

SDN-Based Stateless Firewall Project

Student Name: Jeremy Escobar

Email: jgescoba@asu.edu

Submission Date: May 27, 2024

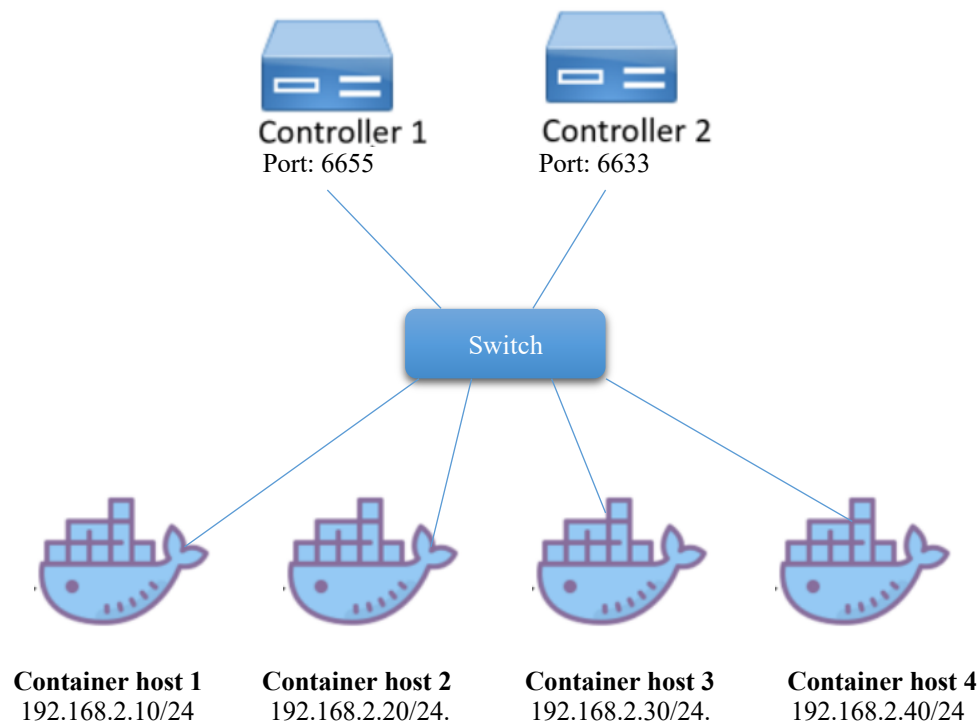
Class Name and Term: CSE548 Summer 2024, Session C

I. PROJECT OVERVIEW

The major aim will be to devise a firewall through OpenFlow rules and an SDN controller—Software Defined Networking. Practical experience that one will gain from the project includes the ability to configure and implement policies, the monitoring of traffic, and devising mitigation strategies of Denial of Service. This will mean setting up the virtual environment through tools like Mininet, Containernet, POX controllers, Open Virtual Switch, and finally by documenting the progress in a report and a demonstration video record. This project will further avail concepts of network security and the SDN architecture.

II. NETWORK SETUP

Network topology for SDN



Container Host	Layer 2 Address (MAC address)	Layer 3 Address (IP address)
Container host h1	00:00:00:00:00:01	10.0.0.1
Container host h2	00:00:00:00:00:02	10.0.0.2
Container host h3	00:00:00:00:00:03	10.0.0.3
Container host h4	00:00:00:00:00:04	10.0.0.4

III. SOFTWARE

I used VirtualBox for this project to emulate an environment that contained this pre-built Ubuntu VDI file from the class material and Mininet, Containernet, POX controllers, and Open Virtual Switch to facilitate the creation and management of the SDN-based stateless firewall.

IV. PROJECT DESCRIPTION

In this project, I set up a network using Mininet and configured a software-defined network (SDN) firewall with POX to enforce security rules. To test the firewall, I performed the following test cases:

- **Ping Tests:** I initiated a ping from h1 to h3 using **h1 ping -c 3 h3** and from h2 to h4 using **h2 ping -c 3 h4** to ensure that ICMP traffic was blocked as intended.
- **TCP Test:** I used **hping3** to send SYN packets from h1 to h2 on port 80 with the command **h1 hping3 -S h2 -p 80 -c 5**, confirming the firewall's effectiveness in blocking TCP packets.
- **UDP Test:** I tested UDP traffic by sending packets from h1 to h2 on port 80 with **hping3** using the command **h1 hping3 --udp h2 -p 80 -c 5** to verify that the firewall blocked these packets.
- **HTTP Request Test:** I used **curl** from h1 to access a web server on h2 with the command **h1 curl http://192.168.2.20 -p 80**, confirming that HTTP traffic was blocked by the firewall.
- **MAC Address Test:** I conducted a MAC address test by pinging from h2 to h4 using **h2 ping -c 3 h4**, ensuring that traffic was blocked based on MAC address rules.

These tests demonstrated the functionality and effectiveness of the firewall rules I implemented.

Step 1: I installed all the necessary dependencies.

Figure 1: Installation of all dependencies.

```
vboxuser@Project2CSE548:~$ sudo apt install wget unzip python python3 mininet openvswitch-switch
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'python-is-python2' instead of 'python'
python3 is already the newest version (3.8.2-0ubuntu2).
python3 set to manually installed.
unzip is already the newest version (6.0-25ubuntu1.2).
unzip set to manually installed.
wget is already the newest version (1.20.3-1ubuntu2).
```

Step 2: Verified installation of the dependencies and tests *mininet* with a *pingall* test.

Figure 2: Testing dependencies and version numbers

```
vboxuser@Project2CSE548:~$ python --version
Python 2.7.18
vboxuser@Project2CSE548:~$ python3 --version
Python 3.8.10
vboxuser@Project2CSE548:~$ mn --version
2.2.2
vboxuser@Project2CSE548:~$ sudo mn --test pingall
*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 0 controllers
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 0.483 seconds
vboxuser@Project2CSE548:~$
```

Step 3: Installed pox via git clone.

Figure 3: Pox installation

```
vboxuser@Project2CSE548:~$ git clone https://github.com/noxrepo/pox.git
Cloning into 'pox'...
remote: Enumerating objects: 13058, done.
remote: Counting objects: 100% (283/283), done.
remote: Compressing objects: 100% (134/134), done.
remote: Total 13058 (delta 174), reused 242 (delta 144), pack-reused 12775
Receiving objects: 100% (13058/13058), 5.02 MiB | 5.21 MiB/s, done.
Resolving deltas: 100% (8423/8423), done.
vboxuser@Project2CSE548:~$ cd pox
vboxuser@Project2CSE548:~/pox$
```

Step 4: Downloaded the l3firewall.config and l2firewall.config and L3Firewall.py from project link.

Figure 4: Installation of all dependencies.

```
vboxuser@Project2CSE548: ~/pox
--2024-05-24 13:16:21-- https://raw.githubusercontent.com/SaburH/CSE548/main/lab-cs-cns-00101.zip
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133,
185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected
.
HTTP request sent, awaiting response... 200 OK
Length: 5497 (5.4K) [application/zip]
Saving to: 'lab-cs-cns-00101.zip'

lab-cs-cns-00101.zip      100%[=====] 5.37K  --.-KB/s  in 0s

2024-05-24 13:16:22 (11.2 MB/s) - 'lab-cs-cns-00101.zip' saved [5497/5497]

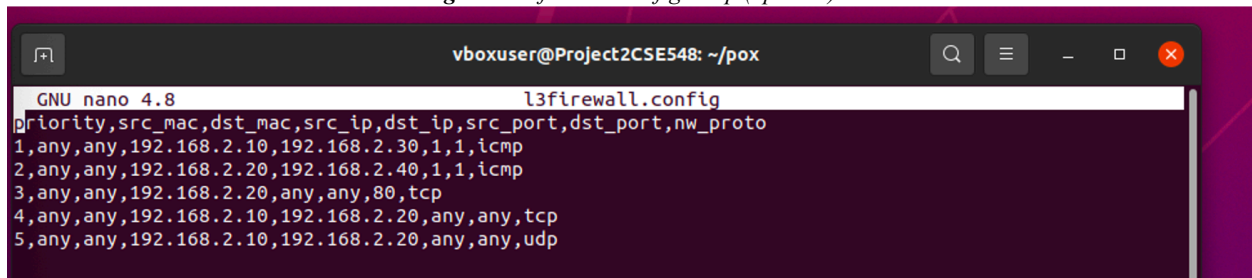
vboxuser@Project2CSE548:~$ unzip lab-cs-cns-00101.zip
Archive: lab-cs-cns-00101.zip
  creating: lab-cs-cns-00101/
  inflating: __MACOSX/._lab-cs-cns-00101
  inflating: lab-cs-cns-00101/l3firewall.config
  inflating: __MACOSX/lab-cs-cns-00101/._l3firewall.config
  inflating: lab-cs-cns-00101/l2firewall.config
  inflating: __MACOSX/lab-cs-cns-00101/._l2firewall.config
  inflating: lab-cs-cns-00101/.DS_Store
  inflating: __MACOSX/lab-cs-cns-00101/._.DS_Store
  inflating: lab-cs-cns-00101/L3Firewall.py
  inflating: __MACOSX/lab-cs-cns-00101/._L3Firewall.py
vboxuser@Project2CSE548:~$ sudo cp lab-cs-cns-00101/l2firewall.config /home/vboxuser/pox/
cp: cannot stat 'lab-cs-cns-00101/l2firewall.config': No such file or directory
vboxuser@Project2CSE548:~$ ls
Desktop  Downloads  lab-cs-cns-00101.zip  Music  pox  Templates
Documents  lab-cs-cns-00101  __MACOSX  Pictures  Public  Videos
vboxuser@Project2CSE548:~$ pwd
/home/vboxuser
vboxuser@Project2CSE548:~$ cd lab-cs-cns-00101
vboxuser@Project2CSE548:~/lab-cs-cns-00101$ ls
l2firewall.config  l3firewall.config  L3Firewall.py
vboxuser@Project2CSE548:~/lab-cs-cns-00101$ sudo cp lab-cs-cns-00101/l2firewall.config /home/vboxuser/
/pox/
cp: cannot stat 'lab-cs-cns-00101/l2firewall.config': No such file or directory
vboxuser@Project2CSE548:~/lab-cs-cns-00101$ sudo cp l2firewall.config /home/vboxuser/pox/
vboxuser@Project2CSE548:~/lab-cs-cns-00101$ sudo cp l3firewall.config /home/vboxuser/pox/
vboxuser@Project2CSE548:~/lab-cs-cns-00101$ sudo cp L3Firewall.py /home/vboxuser/pox/
vboxuser@Project2CSE548:~/lab-cs-cns-00101$ cd
vboxuser@Project2CSE548:~$ cd pox
vboxuser@Project2CSE548:~/pox$ ls
debug-pox.py  l2firewall.config  L3Firewall.py  NOTICE  pox.py  setup.cfg  tools
ext  l3firewall.config  LICENSE  pox  README.md  tests
vboxuser@Project2CSE548:~/pox$
```

Step 5: Set the permissions for each file so I could edit them (l2firewall.config, l3firewall.config).

Figure 5: Setting permissions below.

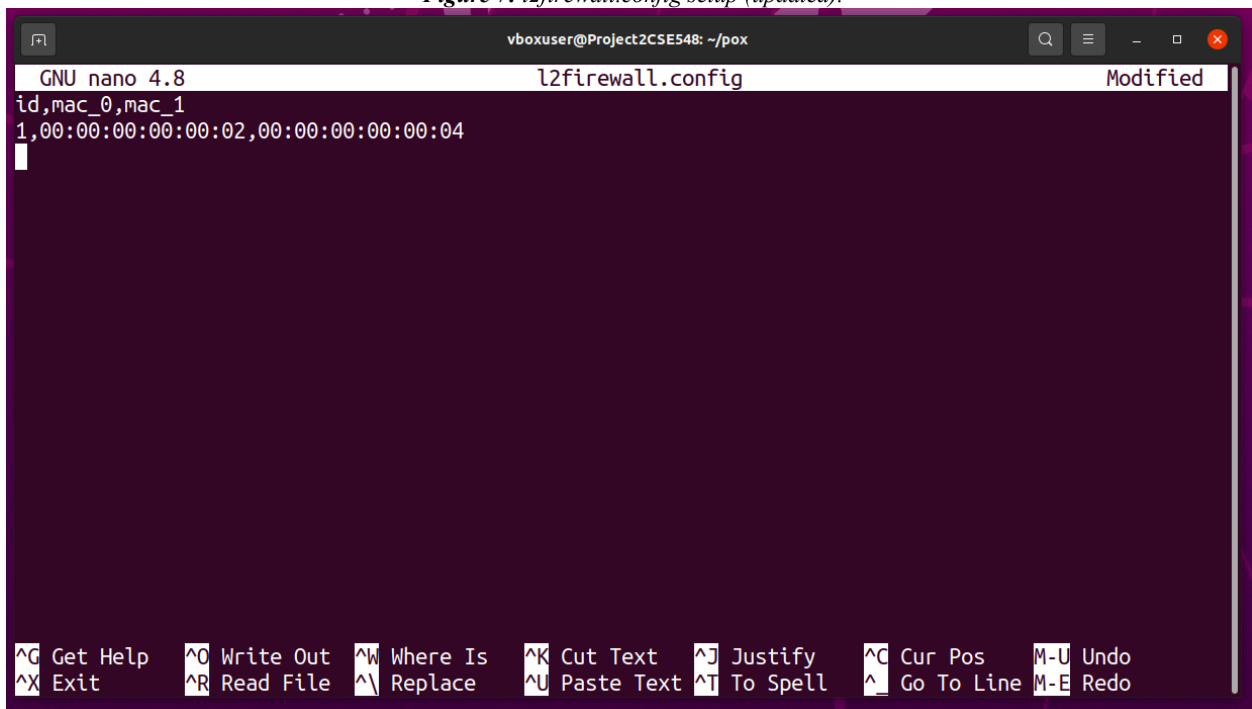
```
vboxuser@Project2CSE548:~/pox$ chmod 777 l2firewall.config
chmod: changing permissions of 'l2firewall.config': Operation not permitted
vboxuser@Project2CSE548:~/pox$ sudo chmod 777 l2firewall.config
vboxuser@Project2CSE548:~/pox$ sudo chmod 777 l3firewall.config
vboxuser@Project2CSE548:~/pox$ nano l3firewall.config
vboxuser@Project2CSE548:~/pox$ nano l2firewall.config
vboxuser@Project2CSE548:~/pox$
```

Figure 6: l3firewall.config setup (updated).



```
GNU nano 4.8 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,any,80,tcp
4,any,any,192.168.2.10,192.168.2.20,any,any,tcp
5,any,any,192.168.2.10,192.168.2.20,any,any,udp
```

Figure 7: l2firewall.config setup (updated).



```
GNU nano 4.8 l2firewall.config Modified
id,mac_0,mac_1
1,00:00:00:00:00:02,00:00:00:00:00:04
```

Step 6: Next, I went ahead and setup pox to run on port:6655 allowing the firewalls to be enabled.

Figure 8: Installation of all dependencies.

```
ubuntu@ubuntu:~/pox$ sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l2_
learning pox.forwarding.L3Firewall --l2config="l2firewall.config" --l3config="l3
firewall.config"
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
src_ip, dst_ip, src_port, dst_port 198.168.2.10 198.168.2.301 1 icmp
src_ip, dst_ip, src_port, dst_port 198.168.2.20 198.168.2.40 1 1
src_ip, dst_ip, src_port, dst_port 198.168.2.20 any any 80
src_ip, dst_ip, src_port, dst_port 198.168.2.10 192.168.2.20 any any
src_ip, dst_ip, src_port, dst_port 198.168.2.10 198.168.2.20 any any
INFO:core:POX 0.5.0 (eel) is up.
```

Step 7: Start the mininet and pox.*Figure 9: This below shows both the pox and the mininet running now.*

The screenshot shows two terminal windows on an Ubuntu system. The top window, titled 'Assignment2CSE548 [Running]', shows the execution of the 'mn' command to create a Mininet network. The output indicates that the network was successfully created with four hosts (h1, h2, h3, h4), one switch (s1), and one controller (c0). The bottom window, titled 'ubuntu@ubuntu: ~/pox', shows the execution of the 'pox.py' command to start the POX controller. The output shows the configuration of the controller and the start of the POX application.

```


ubuntu@ubuntu:~$ sudo mn --topo=single,4 --controller=remote,port=6633 --control
ler=remote,port=6655 --switch=ovsk --mac
[sudo] password for ubuntu:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0 c1
*** Starting 1 switches
s1 ...
*** Starting CLI:
containernet>

ubuntu@ubuntu:~/pox
Name           Default           Active
-----
l2config       l2firewall.config l2firewall.config
l3config       l3firewall.config l3friewall.config
This component does not have a parameter named 'l2congfig'.
ubuntu@ubuntu:~/pox$ sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l2_
learning pox.forwarding.L3Firewall --l2config="l2firewall.config" --l3config="l3
friewall.config"
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
Traceback (most recent call last):
  File "/home/ubuntu/pox/pox/boot.py", line 532, in boot
    if _do_launch(argv):
  File "/home/ubuntu/pox/pox/boot.py", line 243, in _do_launch
    if f(**params) is False:
  File "/home/ubuntu/pox/pox/forwarding/L3Firewall.py", line 204, in launch
    core.registerNew(Firewall,l2config,l3config)
  File "/home/ubuntu/pox/pox/core.py", line 412, in registerNew
    obj = __componentClass(*args, **kw)
  File "/home/ubuntu/pox/pox/forwarding/L3Firewall.py", line 56, in __init__
    with open(l3config) as csvfile:
IOError: [Errno 2] No such file or directory: 'l3friewall.config'
ubuntu@ubuntu:~/pox$ sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l2_
learning pox.forwarding.L3Firewall --l2config="l2firewall.config" --l3config="l3
firewall.config"
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
src_ip, dst_ip, src_port, dst_port 198.168.2.10 198.168.2.301 1 icmp
src_ip, dst_ip, src_port, dst_port 198.168.2.20 198.168.2.40 1 1
src_ip, dst_ip, src_port, dst_port 198.168.2.20 any any 80
src_ip, dst_ip, src_port, dst_port 198.168.2.10 192.168.2.20 any any
src_ip, dst_ip, src_port, dst_port 198.168.2.10 198.168.2.20 any any
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
00:00:00:00:00:02 00:00:00:00:00:04

```

Step 9: Next, I loaded the containers with `xterm h1 h2 h3 h4`

Figure 10: Command to load the containers xterm h1 h2 h3 h4.

A terminal window titled 'ubuntu@ubuntu: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal output shows the process of stopping 4 hosts, creating a network, adding a controller, adding hosts (h1, h2, h3, h4), adding switches (s1), adding links, configuring hosts, starting the controller (c0, c1), starting switches (s1), and starting the CLI. The final command entered is 'xterm h1 h2 h3 h4' and the prompt returns to 'containernet>'.

```
ubuntu@ubuntu: ~  
File Edit View Search Terminal Help  
*** Stopping 4 hosts  
h1 h2 h3 h4  
*** Done  
completed in 126.405 seconds  
ubuntu@ubuntu:~$ sudo mn --topo=single,4 --controller=remote,port=6633 --control  
ler=remote,port=6655 --switch=ovsk --mac  
*** Creating network  
*** Adding controller  
Unable to contact the remote controller at 127.0.0.1:6633  
*** Adding hosts:  
h1 h2 h3 h4  
*** Adding switches:  
s1  
*** Adding links:  
(h1, s1) (h2, s1) (h3, s1) (h4, s1)  
*** Configuring hosts  
h1 h2 h3 h4  
*** Starting controller  
c0 c1  
*** Starting 1 switches  
s1 ...  
*** Starting CLI:  
containernet> xterm h1 h2 h3 h4  
containernet> 
```


Step 10: Next, I setup the host interfaces, with h1 ifconfig as you see below for all 4 hosts.

Figure 11: Setting up the interfaces per host.

```
"Node: h1"
root@ubuntu:~# ifconfig h1-eth0 192.168.2.10/24
root@ubuntu:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.10 netmask 255.255.255.0 broadcast 192.168.2.255
    ether 00:00:00:00:00:01 txqueuelen 1000 (Ethernet)
    RX packets 27 bytes 3800 (3.8 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3 bytes 310 (310.0 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:~#
```

```
"Node: h2"
root@ubuntu:~# ifconfig h2-eth0 192.168.2.20/24
root@ubuntu:~# ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.20 netmask 255.255.255.0 broadcast 192.168.2.255
    ether 00:00:00:00:00:02 txqueuelen 1000 (Ethernet)
    RX packets 28 bytes 3870 (3.8 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3 bytes 310 (310.0 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:~#
```

```
"Node: h3"
root@ubuntu:~# ifconfig h3-eth0 192.168.2.30/24
root@ubuntu:~# ifconfig
h3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.30 netmask 255.255.255.0 broadcast 192.168.2.255
    ether 00:00:00:00:00:03 txqueuelen 1000 (Ethernet)
    RX packets 30 bytes 4163 (4.1 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3 bytes 310 (310.0 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:~#
```

```
"Node: h4"
root@ubuntu:~# ifconfig h4-eth0 192.168.2.40/24
root@ubuntu:~# ifconfig
h4-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.40 netmask 255.255.255.0 broadcast 192.168.2.255
    ether 00:00:00:00:00:04 txqueuelen 1000 (Ethernet)
    RX packets 28 bytes 3987 (3.9 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3 bytes 310 (310.0 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:~#
```

Step 11: Testing ICMP traffic rule, I initiated a ping from h1 to h3 using **h1 ping -c 3 h3** and from h2 to h4 using **h2 ping -c 3 h4** to ensure that ICMP traffic was blocked as intended.

Figure 12: Testing ICMP traffic, h1 ping h3, h2 ping h4 below:

```

ubuntu@ubuntu: ~
File Edit View Search Terminal Help
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

containernet> xterm h1 h2 h3 h4
containernet> h1 ping -c 3 h3
PING 192.168.2.30 (192.168.2.30) 56(84) bytes of data.
64 bytes from 192.168.2.30: icmp_seq=1 ttl=64 time=78.8 ms

--- 192.168.2.30 ping statistics ---
3 packets transmitted, 1 received, 66% packet loss, time 2029ms
rtt min/avg/max/mdev = 78.858/78.858/78.858/0.000 ms
containernet> h1 ping -c 3 h3
PING 192.168.2.30 (192.168.2.30) 56(84) bytes of data.

--- 192.168.2.30 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2039ms

containernet> h2 ping -c 3 h4
PING 192.168.2.40 (192.168.2.40) 56(84) bytes of data.

--- 192.168.2.40 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2017ms

```

Step 12: Testing TCP rule, I used **hping3** to send SYN packets from h1 to h2 on port 80 with the command **h1 hping3 -S h2 -p 80 -c 5**, confirming the firewall's effectiveness in blocking TCP packets.

Figure 13: Setting up the interfaces per host.

```

containernet> h1 hping3 -S h2 -p 80 -c 5
HPING 192.168.2.20 (h1-eth0 192.168.2.20): S set, 40 headers + 0 data bytes

--- 192.168.2.20 hping statistic ---
5 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

```

Step 13: Testing UDP rule, I tested UDP traffic by sending packets from h1 to h2 on port 80 with **hping3** using the command **h1 hping3 --udp h2 -p 80 -c 5** to verify that the firewall blocked these packets.

Figure 14: Setting up the interfaces per host.

```

Mon 20:14
ubuntu@ubuntu: ~
File Edit View Search Terminal Help
containernet> h1 hping3 --udp h2 -p 80 -c 5
HPING 192.168.2.20 (h1-eth0 192.168.2.20): udp mode set, 28 headers + 0 data bytes

--- 192.168.2.20 hping statistic ---
5 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

```


Step 14: Testing HTTP request rule, I used **curl** from h1 to access a web server on h2 with the command **h1 curl http://192.168.2.20 -p 80**, confirming that HTTP traffic was blocked by the firewall.

Figure 15: Setting up the interfaces per host.

```
containernet> h1 curl http://192.168.2.20 -p 80
curl: (7) Failed to connect to 192.168.2.20 port 80: Connection refused
curl: (7) Failed to connect to 192.168.2.20 port 80: Connection refused
```

Step 15: Testing Mac Address rule, I conducted a MAC address test by pinging from h2 to h4 using **h2 ping -c 3 h4**, ensuring that traffic was blocked based on MAC address rules.

Figure 16: Setting up the interfaces per host.

```
containernet> h2 ping -c 3 h4
PING 192.168.2.40 (192.168.2.40) 56(84) bytes of data.

--- 192.168.2.40 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2017ms
```

V. CONCLUSION

Throughout this SDN-based stateless firewall project, I gained a understanding of software-defined networking and the crucial role it plays in modern network security. Setting up and configuring the Mininet environment, along with integrating POX controllers, provided me with hands-on experience in managing and controlling network traffic using OpenFlow rules. This project was not just an exercise; it was a significant learning journey that broadened my perspective on how networks can be dynamically controlled and secured. One of the key takeaways from this project was the importance of configuration. Ensuring that each component, from the Mininet virtual network to the POX controller and firewall rules, was correctly set up was vital. I learned to appreciate the detailed process of verifying installations, setting permissions, and configuring network interfaces. These steps, while sometimes tedious, are fundamental to the successful deployment of network security policies.

Testing the firewall rules was satisfying. By using simple ping tests, I observed firsthand how ICMP traffic can be effectively blocked. This test built my confidence as I moved on to more complex traffic types. Using **hping3** to simulate TCP and UDP traffic allowed me to verify the firewall's ability to handle different protocols. These tests underscored the versatility and reliability of SDN-based firewalls in managing diverse network traffic scenarios. The HTTP request test using **curl** was a practical demonstration of how application-level traffic can be controlled. This was especially interesting because it showed how SDN firewalls could be used to secure web traffic. This part of the project illustrated the applications of what I was learning and how it could be applied in real-life situations. In addition, the MAC address test was another crucial learning moment. It reinforced the concept that network security is not just about IP addresses and ports but also about controlling access based on hardware addresses. This test highlighted the importance of layer 2 security measures and how they complement higher-layer security policies. Overall, this project was a rewarding experience. It was challenging at times, particularly when troubleshooting configuration issues or ensuring that the correct firewall rules were in place. However, these challenges were also opportunities to learn and grow. I developed problem-solving skills and gained confidence in my ability to work with SDN and firewall technologies.

In conclusion, this project provided a comprehensive overview of SDN-based network security. It taught me the importance of detailed configuration, the versatility of SDN in handling various traffic types, and the practical applications of network security measures. I am grateful for this learning experience and feel more prepared to tackle real-world network security challenges in the future.

VI. APPENDIX B: ATTACHED FILES

VII. REFERENCES