A Cooperation-Aware Virtual Network Function for Proactive Detection of Distributed Port Scanning

Igor Jochem Sanz, Martin Andreoni Lopez,
Diogo Menezes Ferrazani Mattos, and Otto Carlos Muniz Bandeira Duarte
Universidade Federal do Rio de Janeiro – GTA/COPPE/UFRJ

Emails: {sanz,martin,menezes,otto}@gta.ufrj.br

***Abstract*—One of the strongest defenses from cyber-threats today is the use of intrusion detection systems. Port scanning is usually the first action that precedes an intrusion. In turn, the use of virtual network functions (VNF) for cloud computing has become a powerful tool for tenants to provide network functions in high-speed networks. In this paper, we propose a virtual network function to detect distributed port scanning based on a cooperative architecture and on the programmable open source intrusion detection system Bro. The contribution of this paper are fourfold: i) the detection of ACK and NULL scan techniques; ii) the detection of the scan techniques TCP Connect, SYN, FIN, XMAS, ACK and NULL performed in a slow and distributed manner; iii) an architecture for cooperation between VNFs that shares historical logs of scans to improve scan detection in the cloud; iv) an implementation of a prototype of the proposed VNF in the Open Platform for Network Function Virtualization (OPNFV). Our prototype uses the Network Function Virtualization architecture from ETSI and respects the Service Function Chaining standards from IETF. We evaluate our prototype and the results show that we are able to detect all port scanning techniques with a high precision rate.**

***Index Terms*—Virtual Network Function, Network Function Virtualization, Software-Defined Networking, Port Scanning, Intrusion Detection Systems, Cloud Computing, OPNFV**

I. Introduction

Advances in information technology and, mainly, Internet of Things increase the number of vulnerabilities and security breaches in connected devices [1]. Attackers and malicious users exploit these breaches for malicious activities, making even harder to maintain the network secure. Traffic monitors and Intrusion Detection Systems (IDS) are fundamental to provide network security, but at the same time a challenge to analyze data traffic from a large number of devices. Port scanning is often the first action to precede an attack on a target host and those cases are estimated to be 50% of all intrusion attacks [2]. Port scanning is a method created for probing a host for open ports and services. Network administrators scans hosts to check network security policies or by attackers to find active and vulnerable services on them. Well-known vulnerabilities are mitigated by maintaining services updated, however, zero-day attacks exploit unknown vulnerabilities and are a great challenge for cyber security systems. Although techniques of anomaly detection can be applied to face this problem, they present high rate of false positives. Thus, preventing malicious port scans is extremely important for the security of the network. Most of intrusion detection systems can detect and prevent simple port scans. Attackers, however, can perform a stealth scan by do not handshaking a TCP connection to host, a slow scan by increasing the time between each port verification or a distributed scan assisted by botnets, and combine them together to make the scan almost undetectable [3].

At the same time, technologies as Network Function Virtualization (NFV) and Software-Defined Networking (SDN) allow the development and implementation of network functions, like Domain Name Systems (DNS), Firewall and Intrusion Detection Systems, in a software manner instead of using dedicated middlebox hardwares [4]. Virtual Network Functions (VNF) now can be flexibly escalated, instantiated and migrated in the cloud. Furthermore, coherent chains of VNFs can be designed to meet applications requirements and this is named as Service Function Chaining (SFC). To accelerate the development and the delivery of VNFs, as well as its standardization, the Open Platform for Network Function Virtualization (OPNFV) was created[1]. The OPNFV uses the NFV Management and Orchestration (NFV-MANO) architecture standardized by ETSI [5], implements the SFC architecture described by IETF [6], and adopts the Network Service Header (NSH) protocol [7] as standard for SFC encapsulation.

In this paper, we propose a Virtualized Network Function for detecting distributed port scanning in a cloud environment using OPNFV cloud. By analyzing the incoming traffic to a server using Bro framework, we are able to detect port-scanning activities on the network. We modify the original Bro script of scan detection to add capabilities for detecting FIN, XMAS, ACK and NULL scans, as well as the original ones for SYN and TCP Connect() scans. We modify and add some events to increase Bro detection sensibility. Slow port scan can be detected by modifying a few variables related to the time threshold of scans. The distributed port scanning is detected by adding new policy of observation for probes coming from different hots. We validate our virtualized network function by testing it on two different scenarios. First, we use an attacker VM to scan a target VM in a vertical manner, i.e. scanning a large range of ports on the target host. In this scenario, we prove our capability of detecting all six techniques. Bro server

[1]Available at http://www.opnfv.org/.

acts as a VNF-IDS between them analyzing the incoming traffic. We also verify the detection over a real dataset from CAIDA UCSD Anonymized 2016 Internet Traces [8] as background network traffic. We repeat both cases with slow port scanning and compare its results. The second scenario consists in performing horizontal and vertical scans from different attackers IP, in order to detect distributed probes. The results from our experimental test show that our VNF was capable of detecting TCP Connect, SYN, FIN, XMAS, ACK and NULL port scans in all the environments created, including those performed in a distributed way.

The remainder of this paper is organized as follows: In Section II, we compare our contribution with previous related work. We describe the experimental procedure of the testbed in section III. The evaluated results are shown in Section IV. Then, we conclude the paper in Section V.

II. Related Work

Paxson developed Bro IDS, a system for detecting network intruders in real-time [9]. Bro is a real-time traffic analyzer and it got notable for its main characteristics: high-speed monitoring, no packet dropping, real-time notification, extensible and easy script language. In our previous work [10], we use Bro IDS combined with OpenFlow to detect Denial of Service (DoS) attacks. We proposed an algorithm implemented in POX network controller, to block DoS from it source. However, BroFlow do not deal with port scan threats.

An intrusion detection method based on a distributed cooperative model was proposed by Zhang *et al.* [11]. The model is divided in five layers: sensor, event generator, event detection agent, fusion center and control center. The intrusion detection can occur in three different manners: feature-based, scenario-based and statistics-based. The authors claim that the method has improved intrusion detection capabilities for scan attacks. Nevertheless, their proposal do not detect slow scan attacks. Slow scans are hard to detect with traditional security tools. Dabbagh *et al.* propose a way to detect slow TCP port scan [12]. The work collects loads of traffic data over a long time window and extracted features of every TCP connections or connection attempts on it. The software was tested using three different slow port scan intervals: 0.4ms, 4 min and 6 min. However, using the number of connections as a feature generates many false positives in a live network traffic due to the large number of port scans performed by non malicious applications and administrators. Larsen described a modified scan script for Bro IDS capable of improving original Bro scan capabilities [13]. The new script was able to detect SYN, TCP Connect, FIN and XMAS slow scans over a background network traffic. It was validated by deployment on a baremetal simulated network. Larsen also compared its Bro with Snort IDS, and concluded Bro has a better efficiency over scan attacks. Shao *et al.* developed a new method to detect slow scan based in collecting information from IP packets during a window time and applying fuzzy set and Dempster–Shafer evidence theory [14]. The proposed combination is able to detect slow scan attacks, but do not differentiate between scan attacks.

In this paper, differently from all related work, we propose a virtual network function for distributed port scan detection based in a cooperative architecture for VNFs. Our VNF relies on the Bro IDS with a improved script for scan detection that detect different techniques of port scanning, like TCP Connect(), SYN, FIN, ACK, NULL and XMAS scans, as well as slow scans. We also propose a cooperation-aware architecture for the VNF that is based on sharing historical logs of scans, thus enhancing scans capabilities. Therefore, the proposed VNF can detect both horizontal and verticals scans performed by an attacker, even in a distributed way.

III. Virtualization and Chaining of Network Functions

Virtualization of network functions is applying the concept of cloud computing on the domain of telecommunications operator networks [15]. To meet similar performance of hardware middleboxes requirements, advances in cloud computing, like multiple hypervisors, hardware assisted virtualization [16], cloud operational systems and efficient software-based switches, have contributed on the software implementation of Virtual Network Functions (VNF).

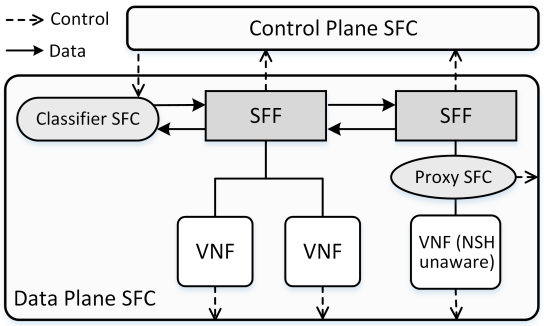


Fig. 1. Main components and the control communication in the Service Function Chaining architecture defined in RFC 7665.

Figure 2 shows how the NFV infrastructure (NFVI), from the NFV Management and Orchestration (NFV-MANO) architecture, can compose end-to-end microservices tailored for each application [5]. NFVI provides abstraction for processing, storage and networking of virtual network functions. Furthermore, the packet forwarding control and the abstraction of the infrastructure in a forwarding graph of virtual functions can be performed in a flexible manner by implementing a Software-defined Networking (SDN) controller [17]. SDN complements NFV technology in the VNF management by logically centralizing the network control, thus facilitates the chaining of VNFs.

Service Functions and Network Functions are defined as synonyms in this work and the coherent sequence of VNFs to meet a predefined goal is named Network Service. The key idea behind Service Function Chaining (SFC) is to offer functionalities through the chaining of VNFs that execute

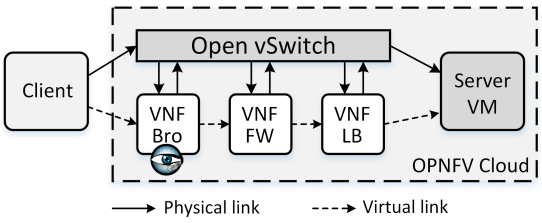


Fig. 2. Example of a chain of a VNF Bro, a VNF Firewall (FW) and a VNF load balancer (LB) that can be composed through the Service Function Chaining architecture.

specific functions in the network layer, like Intrusion Detection Systems, Firewalls, Proxy etc. These virtual network functions supplant the existing hardware middleboxes today. The basic elements of the SFC architecture, according to the RFC 7665, are shown in Figure 1. In this architecture, the virtual path of a service, that comprises an ordered sequence of VNFs, is defined as Service Function Path (SFP). In order to the traffic of a given service to traverse the desired SFP, a flow classifier is configured with specific rules determining which SFP each flow must traverse. In this way, traffic of different microservices can traverse simultaneously the same VNFs, but following different logical path, while being isolated according to the path identifier of each flow. VNFs may also be hosted on different physical nodes. Therefore, the Service Function Forwarder (SFF) is a necessary element in each NFVI node that provides the virtual links to its guest VNFs. In turn, VNFs may or may not be aware of the SFC encapsulation. VNFs that are not aware of the encapsulation can operate normally as long as they are preceded by an element that decapsulates packets. This element is named SFC Proxy. In the implementation of this paper, elements of the SFC architecture are constructed based on rules through the Open vSwitch virtual switch using the OpenFlow protocol. These rules are managed by the OpenDaylight SDN controller allied to the VNF manager and orchestrator Tacker.

To dynamically forward flows across different virtual service paths, such for load balance or granular and accurate policy control, an encapsulation is needed to tag packets. The OPNFV platform specifies the Network Service Header (NSH) for the SFC encapsulation [7]. NSH protocol provides visibility of the end-to-end SFP over the UDP transport protocol and allows classifying and reclassifying flows on each hop between VNFs. The traffic is routed based on two main NSH fields: Service Path Identifier (SPI) and Service Index (SI). The former is responsible for identifying which SFP path to traverse, whereas the latter is an index that identifies the current position in the chain. SI field works as an 8-bit iterator that decreases a unit of 255 each hop on service functions until it reaches a value relative to the final position on the chain. An advantage of NSH encapsulation is to exchange metadata between VNFs, which is possible due to the existence of context fields. This is a positive aspect in case of mobile or home users Internet access provision where the context may contain client or tenant identifier in order to allow contract-based policies [18].

IV. Port scan techniques

Port scanning can be differentiate between many aspects. First, a port scan can be either benign, when network tenants want to check network status or applications require some ports allocation to communicate by, or malign, when malicious users retrieve services information usually seeking vulnerabilities or in malware spreading.

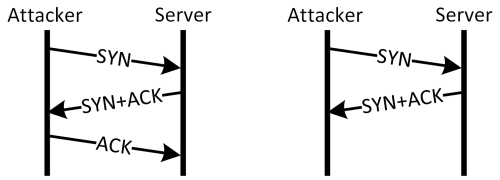
The gathered information from scans can vary between active ports number, service versions and even host operational system. Thus, scanning can be performed in two main perspectives, vertically or horizontally, depending of the scanner objectives. Vertical port scan is defined when a range of ports are probed on a single target machine. Hence, attackers perform this scan when they have a specific target and want to investigate how to exploit it, that includes most of cases of scanning [19]. Whilst in the horizontal scan, a set of hosts are probed for the same port. Attackers scan horizontally when they have knowledge of a specific exploit and want to identify vulnerable victims. The latter is harder to detect by local mechanisms and requires a global view of the network or a sort of cooperation between multiples IDSs. Moreover, vertical and horizontal scans combined derive a scan that can retrieve massive amount of service information for an entire network mask and this is known as block port scan [20].

To circumvent intrusion detection systems, attackers have developed strategies to prevent their scan to be detected. A wise one consists in performing it in a slow manner, i.e. a long time interval between each port probe. This interval can be long enough to exceed the IDS scan threshold, thus probes are disguised as legitimate traffic. Another strategy is to perform a distributed port scan, which the attacker has different source IP for each port probe. This is useful to bypass defense mechanisms in vertical scans or to be stealth in horizontal scans, however, it demands lot of computational resources from attacker, like a botnet.

There are many different techniques for port scanning available. In this paper, we approach the six most common port scan techniques: SYN, ACK, FIN, NULL, XMAS and TCP Connect() scans. The easiest way to port scan is completing the 3-way-handshake of a TCP connection with the target through the system call `connect()`, as shown in Figure 3(a). The port is open if the three-way handshake is successful and closed otherwise. Although it is simple to perform, a successful TCP connection leaves traces and create logs on server, what its usually not intended by attackers. A clever way to circumvent this is by do not concluding the connection, i.e. not sending the last ACK. This is known as SYN or stealth scan and is shown on Figure 3(b).

Other variations for non-traceable scans are possible by setting one or a combination of TCP flags, like FIN, ACK[2], NULL or XMAS (FIN+URG+PUSH). By understanding victim response from each technique, attackers can have total or

[2]ACK scan is a particular type of scan which does not identify between open or closed port, but between filtered by firewall or unfiltered port.



(a) TCP handshake. (b) SYN Scan.

Fig. 3. Two basic techniques to perform a port scan: a) TCP connection handshake and b) stealth or SYN scan.

partial information from victim running services. Table I resumes these techniques and the victim response interpretation for each one.

In our proposed VNF, we focus on the detection of vertical and horizontal of all the six scan techniques, considering they can be performed in a distributed and/or slow manner.

TABLE I
INTERPRETATION OF VICTIM RESPONSE FOR DIFFERENT SCANS.

Scan	Flags Set	Open	Closed
connect()	-	success	failure
SYN	SYN	SYN, ACK	RST
FIN	FIN	none	RST
XMAS	FIN, PSH, URG	none	RST
ACK	ACK	RST (unfilt.)	none (filt.)
NULL	none	none	RST

V. THE COOPERATION-AWARE VNF ARCHITECTURE

The proposed VNF is divided in several modules. Figure 5 shows the VNF architecture that is cooperation-aware as it shares historical information among others VNFs to increase detection capabilities. This cooperation relies on a Master instance of the VNF and the VNF main core scripts for port scanning detection.

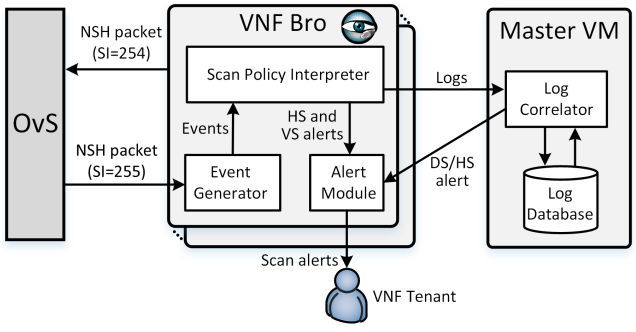


Fig. 4. Architecture of the proposed VNF and the Master VM instance that correlate logs from multiples VNF Bro.

The Master VM is responsible for a historical correlation of scans among all the VNFs Bro and it is initialized when the first VNF Bro is instanced in the cloud. It is composed of two modules, a log correlator and a log database. The log correlator module consists in a sliding window that represents the threshold for a slow scan detection, where its size is the defined time threshold. In this module, it is compared all potentially probes inside the sliding window regarding to the characteristics that define vertical, horizontal, and distributed scans. These characteristics are or probes that comes from the same attacker and are destined to the same port or the same host, or in case of distributed scans, probes that comes from different attackers destined to the same port or the same address. Figure V illustrates how the sliding window works. If a set of probe with the same characteristics is greater than the probe threshold inside the sliding window, then a scan is detected and an alert is generated to all Bro VNFs running in the cloud. This procedure provides a proactive detection of port scanning, once many client VMs behind the VNFs can be aware of a potentially scan before being scanned. The proactive detection is particularly useful against horizontal scans, once a chain of a VNF firewall with a Bro VNF can build up an automatic mitigation of the scan.

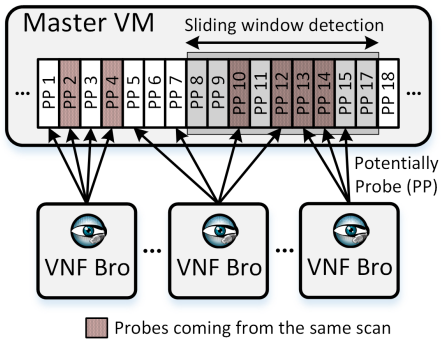


Fig. 5. The log correlator module of the Master VM. Multiple VNFs in the cloud send scan logs that is analyzed by Master VM inside a time threshold sliding window regarding to the characteristics that defines the scans.

The architecture of the core of the VNF is based on Bro IDS framework, an open-source software for real-time traffic analyzing. Bro is vastly used in scientific literature due to its notable characteristics of high-speed monitoring, extensibility and easy script language. Bro uses `libpcap` C++ library and is composed of and event engine and a policy script interpreter. The first is responsible to resume filtered packet streams in events. These events can be programmable to execute specific functions when triggered by the event generator. The latter processes events generated and interpret the actual script correspondent to each event. Those scripts can create or trigger new events, generate logs and execute a specific code.

The main script behind Bro policies for scan detection is an extended version of the original `scan.Bro` script[3]. We modify this script to add the capabilities of detecting both horizontal and vertical distributed port scans, as well as detection of new techniques of scan. Our scan detection

[3]The original scan.Bro script is available at https://www.Bro.org/sphinx/_downloads/scan.Bro.

consists in a threshold-based approach. The basic flowchart describing this approach is shown in Figure 6.

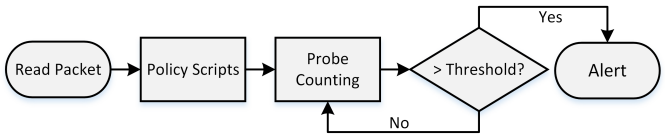


Fig. 6. Flowchart of the proposed port scanning detection.

In order to detect slow port scans, we configure Bro for a high time threshold between scans. Hence, we prevent both horizontal and vertical slow scans. First, we add the global variable declarations for the time threshold and the number of probes threshold of vertical and horizontal distributed scans. We also declare two new policies hooks, `dist_port_scan_policy` and `dist_addr_scan_policy`, in order to evaluate potentially probes of both distributed scans.

```
export {
 const dist_addr_scan_interval = 10min;
 const dist_port_scan_interval = 10min;

 const dist_addr_scan_threshold = 10.0;
 const dist_port_scan_threshold = 10.0;

 global Scan::dist_port_scan_policy:
 hook(scanner: addr, victim: addr,
 scanned_port: port);
 global Scan::dist_addr_scan_policy:
 hook(scanner: addr, victim: addr,
 scanned_port: port);
}
```

Some improvements are done in the function `add_sumstats()`. This function is responsible for summarizing all potentially scan data, by checking if the number of probes with a determined characteristic passes the scan threshold. We added two new cases for analyzing potentially scans in case the probes comes from different hosts, distributed horizontal scans and distributed vertical scans. The modifications are shown below

```
function add_sumstats(id: conn_id,
reverse: bool, scanType: string){
 if (hook Scan::dist_addr_scan_policy(
 scanner, victim, scanned_port))
  SumStats::observe("dist.scan.addr.fail",
  [$str=cat(scanned_port)],
  [$str=cat(scanner)]);
 if ( hook Scan::dist_port_scan_policy(
 scanner, victim, scanned_port) )
  SumStats::observe("dist.scan.port.fail",
  [$str=cat(victim)],[$str=cat(scanner)]);
}
```

The first `if` statement calls a hook for the policy `dist_addr_scan_policy()`, by observing, i.e counting, the number of packets that comes from different hosts and goes for different address using the same port. The latter checks for packets that comes from different host and probes different ports in the same host, in order to trigger a threshold of vertical distributed scans. To adapt the code to the new policies created, we use two local `SumStats::Reducers` to count the number of unique probes that matches specific characteristics of distributed scans. Hence, we create the function that analyzes if the number of probes in a unique set is greater than the threshold with the `SumStats::Create` function.

We also add and modify some events in Bro, in order to increase the sensibility of scan detection. One of the modification is on the `connection_pending()` event by adding the following statements.

```
event connection_pending(c: connection){
 if ((c$orig$state == TCP_CLOSED ||
 c$resp$state == TCP_CLOSED) &&
 ("a" in c$history || "A" in c$history)){
  add_sumstats(c$id, F, "ACK");
 }

 if ((c$orig$state == TCP_INACTIVE ||
 c$resp$state == TCP_INACTIVE) &&
 ("a" in c$history || "A" in c$history)){
  add_sumstats(c$id, F, "ACK");
 }

 if ((c$orig$state == TCP_CLOSED ||
 c$resp$state == TCP_CLOSED)
 && strcmp(c$history,"") == 0){
  add_sumstats(c$id, F, "NULL");
 }
 if ((c$orig$state == TCP_INACTIVE ||
 c$resp$state == TCP_INACTIVE)
 && strcmp(c$history,"") == 0 ){
  add_sumstats(c$id, F, "NULL");
 }
}
```

These statements are responsible for detecting ACK and NULL scans in the incoming packets. It checks if a packet has an ACK flag set up and if the connection is closed or inactive. If true, it calls the function `add_sumstats()`. Similarly, if the packet has no flag and the TCP connection is closed or inactive, the script considers the packet as a potentially null scan and call the function `add_sumstats()` as well.

We also include the event `new_connection_contents()`, as shown below.

```
event new_connection_contents(c:
connection){
 if ( (c$orig$state == TCP_CLOSED ||
 c$resp$state == TCP_CLOSED)
```

```
 && ("F" in c$history || "f" in
 c$history)){
   add_sumstats(c$id, F, "FIN/XMAS");
 }
}
```

The inclusion of this event is to add FIN and XMAS detection capabilities. It checks for FIN flag set in the packet in case the connection is already closed. If triggered, it calls the function `add_sumstats()`.

To adapt the VNF to the SFC architecture, the VNF has also a component that decapsulates and encapsulates incoming NSH packets, based on the open source Python software `vxlan_tool`[4]. This software is extended to mirror the decapsulated packet to a virtual interface, allowing Bro to inspect incoming packets. The application sends the packet back to the Open vSwitch with SI field decreased by one. Meanwhile, Bro reads the mirrored packet and checks if it is part of a potentially scan.

VI. Implementation and Results

We use the Open Platform for Network Function Virtualization (OPNFV) on the Danube 3.0 version to implement the prototype of our VNF. This platform implements the reference architecture NFV-MANO based on the Openstack cloud. The environment is deployed through a Fuel installer over 3 controllers nodes and four compute nodes.

The configuration for each node used to build the cloud is described on Table II. It consists of three controllers nodes in redundant mode, *Arraial, Sossego and Pao de Acucar*. These controllers manage the layer 2 network of all VMs instances along with the OpenDaylight (ODL) controller that is installed on the Arraial node. The controllers allow tenants to have access to the cloud management interface Horizon. Three nodes compose the computing cluster of the cloud - *Ilha Grande, Maua* and *Mangaratiba*. The compute nodes provide the resources (cores, disk and memory) for all instances in the cloud. All nodes, except the SDN and VNF controller Arraial, work for the Ceph OSD distributed file system, making up the storage cluster.

Inside the OPNFV cloud, our first testbed topology consists of four instances and two networks that we identify as trusted and untrusted, interconnected through a router. In the trusted network, we have the target instance (VM) which will receive both background traffic and the port scan attack. Then two instances, one responsible to do the background traffic injection and the other to perform the attack, make up the untrusted network. Both networks are interconnected through a fourth instance, the Bro VNF, that also acts as a router. The Bro VNF is responsible to provide the static route between both trusted and untrusted networks. It will provide the path for all the CAIDA dataset and the scan attack to hit its final destination. We then initialize Bro VNF to analyze the incoming traffic, and at the same time, we perform port scanning from the

[4]Available at https://github.com/opendaylight/sfc/blob/master/sfc-test/nsh-tools/vxlan_tool.py.

TABLE II
OPNFV Nodes Roles and Hardware Configuration.

Node Name	Roles	Configuration
Arraial	Controller Tacker ODL	Intel i7 Quadcore, 3TB disk, 64GB RAM
Sossego	Controller Storage	Intel i7 Quadcore, 3,6TB disk, 20GB RAM
Pao de Acucar	Controller Storage	Intel i7 Quadcore, 3,6TB disk, 20GB RAM
Mangaratiba	Compute Storage	Intel Xeon E5 2650, 12TB disk, 512GB RAM
Maua	Compute Storage	Intel Xeon E5 2650 3,6TB disk, 32GB RAM
Ilha Grande	Compute Storage	Intel i7 Quadcore, 3TB disk, 24GB RAM

Attacker VM on the Target VM. Figure 7 shows the topology of this first scenario of detection tests.

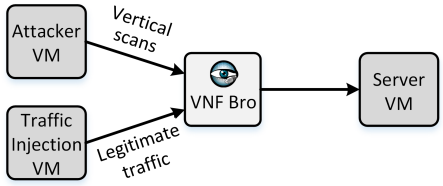


Fig. 7. First scenario of tests. Attacker VM performs multiple vertical scans in a server traversing the VNF Bro in a real traffic environment.

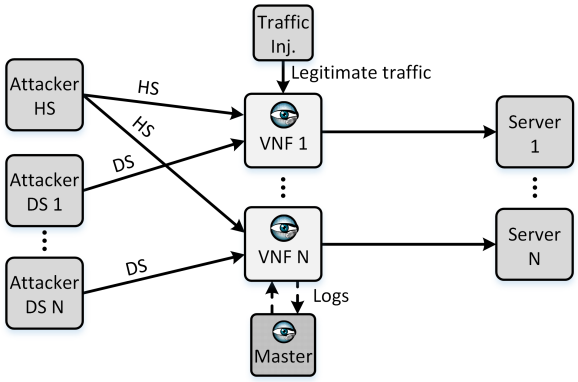
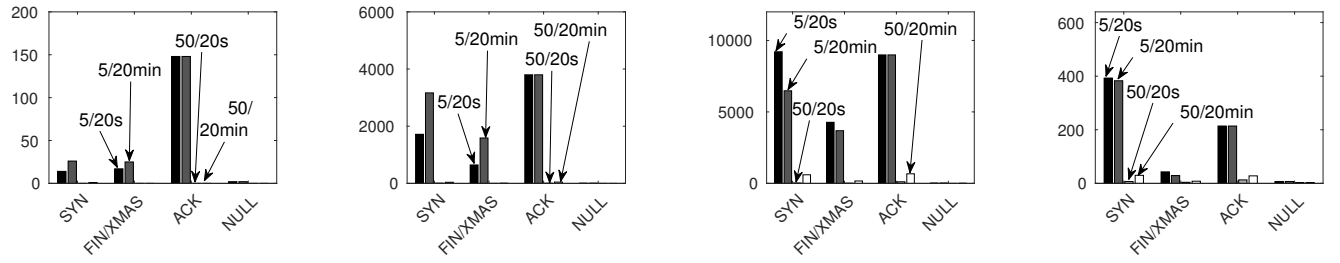


Fig. 8. Second scenario of our testbed. Vertical and horizontal distributed scans are performed from different attackers. HS: Horizontal scan. DS: Distributed scan.

The objective with this scenario is to verify the functionality of detecting different techniques of scan while real traffic is injected in the network by another VM. Bro current version 2.5.1 was used and installed on the Bro VNF. The main reason we choose Bro for our experiment, instead of others IDS, is its customization capability according with our network, that makes Bro more effective when comparing with others IDS like Snort [21]. We configured Bro logs files for the maximum



(a) Vertical scans. (b) Horizontal scans. (c) Distributed vertical scans. (d) Distributed horizontal scans.

Fig. 9. Detection results for SYN, FIN, XMAS, ACK and NULL techniques when analyzing the CAIDA dataset with our Bro VNF.

level and changed a some threshold variables to increase port scan detection rate. We change Bro local site policy to adapt it for our network configuration, modifying the `local.Bro` file as follows

```
redef Site :: local_nets = {
        10.20.0.0/16, #Untrusted network
        146.164.69.0/24 #Trusted network
 };
```

then we start Bro in the VNF with the following command.

```
$ Bro local.Bro -i eth0 -w data.pcap
```

This command starts Bro IDS using the local user configuration file over the network interface eth0 and save the network data in the file `data.pcap`. In both scenarios, we record a pcap file containing all data that traversed the VNF for further analysis.

We use the CAIDA dataset `equinix-chicago.dirA.20160406-130500.UTC.anon.pcap` as real traces for background traffic [8]. To prepare the dataset for our network configuration, we slightly modified the original dataset characteristics, like the source MAC address of the whole dataset to the VM that replays the traffic. Then, the `tcpreplay` framework is used to change source mac address from our dataset, ant to replay it into our network.

`Nmap` scan tool is chosen to perform the scan attacks. The following command is an example of a single slow port scan using the SYN scan technique in `nmap`.

```
$ nmap -sS --top-ports 100 -P0 -T0 \\
--scan-delay 20s --max-scan-delay 21s \\
10.20.10.1 --packet-trace
```

In this example, `nmap` starts a port scan attack using SYN scan for the 100 most used system ports with 20 seconds of scan delay on each probe.

We repeat this procedure for all types of scan Attacks SYN, FIN, TCP Connect, NULL, ACK and XMAS. After obtaining results in a clean network, we repeat all those tests over a background traffic injected by the Traffic Injection instance.

In the second scenario, we try to evaluate the detection of distributed port scans. Hence, we need to perform a scan that each probe comes from different IPs, similar to an attacker orchestrating a port scan from a botnet. To simulate such attack, we use multiple VMs in the cloud and the topology of this scenario is illustrated in Figure 8.

We chose four set of parameters to demonstrate the feasibility of our scan detection. Table III shows the configuration used for each one. In both scenarios described above our VNF managed to detect all six techniques TCP Connect(), SYN, FIN, XMAS, NULL and ACK, for horizontal and vertical scans, as well as slow and distributed scans, with 100% of precision.

TABLE III
THE SET OF PARAMETERS CHOSEN IN OUR BRO VNF ANALYSIS.

	Probe/Time Threshold			
Values	5/20s	5/20min	50/20s	50/20min

Figure 9 shows the total of scans detected when analyzing CAIDA dataset of real Internet traces with Bro VNF. We noted that vertical SYN scan is the most used scan technique in this dataset and also that ACK scans have a high rate of false-positive due to the large amount of ACK packets sent on the Internet. The VNF was able to detect all types of scans, but with a high rate of false positives for distributed scans depending on the threshold values. These high rate of false positives is due to the intrinsic characteristic of the Internet: hosts with a large number of different services and clients, and common services that run on many hosts that are accessed by a a large number of clients. However, we can minimize this rate by do not considering common ports, as dns/23, http/80 and https/443, as scan traffic.

The final detection results is shown on Table IV, where we also compare the contribution of this work to related work for Bro IDS scan detection. Here, VS represents vertical scans, HS are horizontal scans, DHS are distributed horizontal scans and DVS are distributed vertical scans.

VII. CONCLUSION

In this paper, we proposed a virtual network function architecture for detecting both horizontal and vertical distributed port scans in the Open Platform for Network Function Virtualization cloud. The detection is based on Bro IDS with

TABLE IV
BRO SCAN DETECTION RESULTS AND THE CONTRIBUTION COMPARISON TO RELATED WORK.

	Original Bro script					Larsen [13]					This proposal				
Scan type	**VS**	**HS**	**DVS**	**DHS**	**Slow**	**VS**	**HS**	**DVS**	**DHS**	**Slow**	**VS**	**HS**	**DVS**	**DHS**	**Slow**
TCP Connect	✓	✓	-	-	-	✓	✓	-	-	✓	✓	✓	✓	✓	✓
SYN	✓	✓	-	-	-	✓	✓	-	-	✓	✓	✓	✓	✓	✓
FIN	-	-	-	-	-	✓	✓	-	-	✓	✓	✓	✓	✓	✓
XMAS	-	-	-	-	-	✓	✓	-	-	✓	✓	✓	✓	✓	✓
ACK	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓
NULL	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓

improvements on scan detection scripts for monitoring the incoming traffic to a target server. We enhanced the original Bro scan detection script to detect new scan techniques performed in a slow or distributed way. We also proposed a cooperation architecture of VNFs on the cloud, to enhance horizontal and distributed port scans detection, i.e. when the same port is scanned for different hosts or from different attackers. The port scanning detection is evaluated over two different environments: a clean network with only scan traffic and a real traffic network. Our VNF was able to detect horizontal and vertical scans of TCP Connect, SYN, FIN, XMAS, ACK and NULL techniques performed on all environments, even in a slow and distributed manner. Furthermore, we verified a high rate of false positives for distributed scans due to the intrinsic characteristic of the Internet: hosts with a large number of different services and clients, and common services that run on many hosts that are accessed by a a large number of clients. This rate can be minimized considering the most used ports in the Internet traffic as legitimate traffic.

As future research, we will investigate port scanning detection over encrypted traffic, in order to preserve cloud users privacy and to adapt to new communication paradigms.

ACKNOWLEDGMENT

This research is supported by INCT of the Future Internet, CNPq, CAPES, and FAPERJ.

REFERENCES

[1] D. M. F. Mattos and O. C. M. B. Duarte, "Authflow: authentication and access control mechanism for software defined networking," *Annals of Telecommunications*, vol. 71, no. 11, pp. 607–615, Dec 2016.

[2] Pankwani, S. and Tan, S. and Jarrin, K. *et al.*, "An experimental evaluation to determine if port scans are precursors to an attack," *In Proc. Int. Conf. Dependable Systems and Networks*, p. 602, 2005.

[3] B. Claypool, "Stealth port scanning methods," *Global Information Assurance Certification Paper*, vol. 1.4, 2002.

[4] D. M. F. Mattos, O. C. M. B. Duarte, and G. Pujolle, "Reverse update: A consistent policy update scheme for software-defined networking," *IEEE Communications Letters*, vol. 20, no. 5, pp. 886–889, May 2016.

[5] ETSI, "ETSI GS NFV-MAN 001: Network functions virtualisation; management and orchestration," Tech. Rep., 2014.

[6] J. Halpern and C. Pignataro, "Service Function Chaining (SFC) architecture," Internet Requests for Comments, RFC Editor, RFC 7665, October 2015.

[7] P. Quinn and U. Elzur, "Network service header," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-sfc-nsh-12, February 2017.

[8] CAIDA. (2016) The CAIDA UCSD anonymized 2016 internet traces. http://www.caida.org/data/passive/passive_2016_dataset.xml. The Center for Applied Internet Data Analysis (CAIDA).

[9] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer Networks: The International Journal of Computer and Telecommunications*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[10] M. Andreoni Lopez, D. M. F. Mattos, and O. C. M. B. Duarte, "An elastic intrusion detection system for software networks," *Annales des Telecommunications/Annals of Telecommunications*, vol. 71, no. 11-12, pp. 595–605, dec 2016.

[11] W. Zhang, S. Teng, and X. Fu, "Scan attack detection based on distributed cooperative model," in *2008 12th International Conference on Computer Supported Cooperative Work in Design*, April 2008, pp. 743–748.

[12] M. Dabbagh, A. J. Ghandour, K. Fawaz, W. E. Hajj, and H. Hajj, "Slow port scanning detection," in *2011 7th International Conference on Information Assurance and Security (IAS)*, Dec 2011, pp. 228–233.

[13] R. Larsen. (2013) Slow port scanning with bro. MSc. Thesis, Department of Computer Science and Media Technology, Gjovik Univ. College.

[14] G.-l. Shao, X.-s. Chen, X.-y. Yin, and X.-m. Ye, "A fuzzy detection approach toward different speed port scan attacks based on dempster-shafer evidence theory," *Sec. and Commun. Netw.*, vol. 9, no. 15, pp. 2627–2640, Oct. 2016.

[15] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 216–223, Feb. 2017.

[16] D. M. F. Mattos, L. H. G. Ferraz, L. H. M. K. Costa, and O. C. M. B. Duarte, "Virtual network performance evaluation for future internet architectures," *Journal of Emerging Technologies in Web Intelligence*, vol. 4, no. 4, pp. 304–314, 2012.

[17] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, Feb. 2015.

[18] S. Kulkarni, M. Arumaithurai, K. K. Ramakrishnan, and X. Fu, "NeoNSH: Towards scalable and efficient dynamic service function chaining of elastic network functions," in *20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, Mar. 2017, pp. 308–312.

[19] T. Ahanger, "Port scan - a security concern", int. journal of engineering and innovative technology," *International Journal of Engineering and Innovative Technology*, vol. 3, no. 10, 2014.

[20] C. B. Lee, C. Roedel, and E. Silenok, "Detection and characterization of port scan attacks," *Univeristy of California, Department of Computer Science and Engineering*, 2003.

[21] P. Mehra, "A brief study and comparison of snort and bro open source network intrusion detection systems," *Internatinal Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*, vol. 1, 2012.