L3-ARPSec – A Secure Openflow Network Controller Module to control and protect the Address Resolution Protocol

Rogério Leão Santos de Oliveira, Ailton Akira Shinoda, Christiane Marie Schweitzer, Rogério Luiz Iope and Ligia Rodrigues Prete

Abstract—**The Address Resolution Protocol (ARP) is used to map IP addresses to MAC addresses in local area networks. This protocol has some security vulnerabilities and one of them is the Man-in-the-Middle (MITM) attack. Software-Defined Networks (SDNs) represent an innovative approach in the area of computer networks, since they propose a new model to control forwarding and routing data packets that navigate the Internet. This paper presents the module L3-ARPSec, a set of instructions written in the Python programming language that proposes a way to control the switching of ARP messages and also mitigates the MITM attack in local area networks.**

Keywords*— Software-Defined Network; OpenFlow; ARP cache poisoning; MITM.*

I. INTRODUCTION

It is evident how important computer networks have been nowadays. The vast majority of people's activities are, directly or indirectly, centered on the use of resources or services offered by communication networks.

On one hand, this worldwide network of computers known as the Internet provides communication around the world; on the other hand, it makes users dependent, so that it faces a problem. The level of maturity in its structure also reduced its flexibility.

Implementation of new technologies often requires hardware device replacement such as routers and switches, which can be very expensive and very hard-working for network administrators.

The research community of computer networks has been searching for solutions that enable the use of networks with more programmable resources and less dependence on constant replacement of hardware elements, so that new technologies designed to solve new problems can be gradually inserted into the network and without significant costs.

It is exactly in this context that the new paradigm known as Software-Defined Networking (SDN) [1] arose. It is a structure that aims to preserve the current performance on routing and forwarding data packets, since it maintains the actual structure with dedicated routers as it is, but at the same time it delegates the orchestration of the network elements to a new component, the controller.

This new structure controlled by applications manages the packet transfer in a network and does not interfere in the current protocols of network layers, such as ARP, IP, TCP, UDP, and HTTP. Thus, known security problems that affect the current structure will also exist in the new SDN-based structure. One of them is the attack known as Man-in-the-Middle (MITM), in which an attacker using ARP cache poisoning stays in the middle of an end-to-end communication link. After configuring and establishing the attack, the attacker can view all messages exchanged between the victims.

This paper presents a programming module to run in SDN controllers, which aims to control the ARP messaging and to mitigate the MITM attack on local area networks. The remaining sections are organized as follows. We begin in Section II by explaining the ARP protocol operation and how the MITM attack occurs in detail. In Section III, all L3-ARPSec module features are presented. The results are discussed in section IV, and related works are mentioned in Section V. Finally, in section VI the conclusion and suggestions for future works are drawn.

II. BACKGROUND

A. Address Resolution Protocol

All devices connected to a TCP/IP network are identified in the Network layer by its 32-bit Internet Protocol (IP) address and in the Medium Access Control (MAC) layer by its 48-bit MAC address. In a communication, when the network layer receives a packet from higher layers to send it to a particular IP address, it checks if the destination address is at the same local network as that of the sending machine. If so, the packet is delivered to the Medium Access Control layer that sends it through the appropriate physical media. For this to happen, it is necessary that the sender also knows the destination MAC address.

This management is accomplished by the ARP protocol that automatically maps IP addresses to MAC addresses and inserts them into each host inside a temporary table called ARP cache. When needed, any host can search in its own table to find address entries. To manage these entries in the ARP cache there are two messages types: ARP Request and ARP Reply.

If the desired mapping of a MAC address to a specific IP address is not found in ARP cache, an ARP request is broadcast to all hosts in the same Local Area Network (LAN) as the sending host [2]. This message contains the IP and the sender's MAC addresses, the type of the message (an ARP Request) and the target IP address. The ARP request message reaches all hosts and each one analyzes the frame. If the host's own IP address is not equal to the target IP address of the ARP Request message, the host just discards the frame. Conversely,

Rogério Leão Santos de Oliveira, Ailton Akira Shinoda, Christiane Marie Schweitzer, Rogério Luiz Iope and Ligia Rodrigues Prete¸ Departament of Electrical Engineering, State University "Julio de Mesquita Filho", Ilha Solteira-SP, Brazil, E-mails: rogerio.leao@fatec.sp.gov.br, shinoda@dee.feis.unesp.br, chris@mat.feis.unesp.br, rogerio@ncc.unesp.br, ligia.prete@fatec.sp.gov.br. This work is supported in part by the FAPESP under Grant 2014/06022-9.

if it is equal, then the host sends an ARP Reply message directly to the sender to inform its MAC address. When the ARP Reply packet arrives the sender, its ARP cache table is updated. This procedure is repeated whenever it is necessary to update the ARP cache tables in various hosts of the LAN. An example of the ARP request and reply is as follows:

1. Host A wants to send data packets to host D, but host A only knows the IP address of host D; it should also know its MAC address.
2. Host A broadcasts an ARP Request with the IP address of host D as target.
3. All hosts on the local network receive the ARP Request and check the frame. All hosts discard the request packet, except host D.
4. Host D replies to A with its MAC address and also updates its own ARP cache table with the IP and MAC addresses of host A.
5. Host A receives the response (ARP Reply) from D and updates its ARP cache table with the IP and MAC addresses of host D.
6. Now host A and D can deliver packets directly between them because the ARP cache tables have the proper mapping addresses.

An important observation is that these entries into the ARP cache tables are not permanent, so after a short period of time, around two minutes, the hosts must repeat the previous procedure and update their ARP cache tables again.

B. *ARP cache poisoning attacks and MITM*

The ARP cache poisoning occurs when an attacker maliciously modifies the mapping of an IP address to its own MAC address on the ARP cache of other hosts [2]. This technique, also called ARP spoofing, is performed by sending spoofed ARP Reply messages on the network.

As shown in figure 1, host C that represents the attacker, sends ARP Reply messages to hosts A and B, mapping its own MAC address to the IP addresses of each one of the victims. The frame sent to B contains the IP address of host A and the MAC address of host C, and the frame sent to A contains the IP address of host B and the MAC address of host C.

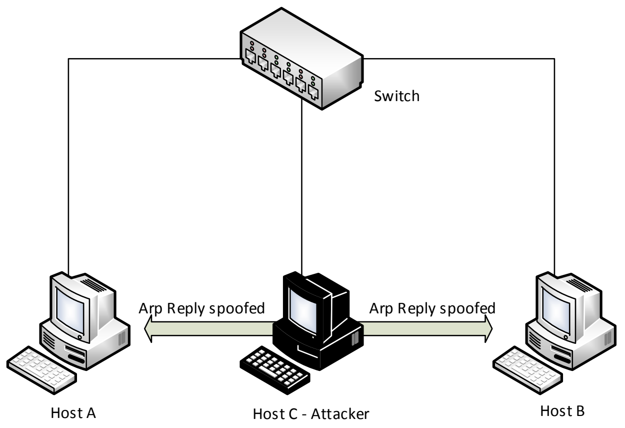


Fig. 1. Example of an MITM attack.

As a result, hosts A and B will update their ARP cache tables with the spoofed addresses. So when A sends data packets to B, these packets will contain the destination attacker's MAC address, and the same happens when B is sending data to A. Thus, the attacker stays in the middle of communication and all traffic between hosts A and B will pass through him. After reading or copying their contents, the attacker's host discreetly forwards data packets to proper destination, so A and B can normally communicate and maybe not realize that they are victims of the MITM attack.

C. *Software-Defined Networks and the MITM attack*

Software Defined Networks have an important feature: the possibility of programming rules and policies that orchestrates packet forwarding. These rules are implemented in the controller that manages all the switches connected to it. This paper describes a way of mitigating the MITM attack on a local area network that uses the OpenFlow [3, 4] protocol, by proposing a mechanism to control the switching of ARP messages between hosts.

D. *Scenario and tests*

To generate a realistic testing scenario in a practical way, we have created a virtual network in an SDN simulator called Mininet [5, 6, 13] containing: 3 hosts, 1 switch, 1 attacker host running BackTrack5 [7, 8] Linux operating system and one POX SDN controller. Figure 2 illustrates this scenario.

In the first test, we configured the L2_learning module in the POX controller, which implements similar features of a common layer-2 switch. With the network running, connection tests were performed between all pairs of hosts using ICMP echo requests (ping). These tests confirmed the communication between all pairs of nodes. After a few seconds analyzing incoming packets in various ports of the switch, the controller configures flow rules in the switch, so that hosts become able to exchange information directly. With the rules implemented in the switch, the packets sent from host A to host B only pass through the ports where they are connected on switch, so neither host C nor the attacker host can view these packets. This self-learning feature is very important because the rules implemented in the switch allow that the next packets will be transferred between known hosts, without the need of being searched in the controller, which increases the network performance.

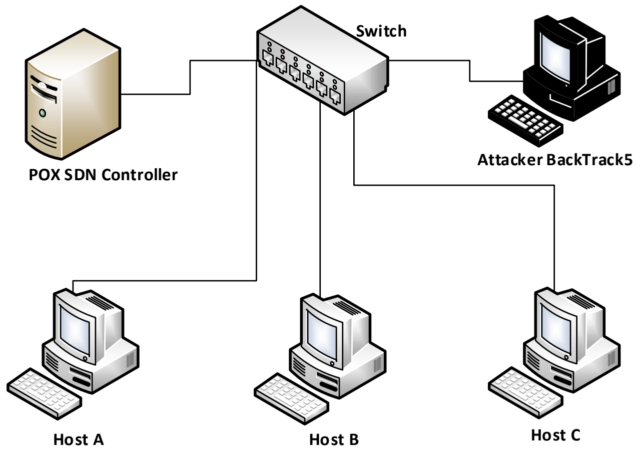


Fig. 2. Test scenario.

After that, we configured the attacker host to execute the MITM attack on hosts A and B by using the following commands:

```
# arpspoof -i eth0 -t 10.0.0.1 10.0.0.2
# arpspoof -i eth0 -t 10.0.0.2 10.0.0.1
```

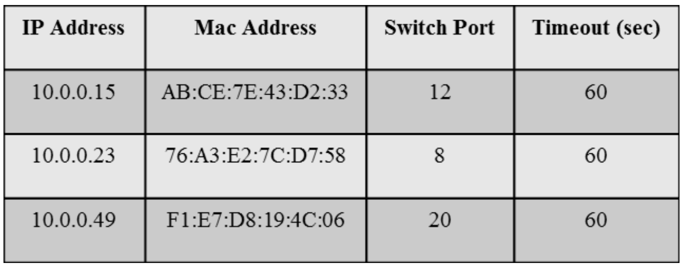
These commands send ARP Reply messages to hosts A and B continuously each second, which poison the ARP caches and the spoofed attacker's MAC address is updated to the victims's ARP caches. After this, all communication between the victims is intercepted by the attacker, which quickly forwards all packets to the correct destination in order to not be perceived.

III. THE L3-ARPSEC MODULE

The L3-ARPSec module is a set of instructions written in the Python programming language that runs on the controller. It proposes a way to control the switching of ARP messages while mitigates MITM attacks on local area networks. The following features are fundamental and guide its functioning:

1. No changes are made to the original ARP protocol, thus no host needs to be configured individually.
2. All the ARP type packets received in any port of the switch will be forwarded to the controller.
3. Only the controller can send ARP Reply messages; therefore, no host will reply an ARP Request to another host directly.

When executed on the controller, the L3-ARPSec module instantiates two virtual tables: one called ArpTable and another called ArpTableCandidate. The structures of these tables are shown in Figure 3.



IP Address	Mac Address	Switch Port	Timeout (sec)
10.0.0.15	AB:CE:7E:43:D2:33	12	60
10.0.0.23	76:A3:E2:7C:D7:58	8	60
10.0.0.49	F1:E7:D8:19:4C:06	20	60

Fig. 3. ArpTable and ArpTableCandidate structure.

A standard feature of the OpenFlow protocol is that when a packet arrives at any port of the switch and finds no matching rule, it is sent to the controller. How the L3-ARPSec module processes each packet received is shown in the flowchart of Figure 4 and explained below.

A. Sending all packets to Self-Learning Function.

This function extracts, from the frame header, the source IP address, the source MAC address and the switch port number into which this packet entered. Then it seeks an entry in the ARPTable with the corresponding IP address. On one hand, if not found, the address is inserted in this table. On the other hand, if found, the function also seeks the ARPTableCandidate and inserts it in this second table if not found. The ARPTableCandidate is used by the Timers to detect attacks and will be explained in subsection D.

B. IP packet type.

If the packet type is IP, the function seeks an entry in the ArpTable with the corresponding IP address. If found, the controller creates a forward rule message containing its own packet source addresses and destination addresses found in the table and then sends it to the switch. As a result, all subsequent packets from the same source and destination address will not need to go through the controller, but it will be automatically forwarded by the switch. This flow rule installed in the switch has a time to live called *iddle_time* with a standard value set to 20 seconds. It means that if no more packets use this flow rule in this interval, the switch will discard the flow rule.

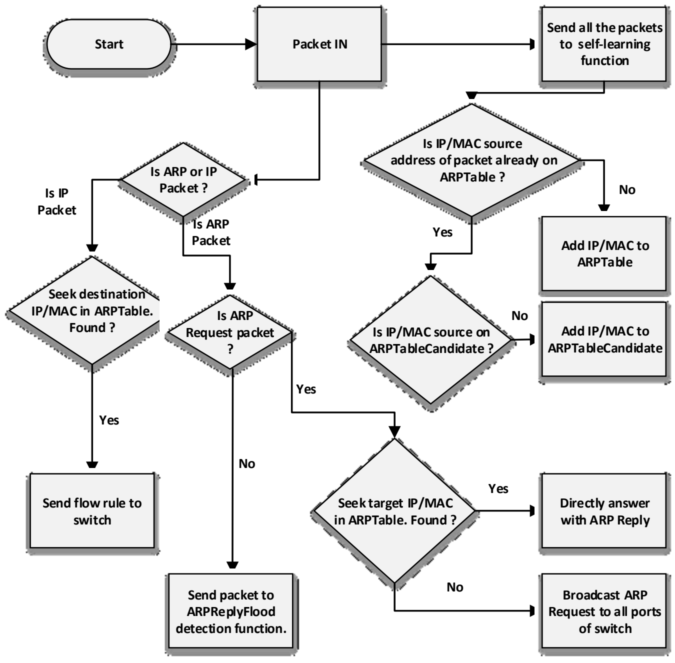

Start
Packet IN
Send all the packets to self-learning function
Is IP/MAC source address of packet already on ARPTable ?
No
Yes
Add IP/MAC to ARPTable
Is ARP or IP Packet ?
Is IP Packet
Is ARP Packet
Seek destination IP/MAC in ARPTable. Found ?
Is ARP Request packet ?
Is IP/MAC source on ARPTableCandidate ?
No
Add IP/MAC to ARPTableCandidate
Yes
Yes
No
Send flow rule to switch
Seek target IP/MAC in ARPTable. Found ?
Yes
Directly answer with ARP Reply
No
Send packet to ARPReplyFlood detection function.
Broadcast ARP Request to all ports of switch

Fig. 4. L3-ARPSec module flowchart.

C. ARP packet type and flooding detection attack.

If the packet type is ARP, the function also checks if it is an ARP Request. If so, it seeks the target IP address in ARPTable. If found, it creates an ARP Reply message containing the corresponding MAC address and sends it directly to the requesting host. If not found in ARPTable, it broadcasts the ARP Request message to all ports of the switch. If the ARP packet represents an ARP Reply message, it is just sent to the Flooding-ARP Reply detection function. This function computes the average time interval for the last five packets received. Moreover, if this value is less than 3 seconds, it means that an attack has been attempted, because some host is flooding the network with ARP Reply packets. The reason for choosing this average time interval is that in a normal situation, ARP requests are not sent by hosts in large numbers and in short periods of time. These values represent an initial constant for the algorithm and can be adjusted in further researches. The procedure shown in Figure 5 represents the detection algorithm of this function.

```
Procedure Flooding-ARP Reply detection (packet)
BEGIN:
   if (MAC address of packet is in ARPReplyFlood) then
        Increase the amount number
        if (amount number is equal to 5) then
            if (average time interval greater than 3) then
                Detected attack/ Block MAC address
            Del MAC address of packet from ARPReplyFlood
   else
        Add MAC address of packet to ARPReplyFlood with
        time value and amount number = 1
END:
```

Fig. 5. Flooding-ARP Reply detection procedure.

In this case, a function that temporarily blocks the source MAC address is invoked. The controller sends a flow rule to the switch that refuses all packets with the same attacker's MAC address during two minutes. Also, any entries in

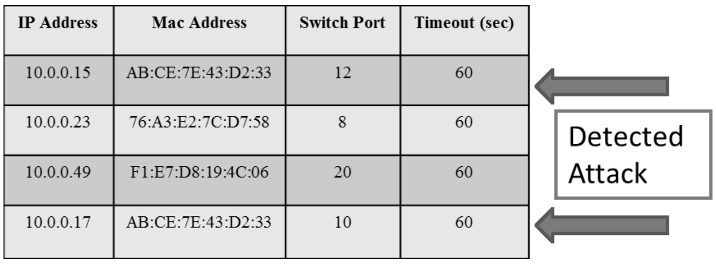
ARPTable or ARPTableCandidate containing the same MAC address will be deleted.

D. Timers and detection per tables analysis.

In a LAN, it is possible that hosts change their IP addresses, the connection port on the switch, or even their MAC address. If the entries in ARPTable and ARPTableCandidate were permanent, these changes would not be possible and always the first entry for a host would not allow future modifications. To avoid this, there is a timeout value in each entry. A function runs every 25 seconds to delete each entry in both tables whose timeout has expired. The default timeout value is set to 20 seconds.

In some cases, the Flooding-ARP Reply detection function explained earlier may be unable to detect the attack, since the attacker can send ARP Reply messages in fractions of seconds and the algorithm may fail to detect. Therefore, another detection mechanism of ARP attacks is to assess the entries in the ARPTable and ARPTableCandidate.

One automatic function runs every 13 seconds and scans the tables, looking for entries containing different IP addresses with the same MAC address. If found, it means that some host is using its own IP address and trying to impersonate any other host in the local network. It also denotes that an attack is occurring and the same blocking function used by the Flooding-ARP Reply detection function will be invoked, temporarily blocking the attacker's MAC address. An example of the suspected entries is shown in Figure 6.



IP Address	Mac Address	Switch Port	Timeout (sec)
10.0.0.15	AB:CE:7E:43:D2:33	12	60
10.0.0.23	76:A3:E2:7C:D7:58	8	60
10.0.0.49	F1:E7:D8:19:4C:06	20	60
10.0.0.17	AB:CE:7E:43:D2:33	10	60

Detected Attack

Fig. 6. Suspected entries in ArpTable or ArpTableCandidate.

IV. VALIDATION TESTS

In our tests, the L3-ARPSec module remained stable and could mitigate MITM attacks and control the switching of ARP messages in the network.

The tests performed as described in section II, using the L2_learning module at network controller, demonstrated that software-defined networks is also susceptible to MITM attack. Therefore, it was necessary to create a module not only to efficiently manage the switching of packets, but also to protect all the hosts from MITM attacks.

Several tests were performed with the L3-ARPSec module proposed in this paper. First, the basic functionalities were tested by checking that packets were switched between all hosts without any problem, proving that the network control and the mapping of IP and MAC addresses were succeed.

After that, the attacker host executed MITM attacks to intercept messages from other network hosts. The module L3-ARPSec could detect the attacks and temporarily isolated the attacker host. Both the Flooding-ARP Reply detection function and the analysis periodically made in ARPTable and ARPTableCandidate were efficient in detecting and reacting to attacks.

A. Results for Flooding-ARP Reply detection function.

Using the same scenario shown in Figure 2, the attacker sent ARP Reply messages to the network continuously each second. The command line used by the attacker was as follows:

```
# arpspoof -i eth0 -t 10.0.0.1 10.0.0.2
```

Each ARP reply message was received by the switch and because no rule that could forward them was defined by default, these ARP packets were sent to the controller. Through the Flooding-ARP Reply detection function, the L3-ARPSec module implemented in the controller was counting each message received and when the fifth identical message has been computed, the attack was detected.

After the detection, immediately the controller sent a flow rule to the switch to temporarily block the attacker. The switch instantly began to discard any packet coming from that sender, leaving him completely inert in the network for a period of two minutes. The existing mappings in ARPTable and ARPTableCandidate virtual tables containing the attacker's MAC address were also excluded, so that no other host was fooled by this fraudulent address.

All this traffic was collected for approximately 5 minutes (300 seconds) and data were compiled in a line graph as shown in Figure 7.

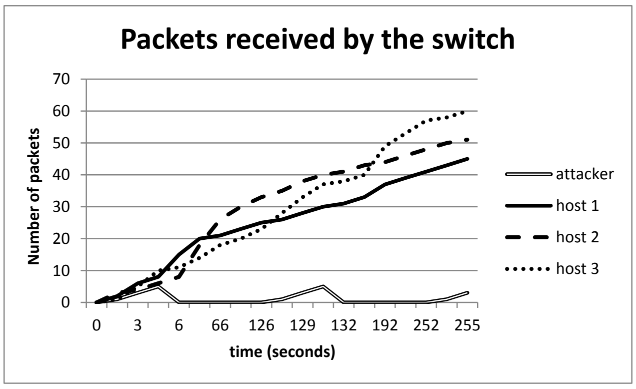


Fig. 7. Packet traffic under ARP flooding-attack.

We can see that hosts 1, 2 and 3 were gradually increasing the flow of packets sent to the switch. However, the attacker host during network monitoring has been blocked two times. As the flooding detection feature counts every five packages and the attacker was sending one packet every second, the first blockade occurred in the fifth second. In the next two minutes after the blockade, the attacker could not send packets, but when the punishment time finished, the attack started again, as can be seen in second 126. Again the Flooding-ARP Reply detection function counted the messages sent by the attacker and immediately after five packets received, the blockade occurred again, leaving him punished and inert in the network for more two minutes.

The values set as two minutes for attacker punishment and five for the number of packets received represent an initial choice; future work may use different values in order to achieve better results.

B. Results for detection function based on timers and tables.

Using the same scenario previously configured, the attacker performed attacks in a different way. Instead of flooding the network by sending ARP reply messages constantly every and each one second, it sends packets at longer intervals, approximately every 4 seconds.

The attacker sends ARP replies containing its own IP address (10.0.0.50) and its own MAC address (00:11:FF:AA:BB:50). In addition to that, it also sends fraudulent ARP replies containing the IP address of the victim (10.0.0.1) and again its own MAC address (00:11:FF:AA:BB:50).

As the attacker sent fraudulent messages in 4-second intervals, the Flooding-ARP Reply detection function was not able to detect the attack and the fraudulent mapping was then installed in ARPTable. After that, any host in the network, that initiates a communication with the victim host (IP 10.0.0.1), will do it with the attacker in the middle because the fraudulent mapping will be replicated on the network.

To solve this problem, a table checking routine was implemented to execute every 13 seconds, and this strategy detected the attack. This routine is executed in the module in such a way that every 13 seconds it scans the tables looking for mappings with different IP addresses, but with the same MAC address.

When the attack is detected, the same punishment used by the Flooding-ARP Reply detection function is invoked, blocking the attacker for two minutes.

As can be seen in Figure 8, although the attacker has escaped from the first detection engine of module (the Flooding-ARP Reply detection function), he was detected two times by the detection function based on timers and tables during this second test. As this routine is performed every 13 seconds, blocking action occurs soon after the entry of the fraudulent mapping in the ARPTable.

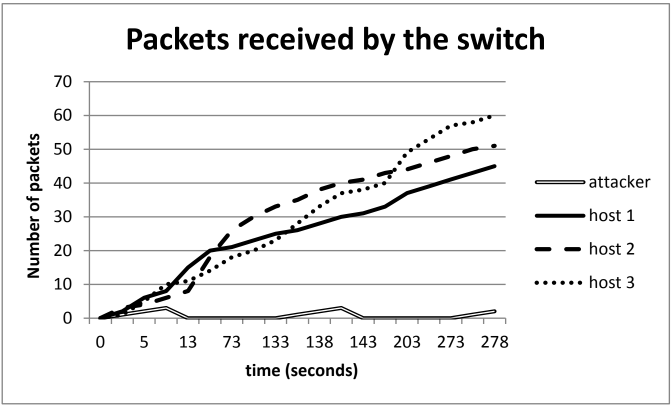


Fig. 8. Packet traffic under ARPTable attack.

V. Other Existing Solutions

Other existing solutions tried to solve the problem of ARP cache poisoning attacks. Some of them, based on cryptography [9, 10], have caused serious performance degradation and were not fully compatible with the standard ARP. The approach proposed by Tripunitara et al. [11] is not practical since it would require changes on all hosts in the network. The solution proposed by Gouda et al. [12] was the most ambitious ones, but it is limited to static networks and also requires changes on all hosts in the network and needs complex installations.

All the cited proposals based their solutions on the structure of traditional networks and not on software-defined networks. Thus, the module proposed in this paper represents an innovative approach to solve an old local area computer network problem.

VI. Conclusion and Future Work

In the first sections of this paper, we presented the SDN features and functionalities. After that, we explained the ARP protocol functioning on a local area computer network. Later, security attacks were showed on the ARP protocol, in particular the MITM attack that can compromise both traditional local area networks and local software-defined networks.

The possibility of programming the SDN controller opens opportunities for the development of new features to manage networks, as we have done in this research. Section three showed the L3-ARPSec module, a set of instructions that controls the switching of ARP messages on a local area network and mitigates the MITM attacks when implemented in an SDN controller.

We referenced other proposal solutions in section five, and although we have found several research works related to attacks on the ARP protocol, we have only found proposals to solve the problem in traditional networks and not in software-defined networks.

After performing a series of tests and simulations, we concluded that the L3-ARPSec module proposed in this paper achieved its objectives, and it represents an open field of research to be explored and improved further. Future work could test this module in larger networks, perform stress tests and rewrite this module in other programming languages to support other SDN controllers.

References

[1] Siamak Azodolmolky, Software Defined Networking with OpenFlow. Packt Publishing Ltd, 2013.

[2] Roney Philip, “Securing Wireless Networks from ARP Cache Poisoning,” (2007).Master's Projects. Paper 131.

[3] M. Nick et al., “OpenFlow: enabling innovation in campus networks”, ACM SIGCOMM Computer Communication, vol. 38, no. 2, pp. 69-74, Apr. 2008.

[4] OpenFlow (2013, Jul). OpenFlow 1.3.2 Specification. [Online]. Available:https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.2.pdf

[5] L. Bob et al., “A network in a laptop: rapid prototyping for softwaredefined networks.” In Proceedings of the Ninth ACM SIGCOMM.

[6] Mininet. (2013, Mar). An Instant Virtual Network on your Laptop (or other PC). [Online]. Available: http://mininet.org/

[7] Jason Dean, Backtrack. Hachette UK, 2013.

[8] Willie Pritchett, BackTrack 5 Cookbook. Packt Publishing Ltd, 2012.

[9] D. Bruschi, A. Omaghi and E. Rosti, “S-ARP: a secure address resolution protocol,” in Proceedings of the 19th Annual Computer Security Applications Conference, December 2003.

[10] W. Lootah, W. Enck and P. McDaniel, “TARP: Ticket-based address resolution protocol,” in Proceedings of the 21st Annual Computer Security Applications Conference, December 2005.

[11] M. Tripunitara and P. Dutta, “A middleware approach to asynchronous and backward compatible detection and prevention of ARP cache poisoning,” in Proceedings of the 15th Annual Computer Security Applications Conference, December 1999.

[12] Mohamed G. Gouda and Chin-Tser Huang, “A secure address resolution protocol” in the International Journal of Computer and Telecommunications Networking, Computer Networks, Elsevier, Volume 41, Issue 1, pages: 57-71, January, 2003.

[13] de Oliveira, Rogerio Leao Santos; Schweitzer, Christiane Marie; Shinoda, Ailton Akira; Ligia Rodrigues Prete, "Using Mininet for emulation and prototyping Software-Defined Networks," Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on , vol., no., pp.1,6, 4-6 June, 2014.