTTP request rate varies between 20 and a maximum value of 440 requests per second that is the maximum capacity of our test Web server. We use these values because several preliminary tests showed that this is a limit to postpone the significant increase in the HTTP response time, which is the time elapsed between requesting and receiving an object, to the maximum. Results are means with 95% confidence intervals. Some confidence intervals are not shown in the figures because they are narrow.

Figure 6.5 shows that the maximum HTTP request rate supported by the Web server, without a significant increase in the response time, is equal to 380 requests per second. From this value, the response time increases considerably, going from approximately zero to about 9 s when the Web client attempts to send 440 HTTP requests per second to the server.

Figure 6.6 shows that the WAF Virtual Security Function chaining reduces the Web server HTTP response rate and there is a significant increase in response time from about 300 requests per second. The rising number of security policies in the WAF Virtual Security Function causes a reduction in performance. Therefore, from 300 HTTP requests per second, the response time increases considerably, reaching about 8 s. Thus, the automatically WAF Virtual Security Function chaining reduces connection rate and HTTP requests per second by approximately 21% when we have ten policies inside the firewall. However, it is possible to use cloud computing

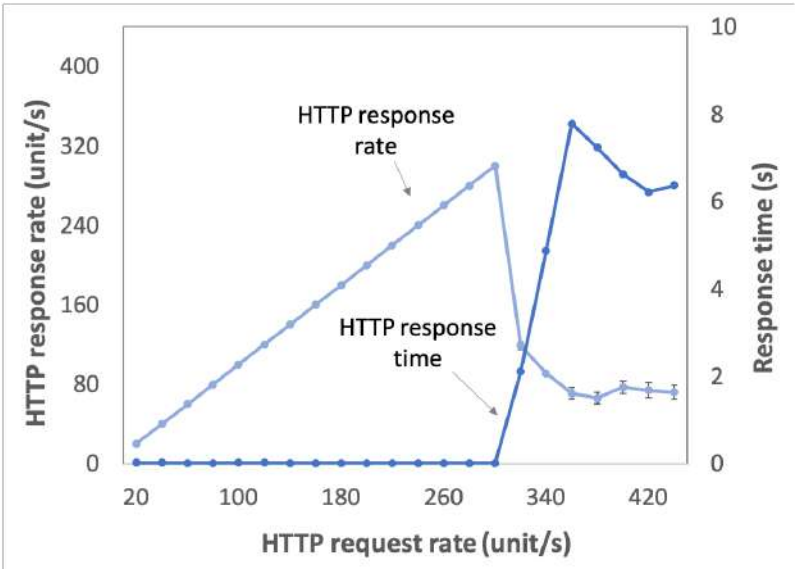

Figure 6.6: Web application performance when WAF-VSF is chained to the Web server.

scalability to increase the capacity of Virtual Network Functions. We can scale up horizontally by increasing the number of Virtual Security Functions or vertically by enhancing memory, CPU or disk resources of the Virtual Security Function to meet the demand growth.

Chapter 7

Conclusion

In this work, we present proposals regarding the problems of benign traffic blocking and performance of security NFV solutions. We analyze the use of Virtual Network Functions to build network security solutions able to dynamically steer malicious traffic through security software-based Service Chains to block exploitation attacks without stopping the benign traffic. Furthermore, this work presents a framework that implements distributed firewall Virtual Network Functions in commodity machines, as an alternative to using TCAMs or specialized security middleboxes.

We implement a virtual firewall named Iptables FW-VSF that can handle a typical number of access policies currently found in the Top-of-Rack routers of the studied data center. Performance results show that one instance of the Iptables FW-VSF is not enough to handle 300 Mb/s of incoming traffic or higher when the number of rules is greater than 2000, but six Iptables FW-VSF instances are enough to sufficiently process traffic up to 900 Mb/s in firewalls with 4000 rules. According to the tests, it is viable to remove access control lists from data centers Top-of-rack routers and process them using firewall Virtual Security Functions built into VMs, because they can scale up to meet the required network performance.

We also propose a security framework named ACLFLOW, which simplifies and automates the installation of distributed OpenFlow firewall Virtual Security Functions (FW-VSFs). ACLFLOW is an NFV/SDN security framework that creates and manages distributed OpenFlow FW-VSFs as an alternative to using router TCAMs or specialized security middleboxes to control traffic from Virtual Machines in a Cloud Computing environment. We implement a translation, management, monitoring, and a prioritization module in ACLFLOW. ACLFLOW translation module is a northbound python API that translates security policies that evaluate the five-tuple source/destination IP, source/destination port, and protocol into OpenFlow filtering rules (with up to 12 tuples). ACLFLOW management module speeds up the deployment of distributed FireWall Virtual Security Functions into production

clouds. ACLFLOW monitoring module collects information from SDN controller APIs and sends them to the prioritization module. Finally, ACLFLOW prioritization module performs an algorithm that accelerates packet classification and improves FW-VSF performance.

FW-VSF has maximum throughput up to 90% higher, HTTP request rate up to 50% better, and RTT up to 70% less than that of a stateless Iptables VSF when the last rule forwards the traffic and the number of rules is 20000. Its prioritization algorithm dynamically improves the HTTP request rate of the most popular rule by about 15% and reduces RTT by approximately 25% when compared with FW-VSF without prioritization.

NFV security architecture is a cost-effective proposal that applies countermeasures against vulnerability exploitation attacks. NFV security architecture central element is the NFV Security Controller module, which interacts with an Intrusion Detection System and a Web Application Firewall. The proposed IDS Virtual Security Function sends alerts to the Security Controller that decides whether an update of the application firewall rules is required and if WAF-VSF should be chained to filter the detected malicious traffic. NFV Security Controller sets up OpenFlow rules into virtual switches to steer traffic through WAF-VSF that can block malicious traffic without stopping the benign one. We choose to build our security architecture creating OpenFlow SFCs to avoid the NSH encapsulation performance overhead. Furthermore, NFV Security Controller has a module that performs periodic vulnerability scan on vulnerable applications to reduce the time between found a vulnerability and mitigate the threat. Besides it automatically removes rules from WAF to reduce the number of security policies implemented and increase its performance when vulnerabilities cease to exist. NFV security architecture dynamically chains an efficient WAF Virtual Security Function that filters 99.12% of malicious HTTP requests without significantly affecting the benign traffic from the same source IP. It allows the creation of dynamic and personalized software-based service chains to meet the demands of specific applications and has a vulnerability scanning module to identify application flaws and automatically apply countermeasures to mitigate and reduce the exposure time to identified threats.

7.1 Future Work

As future work, we want to integrate the NFV security architecture with a continuous deployment tool and extend ACLFLOW management module to optimize the consumption of computing resources by efficiently deploying distributed controllers into the security framework. The dynamic deployment and removal of Virtual Network Functions, depending on the results of security scans, falls into a problem

of optimizing the location of VNFs. For this reason, this type of study must be performed to identify constraints and to develop algorithms that minimize resource consumption, setup times, network delays, and so on. Software-based Service Function Chaining (SFCs) with customized Virtual Network Functions can meet specific user or applications constraints. However, techniques to optimize the performance of software-based service chains in Network Function Virtualization (NFV) are mandatory because they can introduce performance overhead. Thus, we also want to investigate techniques to build shorter asymmetrical bidirectional service chains, by removing virtual network functions from the outbound pathway to reduce the NFV overhead and delivery good performance software-based service chains.

Bibliography

[1] ZANELLA, A., BUI, N., CASTELLANI, A., VANGELISTA, L., et al. "Internet of Things for Smart Cities", *IEEE Internet of Things Journal*, v. 1, n. 1, pp. 22–32, February 2014.

[2] ACCENTURE, PONEMON, I. *Cost of Cyber Crime Study - Insights on the Security Investiments that make a Diference*. Technical report, Accenture, 2017.

[3] AKAMAI. *State of the Internet - Connectivity Report - Additional Resources*. Technical report, Akamai, 2017.

[4] ANDERSON, R., BARTON, C., BÖHME, R., CLAYTON, R., et al. "Measuring the cost of cybercrime". In: *The economics of information security and privacy*, Springer, pp. 265–300, Bohinj, Slovenia, 2013.

[5] MAURICIO, L. A. F., RUBINSTEIN, M. G. "Avaliação experimental das plataformas Xen, KVM e OpenFlow no roteamento de pacotes". In: *XXXIII Simpósio Brasileiro de Telecomunicações (SBRT)*, Juiz de Fora, MG, 2015.

[6] LIN, C.-H., PAO, H.-K., LIAO, J.-W. "Efficient Dynamic Malware Analysis Using Virtual Time Control Mechanics", *Computers & Security*, v. 73, pp. 359–373, 2018.

[7] ALJAWARNEH, S. A., ALAWNEH, A., JARADAT, R. "Cloud security engineering: Early stages of SDLC", *Future Generation Computer Systems*, v. 74, pp. 385 – 392, 2017. ISSN: 0167-739X. doi: https://doi.proxy.ufrj.br/10.1016/j.future.2016.10.005. Available in: <`http://www.sciencedirect.com/science/article/pii/S0167739X16303788`>.

[8] CHEN, Y., MAO, Y., CUI, L., LENG, S., et al. "A Two Layer Model of Malware Propagation in a Search Engine Context". In: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 21–26. IEEE, 2018.

[9] THE NEW YORK TIMES. "Facebook Security Breach Exposes Accounts of 50 Million Users". https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html, 2018.

[10] THE GUARDIAN. "PlayStation Network Hackers Access Data of 77 Million Users". https://bit.ly/2MhfYUp, 2011.

[11] FORBES. "Ashley Madison Hack Data Reveals Interesting Statistics". https://bit.ly/2Fyg5cB, 2015.

[12] DATALOSSDB. "Data Breach QuickView: 2015 Data Breach Trends". https://blog.datalossdb.org/, 2016.

[13] CHOU, T. "Security Threats on Cloud Computing Vulnerabilities", *International Journal of Computer Science & Information Technology*, v. 5, n. 3, pp. 79, 2013.

[14] BERTINO, E., ISLAM, N. "Botnets and Internet of Things Security", *Computer*, v. 50, n. 2, pp. 76–79, 2017.

[15] HP INC. "HP Study Reveals 70 Percent of Internet of Things Devices Vulnerable to Attack: IoT devices averaged 25 vulnerabilities per product, indicating expanding attack surface for adversaries". http://www8.hp.com/us/en/hp-news/press-release.html?id=1744676#.WcFPV9OGMWp, August 2014.

[16] MTIBAA, A., HARRAS, K. A., ALNUWEIRI, H. "From Botnets to Mobibots: A Novel Malicious Communication Paradigm for Mobile Botnets", *IEEE Communications Magazine*, v. 53, n. 8, pp. 61–67, 2015.

[17] MALWARETECH. "Mapping Mirai: A Botnet Case Study". https://www.malwaretech.com/2016/10/mapping-mirai-a-botnet-case-study.html, June 2017.

[18] NGO, D.-M., PHAM-QUOC, C., NGOC THINH, T. "An Efficient High-Throughput and Low-Latency SYN Flood Defender for High-Speed Networks", *Security and Communication Networks*, v. 2018, pp. 14, 2018. ISSN: 9562801. doi: 10.1155/2018/9562801. Available in: <https://doi.org/10.1155/2018/9562801>.

[19] LIN, C. C., HUNG, J. Y., LIN, W. Z., LO, C. P., et al. "7.4 A 256b-wordlength ReRAM-based TCAM with 1ns search-time and 14x improvement in wordlength-energyefficiency-density product using 2.5 T1R cell".

In: *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 136–137, Jan 2016. doi: 10.1109/ISSCC.2016.7417944.

[20] MAQBOOL, Q., AYUB, S., ZULFIQAR, J., SHAFI, A. "Virtual TCAM for Data Center switches". In: *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp. 61–66, Nov 2015. doi: 10.1109/NFV-SDN.2015.7387407.

[21] KATTA, N., ALIPOURFARD, O., REXFORD, J., WALKER, D. "Infinite CacheFlow in Software-defined Networks". In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pp. 175–180, New York, NY, USA, 2014. ACM. ISBN: 978-1-4503-2989-7. doi: 10.1145/2620728.2620734. Available in: <`http://doi.acm.org/10.1145/2620728.2620734`>.

[22] ARBOR. "Arbor Networks DDoS Protection". `https://www.arbornetworks.com/`, May 2017. Acessado: 15/05/2017.

[23] PALO ALTO. "VM-Series 7.1 Deployment Guide". `https://www.paloaltonetworks.com/documentation/80/virtualization/virtualization`, April 2017. Acessado: 14/04/2017.

[24] TAKABI, H., JOSHI, J. B. D., AHN, G. J. "Security and Privacy Challenges in Cloud Computing Environments", *IEEE Security Privacy*, v. 8, n. 6, pp. 24–31, Nov 2010. ISSN: 1540-7993. doi: 10.1109/MSP.2010.186.

[25] COUTO, R. S., CAMPISTA, M. E. M., COSTA, L. H. M. "A reliability analysis of datacenter topologies". In: *IEEE Global Communications Conference (GLOBECOM)*, pp. 1890–1895, 2012.

[26] CUNHA, H. B. L., MAURICIO, L. A. F., CESARIO, M. V. G., BRILHANTE, E., et al. "GloboNetworkAPI". `https://github.com/globocom/GloboNetworkAPI`, November 2017.

[27] SHAIKH, F. B., HAIDER, S. "Security Threats in Cloud Computing". In: *2011 International Conference for Internet Technology and Secured Transactions*, pp. 214–219, 2011.

[28] PADILHA, R., PEDONE, F. "Confidentiality in the Cloud", *IEEE Security Privacy*, v. 13, n. 1, pp. 57–60, 2015.

[29] MIJUMBI, R., SERRAT, J., GORRICHO, J. L., BOUTEN, N., et al. "Network Function Virtualization: State-of-the-Art and Research Challenges",

IEEE Communications Surveys Tutorials, v. 18, n. 1, pp. 236–262, First quarter 2016. ISSN: 1553-877X. doi: 10.1109/COMST.2015.2477041.

[30] ETSI GS NFV 001 V1.1.1. "Network Functions Virtualisation (NFV); Use Cases". 2013. Available in: <https://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_nfv001v010101p.pdf>. ETSI Ind. Spec. Group (ISG).

[31] FIROOZJAEI, M. D., JEONG, J. P., KO, H., KIM, H. "Security challenges with network functions virtualization", *Future Generation Computer Systems*, v. 67, pp. 315 – 324, 2017. ISSN: 0167-739X. doi: https://doi.proxy.ufrj.br/10.1016/j.future.2016.07.002. Available in: <http://www.sciencedirect.com/science/article/pii/S0167739X16302321>.

[32] HAN, B., GOPALAKRISHNAN, V., JI, L., LEE, S. "Network function virtualization: Challenges and opportunities for innovations", *IEEE Communications Magazine*, v. 53, n. 2, pp. 90–97, February 2015. ISSN: 0163-6804.

[33] LIN, Y.-D., LIN, P.-C., YEH, C.-H., WANG, Y.-C., et al. "An Extended SDN Architecture for Network Function Virtualization with a Case Study on Intrusion Prevention", *IEEE Network*, v. 29, n. 3, pp. 48–53, 2015.

[34] HALPERN, J., PIGNATARO, C. *Service Function Chaining (SFC) Architecture*. RFC 7665, RFC Editor, October 2015. Available in: <http://www.rfc-editor.org/rfc/rfc7665.txt>. http://www.rfc-editor.org/rfc/rfc7665.txt.

[35] SANZ, I. J., ALVARENGA, I. D., ANDREONI LOPEZ, M., MAURICIO, L. A. F., et al. "Uma Avaliação de Desempenho de Segurança Definida por Software através de Cadeias de Funções de Rede". In: *XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg 2017*, 2017.

[36] REXFORD, J., DOVROLIS, C. "Future Internet Architecture: clean-slate versus evolutionary research", *Communications of the ACM*, v. 53, n. 9, pp. 36–40, 2010.

[37] OPNFV. "Open Platform for NFV". https://www.opnfv.org/, 2017.

[38] MAURICIO, L. A. F., RUBINSTEIN, M. G., DUARTE, O. C. M. B. "Proposing and evaluating the performance of a firewall implemented as a virtualized network function". In: *2016 7th International Conference on the Network of the Future (NOF)*, pp. 1–3, Nov 2016. doi: 10.1109/NOF.2016.7810127.

[39] MAURICIO, L. A. F., RUBINSTEIN, M. G., DUARTE, O. C. M. B. "ACLFLOW: An NFV/SDN Security Framework for Provisioning and Managing Access Control Lists". In: *2018 9th International Conference on the Network of the Future (NOF)*, pp. 44–51, Nov 2018. doi: 10.1109/NOF.2018.8598136.

[40] DENG, J., HU, H., LI, H., PAN, Z., et al. "VNGuard: An NFV/SDN Combination Framework for Provisioning and Managing Virtual Firewalls". In: *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp. 107–114, 2015.

[41] MAURICIO, L. A. F. "Globo.com Uses OpenDaylight Due to Its Community Size, Flexibility, and Well-defined APIs". `https://www.opendaylight.org/use-cases-and-users/user-stories/globo-com`, December 2018.

[42] MAURICIO, L. A. F., ALVARENGA, I. D., RUBINSTEIN, M. G., DUARTE, O. C. M. B. "Uma Arquitetura de Virtualização de Funções de Rede para Proteção Automática e Eficiente contra Ataques". In: *Anais do XXII Workshop de Gerência e Operação de Redes e Serviços (WGRS - SBRC 2017)*, v. 22, Porto Alegre, RS, Brasil, 2017. SBC. Available in: <`http://portaldeconteudo.sbc.org.br/index.php/wgrs/article/view/2598`>.

[43] PATTARANANTAKUL, M., HE, R., MEDDAHI, A., ZHANG, Z. "SecMANO: Towards Network Functions Virtualization (NFV) Based Security MANagement and Orchestration". In: *IEEE Trustcom/BigDataSE/ISPA*, pp. 598–605, 2016.

[44] FERNANDEZ, M. P. "Comparing OpenFlow Controller Paradigms Scalability: Reactive and Proactive". In: *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 1009–1016, March 2013. doi: 10.1109/AINA.2013.113.

[45] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., et al. "OpenFlow: Enabling Innovation in Campus Networks", *SIGCOMM Comput. Commun. Rev.*, v. 38, n. 2, pp. 69–74, March 2008. ISSN: 0146-4833. doi: 10.1145/1355734.1355746.

[46] SHELLY, N., JACKSON, E. J., KOPONEN, T., MCKEOWN, N., et al. "Flow Caching for High Entropy Packet Fields", *SIGCOMM Comput. Commun. Rev.*, v. 44, n. 4, August 2014. ISSN: 0146-4833. doi: 10.1145/2740070.2620755.

[47] KANG, N., LIU, Z., REXFORD, J., WALKER, D. "Optimizing the "One Big Switch" Abstraction in Software-defined Networks". In: *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pp. 13–24, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2101-3. doi: 10.1145/2535372.2535373. Available in: <http://doi.acm.org/10.1145/2535372.2535373>.

[48] XIE, L., ZHAO, Z., ZHOU, Y., WANG, G., et al. "An adaptive scheme for data forwarding in software defined network". In: *2014 Sixth International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–5, Oct 2014. doi: 10.1109/WCSP.2014.6992181.

[49] BRAUN, W., MENTH, M. "Wildcard Compression of Inter-Domain Routing Tables for OpenFlow-Based Software-Defined Networking". In: *2014 Third European Workshop on Software Defined Networks*, pp. 25–30, Sept 2014. doi: 10.1109/EWSDN.2014.23.

[50] BARI, M. F., ROY, A. R., CHOWDHURY, S. R., ZHANG, Q., et al. "Dynamic Controller Provisioning in Software Defined Networks". In: *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pp. 18–25, Oct 2013. doi: 10.1109/CNSM.2013.6727805.

[51] TOOTOONCHIAN, A., GORBUNOV, S., GANJALI, Y., CASADO, M., et al. "On Controller Performance in Software-defined Networks". In: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'12, pp. 10–10, Berkeley, CA, USA, 2012. USENIX Association. Available in: <http://dl.acm.org/citation.cfm?id=2228283.2228297>.

[52] PFAFF, B., PETTIT, J., KOPONEN, T., JACKSON, E., et al. "The Design and Implementation of Open vSwitch". In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 117–130, Oakland, CA, 2015. USENIX Association. ISBN: 978-1-931971-218. Available in: <https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff>.

[53] GUDE, N., KOPONEN, T., PETTIT, J., PFAFF, B., et al. "NOX: Towards an Operating System for Networks". In: *ACM SIGCOMM Computer Communication Review*, SIGCOMM '08, pp. 105–110. ACM, 2008.

[54] SHALIMOV, A., ZUIKOV, D., ZIMARINA, D., PASHKOV, V., et al. "Advanced Study of SDN/OpenFlow Controllers". In: *Proceedings of the 9th*

Central & Eastern European Software Engineering Conference in Russia, CEE-SECR '13, pp. 1:1–1:6, New York, NY, USA, 2013. ACM.

[55] ETSI. "Network Functions Virtualisation (NFV); Architectural Framework". October 2013. Available in: <https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf>. ETSI GS NFV 002

[56] ETSI GS NFV 003 V1.3.1. "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV". January 2018. Available in: <https://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_nfv003v010201p.pdf>. ETSI Ind. Spec. Group (ISG).

[57] ETSI GS NFV-SEC 013 V3.1.1. "Network Functions Virtualisation (NFV) Release 3; Security; Security Management and Monitoring specification". February 2017. Available in: <https://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/013/03.01.01_60/gs_NFV-SEC013v030101p.pdf>. ETSI Ind. Spec. Group (ISG).

[58] OPENSTACK. "Tacker - OpenStack NFV Orchestration". https://wiki.openstack.org/wiki/Tacker, December 2016. Acessado: 10/12/2016.

[59] MEDHAT, A. M., TALEB, T., ELMANGOUSH, A., CARELLA, G. A., et al. "Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges", *IEEE Communications Magazine*, v. 55, n. 2, pp. 216–223, 2017.

[60] QUINN, P., ELZUR, U., PIGNATARO, C. *Network Service Header (NSH)*. RFC 8300, RFC Editor, January 2018. Available in: <https://tools.ietf.org/html/rfc8300>. https://tools.ietf.org/html/rfc8300.

[61] BARI, M. F., CHOWDHURY, S. R., AHMED, R., BOUTABA, R., et al. "Orchestrating Virtualized Network Functions", *Computer Networks*, v. 63, pp. 221–237, 2015.

[62] LUIZELLI, M. C., BAYS, L. R., BURIOL, L. S., BARCELLOS, M. P., et al. "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions". In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 98–106, 2015.

[63] QUINN, P., NADEAU, T. *Problem Statement for Service Function Chaining*. RFC 7498, RFC Editor, 2015.

[64] FAYAZBAKHSH, S. K., SEKAR, V., YU, M., MOGUL, J. C. "FlowTags: Enforcing Network-wide Policies in the Presence of Dynamic Middlebox Actions". In: *II ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pp. 19–24. ACM, 2013.

[65] SANZ, I. J., MATTOS, D. M. F., DUARTE, O. C. M. B. "SFCPerf: An automatic performance evaluation framework for service function chaining". In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, April 2018. doi: 10.1109/NOMS.2018.8406237.

[66] SANZ IGOR J., ANDREONI LOPEZ MARTIN, MATTOS, D. M. F., DUARTE, O. C. M. B. "A Cooperation-Aware Virtual Network Function for Proactive Detection of Distributed Port Scanning". In: *IEEE/IFIP 1st Cyber Security in Networking Conference (CSNet'17)*, 2017.

[67] ZHANG, Y., BEHESHTI, N., BELIVEAU, L., LEFEBVRE, G., et al. "StEERING: A software-defined networking for inline service chaining". In: *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pp. 1–10, 2013.

[68] PORRAS, P., SHIN, S., YEGNESWARAN, V., FONG, M., et al. "A Security Enforcement Kernel for OpenFlow Networks". In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pp. 121–126, New York, NY, USA, 2012. ACM. ISBN: 978-1-4503-1477-0. doi: 10.1145/2342441.2342466. Available in: <`http://doi-acm-org.ez29.capes.proxy.ufrj.br/10.1145/2342441.2342466`>.

[69] QAZI, Z. A., TU, C.-C., CHIANG, L., MIAO, R., et al. "SIMPLE-fying Middlebox Policy Enforcement Using SDN", *SIGCOMM Comput. Commun. Rev.*, v. 43, n. 4, pp. 27–38, August 2013. ISSN: 0146-4833. doi: 10.1145/2534169.2486022.

[70] MORAES, I. M., MATTOS, D. M. F., FERRAZ, L. H. G., CAMPISTA, M. E. M., et al. "FITS: A Flexible Virtual Network Testbed Architecture", *Computer Networks*, v. 63, pp. 221–237, 2014.

[71] OPENSTACK. "IETF SFC Encapsulation". `https://docs.openstack.org/networking-sfc/queens/contributor/ietf_sfc_encapsulation.html`, 2018.

[72] MARTINS, J., AHMED, M., RAICIU, C., OLTEANU, V., et al. "ClickOS and the Art of Network Function Virtualization". In: *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pp.

459–473, Seattle, WA, 2014. USENIX Association. ISBN: 978-1-931971-09-6. Available in: <https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/martins>.

[73] ZANNA, P., O'NEILL, B., RADCLIFFE, P., HOSSEINI, S., et al. "Adaptive Threat Management through the Integration of IDS into Software Defined Networks". In: *International Conference on the Network of the Future (NOF) - Workshop on Smart Cloud Networks & Systems*, pp. 1–5, dec 2014.

[74] XING, T., HUANG, D., XU, L., CHUNG, C., et al. "SnortFlow: A OpenFlow-Based Intrusion Prevention System in Cloud Environment". In: *2013 Second GENI Research and Educational Experiment Workshop*, pp. 89–92, March 2013. doi: 10.1109/GREE.2013.25.

[75] KONING, R., GRAAFF, B., POLEVOY, G., MEIJER, R., et al. "Measuring the efficiency of SDN mitigations against attacks on computer infrastructures", *Future Generation Computer Systems*, v. 91, pp. 144 – 156, 2019. ISSN: 0167-739X. doi: https://doi.proxy.ufrj.br/10.1016/j.future.2018.08.011. Available in: <http://www.sciencedirect.com/science/article/pii/S0167739X18302255>.

[76] LIU, Y., GUO, Z., SHOU, G., HU, Y. "To Achieve a Security Service Chain by Integration of NFV and SDN". In: *2016 Sixth International Conference on Instrumentation Measurement, Computer, Communication and Control (IMCCC)*, pp. 974–977, July 2016. doi: 10.1109/IMCCC.2016.162.

[77] BLENDIN, J., RUCKERT, J., LEYMANN, N., SCHYGUDA, G., et al. "Position Paper: Software-Defined Network Service Chaining". In: *2014 Third European Workshop on Software Defined Networks (EWSDN)*, v. 00, pp. 109–114, Sept. 2014. doi: 10.1109/EWSDN.2014.14. Available in: <doi.ieeecomputersociety.org/10.1109/EWSDN.2014.14>.

[78] VIVEK, C., RAJAN, S. P. "Z—TCAM: An Efficient Memory Architecture Based TCAM", *Asian Journal of Information Technology*, v. 15, n. 3, pp. 448–454, 2016.

[79] KANIZO, Y., HAY, D., KESLASSY, I. "Palette: Distributing tables in software-defined networks". In: *2013 Proceedings IEEE INFOCOM*, pp. 545–549, April 2013. doi: 10.1109/INFCOM.2013.6566832.

[80] KOURTIS, M. A., XILOURIS, G., RICCOBENE, V., MCGRATH, M. J., et al. "Enhancing VNF performance by exploiting SR-IOV and DPDK packet

processing acceleration". In: *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pp. 74–78, Nov 2015. doi: 10.1109/NFV-SDN.2015.7387409.

[81] EMMERICH, P., RAUMER, D., WOHLFART, F., CARLE, G. "Performance characteristics of virtual switching". In: *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pp. 120–125, 2014.

[82] BONAFIGLIA, R., CERRATO, I., CIACCIA, F., NEMIROVSKY, M., et al. "Assessing the Performance of Virtualization Technologies for NFV: A Preliminary Benchmarking". In: *2015 Fourth European Workshop on Software Defined Networks*, pp. 67–72, 2015.

[83] FAYAZBAKHSH, S. K., SEKAR, V., YU, M., MOGUL, J. C. "FlowTags: Enforcing Network-wide Policies in the Presence of Dynamic Middlebox Actions". In: *II ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pp. 19–24, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2178-5.

[84] ZHANG, Y., BEHESHTI, N., BELIVEAU, L., LEFEBVRE, G., et al. "StEERING: A Software-defined nEtwoking for inlinE seRvice chainING". In: *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pp. 1–10, October 2013.

[85] CALLEGATI, F., CERRONI, W., CONTOLI, C., SANTANDREA, G. "Dynamic chaining of Virtual Network Functions in cloud-based edge networks". In: *1st IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–5, April 2015.

[86] QAZI, Z., TU, C.-C., MIAO, R., CHIANG, L., et al. "Practical and incremental convergence between SDN and middleboxes", *Open Network Summit, Santa Clara, CA*, 2013.

[87] DUAN, Q., AL-SHAER, E. "Traffic-aware dynamic firewall policy management: Techniques and applications", *IEEE Communications Magazine*, v. 51, n. 7, pp. 73–79, July 2013.

[88] CISCO. "Cisco Nexus 6000 Series NX-OS Security Configuration Guide, Release 7.x". `https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus6000/sw/security/7x/b_6k_Security_Config_7x.html`, April 2017. Acessado: 26/04/2017.

[89] PAGIAMTZIS, K., SHEIKHOLESLAMI, A. "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey", *IEEE Journal of Solid-State Circuits*, v. 41, n. 3, pp. 712–727, 2006.

[90] YU, F., KATZ, R. H., LAKSHMAN, T. V. "Gigabit rate packet pattern-matching using TCAM". In: *IEEE International Conference on Network Protocols (ICNP)*, pp. 174–183, 2004.

[91] QIU, L., VARGHESE, G., SURI, S. "Fast firewall implementations for software and hardware-based routers". In: *Proceedings Ninth International Conference on Network Protocols. ICNP 2001*, pp. 241–250, 2001.

[92] SOLDO, F., MARKOPOULOU, A., ARGYRAKI, K. "Optimal Filtering of Source Address Prefixes: Models and Algorithms". In: *IEEE Conference on Computer Communications (INFOCOM)*, pp. 2446–2454, 2009.

[93] DELL-EMC. "Dell EMC Networking Command Line Reference Guide for the S5048F–ON System". `https://dell.to/2GdlQv4`, 2019.

[94] HAMED, H., EL-ATAWY, A., AL-SHAER, E. "Adaptive Statistical Optimization Techniques for Firewall Packet Filtering". In: *IEEE Conference on Computer Communications (INFOCOM)*, pp. 747–755, April 2006.

[95] IPTABLES. `https://wiki.archlinux.org/index.php/iptables`, April 2018.

[96] EL-ATAWY, A., SAMAK, T., AL-SHAER, E., LI, H. "Using Online Traffic Statistical Matching for Optimizing Packet Filtering Performance". In: *IEEE Conference on Computer Communications (INFOCOM)*, pp. 866–874, May 2007.

[97] FORTINET. "FortiGate Virtual Appliances". `https://www.fortinet.com/content/dam/fortinet/assets/data-sheets/FortiGate_VM.pdf`, April 2017. Acessado: 14/04/2017.

[98] GUPTA, P., PRABHAKAR, B., BOYD, S. "Near-optimal routing lookups with bounded worst case performance". In: *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1184–1192, March 2000.

[99] BOUCHER, M., JOSEFSSON, M., KADLECSIK, J., MCHARDY, P., et al. "Netfilter: firewalling, NAT and packet mangling for Linux". `http://www.netfilter.org/`, August 2016. Acessado: 26/08/2016.

[100] MAURICIO, L. A. F., RUBINSTEIN, M. G. *Avaliação de Desempenho de Plataformas de Virtualização de Redes*. M.Sc. dissertation, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, RJ, Brazil, 2013. Available in: <https://bit.ly/2UfKtwd>.

[101] SUN, C., BI, J., ZHENG, Z., YU, H., et al. "NFP: Enabling Network Function Parallelism in NFV". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pp. 43–56, New York, NY, USA, 2017. ACM. ISBN: 978-1-4503-4653-5. doi: 10.1145/3098822.3098826. Available in: <http://doi.acm.org/10.1145/3098822.3098826>.

[102] ODL. "OpenDaylight: Open Source SDN Platform". https://www.opendaylight.org/, 2018.

[103] LUCREZIA, F., MARCHETTO, G., RISSO, F., VERCELLONE, V. "Introducing network-aware scheduling capabilities in OpenStack". In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–5, April 2015. doi: 10.1109/NETSOFT.2015.7116155.

[104] COUTO, R. S., SADOK, H., CRUZ, P., DA SILVA, F. F., et al. "Building an IaaS cloud with droplets: a collaborative experience with OpenStack", *Journal of Network and Computer Applications*, 2018.

[105] SOMMER, R. "Bro: An Open Source Network Intrusion Detection System." In: *DFN-Arbeitstagung über Kommunikationsnetze*, pp. 273–288, 2003.

[106] OWASP. "The OWASP Foundation: the free and open software security community". https://www.owasp.org/index.php/Main_Page, 2018.

[107] CHODOROW, K. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. " O'Reilly Media, Inc.", 2013.

[108] RISTIC, I. *ModSecurity Handbook*. Feisty Duck, 2010.

[109] NETCRAFT. "January 2019 Web Server Survey". https://news.netcraft.com/archives/2019/01/24/january-2019-web-server-survey.html, 2019.

[110] BAYKARA, M., DAS, R. "A novel honeypot based security approach for real-time intrusion detection and prevention systems", *Journal of Information Security and Applications*, v. 41, pp. 103–116, 2018.

[111] AUTOBENCH. "Autobench: An HTTP benchmarking suite". https://github.com/menavaur/Autobench, 2017.