Project Report 3 – SDN-Based DoS Attacks and Mitigation

Student Name: Mehran Tajbakhsh

Email: mtajbakh@asu.edu

Submission Date: 6/25/2022

Class Name and Term: CSE548 Summer 2022

I. Project Overview

In this lab, we wanted to set up and use an SDN-based DoS attack and mitigation by setting up layer 3 firewall rules. In this lab we set up a software defined network based on mininet, containernet, POX controller, and OVS. We created an SDN infrastructure in a VirtualBox VM which had Ubuntu/Linux installed. Then we set up a layer 3 SDN-based firewall to control and filter DoS attack traffic on the target host.

II. Network Setup

Based on the image provided for this lab, I created a VM (Lab3-mininet) on VirtualBox. In this lab we needed to create an SDN-based network based on mininet and containernet. Our network in this lab consists of 4 hosts, one OpenFlow-enabled switch, and one controller.

I used MiniEdit, A GUI editor to set up and test the SDN-based network, to create the network diagram (Figure-1)

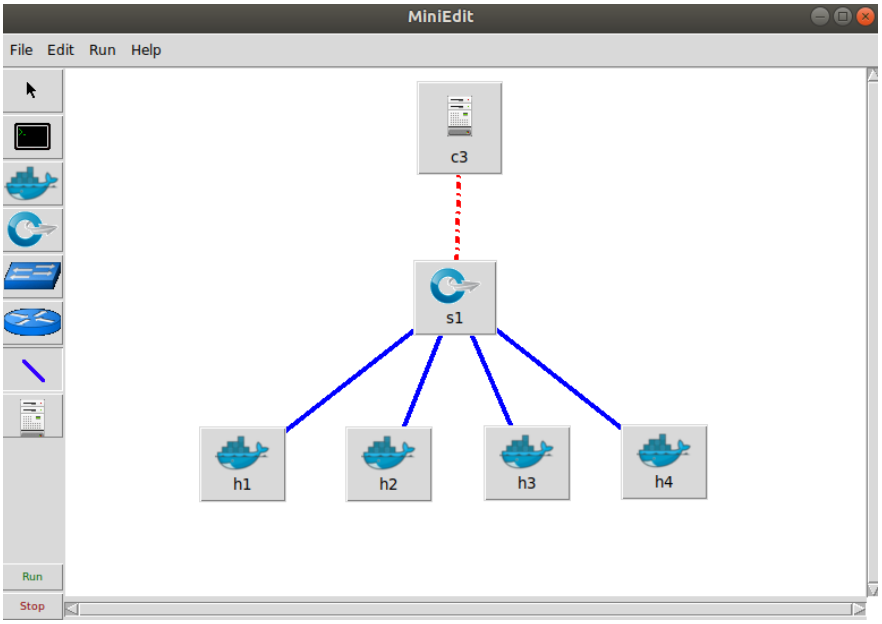

Figure-1 SDN-based network (4-hosts/1-OVS/1-Controller)

III. SOFTWARE

- Network tools (ping, nping, tcpdump, ifconfig, hping3)
- POX – POX is a networking software platform written in Python (https://github.com/noxrepo/pox).
- Open vSwitch (OVS) – Open vSwitch is a production quality, multilayer virtual switch licensed under the open source **Apache 2.0** license (https://www.openvswitch.org/).
- Containernet - Containernet is a fork of the famous Mininet network emulator and allows to use Docker containers as hosts in emulated network topologies. (https://containernet.github.io/).

IV. PROJECT DESCRIPTION

In this lab I implemented an OpenFlow Based Stateless Firewall in an SDN-based virtual network with the mininet and the containernet in a VirtualBox Linux/Ubuntu VM. My network consists of 4 hosts, one OpenFlow based switch (OVS), and One Controller (POX). In the following section you can find all of the steps that I followed in this lab. For every step, I provide a detailed description and screenshot(s), which should make it easy to follow and understand.

1. Setting up mininet and running mininet topology

(a) Create a mininet based topology with 4 container hosts and one controller switches.

- Add link from controller1 to switch 1.
- Add link from controller2 to switch 1.
- Add link from switch 1 to container 1.
- Add link from switch 1 to container 2.
- Add link from switch 1 to container 3.
- Add link from switch 1 to container 4.

In the first step, we needed to create an OpenFlow based controller. Multiple OpenFlow enabled controllers are available, In this lab we used POX (OpenFlow Based controller written in Python). I created a bash file (create_pox.sh) to set up my controller (Figure-2, Figure-3).

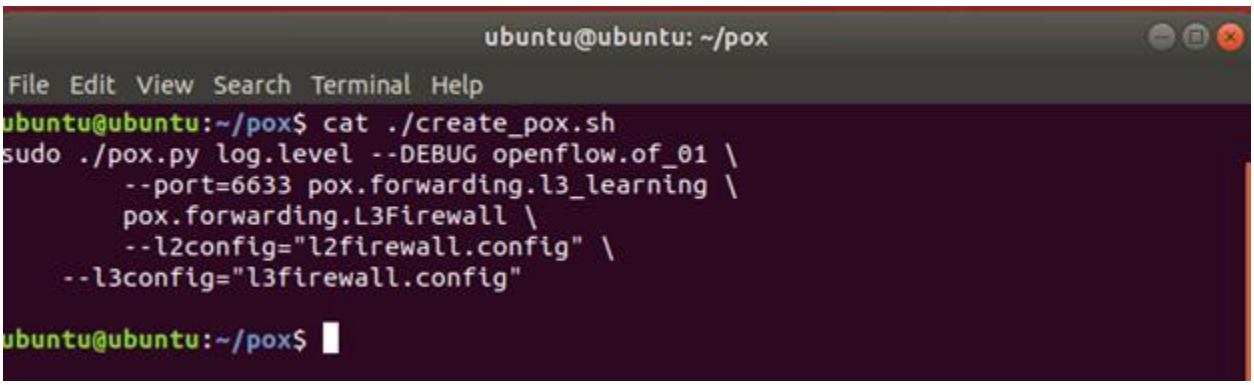

Figure-2 Bash file to set up POX controller

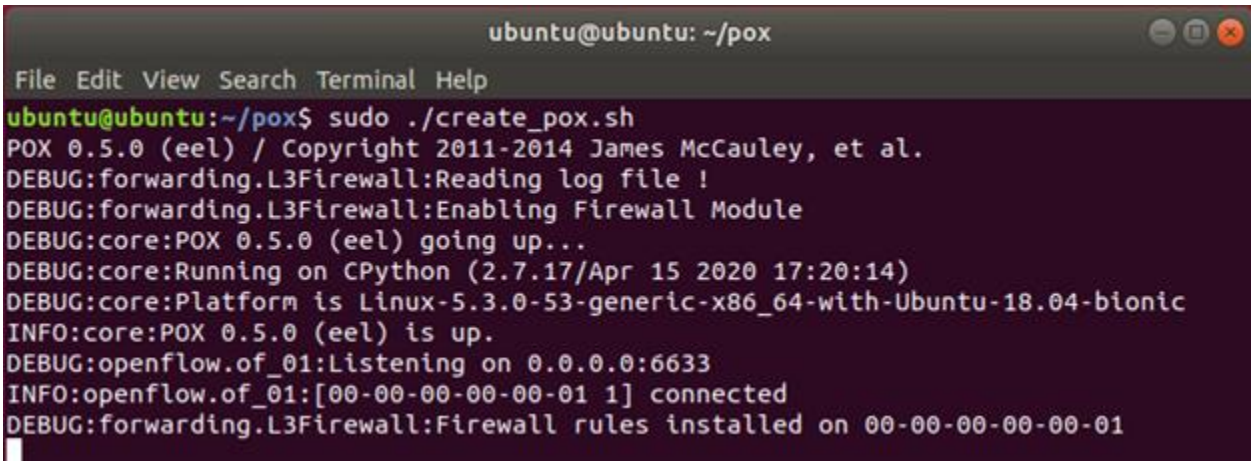

Figure-3 Executing bash file to create POX controller

In this section we needed to define and create our network infrastructure. I created a bash file (create_topo.sh) to create a Mininet virtual network with 4 hosts, and one OpenFlow based vSwitch (OVS) which was connected to our controller (POX) which is shown above at port 6633 (Figure4, Figure 5):

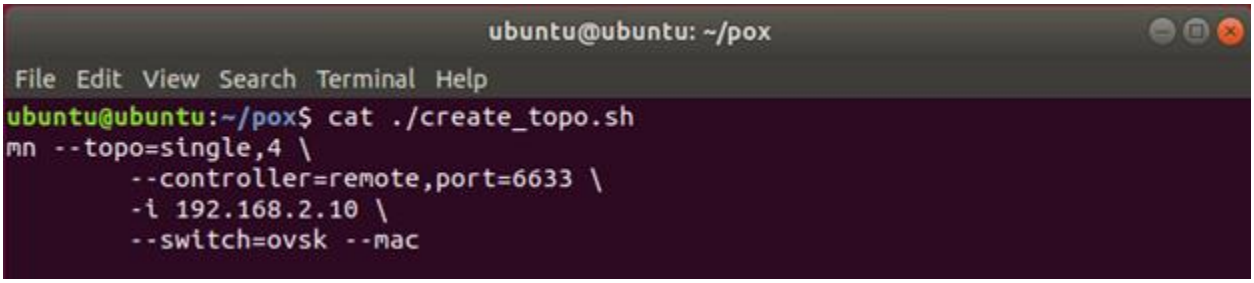

Figure-4 Bash file to create a Mininet virtual network

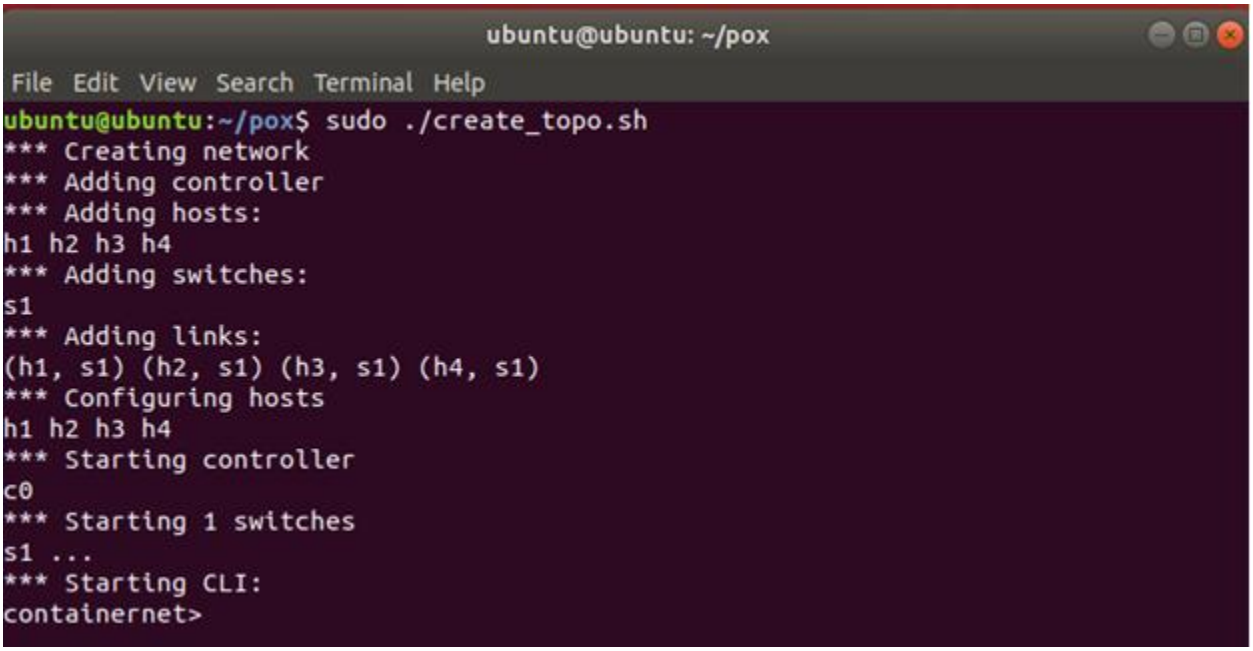

Figure-5 Create a Mininet virtual network by running the bash file

(b) Run the mininet topology. (Create mininet topology using mininet command-line)

Now we can test our network topology and links between the hosts, switch, and controller by running **net** and **nodes** commands in the Mininet terminal window (Figure-6):

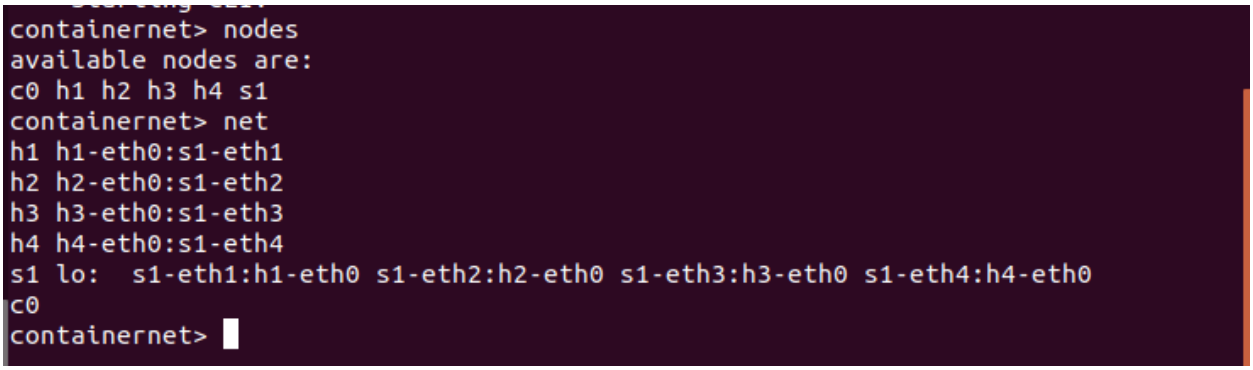

Figure-6 the net and the nodes commands

2. *Assign IP addresses to hosts*

In this section I assigned the above IP addresses to my hosts. For this task I used the ifconfig command in the Mininet command line window (Figure-7):

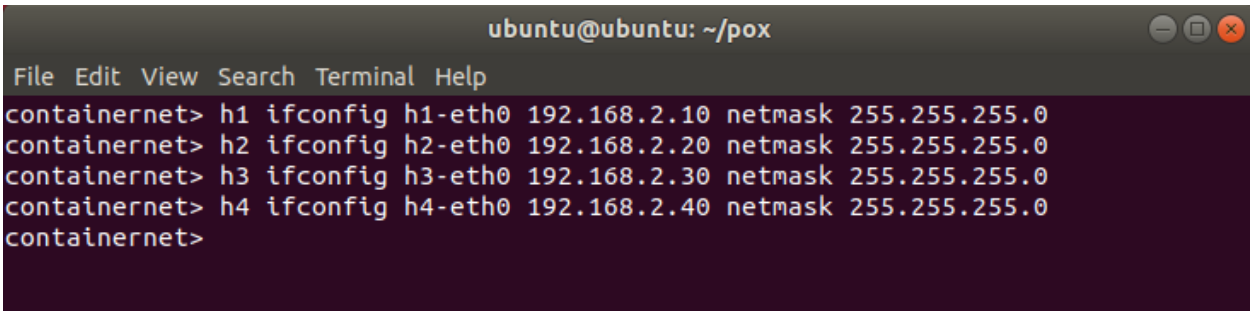

Figure-7 Assign IP addresses to hosts

Then I used the xterm command to open a terminal window for 4 hosts in mininet (Figure-8):

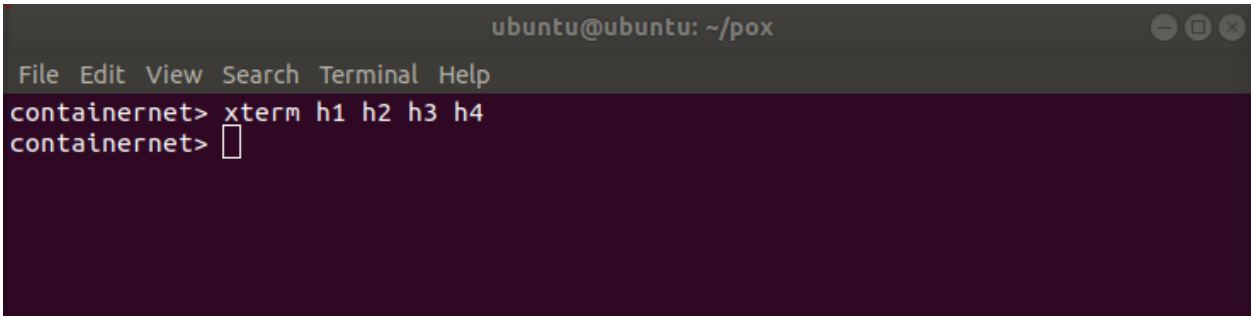

Figure-8 xterm command

In this section I used the ifconfig command in every host's terminal window to see the IP addresses that I assigned to each host above (Figure-9):

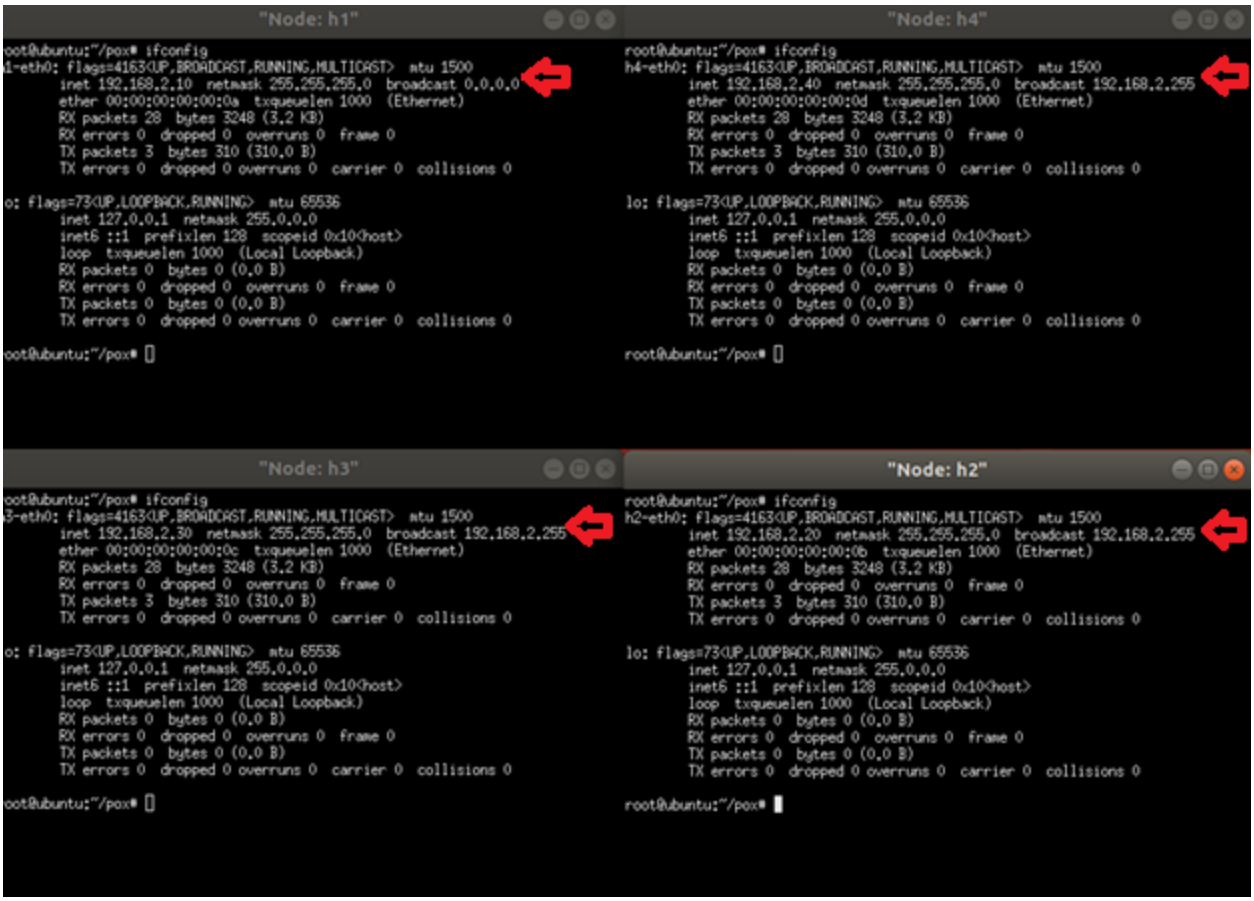

Figure-9 run ifconfig command on each hosts

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

(a) Run l3_learning application in POX controller

(b) Check openflow flow-entries on switch 1.

As shown in the screenshot below, at the beginning we didn't define any layer 3 firewall rules and we didn't have any flow in the OVS switch (Figure-10):

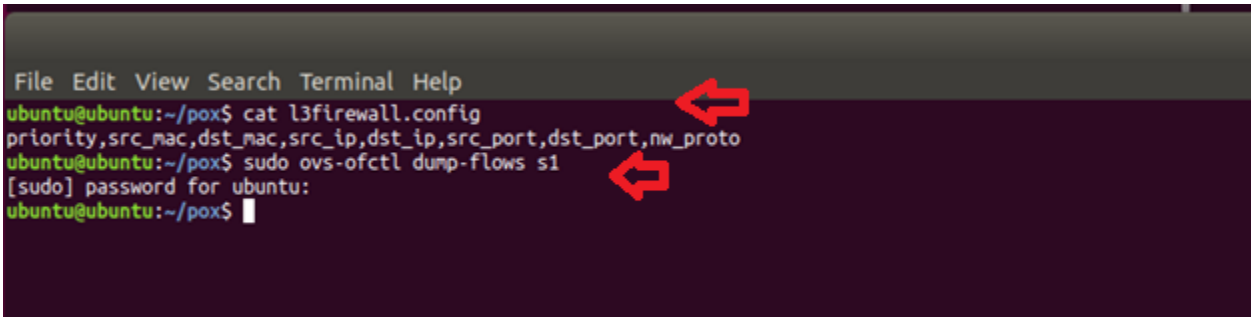

Figure-10 Layer3 firewall rule and OVS switch flow entries

(c) Start flooding from any container host to container host #2.

I used the hping3 command on h1 container host (attacker) to create flooding traffic to h2 container host (target) at IP address 192.168.2.20. The below command sent flooding TCP/SYN traffic with random source IP addresses to h2 container host:

hping3 192.168.2.20 -S --flood -c 10000 --rand-source

Since we didn't define rules to block suspicious traffic on hosts, all traffic was forwarded to the designated targets.

I used the ping command on h4 container host to send ICMP traffic to h2 container host. This command shows that when h2 container host received flooding traffic, it wasn't able to send reply traffic to h4 container host and the ping command displays the message "Unreachable" because the h2 container host was flooded (Figure-11):

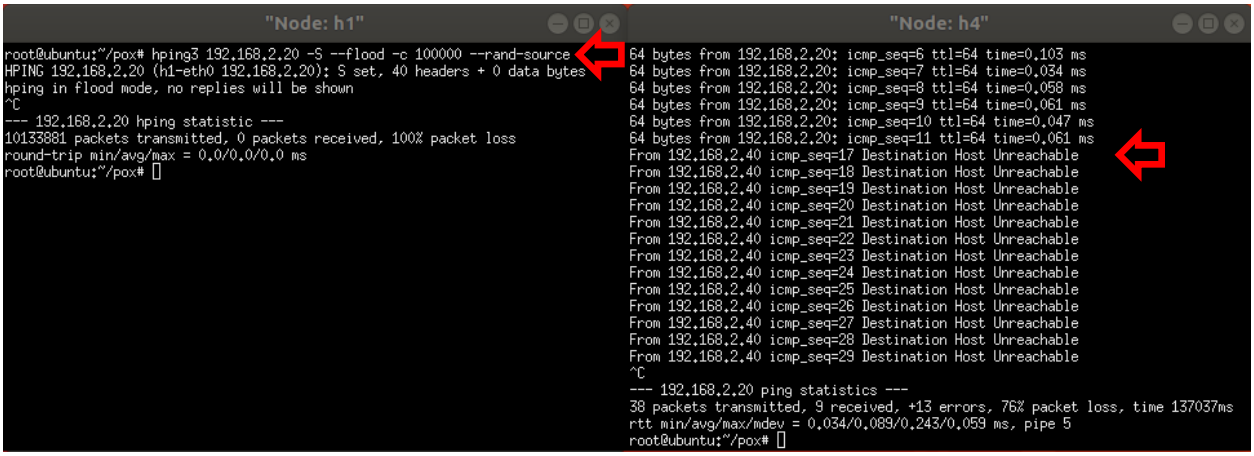

Figure-11 hping3 command on h1 to generate flooding traffic and ping command on h4

(d) Check Openflow flow entries at switch 1

When I checked the Openflow flow-entries in switch s1 (screenshot below), the output confirmed that the switch was flooded by traffic from h1 container host (mac: 00:00:00:00:00:0a/spoofed source IP address) (Figure-12):

sudo ovs-ofctl dump-flows s1

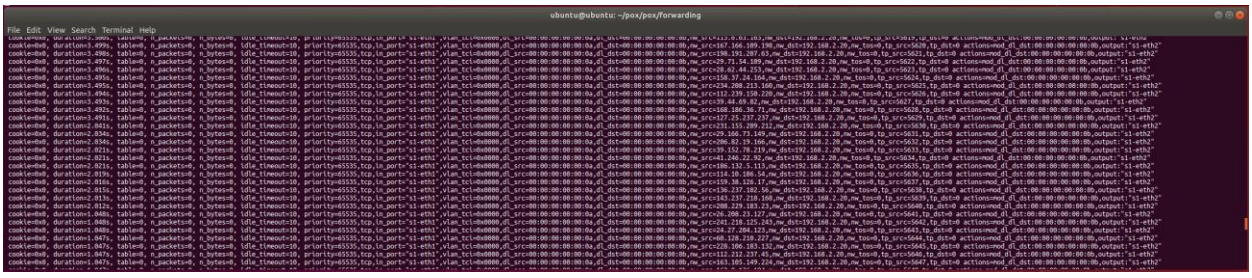

Figure-12 Flow entries on OVS switch s1

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

To satisfy the requirements of this lab, I've modified the L3Firewall.py file to detect and block DoS attack by finding the spoofed MAC/IP addresses in flow traffic. The source code of the L3Firewall.py file is available on GitHub.

I want to describe the logic that I implemented into the L3Firewall.py file to detect and block spoofed MAC/IP addresses flow traffic.

I implemented two array using dict() data structure as bellow (Figure-13):

```
    def __init__ (self,l2config,l3config):
        self.listenTo(core.openflow)
        ## I converted this code into a comment
        #self.disbaled_MAC_pair = [] # Shore a tuple of MAC pair which will be installed into the flow table of each switch.
        #self.fwconfig = list()

        self.Tlogs = dict ()    # Python dictionary to keep logs of the source MAC/IP addressess and destination MAC/IP addresses.
        self.TRules = dict ()  # Python dictionary to keep logs of the new firewall rule to avoid duplications.
```

Figure-13 Tlogs and TRules dictionaries

In this lab I needed to detect DoS traffic from spoofed source MAC or IP addresses. To accomplish this goal I used the following algorithm to detect the source MAC or IP spoofed traffic on every new incomming traffic flow.

In order to find spoofed source MAC or IP addresses, I needed to keep track of all source MAC and IP addresses and save them for further investigations. I also needed to keep the new rules that I created during the lab to avoid duplication. In order to satisfy these requirements, I defined two dictionaries with the following usages:

- Tlogs: To save all source MAC and IP addresses, destination MAC and IP addresses.
- TRules: To save new Firewall rules to avoid duplication.

I implemented a function "TrafficAnalyser" to detect spoofed MAC/IP addresses in the flow traffics. I used the following pseudo-algorithm to implement this function:

First part - detect spoofed source IP addresses

Look for source MAC address in the Tlog dictionary
If source MAC address is found in the Tlogs dictionary,
look for the source IP address in the Tlogs dictionary
if different source IP address with this MAC address is found in the Tlogs dictionary then
***** Spoofed source IP address detected ******
Create a new rule to block it and if this rule hasn't been added to the TRules dictionary, then add it to the TRules dictionary.
Append the new rule into the CSV file and resend Firewall rules to the OVS switch.

Second part - detect spoofed source MAC addresses - BONUS POINTS

else look for the source IP address in the Tlogs dictionary
if source IP address is found in the Tlogs dictionary then
if a different source IP address with the same MAC address is found in the Tlogs dictionary then
***** Spoofed source MAC address detected ******
Create a new rule to block it and if this rule hadn't been added to the TRules dictionary, then added it to the TRules dictionary.
Append the new rule into the CSV file and resend Firewall rules to the OVS switch.
else
Add source MAC/IP addresses to the Tlogs dictionary for future investigation

```
# Add to the main code
# This function is responsible to analyse incoming packets to detect spoofed IP/MAC addresses.
# In the first part I want to detect Spoofed source IP address, In orderto do this I lookup for source IP address
# in the Tlogs dictionary that I used to hold spoofed MAC/IP addresses. If I don't find this IP address in the
# Tlogs dictionary, then I need to check source MAC addreses, IF I find another record with the same source MAC address
# and different source IP address, this means I detect spoofed source IP address.
# I print out a message at the controller's console, "IP Spoofing Detected" and I create a new rule to block this flow from
# source MAC address and any source IP address to the specific destination IP address (h2 container host).
# In the second part I want to detect spoofed source MAC addresses, to do this I lookup for source MAC address
# In the Tlogs dictionary that I used to hold spoofed MAC/IP addresses. If I don't find this MAC address in the
# Tlogs dictionary, then I need to check source IP addresses, If I find another record with the same source IP address
# and different source MAC address, this means I detect spoofed source MAC address.
# I print out a message at controller's console, "MAC Spoofing detected" and I create a new rule to block this flow from
# source IP address and any source MAC address to the specific destination IP address (h2 container host).
def TrafficAnalyser(self, packet, match=None, event=None):

    srcmac = None
    srcip = None
    dstip = None
    if packet.type == packet.IP_TYPE:
        ip_packet = packet.payload
        # Spoofed source IP address detection
        if packet.src in self.Tlogs:
            # Source MAC address already exists in the TSpoofed table. I lookup for the combination of source IP and MAC
            # addrersses in the Tlogs dictionary. If I find the similar record with the same source IP and MAc address, based
            # on the lab's asumptions I conclude the traffic is ligitimate and I print out a message at controller's console
            # "Duplicate Entry".
            if self.Tlog.get(packet.src) == [ip_packet.srcip, ip_packet.dstip, event.port]:
                log.debug("Duplicate Entry: src_MAC: %s, src_IP: %s, dst_IP: %s, Port: %s" %
                          (str(packet.src), str(ip_packet.srcip), str(ip_packet.dstip), str(event.port)))
                return True
            else:
                # If I find a record with the same source MAC address and different source IP address, I can conclude at
                # this traffic the source IP address spoofed.
                newip = self.Tlog.get(packet.src)[0]
                if newip != ip_packet.srcip:
                    log.debug("IP spoofing detected @ Attacker: MAC: %s, Spoofed IP: %s,  Target(Victim): IP: %s, Port %s ***" %
                              (str(packet.src), str(newip), str(ip_packet.dstip), str(event.port)))
                    # I create a rule at the firewall with the source MAC address and destination IP address, with any source
                    # source IP addresses to to protect victim from DoS attack.
                    srcmac = str(packet.src)
                    srcip = None
                    dstip = str(ip_packet.dstip)
                    # I call the UpdateCSV function to add new rules based on the spoofed MAC or IP addresse detection.
                    self.UpdateCSV (srcmac, 'any', dstip)
                    return True
        # Spoofed source MAC address detection.
        else:
            for spoofedMAC, spoofedIPs in self.Tlogs.items():
                # Duplicate source IP address found, MAC address spoofed.
                if str(spoofedIPs[0]) == str(ip_packet.srcip):
                    log.debug(" MAC spoofing detected @ Attacker: IP: %s - MAC: %s, Target(Victim): IP: %s, MAC: %s, Port %s ***" %
                              (str(ip_packet.srcip), str(spoofedMAC), str(spoofedIPs[1]), str(packet.src), str(event.port)))
                    # I create a rule to block flow from the source and destination IP addresses with any source MAC address
                    # to protect the victim from the DoS attack.
                    srcmac = None
                    srcip = str(ip_packet.srcip)
                    dstip = str(ip_packet.dstip)
                    self.UpdateCSV ('any', srcip, dstip)
            # If the combination of the source IP and MAC addresses are unique, based on the assumptions on this lab I
            # consider this traffic as a legitimate traffic, I add source IP and MAC addresses, destination IP addresses, and
            # switch port number to Tlogs dictionary for future traffic investigations.
            self.Tlog [packet.src] = [ip_packet.srcip, ip_packet.dstip, event.port]
            log.debug("Update TSpoofed Table : src_MAC: %s, src_IP: %s, dst_IP: %s, Port: %s" %
                      (str(packet.src), str(ip_packet.srcip), str(ip_packet.dstip), str(event.port)))
            return True

        srcmac = srcmac
        dstmac = None
        sport = None
        dport = None
        nwproto = str(match.nw_proto)
        # In regular case, the firewall rules send to the switch when controller initialized.
        # In this lab we want to add new rules to the switch on the fly based on the content of the TRules dictionary,
        # then we need to recall installFlowcfunction to resend latest filrewall's rules from controller to the OVS switch.
        log.debug("Reinstall the firewall rules... ")
        self.installFlow(event, 10000, srcmac, None, srcip, dstip, None, None, nw_proto)

        return False
# ------
```

Figure-14 TrafficAnalyser function

The function “TrafficAnalyser” will be called by “_handle_PacketIn” function on the first packet of every new traffic arrived into the OVS switch (Figure-14). If this function detect spoofed MAC/IP addresses in the traffic then it create a rule to block this traffic and then it will call the function “UpdateCSV” to add this rule into the l3firewall.config file (Figure-15).

```
## Add to the main code
def UpdateCSV(self, srcmac='any', srcip='any', dstip='any'):

    # In this function I review all rules that defined previously, if the current rule didn't saved in the past
    # then I Add this rule to the firewall rule and I saved new rule to the CSV file.
    # I used AllowAdd variable to specify type of action based on rule's status.

    AllowAdd = True      # Initialize variable, default value allows to add current rule to the firewall
    for spoofedMAC, spoofedIPs in self.TRules.items():
        if spoofedMAC == str(srcmac) and spoofedIPs[0] == str(srcip) and spoofedIPs[1] == str(dstip):
           AllowAdd = False    # Duplicate rule detected.
           break

    if AllowAdd:       # Current rule is a new one and I add this rule to the firewall.
        self.TRules [str(srcmac)] = [str(srcip), str(dstip)]
        # Open csv file in append mode and add new rule at the end of CSV file.
        # I used a large number (e.g, 10000) as the priority number because I want this rule execute with the
        # lowest priority.
        # If I want to block DoS traffic from attacker host then I need to specify source IP/MAC addresses in the
        # new rule, hence I want to mitigate DDoS attack againt target host, then I specify detination IP address
        # to block all traffic from any source to specific IP address.
        with open(l3config, 'a') as csvfile:
            csvwriter = csv.DictWriter(csvfile, fieldnames=[
               'priority','src_mac','dst_mac','src_ip','dst_ip','src_port','dst_port','nw_proto',])
            csvwriter.writerow({
                'priority': 10000,
                'src_mac' : str(srcmac),
                'dst_mac' : 'any',
                'src_ip'  : str(srcip),
                'dst_ip'  : str(dstip),
                'src_port': 'any',
                'dst_port': 'any',
                'nw_proto': 'any',
                })
            log.debug("Rule saved in l3firewall.config file: srcip=%s dstip=%s srcmac=%s" % (str(srcip), str(dstip), str(srcmac)))
## ------
```

Figure-15 UpdateCSV function

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

At this time, I replaced the L3Firewall.py script with the code that I modified to detect spoofed source IP/MAC address traffic and block it. Then I generated flooding-traffic with a spoofed source IP address from h1 (MAC: 00:00:00:00:00:0a) to h2 (IP: 192.168.2.20) and I used the ping command on h4 to send ICMP traffic to h2 to check the status of h2 (Figure-16):

h1: hping3 192.168.2.20 –S –flood –c 10000 --rand-source

h4: ping 192.168.2.20

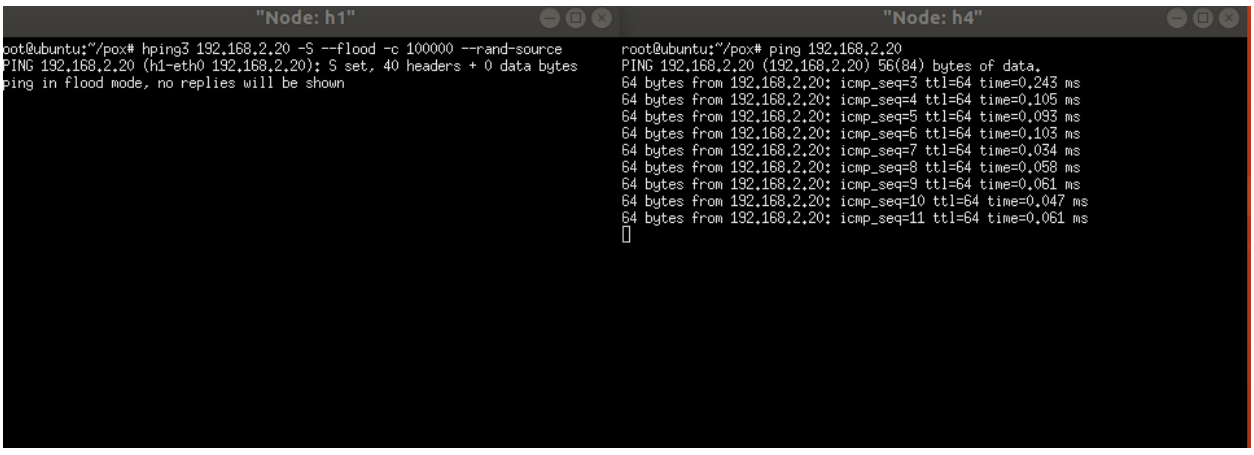

Figure-16 hping3, ping commands

I immediately checked OpenFlow flow-entries at switch s1, and I saw flooding-traffic forwarded to h2 (Figure-17):

sudo ovs-ofctl dump-flows s1

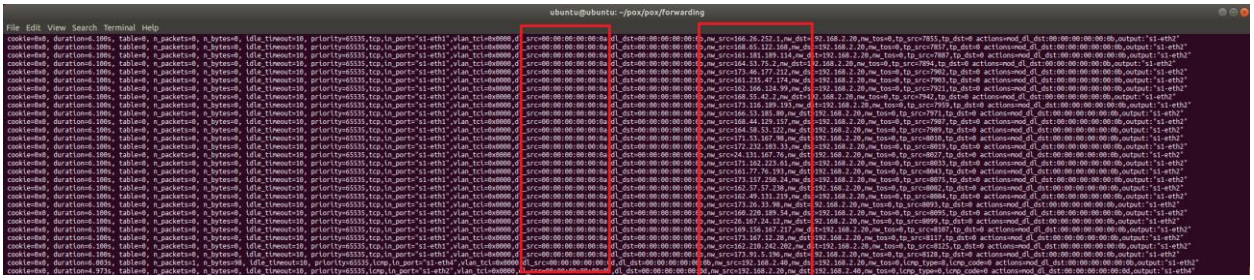

Figure-17 The OpenFlow flow-entries at the switch s1

After a while my code in L3Firewall.py detected the spoofed source IP traffic, and the POX controller added a rule in the l3firewall.config file to block it. The above screenshot shows that the h2 container host didn't flood and was able to send the reply traffic back to h4.

Due to the spoofed source IP address, the firewall can block flooding-traffic based on the source MAC address of the attacker host (h1: 00:00:00:00:00:0a) (Figure-18):

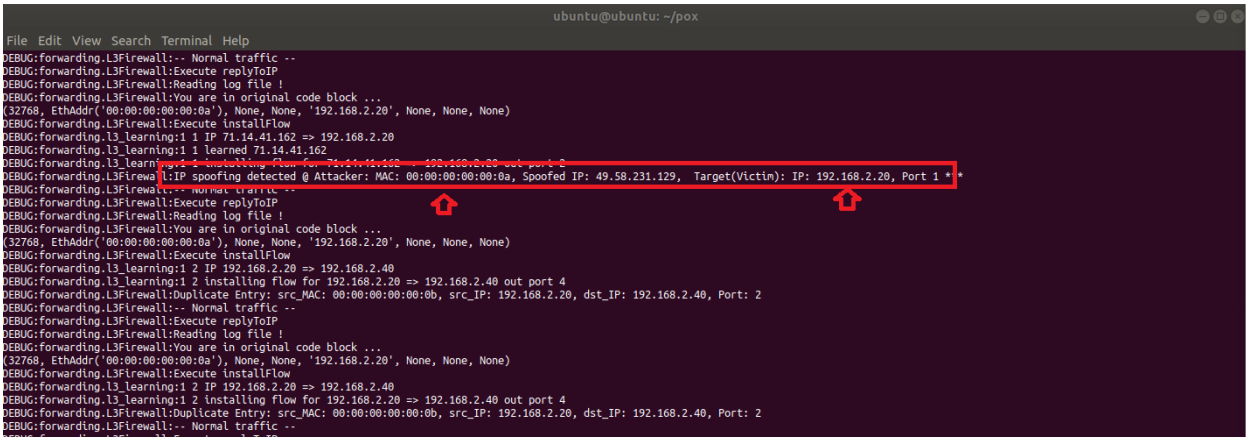

Figure-18 IP spoofed traffic detected

I checked the OpenFlow flow-entries at switch s1again, and the output showed that the flooding-traffic had been blocked by the firewall and we could only see related flow-entries at switch s1 for the ping command (Figure-19):

sudo ovs-ofctl dump-flows s1

When I checked the L3Firewall rules on the l3firewall.config file, I saw that the below rule has been added:

10000,00:00:00:00:00:0a,any,any,192.168.2.20,any,any,any

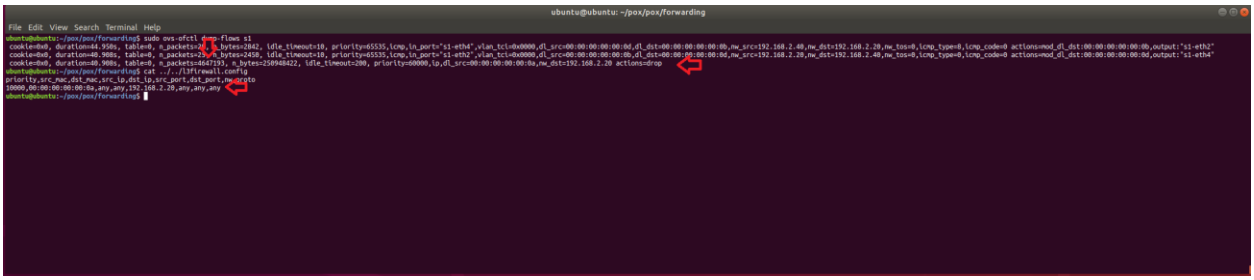

Figure-19 The OpenFlow flow-entries at the switch s1 and firewall rule

In this section I want to show that modified L3Firewall.py script can detect and mitigate DoS attacks based on the spoofed source MAC address.

I checked the OpenFlow flow-entries at switch s1, and the output confirmed no flow-entries had been recorded by switch s1 and I checked the content of the l3firewall.config file, and the output showed that there was a rule to block all traffic based on the source MAC address (h1) and destination IP address (h2). This rule was created in the previous step (Figure-20).

sudo ovs-ofctl dump-flows s1

cat ../../l3firewall.config

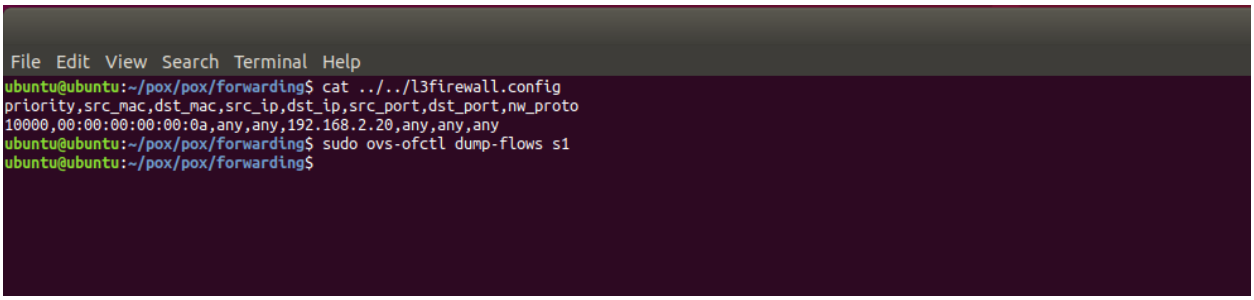

Figure-20 The OpenFlow flow-entries at the switch s1 and firewall rule

Before I started the next step, I remove the rule that I had created to block the traffic between h1 and h2 in the previous step.

I used the nping command to generate 10 ICMP packets with spoofed source MAC addresses (00:11:00:11:00:11) from container host h4 to container host h2 (Figure-21):

nping –icmp –icm-type 0 –source-mac 00:11:00:11:00:11 -dest-ip 192.168.2.20 –c 10

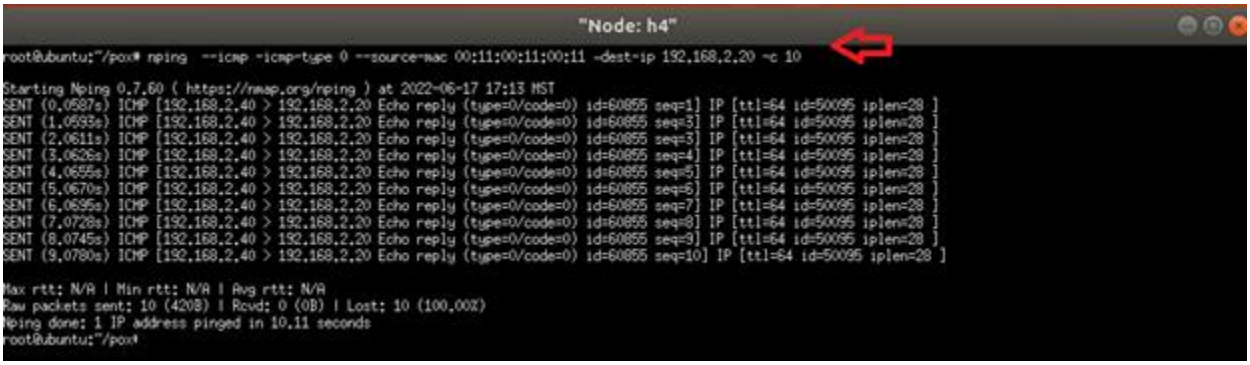

Figure-21 nping – generate spoofed source MAC address ICMP traffic

I received the message “MAC spoofing detected” in the controller’s console that confirmed the spoofed source MAC address traffic had been detected. The information about the detected traffic showed the traffic was sent from the source IP address 192.168.2.40 (with spoofed source MAC address) to the destination IP address 192.168.2.20 (Figure-22):

Figure-22 Spoofed source MAC address traffic detected

When I checked the flow-entries at switch s1, the output showed the traffic had been blocked by the switch, because the modified L3Firewall.py script had detected spoofed source MAC address traffic and had added a rule to block this traffic in the l3firewall.config file. The flow-entries in switch s1 and the content of the l3firewall.config file are shown below (Figure-23):

sudo ovs-ofctl dump-flows s1

cat ../../l3firewall.config

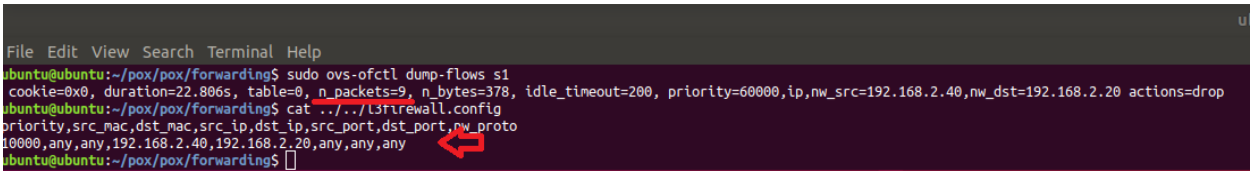

Figure-23 The OpenFlow flow-entries at the switch s1 and firewall rule

In this section I wanted to generate random spoofed source MAC address traffic at the host container h4 and send it to container host h2 to simulate a DoS attack on the target. I used the nping command to generate ICMP traffic as shown below (Figure-24):

nping –icmp –source-mac random 192.168.2.20 –c 10

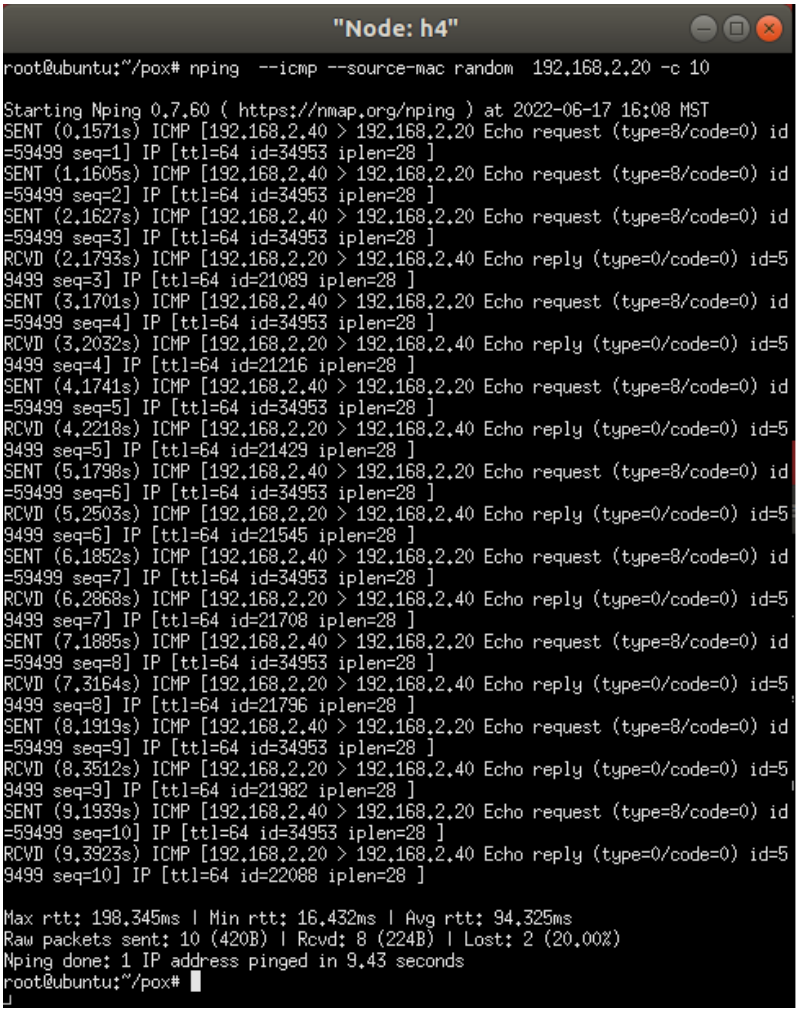

Figure-24 nping – generate spoofed source MAC address ICMP traffic

I used the tcpdump command on the container host h2 to see the contents of the packets with spoofed source MAC addresses before they were blocked by the firewall at switch s1 (Figure-25):

tcpdump –I h2-eth0 -e

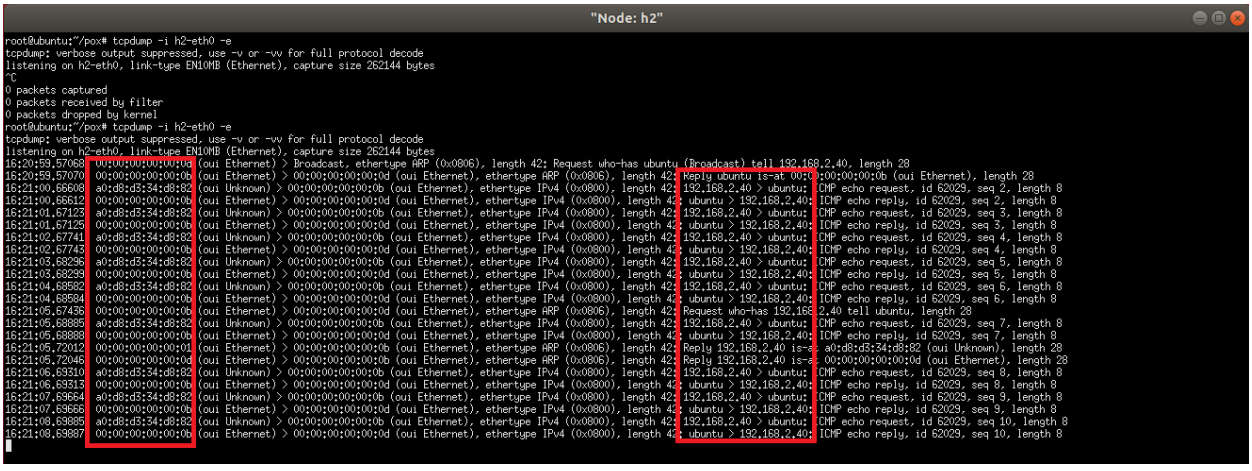

Figure-25 tcpdump command

When the spoofed source MAC address of the traffic was detected by the switch, we saw a message at the controller console, “MAC spoofing detected” and the L3Firewall script generated a rule to block this traffic and added this rule to the l3firewall.config file (Figure 26):

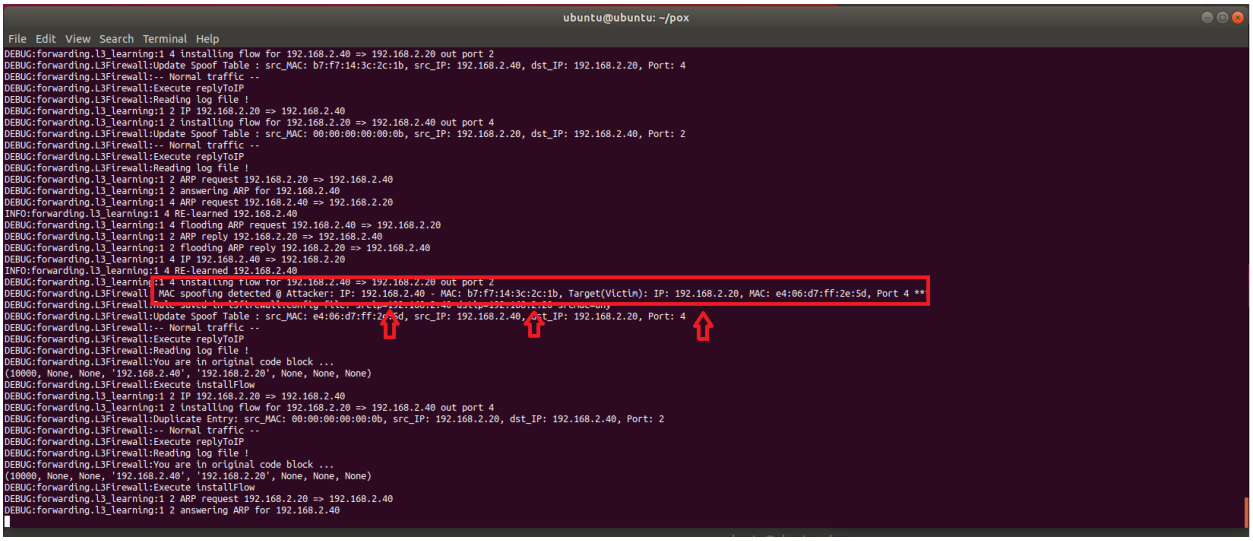

Figure-26 Spoofed source MAC address traffic detected

When I checked the flow-entries at switch s1, the output showed that the traffic had been blocked by the switch, because the modified L3Firewall.py script had detected spoofed source MAC address traffic and had added a rule to block this traffic in the l3firewall.config file. The flow-entries in switch s1 and the content of the l3firewall.config file are shown below (Figure-27):

sudo ovs-ofctl dump-flows s1

cat ../../l3firewall.config

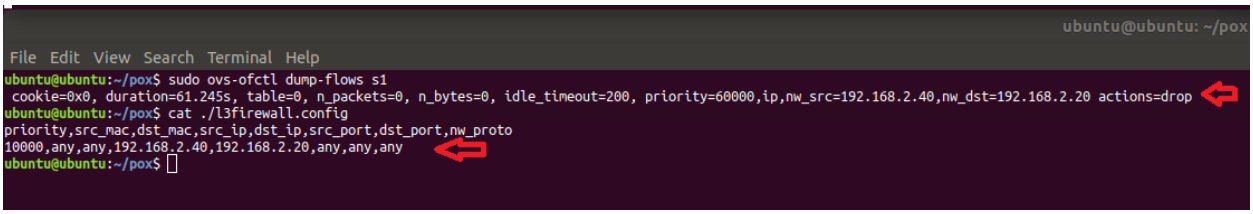

Figure-27 The OpenFlow flow-entries at the switch s1 and firewall rule

When I generated spoofed source MAC address ICMP traffic from container host h2 again, I saw that the traffic had been blocked by the firewall completely and we didn’t receive any reply packet at the container host h4 (Figure-28):

nping –icmp –source-mac random 192.168.2.20 –c 10

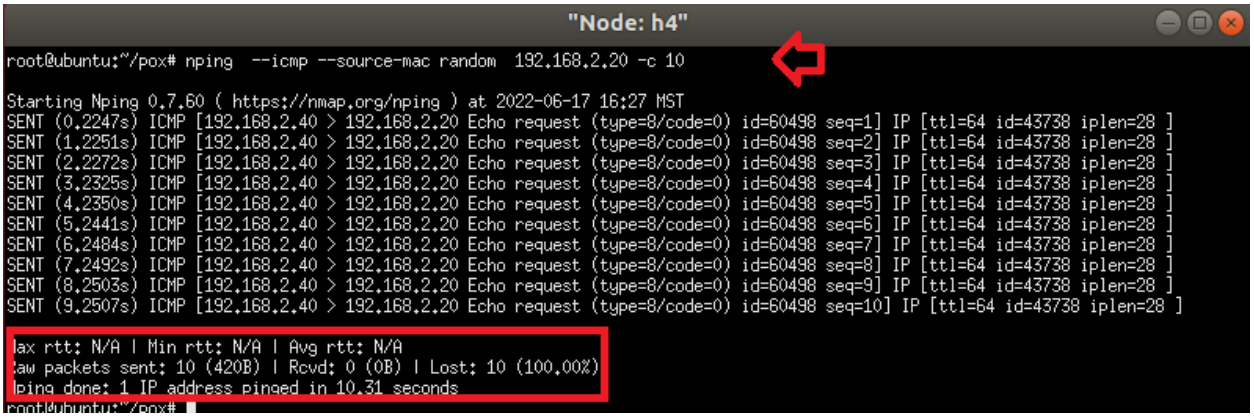

Figure-28 nping – generate spoofed source MAC address ICMP traffic

The flow-entries on switch s1 confirms the all packets had been blocked by the firewall and I showed the content of the l3firewall.config file that a rule has been added to bock spoofed source MAC address traffic from h4 (192.168.2.40) to h2 (192.168.2.20) (figure-29):

Figure-29 The OpenFlow flow-entries at the switch s1 and firewall rule

(c) You should submit the source codes of your implementation.

Please find the source code of my implementation in the GitHub, use provided link at the Appendix.

V. Conclusion

In this lab I implemented an SDN-based network with Mininet and I used a layer 3 firewalls to block suspicious traffic based on the Spoofed source MAC/IP addresses to mitigate the DoS attack.

The Mininet provides a virtual test bed and development environment for software-defined networks (SDN) [1]. As part of this lab, I modified the code of the L3Firewall.py file to detect and block spoofed source MAC/IP addresses.

We have several options to detect and mitigate the DoS attacks on the SDN-based networks.

- Look up for the spoofed source MAC/IP addresses at the OVS switches (I implemented this option in this lab)
- Look up for the spoofed source MAC/IP addresses at the l3_learning.py file at the POX controller.
- Using POX controller's parameters to restrict the packets per second in flow traffic in the OVS switches.

VI. Appendix B: Attached files

L3Firewall.py (Modified version)	https://github.com/MehranTJB/ASU-CSE5548-Advanced-Network-Security/blob/main/L3Firewall.py
Lab-3 (CS-CNS-00103) video	https://mehrantajbakhsh.com/cse548/CS-CNS-00103.mp4

VII. References

1. Mininet - https://wiki.opennetworking.org/display/COM/Mininet
2. POX Documentation - https://noxrepo.github.io/pox-doc/html/
3. Open vSwitch (OVS) Documentation - https://docs.openvswitch.org/en/latest/
4. Tcpdump Documentation - https://www.tcpdump.org/manpages/tcpdump.1.html
5. hping3 - http://www.hping.org/
6. OpenNet - https://github.com/dlinknctu/OpenNet/blob/master/doc/TUTORIAL.md