A Secure Scheme to Counter the Man in the Middle Attacks in SDN Networks-Based Domain Name System

Frank Manuel Vuide Pangop[1], Miguel Landry Foko Sindjoung[2], Mthulisi Velempini[3]
IUT-FV of Bandjoun, University of Dschang, Bandjoun, Cameroon[1]
Department of Computer Science, University of Limpopo, Mankweng, South Africa[2,3]

***Abstract*—Internet and computer networks are vulnerable to cyber-attacks which compromise the services they provide to facilitate the management of data and users. The domain name system (DNS) is the Internet service that translates domain names and computer IP addresses and IPs to domain names. DNS is sometimes a victim of attacks that are difficult to detect and prevent because they are not only very stealthy but also conceal its proper functioning. Among the attacks that DNS is subject to, there are man-in-the-middle (MITM) attacks. Traditional networks that centralize all network functions in a single device complicate the detection and protection of systems against these attacks challenging. Software-defined networking (SDN) is a technology that is widely used to address many traditional network problems such as security and network architectures. Therefore, in this paper, we propose a scheme designed to detect and block man-in-the-middle attacks based on a newly defined architecture. The effectiveness of our secured solution is evaluated in an SDN architecture where an Address Resolution Protocol spoofing MITM attack is generated for the evaluation purpose. The results of our simulations show that we can effectively detect the attack and the performance evaluation of our approach shows that the proposed solution is effective in terms of security, implementation cost and resource consumption. We then recommend the use of our proposed solution to address the MITM attacks in SDN networks-based Domain Name System.**

***Keywords*—Cyber security; domain name system; man in the middle attack; software defined networking**

I. Introduction

Cybersecurity protects computer systems from malicious attacks. Cybersecurity [1] is the set of measures and practices which protect computer systems, networks, data and users against attacks, intrusions and online threats. Interestingly, human error counts for 50-80 % of network outages [2] due to misconfigurations made by administrators when configuring a network. These attacks also affect the domain name system (DNS).

The DNS is a hierarchical, distributed system for associating domain names with IP addresses. DNS is also vulnerable to attacks [3]. New forms of attacks that are stealthy and sophisticated [4] are difficult to address in traditional networks. The use of new technologies, such as software-defined networking (SDN) is a new networking paradigm that facilitates network management and administration by providing an interface to control network infrastructure such as switches which can be employed to address the security problem.

DNS servers are victims of several attacks because of their role and visibility on the Internet. Some of the common attacks are amplification and DoS attacks, botnets and attacks using DNS, DNS Zone transfer attacks, and DNS manipulation. DNS manipulation compromises DNS resolvers responsible for manipulating DNS responses [5]. In recent years, several researchers have focused on finding ways to counter DNS manipulation that uses active measurements to profile open resolvers. In this paper, we focus on DNS manipulation, more precisely on the man-in-the-middle (MITM) attack.

The MITM attack is an interference where an attacker eavesdrops, intercepts, or manipulates communication between two or more parties to steal information. In DNS, the attacker manipulates DNS servers or cache to redirect users to fraudulent sites to steal information or download the malware [6]. The attacker using a MITM attack intercepts communication on the communication channel and pretends to be a receiver to the sender and the sender to a receiver. This can be possible following two scenarios: firstly, communication between the two entities is not secure, in other words, the start of the communication is not preceded by an authentication; secondly, the two parties communicate over an insecure communication channel, such as public Wi-Fi or an unencrypted network. So, an attacker can intercept and manipulate the messages.

It is, therefore, necessary to find effective mechanisms to prevent and detect MITM attacks in DNS to ensure the security and confidentiality of users' communications. This involves implementing protection mechanisms such as encryption of communications, mutual authentication, Intrusion Detection System/Intrusion Prevention System (IDS/IPS), and the use of protocols such as Secure Socket Layer/Transport Layer Security (SSL/TLS). In this article, we propose a solution to detect and block MITM attacks on DNS servers based on a new SDN architecture.

The rest of this paper is organized as follows: in Section II, we present the related work. We then present our proposed scheme to address the MITM attacks in SDN networks-based DNS in Section III. Thereafter, we present the simulation results and analysis in Section IV. Finally, we conclude the paper in Section V.

This work is based on the research supported by the National Research Foundation of South Africa (Grant Numbers: 141918).

II. State of the Art

In the literature, many solutions have been proposed to counter security attacks in SDN architecture. In this section, we review some of the related works.

In study [7], Sungmin et al., proposed a new security extension named Topoguard to the existing OpenFlow controllers to provide automatic and real-time detection of network topology poisoning attacks in an SDN architecture. Their solution is premised on the poisoned network visibility and the upper-layer OpenFlow controller, services/applications may be misled, leading to hijacking, denial of service or MITM attacks. Topoguard is based on SDN/OpenFlow topology service security analysis for vulnerability identification. Sungmin et al. have proposed a network topology poisoning attack to exploit the vulnerabilities their solution has identified. Moreover, they investigated the defence space and proposed an automatic mitigation approach against network poisoning attacks and a prototype defence system.

In study [8], Cheng et al. investigated the potential threats that the OpenFlow control channel may face from the MITM attacks. They proposed an attack model in an IoT-Fog architecture and they also demonstrated the consequences of these attacks. Based on the previous demonstration, Cheng et al. [8] proposed a lightweight countermeasure based on Bloom filters. Anass et al. [9] proposed a Context-Based Node Acceptance (CBNA) framework to mitigate the MITM threats based on the authentication of new nodes using OpenFlow switches in a software-defined network. The CNBA framework takes place when new nodes attempt to connect to the controller for the first time and make a decision based on the response time of nodes.

Sahri et al. [10] proposed a collaborative SDN authentication (CAuth) which blocks the spoofed packet while authenticating the legitimate packet when establishing communications between a client and the DNS server within an SDN architecture. Their solution allows authentication of the two entities wishing to communicate with each other (client and DNS). The authentication is made between the client controller and the DNS server controller. When the DNS server controller receives the query packet, it sends an authentication packet to the clients that initiate the communication; when the DNS server controller receives an authentication from the client, it replies with the DNS reply to the client. During this exchange, the switch flow table is updated from a match check with the incoming packet to detect the attacking packets. This method can discriminate legitimate packets from attacking ones. Moreover. it can block the attacks before they reach the DNS server, but it consumes a lot of bandwidth. Main focus in this paper will be on the MITM attacks

III. The Proposed Scheme to Address the MITM Attacks in DNS based on SDNs

In this section, we present our contribution to counter the MITM attacks in DNS based on SDN networks. We present a SDN architecture (Section III-A) within which our proposed secure MITM attacks algorithm (Section III-B) is executed.

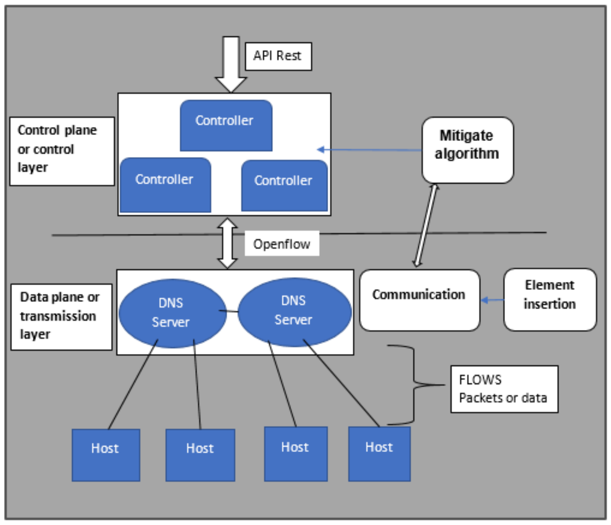


Fig. 1. The proposed architecture.

A. The Proposed Architecture

The proposed architecture is organised into two main layers: the infrastructure layer and the control layer. These two layers are linked by a communication interface which promotes interaction between them. The proposed architecture is shown in Fig. 1 and includes some modules at each layer. The communication interface acts as a bridge between the infrastructure and the control layers. Its main role is to ensure data transfer from one layer to another. The roles of each layer are discussed below:

1) The infrastructure layer: The infrastructure layer is responsible for collecting and transmitting data to the post traffic layer in the network for analysis. The data collection is done according to the rules defined by the control layer. Importantly, this layer has the various network equipment that are DNS server and hosts as shown in Fig. 1. The following modules are incorporated in the infrastructure layer:

- The communication module: It is used to establish communication rules between the DNS server of the infrastructure layer and the controller in the control layer through the communication interface. The collected data is transmitted to the control layer, otherwise, no verification will be made at that layer for attack detection.
- The element insertion module: It is responsible for filtering the data stream collected from the parameters defined by the controller. If these parameters are not defined, then all the network data flow is sent to the control layer, resulting in the increase of the workload in the controller and consequently, the analysis of some incoming data may be ignored, causing a security risk in the network.

2) The control layer: This layer is responsible for overall network monitoring. In other words, it allows the management and control of the flow of data transmitted in the network. This layer works through the controller. The controller fulfils or applies all the functions of the control layer and serves as a bridge between the applications and the infrastructure layer. Our proposed algorithm (Algorithm 1) which mitigates the MITM attacks is executed in this control layer.

B. The Proposed Algorithm

To detect the MITM attacks in DNS based on SDN network, we propose Algorithm 1 named MITM attack detection (MitAL). MitAL works as follows: Firstly, the rules for DNS-controller communication are established. After the traffic is generated, the transmitted packets are captured and sent to the controller for analysis. If the packet is of ARP type then an attack is detected. If the MAC address of the packet is not the expected one or there is no corresponding MAC address in the MAC-IP table, it is classified as malicious. The MAC address received with the incoming packet is considered as MAC address of a malicious node. Therefore, a notification is sent to all the DNS servers of the network to block all communications from the host with the detected malicious MAC address. Moreover, if the packet is of IP type then an attack is detected if the IP address is valid and the three following conditions are satisfied:

1) The IP address of the node is external to the logical network
2) The response time is greater than the defined threshold value, and
3) The received IP address is not the expected one.

In the previous scenario, a notification is sent to all the DNS servers of the network to block all the communications from the host with the detected malicious IP address.

Algorithm 1: MitAL:MITM attack detection

```
Input: $Thr$: Threshold in milliseconds, ip_to_mac = ():
       MAC-IP address table;
1  Capture packets after generating network traffic;
2  Retrieve information (types) about each incoming
     packet;
3  For each packet P Do
4      If ARP packet Then
5          Extract (src_mac, src_ip) of packet;
6          If ip_to_mac[P.src_ip] != P.src_mac Then
7              Block all the communications from the host
                 with MAC address src_mac
8      Else
9          If IP packet Then
10             Extract (src_ip, dst_ip) of packet;
11             If src_ip is valid Then
12                 time ← Calculate response_time();
13                 mask ← Check if is_external (src_ip) to
                     network;
14                 source ← Check if src_ip !=
                     expected(src_ip);
15                 If (time $\geq$ Thr) and mask and source Then
16                     Block all the communications from
                         the host with ip address src_ip
```

To evaluate the effectiveness of our scheme, we perform some simulations where we compare our algorithm to those presented by authors in studies [9] and [10]. The generated results are presented in Section IV.

IV. Simulations and Analysis

This section presents the results of our simulations. These simulations were performed using Mininet* as a network emulator because it creates a network of virtual hosts, switches, controllers and links, and therefore it can be used to build SDN. As a network security tool for analysing, penetration testing and protocol analysis, we have used Ettercap†. Ettercap is a comprehensive suite of tools that allows users to sniff, intercept, and manipulate network traffic in real time. It was used in our simulations to generate the MITM attacks. Wireshark‡ was also used to capture and analyse network traffic in real time. It can support a range of protocols and various capture interfaces. Table I summarizes the details of the used simulation environment and Fig. 2 shows testbed environment.

TABLE I. Simulation Environment

Tool	Version
OS	Ubuntu 22.04 LTS
Python	3.10
Mininet	2.3.0
Ettercap	0.8.3.1
Wireshark	3.6.2
Ryu controller	4.34

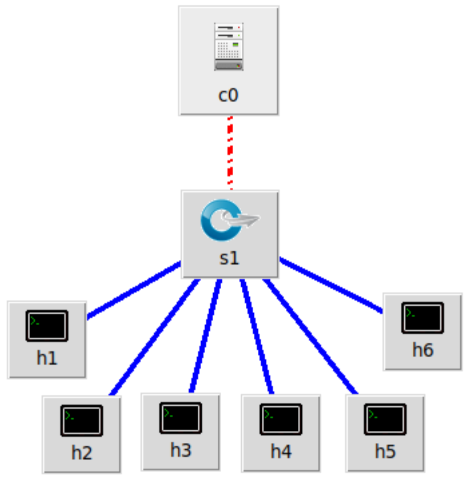


Fig. 2. Testbed environment.

We have used the Mininet generator to create SDN network topology on Ubuntu 22.04 LTS installed on a laptop with an intel core i3 processor, RAM - 4GB, Hard disk - 30GB. We used the Ryu controller for network management and monitoring (We justify the choice of the controller in Section IV-A). The SDN network set-up includes a Ryu controller, one open vSwitches (s1) acting as DNS server, and six end-hosts (h1-h6). The non-malicious hosts are h3 and h4, and the malicious host is h1. To generate normal traffic between non-malicious hosts, we used iperf3 and ping command and we generated an attack (ARP spoofing - MITM attack) traffic using Ettercap. The use of Wireshark assisted us in retrieving more details or information about the attack.

*http://mininet.org
†https://www.ettercap-project.org
‡https://www.wireshark.org

A. Choosing the Suitable Controller for our Simulations

To choose the suitable controller to be used in our simulations, We evaluated three existing controllers designed for mininet environments: Ryu, Pox and Floodlight. The evaluation was inspired by the work presented in study [11]. The parameters used for comparison are presented in Table II.

The evaluation was done by generating traffic and varying the number of switches from 5 to 100 in order to obtain accurate results. During this time, data related to bandwidth and latency were collected and the resulting graphs are presented in Fig. 3(a) and Fig. 3(b). The evaluated controllers are Ryu, Pox and Floodlight. In Fig. 3(a), we observed that overall, the Pox controller consumes more bandwidth compared to the Ryu and Floodlight controllers, especially when the number of switches is less than 64. At the same time, the Ryu controller consumes more bandwidth than Floodlight. However, in Fig. 3(b), we observed that the latency is overall the best when using the Ryu controller compared to the Pox (medium) and Floodlight (Bad) controllers. In conclusion, among the evaluated controllers, Ryu is moderate in bandwidth consumption and the best for latency. For this reason, it was selected and used in our simulations.

B. Attack Detection using Flow Table

After traffic is generated by an attacker, the protocol stores any information about each incoming packet according to different fields defined in the flux table. By inspecting this table and the contained information, we were able to detect the presence of a MITM attacker in the communication between the two non-malicious hosts as illustrated in Fig. 4. It shows the contents of the flow table before and after attack generation. Before the attack, normal traffic between the non-malicious hosts is observed without interference in their communications. After the attack is launched, the attacker initiates, intercepts and manipulates the communication between the legitimate hosts by impersonating one of the two entities according to the sent packet. The highlighted green and red areas respectively represent the non-malicious and the attacking hosts.

C. Capturing Network Traffic using Wireshark

We used wireshark for overall network capture and analysis to detect the attack and to have more details and information about the attack. Fig. 5 illustrates a wireshark capture of all traffic generated in the network. We can observe normal communication between network hosts using the ICMP protocol. An example of attack traffic is observed by the area highlighted in red where we note a host which initiates communication between the two non-malicious nodes to intercept and manipulate the traffic. We also observe that the protocol used is modified in ARP since in the ARP spoofing attack, the size and the information on the packets are modified.

Fig 6(a) and 6(b) illustrate the content of a request and a response sent from the attacking host; when communicating with a non-malicious host. They show the request and reply content from the attacker. Each of these messages contains a header and a body specific to it. The header allows to see some information about the message in transit. That is, the detection of an attacking host is effective, both in packet transmission and reception.

D. Performance Based on Bandwidth and Latency

Bandwidth is used to determine a network's ability to transfer information/packets over a defined period. Generally expressed in Mbits/sec (can also be expressed in Kbits/s or Gbits/sec), in our work, we expressed it in Gbits/sec. Latency in a network is the time it takes for an entire data packet to be sent from one point to another. It is measured in milliseconds (ms) and can be influenced by various factors such as the distance between the two points, the quality of the connection, and the number of hops between the points. To compare our algorithm according to bandwidth and latency in the network, we simulated an exchange of packets and collected the associated data to represent it in graphs illustrated in Fig. 7(a) and 7(b).

Fig. 7(a) depicts the comparative bandwidth results of our algorithm (MitAL) and the work of Anass et al.[9] (CBNA). The results show that: for five (5) switches the CBNA consumes less bandwidth than MitAL, however, for $n = [10, 20, 32, 64, 100]$ where n is the number of switches, MitAL consumes less bandwidth than CBNA. From this comparison, we can conclude that our algorithm minimises the bandwidth consumption.

Fig. 7(b) shows the latency between our algorithm (MitAL) and the work of CBNA framework of Anass Sebbar et al. [9]. The graph obtained from the collected data illustrates that for $n = [5, 10, 32, 100]$ where n is the number of switches, the latency occurred by MitAL is better than that of CBNA. This comparison show that our algorithm minimises the latency since in most of the considered points our algorithm achieved better performances.

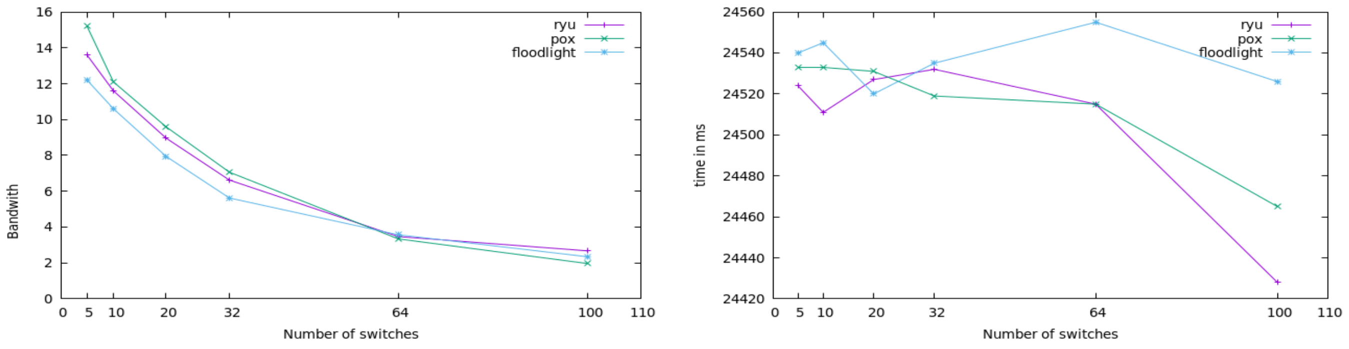
E. Traffic without Protection vs. Traffic with MitAL

Fig. 8 shows the comparison between the network traffic without protection and the traffic with our proposed solution (MitAL). We can observe and conclude that when the network is not protected and the attack is generated, the packet transmission time is high which means that the latency is high. Unlike with our algorithm and the associated security protocols, we observe that this time is considerably reduced. This shows the effectiveness of our protocol.

F. Detection Rate by Number of Generated Attacks

To evaluate the performance of our algorithm, we used the attack detection rate parameter. This rate is determined from the number of generated attacks, and it shows the performance in terms of detection when the MITM attack generated from different sources. It is computed using evaluation metrics such as throughput, available bandwidth when the attack is generated, and analysis of network communications to determine healthy packets from the number of packets exchanged, attacked and lost (percentage).

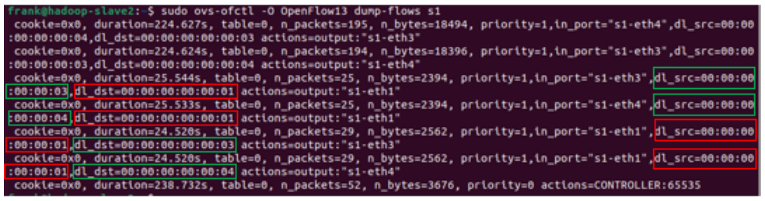
We have compared MitAL to the solution by Anass et al. (CBNA) [9] and that of Sahri et al. (CAuth) [10]. The comparison graph is provided in Fig. 9. When analysing that figure, we can conclude that our solution outperforms CBNA and CAuth when the number of generated attacks increases until it reaches 60. From that point, the detection rate is still better for MitAL compared to CAuth, but less than that of CBNA, which is quite satisfactory.



(a) A comparative test based on bandwidth

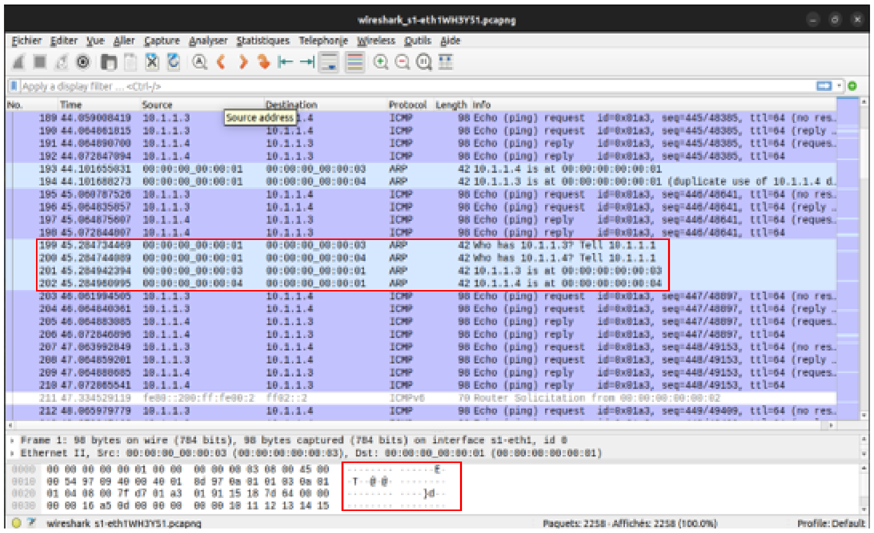
(b) A comparative test based on latency

Fig. 3. A comparative study between controllers that may be considered.



```
frank@hadoop-slave2:~$ sudo ovs-ofctl -O OpenFlow13 dump-flows s1
 cookie=0x0, duration=224.627s, table=0, n_packets=195, n_bytes=18494, priority=1,in_port="s1-eth4",dl_src=00:00
:00:00:00:04,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=224.624s, table=0, n_packets=194, n_bytes=18396, priority=1,in_port="s1-eth3",dl_src=00:00
:00:00:00:03,dl_dst=00:00:00:00:00:04 actions=output:"s1-eth4"
 cookie=0x0, duration=25.544s, table=0, n_packets=25, n_bytes=2394, priority=1,in_port="s1-eth3",dl_src=00:00:00
:00:00:03,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=25.533s, table=0, n_packets=25, n_bytes=2394, priority=1,in_port="s1-eth4",dl_src=00:00:00
:00:00:04,dl_dst=00:00:00:00:00:01 actions=output:"s1-eth1"
 cookie=0x0, duration=24.520s, table=0, n_packets=29, n_bytes=2562, priority=1,in_port="s1-eth1",dl_src=00:00:00
:00:00:01,dl_dst=00:00:00:00:00:03 actions=output:"s1-eth3"
 cookie=0x0, duration=24.520s, table=0, n_packets=29, n_bytes=2562, priority=1,in_port="s1-eth1",dl_src=00:00:00
:00:00:01,dl_dst=00:00:00:00:00:04 actions=output:"s1-eth4"
 cookie=0x0, duration=238.732s, table=0, n_packets=52, n_bytes=3676, priority=0 actions=CONTROLLER:65535
```

Fig. 4. Communication handling by attacker.


wireshark_s1-eth1WH3Y51.pcapng
Fichier Editer Vue Aller Capture Analyser Statistiques Telephonie Wireless Outils Aide
Apply a display filter ... <Ctrl-/>
Source address

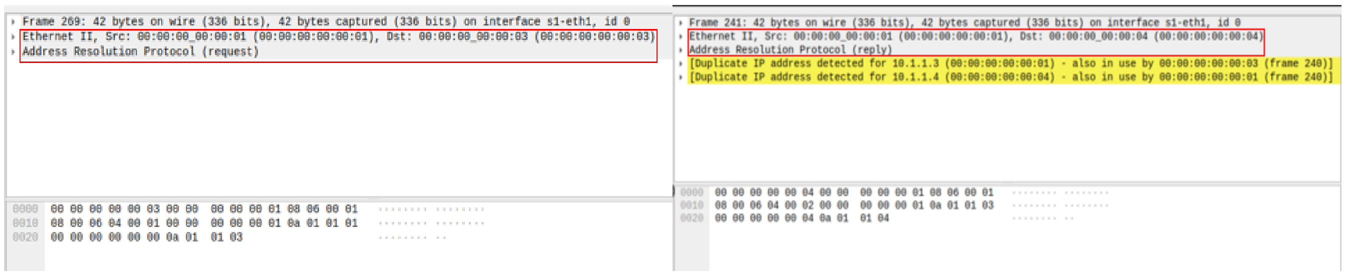
```
No.   Time            Source               Destination          Protocol Length Info
  189 44.059008419    10.1.1.3             1.4                  ICMP       98 Echo (ping) request  id=0x01a3, seq=445/48385, ttl=64 (no res…
  190 44.064861815    10.1.1.3             10.1.1.4             ICMP       98 Echo (ping) request  id=0x01a3, seq=445/48385, ttl=64 (reply …
  191 44.064890700    10.1.1.4             10.1.1.3             ICMP       98 Echo (ping) reply    id=0x01a3, seq=445/48385, ttl=64 (reques…
  192 44.072847094    10.1.1.4             10.1.1.3             ICMP       98 Echo (ping) reply    id=0x01a3, seq=445/48385, ttl=64
  193 44.101655031    00:00:00_00:00:01    00:00:00_00:00:03    ARP        42 10.1.1.4 is at 00:00:00:00:00:01
  194 44.101688273    00:00:00_00:00:01    00:00:00_00:00:04    ARP        42 10.1.1.3 is at 00:00:00:00:00:01 (duplicate use of 10.1.1.4 d…
  195 45.060787526    10.1.1.3             10.1.1.4             ICMP       98 Echo (ping) request  id=0x01a3, seq=446/48641, ttl=64 (no res…
  196 45.064835057    10.1.1.3             10.1.1.4             ICMP       98 Echo (ping) request  id=0x01a3, seq=446/48641, ttl=64 (reply …
  197 45.064875607    10.1.1.4             10.1.1.3             ICMP       98 Echo (ping) reply    id=0x01a3, seq=446/48641, ttl=64 (reques…
  198 45.072844807    10.1.1.4             10.1.1.3             ICMP       98 Echo (ping) reply    id=0x01a3, seq=446/48641, ttl=64
  199 45.284734469    00:00:00_00:00:01    00:00:00_00:00:03    ARP        42 Who has 10.1.1.3? Tell 10.1.1.1
  200 45.284744089    00:00:00_00:00:01    00:00:00_00:00:04    ARP        42 Who has 10.1.1.4? Tell 10.1.1.1
  201 45.284942394    00:00:00_00:00:03    00:00:00_00:00:01    ARP        42 10.1.1.3 is at 00:00:00:00:00:03
  202 45.284960995    00:00:00_00:00:04    00:00:00_00:00:01    ARP        42 10.1.1.4 is at 00:00:00:00:00:04
  203 46.061994505    10.1.1.3             10.1.1.4             ICMP       98 Echo (ping) request  id=0x01a3, seq=447/48897, ttl=64 (no res…
  204 46.064840361    10.1.1.3             10.1.1.4             ICMP       98 Echo (ping) request  id=0x01a3, seq=447/48897, ttl=64 (reply …
  205 46.064883085    10.1.1.4             10.1.1.3             ICMP       98 Echo (ping) reply    id=0x01a3, seq=447/48897, ttl=64 (reques…
  206 46.072846896    10.1.1.4             10.1.1.3             ICMP       98 Echo (ping) reply    id=0x01a3, seq=447/48897, ttl=64
  207 47.063992849    10.1.1.3             10.1.1.4             ICMP       98 Echo (ping) request  id=0x01a3, seq=448/49153, ttl=64 (no res…
  208 47.064859201    10.1.1.3             10.1.1.4             ICMP       98 Echo (ping) request  id=0x01a3, seq=448/49153, ttl=64 (reply …
  209 47.064888685    10.1.1.4             10.1.1.3             ICMP       98 Echo (ping) reply    id=0x01a3, seq=448/49153, ttl=64 (reques…
  210 47.072865541    10.1.1.4             10.1.1.3             ICMP       98 Echo (ping) reply    id=0x01a3, seq=448/49153, ttl=64
  211 47.334529119    fe80::200:ff:fe00:2  ff02::2              ICMPv6     70 Router Solicitation from 00:00:00:00:00:02
  212 48.065979779    10.1.1.3             10.1.1.4             ICMP       98 Echo (ping) request  id=0x01a3, seq=449/49409, ttl=64 (no res…

› Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface s1-eth1, id 0
› Ethernet II, Src: 00:00:00_00:00:03 (00:00:00:00:00:03), Dst: 00:00:00_00:00:01 (00:00:00:00:00:01)

0000  00 00 00 00 00 01 00 00  00 00 00 03 08 00 45 00   ········ ······E·
0010  00 54 97 09 40 00 40 01  8d 97 0a 01 01 03 0a 01   ·T··@·@· ········
0020  01 04 08 00 7f d7 01 a3  01 91 15 18 7d 64 00 00   ········ ····}d··
0030  00 00 16 a5 0d 00 00 00  00 00 10 11 12 13 14 15   ········ ········

wireshark_s1-eth1WH3Y51.pcapng          Paquets: 2258 · Affichés: 2258 (100.0%)          Profile: Default
```

Fig. 5. Packets capture in wireshark.



```
› Frame 269: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface s1-eth1, id 0
› Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:03 (00:00:00:00:00:03)
› Address Resolution Protocol (request)

0000  00 00 00 00 00 03 00 00  00 00 00 01 08 06 00 01   ········ ········
0010  08 00 06 04 00 01 00 00  00 00 00 01 0a 01 01 01   ········ ········
0020  00 00 00 00 00 00 0a 01  01 03                     ········ ··
```

(a) Capturing request body from attacker

```
› Frame 241: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface s1-eth1, id 0
› Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:04 (00:00:00:00:00:04)
› Address Resolution Protocol (reply)
› [Duplicate IP address detected for 10.1.1.3 (00:00:00:00:00:01) - also in use by 00:00:00:00:00:03 (frame 240)]
› [Duplicate IP address detected for 10.1.1.4 (00:00:00:00:00:04) - also in use by 00:00:00:00:00:01 (frame 240)]

0000  00 00 00 00 00 04 00 00  00 00 00 01 08 06 00 01   ········ ········
0010  08 00 06 04 00 02 00 00  00 00 00 01 0a 01 01 03   ········ ········
0020  00 00 00 00 00 04 0a 01  01 04                     ········ ··
```

(b) Capturing reply body from attacker

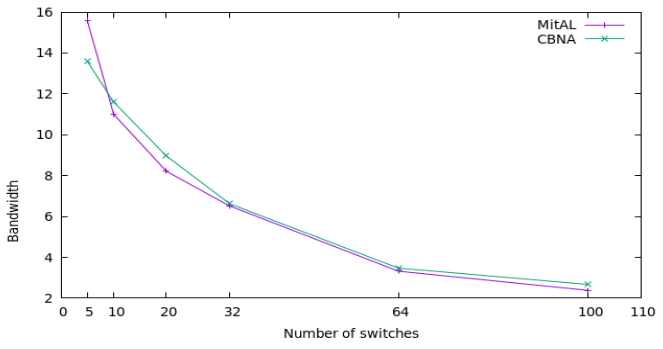
Fig. 6. Data captured from attacker.

TABLE II. Parameters used for Testing

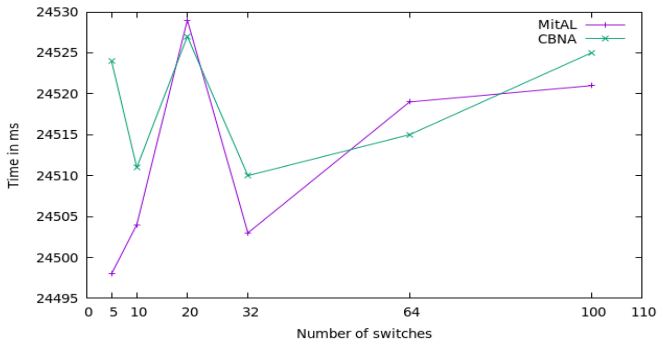
Parameters	Bandwidth	Latency
Topology type	Linear (n switches, n hosts)	Linear (n switches, n hosts)
IP address range for hosts	10.0.0.1-10.0.0.n	10.0.0.1-10.0.0.n
Traffic generating tool	iperf3	ping
Graph generating tool	gnuplot	gnuplot

TABLE III. Comparative Study between some Existing Protocols

Protocol	Type of attacks	Security mesure	Security protocols	Detection rate
Sahri et al. [10] (CAuth)	IP Spoofing	Authentication	TLS	75%
Anass et al. [9] (CBNA)	MITM	Latency	TLS	82%
Our protocol (MitAL)	MITM	Authentication, Latency, Cryptographic Keys	TLS, SSL	86%



(a) Evaluation based on bandwidth



(b) Evaluation based on latency

Fig. 7. Performance evaluation of our algorithm.

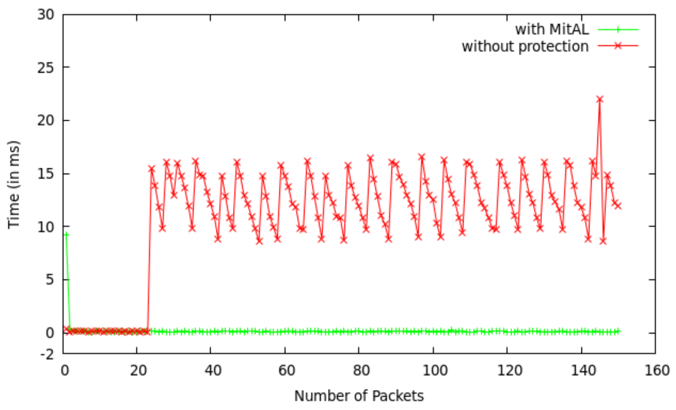


Fig. 8. Traffic comparison.

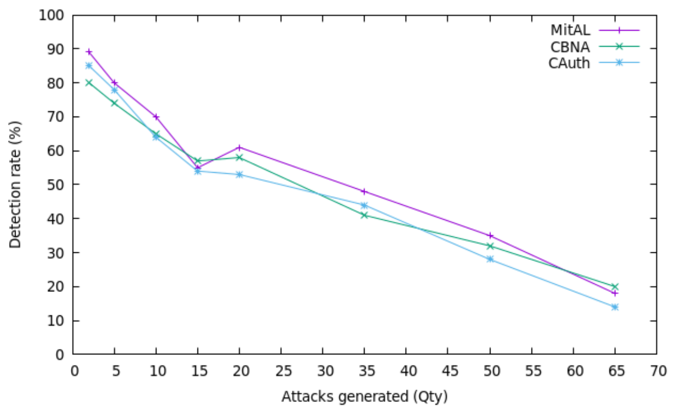
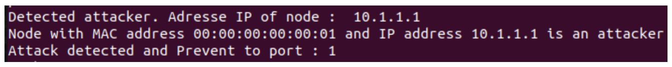


Fig. 9. Detection rate by number of attacks generated.



```
Detected attacker. Adresse IP of node :  10.1.1.1
Node with MAC address 00:00:00:00:00:01 and IP address 10.1.1.1 is an attacker
Attack detected and Prevent to port : 1
```

Fig. 10. System response to block attacker.

G. Discussion

In the previous sections, we presented results that we generated through simulations. The results show that the detection of malicious nodes is efficient. Secondly, the results provide detailed information about the attacks (the MAC addresses, the protocol used, and the data contained in the packets). Once an attack is detected, the protocol displays information about the attacking node and builds a flow rule to block its communication in the network by blocking its connection port as shown in Fig. 10.

As presented in Table III, our proposed MitAL protocol for DNS servers in an SDN architecture aims to address MITM attacks like CBNA [9] unlike CAuth [10] that address the IP Spoofing attacks. The secure methods used by MitAl are authentication, latency and cryptographics keys while CBNA only uses latency and CAuth uses authentication. Furthermore, CAuth and CBNA are based on TLS as a security protocol while MitAl uses both TLS and SSL. Finally, it should be noted that MitAL has a detection rate of 86%, which is higher than the 82% of CBNA and the 75% of CAuth protocols.

V. Conclusion

The DNS is the victim of many kinds of attacks that are difficult to detect and prevent. Among these attacks, are MITM attacks. MITM attacks cause several problems in real-life networks such as tampering, integrity violation and exposure or divulging of confidential or user data. In this paper, we proposed a scheme which is based on SDN architecture. The scheme detects and blocks the MITM attacks on DNS servers. We first presented our designed SDN architecture then we proposed an algorithm that is implemented by the SDN controller to guarantee the security of communications between different trusted hosts of the network, and finally, we presented the simulations results we generated through simulations. The results show that our secure algorithm is effective and efficient in detecting MITM attacks. A comparison with other existing algorithms shows that our proposed solution is superior. We therefore recommend our proposed algorithm (MitAL) as a suitable candidate for use to mitigate the effects of MITM attacks on the DNS server. Despite our results being satisfactory, we believe that the use of Machine Learning techniques can significantly improve the efficiency of our solution. This aspect will be explored in detail in future

References

[1] N. R. Gade and U. Reddy, "A study of cyber security challenges and its emerging trends on latest technologies," 02 2014.

[2] S. Matsumoto, S. Hitz, and A. Perrig, "Fleet: defending sdns from malicious administrators," 08 2014.

[3] J. C. C. Chica, J. C. Imbachi, and J. F. B. Vega, "Security in SDN: A comprehensive survey," *Journal of Network and Computer Applications*, vol. 159, p. 102595, 2020.

[4] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Y. Yau, and J. Wu, "Realtime ddos defense using cots sdn switches via adaptive correlation analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 7, pp. 1838–1853, 2018.

[5] M. Kuhrer, T. Hupperich, J. Bushart, C. Rossow, and T. Holz, "Going wild: Large-scale classification of open dns resolvers," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC'15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 355–368. [Online]. Available: https://doi.org/10.1145/2815675.2815683

[6] A. Khormali, J. Park, H. Alasmary, A. Anwar, M. Saad, and D. Mohaisen, "Domain name system security and privacy: A contemporary survey," *Computer Networks*, vol. 185, p. 107699, 2021.

[7] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Ndss*, vol. 15, 01 2015, pp. 8–11.

[8] C. Li, Z. Qin, E. Novak, and Q. Li, "Securing sdn infrastructure of iot fog networks from mitm attacks," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1156–1164, 2017.

[9] A. Sebbar, K. Zkik, M. Boulmalf, and M. D. E.-C. El Kettani, "New context-based node acceptance cbna framework for mitm detection in sdn architecture," *Procedia Computer Science*, vol. 160, pp. 825–830, 2019.

[10] N. Sahri and K. Okamura, "Protecting dns services from ip spoofing: Sdn collaborative authentication approach," in *Proceedings of the 11th International Conference on Future Internet Technologies*, ser. CFI '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 83–89.

[11] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "Sdn controllers: A comparative study," in *2016 18th mediterranean electrotechnical conference (MELECON)*. IEEE, 2016, pp. 1–6.