

AMERICAN UNIVERSITY OF BEIRUT

DEPARTMENT OF COMPUTER SCIENCE



CMPS396Y: COMPUTER SECURITY PROJECT

Detection and Mitigationn of ARP Spoofing Attack in Software-Defined Networks (SDN)

Authors

Zoulfikar Shmayssani

Hiyam Ghannam

Supervisor

Dr. Wassim El Hajj

Contents

1 Paper

- 1.1 Description of the ARP spoofing in SDN
- 1.2 Explanation of the detection and mitigation mechanism
- 1.3 Explanation of experiments and Results

2 Implementation of the Detection and Mitigation module

- 2.1 Major snippets of the code we used

3 Detailed Experiment on ARP Spoofing in SDN

- 3.1 Experimental Setup
- 3.2 Simulating the Attack using Mininet
- 3.3 Results of the Experiment

4 Running the Mitigation Module that is created

- 4.1 ARP spoofing attack is detected and mitigated
- 4.2 Verification and Test

5 Conclusion and Future Work

6 References

Detection and Mitigation of ARP Spoofing Attack in Software defined Networks (SDN)

Dr. Wassim El Haj, Zoulfikar Shmayssani, Hiyam Ghannam

Abstract—Software-defined networking (SDN) is an approach to cloud computing that facilitates network management and enables programmatically efficient configuration[1]. The security of the information that is passed between the users in the network is an important concern. ARP spoofing or ARP cache-poisoning attack is found in LAN networks, and there is no efficient way to mitigate in the traditional networks[2]. However, SDN networks can allow us to detect and mitigate this attack in an efficient one without any change in the network. In this paper we will explain an efficient way for detecting and mitigating ARP Request and Reply attacks. This solution is based on including an ARP module in the SDN controller, we used POX controller, which detects and terminates the attack.

I. SOFTWARE DEFINED NETWORKS

Software Defined Networking is a network architecture model that allows the management and the control of the network in a programmatic way. This architecture is based on the idea of separating the data plane from the control plane (network brain) in routers and switches. Therefore the control part is taken from hardware and implemented in software[1].

II. ARP SPOOFING IN SDN

A. ARP Definition

ARP is abbreviation of Address Resolution Protocol which is used for mapping an Internet Protocol address (IP) to a physical machine address that is recognized in the local network. This protocol is implemented between the Data Link Layer and the Network layer of the Open Systems Interconnection model (OSI) [3]. ARP cache table is used in networks to manage and store the mappings of <IP,MAC> of the hosts in the network.

B. ARP Spoofing Attack

ARP spoofing attack is used to do cache poisoning by inserting falsified <IP,MAC> mappings in victims ARP cache. The attack has two types which are: Request and Response attacks.

ARP request attack: Sending ARP request packet with forged source <IP,MAC>. Once it is received by the victim it updates its ARP cache table with the forged <IP,MAC>. When the victim wants to send packets they will be traversed to the forged IP, the packets will be go to the MAC of the host of the forged <IP,MAC>[2].

ARP reply attack: This attack can happen in two cases. The first one is when the attacker replies to genuine ARP request with forged ARP reply that results in linking the IP to the attacker MAC address [2]. The second one is sending spoofed ARP replies without sending requests.

C. ARP Spoofing consequences

ARP attack is considered one of the most dangerous attacks on the network since it is the base of launching other attacks. It can be used in the following important attacks:

- 1) *Man-in-the-middle attacks:* ARP spoofing attack is used to in MITM attacks that is considered one of the most dangerous network attacks, in order to intercept and modify the packets and the data transferred of the victim.[4]
- 2) *Denial-of-service attacks:* It is based on linking multiple IP addresses with a single target's MAC address. The traffic will be sent to the target's MAC address this will overload the target host with traffic.[4]
- 3) *Session hijacking:* This type of attacks uses ARP spoofing to sniff session IDs. In addition to that, it can be used for giving the attackers access to secret information and system calls[4]

III. DETECTION AND MITIGATION OF ARP SPOOFING ATTACKS IN SDN

The controller in SDN network is considered the brain of the whole network and it is implemented pragmatically. This gives the opportunity to modify the code of the controller in order to add some rules to detect and prevent various attacks. In our implementation we used SDN POX controller which is an open source development platform for Python-based (SDN) control applications[5]. We will modify this controller pragmatically using Python to analyze and detect the attack signature. Then prevent ARP cache spoofing or poisoning of the hosts in the same SDN. The POX controller examines if the ARP packets are legitimate after separating them into Request and Reply types and check if they satisfy certain rules to be considered malicious

A. Detection of ARP Request Attacks

ARP request packets will be considered spoofed if one of these conditions is met[2]:

- Different Source MAC address of Ethernet header and source MAC address of ARP header.
- Destination IP in the ARP header is not found in the known hosts list of the DHCP.
- There is no match between the MAC address binding present in the known hosts list for the Source IP of the ARP header and the source MAC address of the ARP header.

B. Detection of ARP Reply Attacks

ARP reply packets will be considered spoofed if one of these conditions is met[2]:

- Source MAC address of ethernet header and source MAC address of ARP header are not the same.
- Destination MAC address of ethernet header and destination MAC address of ARP header are not the same.
- The MAC address binding present in the known hosts list for the Source IP of the ARP header doesn't match with the Source MAC address of the ARP header.
- The MAC address binding present in the known hosts list, for the Destination IP of the ARP header doesn't match with the Destination MAC address of the ARP header.
- Destination MAC address of the ethernet header has a value of broadcast.

C. Mitigation of ARP Spoofing Attack

If an ARP request or reply packet met one of its above conditions the POX controller will consider it as an attack signature. It will drop the packets and stop the incoming request from the malicious or the forged sources of ARP requests and replies.

IV. IMPLEMENTATION AND EXPERIMENTS

We done the experiments using Ubuntu on a Virtual Machine and we used the following:

- Mininet which is a network emulator which creates a network of virtual hosts, switches, controllers, and links. It supports SDN.[6].
- POX controller is used for the SDN.
- DHCP server is implemented in the POX controller, which is a protocol used to provide quick, automatic, and central management for the distribution of IP addresses[7].

V. EXPERIMENT RESULTS

We used arp-spoof tool that is available in Mininet. This tool allows us to do arp spoofing attack and poisoning the arp cache table of a certain host.

A. ARP spoofing without implementing the module

We did an experiment after setting the environment and creating the network topology. We were able to do arp spoofing attack on a host in SDN and we did MITM attack on that host.

B. ARP spoofing after running the secure module

After implementing and running the proposed detection and mitigation modules on POX controller, the malicious ARP request and reply messages will be detected. It will drop the malicious packets and stop the incoming requests from the malicious source.

```

def drop (duration = None):
    """
    Drops this packet and optionally installs a flow to continue
    dropping similar ones for a while
    """
    if duration is not None:
        if not isinstance(duration, tuple):
            duration = (duration,duration)
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet)
        msg.idle_timeout = duration[0]
        msg.hard_timeout = duration[1]
        msg.buffer_id = event.ofp.buffer_id
        self.connection.send(msg)
    elif event.ofp.buffer_id is not None:
        msg = of.ofp_packet_out()
        msg.buffer_id = event.ofp.buffer_id
        msg.in_port = event.port
        self.connection.send(msg)

```

Experiment ARP Spoofing Attack

Environment setup:

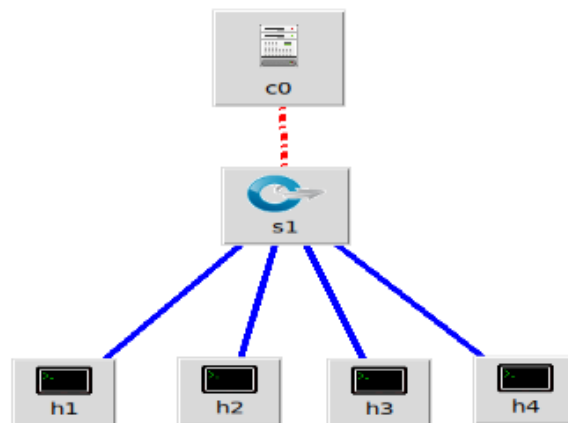
We done the experiments using Ubuntu on a Virtual Machine and we used the following:

- Download Mininet, which is a network emulator which creates a network of virtual hosts in SDN, on Ubuntu .
- Download POX controller that is used for the SDN.

Network Topology Used:

We will be using the following network topology for applying an ARP spoofing/poisoning attack.

We will be having four different hosts (**h1**, **h2**, **h3**, **h4**) that are each connected to a switch **S1**. The switch is further connected to a central controller **CO**. We will be using POX controller for this experiment.

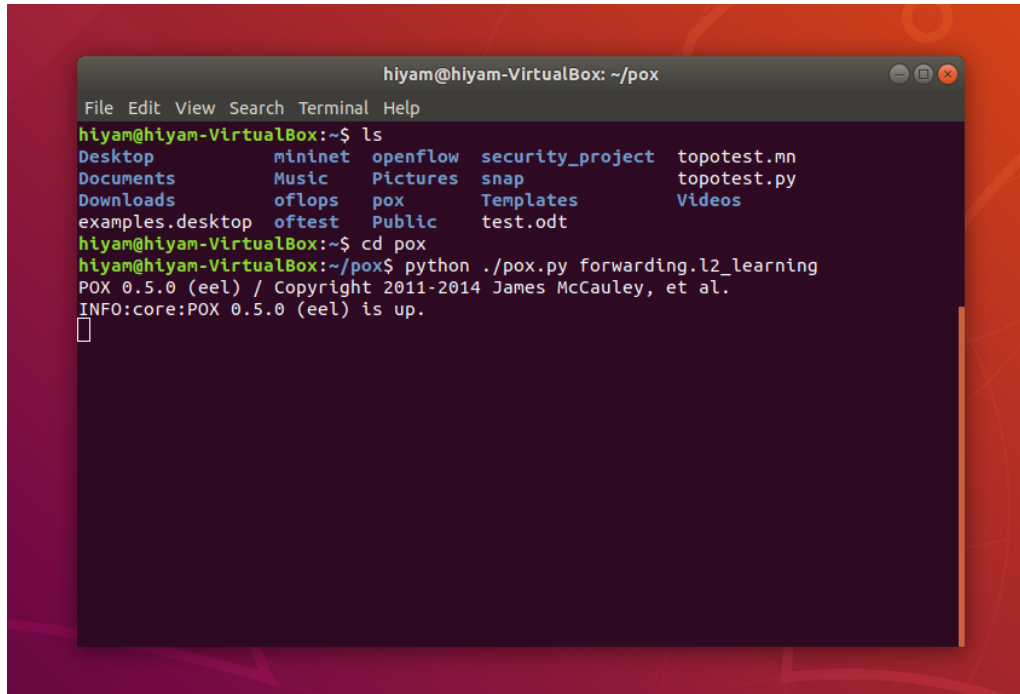


Connecting to POX Controller

Here, we will be connecting our network topology as stated above to a POX controller.

I. Initialize the POX Controller

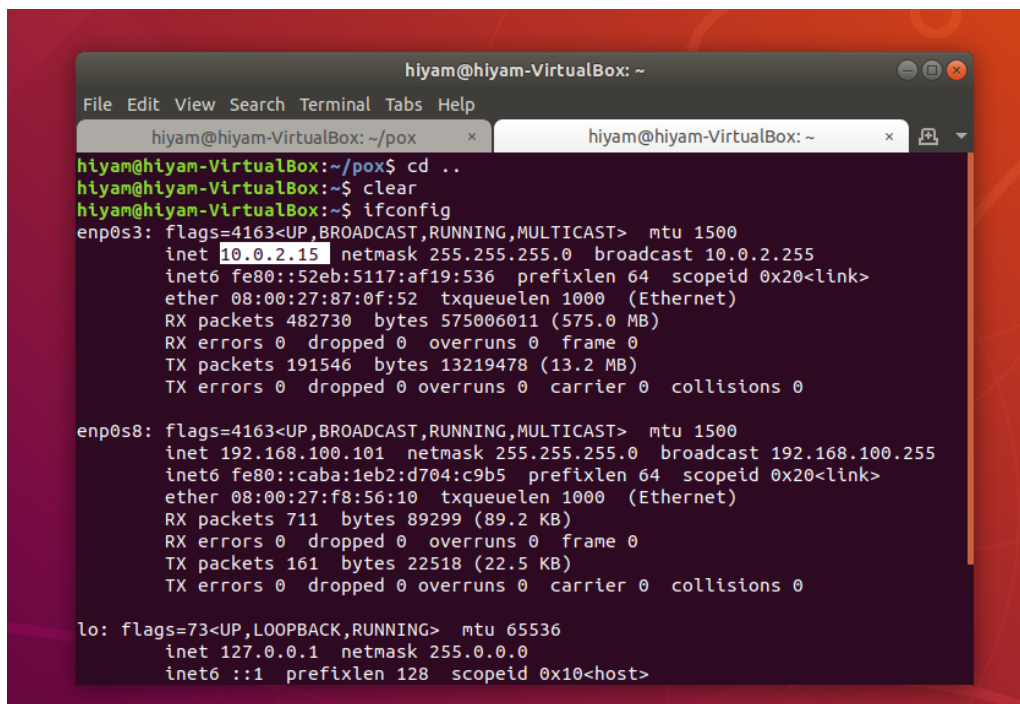
Using the command: `python ./pox.py forwarding-L2-learning`



```
hiyam@hiyam-VirtualBox: ~/pox
File Edit View Search Terminal Help
hiyam@hiyam-VirtualBox:~$ ls
Desktop      mininet      openflow     security_project  toptest.mn
Documents    Music        Pictures     snap              toptest.py
Downloads    oflops      pox          Templates         Videos
examples.desktop  ofttest    Public       test.odt
hiyam@hiyam-VirtualBox:~$ cd pox
hiyam@hiyam-VirtualBox:~/pox$ python ./pox.py forwarding.l2_learning
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
█
```

II. Get the IP Address of the Controller

In order to do that, we use the 'ifconfig' command. Apparently, the IP address is **10.0.2.15**



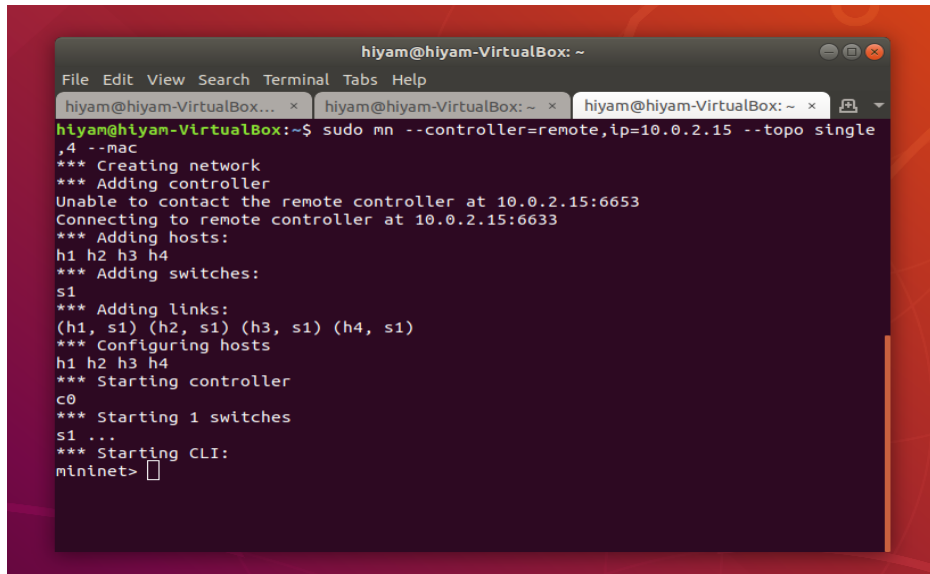
```
hiyam@hiyam-VirtualBox: ~
File Edit View Search Terminal Tabs Help
hiyam@hiyam-VirtualBox:~/pox x hiyam@hiyam-VirtualBox: ~ x
hiyam@hiyam-VirtualBox:~/pox$ cd ..
hiyam@hiyam-VirtualBox:~$ clear
hiyam@hiyam-VirtualBox:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::52eb:5117:af19:536 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:87:0f:52 txqueuelen 1000 (Ethernet)
    RX packets 482730 bytes 575006011 (575.0 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 191546 bytes 13219478 (13.2 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.100.101 netmask 255.255.255.0 broadcast 192.168.100.255
    inet6 fe80::caba:1eb2:d704:c9b5 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:f8:56:10 txqueuelen 1000 (Ethernet)
    RX packets 711 bytes 89299 (89.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 161 bytes 22518 (22.5 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
```

III. Connect the network Topology to the Controller

Using the command: `sudo mn --controller=remote,ip=10.0.2.15 --topo single,4 --mac`

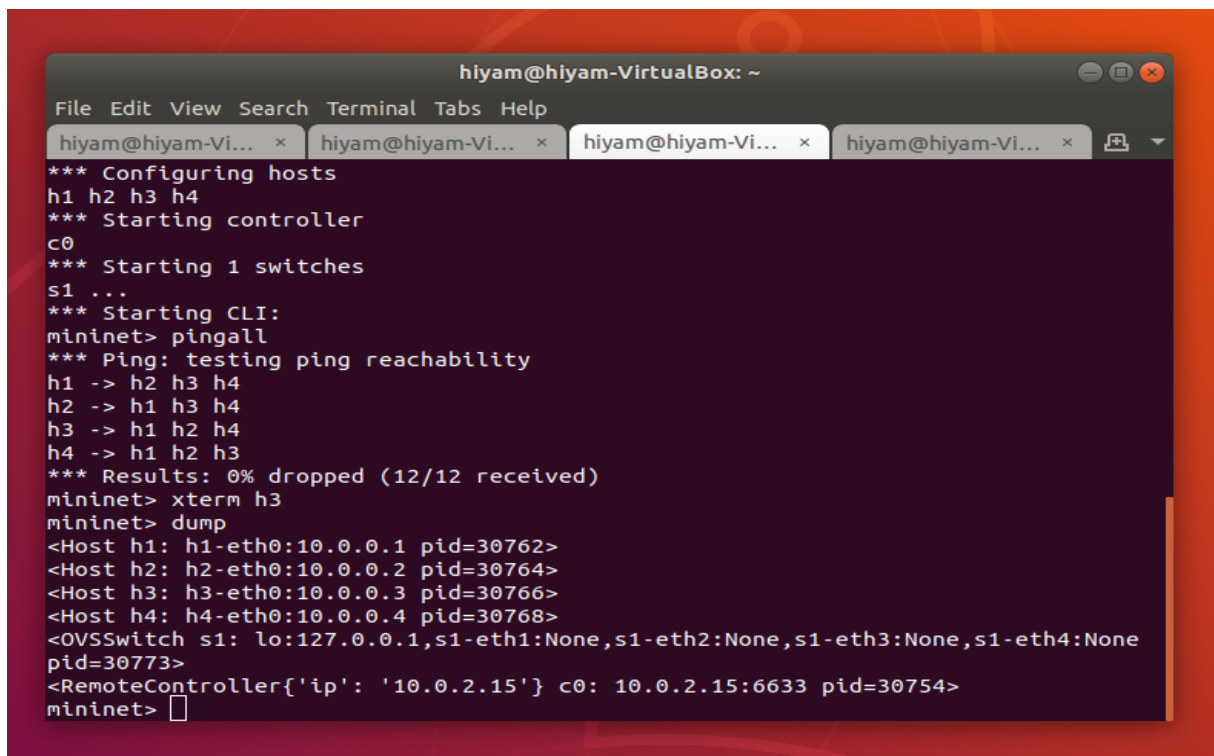


```
hiyam@hiyam-VirtualBox: ~  
File Edit View Search Terminal Tabs Help  
hiyam@hiyam-VirtualBox: ~  
hiyam@hiyam-VirtualBox: ~  
hiyam@hiyam-VirtualBox: ~  
hiyam@hiyam-VirtualBox: ~$ sudo mn --controller=remote,ip=10.0.2.15 --topo single  
,4 --mac  
*** Creating network  
*** Adding controller  
Unable to contact the remote controller at 10.0.2.15:6653  
Connecting to remote controller at 10.0.2.15:6633  
*** Adding hosts:  
h1 h2 h3 h4  
*** Adding switches:  
s1  
*** Adding links:  
(h1, s1) (h2, s1) (h3, s1) (h4, s1)  
*** Configuring hosts  
h1 h2 h3 h4  
*** Starting controller  
c0  
*** Starting 1 switches  
s1 ...  
*** Starting CLI:  
mininet>
```

IV. Maintaining ARP cache entries

After we have connected to the POX controller, and before we do the attack, we have to populate the cache with the appropriate ARP entries that has <IP, MAC> mappings, since initially the cache is empty.

To do so, we will use the **pingall** command in Mininet, which will send a ping request from each host to all other existing hosts as shows below:



```
hiyam@hiyam-VirtualBox: ~  
File Edit View Search Terminal Tabs Help  
hiyam@hiyam-Vi... x hiyam@hiyam-Vi... x hiyam@hiyam-Vi... x hiyam@hiyam-Vi... x  
*** Configuring hosts  
h1 h2 h3 h4  
*** Starting controller  
c0  
*** Starting 1 switches  
s1 ...  
*** Starting CLI:  
mininet> pingall  
*** Ping: testing ping reachability  
h1 -> h2 h3 h4  
h2 -> h1 h3 h4  
h3 -> h1 h2 h4  
h4 -> h1 h2 h3  
*** Results: 0% dropped (12/12 received)  
mininet> xterm h3  
mininet> dump  
<Host h1: h1-eth0:10.0.0.1 pid=30762>  
<Host h2: h2-eth0:10.0.0.2 pid=30764>  
<Host h3: h3-eth0:10.0.0.3 pid=30766>  
<Host h4: h4-eth0:10.0.0.4 pid=30768>  
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-eth4:None  
pid=30773>  
<RemoteController{'ip': '10.0.2.15'} c0: 10.0.2.15:6633 pid=30754>  
mininet>
```

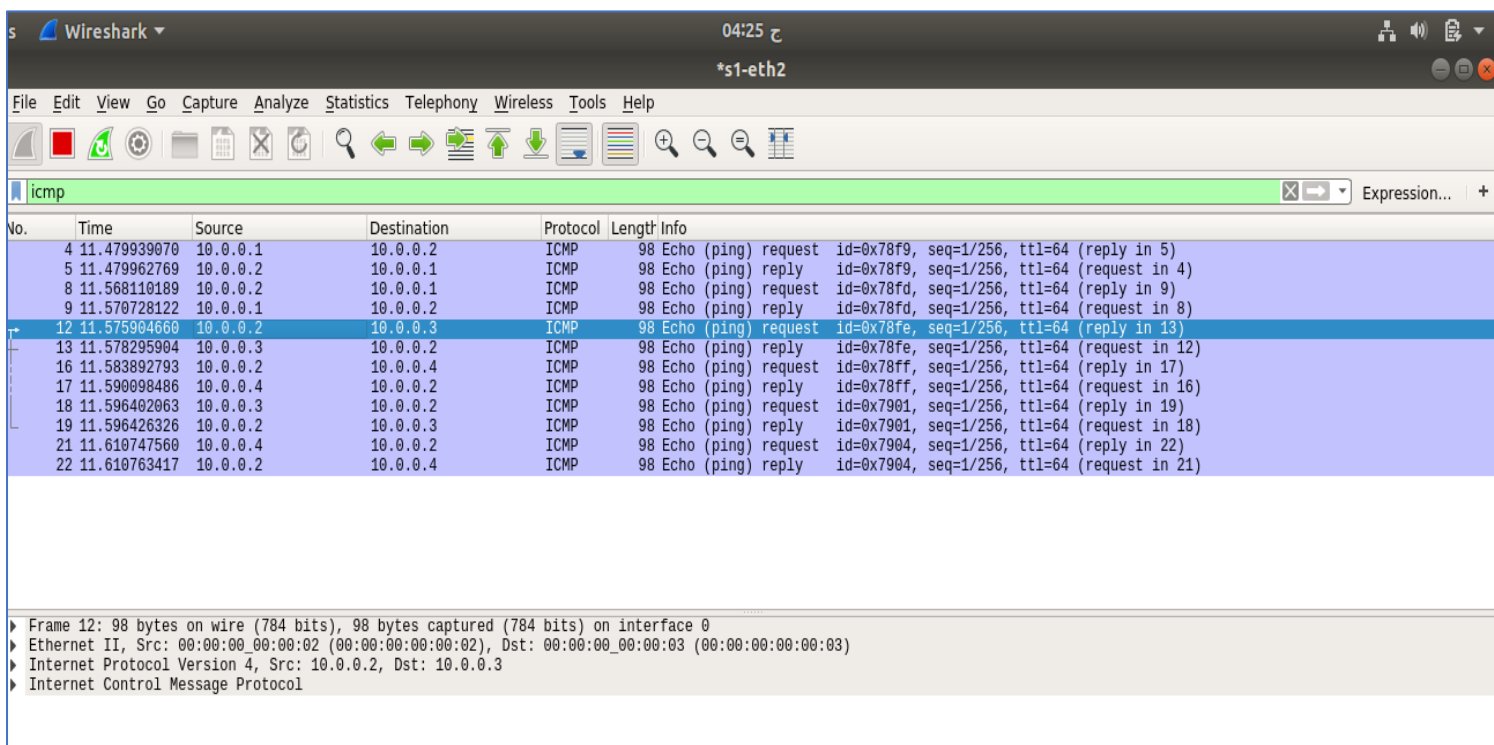

We also used the **dump** tool in Mininet in order to check the IP addresses of each host available in our network, and as shown above, we got the following:

Host	IP Address
h1	10.0.0.1
h2	10.0.0.2
h3	10.0.0.3
h4	10.0.0.4

Figure 1 – IP address of each host

V. Checking ping requests and replies in Wireshark

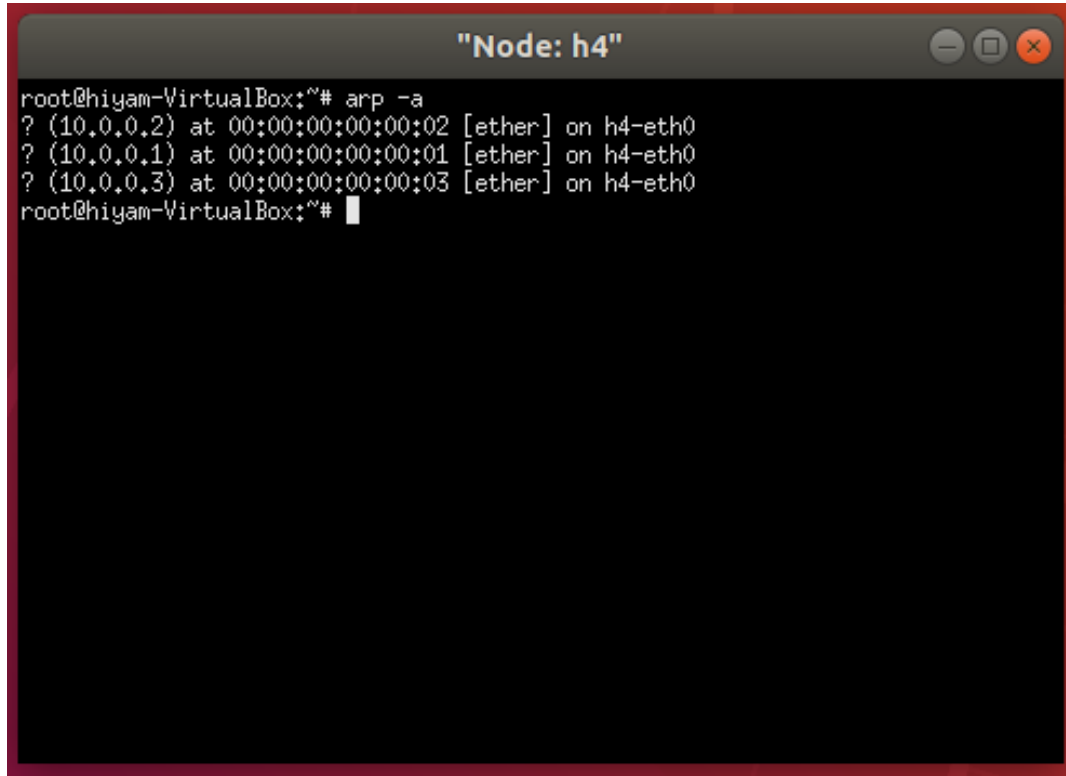
For testing purposes, we will sniff packets using Wireshark on the interface of host2 (**h2**)



We can clearly see that **h2 (10.0.0.2 ~ we are connecting to h2's interface)** has got an ICMP request followed by ICMP replies from all other hosts (10.0.0.1, 10.0.0.3, and 10.0.0.4), which reflects the ping request sent from h2 to all other hosts when we used the **pingall** command in Mininet.

<IP, MAC> mappings

Before proceeding with doing the attack, we will check the MAC address of each of the available hosts: open node of **h4** and use the command **arp-a** to see the arp table of the connected hosts **h1**, **h2**, and **h3**



```
root@hiyam-VirtualBox:~# arp -a
? (10.0.0.2) at 00:00:00:00:00:02 [ether] on h4-eth0
? (10.0.0.1) at 00:00:00:00:00:01 [ether] on h4-eth0
? (10.0.0.3) at 00:00:00:00:00:03 [ether] on h4-eth0
root@hiyam-VirtualBox:~#
```

Host	IP Address	MAC Address
h1	10.0.0.1	00:00:00:00:00:01
h2	10.0.0.2	00:00:00:00:00:02
h3	10.0.0.3	00:00:00:00:00:03
h4	10.0.0.4	00:00:00:00:00:04

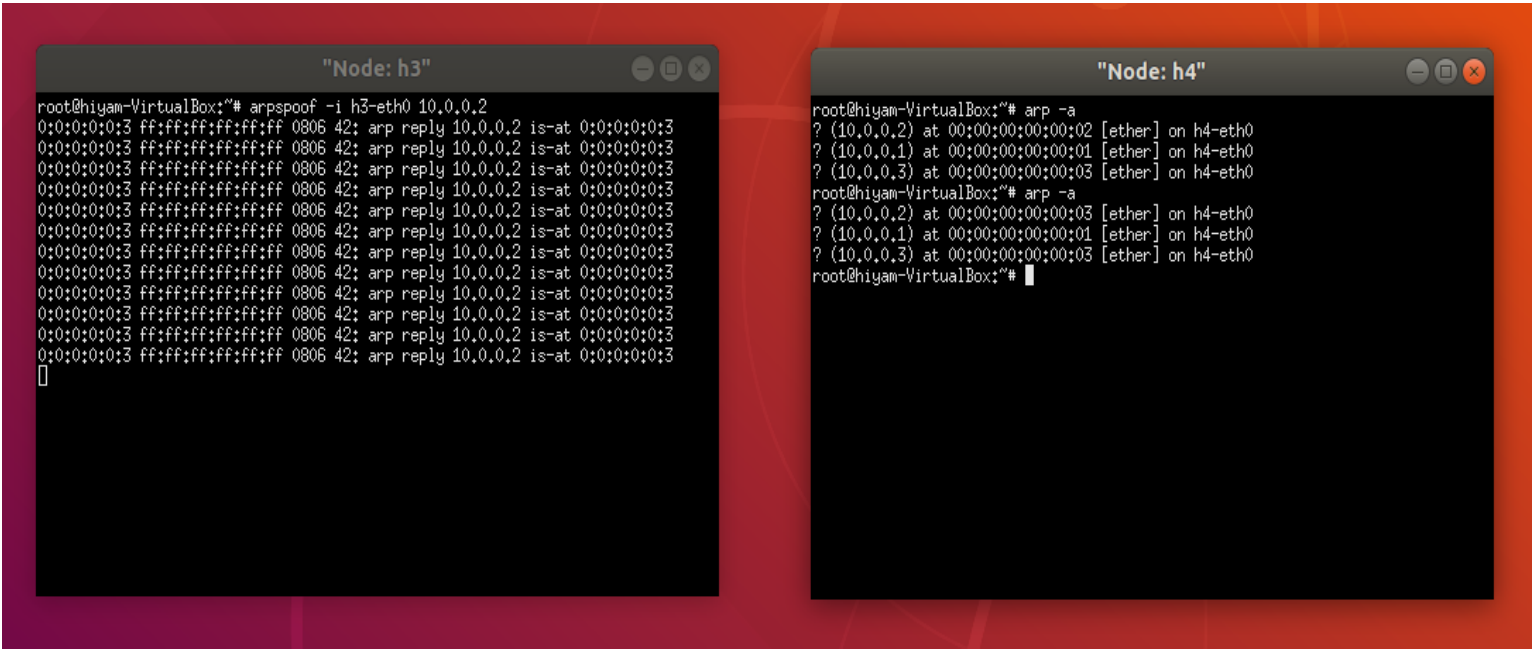
Figure 2 – IP & MAC addresses of each host

ARP Poisoning

Now we will do the ARP Poisoning attack. We will assume host **h3** is the attacker and the victim to be host **h2**. We will tamper with the ARP entry for **h2**, where we will change the MAC address of **h2** and make it become the MAC address of **h3**.

This is done using following command: “*arp spoof – I attacker_ interface victim_IP*”

Arpspoof command will handle doing the arp spoofing/poisoning as shown below:

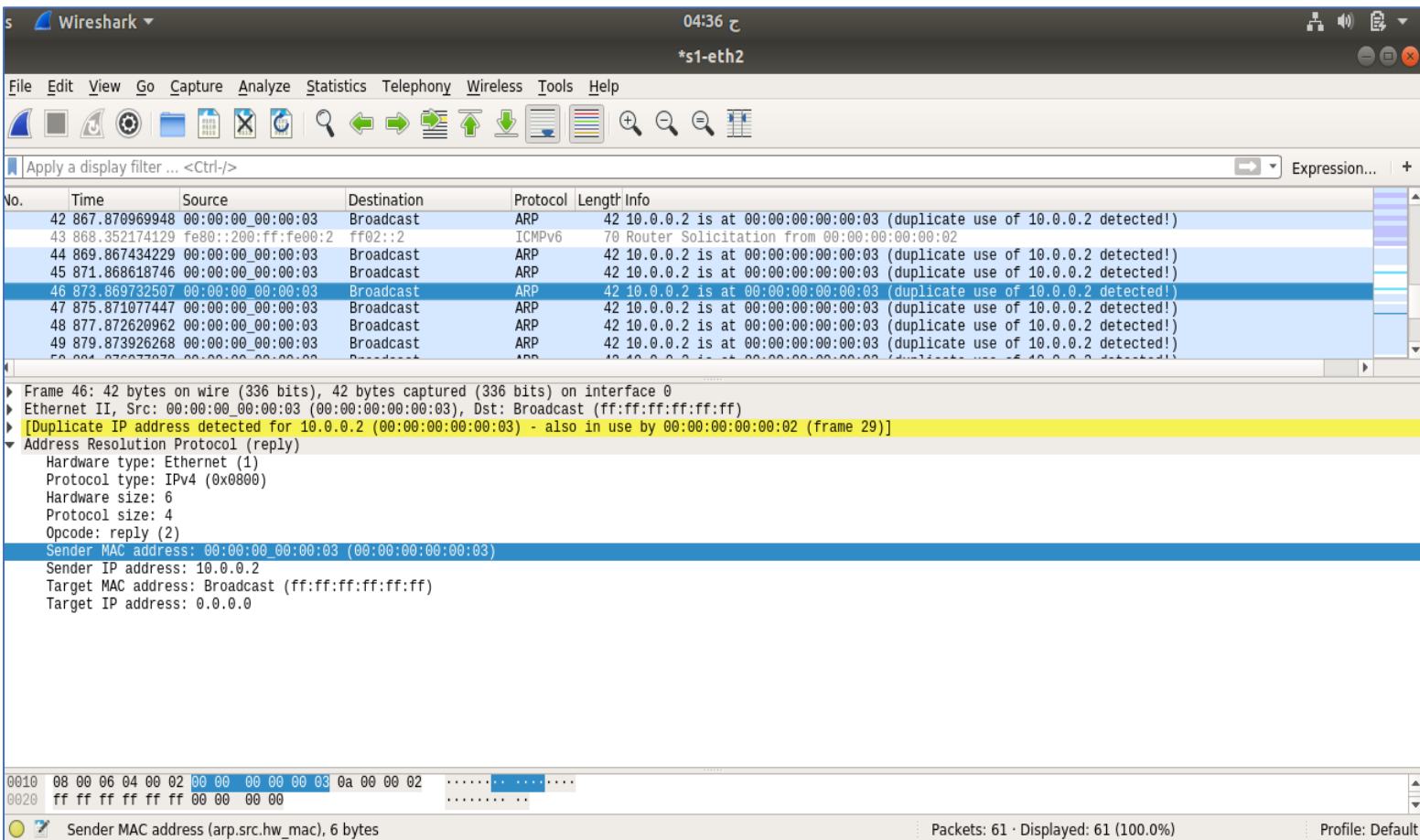


Here we can see that the command has changes the MAC address of h2 from 00:00:00:00:00:02 to 00:00:00:00:00:03. This is also visible in the “Node: h4” window on the right where we can see the MAC address of h2 (with IP address 10.0.0.2) before and after the attack and how it has changed

Host	Before ARP Poisoning		After ARP Poisoning	
	IP Address	MAC Address	IP Address	MAC Address
H1	10.0.0.1	00:00:00:00:00:01	10.0.0.1	00:00:00:00:00:01
H2	10.0.0.2	00:00:00:00:00:02	10.0.0.2	00:00:00:00:00:03
H3	10.0.0.3	00:00:00:00:00:03	10.0.0.3	00:00:00:00:00:03
H4	10.0.0.4	00:00:00:00:00:04	10.0.0.4	00:00:00:00:00:04

Figure 3 – IP & MAC addresses of each host before and after ARP poisoning

Sniffing the poisoned packets using Wireshark



We can see that the ARP reply request has:

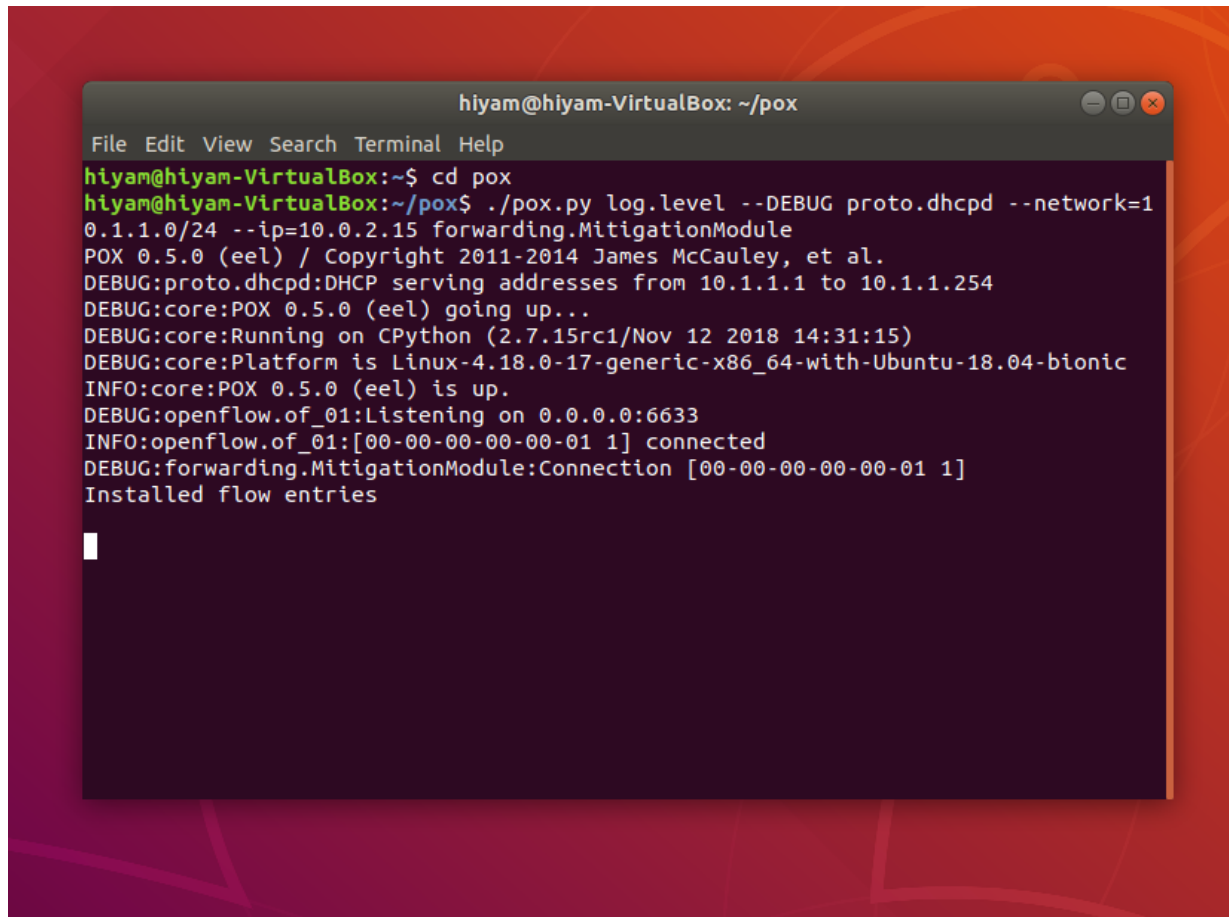
- *Sender MAC address:* 00:00:00:00:00:03 (that of h3)
- *Sender IP address:* 10.0.0.2 (that of h2)

Now the attacker **h3** can receive everything that is intended to be send to **h2**. He can perform Man In the Middle Attack. For example, when we **h1** pings **h2**, the attacker (**h3**) will also receives the ping, and we saw this using Wireshark as demonstrated above.

ARP spoofing after running the secure module

Now we add the Python module that is created (MitigationModule.py) to the POX controller in the Forwarding directory. Then we run our module over the POX controller.

Using: `./pox.py proto.dhcp --network=10.0.1.0/24 --ip=10.0.2.15 forwarding.MitigationModule`

A screenshot of a terminal window titled 'hiyam@hiyam-VirtualBox: ~/pox'. The terminal shows the execution of the command `./pox.py log.level --DEBUG proto.dhcpd --network=10.0.1.0/24 --ip=10.0.2.15 forwarding.MitigationModule`. The output includes several debug and info messages: 'POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.', 'DEBUG:proto.dhcpd:DHCP serving addresses from 10.1.1.1 to 10.1.1.254', 'DEBUG:core:POX 0.5.0 (eel) going up...', 'DEBUG:core:Running on CPython (2.7.15rc1/Nov 12 2018 14:31:15)', 'DEBUG:core:Platform is Linux-4.18.0-17-generic-x86_64-with-Ubuntu-18.04-bionic', 'INFO:core:POX 0.5.0 (eel) is up.', 'DEBUG:openflow.of_01:Listening on 0.0.0.0:6633', 'INFO:openflow.of_01:[00-00-00-00-00-01 1] connected', 'DEBUG:forwarding.MitigationModule:Connection [00-00-00-00-00-01 1]', and 'Installed flow entries'. The terminal window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'.

Now the POX controller in SDN is running with the ARP mitigation module.

If we try to repeat the experiment we did before (ARP spoofing attack), The spoofed ARP packets will be dropped and therefore not changing the ARP cache table of the hosts.

Conclusion: ARP spoofing attack is considered one of the most dangerous attacks that can be done in a network since it is the base of many malicious attacks such as MITM. This attack can be mitigated in an efficient way in SDN through running a module on the SDN Controller.

Future Work: We will calculate the efficiency and the overhead that results from the mitigation module. Also, calculating the time needed to detect and mitigate the spoofed ARP packets.

References

- [1] What is Software Defined Networking? SDN Explained. (n.d.). Retrieved from <https://www.bmc.com/blogs/software-defined-networking>
- [2] AbdelSalam, A. M., El-Sisi, A. B., & Reddy, V. (2016). Mitigating arp spoofing attacks in software-defined networks. In Proceedings of the International conference on Computer Theory and Applications (ICCT).
- [3] What is Address Resolution Protocol (ARP)? - Definition from WhatIs.com. (n.d.). Retrieved from <https://searchnetworking.techtarget.com/definition/Address-Resolution-Protocol-ARP>
- [4] ARP Spoofing. (2019, May 09). Retrieved from <https://www.veracode.com/security/arp-spoofing>
- [5] What is POX? - Definition from WhatIs.com. (n.d.). Retrieved from <https://searchnetworking.techtarget.com/definition/POX>
- [6] <http://mininet.org/overview>
- [7] <https://www.lifewire.com/what-is-dhcp-2625848>