Student Name: Nana Ama Ahenkan Boakye-Ansah  
Email: nboakyea@asu.edu  
Submission Date: June 23, 2024  
Class Name and Term: CSE548 Summer 2024

Project Report 3: SDN-Based DoS Attacks and Mitigation

# Project Overview

In this project, I am required to simulate a DOS attack on the SDN controller using a POX controller and containernet or mininet to simulate the network in a simulated container environment. 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 to implement the required firewall security policies.

# Network Setup

My VM is set-up to a bridged network configuration for this lab because a bridge is required. In Bridged mode my VM will receive its own IP address if DHCP is enabled in the network and not giving it a Static IP address. This IP address will be assigned to my router.

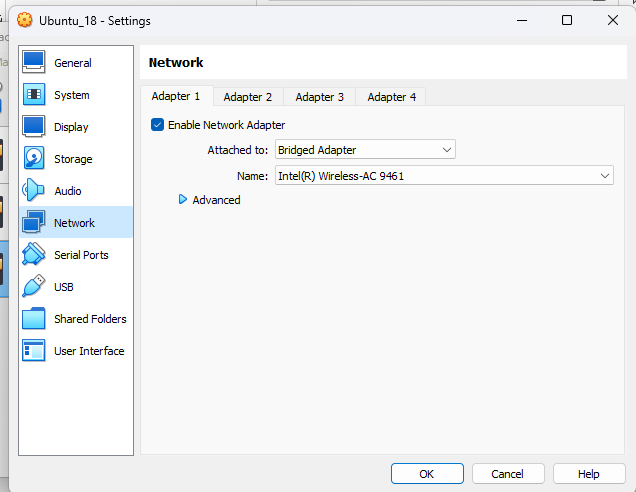


Figure 1 - Bridged network setup in VirtualBox

# Software

Describe major software and network services are used in this project to accomplish your goal.

1. Open vSwitch
2. Various network tools (specifically, tcpdump, ping, traceroute, hping3, and nc – netcat)
3. POX Controller
4. Minenet and Containernet

# Project Description

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

1. *Run Pox Controller:*

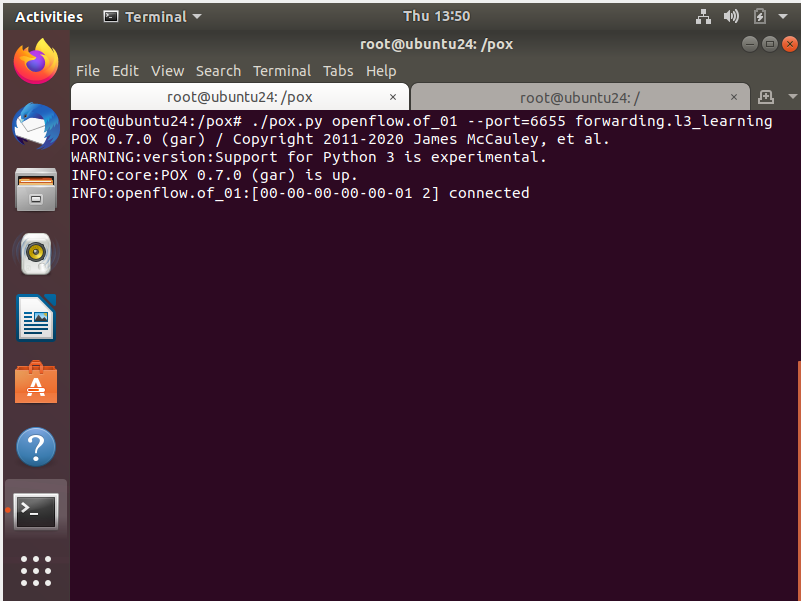


Figure 2-Running Controller

## Setting up mininet and Running mininet topology

To see whether our mininet is working, I am running a mininet topology on the command line

* **sudo mn --topo=single,9 --controller=remote,port=6633 --controller=remote,port=6655 --switch=ovsk –mac**
* **mn -c**



Figure 3 – Running mininet

1. a. Specifying from a host to initiate a command

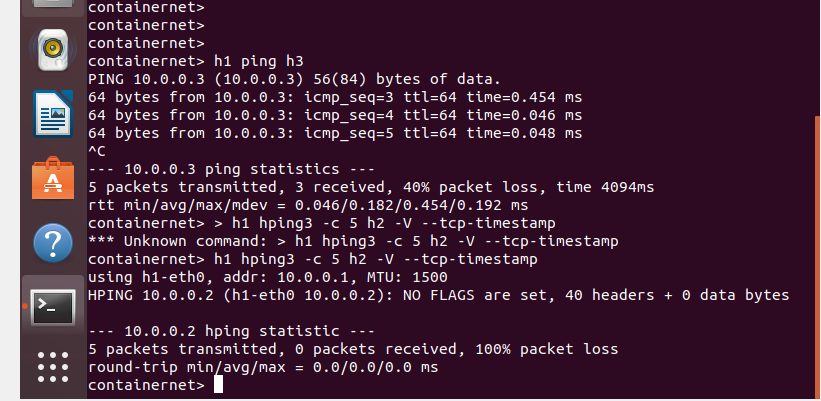


Figure 4- Specifying from a host to initiate a command

b. Start a host terminal and use network configuration to perform the test. For example, the following command is to start an x-terminal of host h1:

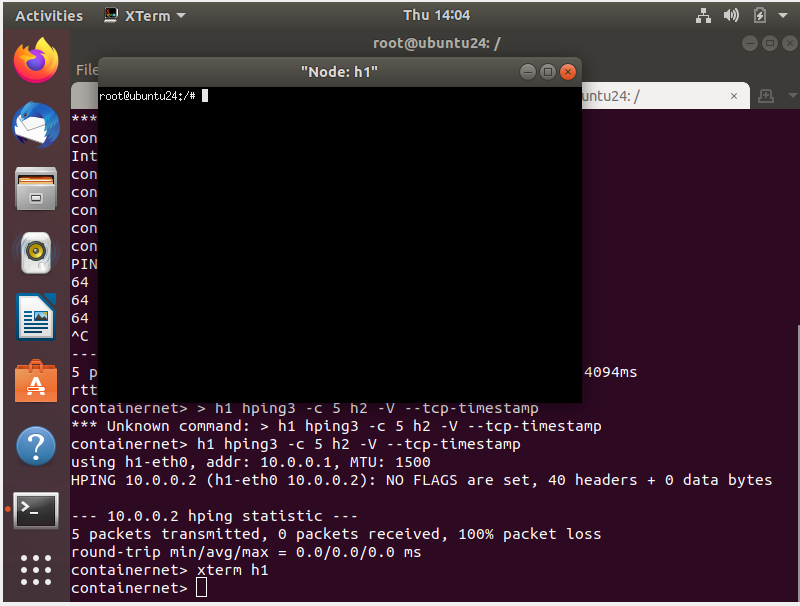


Figure 5-Start a host terminal

c. Once the x-terminal is started you can run the above commands just like in a real host command line, such as:

**$ ping 10.0.0.3 % ping from h1 to h3 (10.0.0.3)**

**$ hping3 -c 5 10.0.0.2 -V --tcp-timestamp % use hping3 to sent tcp packets to h2 (10.0.0.2)**

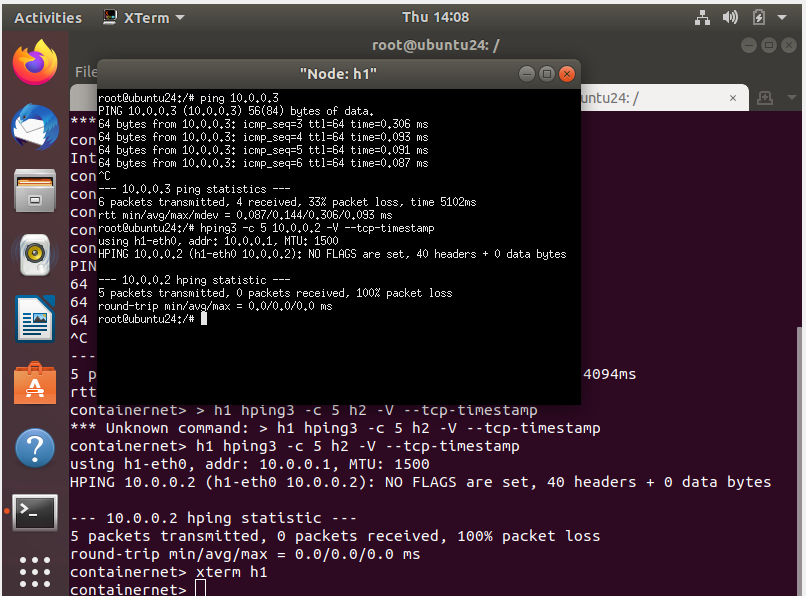


Figure 6- Running hping command

**Task 2.2 Simulate DDoS attack**

1. Open another terminal window and run the following command. Run OVS commands that will verify the functionality of POX.

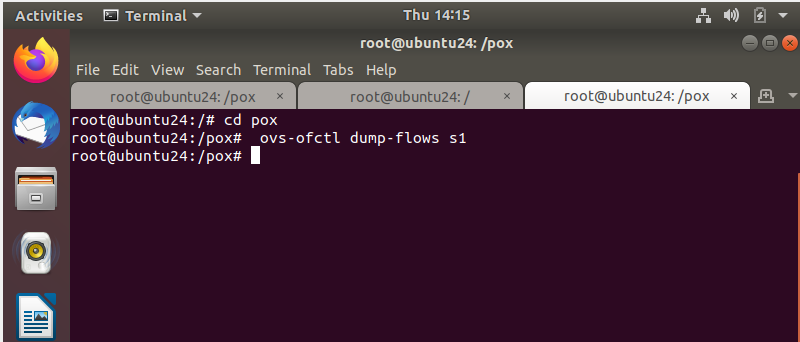


Figure 7-run OVS commands that will verify the functionality of POX

b. Start flooding from host h1 to h2. From the x-terminal window, send spoofed tcp syn-flood packets from h1 to h2 with random source IP to the h2 node

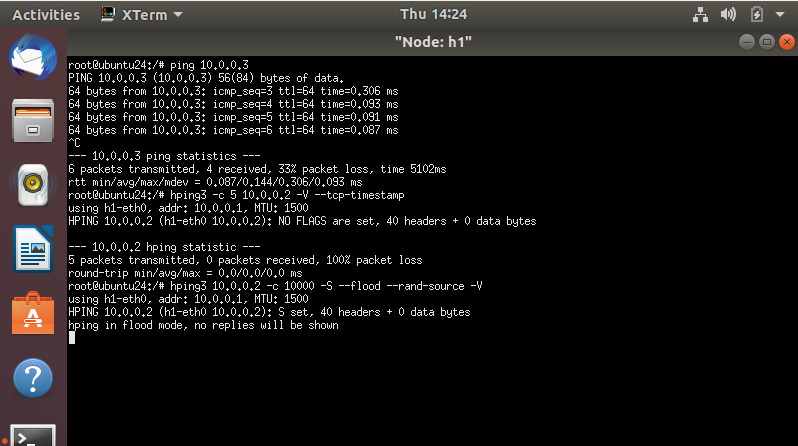


Figure 8-send spoofed tcp syn-flood packets from h1 to h2

c. In a separate terminal, check openvswitch flow entries

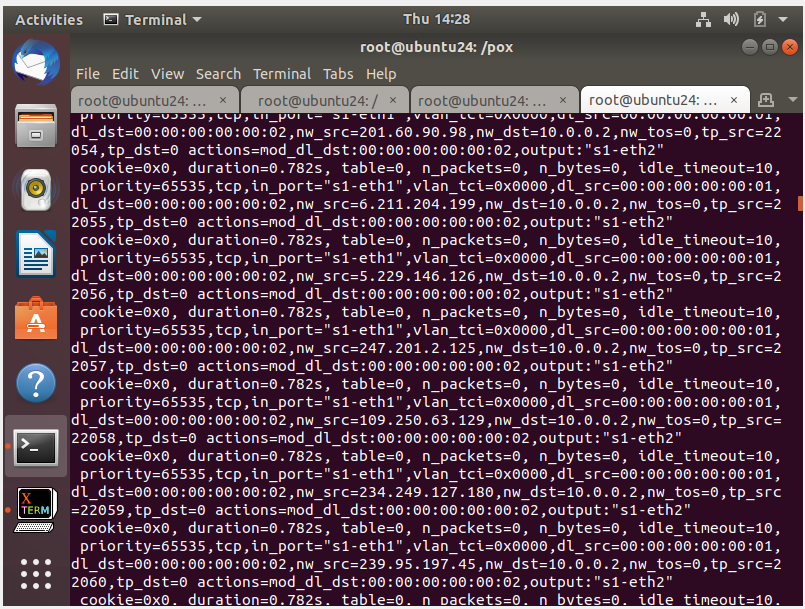


Figure 9-successfully attempted DOS attack on SDN controller

d. Now ping from host h4 to host h9 from the same containernet environment**. $ containernet> h4 ping h9**

We should be able to see ICMP” destination Host Unreachable” response for some time. This demonstrates that the SDN controller was flooded and cannot respond to any new incoming flows.

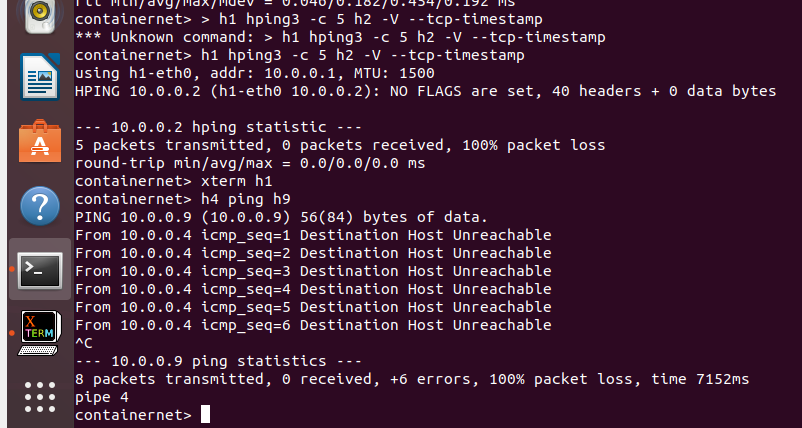


Figure 10-SDN controller was flooded and cannot respond to any new incoming

e. You can stop the flooding by killing the hping3 using Ctrl+C in h1’s x-terminal. After waiting for a few more seconds, the service will resume and we can see valid ping responses from h9 back to h4. This shows the controller is resumed back to work normally

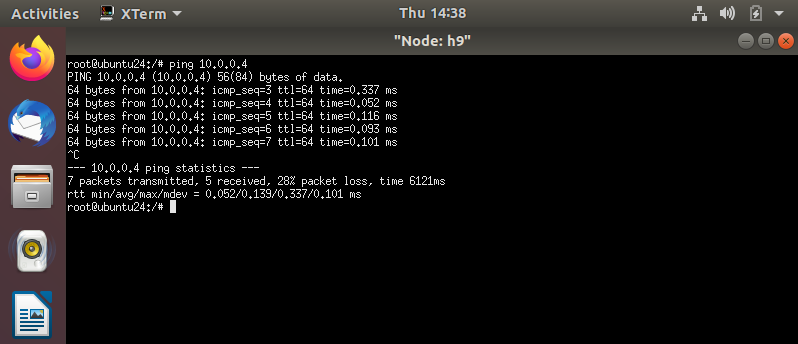


Figure 11- SDN controller stopped flooding

**Task 2.3 Mitigate DoS Attack by implementing port security**

1. Run POX controller with firewall application this time, where we assume $POX DIR is the POX directory where pox source codes present:

**$ cd pox**

It is where your pox file is located, a common location is at **$ sudo ./pox.py openflow.of\_01 --port=6655**

**pox. forwarding.l3\_learning pox.forwarding.L3Firewall**

Here POX controller is invoked by./pox.py command and we run l3 learning.py application and L3Firewall.py file from the POX controller. All the forwarding applications have to be stored in /pox/pox/- forwarding directory. To give a relative path from the directory where the POX binary is present, we follow convention pox.

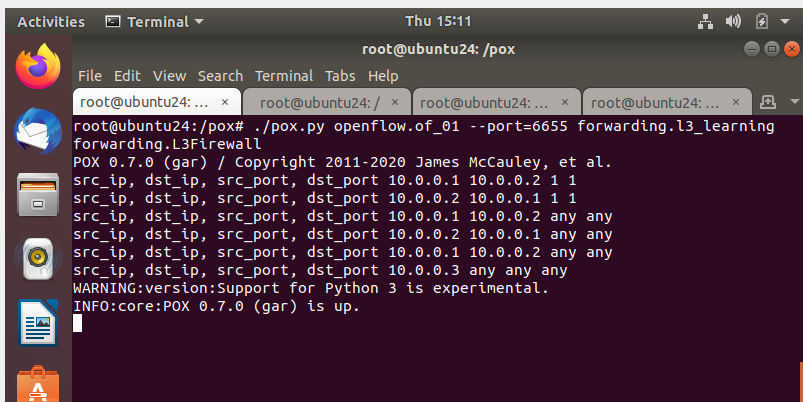


Figure 12-POX controller is invoked by ./pox.py command

b. Now, you can verify if the flooding attack to the DNS controller will work or not by starting flooding again from host h1 to h2. From the x-terminal window, send spoofed tcp syn-flood packets from h1 to h2 with random source IP to the h2 node

$ hping3 10.0.0.2 -c 10000 -S --flood --rand-source -V 3. In a different command-line terminal, check openvswitch flow entries**. $ ovs-ofctl dump-flows s1**

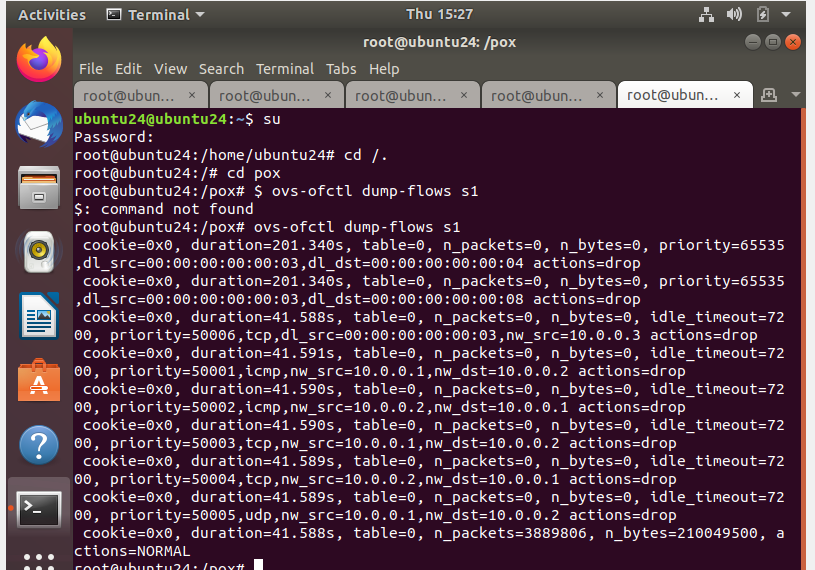


Figure 13-Verify if Flooding attack will work

**Task 3.0 Performing DoS attack and Blocking DoS attack**

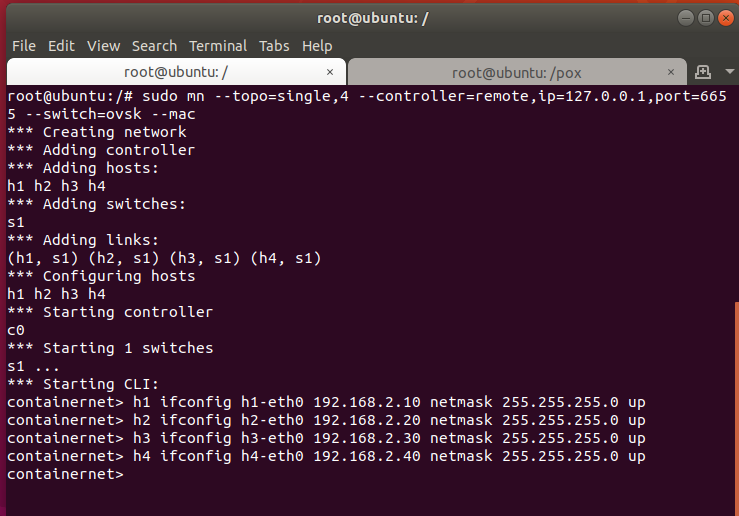


Figure 14-Settiing up Mininet Topology

You run this in the mininet CLI

* **h1 ifconfig h1-eth0 192.168.2.10 netmask 255.255.255.0 up**
* **h2 ifconfig h2-eth0 192.168.2.20 netmask 255.255.255.0 up**
* **h3 ifconfig h3-eth0 192.168.2.30 netmask 255.255.255.0 up**
* **h4 ifconfig h4-eth0 192.168.2.40 netmask 255.255.255.0 up**

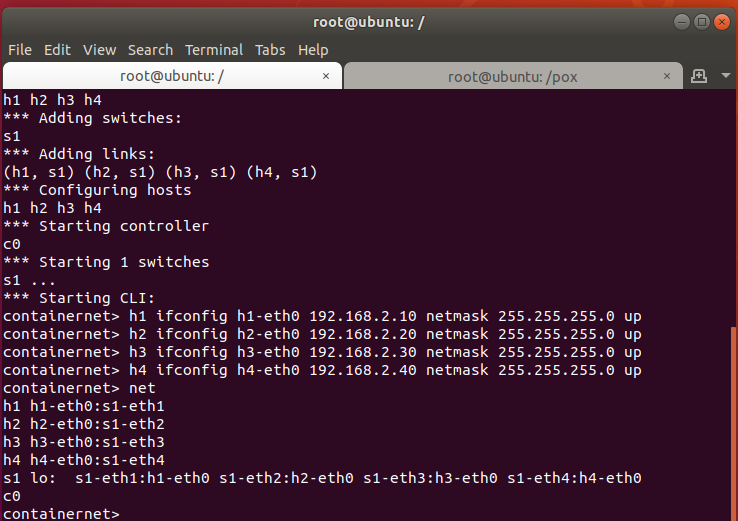


Figure 15-Mininet Topology

Screenshot showing the IP addresses changed:

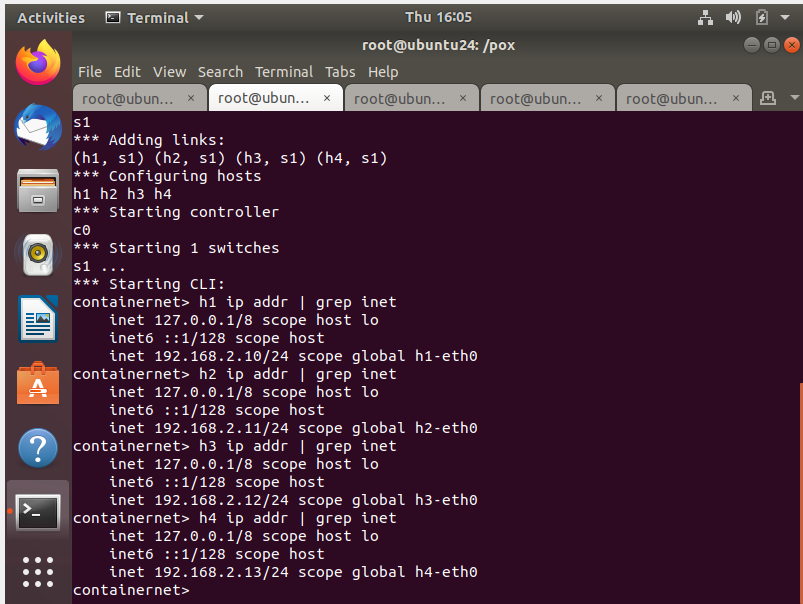


Figure 16-IP Addresses Changed

## **Perform Flood attack on SDN controller following a suggested procedure**

### Run l3 learning application in POX controller.

The script “*lab2.sh*” is executed so we can execute the script multiple times as needed

set -x

sudo ./pox.py openflow.of\_01 \

--port=6655 pox.forwarding.l3\_learning \

pox.forwarding.L3Firewall \

--l2config="l2firewall.config" \

--l3config="l3firewall.config" \

log.level –DEBUG

### Check openflow flow-entries on switch 1.

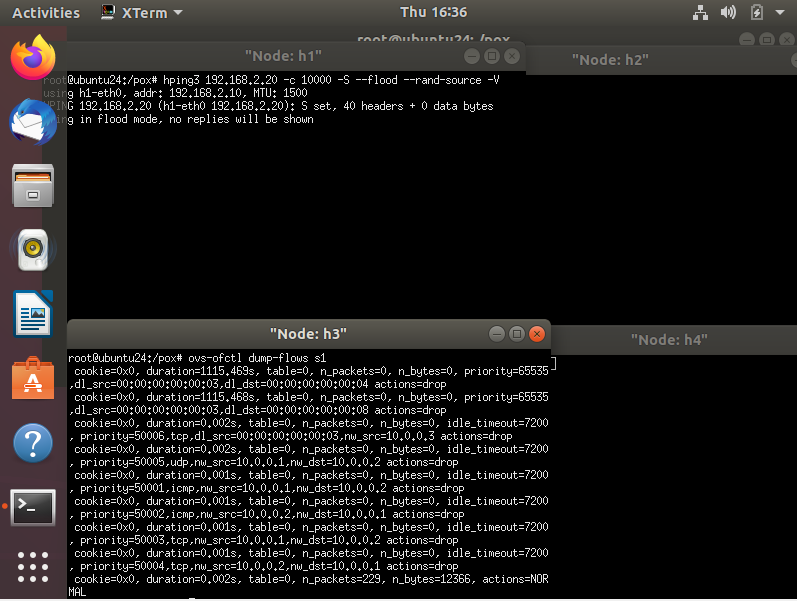
### Start flooding from any container host to container host #2

### Check Openflow flow entries at switch 1

I will illustrate the Denial of Service happening on OVS because of the flood attack.

* I have set up four X terminals of the four containers on containernet, 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.



Next, I have started the flood attack from the container h1 to h3. On container h2 I am trying to ping container 4, but with no success – the OVS controller is effectively working and does not forward the flows. Conversely, the containers are quite responsive – it’s just their networking that is affected.

I dumped the OVS flows on container – there are hundreds, and do not stop increase even after I have stopped the flood.

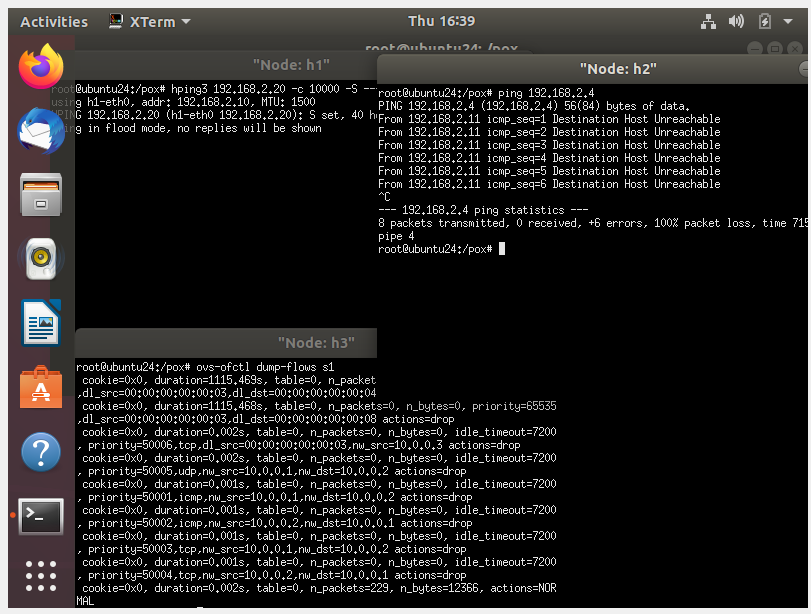


Figure 17-Start flooding from any container host to container host #2

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.

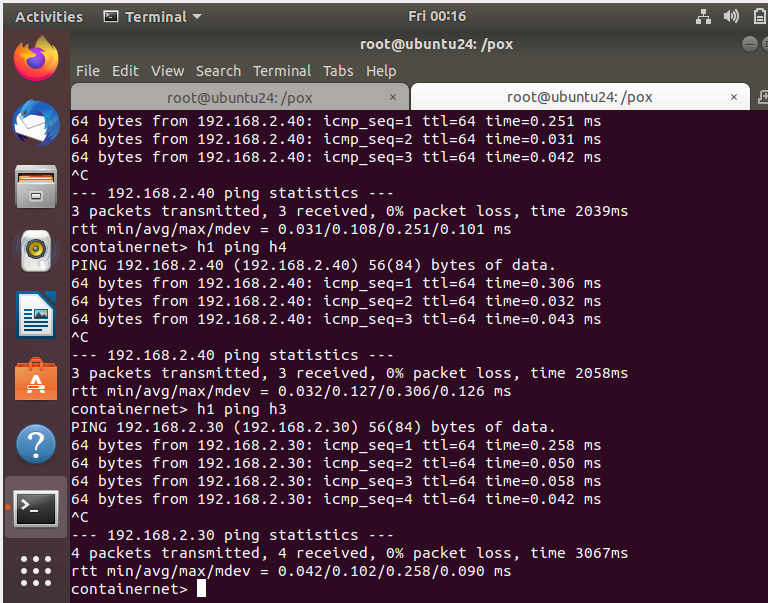


Figure 18-OVS resume Operation

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

### You should illustrate (through screenshots and descriptions) your implemented program codes.

### You should demo how your implementation can mitigate the DoS through a sequence of screenshots with explanation.

### You should submit the source codes of your implementation.

In this lab, I have modified the provided L3Firewall.py to be Lspoof.py

by creating an empty table to keep track of MAC addresses and their associated IP addresses.

The logic for spoofing and DoS detection is implemented by creating an empty table to keep track of MAC addresses and their associated IP addresses. When a new data packet (flow) arrives from a device with a specific MAC address, 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 example:

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)

This is how to go about with the process:

If the IP address is new, add or update the port table to link this MAC address with the new IP address.

If the IP address is not new (meaning the MAC address already has this IP address recorded):

If the MAC address is trying to use multiple IP addresses (indicating possible spoofing), block this MAC address from sending more data.

* 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, 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**

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:

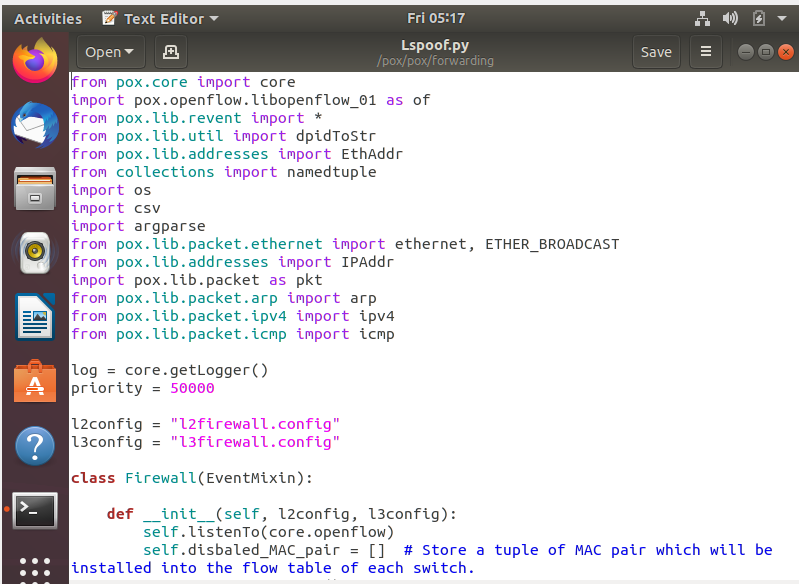
Source MAC Source IP Victim’s IP OVS switchport

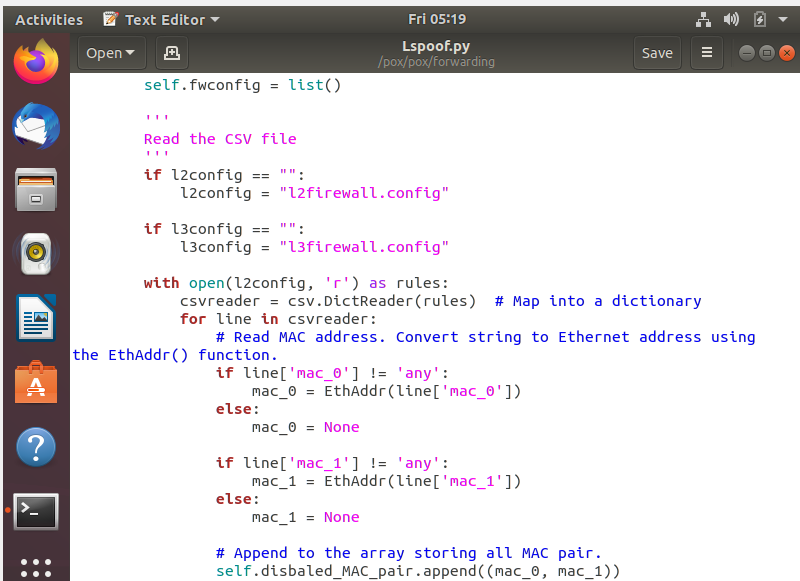
Changing Constant Constant Constant

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:

* **Initiate a Port Table (PT)**: Start by creating an empty table to keep track of MAC addresses and their associated IP addresses.
* **For any newly received flow (F) from a source MAC address (F.SrcMAC)**: When a new data packet (flow) arrives from a device with a specific MAC address:
* **If F.SrcIP is new**: Check if the IP address (F.SrcIP) in the flow is new for the given MAC address.
* **Update PT with the mapping F.SrcMAC <--> F.SrcIP**: If the IP address is new, add or update the port table to link this MAC address with the new IP address.
* **Else**: If the IP address is not new (meaning the MAC address already has this IP address recorded):
* **Block F.SrcMAC**: If the MAC address is trying to use multiple IP addresses (indicating possible spoofing), block this MAC address from sending more data.
* **End**: Finish the process and wait for the next flow.





A screenshot of a computer program

Description automatically generated

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

### IP Spoofing Attack Mitigation

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

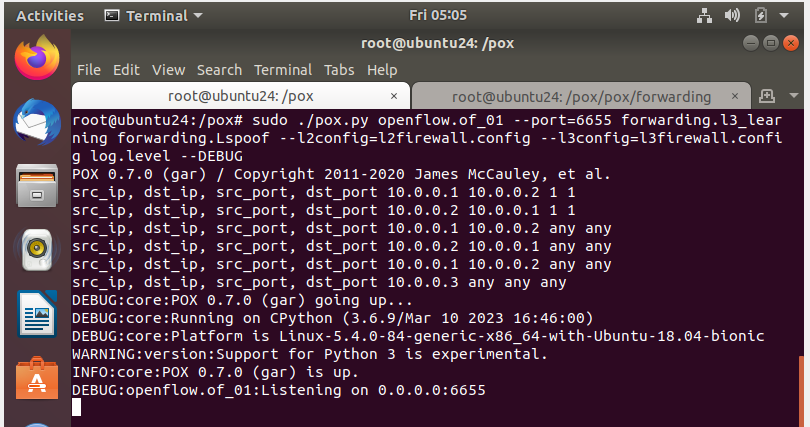


Figure 19- IP Spoofing - Running POX

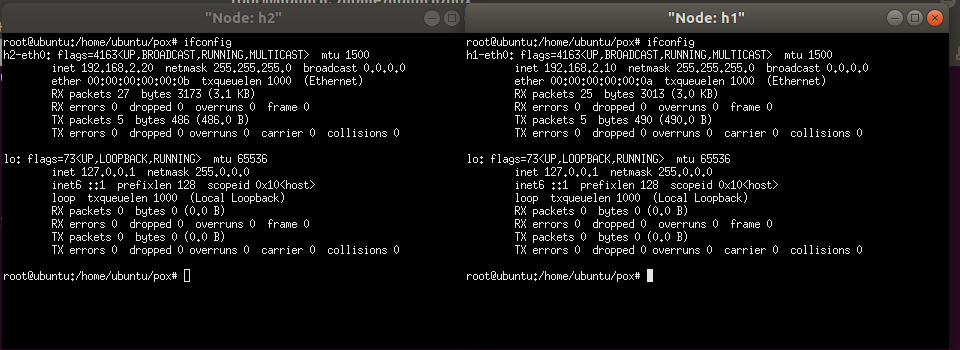


Figure 20 - IP Spoofing - h1 and h2 with their starting IPs and MAC addresses

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.

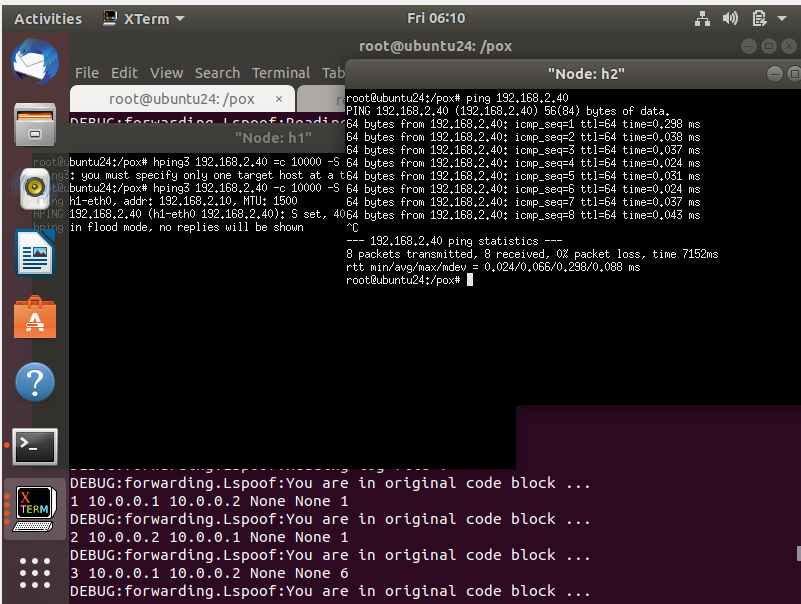


Figure 21 - IP Spoofing - Running hping3 on h1 and running ping on h2

This sets up a firewall using the POX OpenFlow controller framework. It monitors incoming packets, maintains a mapping of MAC addresses to IP addresses, detects IP spoofing attacks, and blocks or allows flows based on the detection results.

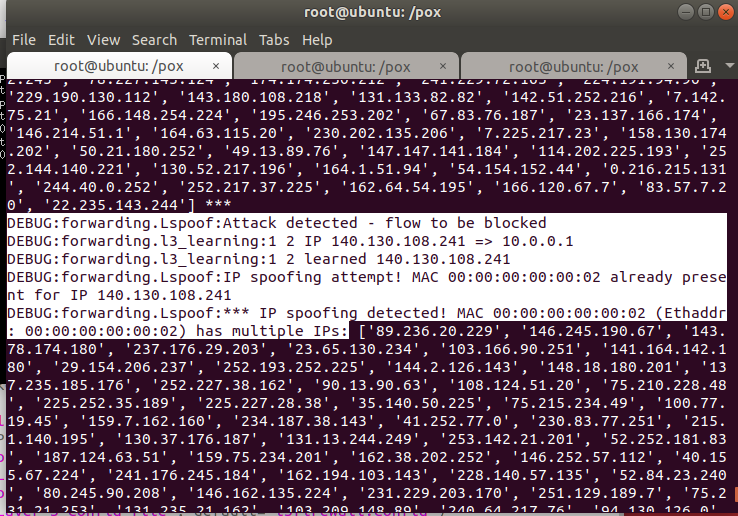


Figure 22-IP Spoofing - debug messages showing malicious packets being blocked

Debug messages:

DEBUG:forwarding.Lspoof:Attack detected - flow to be blocked

DEBUG:forwarding.l3\_learning:1 2 IP 83.57.7.20 => 10.0.0.1

DEBUG:forwarding.l3\_learning:1 2 learned 83.57.7.20

DEBUG:forwarding.Lspoof:IP spoofing attempt! MAC 00:00:00:00:00:02 already present for IP 83.57.7.20

DEBUG:forwarding.Lspoof:\*\*\* IP spoofing detected! MAC 00:00:00:00:00:02 (Ethaddr: 00:00:00:00:00:02) has multiple IPs: ['89.236.20.229', '146.245.190.67', '143.78.174.180', '237.176.29.203', '23.65.130.234', '103.166.90.251', '141.164.142.180', '29.154.206.237', '252.193.252.225', '144.2.126.143', '148.18.180.201', '137.235.185.176', '252.227.38.162', '90.13.90.63', '108.124.51.20', '75.210.228.48', '225.252.35.189', '225.227.28.38', '35.140.50.225', '75.215.234.49', '100.77.19.45', '159.7.162.160', '234.187.38.143', '41.252.77.0', '230.83.77.251', '215.1.140.195', '130.37.176.187', '131.13.244.249', '253.142.21.201', '52.252.181.83', '187.124.63.51', '159.75.234.201', '162.38.202.252', '146.252.57.112', '40.155.67.224', '241.176.245.184', '162.194.103.143', '228.140.57.135', '52.84.23.240', '80.245.90.208', '146.162.135.224', '231.229.203.170', '251.129.189.7', '75.231.21.253', '131.235.21.162', '103.209.148.89', '240.64.217.76', '94.130.126.0', '209.57.174.92', '1.208.23.184', '131.222.225.163', '48.176.23.251', '90.225.187.112',………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.

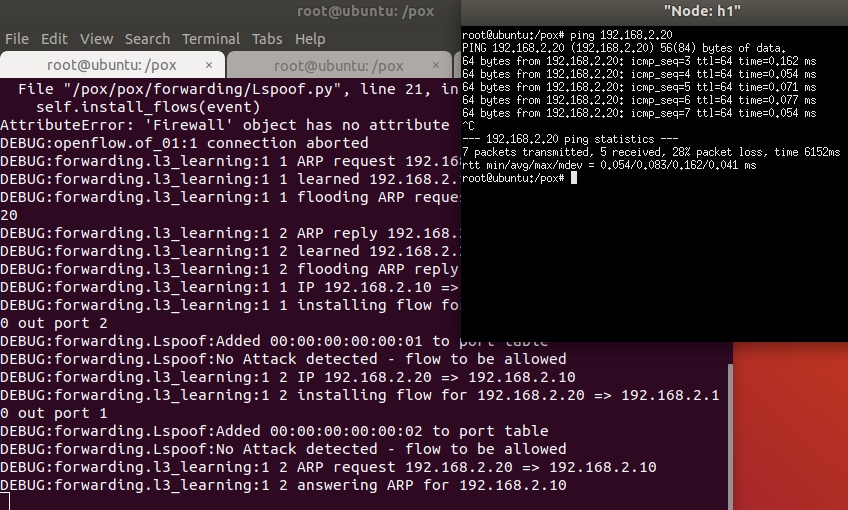


Figure 23-MAC Spoofing Set up



Figure 24 - 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.

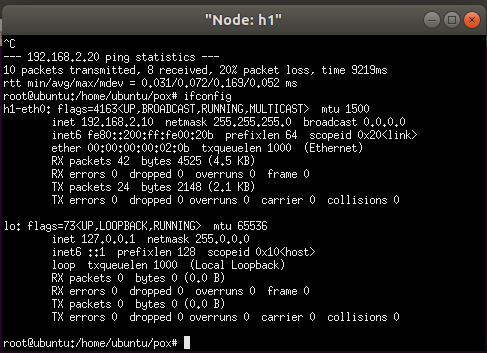


Figure 25 - MAC spoofing - confirmation of MAC change

From the next screenshot the whole picture is presented.

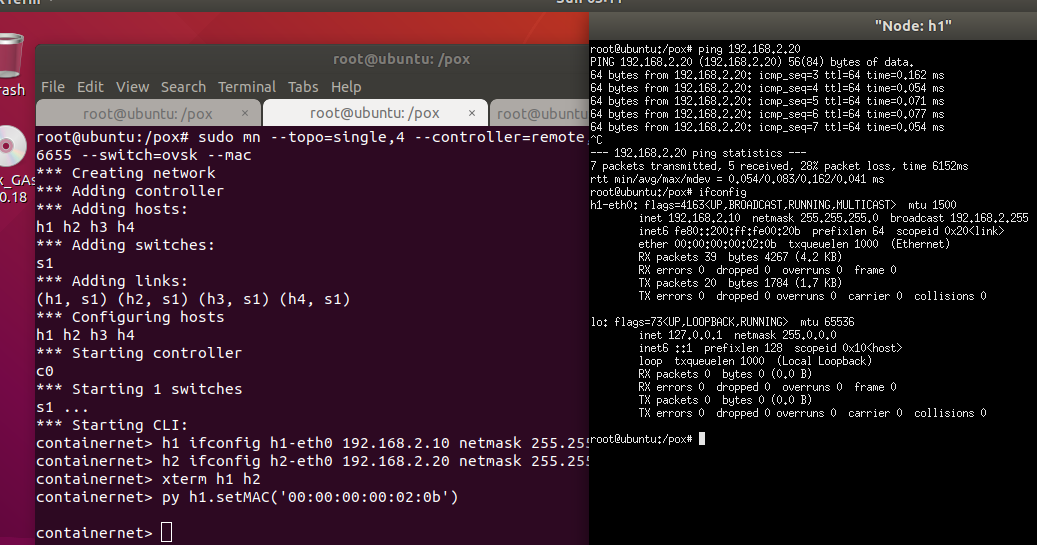


Figure 26-MAC spoofing-Before and after MAC change

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”.

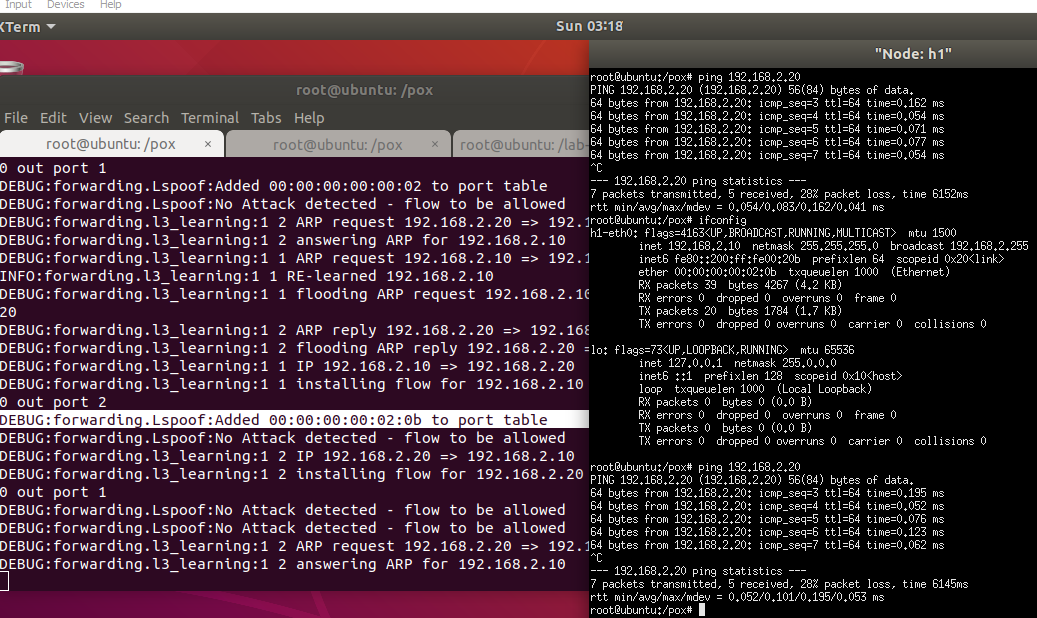


Figure 27 - MAC spoofing - detection after a certain number of packets

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

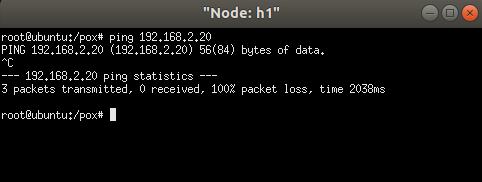


Figure 28-MAC spoofing - xterm showing all dropped ICMP packets

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”.

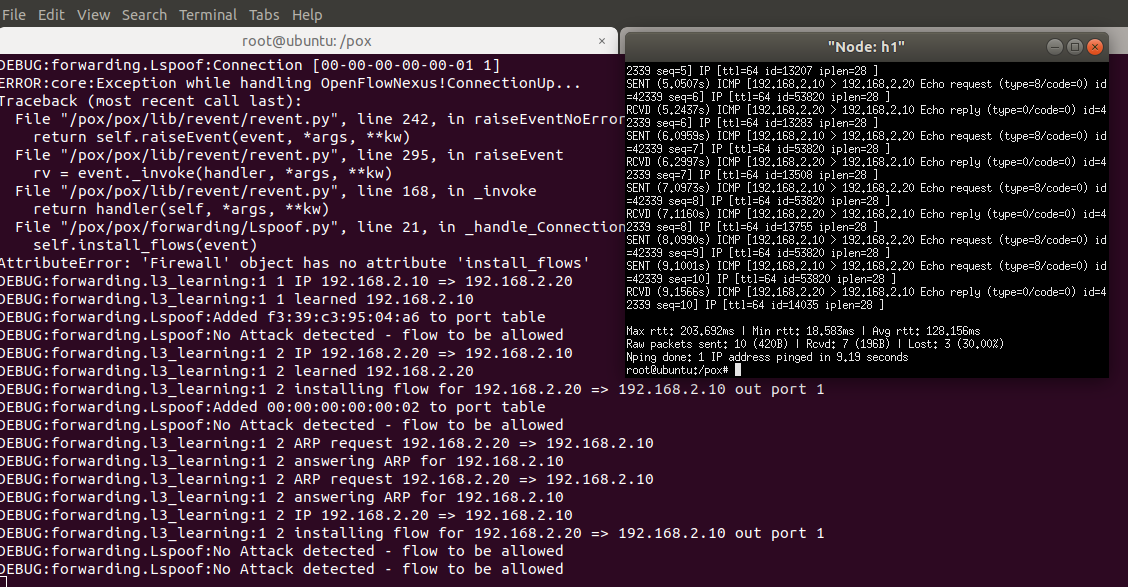


Figure 29-MAC spoofing – nping – first attack

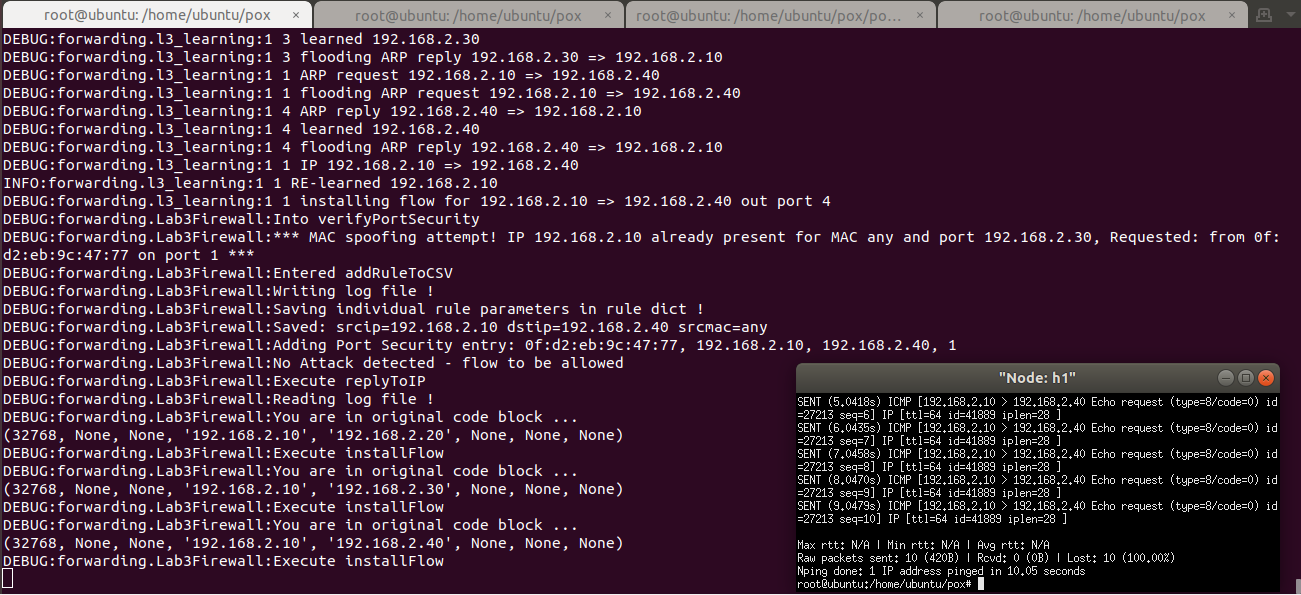


Figure 30 - MAC spoofing – nping – second attack

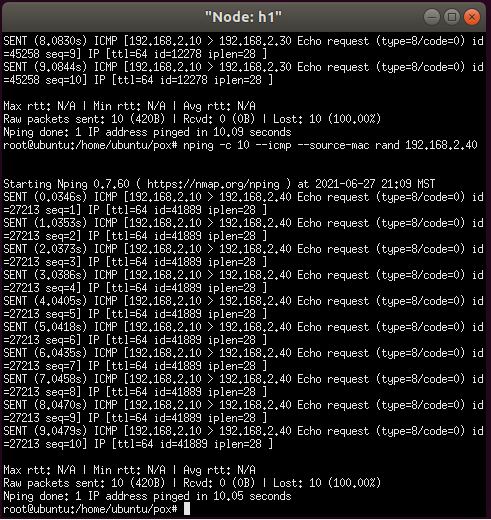


Figure 31 - MAC spoofing – nping – second attack, zoom in

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.

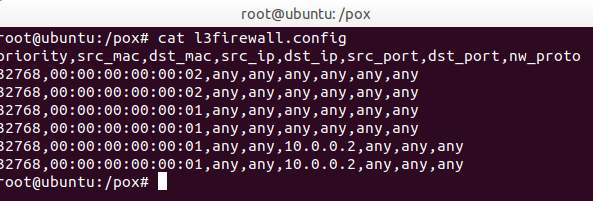


Figure 32-MAC spoofing – flows captured in l3firewall.config

# 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 |  |
| L3firewall.py | https://github.com/nanaama/CSE-548-Advanced-Computer-Network-Security/blob/main/L3firewall.py |
|  |  |
| l3firewall.config | https://github.com/nanaama/CSE-548-Advanced-Computer-Network-Security/blob/main/l3firewall.config |
| l2firewall.config | https://github.com/nanaama/CSE-548-Advanced-Computer-Network-Security/blob/main/l2firewall.config |
| lab.sh | https://github.com/nanaama/CSE-548-Advanced-Computer-Network-Security/blob/main/lab.sh |
| lab2.sh | https://github.com/nanaama/CSE-548-Advanced-Computer-Network-Security/blob/main/lab2.sh |

# 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

[Figure 1 - Bridged network setup in VirtualBox 1](#_Toc75762346)

[Figure 2 – Running mininet 2](#_Toc75762347)

[Figure 3 - Mininet network **Error! Bookmark not defined.**](#_Toc75762348)

[Figure 4 - Setting IP addresses for the lab **Error! Bookmark not defined.**](#_Toc75762349)

[Figure 5 - Checking IP addresses for the containers in the lab **Error! Bookmark not defined.**](#_Toc75762350)

[Figure 6 - Running POX **Error! Bookmark not defined.**](#_Toc75762351)

[Figure 7 - Check openflow flow-entries on switch 1 **Error! Bookmark not defined.**](#_Toc75762352)

[Figure 8 - Start flooding from any container host to container host #2 **Error! Bookmark not defined.**](#_Toc75762353)

[Figure 9d. - OVS Slowly resuming operation **Error! Bookmark not defined.**](#_Toc75762354)

[Figure 10 - Counting the flows on OVS **Error! Bookmark not defined.**](#_Toc75762355)

[Figure 11 - Function "verifyPortSecurity" **Error! Bookmark not defined.**](#_Toc75762356)

[Figure 12 - IP Spoofing - Running POX **Error! Bookmark not defined.**](#_Toc75762357)

[Figure 13 - IP Spoofing - Running mininet **Error! Bookmark not defined.**](#_Toc75762358)

[Figure 14 - IP Spoofing - h1 and h2 with their starting IPs and MAC addresses 17](#_Toc75762359)

[Figure 15 - IP Spoofing - Running hping3 on h2 and running ping on h1 17](#_Toc75762360)

[Figure 16 - IP Spoofing - debug messages showing malicious packets being blocked **Error! Bookmark not defined.**](#_Toc75762361)

[Figure 17 - MAC spoofing - initial setup **Error! Bookmark not defined.**](#_Toc75762362)

[Figure 18 - MAC spoofing - changing MAC manually 19](#_Toc75762363)

[Figure 19 - MAC spoofing - confirmation of MAC change 20](#_Toc75762364)

[Figure 20 - MAC spoofing - Before and after MAC change **Error! Bookmark not defined.**](#_Toc75762365)

[Figure 21 - MAC spoofing - detection after a certain number of packets 21](#_Toc75762366)

[Figure 22 - MAC spoofing - xterm showing all dropped ICMP packets **Error! Bookmark not defined.**](#_Toc75762367)

[Figure 23 - MAC spoofing – nping – first attack **Error! Bookmark not defined.**](#_Toc75762368)

[Figure 24 - MAC spoofing – nping – second attack 23](#_Toc75762369)

[Figure 25 - MAC spoofing – nping – second attack, zoom in 23](#_Toc75762370)

[Figure 26 – MAC spoofing – flows captured in l3firewall.config **Error! Bookmark not defined.**](#_Toc75762371)

Contents

[I. Project Overview 1](#_Toc75762393)

[II. Network Setup 1](#_Toc75762394)

[III. Software 1](#_Toc75762395)

[IV. Project Description 1](#_Toc75762396)

[1. Setting up mininet and Running mininet topology 2](#_Toc75762397)

[2. Should assign IP addresses to hosts. **Error! Bookmark not defined.**](#_Toc75762398)

[3. Perform Flood attack on SDN controller following a suggested procedure: 10](#_Toc75762399)

[a. Run l3 learning application in POX controller. 10](#_Toc75762400)

[b. Check openflow flow-entries on switch 1. 10](#_Toc75762401)

[c. Start flooding from any container host to container host #2 10](#_Toc75762402)

[d. Check Openflow flow entries at switch 1 10](#_Toc75762403)

[4. Mitigate DoS attack by implementing port security and using OpenFlow based firewall. 12](#_Toc75762404)

[a. You should illustrate (through screenshots and descriptions) your implemented program codes. 12](#_Toc75762405)

[b. You should demo how your implementation can mitigate the DoS through a sequence of screenshots with explanation. 12](#_Toc75762406)

[c. You should submit the source codes of your implementation. 12](#_Toc75762407)

[d. IP Spoofing Attack Mitigation 16](#_Toc75762408)

[a. MAC Spoofing Attack Mitigation 19](#_Toc75762409)

[V. Appendix A: Files for the Lab 25](#_Toc75762410)

[VI. File Content **Error! Bookmark not defined.**](#_Toc75762411)

[VII. References 25](#_Toc75762412)

[VIII. Table of Figures 25](#_Toc75762413)