Flow-based Reconnaissance Attacks Detection in SDN-based Environment

Abdulmohsen Alsaedi, Adel Alshamrani, and Talal Alharbi
aalsaedi0063.stu@uj.edu.sa *asalshamrani@uj.edu.sa* *tralharbi@uj.edu.sa*
College of Computer Science and Engineering, University of Jeddah, Saudi Arabia

Summary
Software-Defined Networking (SDN) is an optimistic network architecture that seeks to provide increased flexibility via splitting forwarding functions (data plane) and network logic (control plane). This break feeds the logical centralization of controls, an overview of the global network, scalability, ease of programmability, and scope for pristine SDN-compliant services. In recent years, SDN in industry networks has continually grown. In the meantime, new challenges have appeared in different categories, such as security, management, and scalability. This paper will elaborate on the complex security issues existing in current SDN architecture, especially reconnaissance attacks, where attackers generate traffic to explore existing services, assets, and overall network topology. The proposed flow-based detection solution utilizes, in a slow-rate manner, OpenFlow counters to detect reconnaissance traffic techniques in the SDN environment. The results show that the proposed solution can detect reconnaissance attacks.

Keywords:
software-defined networking; reconnaissance attack; flow-based.

1. Introduction

Traditional networks have evolved the carriers between network devices in the digital package structure, employing traditional networks responsible for routing data and addressing. Although traditional networks are widely adopted, they are not easy to manage. Operators must configure every device in the network separately, which uses low-level commands to express and design high-level network policies and the complexity and difficulty of configuration. The network environment should endure the faults of the dynamics to fit for changes. Response and automatic reconfiguration do not exist in current traditional networks. It is challenging to enforce and implement the required policies in a traditional network environment [1]. Traditional networks are complicated in that they merge the control plane and the data plane. The control plane is responsible for transferring network traffic, and the data plane is responsible for forwarding traffic. This integration does not provide flexibility or innovation in the future, especially with the existing spread of technologies; thus, it is simply not achievable in practice [2] [3].

Software-defined networking (SDN) separates the control plane's vertical integration from routers and the data plane from switches. Fig. 1 shows the difference between traditional networking and SDN. This separation of the control plane and data plane switches the forwarding and controlling of the network operating system's logical controller. This split simplifies policy execution and network configuration, reconfiguration, and evolution [4]. The SDN networking model gives network programming increased scalability and provides more networking capabilities. SDN is present in all areas of networks, beginning with data centers, the Wide area network (WAN) and wireless network, and now the 5G network. The International Data Corporation (IDC) published that the annual rate of increase worldwide SDN market and investment growth was 54 percent from 2014 to 2020 [5].

Although SDN is fast-growing, security is critical and has become an attractive target for attackers, who try to penetrate the networks in several ways, the foremost step of which is the reconnaissance attack. This attack aims to collect and gather information about the targeted infrastructure network services, resources, network devices, network topology, and data exfiltration through methods such as scanning the port, packet sniffers, and sweeping the ping to exploit and use the information to plan for other dangerous attacks, such as denial-of-service attacks. The attacker exploits the network vulnerabilities that he discovered through reconnaissance attacks. Reconnaissance attacks are common and threaten networks quietly, which is difficult to detect. Some works have shown that reconnaissance attacks are applicable through SDN environments [6] [7] [8]. There are many methods and techniques for reconnaissance attacks, including evaluating the response from target network devices by using port scanning, user datagram protocol (UDP) ports, internet control message protocol (ICMP) ping, traceroutes, and footprinting.

Manuscript received September 5, 2022
Manuscript revised September 20, 2022
https://doi.org/10.22937/IJCSNS.2022.22.9.98

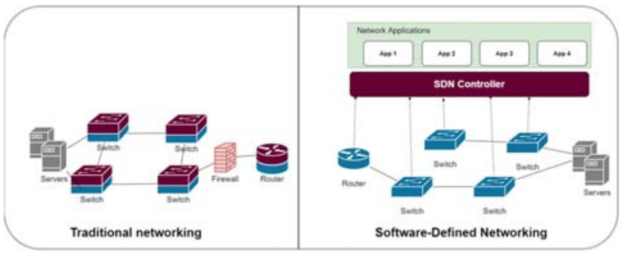


Fig. 1 Traditional networking versus software-defined networking

The data and control plane separators in SDN may introduce new reconnaissance attacks. However, different reconnaissance attacks against SDN-based environments can be categorized based on how the attacker performs the attack. For example, he/she may measure the packet travel time to understand the existing network topology or may send probing packets to identify existing security solutions such as Network Intrusion Detection Systems (NIDS).

The reconnaissance attacks target scans of various ports within a short period. Moreover, existing next-generation firewalls and network intrusion detection mechanisms can easily detect normal scan mode. However, advanced reconnaissance attacks scan targets slowly to bypass suspicion and avoid detection from the SDN network's existing security solutions, which means that an attacker does not send probe packets permanently. Instead, attackers send probe packets to a host. Therefore, it is more challenging to detect slow port scans. Since scanning is an essential phase within a typical attack scenario, it is vital to detect slow port scans to identify new attacks. This type of reconnaissance attack aims to gather information about the deployment of SDN to plan and prepare for another dangerous attack. Accordingly, the detection of slow port scans must consider intrusion and insider threat detection.

This paper tackles the challenge of detecting slow port scans in an SDN network. Reconnaissance attacks do not cause network damage but continually alert announcers of attacks that might cause severe damage. Hence, our outcome contributes to detecting attacks in an early initial stage during the scanning phase. Reconnaissance attacks are problems in which attackers exploit vulnerabilities in an SDN environment. We aim to use SDN features to counteract reconnaissance attacks. To summarize, our contributions are as follows:

- We propose a detection approach attack model to detect slow-rate reconnaissance attacks in SDN environments.
- The detection approach uses OpenFlow features by analyzing flow entry counter and identifying suspicious traffic.

The paper is organized as follows. The background is covered in Section 2. Section 3 describes the related work. Section 4 presents the proposed flow-based detection approach and the experimental setup covered in Section 5. Results and discussion are explained in Section 6. Conclusions and future work are discussed in Section 7.

2. Background

2.1 Software-Defined Networking (SDN)

The separation of the control plane and data plane can be recognized by using a well-defined programming interface between the SDN control plane (controller) and data plane (switches). The controller controls the data plane's components via an application programming interface (API). The most typical example of an API is OpenFlow [9], [10], which consists of one or more tables containing the rules for handling data packages. Every rule has a traffic match and implements certain actions, such as modifying dropping on traffic according to the rules on the controller application. An OpenFlow switch can inform the controller; it works like a switch, firewall, router, load balancer, and traffic shaper [11].

2.2 OpenFlow Counters

Every OpenFlow counter supports a flow table and flow entry, which helps in counting and measuring the statistical information, such as byte count, received packets, and matched packets. Moreover, the time of the flow entry refers to the duration installed in the SDN switch, and this must be tracked with double precision. In addition, the OpenFlow must count every packet employing that flow, even if the flow entry contains no impact on the packet or if the packet is eventually dropped or transmitted to the controller. Counters are the primary component of OpenFlow statistics and collect different characteristic attributes of the pipeline [12]. We expanded the OpenFlow table [10] to include two more columns that cover the units and description of OpenFlow counters. There are additional counters in Open vSwitch version 1.6 or later that are included in Table 1.

Table 1: Detailed explanation of the counters

Counter	Uint	Description	Action
Per Flow Table : set of the pipeline that includes flow entries .			
Reference Count (active entries)	active_count	Number of active entries	Required
Packet Lookups	lookup_count	Number of packets looked up in table	Optional
Packet Matches	matched_count	Number of packets that hit table	Optional
Per Flow Entry: is a component in a flow table operated to correspond and process packets.			
Received Packets	n_packets	Number of packets that have matched the entry.	Optional
Received Bytes	n_bytes	The total number of bytes from packets that have matched the entry	Optional
Duration (seconds)	duration_secs	The time, in seconds, that the entry has been in the table.	Required
Duration (nanoseconds)	duration_nsec	The time, in nanoseconds, that the entry has been in the table.	Optional
Idle Timeout	idle_timeout	Number of seconds of inactivity that occurs when the flow expires.	Optional
Hard Timeout	hard_timeout	Number of seconds, regardless of activity, occurs for the flow to expire.	Optional
Idle Age	idle_age	Number of seconds passed without packets.	Optional
Per Port : defines the port where packets enter and leave the OpenFlow pipeline as physical, logical, or reserved ports.			
Received Packets	rx_packets	Number of received packets	Required
Transmitted Packets	tx_packets	Number of transmitted packets	Required
Received Bytes	rx_bytes	Number of received bytes	Optional
Transmitted Bytes	tx_bytes	Number of transmitted bytes	Optional
Receive Drops	rx_dropped	Number of packets dropped by RX	Optional
Transmit Drops	tx_dropped	Number of packets dropped by TX	Optional
Receive Errors	rx_errors	Number of receive errors.	Optional
Transmit Errors	tx_errors	Number of transmit errors.	Optional
Receive Frame Alignment Errors	rx_frame_err	Number of frame alignment errors	Optional
Receive Overrun Errors	rx_over_err	Number of packets with RX overrun	Optional
Receive CRC Errors	rx_crc_err	Number of CRC errors	Optional
Collisions	collisions	Number of collisions	Optional
Duration (seconds)	duration_sec	Time port has been alive in seconds	Required
Duration (nanoseconds)	duration_nsec	Time port has been alive in nanoseconds	Optional
Per Queue: scheduled packets based on their priority on an output port			
Transmit Packets	tx_bytes	Number of transmitted bytes	Required
Transmit Bytes	tx_packets	Number of transmitted packets	Optional
Transmit Overrun Errors	tx_errors	Number of packets dropped due to overrun	Optional
Duration (seconds)	duration_sec	Time queue has been alive in seconds	Required
Duration (nanoseconds)	duration_nsec	Time queue has been alive in nanoseconds	Optional
Per Group: define a group of ports as one entity for forwarding packets.			
Reference Count (flow entries)	ref_count	Number of flows or groups that directly forward to this group	Optional
Packet Count	packet_count	Number of packets processed by group	Optional
Byte Count	byte_count	Number of bytes processed by group	Optional
Duration (seconds)	duration_sec	Time group has been alive in seconds	Required
Duration (nanoseconds)	duration_nsec	Time group has been alive in nanoseconds	Optional
Per Group Bucket: includes the collection of actions to be utilized before forwarding to the port			
Packet Count	packet_count	Number of packets processed by bucket	Optional
Byte Count	byte_count	Number of bytes processed by bucket	Optional
Per Meter: allow OpenFlow to execute rate-limiting, simple QoS procedures restraining a group of flows to a selected bandwic			
Flow Count	ref_count	Number of flows or groups that directly reference this meter	Optional
Input Packet Count	packet_in_count	Number of packets in input	Optional
Input Byte Count	byte_in_count	Number of bytes in input	Optional
Duration (seconds)	duration_sec	Time meter has been alive in seconds	Required
Duration (nanoseconds)	duration_nsec	Time meter has been alive in nanoseconds	Optional
Per Meter Band : utilized to describe the behavior of the meters on packets for different coverages of the meter measured rat			
In Band Packet Count	packet_band_count	Number of packets in band	Optional
In Band Byte Count	byte_band_count	Number of bytes in band	Optional

2.3 Types of Reconnaissance Attacks

The first step for any attack on networks is reconnaissance, which involves the collection of information about the network to plan the attack. This may include information such as the IP addresses of devices, servers, and ports [13]. Reconnaissance attacks can be categorized into the following:

- Packet Sniffers
  A particular device that catches data addressed to other devices and keeps it for later analysis by spying on transit traffic between network nodes.
- Scanning the Port
  A sequence of traffic sent by an attacker that attempts to discover which computer services are related with a certain prominent port number. There are three types of scanning [14]. The first scan is horizontal, where a source host scans numerous hosts on a particular port. The second is a vertical scan, where a source host scans a target host with several ports. Finally, a mixed scan is a mix of horizontal and vertical scans.
- Sweeping the Ping
  A scanning technique utilized by attackers to define a range of IP addresses to map live hosts.
- Queries Regarding Internet Information
  An attacker can exploit DNS queries to discover the domain owners and IP addresses in that domain.
- Time-based reconnaissance

These types of reconnaissance attacks target SDN environments that aim at flow table size, such as time-based reconnaissance. This attack helps the attacker collect and gather information about SDN resources. Many works have addressed how reconnaissance attacks can be conducted in an SDN environment [15], [16], [17],[18].

3. Related Works

In previous studies of recent years, several studies contained a definition of threat models reconnaissance attack on Software Defined Network (SDN), and they proposed several detection solutions.

Neu et al. [19] created a lightweight intrusion prevention system (LIPS) to enhance the performance of SDN networks. Port scan attacks, which frequently target SDN network devices, were detected and mitigated by the suggested LIPS methodology presented in this paper. Collection module (CM), detection module (DM), and prevention module (PM) were the stages in the proposed LIPS technique. The purpose of CM was to gather data resources from different network switch devices and then use that data to analyze and identify port scan attacks through predetermined criteria. To achieve detection, flow information was stored in the database. Neu et al. determined stored flows that included a decreased number of packets that illustrated port scanning flows. They collected the selected flows by source, destination IP, and destination port and obtaining the number of flows with similar addresses of source and destination. The scan was classified as horizontal, vertical, or mixed based on packet counts, source, destination address information, and destination ports. The source IP was added to a blacklist to determine if a scan attack was detected. All flows addresses stored on this list are dropped. As a result of the observed port scan attacks, the PM module places a set of rules on the SDN controller to mitigate these attacks.

Adel Alshamrani [20] eliminated reconnaissance attacks in SDN by proposing an SDN-based solution using OpenFlow counters, distributed firewall applications, and security policies. A distributed firewall application is qualified to track the flow based on pre-defined states that would observe the connection to critical nodes by malicious activity. Counters such as the number of received packets on each flow-matched packet, number of bytes, etc. can be employed to determine the purpose behind generating a flow., for example, a probing packet that is used to infer whether there is a match for such a flow can be effortlessly determined by counting the number of bytes and the number of received packets on the same flow. This application predicts meager stats from the probing packet compared to a legitimate flow. The proposed SDN-based solution result can detect and mitigate this type of attack at an early stage.

Ono et al. [21] developed a port scan detector based on statistics to locate hosts producing a large number of packet-ins. Switches are taught to transmit statistics signals to a controller when abnormal activity is observed. Flow entry is put on the host's blacklist in the event of detection. The detection algorithm differs from their previous work [22]. They used the k-Nearest Neighbor (kNN) algorithm; this may not be accurate with more complicated data patterns, but it is helpful for basic time-series data. Thus, they employed a new detection method named spectral residual (SR). SR is quick, unsupervised, and does not require manual labeling. The results verify that the proposed method can detect port scans with lower overhead when compared to existing methods.

Patil et al. [23] developed a model to protect SDN networks against malicious Transmission Control Protocol (TCP) – synchronize (SYN) attacks. The model involves sending TCP-FIN packets to validate the legitimacy of the attacker's IP address. When an attack is identified, the port scanner employs generated TCP- Finish (FIN) packets to verify the suspicious host's source IP address and port number. If it is deemed malicious, the flow is blacklisted and deleted from the flow table.

From our literature review, we believe that flow-based approaches are still a potential solution for detecting malicious flows. By reviewing the related works, we found that the following points are not handled correctly in many studies. A summary of the research gaps from the above research articles is presented as follows:

- One metric is used, such as the rate of incoming traffic. Thus, in peak hours, incoming traffic grows; therefore, conducting false positives on legitimate traffic flows traffic increases, thus leading to false positives on legitimate traffic.
- Statistical strategies require the processing many mathematical models and extensive network traffic.
- Some studies need time to provide early detection, the amount of traffic to be processed, and the target accuracy of the strategy.
- Some studies cannot detect slow scanning.

4. The Proposed Flow-based Detection Approach

4.1 Attack Detection

This section presents the research methodology. In addition, we present the steps that will be followed during the research to achieve the research objectives.

4.1.1 An Approach

Due to the characteristic of a reconnaissance attack, it might be challenging to detect, mainly when a threat actor follows a clever technique and crafts special packets to evade in-place security solutions such as using slow rate scanning. However, we utilize SDN-based features in the data and control planes to detect this attack. We utilize the data plane to capture significant statistics from OpenFlow counters [10]. These OpenFlow counters in Table. 1 support flow table and flow entry in counting and measuring the statistical information such as bytes count, received packets, matched packets, and the amount of time of the flow entry, a port, etc. Therefore, our approach considers various counters to look for suspicious traffic, as shown in Table. 2. We monitor each connection that belongs to the current flow or new flow. Algorithm 1 depicts the steps and procedures that collect statistics from selected counters and then extract and compare some features of all monitored connections. For recall, some counters are enabled and required by default. However, other counters are optional. In our approach, we use some counters that we believe would help us in enriching our decision.
The main challenge is to analyze and identify suspicious traffic in a lot of traffic records. This can be done by monitoring the user's behavior and relationship with this attack. To discover the OpenFlow counters that can use to identify suspicious traffic, we experimented with monitoring the behavior of users in the network with normal traffic and suspicious traffic. We took the challenge by considering the ability of the attacker to scan and gather the information of the SDN environment, especially with a slow rate scan.

4.1.2 System design

An SDN-based network employs the system model that includes an SDN controller. SDN switches Open Switch, end-hosts (e.g., hosts), Firewalls/routers. The SDN application layer implements the proposed detection mechanism. Moreover, it contains the network component's data forwarding and routing functions. Our detecting solution, as shown in Fig. 2, will be in the application layer and consists of two parts monitoring user behavior and detection methodology will monitor the counters, and if suspicious traffic is detected, our solution will identify the attacks according to the sequence described in Fig. 3.

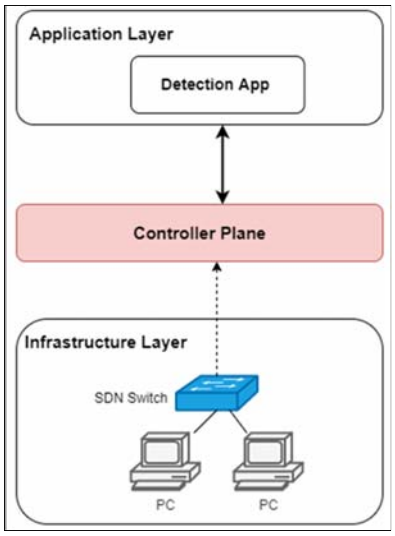
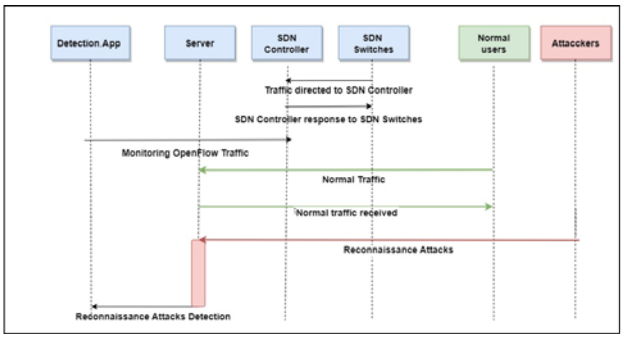


Fig. 2 Proposed detection architecture

As shown in Fig. 3, at the beginning of the sequence, the controller can reactively and proactively take actions like adding, updating and deleting flow entries in flow tables using the OpenFlow switch protocol. For example, according to configuration, if a flow table does not contain a match, the OpenFlow switch may pass the packet to the controller over OpenFlow channels so that the controller can take appropriate action. In the next stage, our proposed approach will monitor OpenFlow traffic in the network by collecting OpenFlow statistics. The traffic becomes normal if it does not meet our proposed approach conditions. In comparison, the traffic becomes an attack when the conditions are fulfilled.


Detection App
Server
SDN Controller
SDN Switches
Normal users
Attacckers
Traffic directed to SDN Controller
SDN Controller response to SDN Switches
Monitoring OpenFlow Traffic
Normal Traffic
Normal traffic received
Reconnaissance Attacks
Reconnaissance Attacks Detection
Fig. 3 The sequence of the proposed flow-based detection approach

4.1.3 Selected counters and detection algorithm

We can determine the suspicious traffic in the network based on scan behavior by selecting the number of packets that have matched the entry counter, the number of the destination port, and idle age. We use the source and destination IP addresses.

Our proposed detection, as shown in Algorithm 1 one divided into two stages. The first stage, obtaining all the flows, collects the information statistics from the switch every 1 second to avoid delay in detecting scanning in normal mode and stores incoming information in a temp filet to apply the algorithm. Neu et al. [19] have set the number of packets determined to be a port scan when the number of packets is equal to or less than five, which three packets represent the handshake of a TCP connection SYN, SYN/ Acknowledgment (ACK), and ACK, and the event of re-transmission adds tow tolerance packets. According to the monitoring and analysis of the attacker's behavior in the experiment, there is traffic with six packets that was scanning the server, while at the same time the attacker was sending normal traffic, and this is the case that we consider six packets as scan traffic and then the algorithm checks if the number of packets is between one to six. To improve detection and accuracy algorithm will check if the idle age counter is between 1 to 10 seconds, which describes that there is no traffic in the flow. Therefore, a low stream without traffic is considered suspicious. The second stage is to fulfill the condition of scanning more than one destination port to detect the attacker, even if the attacker scans on one host, by utilizing the flow number of the destination port. One port scan is not defined as an attack to reduce false positives. If the flow meets the first and second stage's requirements, the traffic will become attack traffic.

Algorithm 1 Flow-based Reconnaissance Attacks Detection

```
Input Collect selected OpenFlow Features (n_packets ,idle_age , tp_dst nw_src ,nw_dst)  every 1 seconds.
Output OpenFlow counters initialization to determine the IP source and destinations of reconnaissance traffic
1  If n_packets  ≥ 6 AND idle age = 1 to 10
2   Store to temp file as a dataset for monitoring. And if the following port condition is met, the traffic is an attack.
3    If the number of scanned ports (tp_dst) > 1 for 1 (nw_dst) OR scanned 1 port (tp_dst) for many (nw_dst)
      OR the number of scanned ports (tp_dst) > 1 for many (nw_dst)
4    Ignore responses based on the direction or if the detected attacker is a former victim,
     then ignore and take this as a response.
5    Then
6      Add Source IP address nw_Src as an attacker
7       Add Destination IP address nw_dst as victims
8       Set detection time in seconds
8   else
9   Normal SDN Forwarding
```

To reduce false positives detection, our solution categorizes Low, Moderate, and High users' behavior as shown in Table. 2, representing Normal, Suspicious, and Attack from a traffic classification perspective. Traffic becomes a normal or low risk if it does not meet any of the algorithm conditions, and traffic becomes suspicious if the number of packets is less than six and the idle age is between 1 to 10. Moreover, suspicious host machine changes to attack traffic when the scanning occurs for more than one destination port in one host.

Table 2: Classification of monitoring traffic behavior

Risk Level	***Traffic classification***	***Description***
High	Attack	The algorithm condition is met, the number of packets is less than six, the idle age is between 1 to 10, and it meets horizontal, vertical, and mixed scans requirements
Moderate	Suspicious	The algorithm condition is met, the number of packets is less than six, and the idle age is between 1 to 10
Low	Normal	The algorithm condition has not met the requirements.

5. Experiment

5.1 Experiment Setup

To create a virtual network and experiment with SDN networks using OpenFlow, we used Mininet [24]. The network contains four hosts, one controller, a switch, and a server. We use the POX OpenFlow SDN controller [25] written in Python [26], as shown in Table. 4, h1 is defined as a server for h2, h3, and h4 to act as clients and to conduct experiments. We used Iperf [27] tool to produce traffic to measure the throughput between the two destinies in one or both paths.

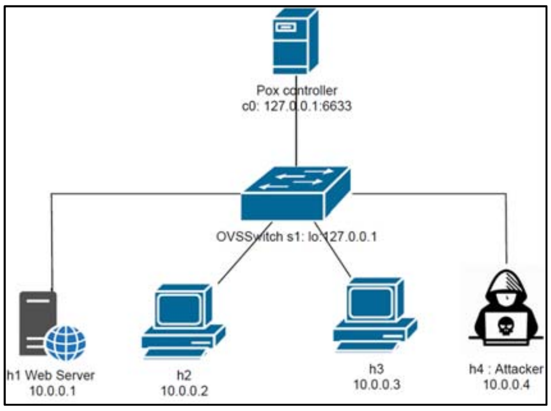


Fig. 4 Environmental SDN network topology

Moreover, Iperf has server and client functionality that we will utilize in the experiment and assume Iperf traffic acts as normal traffic. We used Nmap as a reconnaissance tool .h4 scans the entire network to slow suspicious rate traffic. We use scan timing (-T0) to (-T2) for IDS evasion. It is a polite method to delay the scan to utilize less bandwidth and target network resources. We used a script to gather information from the OpenFlow counters that classify under per flow entry.

Table 3: Environmental setup

VM Name	***IP address***	***Type***
c0	127.0.0.1	POX Controller
s1	127.0.0.1	OVS Switch
h1	10.0.0.1	Server
h2	10.0.0.2	Normal host
h3	10.0.0.3	Normal host
h4	10.0.0.4	Attacker host

5.2 Attack model and scenarios

5.2.1Attack model

We have made the following assumptions about the attack model:

- The attacker plans to discover and collect information about the deployment of SDN infrastructure.
- The attacker can access compromised machines inside the network.
- The existing security solutions in the network detect high-rate reconnaissance attacks and prevent information gathering.
- The attacker uses slow-rate reconnaissance attacks to avoid detection by the existing security solutions in the network.
- The attacker scans the server, such as a web server.

We used to scan ports, one of the reconnaissance attack types by Nmap time options as shown in Table. 5, to make the attack more challenging to be detected.

Table 4: Reconnaissance attacks setting.

Tool	Nmap
Scan Packet rate. The difference between the two packets sent	• (T0) paranoid,300 seconds. • (T1) sneaky,15 seconds. • (T2) polite, up to 0.4 seconds. • (T3) normal, up to 0.1seconds
Experimental Duration	60 minutes

Moreover, we used timing (T2 to T0) to slow-rate scan attacks to avoid detection by the existing security solutions in the network. Furthermore, we use T3 for the normal scan. We used Nmap TCP SYN Scan (-sS) for the port scanning method supported by Nmap and Versions Scans (-sV) to allow version detection probes of those ports to determine running services. The mentioned scan techniques performed using , e.g., “Nmap -sS -sV -T1 10.0.0.1-3” “Nmap -sS -sV -T2 10.0.0.1-3” and “Nmap -sS -sV -T3 10.0.0.1-3” commands That's what makes it mixed scan it scans more than one port for multiple hosts. The delay between two consecutive scan packets is 5 minutes in T0 scan mode is practically impossible, as the scanning operation of 65536 ports for just one IP address would take almost 228 days. Moreover, for proof of concept for our proposed solution, we chose to scan two devices and select ports, e.g., 8080 and 80. We use for example “Nmap -sS -sV -p8080,80 -T0 10.0.0.1-2” command. And we consider vertical, mixed, and horizontal for testing.

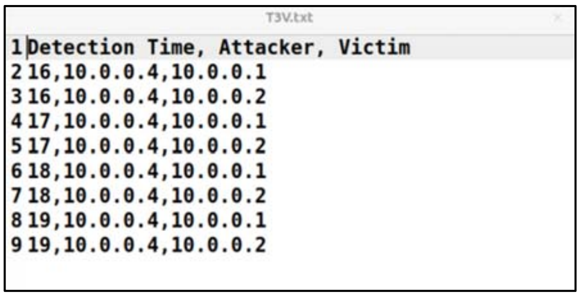
5.2.2 Scenarios

To evaluate the accuracy of our detection algorithm. The algorithm was tested in several scenarios, according to the case of normal scanning and gradually slow scanning, up to five minutes between each packet, as follows:

- (T0) paranoid scanning,300 seconds.
- (T1) sneaky scanning, 15 seconds.
- (T2) polite scanning, up to 0.4 seconds.
- (T3) normal scanning, up to 0.1seconds

6. Results and Discussion

In this subsection, we show the detection results for various types of timing scanning ports. The program reads and collects information from the counters every 1 second and issues a text output file, as shown in Fig. 5, an example of an output file screenshot.



```
T3V.txt
Detection Time, Attacker, Victim
16,10.0.0.4,10.0.0.1
16,10.0.0.4,10.0.0.2
17,10.0.0.4,10.0.0.1
17,10.0.0.4,10.0.0.2
18,10.0.0.4,10.0.0.1
18,10.0.0.4,10.0.0.2
19,10.0.0.4,10.0.0.1
19,10.0.0.4,10.0.0.2
```

Fig. 5 Screenshot showing the results file after detecting the attacker with his targets, and the time of detection is in seconds.

6.1 Paranoid scanning (T0)

The proposed application is Detected in the case of a very slow scan that sends a packet every five minutes within 10.3167 minutes in the case of mixed scans. Vertical scans were detected within 15.1667 minutes. Horizontal scans were detected within 10.2833 minutes. It is normal for this detection delay because scanning is slow.

6.2 Sneaky scanning (T1)

In the case of mixed scans, the detection application was able to detect the source of the scan in T1 mode within 47 seconds. Vertical scans were detected within 63 seconds. While Horizontal scans were detected within 46 seconds ,it is close to the time that was detected in the case of mixed scans .

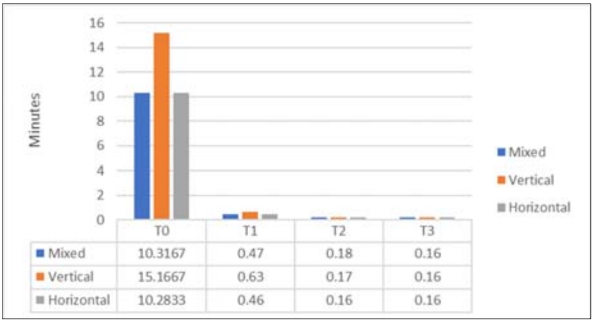
6.3 Polite scanning (T2)

When testing the scan in T2 mode, the results are shown that 18 seconds was enough to detect the attacker in the mixed mode, which is a note that the faster the port scan, the less the detection time. 17 seconds through vertical scans moreover 16 seconds for horizontal scans. Almost all results are of the same duration in the case of polite scanning.

6.4 Normal scanning (T3)

We tested the detection application in a normal scan that was able within 16 seconds to detect the source of all types of port scans .

As shown in Fig. 6, due to slow traffic, vertical scanning took a little longer to detect the attacker in T0 and T1 modes. While there was not much difference in the detection times between all types of scans in T2 and T3.



	T0	T1	T2	T3
🟦 Mixed	10.3167	0.47	0.18	0.16
🟧 Vertical	15.1667	0.63	0.17	0.16
⬜ Horizontal	10.2833	0.46	0.16	0.16

Fig. 6 The time it took for the proposed solution to detect the attacker in several Nmap scanning modes (T3-T0).

From the previous explanation of the results, our proposed approach proved with high accuracy to identify the attacker and the victims in the several types of scanning. We made a comparison with related works in several respects, as you can see in Table 5.

Table 5: Comparison with related works in terms of slow scanning

Proposed framework	***Slow Scanning***	***SDN features***
Lightweight IPS [19}	No	Openflow statistics
Spectral Residual [21]	No	Packet-In Messages
Our proposed approach	Yes	Openflow statistics

7. Conclusion and future work

Reconnaissance attacks seek information about the targeted infrastructure network services, resources, and network devices to plan for further dangerous attacks. SDN splits the control plane's vertical integration from routers and the data plane from switches. The data and control plan separators in SDN introduce reconnaissance attacks. One type of reconnaissance attack is port scanning which discovers the SDN environment. To detect a reconnaissance attack, we monitored the attacking traffic behavior using SDN features and then used it in the detection stage. Unlike other solutions, our proposed can detect slow rate scans. We experimented with testing our real-time proposed solution by using a realistic virtual network. The results show that the proposed solution can detect reconnaissance attacks.

In future work, we plan to develop our work with the other SDN controllers and extend the benefit of OpenFlow counters for detecting reconnaissance attacks.

References

[1] Benson, T., Akella, A., & Maltz, D. A. (2009, April). Unraveling the Complexity of Network Management. In NSDI (pp. 335-348).

[2] Raghavan, B., Casado, M., Koponen, T., Ratnasamy, S., Ghodsi, A., & Shenker, S. (2012, October). Software-defined internet architecture: decoupling architecture from infrastructure. In Proceedings of the 11th ACM Workshop on Hot Topics in Networks (pp. 43-48)

[3] Ghodsi, A., Shenker, S., Koponen, T., Singla, A., Raghavan, B., & Wilcox, J. (2011, November). Intelligent design enables architectural evolution. In Proceedings of the 10th ACM Workshop on Hot Topics in Networks (pp. 1-6).

[4] Kim, H., & Feamster, N. (2013). Improving network management with software defined networking. IEEE Communications Magazine, 51(2), 114-119

[5] Antonella Corno, Taking control of the programmable network. .2017. [Online]. Available: https://www.sdxcentral.com/articles/contributed/taking-control-programmable-network/2017/02/

[6] Seungwon Shin and Guofei Gu. Attacking software-defined networks: A first feasibility study. In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, pages 165-166. ACM, 2013.

[7] Heng Cui, Ghassan 0 Karame, Felix Klaedtke, and Roberto Bifulco. On the fingerprinting of software-defined networks. IEEE Transactions on Information Forensics and Security, 11(10):2160-2173, 2016.

[8] John Sonchack, Anurag Dubey, Adam J Aviv, Jonathan M Smith, and Eric Keller. Timing-based reconnaissance and defense in software-defined networks. In Proceedings of the 32nd Annual Conference on Computer Security Applications, pages 89-100. ACM, 2016.

[9] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., ... & Turner, J. (2008). OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 38(2), 69-74.

[10] Specification, Open Flow Switch. "Open Networking Foundation ONF." Technical Specification (2015).

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

[12] McKeown, N. (2015). OpenFlow switch specification version 1.5. 1, Open Networking Foundation, Palo Alto, CA. USA, Tech. Rep. TS-025.

[13] Uma, M., and Ganapathi Padmavathi. "A Survey on Various Cyber Attacks and their Classification." Int. J. Netw. Secur. 15.5 (2013): 390-396.

[14] Sperotto, Anna, et al. "An overview of IP flow-based intrusion detection." IEEE communications surveys & tutorials 12.3 (2010): 343-356.

[15] Seungwon Shin and Guofei Gu. Attacking software-defined networks: A first feasibility study. In ProCeedings of the seCond ACM SIGCOMM inserted new security policy, at the time 0.7 second all the workshop on Hot topiCs in software defined networking, pages 165- traffic now are permitted following PDSP.166. ACM, 2013

[16] Heng Cui, Ghassan O Karame, Felix Klaedtke, and Roberto Bifulco. On the fingerprinting of software-defined networks. IEEE TransaCtions on Information ForensiCs and SeCurity, 11(10):2160—2173, 2016.

[17] John Sonchack, Anurag Dubey, Adam J Aviv, Jonathan M Smith, and Eric Keller. Timing-based reconnaissance and defense in software-defined networks. In ProCeedings of the 32nd Annual ConferenCe on Computer SeCurity AppliCations, pages 89-100. ACM, 2016.

[18] Yong Li and Min Chen. Software-defined network function virtualiza-tion: A survey. IEEE ACCess, 3:2542-2553, 2015.

[19] Neu, Charles V., et al. "Lightweight IPS for port scan in OpenFlow SDN networks." NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium. IEEE, 2018.

[20] Alshamrani, Adel. “Reconnaissance attack in sdn based environments.” 2020 27th International Conference on Telecommunications (ICT). IEEE, 2020..

[21] Ono, Daichi, et al. "A proposal of port scan detection method based on Packet - In Messages in OpenFlow networks and its evaluation." International Journal of Network Management 31.6 (2021): e2174.

[22] Ono, Daichi, et al. "A design of port scan detection method based on the characteristics of packet-in messages in openflow networks." 2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE, 2020

[23] Patil, Jitendra, et al. "Port scanning based model to detect Malicious TCP traffic and mitigate its impact in SDN." 2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC). IEEE, 2021.

[24] Lantz, Bob, Brandon Heller, and Nick McKeown. “A network in a laptop: rapid prototyping for software-defined networks.” In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, p. 19. ACM, 2010.

[25] Fernandez, Marcial. “Evaluating OpenFlow controller paradigms.” In ICN 2013, The Twelfth International Conference on Networks, pp. 151-157. 2013

[26] G. F. Lyo, Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning Paperback, Nmap.org, Jan. 2009

[27] Sanner, Michel F. “Python: a programming language for software integration and development.” J Mol Graph Model 17.1 (1999): 57-61.

[28] IPERF, Networking with iperf .[Online]. Available: http://openmaniak.com/iperf.php. Accessed December 29, 2013.