Lightweight IPS for Port Scan in Openflow SDN networks

Charles V. Neu*, Cássio G. Tatsch‡, Roben C. Lunardi*†, Regio A. Michelin*†, Alex M. S. Orozco*§, Avelino F. Zorzo*
*PUCRS, †IFRS, ‡UNISC, §IFSul - Brazil
E-mail: tatschcassio@gmail.com, {charles.neu, roben.lunardi, regio.michelin, alex.orozco}@acad.pucrs.br, avelino.zorzo@pucrs.br

***Abstract*—Security has been one of the major concerns for the computer network community due to resource abuse and malicious flows intrusion. Before a network or a system is attacked, a port scan is typically performed to discover vulnerabilities, like open ports, which may be used to access and control them. Several studies have addressed Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) methods for detecting malicious activities, based on received flows or packet data analysis. However, those methods lead to an increase in switching latency, due to the need to analyze flows or packets before routing them. This may also increase network overhead when flows or packets are duplicated to be parsed by an external IDS. On the one hand, an IDS/IPS may be a bottleneck on the network and may not be useful. On the other hand, the new paradigm called Software Defined Networking (SDN) and the OpenFlow protocol provide some statistical information about the network that may be used for detecting malicious activities. Hence, this work presents a new port scan IPS for SDN based on the OpenFlow switch counters data. A non-intrusive and lightweight method was developed and implemented, with low network overhead, and low memory and processing power consumption. The results showed that our method is effective on detecting and preventing port scan attacks.**

***Index Terms*—IPS, OpenFlow, SDN, Port Scan, Lightweight.**

I. Introduction

Nowadays the increasing number of devices connected to the Internet, along with the high number of transactions performed online, augment the challenge to keep devices and transactions secure. Attackers are constantly evolving their methods, materializing different kinds of attempts to steal or damage data or computer environments. Even when an attacker plans to invade a system or network, they may need some information about vulnerabilities that allow the intrusion to happen. Therefore, typically a scan attack is used. According to CERT [1], a Brazilian center that monitors incident reports, scan attempts represented around 60% of all the reported incidents in 2016.

In the literature, several studies address Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) methods for detecting malicious activities, like port scan, based on received flows or packet data analysis. However, those methods lead to an increase in switching latency, due to the need to analyze flows or packets before routing them, and the need for high processing power and memory consumption on high speed networks. This may also increase network overhead when flows or packets are duplicated to be parsed by an external IDS. In this context, an IDS/IPS may be a bottleneck on the network and may not be useful on a system or network.

978-1-5386-3416-5/18/$31.00

Software Defined Network (SDN) [2] are emerging as a new paradigm on computer networks, offering new opportunities to develop new IDS methods. Being managed by a centralized entity, called controller, and using a protocol, like Openflow [2], SDN provides features and resources that may be used by an IDS method. One such feature, provided by the Openflow protocol, is the switch statistical information stored in a counters table. Moreover, as the controller has a global view on the network and may exchange information with other controllers that manage other networks, a more sophisticated protection system can be implemented, extending protection to the entire network as well as to other networks.

Therefore, this work presents a new port scan IPS for SDN based on the OpenFlow switch counters data. These data are available to the controller and may be collected through specific OpenFlow messages on predefined time intervals. Our method uses these data to perform detection and then to block threats, being non-intrusive and lightweight. Furthermore, it does not overload the network and uses low processing power and memory. Thus, scan attempts may be prevented, improving security of SDN infrastructures.

This paper is organized as follows. Section II describes some background on SDN, Openflow, Port Scan, Intrusion Detection and Prevention Systems and discusses some related work. Section III presents a new solution to detect Port Scan. Section IV presents results and our evaluation. Finally, Section V concludes this paper and indicates some future work.

II. Background

A. Software Defined Networking - SDN

During the last decades, the network infrastructure and protocols evolution was restrained by some factors, such as vendor-specific technologies, compatibility with legacy technologies, and so forth. That network ossification is a significant barrier to innovation, and disruptive approaches are demanded to offer alternatives to this stagnation. Hence, the SDN concept arises as an opportunity to break the *status quo*, through the separation of control logic from the underlying network devices. Furthermore, the control logic is implemented in a logically centralized controller [3].

The SDN architecture can be split into 3 functional layers: Application, Control and Infrastructure. The Application layer is responsible for providing end-user networking management, analytic and business applications that consume the SDN communication and network services, such as load balancing and firewall. The Control layer (or Control Plane) is composed by a set of controllers that provide a control functionality, receiving instructions or data from the Application layer and relays them to the Infrastructure layer. It also extracts information from the Infrastructure layer (such as statistics and events) and communicates that back to the Application layer. The Infrastructure layer (or Data Plane) is composed by networking devices, which control the forwarding and switching for the network, including physical and virtual switches. Both Application and Control layers communicate through a northbound interface. Likewise, Control and Infrastructure layers communicate through a southbound interface [2].

The OpenFlow protocol is a *de-facto* open SDN southbound messaging standard that provides communication between the Control and Data planes. An OpenFlow switch is composed by flow tables, responsible for maintaining the information about flows. Each flow entry contains match fields, counters and a set of instructions to apply to matching packets. Openflow Switch Specification supports a set of counters, like transmitted and received packets, flow duration, received and transmitted drops and errors, collisions and flow count [2].

B. Port scanning

Port scan consists of sending several types of packets in order to know more about a target host or network. Through the answers obtained from analyzing these packets, the attacker is able to gather information that may help in future attacks. Some types of information that can be discovered include (not only): the activity of the servers, information regarding software used in the system, information about the firewall or network topology.

Due to their nature, scans can easily create a large number of different streams. According to Speroto *et al.* [4], there are three categories of scanning:

- horizontal scan - when a source host scans a specific port on different target hosts;
- vertical scan - when a source host scans several distinct ports of the same target host; and,
- mixed scan - when there is a combination of vertical and horizontal scans.

One way to protect the network from attacks is to monitor traffic for malicious activities or policy violations. In order to perform packet monitoring, the most appropriate solution is to use an Intrusion Detection System (IDS), which performs passive packets monitoring on the network. However, this type of analysis does not allow actions to be taken to prevent such attacks, and therefore, an Intrusion Prevention System (IPS) is required to block such attacks.

Several IDS proposals have been developed in the past. These proposals can be classified according to several characteristics, such as type of analyzed data (logs) or package data, type (real-time or offline) or by the type of processing (centralized or distributed). However, the best-known classification models are signature-based and anomaly-based [5].

A signature-based IDS performs detection by comparing packet data with packets stored in a database. Those systems perform packet analysis by checking the payload. However, packet inspection is difficult and even impossible to perform on networks with large bandwidth, *e.g.* several gigabits per second [6], or on encrypted packets.

An anomaly-based IDS, in turn, compares incoming data with a "normality model" that describes the normal behavior of the network. Significant changes on the behaviour of the flows, when compared with this model, are considered anomalies. Examples of models to represent the network behavior can use neural networks, statistical analysis techniques, and probability theory. The main advantage of this type of detection is that it also detects previously unknown flows [7]. However, there may be instances where flows may be different from the expected normality but not necessarily malicious, resulting in false positive alarms.

Considering this inflexibility on the current network equipment, concerns on abstracting network functions of switches dedicated to applications in SDN have been increasing. Security policies can be set by the controller as rules in the flow tables [8] instead of manual and independent configurations. Thus, a switch provides only the filter function according to the rule in the flow table, not significantly influencing the performance of the network. Furthermore, SDN has natural statistics features that are useful for intrusion detection analysis, so that the controller gets more visibility of the network traffic. Therefore, SDN seems to provide a more suitable architecture for IPS.

C. Related Work

Lagraa and François [9] proposed a solution to detect and to discover patterns of port scans. In order to analyze a large amount of data, they proposed a model to convert TCP packets into a graph containing packet information and their relations. Although the results presented are promising, performance and hardware requirements were not discussed.

Botie *et al.* [10] proposed an IPS to detect DDoS (Distributed Denial of Service) on SDN. Their approach consists of an anomaly detection based on an entropy-based algorithm. However, performance costs to implement it in an SDN Controller were not discussed.

Considering overhead to an SDN Controller, Zhang *et al.* [11] proposed a solution to mitigate port scans in SDN networks using a technique called Port Hoping. Their approach consists of dynamically mapping ports to unused ports. Consequently, it can confuse potential attackers and increase the attack cost. Although measured overhead was low (increase of 2.1%-4.9% on CPU usage), this approach did not stopped attacks. Consequently, attackers could be able to continue the attacks using other ports.

In order to help to prevent port scans with a low overhead, we present our proposed solution in the next section.

III. Protecting SDN from Port Scan attacks

This work describes a lightweight anomaly-based port scan IPS for SDN that uses the OpenFlow protocol through statistical OpenvSwitch data. On SDN, the controller has a global view of the network and may block malicious flows closer to their source. An application that implements our proposal was developed as an extension (plugin) to the OpenDaylight controller [12].

In the developed architecture, the controller manages and stores the rules that define packet forwarding on the network in addition to collecting statistics data from OpenFlow switches, as can be seen in Figure 1 and discussed in the next sections.

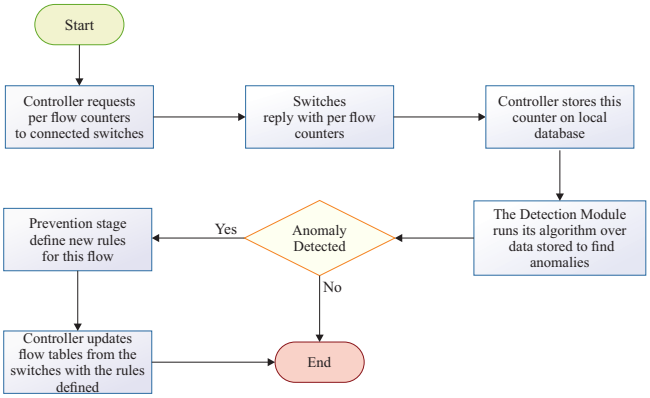

Fig. 1. IPS operation flow

Our solution was developed according to three basic stages. The first one is the collection stage, which performs data collection on the OpenFlow switches flow tables. Then, detection is performed on the collected data stage, and finally, in the third stage, the prevention method is implemented to react on attacks based on the detection stage results.

1) Collection: In the first stage, a job periodically (every 3 seconds) requests flow counters to the switches and stores them on a database. These requests collect data by using some OpenFlow methods to get detailed information about a particular flow, or a set of flows, which are defined on the OpenFlow specification [13].

Through an API, the controller sends a message to the switch asking for flow counters and the switch replies this message with one or more messages with all the counters data. On the controller, our application reads this data and extracts the information that will be used by the IPS. This information contains the source and destination addresses, ports and the number of packets from each flow.

The statistic collection interval is a critical point to be considered. On the one hand, if the collection is performed between wide periods of time, there will be a delay in the attacks detection. On the other hand, if the time interval is too narrow, there will be an increase of traffic related to collection requests. In this work, this interval was defined as 3 seconds, as it was considered the optimal time based on our test results, as this period did not cause any network overload.

2) Detection: To perform detection, the flow information stored on the database on the previous stage is analyzed. Each analysis consists on the following steps:

- Selection of stored flows that have a reduced number of transmitted packets (which characterizes port scanning flows) and that have been generated in the last 3 seconds;
- Grouping of the selected flows, by source and destination IP and destination port;
- Obtaining the number of flows with similar source and destination addresses in a small predefined interval (1s). Eventual port scan attempts may occur on any system, so a single stream cannot be considered an attack.
- Scan categorization (horizontal, vertical or mixed) based on source and destination address information, destination port and packet count.
- If a scan attack is detected, the source IP is added to a blacklist. All the flows from an address stored on this list are dropped.

Port scanning streams are typically limited to a maximum of 3 packets, for example TCP Connect has SYN, SYN/ACK and ACK packets, except in cases in which there are packets re-transmission. Furthermore, statistically, port-scan attacks occur at small intervals of time. Since there is a periodical collection of flow statistics, it is possible to analyze whether there was an increase on the number of packets between two queries.

Therefore, we consider that a stream is a scan attempt when the number of packets is equal or less than five. This number corresponds to the three packets responsible for the handshake of a TCP connection added with two tolerance packets, in case of re-transmission. Since this number of packets is very low, the probability of having a non-malicious flow detected as scanning is also very low.

Only one port scan is not considered an attack, so that streams should be considered attacks, some rules have been defined according to the type of scan performed: source IP addresses that have streams with number of packets lower than 6, and have at least 3 target hosts with same destination port can be considered an anomaly that represents horizontal port scanning. Thus, the source address is added to the blacklist.

The number of target hosts needed to characterize the attack can be adjusted according to the size of the network in which it will be used. For this work this number was defined as 3 hosts due to the limited size of the network used for testing. In the vertical scanning detection, it is assumed that a source host performs the scanning of several ports on the same destination host. In this case, source hosts that have streams for different ports on the same target hosts may be recognized by this application as an anomaly.

For this classification a weight-based method was created in order to allow greater sensitivity to the attacks for the most used ports. Typically, the most common probed ports, such as those of Telnet services [14], have a higher weight compared to other ports. We defined what ports have a higher probability to being scanned (22, 23, 25 and 3389) according the statistics of incidents reported by CERT.br [1]. These ports weight was defined as 5, while the weight of other ports is 3.

Our detection method computes the sum of the total weight from flows of the same source and destination. If this sum exceeds a threshold, the flows are considered as port scan, then the source IP is added to the blacklist. The detection threshold, on our tests, was defined as 15, but it can be adjusted as required by the network. We have used 15 as threshold to simplify the tests, being necessary 3 attacks to ports with more probability to be scanned and 5 attacks to the other ports to characterize the flows as an attack. When the sum of weights from flows of the same source and destination exceeds the threshold, a scanning attack may have been performed, and the source address is added to the blacklist.

Equation 1 is used for computing the amount of connections or attempts from a given source to be considered as a port scan, where CPS indicates the number of scans on common ports and OPS the number of scans on other ports. The calculated value TotalWeight is compared to the defined threshold for detecting a port scan attack.

$$\textbf{TotalWeight} = 5 * \textbf{CPS} + 3 * \textbf{OPS} \quad (1)$$

For mixed scans, both assumptions already mentioned must be considered. Our IPS analyses the source hosts that attempt to establish connections on at least two hosts and at least one of these hosts must also present a sum of vertical scan weights set to 6. In short, if 2 hosts have ports scanned and at least one of these has a different port scanned, it is considered a mixed scan attack.

3) Prevention: Initially, all switches have empty flow tables. Thus, upon receiving the first packet, a table miss will occur and the packet will be sent to the controller through a PacketIn message. On the controller, a PacketIn handler was developed to read those packets, analyze the header and extract the source and destination IP addresses and ports to compare with the data stored on the database. If no related entry is found in the database, a new one is created and added in the OpenFlow switch flow table to provide forwarding to the other packets related to the same stream.

Flow entries can be removed from flow tables through a request from the controller and by timeout. The timeout mechanism is executed by the switch independently from the controller and is based on the configuration and state of flow inputs. We defined a timeout for each stream to avoid overflow on flow tables. A flow will only remain on the table for fifteen seconds after the receipt of its last packet.

By defining forwarding rules, in addition to react to the attack, it also protects against new flows from the same source, without sending a request to the controller. Some prevention measures were also added on the packetIn handler, by default every non-malicious connection must be initialized with a SYN flag. Therefore, when receiving a packet with a flag different from SYN, the controller will immediately discard the packet, thus, preventing ACK, FIN exploitation and Xmas Tree attacks.

Flows added by the controller in the flow table have packet header information as the rules, so each received stream will have a single entry in the table. When inserting a flow in the flow table, a fifteen seconds timeout is set, which allows the flow to be kept in the table if there is a small delay in packets transmission. Table I shows a flow table with the rules used by our method. Unused rules were omitted from the table. In this example, the first three streams are forwarded

TABLE I
EXAMPLE OF SOME FLOW TABLE RECORDS,

MATCH RULES				ACTION	TIMEOUT
ADDRESS		PORT			
SRC	DEST	SRC	DEST		IDLE
10.0.0.11	10.0.1.124	36987	80	forward	15
10.10.0.45	10.10.2.43	23234	80	forward	15
10.0.0.114	10.10.2.29	35455	22	forward	15
10.10.2.24	10.0.0.12	32444	25	drop	0

and its idle_timeout was defined to 15 seconds, so that if no related packet is received in that time, the stream will be removed from the flow table. The fourth flow has as action its discarding, *i.e.*, when a packet is received and its header fields match the rules fields, this packet will be discarded. The fourth flow also has the timeout set to null, which means that this is a permanent rule and, if necessary, must be removed from the table by the controller through a specific message.

The global view nature of the network enabled by SDN allows the application to scan individual switches and send discarded streams to others, making it impossible for such streams to take different routes to the target host. In addition, different controllers can exchange information about port scanning flows, extending protection to other networks, as a collaborative intrusion detection mechanism.

IV. EXPERIMENTAL EVALUATION

To evaluate our method, some tests were performed. This section describes our test methodology, results and evaluation.

A. Environment and performance

For our experiments, a virtual network was configured on Mininet, a platform that allows the creation and emulation of scalable virtual networks with controllers, switches, hosts, links, and also enables the development of different topology through Python scripts. The Data plane was established using OpenvSwitch [15] virtual switches. On the Control plane, an OpenDaylight [12] controller was installed and configured on a remote computer.

The experimental network topology used in our experiments was modeled in a simplified way, as shown in Figure 2. It is composed by an OpenDaylight controller managing 4 switches. There are also 5 servers, running services, such as web servers and databases, and four computers that generate requests and port scans to the servers.

We performed the experiments on an Apple Macbook Pro computer, with 16GB of RAM (2133MHz), an Intel Core i7 @ 2.8GHz CPU and 256GB SSD. A Virtualbox machine with 4GB of RAM and 1 CPU core was configured on this hardware, with an Ubuntu 17.1 Operating System. An Opendaylight controller was installed on this virtual machine. This controller was used to test and to evaluate our IPS. In order to measure hardware consumption, we injected some

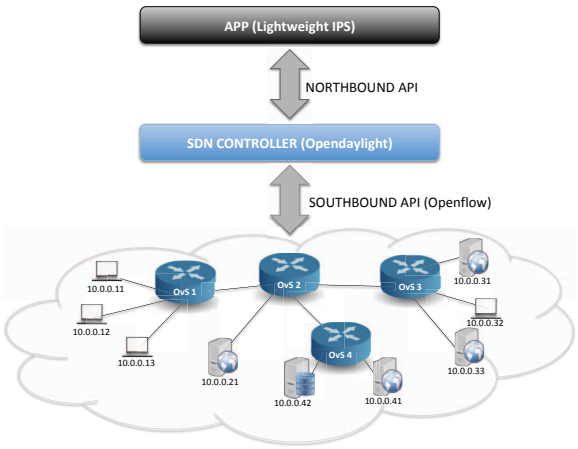

Fig. 2. Network topology.

networkd traffic and observed the controller CPU time and memory consumption without activating our IPS. After that, our IPS was activated and the same traffic was injected. We then computed the difference on hardware consumption to determine the amount of resources our IPS uses. When the controller was running without the IPS, it consumes 1.8% CPU and 29% RAM. With the IPS, the resource usage increases to 3.2% CPU and 32% RAM. Thus, the developed IPS consumes 1.4% of CPU and 3% of RAM from the used virtual machine, considering a 95% confidence interval. To achieve this confidence level, 17 repetitions were performed.

B. Performing Port Scan

To perform some port scan attacks on our network, we defined a threat model based on Attack Graphs [16], which is very useful for network administrators to identify system vulnerabilities, for example. Firstly, tests were generated to verify the possibility of detecting false positives, when normal requests are made to the servers. Therefore, a script that makes 200,000 requests to the different servers was created and executed. The number of packets vary between 10 and 3,000, depending on the request. Hence, the system must allow its routing, because none of the flows has a number of packets equal to or less than 5. By analyzing the generated logs and the blacklist, no entry of port scan attempt was found, showing that our system is did not detect any false positive in such situations.

After that, port scanning flows were generated using nmap. Initially, the same port was scanned on different hosts, performing a horizontal scan on the network, using the "nmap -p PORT ADDRESS" command. The vertical scan test was performed using the "nmap -sT ADDRESS" command, which scans distinct ports on a target address. The last scan returned 3 open ports (22, 53 and 80).

This test allowed the attacker to enhance a large number of scans at a lower interval than the collection because it was defined on 3 seconds. However, after this interval, new attempts will be blocked because the sum of streams with a number of packets equal to or less than 5 will be greater than the established threshold for being considered a scan.

Scanning tests were also executed without the SYN flag trough the "nmap -sf ADDRESS" command, which performs FIN scanning. In this case the controller has discarded the packets immediately and the scanning did not materialize.

C. Detecting and preventing port scan

After the tests, the statistic data stored in the database were compared to the originated flows. Table II shows flows stored in the database of a given host. The stored flow information is source and destination IP and ports, number of packets and an indication whether the entry corresponds to the request or to a response to the request that originated the flow.

In the example in Table II, host 10.0.0.11 originated 9 flows. Each flow has 2 entries on the table, indicating the destination and the request return. Through the column "Packets", a normal access behavior is shown in the first 4 flows, since the number of packets is greater than 5. Lines 9 to 14 shows that the number of packets of the respective flows is less than 5, characterizing a port scan attack.

According to the defined rules, each port scan has a weight of 5, and as the sum of these reaches the 15 threshold, the IPS defines these flows as being an attack, and discards them. After this, the attacker originated 2 more streams, which were discarded and information is stored in the database. This can be seen on the last two entries in Table II. They show an amount of one packet per stream, which corresponds to the SYN packet received by the controller.

TABLE II
SOME FLOWS INFORMATION STORED IN THE CREATED DATABASE.

SRC. ADDR.	DST. ADDR.	SRC. PORT	DST. PORT	PACKETS	DIRECTION
10.0.0.11/32	10.0.0.21/32	33666	80	103	REQUEST
10.0.0.21/32	10.0.0.11/32	8080	33666	96	RESPONSE
10.0.0.11/32	10.0.0.21/32	33668	22	65	REQUEST
10.0.0.21/32	10.0.0.11/32	22	33668	61	RESPONSE
10.0.0.11/32	10.0.0.21/32	33670	110	68	REQUEST
10.0.0.21/32	10.0.0.11/32	110	33670	62	RESPONSE
10.0.0.11/32	10.0.0.21/32	45985	110	68	REQUEST
10.0.0.21/32	10.0.0.11/32	110	45985	61	RESPONSE
10.0.0.11/32	10.0.0.31/32	56974	23	2	REQUEST
10.0.0.31/32	10.0.0.11/32	8080	56974	1	RESPONSE
10.0.0.11/32	10.0.0.31/32	26261	22	1	REQUEST
10.0.0.31/32	10.0.0.11/32	22	26261	1	RESPONSE
10.0.0.11/32	10.0.0.31/32	56974	25	2	REQUEST
10.0.0.31/32	10.0.0.11/32	25	56974	1	RESPONSE
10.0.0.11/32	10.0.0.31/32	48906	110	1	REQUEST
10.0.0.31/32	10.0.0.11/32	110	48906	0	RESPONSE
10.0.0.11/32	10.0.0.31/32	27695	110	1	REQUEST
10.0.0.31/32	10.0.0.11/32	110	27695	0	RESPONSE

D. Results

Table III presents a quantitative analysis of the flows. Non-malicious flows are shown on the second column. All the 200,000 generated flows were forwarded, with no false negative alarms.

The third column shows the horizontal scan results. For this, 20 distinct flows were generated. Since the validation is performed after scanning 3 distinct hosts, the maximum number of malicious streams to be forwarded is 12 (3 per target). However, some packages were re-transmitted and the IPS received both, original and re-transmitted packets,

recognizing these as two distinct flows and thus, the analysis resulted in a number of 14 streams. These streams were now considered malicious only in the fourth or fifth checked host. These values can be seen in the third column of Table III.

In the fourth column, which represents the execution of vertical scan, 1,000 malicious flows were generated. This generation, being automatic, was considerably faster than the collection interval, which resulted in the routing of 658 malicious flows. By the time our software terminated the analysis, they were considered as malicious source and were added to the blacklist and the system discarded the remaining 342 packets. In addition, this scan also had some duplicated packets, causing 12 malicious streams that were considered as non-malicious streams.

The last column shows the number of FIN exploitation scans. As explained previously, uninitialized streams with SYN packets are immediately discarded. All originated flows were discarded.

TABLE III
PORT SCAN TEST RESULT WITH AN EMPTY BLACKLIST.

Flows	**Not scan**	**Hor. scan**	**Vert. scan**	**FIN scan**
Gernerated	200000	20	1000	1000
Forwarded	200000	14	658	0
Blocked	0	8	342	1000
False negatives	0	2	12	0

The same procedure explained above was performed again to check if all streams were indeed added to the blacklist. In this procedure the blacklist has not been cleared and the result is shown in Table IV. It can be observed that all generated flows, including non-malicious ones, were discarded. As in the previous test, at some point the hosts performed a scan, then they were added to the blacklist, and therefore were now blocked. It is important to notice that both false positives and false negatives were not observed in this scenario.

TABLE IV
PORT SCAN TEST RESULT WHEN THE BLACKLIST CONTAINS ENTRIES.

Flows	**Not scan**	**Hor. scan**	**Vert. scan**	**FIN scan**
Gernerated	200000	20	1000	1000
Forwarded	0	0	0	0
Blocked	200000	20	1000	1000

V. FINAL CONSIDERATIONS

This work presented a non-intrusive solution for preventing port scan attacks. By using flow statistics collected on SDN networks, we detected port scan flows and then provided network security by updating the OpenFlow routing rules. Our results, presented on the experimental evaluation, show the effectiveness of this method in detecting malicious flows through statistics, which resulted in the absence of false positives and a very low number of false negatives. Moreover, we showed that our system is lightweight when considering resource consumption, like network bandwidth switching, memory usage and processing power.

It is important to mention that, in the literature, there are few works related to detecting port scan attacks for SDN. In addition, the available solutions do not have protection against them, which makes this work a relevant and promising alternative for detecting and preventing malicious threats. Besides, as our system works on the SDN controller, it may exchange information with other controllers, allowing a collaborative intrusion detection and prevention system for port scan attempts, providing more security on SDN infrastructures.

As future work, we intend to extend the use of Openflow counters for detecting other attacks, such as denial of service and insider threats, on a way that the controllers can exchange information by performing a collaborative IDS/IPS among different networks. Moreover, as the usage of encrypted traffic is rising on the Internet, and nowadays it is not possible to check this traffic by traditional packet based IDS/IPS, we also are designing a solution to analyze encrypted traffic to detect threats through Openflow statistics. **Acknowledgment.** Charles, Roben and Regio receive grants from CAPES/Brazil.

REFERENCES

[1] CERT.br, "Centro de estudos, resposta e tratamento de incidentes de segurança no brasil," Oct. 2016. [Online]. Available: http://www.cert.br/
[2] OPEN NETWORKING FOUNDATION, "Openflow," Oct. 2016. [Online]. Available: https://opennetworking.org/sdn-resources/openflow
[3] D. Kreutz, F. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.
[4] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of ip flow-based intrusion detection," *Communication Surveys Tutorials*, vol. 12, no. 3, pp. 343–356, Jul. 2010.
[5] A. A. Kolpyakwar, M. G. Ingle, and R. V. Deshmukh, "A survey on data mining approaches for network intrusion detection system," *International Journal of Computer Applications*, vol. 159, no. 1, 2017.
[6] M. Gao, K. Zhang, and J. Lu, "Efficient packet matching for gigabit network intrusion detection using tcams," in *20th Inter. Conf. on Advanced Information Networking and Applications (AINA)*, 2006, pp. 249–254.
[7] P. Owezarski, J. Mazel, and Y. Labit, "0day anomaly detection made possible thanks to machine learning," in *8th Inter. Conf. on Wired/Wireless Internet Communications (WWIC)*, 2010, pp. 327–338.
[8] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, Feb. 2013.
[9] S. Lagraa and J. François, "Knowledge discovery of port scans from darknet," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 935–940.
[10] J. Boite, P. A. Nardin, F. Rebecchi, M. Bouet, and V. Conan, "Statesec: Stateful monitoring for ddos protection in software defined networks," in *2017 IEEE Conf. on Network Softwarization (NetSoft)*, Jul. 2017.
[11] L. Zhang, Y. Guo, H. Yuwen, and Y. Wang, "A port hopping based DoS mitigation scheme in SDN network," in *12th Inter. Conf. on Computational Intelligence and Security (CIS)*, Dec. 2016, pp. 314–317.
[12] OPENDAYLIGHT, "Opendaylight, a linux foundation collaborative project," Oct. 2016. [Online]. Available: http://www.opendaylight.org
[13] OPEN NETWORKING FOUNDATION, *OpenFlow Switch Specification*, Dec. 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf
[14] J. Postel and J. Reynolds, "Telnet Protocol Specification," RFC 854 (Standard), Internet Engineering Task Force, May 1983, updated by RFC 5198. [Online]. Available: http://www.ietf.org/rfc/rfc854.txt
[15] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," in *Workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.
[16] H. Al-Mohannadi, Q. Mirza, A. Namanya, I. Awan, A. Cullen, and J. Disso, "Cyber-attack modeling analysis techniques: An overview," in *4th IEEE International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, Aug. 2016.