Journal of Internet Services and Applications, 2023, 14:1, doi: 10.5753/jisa.2023.2847

A Network Function Virtualization Architecture for Automatic and Efficient Detection and Mitigation against Web Application Malware

Leopoldo Mauricio [**Globo and COPPE/PEE/GTA, Universidade Federal do Rio de Janeiro** | ***leopoldo@g.globo*****|** ***leopoldo@gta.ufrj.br*** **]**
Marcelo Rubinstein [**PEL/DETEL-FEN, Universidade do Estado do Rio de Janeiro** | ***rubi@uerj.br***]

Universidade Federal do Rio de Janeiro, Av. Horácio Macedo, 2030, Centro de Tecnologia - Sala H-301 - Cidade Universitária do Rio de Janeiro - RJ, 21941-598

Received: 04 September 2021 • **Accepted:** 27 October 2022 • **Published:** 16 February 2023

Abstract This paper proposes and implements a Network Function Virtualization (NFV) security architecture to provide automatic and efficient detection and mitigation against Web application malware. The mitigation is given by dynamically chaining a Virtual Security Function (VSF) to the data stream to block malicious exploitation traffic without affecting the benign traffic. We implement an NFV Security Controller (NFV-SC) that interacts with an Intrusion Detection System and a Web Application Firewall (WAF), both implemented as VSFs. We also implement a vulnerability scanner and a mechanism to automatically create rules in advance in the WAF-VSF when a security vulnerability is found in an application, even if no malicious traffic has attempted to exploit the flaw. In addition, it dynamically identifies and removes no longer used security rules to improve performance. We implement and evaluate our security proposal in the Open Platform for NFV (OPNFV). The evaluation results in our experimental scenarios show that the NFV security architecture automatically blocks 99.12% of the HTTP malicious traffic without affecting 93.6% of the benign HTTP requests. Finally, we show that the number of rules in the WAF-VSF severely affects the latency to load HTTP response headers and that the number of redirection OpenFlow rules within Open vSwitches is not enough to significantly impact the end-user experience in modern web browser applications.

Keywords: Security, Malware, Network Function Virtualization, Software-Defined Networking

1 Introduction

Web application malware is increasingly frequent and causes significant harm to people and organizations, despite the substantial amount of research conducted on the detection and prevention of their malicious intent [Ashodia and Makadiya (2022); Abdelrahman *et al.* (2021); Zolotukhin *et al.* (2021); Andreoni Lopez *et al.* (2019)]. In the last ten years, the magnitude of the cybersecurity issues has not decreased, and several companies have contended with serious data breaches and significant financial losses caused by malware [The Guardian (2011); Varonis (2020); CBS News (2019); FORBES (2015)]. Only one database attack illegally exposed 191 million users' personal information in 2013 [Chou (2013)], while 98% of Internet companies had to deal with malware attacks in 2017 [Ponemon and Accenture (2017)]. Five hundred forty million personal information records of Facebook users were stolen and disclosed in 2019 [CBS News (2019)], and Capital One had to deal with a data breach that exposed more than 100 million customer records in 2020 [Varonis (2020)]. Additionally, recent security research has estimated a 6 trillion US dollar cost related to cybersecurity threats in 2020 [Williams *et al.* (2020)]. Therefore, innovative architectures are needed for the automatic and efficient detection and mitigation of cybercrime.

Many security attack sources are botnet nodes [Mtibaa *et al.* (2015)], which are a set of zombie machines with malware that allows an attacker to control them remotely. They are also known as "infected" devices. In many cases, owners of enslaved hosts, which may be computers, TVs, cameras, and mobile devices, do not know that their devices are generating malicious traffic. Some estimates indicate that botnets with more than one million zombie devices already exist [Malwaretech (2017)]. Furthermore, the source IPs of a security attack may belong to a Network Address Translation (NAT) gateway that serves many network nodes. Therefore, efficient Web application security systems must be able to prevent vulnerability exploitations, such as SQLI (Structured Query Language Injection), LFI (Local File Inclusion), and Cross-Site Scripting (XSS), without affecting the benign traffic of the same source IPs.

Network operators typically protect systems, applications, and control traffic between networks by deploying and chaining sequential hardware Network Functions (NFs). Security NFs are typically expensive specialized middleboxes found in both data center and Internet Service Provider (ISP) networks. They require a high deployment time and rely on manual configuration by network administrators. An Intrusion Prevention System (IPS), for instance, is an NF that examines traffic flows and detects and prevents vulnerability exploits. Web Application Firewalls filter traffic based on an analysis of the packet contents.

Network Function Virtualization (NFV) is a cost-effective new paradigm that increases flexibility by enabling customized and dynamic software-based service chains with suitable Virtual Network Functions (VNFs). Thus, by using software-based Service Function Chains (SFCs), different route policies can be implemented to steer traffic through hardware-agnostic VNFs [Han *et al.* (2015)]. Software-based SFCs are not tied to network topologies and their physical connections since NFV decouples NFs from specialized hardware to increase flexibility. Because of this, we can chain VNFs to deliver services at the edge, closer to end-users, and fine-grain service chains can deploy VNFs on-demand to meet security or other specific constraint [Mauricio *et al.* (2018)]. In addition, there is a relationship between NFV, cloud computing, and Software-Defined Networking (SDN). 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. [Mijumbi *et al.* (2016)], for instance, state that the best approach to accelerate the development and maturation of NFV is by combining it with SDN technologies and cloud computing capabilities. Regarding security applications, ETSI suggests the development of NFV security architectures containing suitable Virtual Security Functions to deal with malicious activity [Dutta *et al.* (2017)]. We argue that efficient mitigation systems should be able to dynamically steer malicious traffic through security software-based service chains to block exploitation attacks without affecting the benign traffic.

In this paper, we propose an NFV security architecture to provide automatic and efficient detection and mitigation against Web application malware. The security architecture dynamically steers traffic through a WAF (Web Application Firewall) implemented as a Virtual Security Function[1] (WAF-VSF), which can block only malicious activity without interrupting the benign traffic from the same source. We implement an NFV Security Controller (NFV-SC) that executes an algorithm to dynamically create secure chains by applying redirection OpenFlow rules on virtual switches. The NFV Security Controller is also able to periodically assess applications to remove no longer used rules. Additionally, it creates rules in advance in the WAF-VSF when it finds a security vulnerability, even if no malicious traffic has attempted to exploit the flaw. We implement a prototype in the Open Platform for NFV (OPNFV) [OPNFV (2021)] using commodity servers to investigate the efficiency of the proposed NFV security architecture in capturing traffic samples, detecting threats, and automatically filtering malicious activity. The evaluation results show that 99.12% of exploit attacks are automatically blocked, and 93.6% of benign HTTP requests are not affected when the WAF-VSF is dynamically chained. The results also show that the number of rules on the WAF-VSF can impact the latency to load HTTP response headers. Thus, we implement a mechanism to automatically remove unnecessary rules from WAF-VSF to reduce its consumption of resources and increase its performance. Furthermore, we show that the redirection OpenFlow rules within the Open vSwitches are not sufficient to significantly impact the end-user experience in modern web browser applications [Pourghassemi *et al.* (2019)].

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents the proposed NFV security architecture and its prototype implementation. We discuss the performance evaluation results in Section 4, and Section 5 presents conclusions and future work.

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

2 Related Work

Some studies propose security solutions using Network Function Virtualization and Software-Defined Networking properties [Porras *et al.* (2012); Xing *et al.* (2013); Zanna *et al.* (2014); Lin *et al.* (2015); Deng *et al.* (2015); Haugerud *et al.* (2021); Repetto *et al.* (2022); Mauricio *et al.* (2016)]. [Porras *et al.* (2012)] present 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 and IDSs (Intrusion Detection Systems). They can also 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 (OF) rules. However, it is not able to program application firewalls to automatically block malware without disrupting the nonmalicious traffic from the same source IP.

[Xing *et al.* (2013)] propose SnortFlow, which is a Snort-based IDS that uses the OpenFlow protocol to detect attacks. SnortFlow implements countermeasures against attacks through a network reconfiguration. However, the proposal is limited only to the Xen hypervisor virtualization platform, and the Snort [Gupta and Sharma (2020)] agent installed in each management domain (Dom 0) can overload Xen Dom 0. Dom 0 overhead can become a bottleneck, degrading the network performance of all Virtual Machines (VMs) hosted on the same physical server. This can happen because all VMs use the same Dom 0 network drivers to access the network resources located on other machines [Fernandes *et al.* (2011)]. Furthermore, SnortFlow also blocks benign traffic from the same IPs.

[Zanna *et al.* (2014)] integrate an Intrusion Detection System with an SDN controller to identify and block Denial-of-Service (DoS) attacks. The authors configured an IDS Bro to send a blocking flow request to the Application Programming Interface (API) of the Ryu controller to filter IPs that generate malicious traffic. However, in addition to blocking the malicious network traffic, the proposed solution also blocks the benign traffic that is generated by the same source IPs.

Lin *et al.* (2015) propose an SDN security architecture to classify traffic and perform intrusion prevention. The proposal detects Denial of Service attacks and vulnerability exploits by identifying malicious patterns in the HTTP requests. However, despite the significant reduction in network performance when the number of rules increases in the VNF, no workaround is presented. Moreover, the authors do not deal with the automation of attack detection and blocking.

Deng *et al.* (2015) propose the VNGuard framework, which uses SDN and NFV to provide and manage virtualized firewalls. The authors use ClickOS [Martins *et al.* (2014)]

to implement VNGuard components. VNGuard defines a high-level language to simplify policy management so that users do not need to know the low-level information from the virtual networks to create filtering rules. However, the proposed framework does not deal with the meaningful reduction in network performance when the number of rules in the VNF increases.

Haugerud *et al.* (2021) propose an architecture for distributing network traffic between parallel Network Intrusion Detection System (NIDS) nodes based on Network Function Virtualization to reduce the processing time of the pattern matching and reduce the number of packets dropped. The authors have created adaptive algorithms to distribute both the network traffic and the signature rules across the NIDS nodes to make an elastic virtualized IDS. They adjust the number of virtual components which analyze the network traffic in response to the workload. Additionally, they dynamically adjust and divide the signature rules evenly across different Snort NIDS nodes using a node level parallelism method [Jiang *et al.* (2013)] in order to save system resources to keep a zero percent packet drop ratio. The proposed solution duplicates the traffic to send them to several Snort sensors and does not automatically drop packets that raise an alert.

Repetto *et al.* (2022) designed the ASTRID framework to detect and mitigate amplification attacks into a 5G Core (5GC) architecture implemented as an NFV service aiming to block them at their origin, at the boundary of a 5G network, before they get amplified by vulnerable servers in the Internet. ASTRID is a Java Spring Framework, created inside a 5GC network by using the Open5GS project, running inside a Kubernetes cluster. It automates the mitigation of Distributed DoS (DDoS) attacks sent to the User Plane Function (UPF), which is a relevant part of the 5GC. An Analytics Toolkit (ATk) has been created to monitor and detect amplification attacks by distinguishing between periodic fluctuations of requests and anomalies. ASTRID also has a Context Manager (CM) and a Security Controller (SC) that automate as much as possible the collection of data and the implementation of response actions to DDoS. It is a solution suitable to deal only with DDoS, not able to mitigate exploitation attacks.

In a previous work [Mauricio *et al.* (2016)], we study the performance of a firewall implemented as a Virtual Security Function in OPNFV using commodity servers. The implementation exhibits great adaptability, scaling the number of firewall VSFs according to the amount of network traffic. However, it does not include a dynamic mechanism to improve the VSF performance, and it does not deal with malware.

This article proposes and implements an automated and integrated solution for detecting and mitigating Web application malware using the SDN and NFV technologies. The proposal is compliant with the ETSI Network Function Virtualization security management and monitoring specification [Dutta *et al.* (2017)]. Our NFV security architecture algorithm chains the data stream to an application firewall Virtual Security Function with suitable policy rules to only block the attack traffic without harming the benign traffic generated from the same source IPs. Additionally, we propose a mechanism to dynamically improve the performance of the implemented VSF.

Table 1 lists the main features of our NFV/SDN proposal versus those implemented by others.

3 NFV Security Architecture Proposal and Implementation

In this section, we present an NFV security architecture proposal that dynamically creates a software-based chain with a WAF Virtual Security Function to block attacks. We focus on Web Application attacks in this paper.

Figure 1 illustrates the ETSI NFV reference architecture [Dutta *et al.* (2017)] extended with our security components, which aim to provide automatic and efficient detection and mitigation against Web application malware. Our three new components are highlighted in Figure 1. The central element is the NFV Security Controller module, which interacts with an Intrusion Detection System (IDS-VSF) and a Web Application Firewall (WAF-VSF). The IDS Virtual Security Function analyzes the data traffic, detects threats, and sends alerts to Security Controller (Figure 1 (a)). Upon receiving an alert and identifying the type of attack, the NFV Security Controller decides whether an update of the application firewall rules is required. Then, it chains the WAF-VSF to filter the detected malicious activity. The NFV Security Controller writes suitable security rules into the WAF Virtual Security Function to block the detected attacks (Figure 1(b)). Then, the NFV Security Controller sets up redirection OpenFlow rules into the virtual switches to steer traffic (Figure 1(c)), which blocks the malicious traffic without stopping the benign traffic.

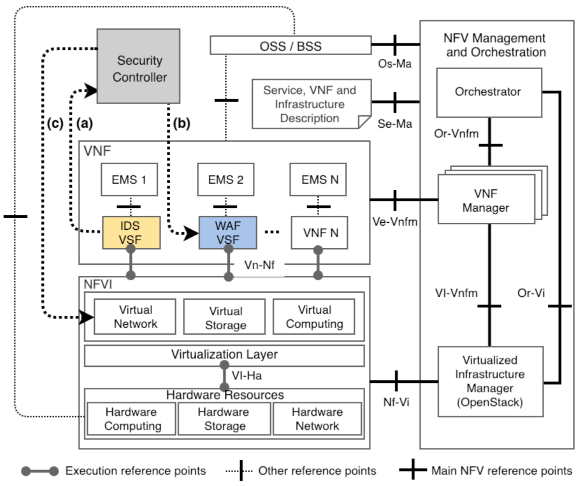

Figure 1. Extended Network Function Virtualization architecture. The proposed security components are colored. The main interactions of the proposed Security Controller are a) receiving alerts; b) automatic installation of the customized security rules in a Web Application Firewall; and c) firewall chaining to block malicious traffic.

Figure 2 details how the proposed modules of the NFV security architecture work. Furthermore, we detail the interactions when malicious traffic sent to a vulnerable Web application is blocked. We have used Service Function Chaining to steer traffic when needed. The NFV Security Controller cre-

Table 1. NFV/SDN features implemented versus related work.

Features	Porras *et al.* (2012)	Xing *et al.* (2013)	Zanna *et al.* (2014)	Lin *et al.* (2015)	Deng *et al.* (2015)	Haugerud *et al.* (2021)	Repetto *et al.* (2022)	Mauricio *et al.* (2016)	This work
Provides access control	✓	✓	✓	✓	✓		✓	✓	✓
Detects malware	✓	✓	✓	✓		✓			✓
Automatically blocks malware		✓	✓						✓
Dynamically reconfigures the network to block malicious traffic		✓	✓						✓
Filters malware without affecting the benign traffic									✓

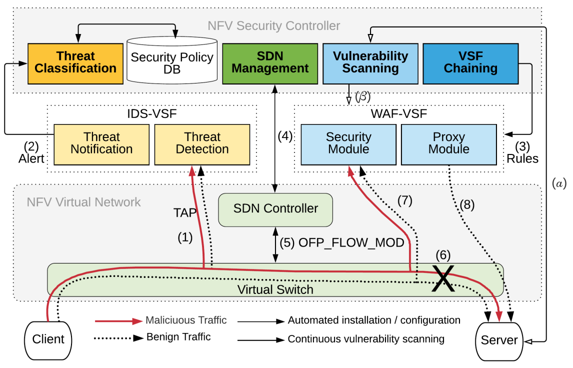

Figure 2. NFV Security Controller interactions with Virtual Security Functions and the NFV Virtual Network.

ates OpenFlow Service Function Chains managing an SDN controller that is inside the NFVI Virtual Network module (Figure 1). We chose to create OpenFlow SFCs because SFC using the Network Service Header (NSH) protocol was still an experimental feature that exhibited poor network performance when our tests were executed [Sanz *et al.* (2018)].

We have also implemented the Vulnerability Scanning module, which performs a periodic vulnerability scan, based on known vulnerability signatures, on the attacked Web Application. Therefore, it can also validate if other vulnerabilities not yet exploited and detected exist in the attacked Web Application. This is because Web application developers frequently update their source code to provide new functions. If a vulnerability is no longer identified and blocked the NFV Security Controller will automatically remove the unnecessary rule related to it. Thus, with fewer rules, we are able to reduce the consumption of resources in the WAF-VSF and increase its performance [Mauricio *et al.* (2016); Sanz *et al.* (2018)]. Consequently, the efficiency of the software-based Service Chain will also rise as the consumption of computational resources decreases.

We have implemented the Intrusion Detection System as a Virtual Security Function with Threat Detection and Threat Notification mechanisms (Figure 2). To build the IDS-VSF Threat Detection, we use the open-source IDS Bro [Sommer (2003)], which we configure to passively monitor network traffic, searching for suspicious activity. Thus, it can analyze network traffic to find events with malicious patterns, such as those related to XSS, SQL Injection, and LFI. We have used the TAP technique[2] to send a copy of each network packet to the IDS-VSF. The IDS-VSF receives copies of each network packet without being chained to the data stream so that it does not overwhelm the end-to-end communication of the software-based Security Chain, which we automatically create if needed.

Interaction (1) in Figure 2 shows that the Intrusion Detection System receives traffic through its TAP network interface. The IDS-VSF Threat Notification is a Python code we have designed that extends the Bro IDS code to send HTTP messages with threat alerts to the NFV Security Controller (Interaction (2) in Figure 2). Each alert reports the IP address generating the malicious traffic, the destination IP address, the attack-related Uniform Resource Locator (URL), the Uniform Resource Identifier (URI) used, and the type of threat detected.

The Threat Classification of the NFV Security Controller, its VSF chaining mechanism, and the SDN Management engine are Python applications. We use w3af[3], which is an open-source web vulnerability scanner, to build the Vulnerability Scanning module. In addition, the NFV Security Controller has a Security Policy Database (DB) that we have built to store threat alert types; in our case, the OWASP top ten most critical Web Application security risks [OWASP (2021)], and suitable security rules able to block them. We have used MongoDB to create this database. Moreover, we have created our security rules in Mod Security, which is a widely deployed open-source WAF [Ristic (2010)].

The NFV Security Controller executes the proposed Algorithm 1 to handle packets when malicious activity is detected. Upon receiving alerts from the IDS-VSF Threat Notification, the Threat Classification engine of the NFV Security Controller validates whether malicious traffic is a Denial of Service (Ψ) attack (Algorithm 1, Line 2). If so, the NFV Security Controller automatically sends OpenFlow rules to the OpenDaylight controller (Interaction (4) in Figure 2) to filter all the network traffic generated by the identified source IP address (Line 3).

Unknown vulnerability exploitation is a zero-day attack that has no available fix. Therefore, every time the Threat Classification module of the NFV Security Controller receives an unknown vulnerability exploitation alert (Line 4), it sends HTTP messages to the OpenDaylight REST API to block all the network traffic generated by the identified source IP address, protecting the vulnerable application IP

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

[3]http://w3af.org/

Algorithm 1: Security Attack Treatment

```
input :$\delta$ = {srcIP,dstIP,URL,URI,type}: Detected attacks;
       $\Phi$: Known vulnerability exploitations;
       WAF-VSF: the WAF Virtual Security Function;
       $\Psi$: Denial of service attacks;
1  begin
2      if $\delta.type \in \Psi$ then
3          Filter security attack($\delta.srcIP, *$);
4      else if $\delta.type \notin \Phi$ then
5          Filter security attack($\delta.srcIP, \delta.dstIP$);
6      else
7          edit_waf-vsf_rules($\delta.URL, \delta.URI, \delta.type$);
8          create_chain(WAF-VSF);
9          divert_traffic($\delta.srcIP, \delta.dstIP$);
10         Register($\delta$);
11         while $\Phi \in (URL \parallel URI)$ do
12             if $\Phi \notin$ WAF-VSF then
13                 write_policies(WAF-VSF($security\_module$));
14         end
15         remove_policies(WAF-VSF($security\_module$))
16 end
```

(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 the malicious traffic. However, security network operators can register new security rules on the Security Policy Database at any time.

When IDS-VSF detects known vulnerability exploitation different from DoS (Line 6), the Threat Classification engine will find suitable rules on the Security Policy Database to block the malicious activity. Then, it triggers the VSF chaining mechanism of the NFV Security Controller that automatically configures rules inside the WAF Virtual Security Function (Interaction (3) in Figure 2) to mitigate the detected attack. Then, through the SDN controller OpenDaylight (Interaction (4) in Figure 2), the NFV Security Controller inserts redirection OpenFlow rules into the OVS (Interaction (5) in Figure 2). Thus, all traffic previously chained directly to the vulnerable Website (Interaction (6) in Figure 2) is routed first through the WAF Virtual Security Function (Interaction (7) in Figure 2), which filters the malicious traffic without blocking the benign HTTP requests (Lines 7-9). After that, the NFV Security Controller starts its Vulnerability Scanning module (Lines 11-15) to periodically assess the application attacked (Interaction (α) in Figure 2). If it finds a new vulnerability different from the previous one detected by the IDS-VSF Threat Detection, the NFV Security Controller adds rules to the WAF-VSF Security Module (Line 13). Thus, we can discover vulnerabilities and mitigate the threats before an attacker can exploit other weaknesses of the attacked Web application. Furthermore, if a known vulnerability that has been once identified no longer exists, the NFV Security Controller removes this unnecessary policy from the WAF-VSF Security Module (Line 15). The NFV Security Controller also records all attacks received (Line 10).

We have used Mod Security to create our WAF-VSF Security Module. We have implemented our Proxy Module using NGINX, which is the second most used Web server on the Internet [Netcraft (2019)]. 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 the benign HTTP requests it receives to the vulnerable Web server (Interaction (8) in Figure 2). In this way, the WAF Virtual Security Function removes malicious traffic without blocking the benign HTTP requests generated from the same source address.

We have implemented this proposal using the Open Platform for NFV. Our OPNFV installation involves five Commercial Off-The-Shelf (COTS) machines. The OPNFV manager is a desktop, we use one machine to build an OpenStack [OpenStack (2022)] controller node, and the three others are OpenStack compute nodes, which host our three VSFs: the NFV Security Controller, the WAF, and the IDS. The OPNFV virtual network module uses OpenDaylight (ODL) as an SDN controller, the virtual switches are Open vSwitches (OVSes), and OpenStack acts as an NFV VIM, managing all the physical resources of the Network Function Virtualization Infrastructure. We create an OpenFlow data plane using the Open vSwitches from the OpenStack compute nodes. Furthermore, we built the SDN control plane using the OpenDaylight controller that we install on the OpenStack network node.

As our prototype has served as the basis for implementing Globo's production security architecture, we can not provide its source code. However, Globo has provided detailed instructions and an open-source code [Globo (2021b)] that allow a researcher to carry out the OWASP top ten attacks discussed in the paper by provisioning local environments via docker-compose. We teach how to exploit the most critical Web application security risks and how to fix the given vulnerable codes to mitigate them. We have also released huskyCI [Globo (2021a)] as an open-source tool that can perform static security analysis in Python, Ruby, JavaScript, Golang, Java, and HCL.

4 Experimental Results

In this section, we detail the experiments carried out to evaluate the proposed NFV security architecture.

4.1 Efficiency

To assess the efficiency of the proposal in blocking the malicious traffic, without affecting the benign traffic generated by the same source, we developed a script to send malicious and benign HTTP requests from a client to a honeypot [OWASP (2021)]. The honeypot acts as a vulnerable Web server. The script runs for a fixed time interval, in which it sends 220 benign and 340 malicious HTTP requests, on average. Benign HTTP requests access the main page of the vulnerable Website[4]. In contrast, the malicious HTTP requests (malware) try to exploit the OWASP top ten security vulnerabilities [OWASP (2021)].

Table 2 illustrates some security attack types we use in the first evaluation. The HTTP request of Line 1 tries to obtain the encrypted password, IDs, and names of all the registered

[4]We use the curl command to send several HTTP requests to the vulnerable Website.

Table 2. 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

users of a vulnerable Operating System (OS) hosting the exposed Website. Line 2 shows an HTTP request used to insert a file named "9zseqh" on the Web server. This file could be malware that is able to execute malicious code to enslave or damage an operating system. Line 3 is an SQL Injection attack to obtain all the information from three registered users in the "adminusers" table of the honeypot database. We can also inject malicious JavaScript to redirect the users to phishing sites or modify the visited HTML. Line 4 describes an attack to list all the running processes on the honeypot OS. We have also carried out attacks to explore more of the OWASP's top ten Web application security risks. The other types of attacks and the steps to perform each of them are in the lab we have created to teach secure Web and mobile development in a practical way [Globo (2021b)].

The evaluation results can have different HTTP response types. Malicious HTTP requests described in our example of Table 2, Lines 1, 2, and 3 are successful because they receive 2xx (OK) HTTP response status codes. This can happen when our automatic NFV security architecture does not block the malicious traffic sent to the honeypot. On the other hand, the HTTP response status code of Table 2, Line 4 indicates that the malicious request is unsuccessful, which means that our security architecture would have been able to prevent the attack.

As already mentioned, we have installed the Open Platform for NFV with one OpenStack controller node, three computing nodes, and one OPNFV installation manager. The computing nodes are Ubuntu 14.04, KVM is the virtualization platform, and OpenStack is the OPNFV Virtualized Infrastructure Manager. The OPNFV manager and the OPNFV controller node are Intel (R) Core (TM) i7-4770 CPU @ 3.40 GHz. Each controller has four cores, 8 threads, 32 GB of RAM, and three 1 Gb/s Ethernet interfaces. The OpenStack compute nodes are Intel (R) Xeon (R) E5-2650 CPU @ 2.00 GHz. Each OpenStack compute node has eight cores, 16 threads, 64 GB of RAM, and three 1 Gb/s Ethernet interfaces. The NFV Security Controller, the WAF Virtual Security Function, the IDS-VSF, the Security Policy Database, the Web client, and the vulnerable Web server are VMs with four virtual CPUs and 4 GB of RAM. The evaluations are averages of 10 runs with 95% confidence intervals. Some confidence intervals are hard to see because they are too small.

In the initial scenario, we have sent malicious and benign HTTP requests to the honeypot without enabling the proposed security modules on the NFV security architecture, and, as expected, all the malicious HTTP requests successfully received 2xx (OK) status code responses. Next, we evaluate the NFV security architecture efficiency in dynamically blocking malicious traffic without stopping benign traffic. There are no preconfigured rules on the WAF-VSF Security Module and no previous URL redirection policies on its Proxy Module when we start this evaluation. Furthermore, all the malicious HTTP requests generated in the evaluation have appropriate security rules stored in the Security Policy Database.

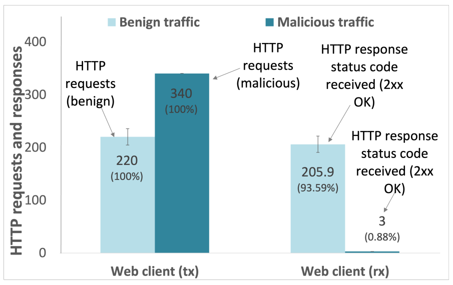

Figure 3. Web client's benign and malicious HTTP requests (tx) and responses (rx).

Figure 3 illustrates that the Web client test script generates an average of 340 malicious HTTP requests and 220 benign ones (see HTTP requests (malicious) and HTTP requests (benign) sent by the Web client (tx)). As soon as the Web client generates the malicious traffic, the IDS-VSF identifies the malicious requests and sends an alert message to the NFV Security Controller. Figure 4 shows that the NFV Security Controller automatically chains the WAF-VSF in less than 8.2 s after it receives an alert message. The NFV Security Controller diverts all the traffic to the WAF-VSF by applying redirection OpenFlow rules on the virtual switches. Moreover, the NFV Security Controller automatically configures the WAF-VSF Proxy Module to deal with the HTTP requests sent to the attacked URL, and it inserts security rules into the WAF-VSF to filter all the identified malicious traffic.

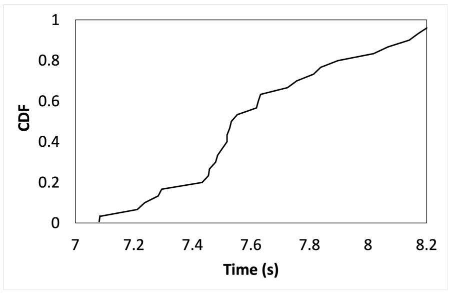

Figure 4. Cumulative Distribution Function of the time to chain the WAF-VSF after the NFV Security Controller receives an alert message.

Figure 5 shows that the vulnerable Web server sends back, on average, three (0.88%) successful HTTP response status codes related to the malicious requests. 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 3). The evaluation results indicate that there are network packet loss issues when the NFV 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 the packets with a Web client source IP and Web server destination IP

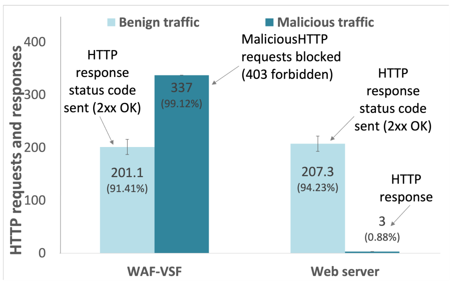

Figure 5. Malicious HTTP requests blocked by the WAF virtual security function, successful HTTP responses sent by the Web server, and how many pass through the firewall when WAF-VSF is automatically chained.

to the WAF Virtual Security Function. During this operation, packet losses occur, such as those that occur when a router in a traditional packet-switching network fails. For this reason, on average, 207.3 (94.23%) (see HTTP response status code sent (2xx OK) by the Web server in Figure 5) of the 220 benign HTTP requests reach the Web server without problems.

Likewise, on average, 205.9 (93.59% of the benign requests - see the HTTP response status code received (2xx OK) by the Web client in Figure 3) of the 207.3 of the HTTP response status codes sent by the Web server successfully reach the Web client. Moreover, the WAF Virtual Security Function chained to the data stream blocks 337 (99.12% of the malicious HTTP requests - see the malicious HTTP requests blocked (403 forbidden) by the WAF-VSF in Figure 5), and its reverse proxy handles 201.1 (see the 201.1 HTTP response status code sent (2xx OK) by the WAF-VSF in Figure 5) (91.41%) of the 220 benign HTTP requests made by the Web client. There were an average of 12.7 (5.78%) benign HTTP requests lost between the client and the vulnerable Web server (220-207.3). The proposed WAF-VSF proxy module handles approximately, on average, 201.1 of the 207.3 benign HTTP responses sent from the server to the client, so six of these HTTP responses do not go through the WAF-VSF. In addition, 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 evaluation results show that the proposed NFV security architecture is efficient since it dynamically blocks a total of 99.12% of the generated attacks, and 93.59% of the benign traffic is not affected when dynamically chained with the WAF Virtual Security Function.

4.2 Performance

Afterward, we have evaluated the performance of our NFV security architecture. Our goal was to evaluate how the new software-based Service Chain containing the WAF-VSF affects the HTTP response rate and the Website response time. We have first varied the number of HTTP requests per second sent to a Web server, and we have also generated OWASP's top ten attacks. Consequently, the proposed NFV security architecture dynamically configured up to ten suitable rules in WAF-VSF to block the malicious HTTP traffic sent to the honeypot, which we use as a vulnerable Web server. As already mentioned, we have used Mod Security to create the WAF-VSF Security Module, which handles all communication between the Web client and the Web server when malicious traffic is detected. Therefore, we have first evaluated how the WAF-VSF affects the HTTP response rate and the website response time. We generate the same benign HTTP requests using HTTPerf [Mosberger and Jin (1998)], as we did in the initial scenario, and a Python script orchestrates the security attacks. Moreover, HTTPerf generates and maintains sustainable rates of HTTP requests to overload the Web server.

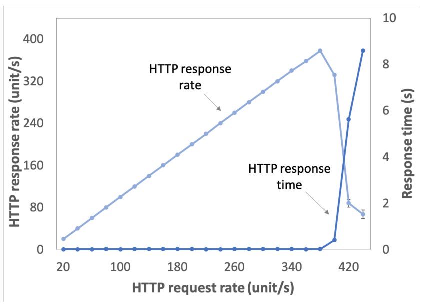

Figure 6. Web application performance when the WAF-VSF is not chained.

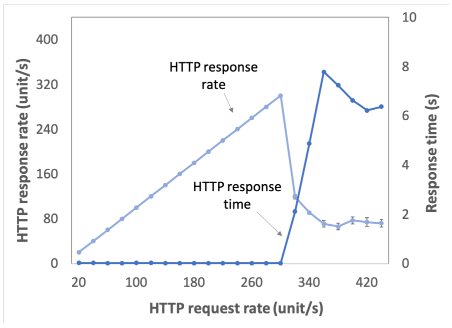

Figure 7. Web application performance when the WAF-VSF is chained to the Web server.

We automate the HTTPerf using Autobench [Midgley (2020)] and vary the TCP connection rate between 10 and 220, increasing the rate by ten at each 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, which is the maximum capacity of our honeypot Web server. We use these values because several preliminary tests have shown that HTTP request rates over 440 requests per second cause a significant increase in the HTTP response time, which is the time elapsed between requesting and receiving an object from the server. The results are means with 95% confidence intervals. Some confidence intervals are not shown in the figures because they are narrow.

Figure 6 illustrates the Web application performance when the WAF-VSF is not chained. The maximum HTTP request rate supported by the Web server without a significant increase in the HTTP response time is equal to 380 requests per second. For higher rates, the response time increases considerably, going from approximately zero to approximately 9 s when the Web client attempts to send 440 HTTP requests per second to the server.

Figure 7 shows that the automatic WAF Virtual Security Function chaining reduces the maximum HTTP response rate to 300 requests per second when the WAF-VSF handles the benign traffic and OWASP's top ten attacks at the same time. Therefore, the HTTP request rate is reduced by approximately 21% when the NFV security architecture dynamically chains and inserts rules into the WAF-VSF to block OWASP's top ten attacks. Traffic filtering performed by WAF-VSF is a high CPU usage technique. Therefore, by increasing the number of virtual CPUs (vCPUs) we can improve the WAF-VSF performance [ur Rahman *et al.* (2018)]. We can also raise the WAF-VSF performance by associating one vCPU to one physical CPU core instead of associating each vCPU to a logical CPU core [Wang *et al.* (2017)]. However, only modern CPUs and virtualization platforms offer that feature.

4.2.1 Effect of Varying the Number of Rules on the WAF-VSF

As already mentioned, the proposed NFV security architecture has a mechanism that dynamically removes unused security rules from the WAF-VSF when the Vulnerability Scanning module no longer finds the related vulnerabilities in the Web server. To evaluate the performance of this mechanism, we have also conducted experiments to emulate its behavior. We have varied the number of rules on the WAF-VSF between 0 and 100. Hence, we did not just create the number of rules needed to block attacks, as shown in the previous sections. Then, we measure the amount of time to load the HTTP response header when we get the index.html page from the Web server. In this particular test, the generated benign traffic corresponds to HTTP HEAD requests so that we avoid Web browser caching, ensuring more stable results. All HTTP requests pass through the WAF-VSF.

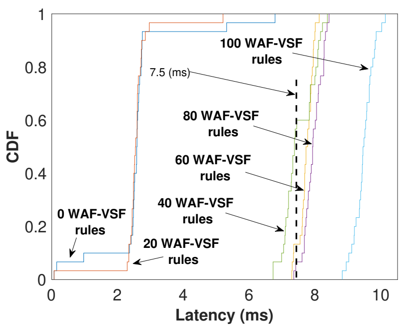

Figure 8. Cumulative Distribution Function of the time to load the HTTP response headers from the main Web server page (index.html) when varying the number of rules in the WAF-VSF between 0 and 100.

Figure 8 illustrates the Cumulative Distribution Function (CDF) of the amount of time, in milliseconds, to load the HTTP response header of the index.html honeypot Web page (latency) when varying the number of rules inside the WAF-VSF. The latency is less than 2.8 ms, in more than 90% of the cases when the WAF-VSF acts only as a Web proxy because its number of security rules is equal to zero. In addition, we have approximately the same behavior, with a latency of less than 2.8 ms, in more than 90% of the cases, when there are 20 Web Application Firewall rules in the WAF-VSF Security Module. However, when we insert 40 security rules into the WAF-VSF, the latency is greater than 7.5 ms in approximately 40% of the cases. When we create 60 security rules, the latency is greater than 7.5 ms in more than 80% of the cases. With 80 Web Application Firewall rules in the WAF-VSF, the latency is greater than 7.5 ms in more than 90% of the cases. In addition, it is greater than 8 ms in approximately 40% of the cases. With 100 security rules inside the proposed WAF-VSF, the minimum latency is equal to 8.8 ms, and in more than 90% of the cases, it is greater than 9.0 ms. The latency is, in 90% of the cases, less than 2.8 ms when the number of rules inside the WAF-VSF is zero, and greater than 9.0 ms when the number of rules is equal to 100. Therefore, when the number of rules inside the WAF-VSF varies between 0 and 100, the increase in latency is approximately 221%.

The results show that the latency is lower when there are fewer rules inside the WAF-VSF. This indicates the benefits of the mechanism that dynamically removes unused security rules from the WAF-VSF.

4.2.2 Effect of Increasing the Number of Redirection OpenFlow Rules on the Open vSwitches

As already mentioned, we have created an OpenFlow data plane using the Open vSwitches from OpenStack compute nodes, and we have built the SDN control plane using the OpenDaylight controller. The virtual switches of the NFV/SDN environments in production clouds can have thousands of redirection OpenFlow rules [Mauricio *et al.* (2018)]. Therefore, we have also conducted experiments to understand the effect of increasing the number of those redirection OpenFlow rules inside the Open vSwitches of our OPNFV cloud.

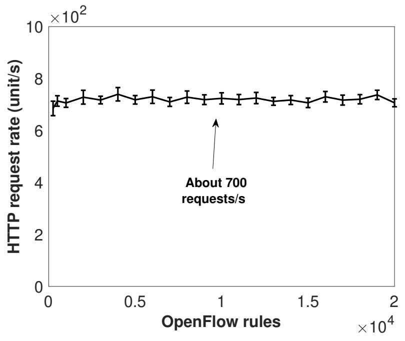

Figure 9. Maximum HTTP request rate for different numbers of redirection OpenFlow rules in the Open vSwitches.

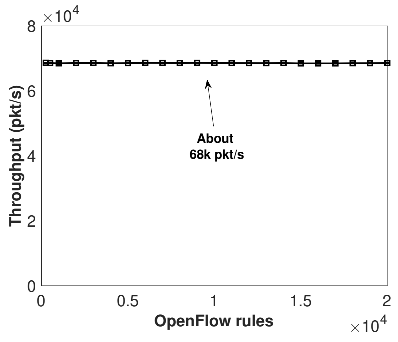

Figure 10. Maximum throughput for different numbers of redirection OpenFlow rules in the Open vSwitches.

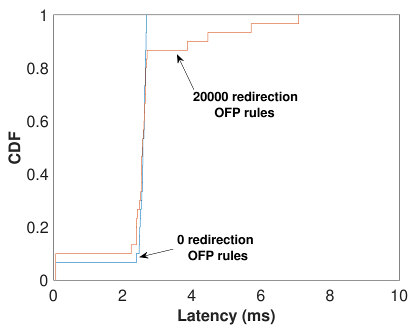

Figure 11. Cumulative distribution function of the time to load the HTTP response headers from the Web test page we have created in the WAF-VSF when varying the number of redirection OpenFlow rules inserted in the Open vSwitches between 0 and 20000.

We use HTTPerf to also evaluate how much the increase in the number of redirection OpenFlow rules in Open vSwitches can affect the maximum HTTP request rate and the network throughput when a client sends traffic to a server. In this particular test, we use the WAF-VSF as a Web server so that the traffic test does not traverse a VSF. It follows only through the two Open vSwitches since each VM used is in a different physical machine. Thus, we avoid the VSF overhead while measuring the mean time to load the header (HTTP HEAD requests) of a test page that we have created within the WAF-VSF. We use HTTPerf to generate only HTTP benign traffic. To evaluate the network throughput, we use the Iperf tool to send UDP packets with 1472 bytes at 1 Gb/s from the client to the WAF-VSF. We vary the number of redirection OpenFlow rules from 250 to 20000. In these particular tests, the results are means of 15 runs with 95% confidence intervals.

Figures 9 and 10 illustrate that the increase in the number of redirection OpenFlow rules within OVSes does not significantly affect the HTTP request rate (approximately 700 requests/s) and the network throughput (approximately 68k packets/s), respectively.

Figure 11 illustrates that the latency is less than 2.7 ms in 100% of the cases when the number of redirection OpenFlow rules within the OVSes is equal to zero. With 20000 redirection OpenFlow rules inside the OVSes, the latency is less than 2.7 ms in approximately 88% of the cases. However, the latency is greater than 4 ms in 10% of the cases and higher than 6 ms in less than 5% of the cases. Therefore, the results show that we do not have a considerable increase in the latency when the number of redirection OpenFlow rules varies between 0 and 20000. This occurs because OVS processes OpenFlow rules using hash tables [Mauricio *et al.* (2018)]. In practical terms, an HTTP latency growth of 3.3 ms (6 ms - 2.7 ms) is not enough to significantly impact the end-user experience in modern web browser applications [Pourghassemi *et al.* (2019)]. Google, for instance, states that 53% of mobile site visitors leave a page only when it takes longer than 3 s to load [An (2018)].

5 Conclusions and Future Work

In this paper, we have implemented an NFV security architecture that applies countermeasures against vulnerability exploitation attacks without affecting benign traffic. The central element of the NFV security architecture 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, which decides whether an update of the application firewall rules is required and if the WAF-VSF should be dynamically chained to filter the detected malicious traffic. The NFV Security Controller sets up redirection OpenFlow rules into virtual switches to steer traffic through the WAF-VSF that efficiently blocks 99.12% of the malicious HTTP traffic without significantly affecting the benign traffic from the same source IP. Furthermore, the NFV Security Controller implements a Vulnerability Scanning module and a mechanism that reduces the time between finding a vulnerability and the mitigation of the identified threats. In addition, the NFV Security Controller automatically removes rules from the WAF-VSF to reduce the number of security rules implemented and increases its performance when the vulnerabilities cease to exist. As future work, we intend to focus on chaining containers to VSFs created as Kubernetes pods so that we can secure microservices running on PaaS (Platform as a Service).

Acknowledgements

This work has been sponsored by CNPq, CAPES, and FAPERJ. This paper is dedicated to the memory of Professor Otto Carlos M. B. Duarte.

Declarations

Authors' Contributions

All authors contributed to the writing of this article, read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

Data can be made available upon request.

References

Abdelrahman, A. M., Rodrigues, J. J., Mahmoud, M. M., Saleem, K., Das, A. K., Korotaev, V., and Kozlov, S. A. (2021). Software-defined networking security for private data center networks and clouds: vulnerabilities, attacks, countermeasures, and solutions. *International Journal of Communication Systems*, 34(4). e4706. DOI: 10.1002/dac.4706.

An, D. (2018). Find out how you stack up to new industry benchmarks for mobile page speed. Avalable at: `https://bit.ly/3TEpLFz`.

Andreoni Lopez, M., Mattos, D. M. F., Duarte, O. C. M. B., and Pujolle, G. (2019). A fast unsupervised preprocessing method for network monitoring. *Annals of Telecommunications*, 74(11-12):139–155. DOI: 10.1007/s12243-018-0663-2.

Ashodia, N. and Makadiya, K. (2022). Detection and mitigation of DDoS attack in software defined networking: A survey. In *2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, pages 1175–1180. IEEE. DOI: 10.1109/ICSCDS53736.2022.9760911.

CBS News (2019). Hundreds of millions of facebook user records were exposed on amazon cloud server. Available at: `https://www.cbsnews.com/news/millions-facebook-user-records-exposed-amazon-cloud-server/`.

Chou, T. (2013). Security threats on cloud computing vulnerabilities. *International Journal of Computer Science & Information Technology*, 5(3):79. DOI: 10.5121/ijcsit.2013.5306.

Deng, J., Hu, H., Li, H., Pan, Z., Wang, K.-C., Ahn, G.-J., Bi, J., and Park, Y. (2015). 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)*, pages 107–114. DOI: 10.1109/NFV-SDN.2015.7387414.

Dutta, A., Sood, K., Lu, W., *et al.* (2017). Network functions virtualisation (nfv) release 3; security; security management and monitoring specification. Technical report, ETSI. `https://bit.ly/2Gbn8II`.

Fernandes, N., Moreira, M., Moraes, I., Ferraz, L., Couto, R., Carvalho, H., Campista, M., Costa, L., and Duarte, O. C. M. B. (2011). Virtual networks: Isolation, performance, and trends. *Annals of Telecommunications*, 40(1):339–355. DOI: 10.1007/s12243-010-0208-9.

FORBES (2015). Ashley madison hack data reveals interesting statistics. Available at: `https://www.forbes.com/sites/tonybradley/2015/08/19/ashley-madison-hack-data-reveals-interesting-statistics/?sh=55779de1cfdc`.

Globo (2021a). huskyci - an open source tool that orchestrates security tests and centralizes all results into a database for further analysis and metrics. `https://github.com/leopoldomauricio/huskyCI`.

Globo (2021b). secdevlabs - a laboratory for learning secure web and mobile development in a practical manner. Available at: `https://github.com/leopoldomauricio/secDevLabs`.

Gupta, A. and Sharma, L. S. (2020). A categorical survey of state-of-the-art intrusion detection system-snort. *Int. J. Inf. Comput. Secur.*, 13(3/4):337–356. DOI: 10.1504/IJICS.2020.109481.

Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97. DOI: 10.1109/MCOM.2015.7045396.

Haugerud, H., Tran, H. N., Aitsaadi, N., and Yazidi, A. (2021). A dynamic and scalable parallel network intrusion detection system using intelligent rule ordering and network function virtualization. *Future Generation Computer Systems*, 124:254–267. DOI: 10.1016/j.future.2021.05.037.

Jiang, H., Xie, G., and Salamatian, K. (2013). Load balancing by ruleset partition for parallel IDS on multi-core processors. In *International Conference on Computer Communications and Networks, ICCCN.*

Lin, Y.-D., Lin, P.-C., Yeh, C.-H., Wang, Y.-C., and Lai, Y.-C. (2015). An extended SDN architecture for network function virtualization with a case study on intrusion prevention. *IEEE Network*, 29(3):48–53. DOI: 10.1109/MNET.2015.7113225.

Malwaretech (2017). Mapping mirai: A botnet case study. `https://www.malwaretech.com/2016/10/mapping-mirai-a-botnet-case-study.html`.

Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R., and Huici, F. (2014). Clickos and the art of network function virtualization. Available at:`https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/martins`.

Mauricio, L. A. F., Rubinstein, M. G., and Duarte, O. C. M. B. (2016). Proposing and evaluating the performance of a firewall implemented as a virtualized network function. In *International Conference on the Network of the Future (NOF)*, pages 1–3. DOI: 10.1109/NOF.2016.7810127.

Mauricio, L. A. F., Rubinstein, M. G., and Duarte, O. C. M. B. (2018). Aclflow: An NFV/SDN security framework for provisioning and managing access control lists. In *International Conference on the Network of the Future (NOF)*, pages 44–51. DOI: 10.1109/NOF.2018.8598136.

Midgley, J. T. J. (2020). Autobench: An http benchmarking suite. Available at: `https://github.com/menavaur/Autobench`.

Mijumbi, R., Serrat, J., Gorricho, J. L., Bouten, N., Turck, F. D., and Boutaba, R. (2016). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys Tutorials*, 18(1):236–262. DOI: 10.1109/COMST.2015.2477041.

Mosberger, D. and Jin, T. (1998). Httperf — a tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):31–37. DOI: 10.1145/306225.306235.

Mtibaa, A., Harras, K. A., and Alnuweiri, H. (2015). From botnets to mobibots: A novel malicious communication paradigm for mobile botnets. *IEEE Communications Magazine*, 53(8):61–67. DOI: 10.1109/MCOM.2015.7180509.

Netcraft (2019). January 2019 Web Server Survey. Available at: `https://news.netcraft.com/archives/2019/01/24/january-2019-web-server-survey.html`.

OpenStack (2022). The most widely deployed open source cloud software in the world. Available at: `https://www.openstack.org`.

OPNFV (2021). Open platform for NFV. Available at: `https://www.opnfv.org`.

OWASP (2021). Owasp honeypot. `https://github.com/OWASP/Python-Honeypot`.

OWASP (2021). Top 10 web application security risks. Available at: `https://owasp.org/www-project-top-ten/`.

Ponemon and Accenture (2017). 2017 cost of cyber crime study - insights on the security investments that make a difference. Technical report. `https://accntu.re/3H5VPgA`.

Porras, P., Shin, S., Yegneswaran, V., Fong, M., Tyson, M., and Gu, G. (2012). A security enforcement kernel for openflow networks. In *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pages 121–126. DOI: 10.1145/2342441.2342466.

Pourghassemi, B., Amiri Sani, A., and Chandramowlishwaran, A. (2019). What-if analysis of page load time in web browsers using causal profiling. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(2). DOI: 10.1145/3341617.3326142.

Repetto, M., Bruno, G., Yusupov, J., Lamanna, G., Ertl, B., and Carrega, A. (2022). Automating mitigation of amplification attacks in NFV services. *IEEE Transactions on Network and Service Management*. DOI: 10.1109/TNSM.2022.3172880.

Ristic, I. (2010). *ModSecurity Handbook: The Complete Guide to the Popular Open Source Web Application Firewall*. Feisty Duck. ISBN 978-1907117022.

Sanz, I. J., Mattos, D. M. F., and Duarte, O. C. M. B. (2018). Sfcperf: An automatic performance evaluation framework for service function chaining. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–9. DOI: 10.1109/NOMS.2018.8406237.

Sommer, R. (2003). Bro: An open source network intrusion detection system. Available at: `https://dl.gi.de/handle/20.500.12116/29277`.

The Guardian (2011). Playstation network hackers access data of 77 million users. Available at: `https://www.theguardian.com/technology/2011/apr/26/playstation-network-hackers-data`.

ur Rahman, H., Wang, G., Chen, J., and Jiang, H. (2018). Performance evaluation of hypervisors and the effect of virtual CPU on performance. In *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 772–779. DOI: 10.1109/SmartWorld.2018.00146.

Varonis (2020). Inside out security - capital one's cloud breach & why data-centric security matters. Available at: `https://www.varonis.com/blog/capital-ones-cloud-breach-why-data-centric-security-matters`.

Wang, C., Urgaonkar, B., Nasiriani, N., and Kesidis, G. (2017). Using burstable instances in the public cloud: Why, when and how? *Proc. ACM Meas. Anal. Comput. Syst.*, 1(1). DOI: 10.1145/3084448.

Williams, C. M., Chaturvedi, R., and Chakravarthy, K. (2020). Cybersecurity risks in a pandemic. *J Med Internet Res*, 22(9). e23692. DOI: 10.2196/23692.

Xing, T., Huang, D., Xu, L., Chung, C., and Khatkar, P. (2013). Snortflow: A openflow-based intrusion prevention system in cloud environment. In *GENI Research and Educational Experiment Workshop*, pages 89–92. DOI: 10.1109/GREE.2013.25.

Zanna, P., O'Neill, B., Radcliffe, P., Hosseini, S., and Hoque, M. S. U. (2014). 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*, pages 1–5. DOI: 10.1109/NOF.2014.7119792.

Zolotukhin, M., Kotilainen, P., and Hämäläinen, T. (2021). Intelligent IDS chaining for network attack mitigation in SDN. In *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*, pages 786–791. IEEE. DOI: 10.1109/MSN53354.2021.00123.