Project Report 3: SDN-Based DoS Attacks and Mitigation

Student Name: Marco Ermini
Email: mermini@asu.edu
Submission Date: 28th Jun, 2021
Class Name and Term: CSE548 Summer 2021

I. Project Overview

In this lab I am emulating Denial of Service (DoS) attacks in an SDN networking environment. DDoS Attacks can target various components in the SDN infrastructure. I am setting up an SDN-based firewall environment based on containernet, POX controller, and Over Virtual Switch (OVS). To mitigate DoS attacks, I have developed a "port security" solution to counter the implemented DoS attacks.
In the lab I am implementing firewall filtering rules in order to implement the required firewall security policies, along with a sequence of screenshots and corresponding illustrations to demonstrate how I have fulfilled the firewall's packet filtering requirements.

All the files and configurations used for this lab have been uploaded on GitHub; references are provided throughout the text and in the Appendix A at the bottom of the file.

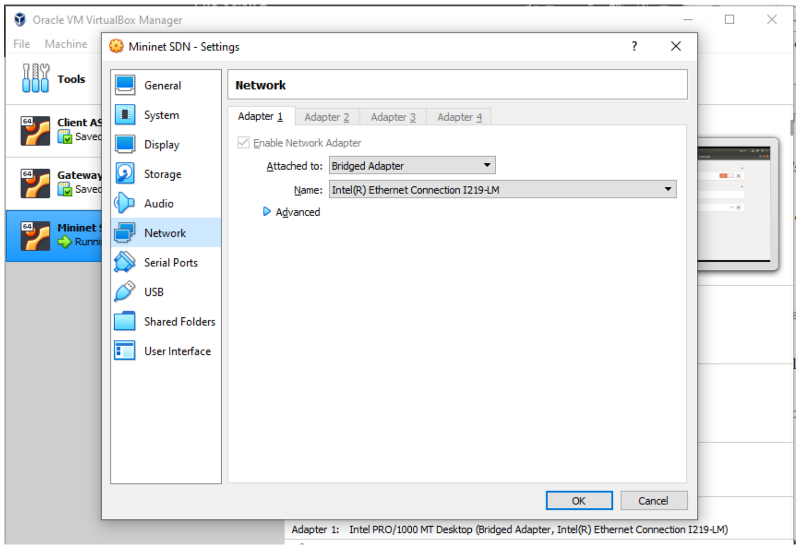
II. Network Setup

Since I have experienced some issues with connecting the VM to my lab through my host PC running Windows 10, I have chosen to set up the VM in VirtualBox in a bridged network configuration.

In this way, I could avoid configuring a static IP address to the VM and simply opted to fetch an IP for the VM via DHCP; this IP is then assigned directly to my router, which can be useful to troubleshoot eventual connectivity issues. On the Ubuntu/Linux side this has brought no issues whatsoever once the configuration commands are adjusted (e.g. use "dhclient br0" rather than assigning an IP address with "ifconfig br0").

Because of the use of DHCP, depending on the lab run and when I restarted the VM, it may have assumed a different address in the 192.168.1/24 network. This does not affect the outcome of the lab exercises, but it may look inconsistent in the screenshots. Apologies about that.

Please find below the initial set-up of the virtual infrastructure as I have configured it in VirtualBox.



Network

Enable Network Adapter

Attached to: Bridged Adapter

Name: Intel(R) Ethernet Connection I219-LM

Adapter 1: Intel PRO/1000 MT Desktop (Bridged Adapter, Intel(R) Ethernet Connection I219-LM)

Figure 1 - Bridged network setup in VirtualBox

III. Software

For this first lab, the following software has been used:

- Various network tools (specifically: ping, hping3, and nping)
- POX (GitHub link: https://noxrepo.github.io/pox-doc/html/
- Open vSwitch: http://www.openvswitch.org/
- Open vSwitch Cheat Sheet: https://therandomsecurityguy.com/openvswitch-cheatsheet/
- Containernet: https://containernet.github.io/
- Containernet tutorial: https://github.com/containernet/containernet/wiki/Tutorial:-Getting-Started

IV. Project Description

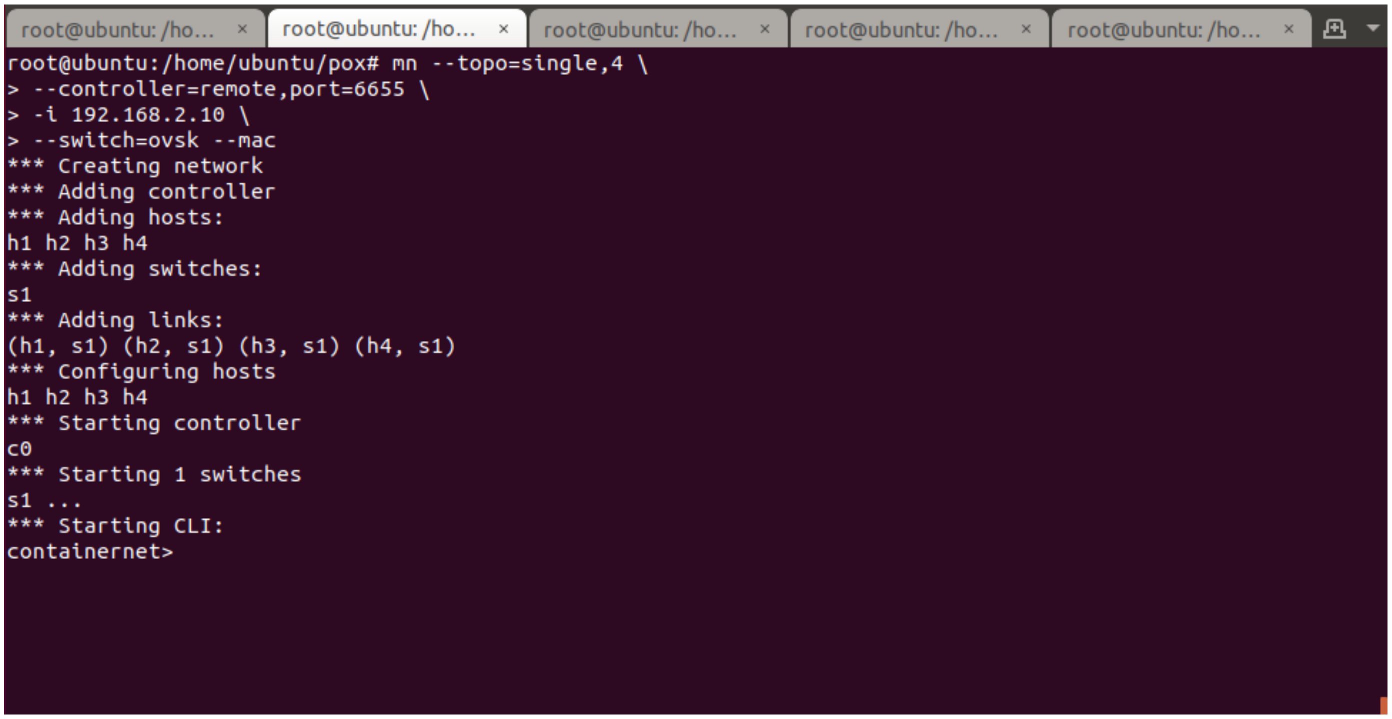
In this assignment, I have executed the various labs assignments, obtaining the proofs that they have been successfully completed.

.

1. Setting up mininet and Running mininet topology

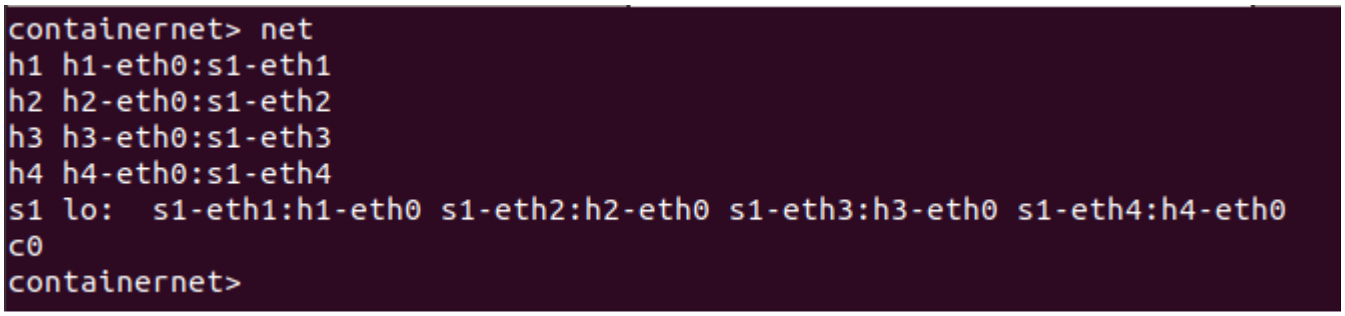
I have created a script to execute the mininet command line, since I need to restart it several times for troubleshooting purposes. The script "*runlab3.sh*" is produced in GitHub and reported here:

```
mn --topo=single,4 \
--controller=remote,port=6655 \
-i 192.168.2.10 \
--switch=ovsk --mac

mn -c
```



```
root@ubuntu:/home/ubuntu/pox# mn --topo=single,4 \
> --controller=remote,port=6655 \
> -i 192.168.2.10 \
> --switch=ovsk --mac
*** Creating network
*** Adding controller
*** 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:
containernet>
```

Figure 2 – Running mininet



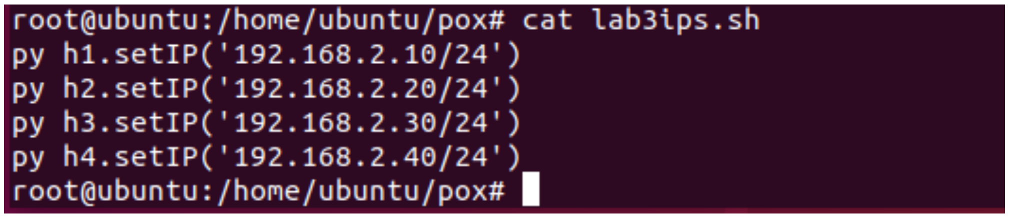
```
containernet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:h4-eth0
c0
containernet>
```

Figure 3 - Mininet network

2. Should assign IP addresses to hosts.

I have executed this again using a very simple source script called “lab3ips.sh” and executed from the mininet CLI with the command “*source lab3ips.sh*”. The script is referenced in GitHub and reproduced here:

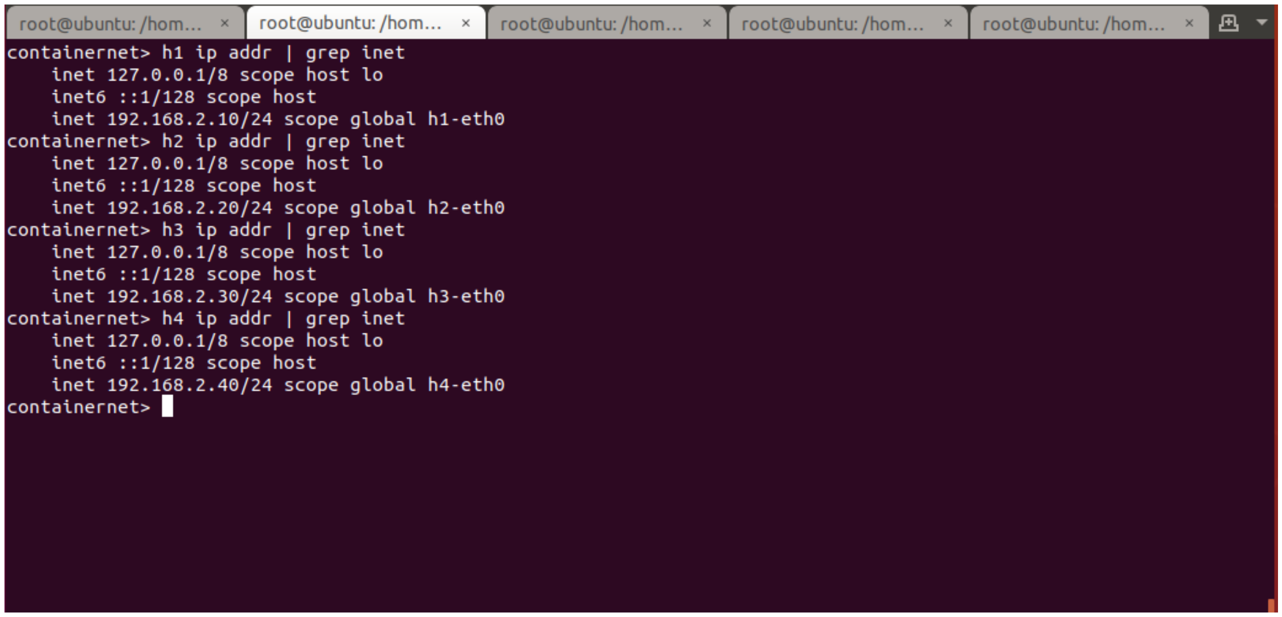
```
py h1.setIP('192.168.2.10/24')
py h2.setIP('192.168.2.20/24')
py h3.setIP('192.168.2.30/24')
py h4.setIP('192.168.2.40/24')
```



```
root@ubuntu:/home/ubuntu/pox# cat lab3ips.sh
py h1.setIP('192.168.2.10/24')
py h2.setIP('192.168.2.20/24')
py h3.setIP('192.168.2.30/24')
py h4.setIP('192.168.2.40/24')
root@ubuntu:/home/ubuntu/pox#
```

Figure 4 - Setting IP addresses for the lab

Screenshot showing the IP addresses changed:



```
containernet> h1 ip addr | grep inet
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 192.168.2.10/24 scope global h1-eth0
containernet> h2 ip addr | grep inet
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 192.168.2.20/24 scope global h2-eth0
containernet> h3 ip addr | grep inet
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 192.168.2.30/24 scope global h3-eth0
containernet> h4 ip addr | grep inet
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 192.168.2.40/24 scope global h4-eth0
containernet>
```

Figure 5 - Checking IP addresses for the containers in the lab

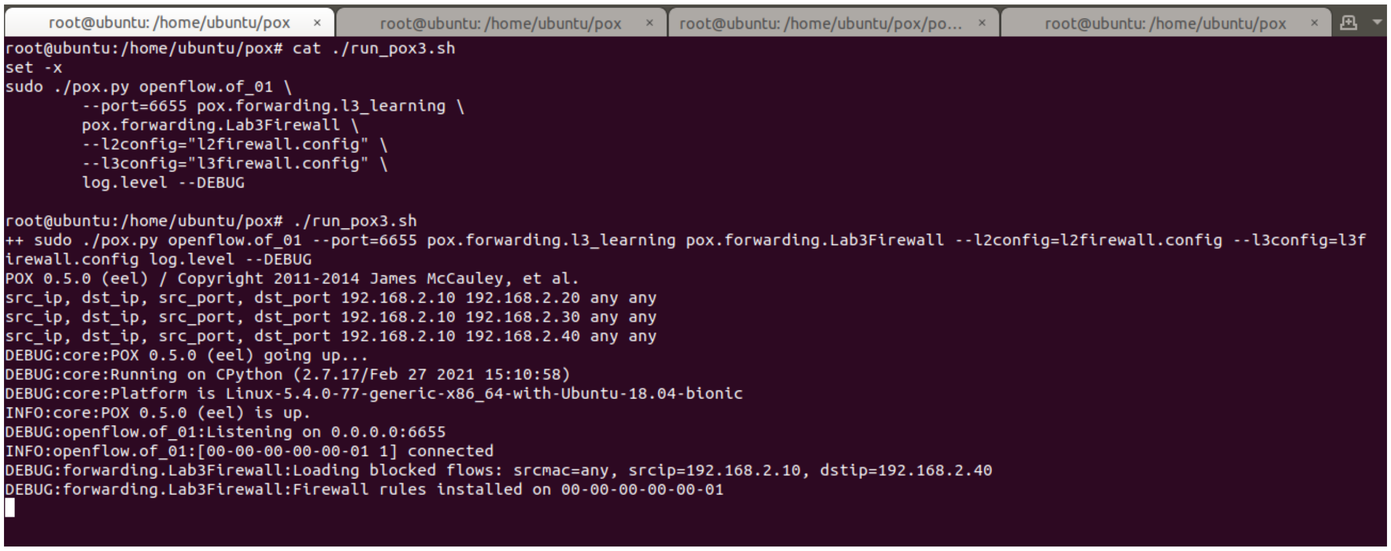
3. Perform Flood attack on SDN controller following a suggested procedure:

a. Run l3 learning application in POX controller.

I have again used a simple script to allow executing multiple times as needed in an easier manner. The script "*run_pox3.sh*" is on GitHub and reported below.

Please notice that I set the log level to "DEBUG" in order to work more effectively on the software by inserting debug comments in the code.

```
set -x
sudo ./pox.py openflow.of_01 \
   --port=6655 pox.forwarding.l3_learning \
   pox.forwarding.Lab3Firewall \
   --l2config="l2firewall.config" \
   --l3config="l3firewall.config" \
   log.level –DEBUG
```



```
root@ubuntu:/home/ubuntu/pox# cat ./run_pox3.sh
set -x
sudo ./pox.py openflow.of_01 \
        --port=6655 pox.forwarding.l3_learning \
        pox.forwarding.Lab3Firewall \
        --l2config="l2firewall.config" \
        --l3config="l3firewall.config" \
        log.level --DEBUG

root@ubuntu:/home/ubuntu/pox# ./run_pox3.sh
++ sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l3_learning pox.forwarding.Lab3Firewall --l2config=l2firewall.config --l3config=l3f
irewall.config log.level --DEBUG
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
src_ip, dst_ip, src_port, dst_port 192.168.2.10 192.168.2.20 any any
src_ip, dst_ip, src_port, dst_port 192.168.2.10 192.168.2.30 any any
src_ip, dst_ip, src_port, dst_port 192.168.2.10 192.168.2.40 any any
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.17/Feb 27 2021 15:10:58)
DEBUG:core:Platform is Linux-5.4.0-77-generic-x86_64-with-Ubuntu-18.04-bionic
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6655
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:forwarding.Lab3Firewall:Loading blocked flows: srcmac=any, srcip=192.168.2.10, dstip=192.168.2.40
DEBUG:forwarding.Lab3Firewall:Firewall rules installed on 00-00-00-00-00-01
```

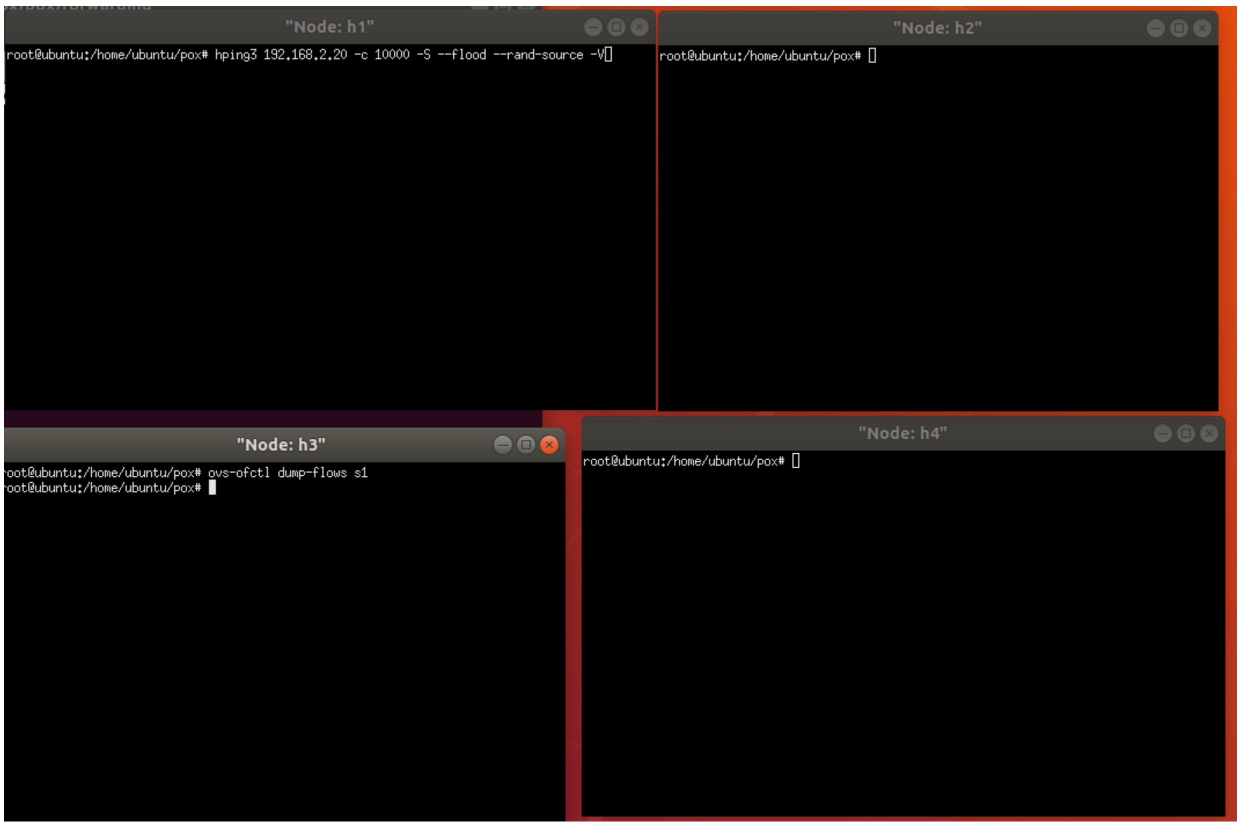
Figure 6 - Running POX

b. Check openflow flow-entries on switch 1.
c. Start flooding from any container host to container host #2
d. Check Openflow flow entries at switch 1

In the next sequence of screenshots, I will illustrate the Denial of Service happening on OVS because of the flood attack.

I have aligned the four X terminals of the four containers of the mininet, and used one to produce the flood, one to show the OVS flows, and another to try to ping another container.

In the first image, I am showing that the command "*ovs-ofctl dump-flows s1*" (used to dump the OVS flows) does not show anything – this is correct because there is no traffic yet.



```
root@ubuntu:/home/ubuntu/pox# hping3 192.168.2.20 -c 10000 -S --flood --rand-source -V
```

```
root@ubuntu:/home/ubuntu/pox#
```

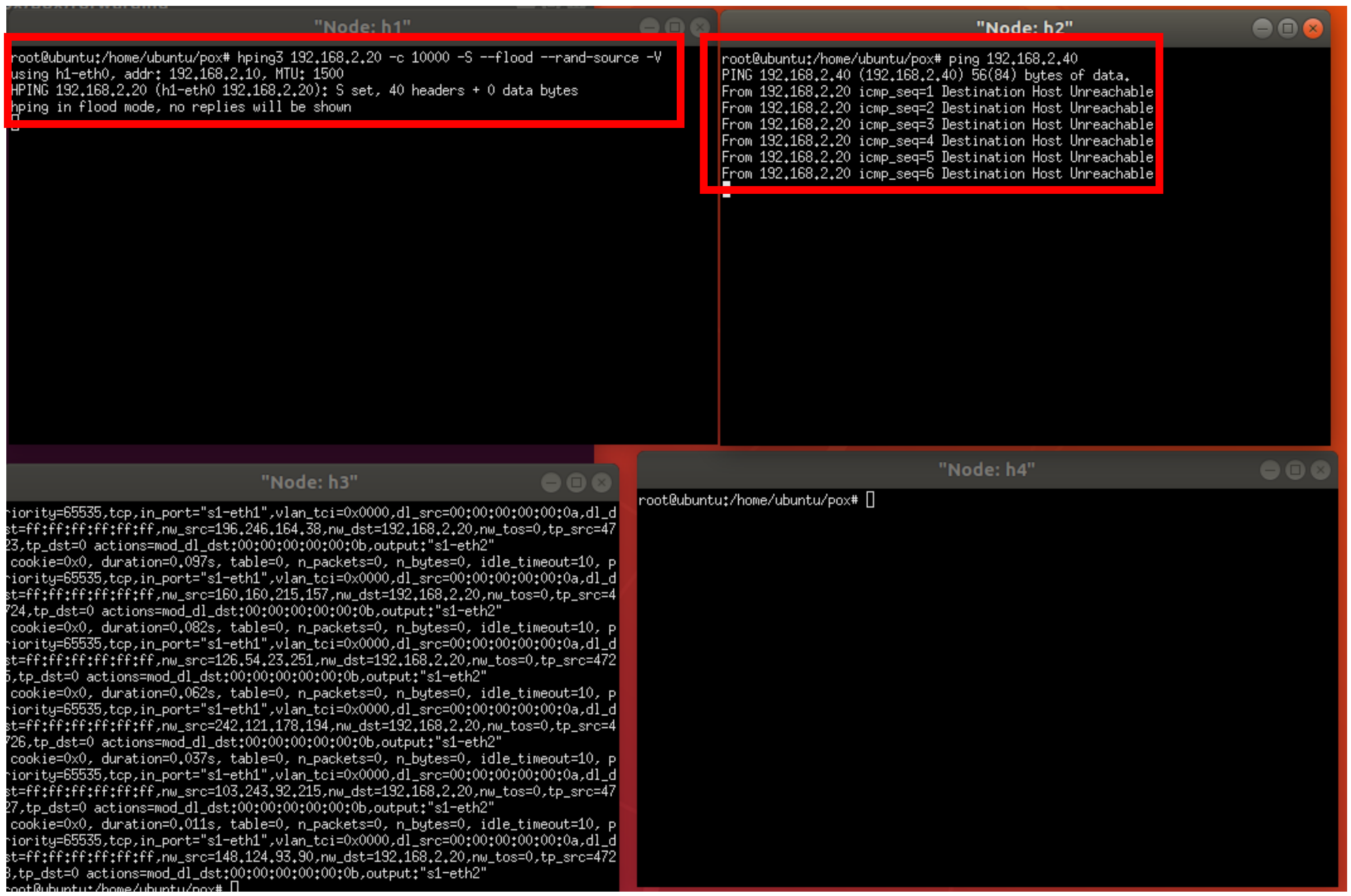
```
oot@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1
oot@ubuntu:/home/ubuntu/pox#
```

```
root@ubuntu:/home/ubuntu/pox#
```

Figure 7 - Check openflow flow-entries on switch 1

In the next screenshot, I have started the flood attack from the container h1 against h2. On container h2 I am trying to ping container 4, but with no success – the OVS controller is effectively under stress and does not forward the flows. Conversely, the containers are quite responsive – it's just their networking that is affected.

On the container 3 I am dumping the OVS flows – they are hundreds, and do not stop increase even after I have stopped the flood.



```
root@ubuntu:/home/ubuntu/pox# hping3 192.168.2.20 -c 10000 -S --flood --rand-source -V
using h1-eth0, addr: 192.168.2.10, MTU: 1500
HPING 192.168.2.20 (h1-eth0 192.168.2.20): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

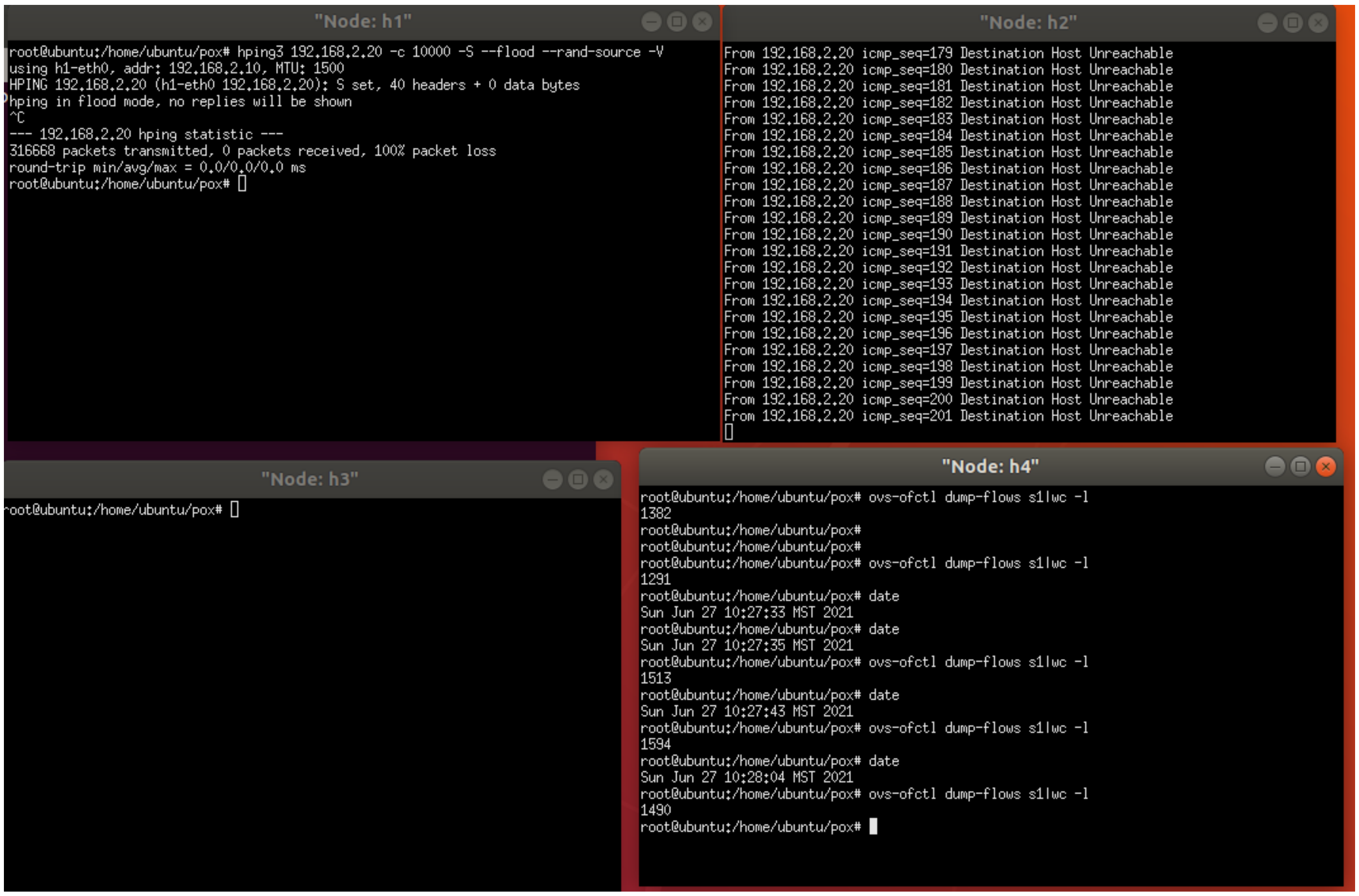
```
root@ubuntu:/home/ubuntu/pox# ping 192.168.2.40
PING 192.168.2.40 (192.168.2.40) 56(84) bytes of data.
From 192.168.2.20 icmp_seq=1 Destination Host Unreachable
From 192.168.2.20 icmp_seq=2 Destination Host Unreachable
From 192.168.2.20 icmp_seq=3 Destination Host Unreachable
From 192.168.2.20 icmp_seq=4 Destination Host Unreachable
From 192.168.2.20 icmp_seq=5 Destination Host Unreachable
From 192.168.2.20 icmp_seq=6 Destination Host Unreachable
```

```
iority=65535,tcp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=00:00:00:00:00:0a,dl_d
st=ff:ff:ff:ff:ff:ff,nw_src=196.246.164.38,nw_dst=192.168.2.20,nw_tos=0,tp_src=47
23,tp_dst=0 actions=mod_dl_dst:00:00:00:00:00:0b,output:"s1-eth2"
 cookie=0x0, duration=0.097s, table=0, n_packets=0, n_bytes=0, idle_timeout=10, p
iority=65535,tcp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=00:00:00:00:00:0a,dl_d
st=ff:ff:ff:ff:ff:ff,nw_src=160.160.215.157,nw_dst=192.168.2.20,nw_tos=0,tp_src=4
724,tp_dst=0 actions=mod_dl_dst:00:00:00:00:00:0b,output:"s1-eth2"
 cookie=0x0, duration=0.082s, table=0, n_packets=0, n_bytes=0, idle_timeout=10, p
iority=65535,tcp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=00:00:00:00:00:0a,dl_d
st=ff:ff:ff:ff:ff:ff,nw_src=126.54.23.251,nw_dst=192.168.2.20,nw_tos=0,tp_src=472
5,tp_dst=0 actions=mod_dl_dst:00:00:00:00:00:0b,output:"s1-eth2"
 cookie=0x0, duration=0.062s, table=0, n_packets=0, n_bytes=0, idle_timeout=10, p
iority=65535,tcp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=00:00:00:00:00:0a,dl_d
st=ff:ff:ff:ff:ff:ff,nw_src=242.121.178.194,nw_dst=192.168.2.20,nw_tos=0,tp_src=4
726,tp_dst=0 actions=mod_dl_dst:00:00:00:00:00:0b,output:"s1-eth2"
 cookie=0x0, duration=0.037s, table=0, n_packets=0, n_bytes=0, idle_timeout=10, p
iority=65535,tcp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=00:00:00:00:00:0a,dl_d
st=ff:ff:ff:ff:ff:ff,nw_src=103.243.92.215,nw_dst=192.168.2.20,nw_tos=0,tp_src=47
27,tp_dst=0 actions=mod_dl_dst:00:00:00:00:00:0b,output:"s1-eth2"
 cookie=0x0, duration=0.011s, table=0, n_packets=0, n_bytes=0, idle_timeout=10, p
iority=65535,tcp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=00:00:00:00:00:0a,dl_d
st=ff:ff:ff:ff:ff:ff,nw_src=148.124.93.90,nw_dst=192.168.2.20,nw_tos=0,tp_src=472
8,tp_dst=0 actions=mod_dl_dst:00:00:00:00:00:0b,output:"s1-eth2"
root@ubuntu:/home/ubuntu/pox# 
```

```
root@ubuntu:/home/ubuntu/pox# 
```

Figure 8 - Start flooding from any container host to container host #2

After the flood is concluded, the OVS comes back to normal slowly. From the next screenshot, it is possible to appreciate that it needs at least five minutes after the attack is terminated, to start resume the network.



```
root@ubuntu:/home/ubuntu/pox# hping3 192.168.2.20 -c 10000 -S --flood --rand-source -V
using h1-eth0, addr: 192.168.2.10, MTU: 1500
HPING 192.168.2.20 (h1-eth0 192.168.2.20): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.2.20 hping statistic ---
316668 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@ubuntu:/home/ubuntu/pox#
```

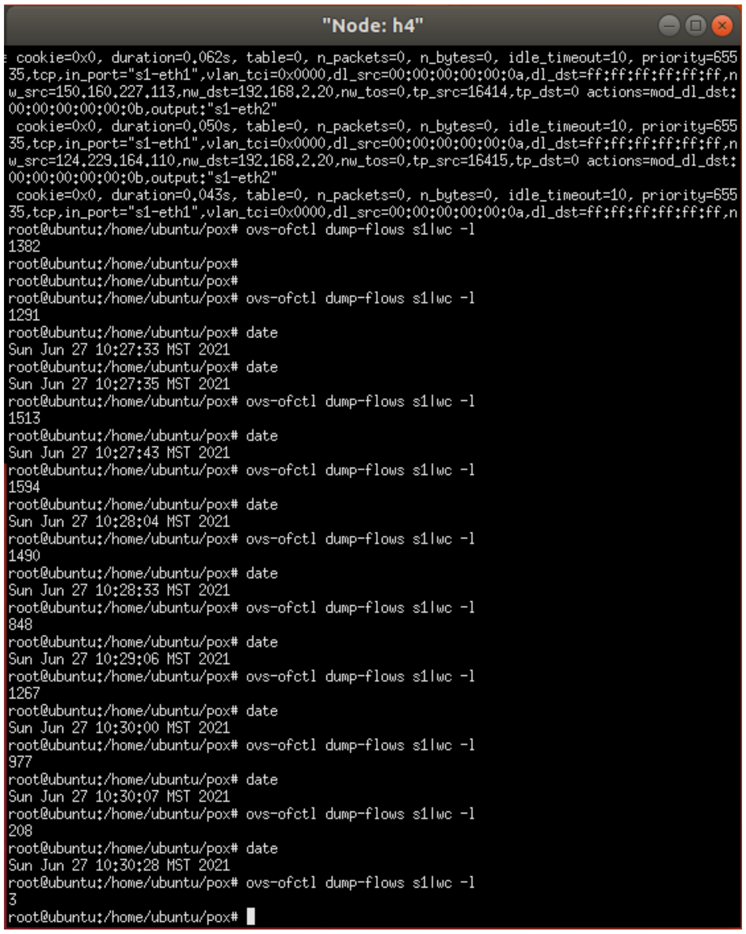
```
From 192.168.2.20 icmp_seq=179 Destination Host Unreachable
From 192.168.2.20 icmp_seq=180 Destination Host Unreachable
From 192.168.2.20 icmp_seq=181 Destination Host Unreachable
From 192.168.2.20 icmp_seq=182 Destination Host Unreachable
From 192.168.2.20 icmp_seq=183 Destination Host Unreachable
From 192.168.2.20 icmp_seq=184 Destination Host Unreachable
From 192.168.2.20 icmp_seq=185 Destination Host Unreachable
From 192.168.2.20 icmp_seq=186 Destination Host Unreachable
From 192.168.2.20 icmp_seq=187 Destination Host Unreachable
From 192.168.2.20 icmp_seq=188 Destination Host Unreachable
From 192.168.2.20 icmp_seq=189 Destination Host Unreachable
From 192.168.2.20 icmp_seq=190 Destination Host Unreachable
From 192.168.2.20 icmp_seq=191 Destination Host Unreachable
From 192.168.2.20 icmp_seq=192 Destination Host Unreachable
From 192.168.2.20 icmp_seq=193 Destination Host Unreachable
From 192.168.2.20 icmp_seq=194 Destination Host Unreachable
From 192.168.2.20 icmp_seq=195 Destination Host Unreachable
From 192.168.2.20 icmp_seq=196 Destination Host Unreachable
From 192.168.2.20 icmp_seq=197 Destination Host Unreachable
From 192.168.2.20 icmp_seq=198 Destination Host Unreachable
From 192.168.2.20 icmp_seq=199 Destination Host Unreachable
From 192.168.2.20 icmp_seq=200 Destination Host Unreachable
From 192.168.2.20 icmp_seq=201 Destination Host Unreachable
```

```
root@ubuntu:/home/ubuntu/pox#
```

```
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
1382
root@ubuntu:/home/ubuntu/pox#
root@ubuntu:/home/ubuntu/pox#
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
1291
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:27:33 MST 2021
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:27:35 MST 2021
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
1513
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:27:43 MST 2021
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
1594
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:28:04 MST 2021
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
1490
root@ubuntu:/home/ubuntu/pox#
```

Figure 9d. - OVS Slowly resuming operation

I have used "*wc -l*" to count the number of flows present on the switch, and they do not necessarily start diminishing after the flood is terminated.



```
"Node: h4"
: cookie=0x0, duration=0.062s, table=0, n_packets=0, n_bytes=0, idle_timeout=10, priority=655
35,tcp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=00:00:00:00:00:0a,dl_dst=ff:ff:ff:ff:ff:ff,n
w_src=150.160.227.113,nw_dst=192.168.2.20,nw_tos=0,tp_src=16414,tp_dst=0 actions=mod_dl_dst:
00:00:00:00:00:0b,output:"s1-eth2"
 cookie=0x0, duration=0.050s, table=0, n_packets=0, n_bytes=0, idle_timeout=10, priority=655
35,tcp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=00:00:00:00:00:0a,dl_dst=ff:ff:ff:ff:ff:ff,n
w_src=124.229.164.110,nw_dst=192.168.2.20,nw_tos=0,tp_src=16415,tp_dst=0 actions=mod_dl_dst:
00:00:00:00:00:0b,output:"s1-eth2"
 cookie=0x0, duration=0.043s, table=0, n_packets=0, n_bytes=0, idle_timeout=10, priority=655
35,tcp,in_port="s1-eth1",vlan_tci=0x0000,dl_src=00:00:00:00:00:0a,dl_dst=ff:ff:ff:ff:ff:ff,n
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
1382
root@ubuntu:/home/ubuntu/pox#
root@ubuntu:/home/ubuntu/pox#
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
1291
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:27:33 MST 2021
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:27:35 MST 2021
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
1513
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:27:43 MST 2021
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
1594
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:28:04 MST 2021
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
1490
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:28:33 MST 2021
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
848
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:29:06 MST 2021
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
1267
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:30:00 MST 2021
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
977
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:30:07 MST 2021
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
208
root@ubuntu:/home/ubuntu/pox# date
Sun Jun 27 10:30:28 MST 2021
root@ubuntu:/home/ubuntu/pox# ovs-ofctl dump-flows s1|wc -l
3
root@ubuntu:/home/ubuntu/pox# █
```

Figure 10 - Counting the flows on OVS

4. Mitigate DoS attack by implementing port security and using OpenFlow based firewall.

a. You should illustrate (through screenshots and descriptions) your implemented program codes.
b. You should demo how your implementation can mitigate the DoS through a sequence of screenshots with explanation.
c. You should submit the source codes of your implementation.

In this lab, I have modified the provided L3Firewall.py implementing "Lab3Firewall.py". The complete code is presented on GitHub; however, I will highlight here the most noticeable points.

The logic for spoofing and DoS detection is implemented through a Python Dict() object. Every time a new flow is observed by the OVS, the source MAC address of the processed packets is stored as the key; the source and destination IP addresses, as well as the OVS switchport, are stored as Dict values. For instance:

```
SpoofingTable = {
  # src MAC address     src IP              dst IP           src OVS port
  "00:00:00:00:00:0a": ["192.168.2.10", "192.168.2.30", "1"],
  "00:00:00:00:00:0b": ["192.168.2.20", "192.168.2.10", "2"],
  "00:00:00:00:00:0c": ["192.168.2.30", "192.168.2.40", "3"]
}
```

I have also implemented an algorithm that implements both the two spoofing attacks (spoofing of IP and spoofing of the MAC address), as required by the "Bonus Points" section.
I have extensively commented the source code in order to understand the program as it is being red.
The main principles for are the following:

- In the case of **IP address spoofing** ("base" case for the Lab):

  The packet from the attacker arrives with a certain MAC address and source IP, destined against the victim:

  Source MAC Source IP Victim's IP OVS switchport
  "00:00:00:00:00:0a": ["192.168.2.10", "192.168.2.30", "1"]

  The subsequent spoofed packets change their IP addresses, but keep the Source MAC and Victim's IP the same:

  Source MAC Source IP Victim's IP OVS switchport
  "00:00:00:00:00:0a": ["192.168.2.11", "192.168.2.30", "1"]
  "00:00:00:00:00:0a": ["192.168.2.12", "192.168.2.30", "1"]

  The pattern for this attack is the following:

Source MAC	Source IP	Victim's IP	OVS switchport
Constant	Changing	Constant	Constant

- In the case of **MAC address spoofing** ("Bonus" case for the Lab):

  The packet from the attacker arrives with a certain MAC address and source IP, destined against the victim:

  Source MAC Source IP Victim's IP OVS switchport
  "00:00:00:00:00:0a": ["192.168.2.10", "192.168.2.30", "1"]

  The subsequent spoofed packets change their IP addresses, but keep the Source MAC and Victim's IP the same:

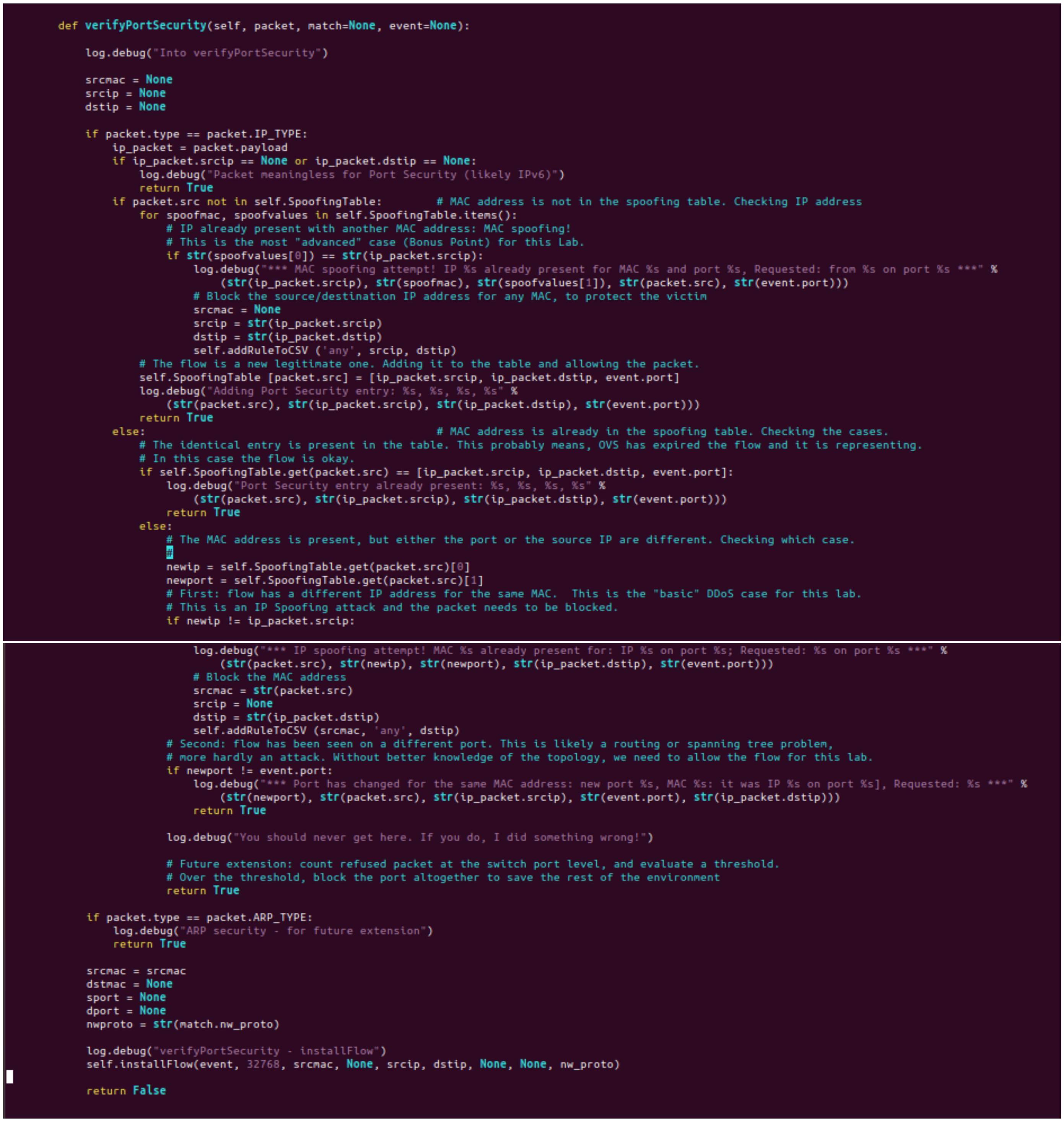
  Source MAC Source IP Victim's IP OVS switchport
  "00:00:00:00:01:0a": ["192.168.2.10", "192.168.2.30", "1"]
  "00:00:00:00:02:0a": ["192.168.2.10", "192.168.2.30", "1"]

  The pattern for this attack is the following:

Source MAC	Source IP	Victim's IP	OVS switchport
Changing	Constant	Constant	Constant

It is to be noticed, that the best way to block a Denial of Service attack is always by indicating a tuple of attack source + destination and not just blocking the attack source, as this can be a too wide rule which will end up also blocking legitimate users. In any case, in this Lab considerations over Distributed Denial of Services (and therefore, multiple attack sources against multiple targets) are not considered, and we are only resorting to relatively simple cases.
Please find below the pseudo-algorithm and the implementation for the function "verifyPortSecurity" which detects both floods:

```
if packet.source.MAC not in the SpoofingTable
  iterates through the SpoofingTable
    if packet.source.IP is present in the SpoofingTable = MAC spoofing detected
      block packet.source.IP + packet.destination.IP
  add packet.source.MAC, packet.source.IP, packet.destination.IP in the SpoofingTable
else
  if both packet.source.IP & packet.destination.IP are the same as saved in the SpoofingTable for packet.source.MAC
    the packet has been already observed, nothing to be done
  if packet.source.IP is different than source.IP in the SpoofingTable = IP spoofing detected
    block packet.source.MAC + packet.destination.IP
end-if
```



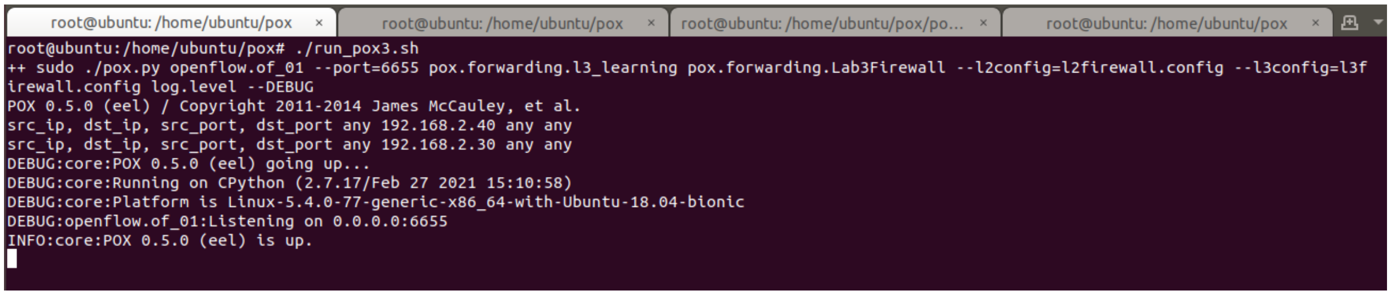
```
def verifyPortSecurity(self, packet, match=None, event=None):

    log.debug("Into verifyPortSecurity")

    srcmac = None
    srcip = None
    dstip = None

    if packet.type == packet.IP_TYPE:
        ip_packet = packet.payload
        if ip_packet.srcip == None or ip_packet.dstip == None:
            log.debug("Packet meaningless for Port Security (likely IPv6)")
            return True
        if packet.src not in self.SpoofingTable:        # MAC address is not in the spoofing table. Checking IP address
            for spoofmac, spoofvalues in self.SpoofingTable.items():
                # IP already present with another MAC address: MAC spoofing!
                # This is the most "advanced" case (Bonus Point) for this Lab.
                if str(spoofvalues[0]) == str(ip_packet.srcip):
                    log.debug("*** MAC spoofing attempt! IP %s already present for MAC %s and port %s, Requested: from %s on port %s ***" %
                        (str(ip_packet.srcip), str(spoofmac), str(spoofvalues[1]), str(packet.src), str(event.port)))
                    # Block the source/destination IP address for any MAC, to protect the victim
                    srcmac = None
                    srcip = str(ip_packet.srcip)
                    dstip = str(ip_packet.dstip)
                    self.addRuleToCSV ('any', srcip, dstip)
            # The flow is a new legitimate one. Adding it to the table and allowing the packet.
            self.SpoofingTable [packet.src] = [ip_packet.srcip, ip_packet.dstip, event.port]
            log.debug("Adding Port Security entry: %s, %s, %s, %s" %
                (str(packet.src), str(ip_packet.srcip), str(ip_packet.dstip), str(event.port)))
            return True
        else:                                           # MAC address is already in the spoofing table. Checking the cases.
            # The identical entry is present in the table. This probably means, OVS has expired the flow and it is representing.
            # In this case the flow is okay.
            if self.SpoofingTable.get(packet.src) == [ip_packet.srcip, ip_packet.dstip, event.port]:
                log.debug("Port Security entry already present: %s, %s, %s, %s" %
                    (str(packet.src), str(ip_packet.srcip), str(ip_packet.dstip), str(event.port)))
                return True
            else:
                # The MAC address is present, but either the port or the source IP are different. Checking which case.
                #
                newip = self.SpoofingTable.get(packet.src)[0]
                newport = self.SpoofingTable.get(packet.src)[1]
                # First: flow has a different IP address for the same MAC.  This is the "basic" DDoS case for this lab.
                # This is an IP Spoofing attack and the packet needs to be blocked.
                if newip != ip_packet.srcip:
                    log.debug("*** IP spoofing attempt! MAC %s already present for: IP %s on port %s; Requested: %s on port %s ***" %
                        (str(packet.src), str(newip), str(newport), str(ip_packet.dstip), str(event.port)))
                    # Block the MAC address
                    srcmac = str(packet.src)
                    srcip = None
                    dstip = str(ip_packet.dstip)
                    self.addRuleToCSV (srcmac, 'any', dstip)
                # Second: flow has been seen on a different port. This is likely a routing or spanning tree problem,
                # more hardly an attack. Without better knowledge of the topology, we need to allow the flow for this lab.
                if newport != event.port:
                    log.debug("*** Port has changed for the same MAC address: new port %s, MAC %s: it was IP %s on port %s], Requested: %s ***" %
                        (str(newport), str(packet.src), str(ip_packet.srcip), str(event.port), str(ip_packet.dstip)))
                    return True

                log.debug("You should never get here. If you do, I did something wrong!")

                # Future extension: count refused packet at the switch port level, and evaluate a threshold.
                # Over the threshold, block the port altogether to save the rest of the environment
                return True

    if packet.type == packet.ARP_TYPE:
        log.debug("ARP security - for future extension")
        return True

    srcmac = srcmac
    dstmac = None
    sport = None
    dport = None
    nwproto = str(match.nw_proto)

    log.debug("verifyPortSecurity - installFlow")
    self.installFlow(event, 32768, srcmac, None, srcip, dstip, None, None, nw_proto)

    return False
```

Figure 11 - Function "verifyPortSecurity"

Please find below the screenshots that demonstrate the mitigation of the two attacks.

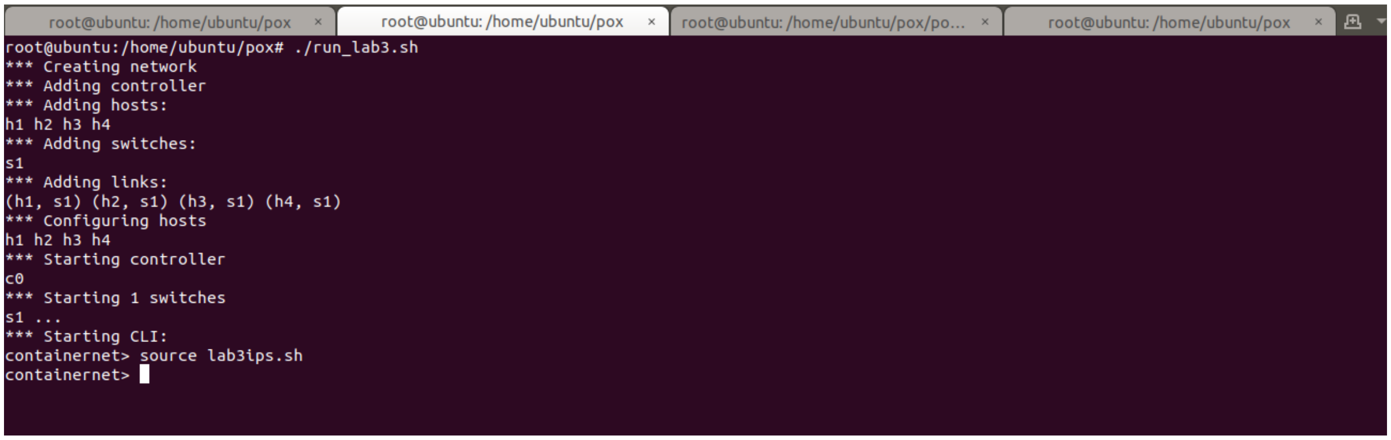
d. IP Spoofing Attack Mitigation

I have done my best to illustrate the software working properly, I hope this comes through as clear enough.



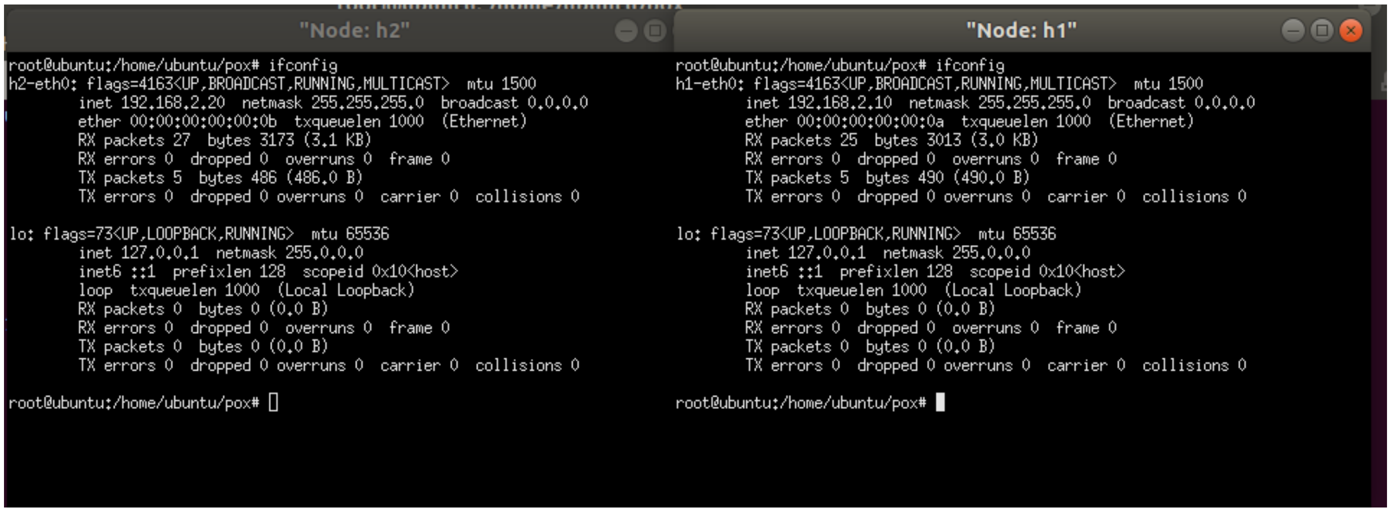
```
root@ubuntu:/home/ubuntu/pox# ./run_pox3.sh
++ sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l3_learning pox.forwarding.Lab3Firewall --l2config=l2firewall.config --l3config=l3f
irewall.config log.level --DEBUG
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
src_ip, dst_ip, src_port, dst_port any 192.168.2.40 any any
src_ip, dst_ip, src_port, dst_port any 192.168.2.30 any any
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.17/Feb 27 2021 15:10:58)
DEBUG:core:Platform is Linux-5.4.0-77-generic-x86_64-with-Ubuntu-18.04-bionic
DEBUG:openflow.of_01:Listening on 0.0.0.0:6655
INFO:core:POX 0.5.0 (eel) is up.
```

Figure 12 - IP Spoofing - Running POX



```
root@ubuntu:/home/ubuntu/pox# ./run_lab3.sh
*** Creating network
*** Adding controller
*** 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:
containernet> source lab3ips.sh
containernet>
```

Figure 13 - IP Spoofing - Running mininet



"Node: h2"

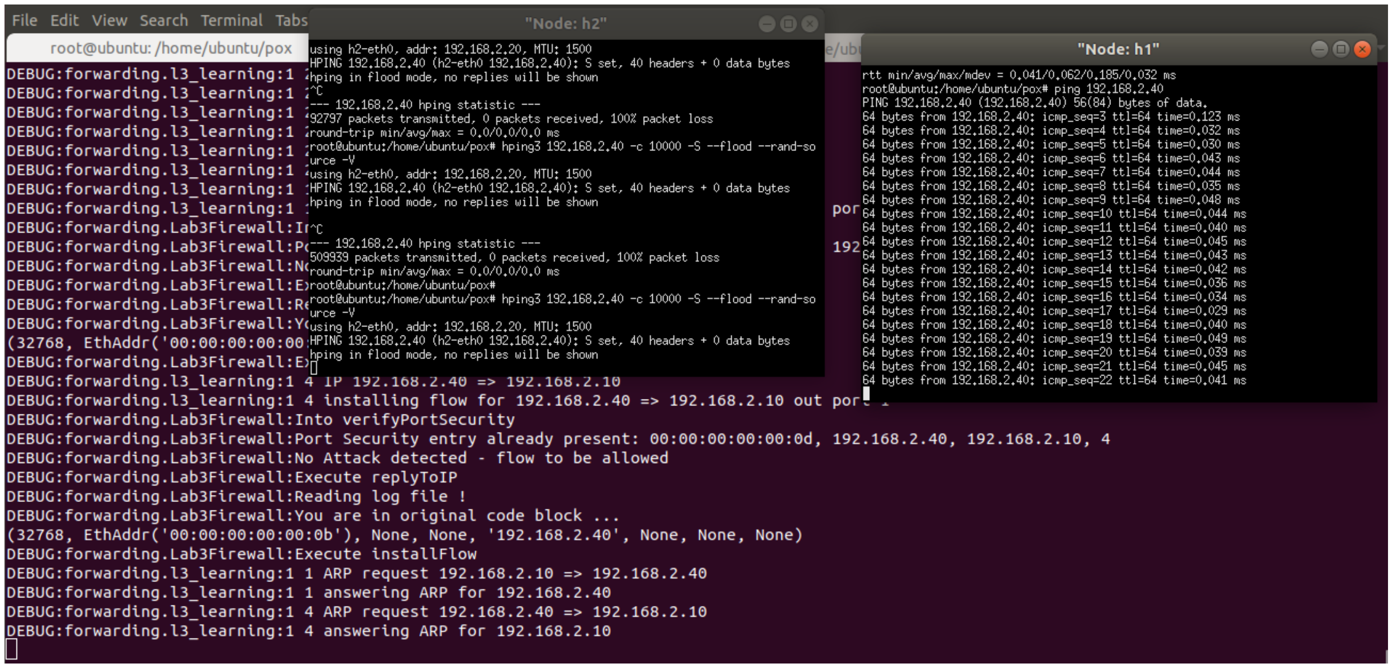
```
root@ubuntu:/home/ubuntu/pox# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.20  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 00:00:00:00:00:0b  txqueuelen 1000  (Ethernet)
        RX packets 27  bytes 3173 (3.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5  bytes 486 (486.0 B)
        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>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu:/home/ubuntu/pox#
```

"Node: h1"

```
root@ubuntu:/home/ubuntu/pox# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.10  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 00:00:00:00:00:0a  txqueuelen 1000  (Ethernet)
        RX packets 25  bytes 3013 (3.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5  bytes 490 (490.0 B)
        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>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu:/home/ubuntu/pox#
```

Figure 14 - IP Spoofing - h1 and h2 with their starting IPs and MAC addresses

As visible from the next two screenshots, hping3 is running a flood on one container, while the other is able to ping without issues, meaning that the OVS is not flooded.



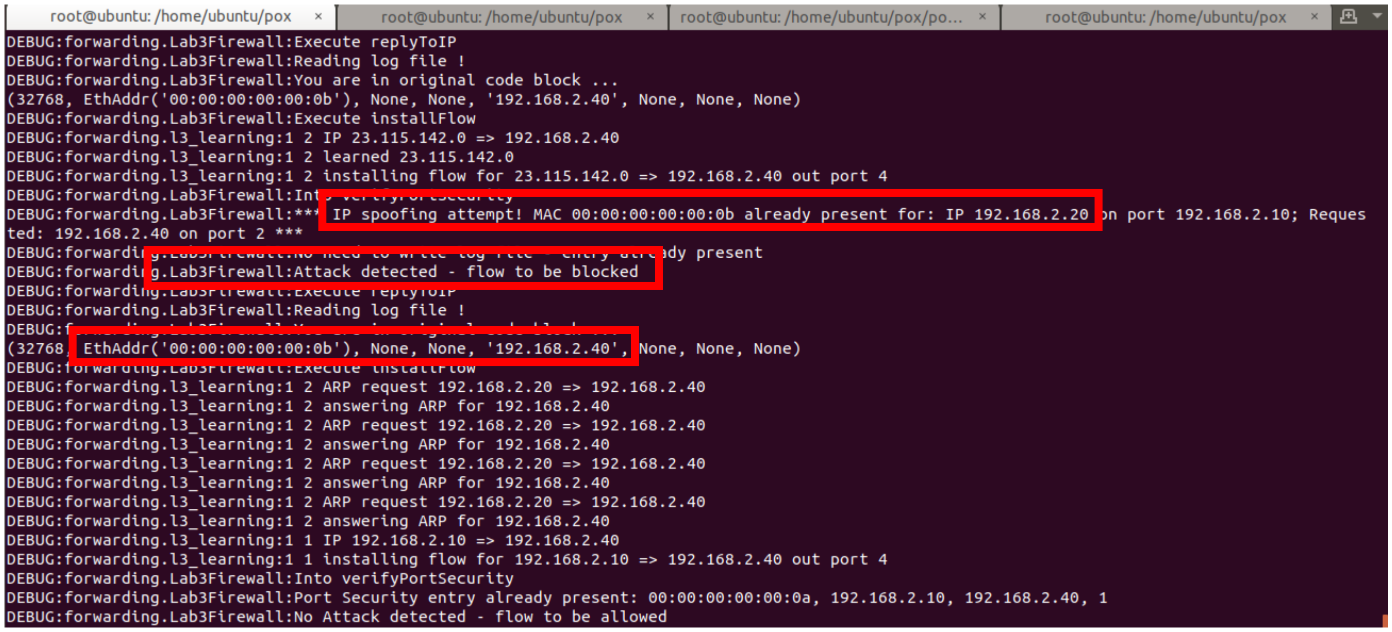
```
using h2-eth0, addr: 192.168.2.20, MTU: 1500
HPING 192.168.2.40 (h2-eth0 192.168.2.40): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.2.40 hping statistic ---
92797 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@ubuntu:/home/ubuntu/pox# hping3 192.168.2.40 -c 10000 -S --flood --rand-so
urce -V
using h2-eth0, addr: 192.168.2.20, MTU: 1500
HPING 192.168.2.40 (h2-eth0 192.168.2.40): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown

^C
--- 192.168.2.40 hping statistic ---
509939 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@ubuntu:/home/ubuntu/pox#
root@ubuntu:/home/ubuntu/pox# hping3 192.168.2.40 -c 10000 -S --flood --rand-so
urce -V
using h2-eth0, addr: 192.168.2.20, MTU: 1500
HPING 192.168.2.40 (h2-eth0 192.168.2.40): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

```
rtt min/avg/max/mdev = 0.041/0.062/0.185/0.032 ms
root@ubuntu:/home/ubuntu/pox# ping 192.168.2.40
PING 192.168.2.40 (192.168.2.40) 56(84) bytes of data.
64 bytes from 192.168.2.40: icmp_seq=3 ttl=64 time=0.123 ms
64 bytes from 192.168.2.40: icmp_seq=4 ttl=64 time=0.032 ms
64 bytes from 192.168.2.40: icmp_seq=5 ttl=64 time=0.030 ms
64 bytes from 192.168.2.40: icmp_seq=6 ttl=64 time=0.043 ms
64 bytes from 192.168.2.40: icmp_seq=7 ttl=64 time=0.044 ms
64 bytes from 192.168.2.40: icmp_seq=8 ttl=64 time=0.035 ms
64 bytes from 192.168.2.40: icmp_seq=9 ttl=64 time=0.048 ms
64 bytes from 192.168.2.40: icmp_seq=10 ttl=64 time=0.044 ms
64 bytes from 192.168.2.40: icmp_seq=11 ttl=64 time=0.040 ms
64 bytes from 192.168.2.40: icmp_seq=12 ttl=64 time=0.045 ms
64 bytes from 192.168.2.40: icmp_seq=13 ttl=64 time=0.043 ms
64 bytes from 192.168.2.40: icmp_seq=14 ttl=64 time=0.042 ms
64 bytes from 192.168.2.40: icmp_seq=15 ttl=64 time=0.036 ms
64 bytes from 192.168.2.40: icmp_seq=16 ttl=64 time=0.034 ms
64 bytes from 192.168.2.40: icmp_seq=17 ttl=64 time=0.029 ms
64 bytes from 192.168.2.40: icmp_seq=18 ttl=64 time=0.040 ms
64 bytes from 192.168.2.40: icmp_seq=19 ttl=64 time=0.049 ms
64 bytes from 192.168.2.40: icmp_seq=20 ttl=64 time=0.039 ms
64 bytes from 192.168.2.40: icmp_seq=21 ttl=64 time=0.045 ms
64 bytes from 192.168.2.40: icmp_seq=22 ttl=64 time=0.041 ms
```

```
DEBUG:forwarding.l3_learning:1 4 IP 192.168.2.40 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 4 installing flow for 192.168.2.40 => 192.168.2.10 out port 1
DEBUG:forwarding.Lab3Firewall:Into verifyPortSecurity
DEBUG:forwarding.Lab3Firewall:Port Security entry already present: 00:00:00:00:00:0d, 192.168.2.40, 192.168.2.10, 4
DEBUG:forwarding.Lab3Firewall:No Attack detected - flow to be allowed
DEBUG:forwarding.Lab3Firewall:Execute replyToIP
DEBUG:forwarding.Lab3Firewall:Reading log file !
DEBUG:forwarding.Lab3Firewall:You are in original code block ...
(32768, EthAddr('00:00:00:00:00:0b'), None, None, '192.168.2.40', None, None, None)
DEBUG:forwarding.Lab3Firewall:Execute installFlow
DEBUG:forwarding.l3_learning:1 1 ARP request 192.168.2.10 => 192.168.2.40
DEBUG:forwarding.l3_learning:1 1 answering ARP for 192.168.2.40
DEBUG:forwarding.l3_learning:1 4 ARP request 192.168.2.40 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 4 answering ARP for 192.168.2.10
```

Figure 15 - IP Spoofing - Running hping3 on h2 and running ping on h1

I have minimized the two xterm in order to see the debug messages more clearly. It is possible to appreciate the fact that the attack is noticed at the second packet and the flow is blocked. The algorithm obviously requires two packets with the same MAC and two different IPs as a minimum, which will block the source MAC + victim's IP. After blocking, the next attack packets do not require further action, as they are automatically dropped by the OVS.



```
DEBUG:forwarding.Lab3Firewall:Execute replyToIP
DEBUG:forwarding.Lab3Firewall:Reading log file !
DEBUG:forwarding.Lab3Firewall:You are in original code block ...
(32768, EthAddr('00:00:00:00:00:0b'), None, None, '192.168.2.40', None, None, None)
DEBUG:forwarding.Lab3Firewall:Execute installFlow
DEBUG:forwarding.l3_learning:1 2 IP 23.115.142.0 => 192.168.2.40
DEBUG:forwarding.l3_learning:1 2 learned 23.115.142.0
DEBUG:forwarding.l3_learning:1 2 installing flow for 23.115.142.0 => 192.168.2.40 out port 4
DEBUG:forwarding.Lab3Firewall:Into verifyPortSecurity
DEBUG:forwarding.Lab3Firewall:*** IP spoofing attempt! MAC 00:00:00:00:00:0b already present for: IP 192.168.2.20 on port 192.168.2.10; Reques
ted: 192.168.2.40 on port 2 ***
DEBUG:forwarding.Lab3Firewall:No need to write log file - entry already present
DEBUG:forwarding.Lab3Firewall:Attack detected - flow to be blocked
DEBUG:forwarding.Lab3Firewall:Execute replyToIP
DEBUG:forwarding.Lab3Firewall:Reading log file !
DEBUG:forwarding.Lab3Firewall:You are in original code block ...
(32768, EthAddr('00:00:00:00:00:0b'), None, None, '192.168.2.40', None, None, None)
DEBUG:forwarding.Lab3Firewall:Execute installFlow
DEBUG:forwarding.l3_learning:1 2 ARP request 192.168.2.20 => 192.168.2.40
DEBUG:forwarding.l3_learning:1 2 answering ARP for 192.168.2.40
DEBUG:forwarding.l3_learning:1 2 ARP request 192.168.2.20 => 192.168.2.40
DEBUG:forwarding.l3_learning:1 2 answering ARP for 192.168.2.40
DEBUG:forwarding.l3_learning:1 2 ARP request 192.168.2.20 => 192.168.2.40
DEBUG:forwarding.l3_learning:1 2 answering ARP for 192.168.2.40
DEBUG:forwarding.l3_learning:1 2 ARP request 192.168.2.20 => 192.168.2.40
DEBUG:forwarding.l3_learning:1 2 answering ARP for 192.168.2.40
DEBUG:forwarding.l3_learning:1 1 IP 192.168.2.10 => 192.168.2.40
DEBUG:forwarding.l3_learning:1 1 installing flow for 192.168.2.10 => 192.168.2.40 out port 4
DEBUG:forwarding.Lab3Firewall:Into verifyPortSecurity
DEBUG:forwarding.Lab3Firewall:Port Security entry already present: 00:00:00:00:00:0a, 192.168.2.10, 192.168.2.40, 1
DEBUG:forwarding.Lab3Firewall:No Attack detected - flow to be allowed
```

Figure 16 - IP Spoofing - debug messages showing malicious packets being blocked

Debug messages:

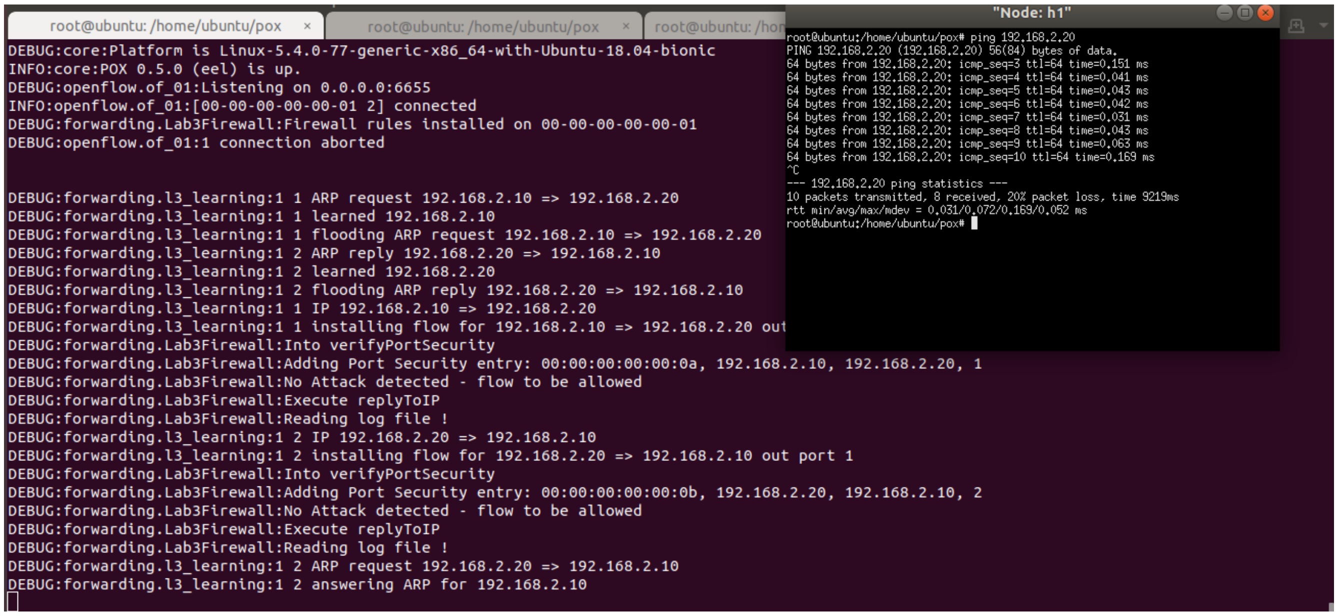
```
DEBUG:forwarding.Lab3Firewall:Execute replyToIP
DEBUG:forwarding.Lab3Firewall:Reading log file !
```

```
DEBUG:forwarding.Lab3Firewall:You are in original code block ...
(32768, EthAddr('00:00:00:00:00:0b'), None, None, '192.168.2.40', None, None, None)
DEBUG:forwarding.Lab3Firewall:Execute installFlow
DEBUG:forwarding.l3_learning:1 2 IP 23.115.142.0 => 192.168.2.40
DEBUG:forwarding.l3_learning:1 2 learned 23.115.142.0
DEBUG:forwarding.l3_learning:1 2 installing flow for 23.115.142.0 => 192.168.2.40 out port 4
DEBUG:forwarding.Lab3Firewall:Into verifyPortSecurity
DEBUG:forwarding.Lab3Firewall:*** IP spoofing attempt! MAC 00:00:00:00:00:0b already present
for: IP 192.168.2.20 on port 192.168.2.10; Requested: 192.168.2.40 on port 2 ***
DEBUG:forwarding.Lab3Firewall:No need to write log file - entry already present
DEBUG:forwarding.Lab3Firewall:Attack detected - flow to be blocked
DEBUG:forwarding.Lab3Firewall:Execute replyToIP
DEBUG:forwarding.Lab3Firewall:Reading log file !
DEBUG:forwarding.Lab3Firewall:You are in original code block ...
```

a. MAC Spoofing Attack Mitigation

I have performed two kind of attacks: one to simply show that the mitigation is conceptually working, by simply manually changing the MAC address of the attacker container; the second by running a tool called *nping*.

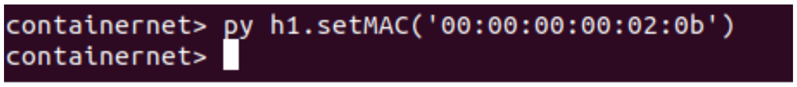
At the beginning, the environment is readied, and pings are executed to be sure that networking is functioning, and the Lab firewall is processing the packets – and therefore registering the flow in the SpoofingTable.



```
DEBUG:core:Platform is Linux-5.4.0-77-generic-x86_64-with-Ubuntu-18.04-bionic
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6655
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:forwarding.Lab3Firewall:Firewall rules installed on 00-00-00-00-00-01
DEBUG:openflow.of_01:1 connection aborted


DEBUG:forwarding.l3_learning:1 1 ARP request 192.168.2.10 => 192.168.2.20
DEBUG:forwarding.l3_learning:1 1 learned 192.168.2.10
DEBUG:forwarding.l3_learning:1 1 flooding ARP request 192.168.2.10 => 192.168.2.20
DEBUG:forwarding.l3_learning:1 2 ARP reply 192.168.2.20 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 2 learned 192.168.2.20
DEBUG:forwarding.l3_learning:1 2 flooding ARP reply 192.168.2.20 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 1 IP 192.168.2.10 => 192.168.2.20
DEBUG:forwarding.l3_learning:1 1 installing flow for 192.168.2.10 => 192.168.2.20 out
DEBUG:forwarding.Lab3Firewall:Into verifyPortSecurity
DEBUG:forwarding.Lab3Firewall:Adding Port Security entry: 00:00:00:00:00:0a, 192.168.2.10, 192.168.2.20, 1
DEBUG:forwarding.Lab3Firewall:No Attack detected - flow to be allowed
DEBUG:forwarding.Lab3Firewall:Execute replyToIP
DEBUG:forwarding.Lab3Firewall:Reading log file !
DEBUG:forwarding.l3_learning:1 2 IP 192.168.2.20 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 2 installing flow for 192.168.2.20 => 192.168.2.10 out port 1
DEBUG:forwarding.Lab3Firewall:Into verifyPortSecurity
DEBUG:forwarding.Lab3Firewall:Adding Port Security entry: 00:00:00:00:00:0b, 192.168.2.20, 192.168.2.10, 2
DEBUG:forwarding.Lab3Firewall:No Attack detected - flow to be allowed
DEBUG:forwarding.Lab3Firewall:Execute replyToIP
DEBUG:forwarding.Lab3Firewall:Reading log file !
DEBUG:forwarding.l3_learning:1 2 ARP request 192.168.2.20 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 2 answering ARP for 192.168.2.10
```

```
root@ubuntu:/home/ubuntu/pox# ping 192.168.2.20
PING 192.168.2.20 (192.168.2.20) 56(84) bytes of data.
64 bytes from 192.168.2.20: icmp_seq=3 ttl=64 time=0.151 ms
64 bytes from 192.168.2.20: icmp_seq=4 ttl=64 time=0.041 ms
64 bytes from 192.168.2.20: icmp_seq=5 ttl=64 time=0.043 ms
64 bytes from 192.168.2.20: icmp_seq=6 ttl=64 time=0.042 ms
64 bytes from 192.168.2.20: icmp_seq=7 ttl=64 time=0.031 ms
64 bytes from 192.168.2.20: icmp_seq=8 ttl=64 time=0.043 ms
64 bytes from 192.168.2.20: icmp_seq=9 ttl=64 time=0.063 ms
64 bytes from 192.168.2.20: icmp_seq=10 ttl=64 time=0.169 ms
^C
--- 192.168.2.20 ping statistics ---
10 packets transmitted, 8 received, 20% packet loss, time 9219ms
rtt min/avg/max/mdev = 0.031/0.072/0.169/0.052 ms
root@ubuntu:/home/ubuntu/pox#
```

Figure 17 - MAC spoofing - initial setup

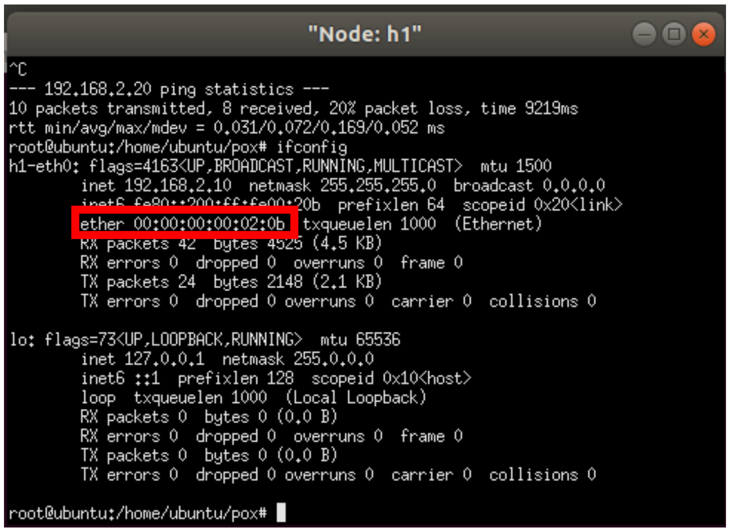


```
containernet> py h1.setMAC('00:00:00:00:02:0b')
containernet>
```

Figure 18 - MAC spoofing - changing MAC manually

From here, I manually change the MAC address of the contained with a Python-one-liner using the mininet API. The command is "*py h1.setMAC('00:00:00:00:02:0b')*"

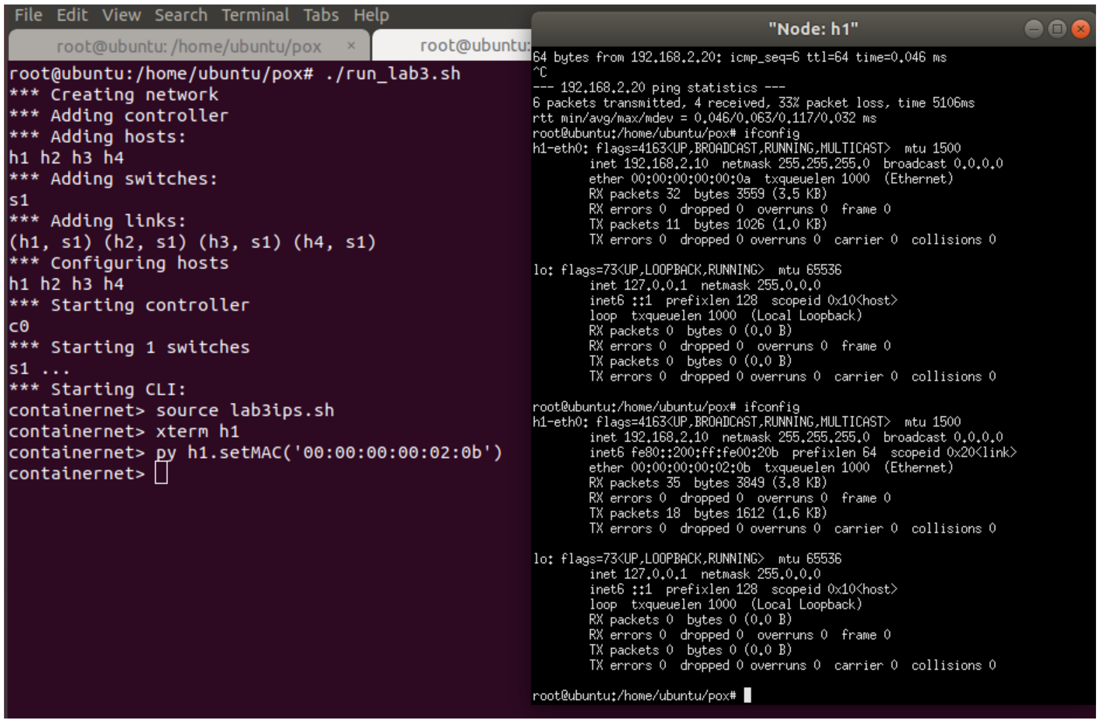
We can see from the xterm's screenshot that the container has changed its MAC address to the value we have chosen.



```
^C
--- 192.168.2.20 ping statistics ---
10 packets transmitted, 8 received, 20% packet loss, time 9219ms
rtt min/avg/max/mdev = 0.031/0.072/0.169/0.052 ms
root@ubuntu:/home/ubuntu/pox# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.10  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::200:ff:fe00:20b  prefixlen 64  scopeid 0x20<link>
        ether 00:00:00:00:02:0b  txqueuelen 1000  (Ethernet)
        RX packets 42  bytes 4525 (4.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 24  bytes 2148 (2.1 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>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu:/home/ubuntu/pox#
```

Figure 19 - MAC spoofing - confirmation of MAC change

From the next screenshot the whole picture is presented.



File Edit View Search Terminal Tabs Help
root@ubuntu: /home/ubuntu/pox
root@ubuntu:
"Node: h1"

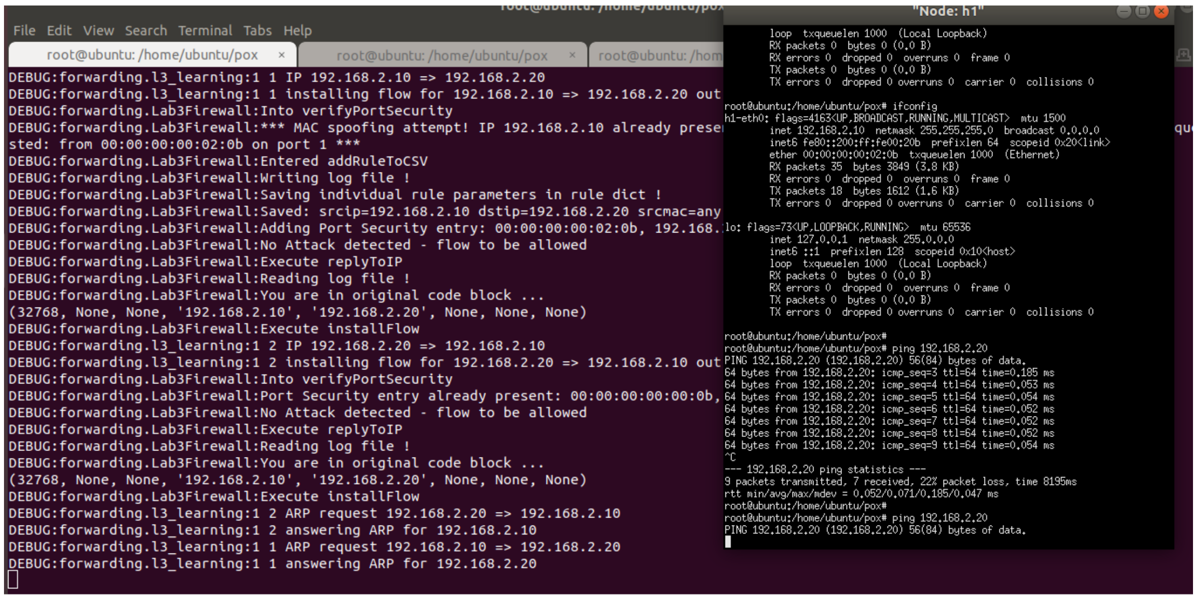
```
root@ubuntu:/home/ubuntu/pox# ./run_lab3.sh
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
containernet> source lab3ips.sh
containernet> xterm h1
containernet> py h1.setMAC('00:00:00:00:02:0b')
containernet>
```

```
64 bytes from 192.168.2.20: icmp_seq=6 ttl=64 time=0.046 ms
^C
--- 192.168.2.20 ping statistics ---
6 packets transmitted, 4 received, 33% packet loss, time 5106ms
rtt min/avg/max/mdev = 0.046/0.063/0.117/0.032 ms
root@ubuntu:/home/ubuntu/pox# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.10  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 00:00:00:00:00:0a  txqueuelen 1000  (Ethernet)
        RX packets 32  bytes 3559 (3.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 1026 (1.0 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>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu:/home/ubuntu/pox# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.10  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::200:ff:fe00:20b  prefixlen 64  scopeid 0x20<link>
        ether 00:00:00:00:02:0b  txqueuelen 1000  (Ethernet)
        RX packets 35  bytes 3849 (3.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 1612 (1.6 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>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu:/home/ubuntu/pox#
```

Figure 20 - MAC spoofing - Before and after MAC change

After the MAC address has changed, the networking takes a bit of time to "reconverge" – in my system, it has taken 9 echo-request cycles, but it will vary depending on the case. Basically, the switch needs time to re-adjust to the fact that the topology has changed, and only after this has been reflected in the OVS, we can effectively catch the offending packets. This is because our software is executed after the basic switch functionalities – included in *l3_learning.py* – are executed.

The only way to improve and have a faster detection for MAC address spoofing, would be to work directly on the l3_learning.py or *l2_learning.py* Python programs in order to access the packets "on the (virtual) wire".



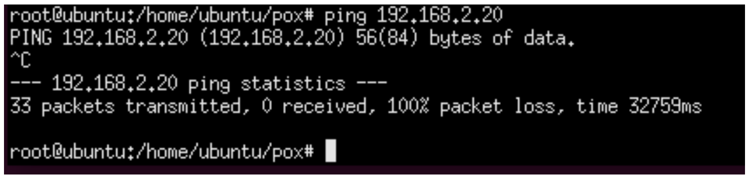
File Edit View Search Terminal Tabs Help
root@ubuntu: /home/ubuntu/pox
root@ubuntu: /home/ubuntu/pox
root@ubuntu: /hom
"Node: h1"

```
DEBUG:forwarding.l3_learning:1 1 IP 192.168.2.10 => 192.168.2.20
DEBUG:forwarding.l3_learning:1 1 installing flow for 192.168.2.10 => 192.168.2.20 out
DEBUG:forwarding.Lab3Firewall:Into verifyPortSecurity
DEBUG:forwarding.Lab3Firewall:*** MAC spoofing attempt! IP 192.168.2.10 already prese
sted: from 00:00:00:00:02:0b on port 1 ***
DEBUG:forwarding.Lab3Firewall:Entered addRuleToCSV
DEBUG:forwarding.Lab3Firewall:Writing log file !
DEBUG:forwarding.Lab3Firewall:Saving individual rule parameters in rule dict !
DEBUG:forwarding.Lab3Firewall:Saved: srcip=192.168.2.10 dstip=192.168.2.20 srcmac=any
DEBUG:forwarding.Lab3Firewall:Adding Port Security entry: 00:00:00:00:02:0b, 192.168.
DEBUG:forwarding.Lab3Firewall:No Attack detected - flow to be allowed
DEBUG:forwarding.Lab3Firewall:Execute replyToIP
DEBUG:forwarding.Lab3Firewall:Reading log file !
DEBUG:forwarding.Lab3Firewall:You are in original code block ...
(32768, None, None, '192.168.2.10', '192.168.2.20', None, None, None)
DEBUG:forwarding.Lab3Firewall:Execute installFlow
DEBUG:forwarding.l3_learning:1 2 IP 192.168.2.20 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 2 installing flow for 192.168.2.20 => 192.168.2.10 out
DEBUG:forwarding.Lab3Firewall:Into verifyPortSecurity
DEBUG:forwarding.Lab3Firewall:Port Security entry already present: 00:00:00:00:00:0b,
DEBUG:forwarding.Lab3Firewall:No Attack detected - flow to be allowed
DEBUG:forwarding.Lab3Firewall:Execute replyToIP
DEBUG:forwarding.Lab3Firewall:Reading log file !
DEBUG:forwarding.Lab3Firewall:You are in original code block ...
(32768, None, None, '192.168.2.10', '192.168.2.20', None, None, None)
DEBUG:forwarding.Lab3Firewall:Execute installFlow
DEBUG:forwarding.l3_learning:1 2 ARP request 192.168.2.20 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 2 answering ARP for 192.168.2.10
DEBUG:forwarding.l3_learning:1 1 ARP request 192.168.2.10 => 192.168.2.20
DEBUG:forwarding.l3_learning:1 1 answering ARP for 192.168.2.20
```

```
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu:/home/ubuntu/pox# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.2.10  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::200:ff:fe00:20b  prefixlen 64  scopeid 0x20<link>
        ether 00:00:00:00:02:0b  txqueuelen 1000  (Ethernet)
        RX packets 35  bytes 3849 (3.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18  bytes 1612 (1.6 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>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ubuntu:/home/ubuntu/pox#
root@ubuntu:/home/ubuntu/pox# ping 192.168.2.20
PING 192.168.2.20 (192.168.2.20) 56(84) bytes of data.
64 bytes from 192.168.2.20: icmp_seq=3 ttl=64 time=0.185 ms
64 bytes from 192.168.2.20: icmp_seq=4 ttl=64 time=0.053 ms
64 bytes from 192.168.2.20: icmp_seq=5 ttl=64 time=0.054 ms
64 bytes from 192.168.2.20: icmp_seq=6 ttl=64 time=0.052 ms
64 bytes from 192.168.2.20: icmp_seq=7 ttl=64 time=0.052 ms
64 bytes from 192.168.2.20: icmp_seq=8 ttl=64 time=0.052 ms
64 bytes from 192.168.2.20: icmp_seq=9 ttl=64 time=0.054 ms
^C
--- 192.168.2.20 ping statistics ---
9 packets transmitted, 7 received, 22% packet loss, time 8195ms
rtt min/avg/max/mdev = 0.052/0.071/0.185/0.047 ms
root@ubuntu:/home/ubuntu/pox#
root@ubuntu:/home/ubuntu/pox# ping 192.168.2.20
PING 192.168.2.20 (192.168.2.20) 56(84) bytes of data.
```

Figure 21 - MAC spoofing - detection after a certain number of packets

Zooming into the xterm to show that after a while, packets are effectively blocked.



```
root@ubuntu:/home/ubuntu/pox# ping 192.168.2.20
PING 192.168.2.20 (192.168.2.20) 56(84) bytes of data.
^C
--- 192.168.2.20 ping statistics ---
33 packets transmitted, 0 received, 100% packet loss, time 32759ms

root@ubuntu:/home/ubuntu/pox# 
```

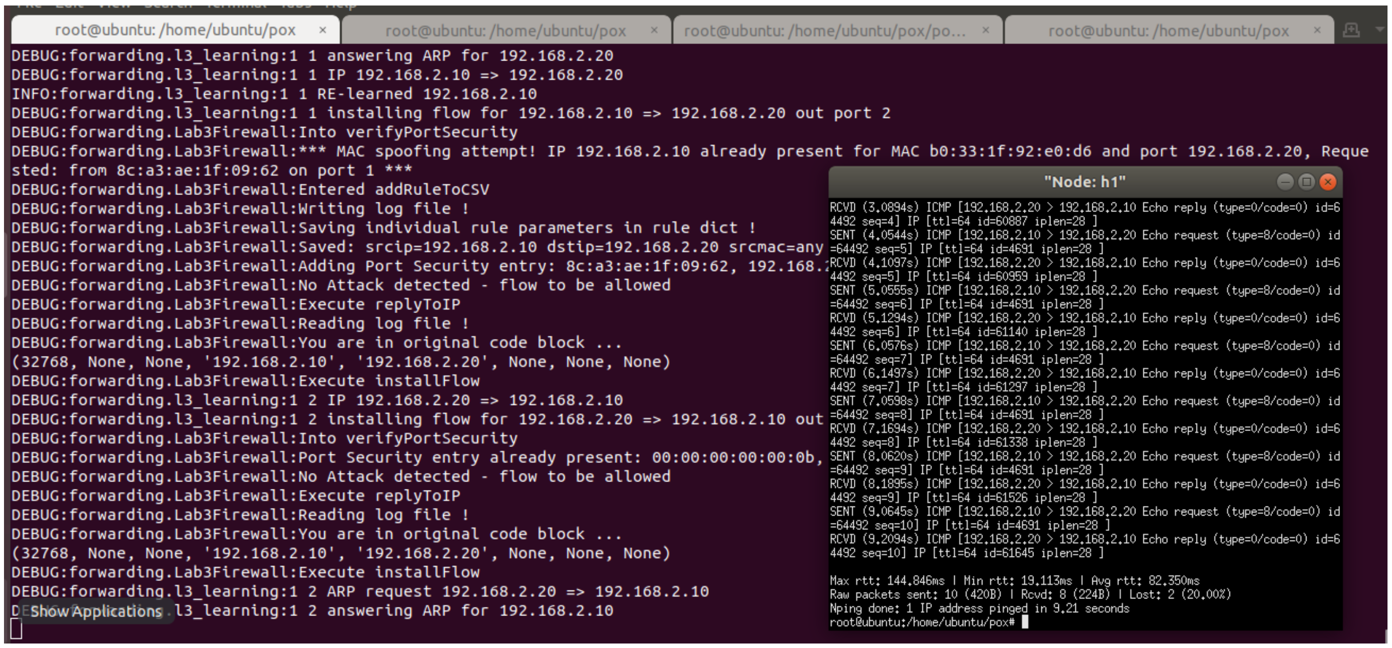
Figure 22 - MAC spoofing - xterm showing all dropped ICMP packets

The next attack is performed using *nping*, which allows generating random IP addresses for an attack. The command I used is the following:

---

nping -c 10 --icmp --source-mac rand 192.168.2.20

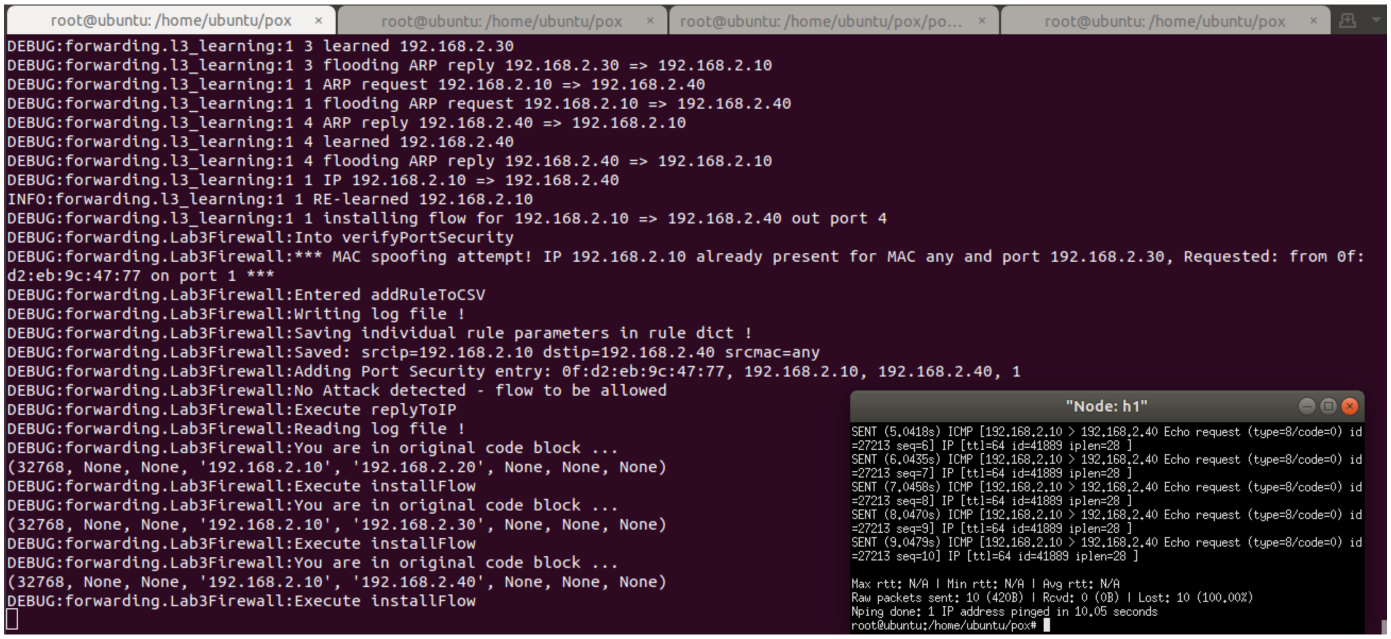
As it is possible to see, here as well some packet comes through in the first attack, but nothing comes through in the second as the switch has "learned".


root@ubuntu: /home/ubuntu/pox
root@ubuntu: /home/ubuntu/pox
root@ubuntu: /home/ubuntu/pox/po...
root@ubuntu: /home/ubuntu/pox
"Node: h1"
Show Applications

```
DEBUG:forwarding.l3_learning:1 1 answering ARP for 192.168.2.20
DEBUG:forwarding.l3_learning:1 1 IP 192.168.2.10 => 192.168.2.20
INFO:forwarding.l3_learning:1 1 RE-learned 192.168.2.10
DEBUG:forwarding.l3_learning:1 1 installing flow for 192.168.2.10 => 192.168.2.20 out port 2
DEBUG:forwarding.Lab3Firewall:Into verifyPortSecurity
DEBUG:forwarding.Lab3Firewall:*** MAC spoofing attempt! IP 192.168.2.10 already present for MAC b0:33:1f:92:e0:d6 and port 192.168.2.20, Reque
sted: from 8c:a3:ae:1f:09:62 on port 1 ***
DEBUG:forwarding.Lab3Firewall:Entered addRuleToCSV
DEBUG:forwarding.Lab3Firewall:Writing log file !
DEBUG:forwarding.Lab3Firewall:Saving individual rule parameters in rule dict !
DEBUG:forwarding.Lab3Firewall:Saved: srcip=192.168.2.10 dstip=192.168.2.20 srcmac=any
DEBUG:forwarding.Lab3Firewall:Adding Port Security entry: 8c:a3:ae:1f:09:62, 192.168.2
DEBUG:forwarding.Lab3Firewall:No Attack detected - flow to be allowed
DEBUG:forwarding.Lab3Firewall:Execute replyToIP
DEBUG:forwarding.Lab3Firewall:Reading log file !
DEBUG:forwarding.Lab3Firewall:You are in original code block ...
(32768, None, None, '192.168.2.10', '192.168.2.20', None, None, None)
DEBUG:forwarding.Lab3Firewall:Execute installFlow
DEBUG:forwarding.l3_learning:1 2 IP 192.168.2.20 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 2 installing flow for 192.168.2.20 => 192.168.2.10 out
DEBUG:forwarding.Lab3Firewall:Into verifyPortSecurity
DEBUG:forwarding.Lab3Firewall:Port Security entry already present: 00:00:00:00:00:0b,
DEBUG:forwarding.Lab3Firewall:No Attack detected - flow to be allowed
DEBUG:forwarding.Lab3Firewall:Execute replyToIP
DEBUG:forwarding.Lab3Firewall:Reading log file !
DEBUG:forwarding.Lab3Firewall:You are in original code block ...
(32768, None, None, '192.168.2.10', '192.168.2.20', None, None, None)
DEBUG:forwarding.Lab3Firewall:Execute installFlow
DEBUG:forwarding.l3_learning:1 2 ARP request 192.168.2.20 => 192.168.2.10
D                   l3_learning:1 2 answering ARP for 192.168.2.10
```

```
RCVD (3.0894s) ICMP [192.168.2.20 > 192.168.2.10 Echo reply (type=0/code=0) id=6
4492 seq=4] IP [ttl=64 id=60887 iplen=28 ]
SENT (4.0544s) ICMP [192.168.2.10 > 192.168.2.20 Echo request (type=8/code=0) id
=64492 seq=5] IP [ttl=64 id=4691 iplen=28 ]
RCVD (4.1097s) ICMP [192.168.2.20 > 192.168.2.10 Echo reply (type=0/code=0) id=6
4492 seq=5] IP [ttl=64 id=60959 iplen=28 ]
SENT (5.0555s) ICMP [192.168.2.10 > 192.168.2.20 Echo request (type=8/code=0) id
=64492 seq=6] IP [ttl=64 id=4691 iplen=28 ]
RCVD (5.1294s) ICMP [192.168.2.20 > 192.168.2.10 Echo reply (type=0/code=0) id=6
4492 seq=6] IP [ttl=64 id=61140 iplen=28 ]
SENT (6.0576s) ICMP [192.168.2.10 > 192.168.2.20 Echo request (type=8/code=0) id
=64492 seq=7] IP [ttl=64 id=4691 iplen=28 ]
RCVD (6.1497s) ICMP [192.168.2.20 > 192.168.2.10 Echo reply (type=0/code=0) id=6
4492 seq=7] IP [ttl=64 id=61297 iplen=28 ]
SENT (7.0598s) ICMP [192.168.2.10 > 192.168.2.20 Echo request (type=8/code=0) id
=64492 seq=8] IP [ttl=64 id=4691 iplen=28 ]
RCVD (7.1694s) ICMP [192.168.2.20 > 192.168.2.10 Echo reply (type=0/code=0) id=6
4492 seq=8] IP [ttl=64 id=61338 iplen=28 ]
SENT (8.0620s) ICMP [192.168.2.10 > 192.168.2.20 Echo request (type=8/code=0) id
=64492 seq=9] IP [ttl=64 id=4691 iplen=28 ]
RCVD (8.1895s) ICMP [192.168.2.20 > 192.168.2.10 Echo reply (type=0/code=0) id=6
4492 seq=9] IP [ttl=64 id=61526 iplen=28 ]
SENT (9.0645s) ICMP [192.168.2.10 > 192.168.2.20 Echo request (type=8/code=0) id
=64492 seq=10] IP [ttl=64 id=4691 iplen=28 ]
RCVD (9.2094s) ICMP [192.168.2.20 > 192.168.2.10 Echo reply (type=0/code=0) id=6
4492 seq=10] IP [ttl=64 id=61645 iplen=28 ]

Max rtt: 144.846ms | Min rtt: 19.113ms | Avg rtt: 82.350ms
Raw packets sent: 10 (420B) | Rcvd: 8 (224B) | Lost: 2 (20.00%)
Nping done: 1 IP address pinged in 9.21 seconds
root@ubuntu:/home/ubuntu/pox# 
```

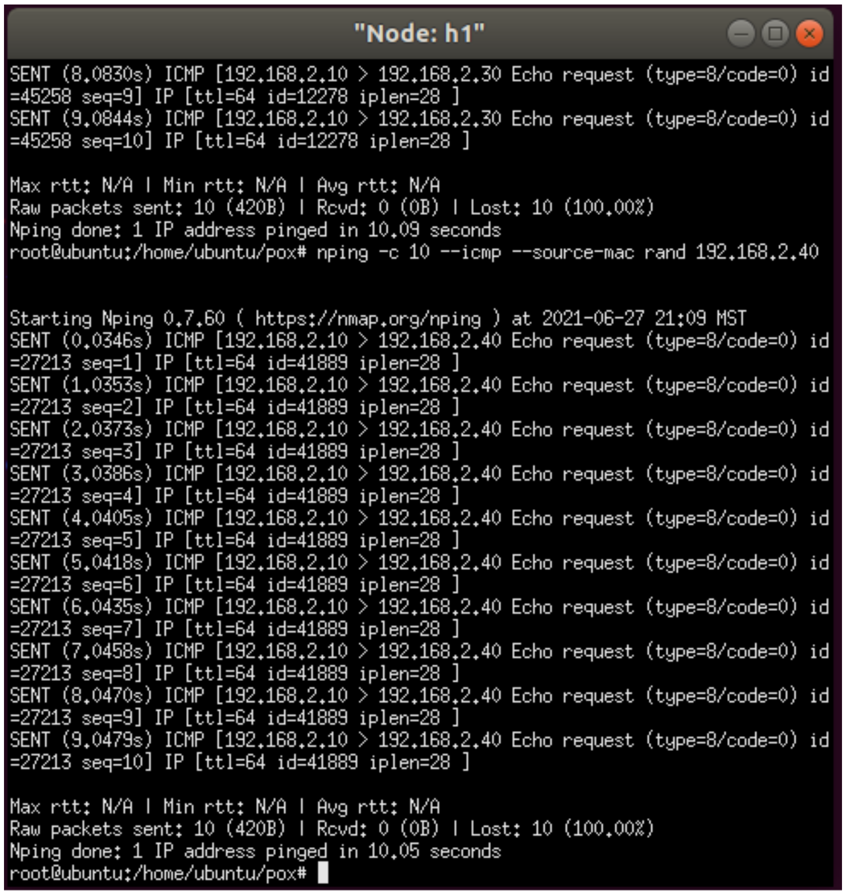
Figure 23 - MAC spoofing – nping – first attack


root@ubuntu: /home/ubuntu/pox
root@ubuntu: /home/ubuntu/pox
root@ubuntu: /home/ubuntu/pox/po...
root@ubuntu: /home/ubuntu/pox
"Node: h1"

```
DEBUG:forwarding.l3_learning:1 3 learned 192.168.2.30
DEBUG:forwarding.l3_learning:1 3 flooding ARP reply 192.168.2.30 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 1 ARP request 192.168.2.10 => 192.168.2.40
DEBUG:forwarding.l3_learning:1 1 flooding ARP request 192.168.2.10 => 192.168.2.40
DEBUG:forwarding.l3_learning:1 4 ARP reply 192.168.2.40 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 4 learned 192.168.2.40
DEBUG:forwarding.l3_learning:1 4 flooding ARP reply 192.168.2.40 => 192.168.2.10
DEBUG:forwarding.l3_learning:1 1 IP 192.168.2.10 => 192.168.2.40
INFO:forwarding.l3_learning:1 1 RE-learned 192.168.2.10
DEBUG:forwarding.l3_learning:1 1 installing flow for 192.168.2.10 => 192.168.2.40 out port 4
DEBUG:forwarding.Lab3Firewall:Into verifyPortSecurity
DEBUG:forwarding.Lab3Firewall:*** MAC spoofing attempt! IP 192.168.2.10 already present for MAC any and port 192.168.2.30, Requested: from 0f:
d2:eb:9c:47:77 on port 1 ***
DEBUG:forwarding.Lab3Firewall:Entered addRuleToCSV
DEBUG:forwarding.Lab3Firewall:Writing log file !
DEBUG:forwarding.Lab3Firewall:Saving individual rule parameters in rule dict !
DEBUG:forwarding.Lab3Firewall:Saved: srcip=192.168.2.10 dstip=192.168.2.40 srcmac=any
DEBUG:forwarding.Lab3Firewall:Adding Port Security entry: 0f:d2:eb:9c:47:77, 192.168.2.10, 192.168.2.40, 1
DEBUG:forwarding.Lab3Firewall:No Attack detected - flow to be allowed
DEBUG:forwarding.Lab3Firewall:Execute replyToIP
DEBUG:forwarding.Lab3Firewall:Reading log file !
DEBUG:forwarding.Lab3Firewall:You are in original code block ...
(32768, None, None, '192.168.2.10', '192.168.2.20', None, None, None)
DEBUG:forwarding.Lab3Firewall:Execute installFlow
DEBUG:forwarding.Lab3Firewall:You are in original code block ...
(32768, None, None, '192.168.2.10', '192.168.2.30', None, None, None)
DEBUG:forwarding.Lab3Firewall:Execute installFlow
DEBUG:forwarding.Lab3Firewall:You are in original code block ...
(32768, None, None, '192.168.2.10', '192.168.2.40', None, None, None)
DEBUG:forwarding.Lab3Firewall:Execute installFlow
```

```
SENT (5.0418s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=6] IP [ttl=64 id=41889 iplen=28 ]
SENT (6.0435s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=7] IP [ttl=64 id=41889 iplen=28 ]
SENT (7.0458s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=8] IP [ttl=64 id=41889 iplen=28 ]
SENT (8.0470s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=9] IP [ttl=64 id=41889 iplen=28 ]
SENT (9.0479s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=10] IP [ttl=64 id=41889 iplen=28 ]

Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Raw packets sent: 10 (420B) | Rcvd: 0 (0B) | Lost: 10 (100.00%)
Nping done: 1 IP address pinged in 10.05 seconds
root@ubuntu:/home/ubuntu/pox#
```

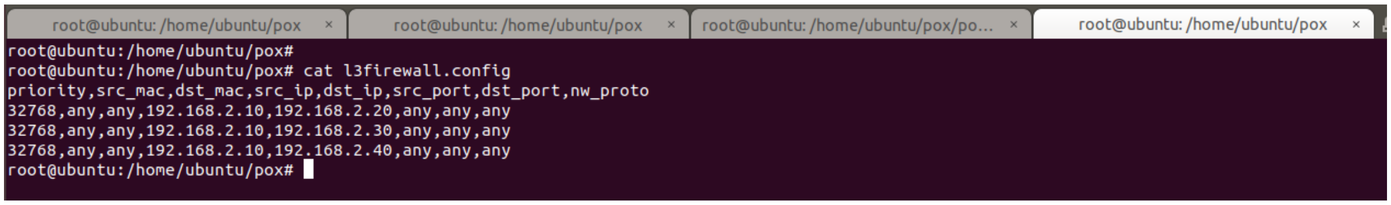
Figure 24 - MAC spoofing – nping – second attack



```
SENT (8.0830s) ICMP [192.168.2.10 > 192.168.2.30 Echo request (type=8/code=0) id
=45258 seq=9] IP [ttl=64 id=12278 iplen=28 ]
SENT (9.0844s) ICMP [192.168.2.10 > 192.168.2.30 Echo request (type=8/code=0) id
=45258 seq=10] IP [ttl=64 id=12278 iplen=28 ]

Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Raw packets sent: 10 (420B) | Rcvd: 0 (0B) | Lost: 10 (100.00%)
Nping done: 1 IP address pinged in 10.09 seconds
root@ubuntu:/home/ubuntu/pox# nping -c 10 --icmp --source-mac rand 192.168.2.40


Starting Nping 0.7.60 ( https://nmap.org/nping ) at 2021-06-27 21:09 MST
SENT (0.0346s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=1] IP [ttl=64 id=41889 iplen=28 ]
SENT (1.0353s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=2] IP [ttl=64 id=41889 iplen=28 ]
SENT (2.0373s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=3] IP [ttl=64 id=41889 iplen=28 ]
SENT (3.0386s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=4] IP [ttl=64 id=41889 iplen=28 ]
SENT (4.0405s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=5] IP [ttl=64 id=41889 iplen=28 ]
SENT (5.0418s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=6] IP [ttl=64 id=41889 iplen=28 ]
SENT (6.0435s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=7] IP [ttl=64 id=41889 iplen=28 ]
SENT (7.0458s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=8] IP [ttl=64 id=41889 iplen=28 ]
SENT (8.0470s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=9] IP [ttl=64 id=41889 iplen=28 ]
SENT (9.0479s) ICMP [192.168.2.10 > 192.168.2.40 Echo request (type=8/code=0) id
=27213 seq=10] IP [ttl=64 id=41889 iplen=28 ]

Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Raw packets sent: 10 (420B) | Rcvd: 0 (0B) | Lost: 10 (100.00%)
Nping done: 1 IP address pinged in 10.05 seconds
root@ubuntu:/home/ubuntu/pox#
```

Figure 25 - MAC spoofing – nping – second attack, zoom in

It is worth noticing that detected malicious flows are both blocked immediately – by issuing a message to OVS – and also saved into the l3firewall.config configuration file, to be resumed later.



```
root@ubuntu:/home/ubuntu/pox#
root@ubuntu:/home/ubuntu/pox# cat l3firewall.config
priority,src_mac,dst_mac,src_ip,dst_ip,src_port,dst_port,nw_proto
32768,any,any,192.168.2.10,192.168.2.20,any,any,any
32768,any,any,192.168.2.10,192.168.2.30,any,any,any
32768,any,any,192.168.2.10,192.168.2.40,any,any,any
root@ubuntu:/home/ubuntu/pox#
```

Figure 26 – MAC spoofing – flows captured in l3firewall.config

To be able to restart the Lab without being affected by flows detected in a previous run, I have created an empty l3firewall.config (*l3firewall.config.empty*) file with just the row headers, and I am copying it over the l3firewall.config file generated by the mitigation software in the previous run.

V. Appendix A: Files for the Lab

Please find the list of files created for this lab and mentioned throughout this document, plus their GitHub link for download.

The overall GitHub directory for the project is: https://github.com/markoer73/CSE-548/tree/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall

Project-Report-3 SDN-Based DoS Attacks and Mitigation.docx	https://github.com/markoer73/CSE-548/blob/b44720f56fb7d262a8fb80d8077dc88e96812c74/Project%203%20-%20SDN-Based%20DoS%20Attacks%20and%20Mitigation/Project-Report-3%20SDN-Based%20Stateless%20Firewall.docx
Lab3Firewall.py	https://github.com/markoer73/CSE-548/blob/b44720f56fb7d262a8fb80d8077dc88e96812c74/Project%203%20-%20SDN-Based%20DoS%20Attacks%20and%20Mitigation/Lab3Firewall.py
l3firewall.config.empty	https://github.com/markoer73/CSE-548/blob/b44720f56fb7d262a8fb80d8077dc88e96812c74/Project%203%20-%20SDN-Based%20DoS%20Attacks%20and%20Mitigation/l3firewall.config.empty
l3firewall.config	https://github.com/markoer73/CSE-548/blob/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall/l3firewall.config
lab3ips.sh	https://github.com/markoer73/CSE-548/blob/b44720f56fb7d262a8fb80d8077dc88e96812c74/Project%203%20-%20SDN-Based%20DoS%20Attacks%20and%20Mitigation/lab3ips.sh
run_lab3.sh	https://github.com/markoer73/CSE-548/blob/b44720f56fb7d262a8fb80d8077dc88e96812c74/Project%203%20-%20SDN-Based%20DoS%20Attacks%20and%20Mitigation/run_lab3.sh
run_pox3.sh	https://github.com/markoer73/CSE-548/blob/b44720f56fb7d262a8fb80d8077dc88e96812c74/Project%203%20-%20SDN-Based%20DoS%20Attacks%20and%20Mitigation/run_pox3.sh

VI. File Content

The file content has been reported inline with the text where applicable.

VII. References

- POX Github: https://noxrepo.github.io/pox-doc/html/
- POX Controller Tutorial: http://sdnhub.org/tutorials/pox/
- Open vSwitch Cheat Sheet: https://therandomsecurityguy.com/openvswitch-cheat-sheet/
- Containernet: https://containernet.github.io/
- Containernet tutorial: https://github.com/containernet/containernet/wiki/Tutorial:-Getting-Started
- Port security: https://packetlife.net/blog/2010/may/3/port-security/

VIII. Table of Figures

Contents