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

Project Report 2 – DOS attack on SDN controller

# 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.

# 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.

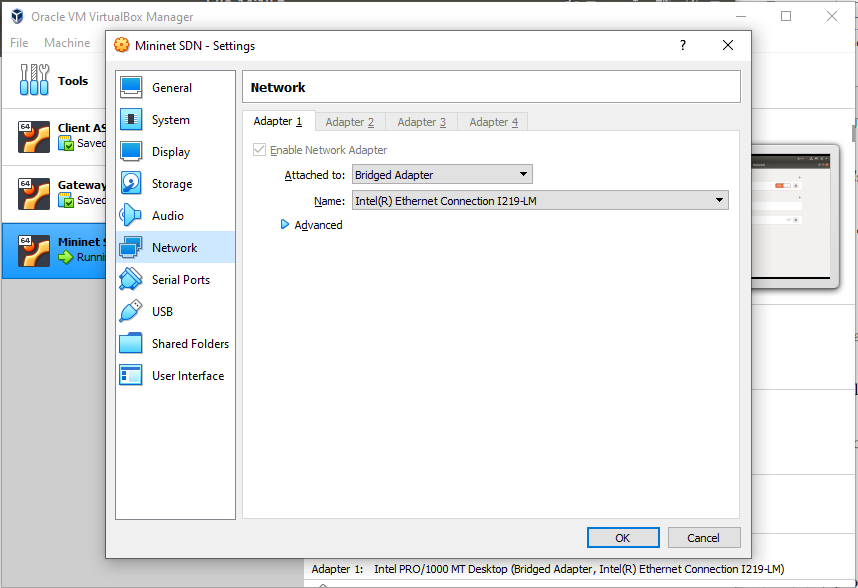


Figure 1 - Bridged network setup in VirtualBox

# 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>

# Project Description

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

.

## 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:

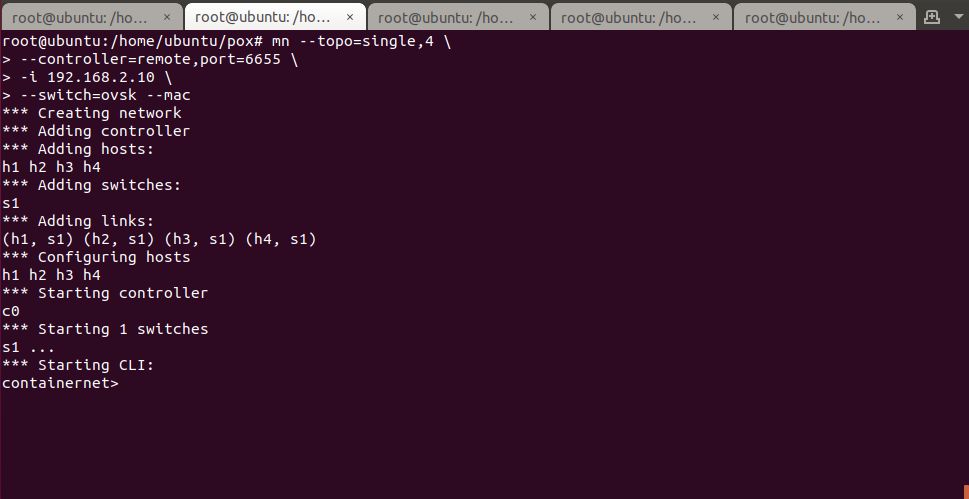
mn --topo=single,4 \

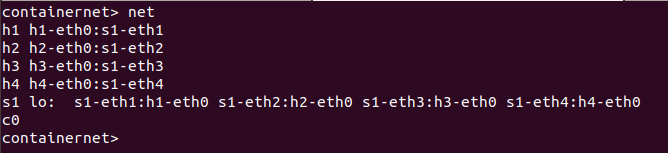
--controller=remote,port=6655 \

-i 192.168.2.10 \

--switch=ovsk --mac

mn -c





## Should assign IP addresses to hosts.

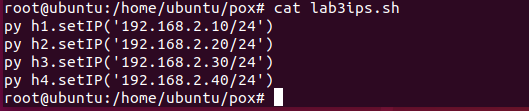
I have executed this again using a very simple source script called “lab3ips.sh” and executed from the mininet CLI with the command “*source lab3ips.sh*”. The script is referenced in GitHub and reproduced here:

py h1.setIP('192.168.2.10/24')

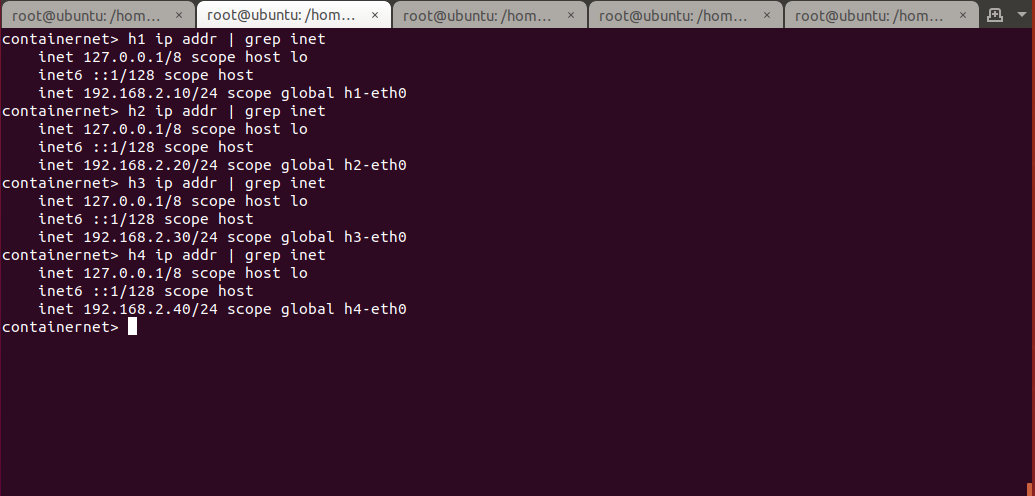
py h2.setIP('192.168.2.20/24')

py h3.setIP('192.168.2.30/24')

py h4.setIP('192.168.2.40/24')



Screenshot showing the IP addresses changed:



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

### 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 \

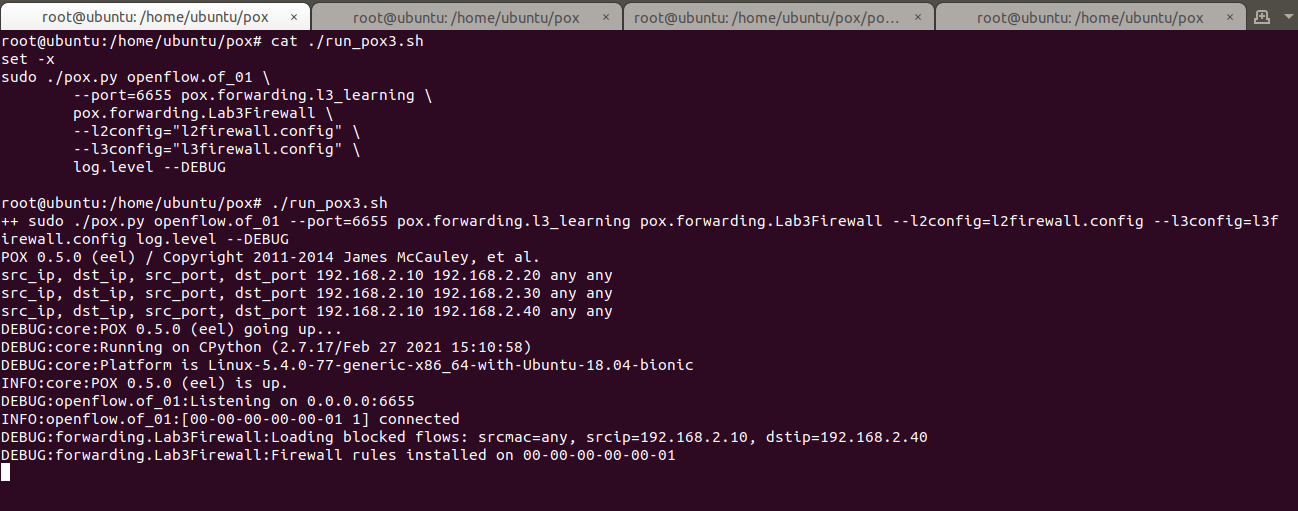
--port=6655 pox.forwarding.l3\_learning \

pox.forwarding.Lab3Firewall \

--l2config="l2firewall.config" \

--l3config="l3firewall.config" \

log.level –DEBUG



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

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

### 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.

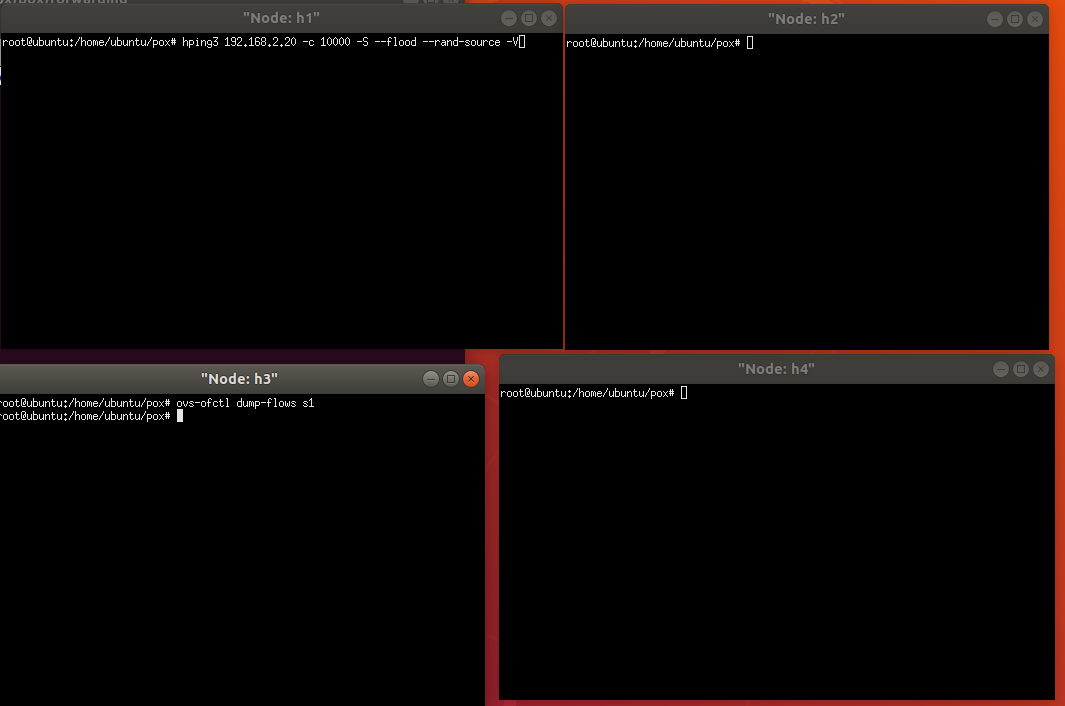


Figure 2 - Check openflow flow-entries on switch 1

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.

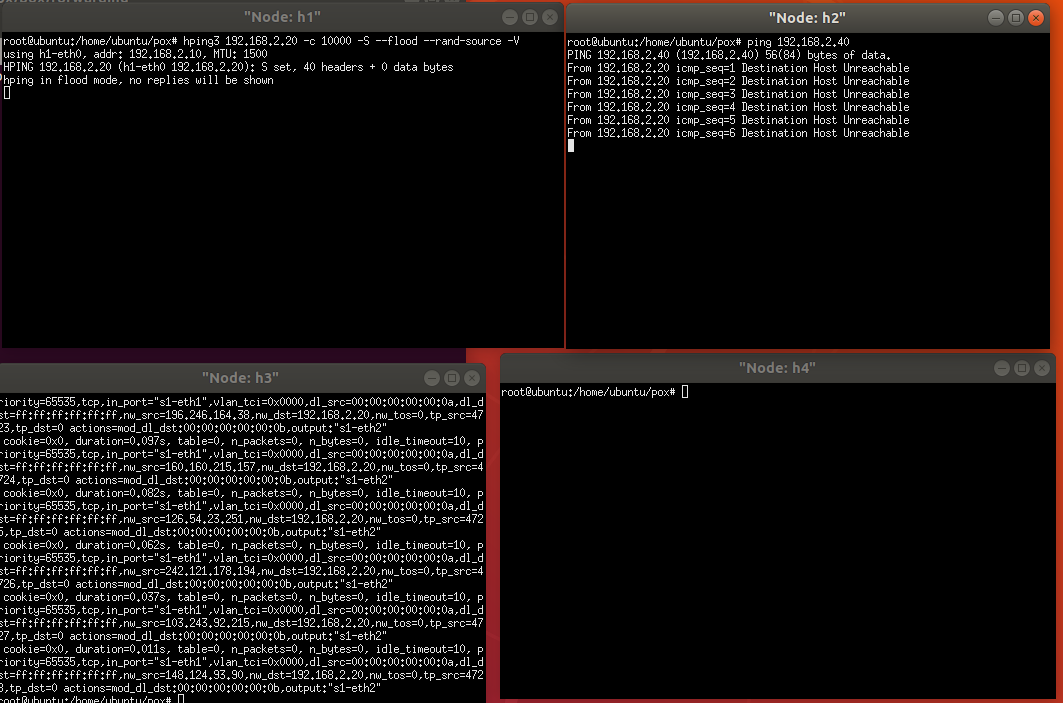


Figure 3 - Start flooding from any container host to container host #2

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.

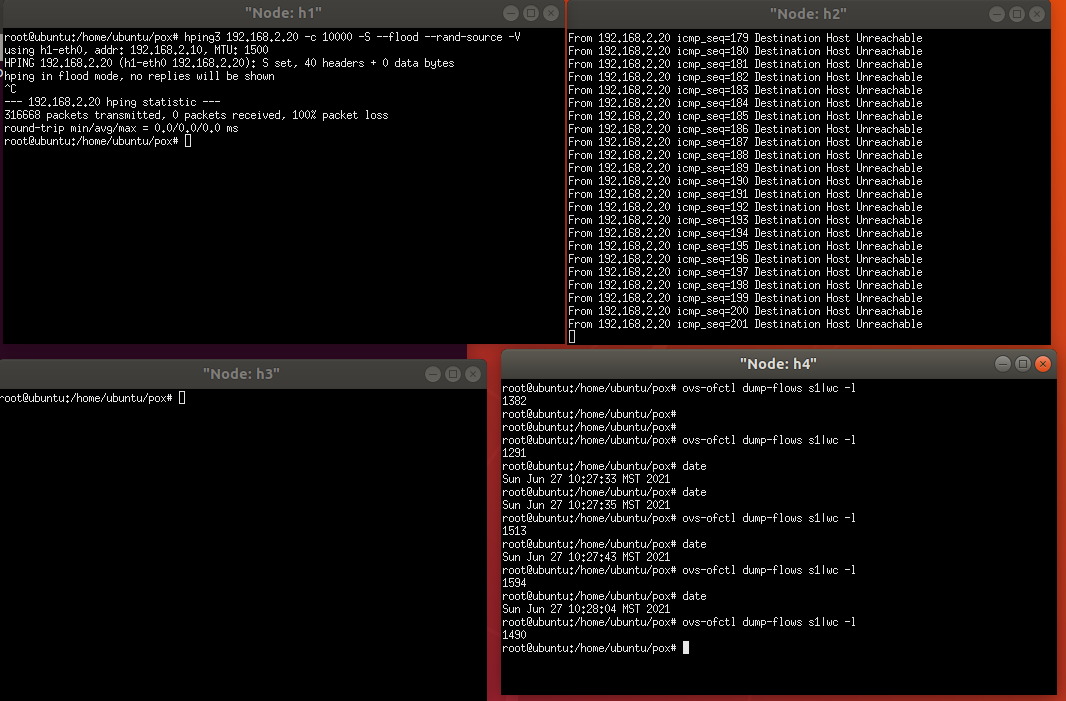


Figure 4d. - OVS Slowly resuming operation

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.

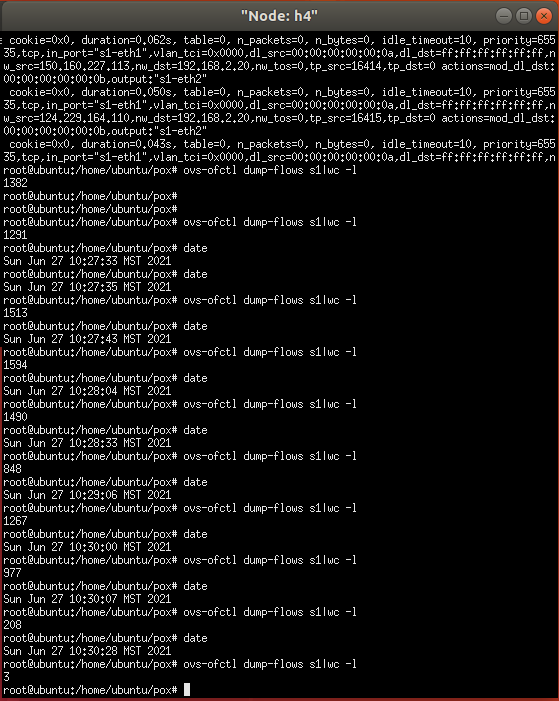


Figure 5 - Counting the flows on OVS

## 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 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





Figure 6 - Function "verifyPortSecurity"

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.

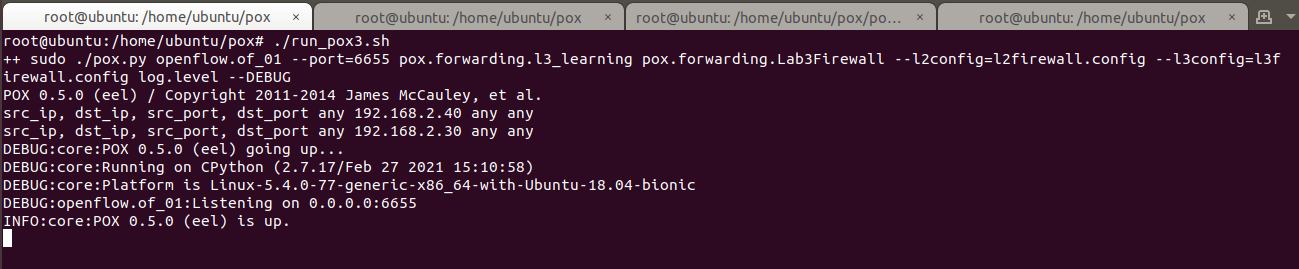


Figure 7 - IP Spoofing - Running POX

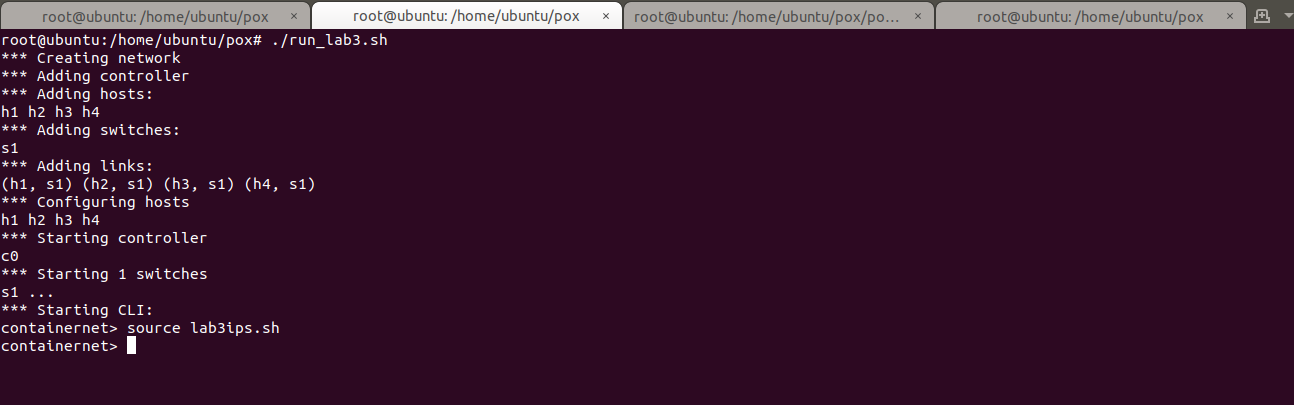


Figure 8 - IP Spoofing - Running mininet

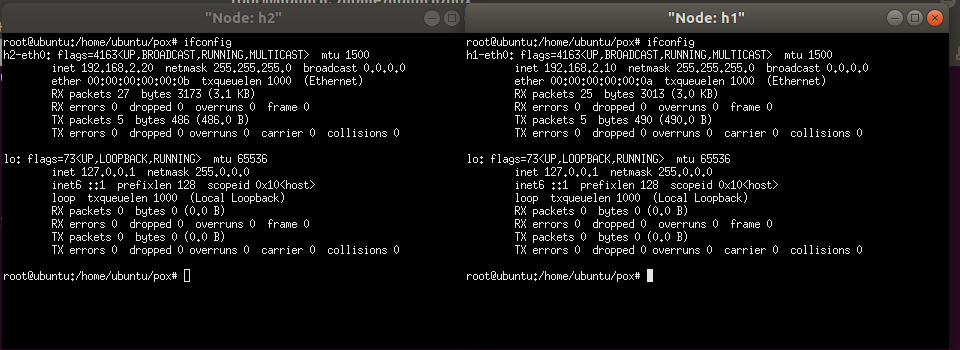


Figure 9 - IP Spoofing - h1 and h2 with their starting IPs and MAC addresses

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.

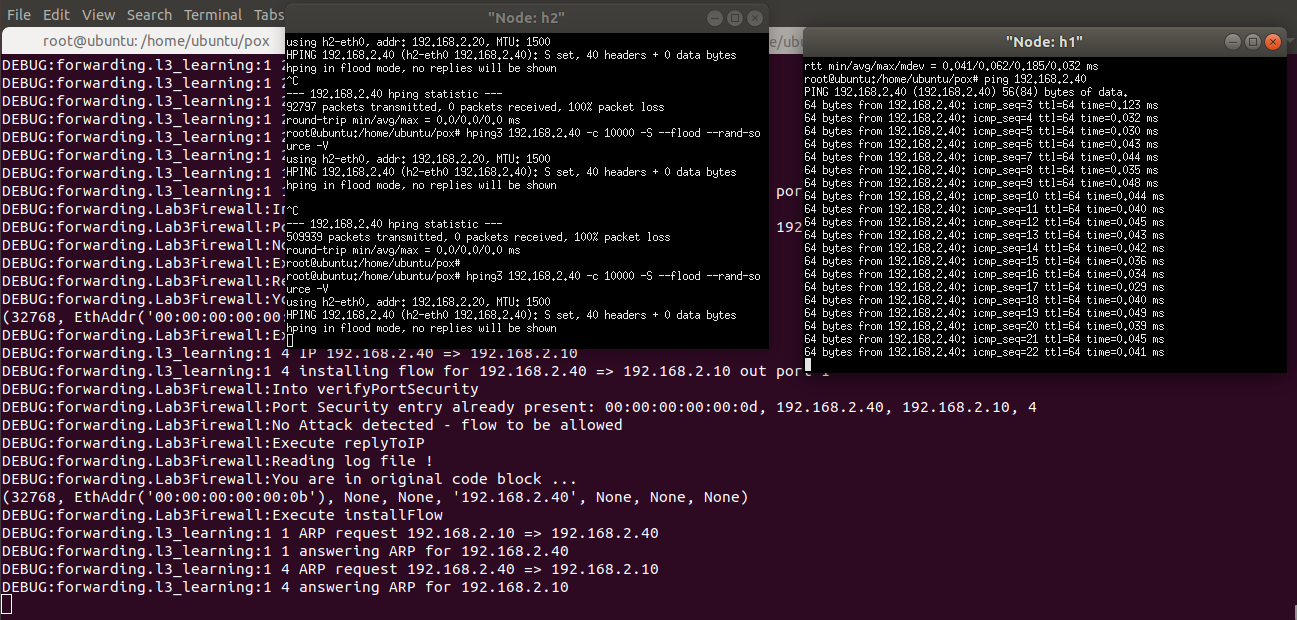


Figure 10 - IP Spoofing - Running hping3 on h2 and running ping on h1

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.

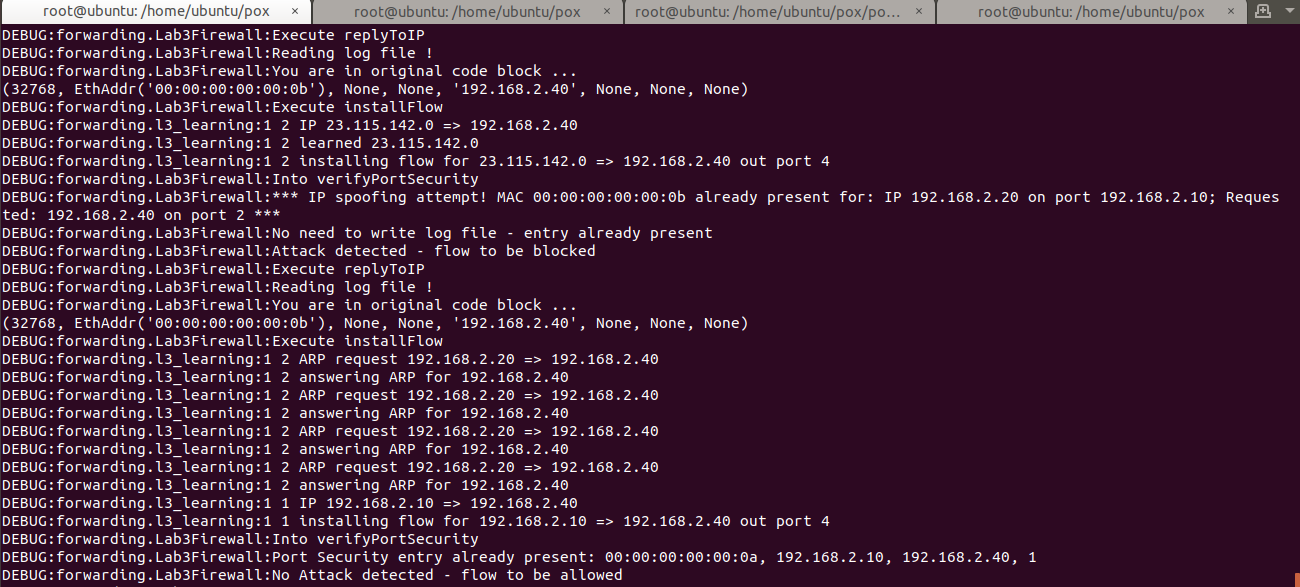


Figure 11 - IP Spoofing - debug messages showing malicious packets being blocked

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 ...

### 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.

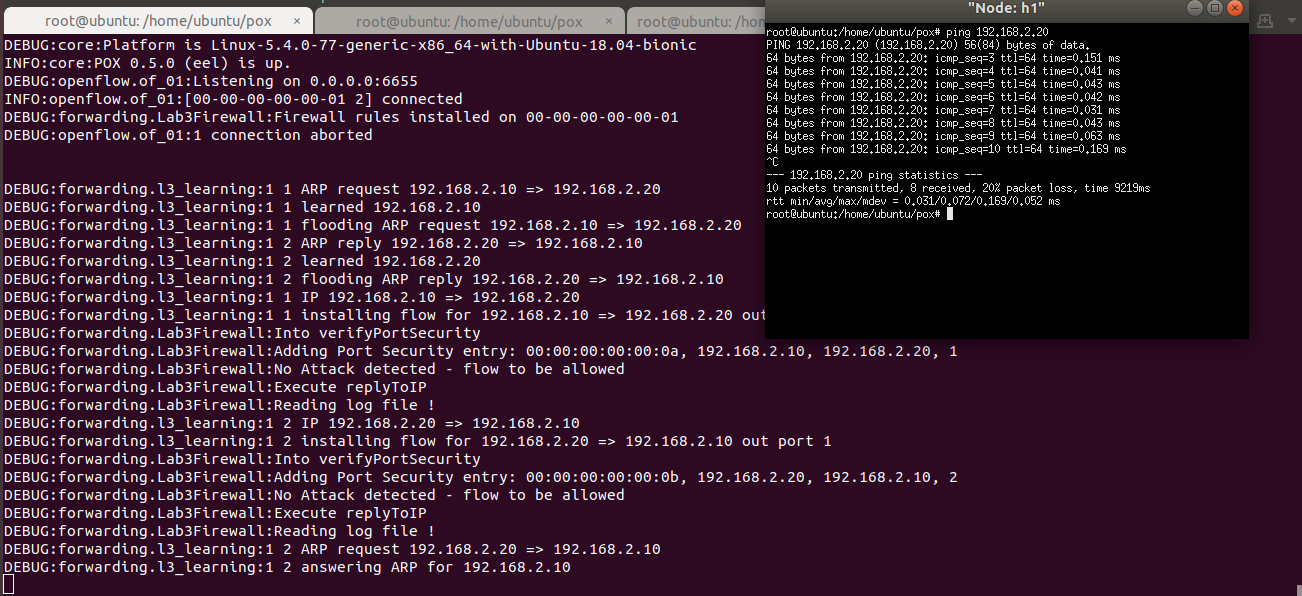
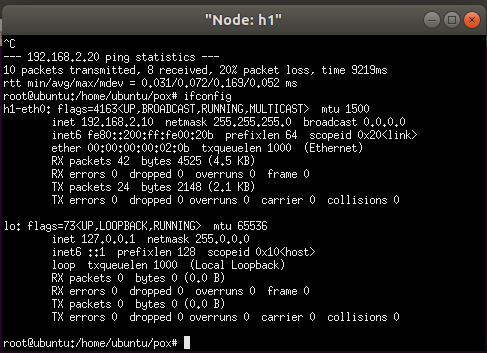


Figure 12 - MAC spoofing - initial setup

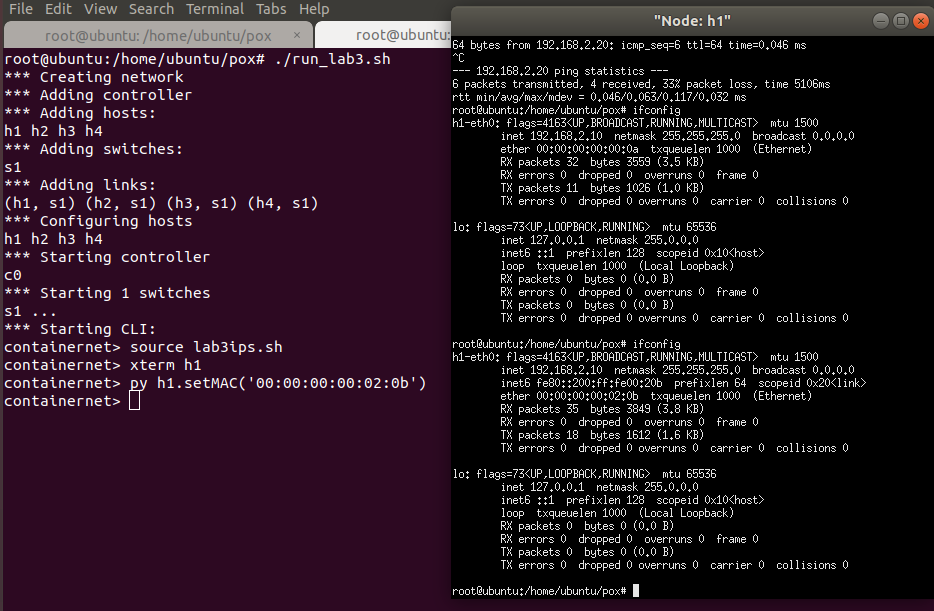


From here, I manually change the MAC address of the contained with a Python-one-liner using the mininet API. The command is “*py h1.setMAC('00:00:00:00:02:0b')*”

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

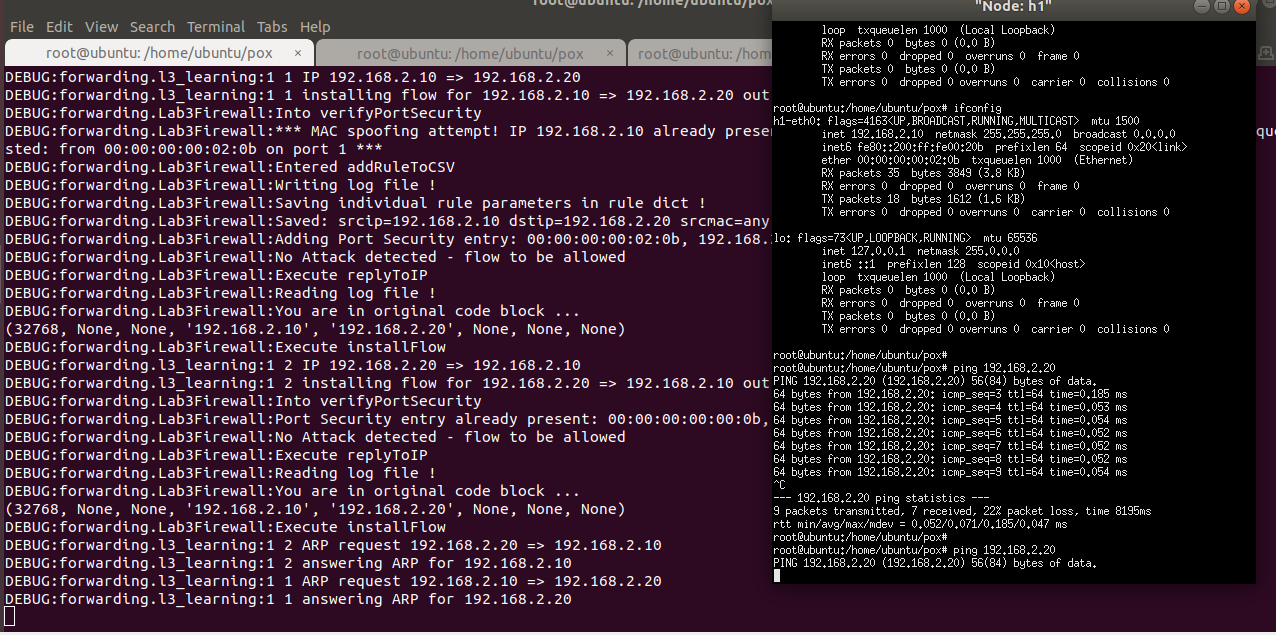


From the next screenshot the whole picture is presented.

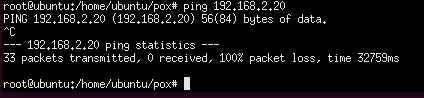


After the MAC address has changed, the networking takes a bit of time to “reconverge” – in my system, it has taken 9 echo-request cycles, but it will vary depending on the case. Basically, the switch needs time to re-adjust to the fact that the topology has changed, and only after this has been reflected 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”.



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



# 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-2 SDN-Based Stateless Firewall.docx | <https://github.com/markoer73/CSE-548/blob/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall/Project-Report-2%20SDN-Based%20Stateless%20Firewall.docx> |
| l2firewall.config | <https://github.com/markoer73/CSE-548/blob/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall/l2firewall.config> |
| l3firewall.config | <https://github.com/markoer73/CSE-548/blob/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall/l3firewall.config> |
| run\_lab.sh | <https://github.com/markoer73/CSE-548/blob/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall/run_lab.sh> |
| run\_pox.sh | <https://github.com/markoer73/CSE-548/blob/main/Project%202%20-%20SDN-Based%20Stateless%20Firewall/run_pox.sh> |

## File Content

### l3firewall.config

priority,src\_mac,dst\_mac,src\_ip,dst\_ip,src\_port,dst\_port,nw\_proto

1,any,any,192.168.2.10,192.168.2.30,1,1,icmp

2,any,any,192.168.2.20,192.168.2.40,1,1,icmp

3,any,any,192.168.2.20,any,1,80,tcp

4,any,any,192.168.2.10,192.168.2.20,1,1,tcp

5,any,any,192.168.2.10,192.168.2.20,1,1,udp

### l2firewall.config

id,mac\_0,mac\_1

1,00:00:00:00:00:02,00:00:00:00:00:04

### run\_pox.sh

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

--port=6655 pox.forwarding.l2\_learning \

pox.forwarding.L3Firewall --l2config="l2firewall.config" \

--l3config="l3firewall.config" &

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

--port=6633 pox.forwarding.l2\_learning \

pox.forwarding.L3Firewall --l2config="l2firewall.config" \

--l3config="l3firewall.config" &

### run\_lab.sh

mn --topo=single,4 \

--controller=remote,port=6633 \

--controller=remote,port=6655 \

--switch=ovsk --mac

# References

* Linux NAT Tutorial: <https://www.karlrupp.net/en/computer/nat_tutorial>
* Ubuntu “Basic Iptables HOWTO”: <https://help.ubuntu.com/community/IptablesHowTo>
* “Iptables Tutorial: Ultimate Guide to Linux Firewall”: <https://phoenixnap.com/kb/iptables-tutorial-linux-firewall>

VIII. Table of Figures

[Figure 1 - Bridged network setup in VirtualBox 1](#_Toc74549301)

[Figure 2 - On the created bridge br0, dhclient is run 3](#_Toc74549302)

[Figure 3 - The bridge inherits the same IP of the enp0s3 interface 3](#_Toc74549303)

[Figure 4 - Pinging the outside world 3](#_Toc74549304)

[Figure 5 - lab-cs-net-00006 - Lab 1-1 4](#_Toc74549305)

[Figure 6 - lab-cs-net-00006 - Lab 1-2 4](#_Toc74549306)

[Figure 7- lab-cs-net-00006 - Lab 1-3/1 4](#_Toc74549307)

[Figure 8 - lab-cs-net-00006 - Lab 1-3/2 5](#_Toc74549308)

[Figure 9 - lab-cs-net-00006 - Lab 2-1 5](#_Toc74549309)

[Figure 10 - Ping outside 5](#_Toc74549310)

[Figure 11 - lab-cs-net-0007 - Lab 1-1 6](#_Toc74549311)

[Figure 12- lab-cs-net-0007 - Lab 1-2 6](#_Toc74549312)

[Figure 13 - lab-cs-net-0007 - Lab 2-1 – topology 1 6](#_Toc74549313)

[Figure 14 - lab-cs-net-0007 - Lab 2-1 – topology 2 7](#_Toc74549314)

[Figure 15- lab-cs-net-0007 - Lab 2-2 - Topology 1 7](#_Toc74549315)

[Figure 16- lab-cs-net-0007 - Lab 2-2 - Topology 2 7](#_Toc74549316)

[Figure 17- lab-cs-net-0007 - Lab 2-3 - Topology 1 8](#_Toc74549317)

[Figure 18- lab-cs-net-0007 - Lab 2-3 - Topology 2 8](#_Toc74549318)

[Figure 19- lab-cs-net-0007 - Lab 3-1 9](#_Toc74549319)

[Figure 20- lab-cs-net-0007 - Lab 3-2 9](#_Toc74549320)

[Figure 21 - lab-cs-net-0087 - Lab 1-1/1 10](#_Toc74549321)

[Figure 22 - lab-cs-net-0087 - Lab 1-1/2 10](#_Toc74549322)

[Figure 23 - lab-cs-net-0087 - Lab 1-1/3 10](#_Toc74549323)

[Figure 24 - lab-cs-net-0087 - Lab 1-2/1 10](#_Toc74549324)

[Figure 25 - lab-cs-net-0087 - Lab 1-2/2 11](#_Toc74549325)

[Figure 26 - lab-cs-net-0087 - Lab 2-1 11](#_Toc74549326)

[Figure 27 - lab-cs-net-0087 - Lab 2-2 12](#_Toc74549327)

[Figure 28 - - lab-cs-net-0087 - Extra Activity 12](#_Toc74549328)

[Figure 29 - lab-cs-net-0009 – Lab 1-1 13](#_Toc74549329)

[Figure 30 - lab-cs-net-0009 – Lab 1-2 13](#_Toc74549330)

[Figure 31 - lab-cs-net-0009 – Lab 1-3 14](#_Toc74549331)

[Figure 32 - CS-CNS-00101 – run\_pox.sh 15](#_Toc74549332)

[Figure 33 - CS-CNS-00101 – run\_pox.sh running 15](#_Toc74549333)

[Figure 34 - CS-CNS-00101 – run\_lab.sh 16](#_Toc74549334)

[Figure 35 - CS-CNS-00101 – run\_lab.sh running 16](#_Toc74549335)

[Figure 36 - CS-CNS-00101 – Lab 1 16](#_Toc74549336)

[Figure 37 - CS-CNS-00101 – Change IP addresses 17](#_Toc74549337)

[Figure 38 - CS-CNS-00101 – Lab 2 17](#_Toc74549338)

[Figure 39 - CS-CNS-00101 – l3firewall.config 18](#_Toc74549339)

[Figure 40 - CS-CNS-00101 – l2firewall.config 18](#_Toc74549340)

[Figure 41 - CS-CNS-00101 – checking nohup.out 18](#_Toc74549341)

[Figure 42 - CS-CNS-00101 – Changing IP addresses again 18](#_Toc74549342)

[Figure 43 - CS-CNS-00101 – first ping from h1 to h3 gets through, then the rest are blocked 19](#_Toc74549343)

[Figure 44 - CS-CNS-00101 – pings blocked from h2 to h4 19](#_Toc74549344)

[Figure 45 - CS-CNS-00101 – h4 can reach web server on h3 20](#_Toc74549345)

[Figure 46 - CS-CNS-00101 – h2 cannot reach web server on h3 20](#_Toc74549346)

[Figure 47 - CS-CNS-00101 – h1 can reach web server on h3 21](#_Toc74549347)

[Figure 48 - CS-CNS-00101 – h2 ping and UDP scan against h4 fail 21](#_Toc74549348)

[Figure 49 - CS-CNS-00101 – h2 TCP scan against h4 fails 22](#_Toc74549349)

[Figure 50 - CS-CNS-00101 – h2 TCP scan against h4 fails 22](#_Toc74549350)

[Figure 51 - CS-CNS-00101 – ovs-ofctl dump-flows s1 22](#_Toc74549351)

Contents

[I. Project Overview 1](#_Toc74549358)

[II. Network Setup 1](#_Toc74549359)

[III. Software 2](#_Toc74549360)

[IV. Project Description 2](#_Toc74549361)

[A. Lab “lab-cs-net-00006 OpenVirtual Switch” 3](#_Toc74549362)

[1) Lab Assessment 1: The OVS… 4](#_Toc74549363)

[2) Lab Assessment 2: After finishing the configuration of br0 in OVS, the machine… 5](#_Toc74549364)

[B. Lab “lab-cs-net-00007 - Mininet” 6](#_Toc74549365)

[1) Setting up mininet and Running mininet topology… 6](#_Toc74549366)

[2) For each topology above, they should be able to: 6](#_Toc74549367)

[3) Create another tree topology of depth 2 and fanout 8. 8](#_Toc74549368)

[C. Lab “CS-NET-00008 – SDN Controller (POX)” 10](#_Toc74549369)

[1) The POX controller can: 10](#_Toc74549370)

[2) While connect to the controller, the mininet topology can (Please take snapshots for each task) 11](#_Toc74549371)

[3) Extra Activity 12](#_Toc74549372)

[D. Lab “CS-NET-00009 – Containernet” 13](#_Toc74549373)

[1) The containernet topology can: 13](#_Toc74549374)

[E. Lab “CS-CNS-00101” – OpenFlow Based Stateless firewall 15](#_Toc74549375)

[1) Create a mininet based topology with 4 container hosts and one controller switches and run it. 15](#_Toc74549376)

[2) Make the interfaces up and assign IP addresses to interfaces of container hosts. 17](#_Toc74549377)

[3) Add new rule to l3config file for blocking ICMP traffic from source IP 192.168.2.10 and destination IP 192.168.2.30. 17](#_Toc74549378)

[4) Add new rule to l3config file for blocking ICMP traffic from source IP 192.168.2.20 and destination IP 192.168.2.40. 17](#_Toc74549379)

[5) Add new rule to l3config file for blocking HTTP traffic from source IP 192.168.2.20. 17](#_Toc74549380)

[6) Add new rule to l2config file for blocking traffic from MAC address 00:00:00:00:00:02 to destination MAC address 00:00:00:00:00:04. 17](#_Toc74549381)

[7) Add new rule to l3config file for blocking tcp traffic from 192.168.2.10 to 192.168.2.20. 17](#_Toc74549382)

[8) Add new rule to l3config file for blocking udp traffic from 192.168.2.10 to 192.168.2.20. 17](#_Toc74549383)

[V. Appendix A: Files for the Lab 23](#_Toc74549384)

[A. File Content 23](#_Toc74549385)

[1) l3firewall.config 23](#_Toc74549386)

[2) l2firewall.config 23](#_Toc74549387)

[3) run\_pox.sh 23](#_Toc74549388)

[4) run\_lab.sh 23](#_Toc74549389)

[VI. References 24](#_Toc74549390)

[VIII. Table of Figures 24](#_Toc74549391)