

Project 2: SDN-Based Stateless Firewall

Student Name: Honglin Ma

Email: mhl970128@gmail.com

Submission Date: 06/03/2022

Class Name and Term: CSE548 spring 2022

I. PROJECT OVERVIEW

In this project, the main goal was to understand learn and build a software-defined environment based on mininet as well as containerernet (a kind of dockers) in a Linux system environment. In this environment, after creating virtual hosts through mininet, I practiced how to build an OpenFlow-based stateless flow firewall on SDN. By setting up the required firewall filtering rules in this environment, the function of blocking malicious attacking network traffic is achieved.

II. NETWORK SETUP

- In this experiment, a total of one virtual machine was configured on Oracle VM VirtualBox 6.0.24, it will be referred to as project23 in the following. The network was configured for this machine with the help of a virtual NAT device and a virtual DHCP service (Client-side Net 1: 10.0.2.0/24). The network was selected as shown in the figure below. initial reachability among network nodes



II-1 VM project23 net set

- In the Virtual machine project23, we need to create four containerernet hosts h1-h4 via mininet. These four hosts will be linked to a switch, and the switch will be linked to two controllers, as will be shown in the following implementation.

III. SOFTWARE

For this project, the following software has been used:

- Various network tools (tcpdump, ping, traceroute, hping3, curl etc.)
- Oracle VM VirtualBox 6.0.24
- Pox Controller
- OpenVirtual Switch
- Mininet
- Containernet

IV. PROJECT DESCRIPTION

Lab CS-CNS-00101 – OpenFlow Based Stateless firewall

The Youtube video demonstration address is: <https://www.youtube.com/watch?v=I-PvRwsgm-U>

Next I will show step by step the implementation process for the Lab CS-CNS-00101 (SDN Firewall):

First we need to check the installation of all required services and software in the host.

1) Check Python

```

Try python -h for more information.
ubuntu@ubuntu:~$ python --version
Python 2.7.17
ubuntu@ubuntu:~$ python3 --version
Python 3.6.9
ubuntu@ubuntu:~$

```

IV-1 Python check

2) Check Mininet

```

root@ubuntu:/home/ubuntu# mn --version
2.3.0d5
root@ubuntu:/home/ubuntu# mn --test pingall
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** 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 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.741 seconds
root@ubuntu:/home/ubuntu#

```

IV-2 Install Mininet

3) Check Pox controller

```

root@ubuntu:/home/ubuntu# ls
containernet  Documents  examples.desktop  lab-cs-cns-00101.zip  Music  oftest  Pictures  Public  Videos
Desktop       Downloads  lab-cs-cns-00101  __MACOSX             oflops  openflow  pox       Templates
root@ubuntu:/home/ubuntu# cd pox
root@ubuntu:/home/ubuntu/pox# ./pox.py -verbose forwarding.hub
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:forwarding.hub:Proactive hub running.
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.17/Apr 15 2020 17:20:14)
DEBUG:core:Platform is Linux-5.3.0-53-generic-x86_64-with-Ubuntu-18.04-bionic
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
^CINFO:core:Going down...
INFO:core:Down.
root@ubuntu:/home/ubuntu/pox#

```

IV-3 Install Pox

4) Check OVS(OpenVirtual Switch)

```

root@ubuntu:/home/ubuntu/pox# ovs-vsctl --version
ovs-vsctl (Open vSwitch) 2.9.5
DB Schema 7.15.1

```

IV-4 Install OVS

Lab Assessment 1):

Create a mininet based topology with 4 container hosts and one controller switches and run it.

- 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.

This problem requires us to create 4 container hosts, 1 switch, 2 controllers and create links among them.

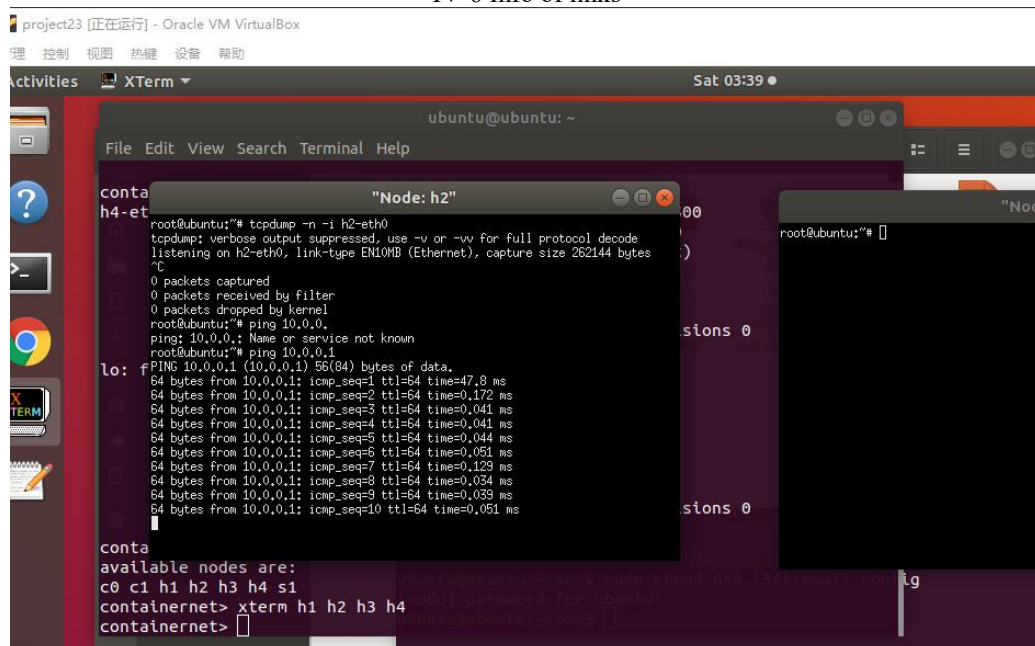
Here we use the mininet command: `sudo mn --topo=single,4 --controller=remote,port=6633 --controller=remote,port=6655 --switch=ovsk --mac`

```
File Edit View Search Terminal Help
ubuntu@ubuntu:~$ sudo mn --topo=single,4 --controller=remote,port=6633 --controller=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
Unable to contact the remote controller at 127.0.0.1:6655
*** 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>
```

IV-5 Lab CS-CNS-00101 Assessment 1

```
s1 ...
*** Starting CLI:
containernet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:h4-eth0
c0
c1
containernet>
```

IV-6 Info of links



```
project23 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Activities XTerm Sat 03:39
ubuntu@ubuntu: ~
File Edit View Search Terminal Help
containernet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s1-eth3
h4 h4-eth0:s1-eth4
s1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:h3-eth0 s1-eth4:h4-eth0
c0
c1
containernet>
containernet> xterm h1 h2 h3 h4
containernet>
```

```
root@ubuntu:~# tcpdump -n -i h2-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@ubuntu:~# ping 10.0.0.1
ping: 10.0.0.1: Name or service not known
root@ubuntu:~# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data:
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=47.8 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.172 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.041 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.041 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.044 ms
64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=0.051 ms
64 bytes from 10.0.0.1: icmp_seq=7 ttl=64 time=0.129 ms
64 bytes from 10.0.0.1: icmp_seq=8 ttl=64 time=0.034 ms
64 bytes from 10.0.0.1: icmp_seq=9 ttl=64 time=0.033 ms
64 bytes from 10.0.0.1: icmp_seq=10 ttl=64 time=0.051 ms
^C
root@ubuntu:~#
```

IV-7 Test 10.0.0.2 ping 10.0.0.1

In the figure IV-8 we can see that the required hosts, switches and controllers have been created and the links have been added among them. And a simple test of the data flow between hosts shows that it is possible to ping from h2 to h1.

Lab Assessment 2):

Make the interfaces up and assign IP addresses to interfaces of container hosts.

Assign IP address 192.168.2.10 to container host #1.

Assign IP address 192.168.2.20 to container host #2.

Assign IP address 192.168.2.30 to container host #3.

Assign IP address 192.168.2.40 to container host #4.

Here we need to assign new IP addresses to the four container network hosts created (since the initial IPs created by system default are 10.0.0.1-10.0.0.4) with the command(in mininet terminal): *h1 ifconfig h1-eth0 192.168.2.10 etc.*

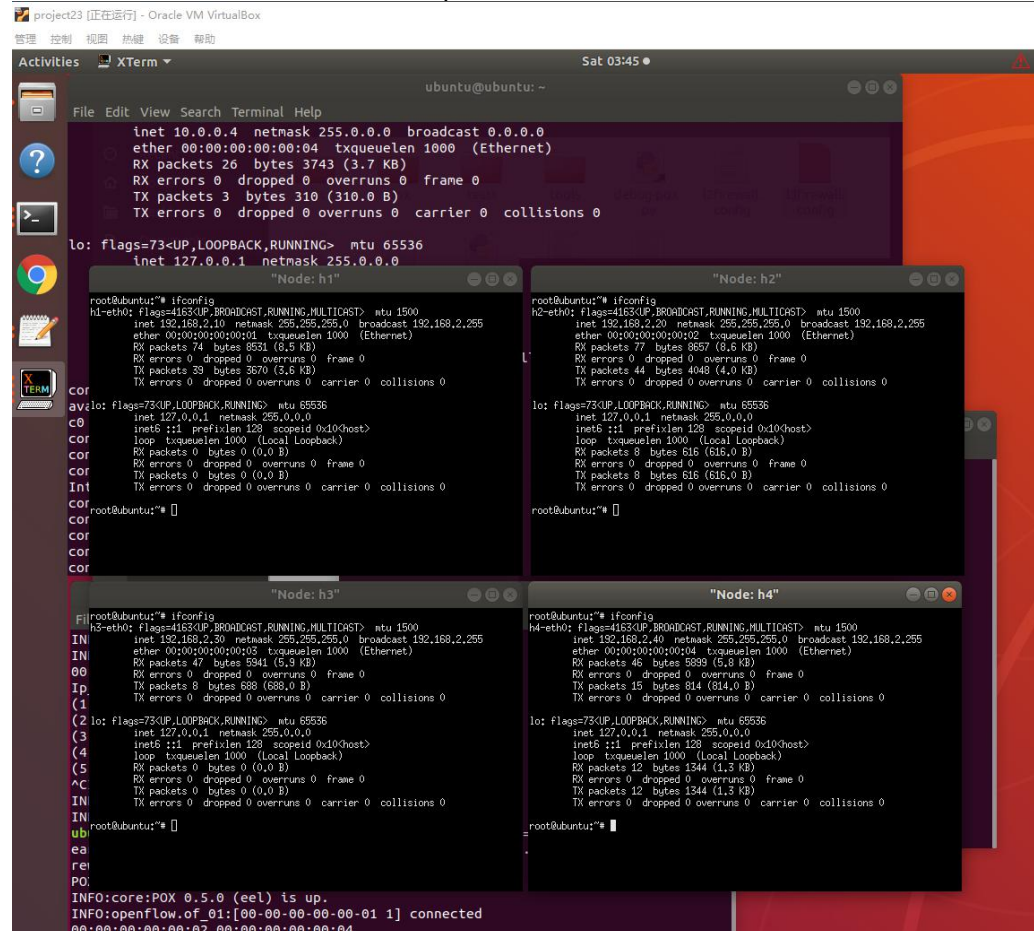
```

containernet> nodes
available nodes are:
c0 c1 h1 h2 h3 h4 s1
containernet> xterm h1 h2 h3 h4
containernet> h1 ifconfig h1-eth0 192.168.2.10
containernet>
Interrupt
containernet> h2 ifconfig h2-eth0 192.168.2.20
containernet> h3 ifconfig h3-eth0 192.168.2.30
containernet> h4 ifconfig h4-eth0 192.168.2.40
containernet> xterm h1 h2 h3 h4

```

IV-9 Assign IP addresses to container hosts

Then we can turn on all of the hosts and check the ip of each of them.



```

project23 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Activities XTerm ubuntu@ubuntu: ~ Sat 03:45
File Edit View Search Terminal Help
inet 10.0.0.4 netmask 255.0.0.0 broadcast 0.0.0.0
ether 00:00:00:00:00:04 txqueuelen 1000 (Ethernet)
RX packets 26 bytes 3743 (3.7 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

"Node: h1"
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 00:00:00:00:00:04 txqueuelen 1000 (Ethernet)
RX packets 74 bytes 8531 (8.5 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 39 bytes 3670 (3.6 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

"Node: h2"
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 00:00:00:00:00:04 txqueuelen 1000 (Ethernet)
RX packets 77 bytes 8657 (8.6 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 44 bytes 4048 (4.0 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

"Node: h3"
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 00:00:00:00:00:04 txqueuelen 1000 (Ethernet)
RX packets 47 bytes 5941 (5.9 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 8 bytes 688 (688.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

"Node: h4"
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 46 bytes 5939 (5.9 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 15 bytes 814 (814.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

IV-10 Net information of each host

Lab Assessment 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.

First, i tested the ICMP traffic from source IP 192.168.2.10 and destination IP 192.168.2.30, and here we see the traffic is not blocked.

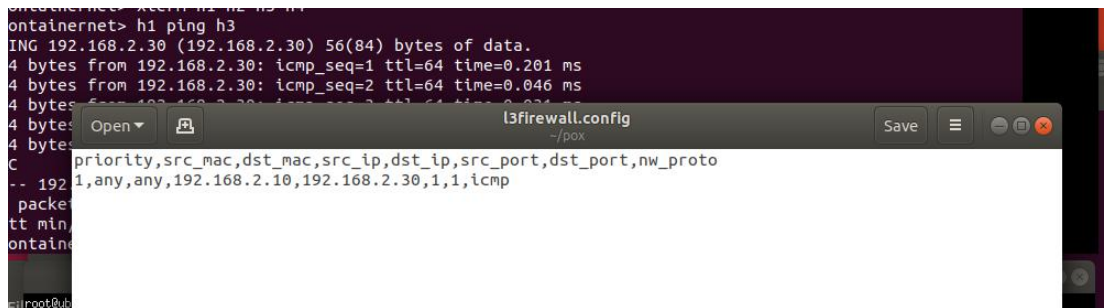
```

containernet> h1 ping 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=0.201 ms
 64 bytes from 192.168.2.30: icmp_seq=2 ttl=64 time=0.046 ms
 64 bytes from 192.168.2.30: icmp_seq=3 ttl=64 time=0.031 ms
 64 bytes from 192.168.2.30: icmp_seq=4 ttl=64 time=0.040 ms
 64 bytes from 192.168.2.30: icmp_seq=5 ttl=64 time=0.193 ms
^C
--- 192.168.2.30 ping statistics ---
 5 packets transmitted, 5 received, 0% packet loss, time 4098ms
 rtt min/avg/max/mdev = 0.031/0.102/0.201/0.077 ms
containernet>

```

IV-11 Traffic status before rule added

Then we add the rule: *1,any,any,192.168.2.10,192.168.2.30,1,1,icmp* in l3firewall.config, which means the layer 3 level added.



```

containernet> h1 ping h3
PING 192.168.2.30 (192.168.2.30) 56(84) bytes of data.
 4 bytes from 192.168.2.30: icmp_seq=1 ttl=64 time=0.201 ms
 4 bytes from 192.168.2.30: icmp_seq=2 ttl=64 time=0.046 ms
 4 bytes from 192.168.2.30: icmp_seq=3 ttl=64 time=0.031 ms
 4 bytes from 192.168.2.30: icmp_seq=4 ttl=64 time=0.040 ms
 4 bytes from 192.168.2.30: icmp_seq=5 ttl=64 time=0.193 ms
^C
--- 192.168.2.30 ping statistics ---
 5 packets transmitted, 5 received, 0% packet loss, time 4098ms
 rtt min/avg/max/mdev = 0.031/0.102/0.201/0.077 ms
containernet>

```

l3firewall.config

```

priority,src_mac,dst_mac,src_ip,dst_ip,src_port,dst_port,nw_proto
-- 192.1,any,any,192.168.2.10,192.168.2.30,1,1,icmp

```

IV-12 layer 3 rule added for assessment 3

Then i invoked pox and start these rules files and test the icmp traffic between h1 and h3.

```

INFO:core:Down.
ubuntu@ubuntu:~/pox$ sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l2_learning pox.forwarding.L3Firewall --l2config="l2firewall.config" --l3config="l3firewall.config"
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
src_ip, dst_ip, src_port, dst_port 192.168.2.10 192.168.2.30 1 1
src_ip, dst_ip, src_port, dst_port 192.168.2.20 192.168.2.40 1 1
src_ip, dst_ip, src_port, dst_port 192.168.2.20 any 1 80
src_ip, dst_ip, src_port, dst_port 192.168.2.10 192.168.2.20 1 1
src_ip, dst_ip, src_port, dst_port 192.168.2.10 192.168.2.20 1 1
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
00:00:00:00:00:02 00:00:00:00:00:04

```

IV-13 Invoke the rules

```

containernet> h1 ping h3
PING 192.168.2.30 (192.168.2.30) 56(84) bytes of data.
^C
--- 192.168.2.30 ping statistics ---
 6 packets transmitted, 0 received, 100% packet loss, time 5125ms
containernet>

```

IV-14 ICMP traffic between h1 and h3

```

ubuntu@ubuntu: ~
File Edit View Search Terminal Help
ubuntu@ubuntu:~$ sudo ovs-ofctl dump-flows s1
[sudo] password for ubuntu:
 cookie=0x0, duration=59.554s, table=0, n_packets=0, n_bytes=0, priority=65535,d
l_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04 actions=drop
 cookie=0x0, duration=51.475s, table=0, n_packets=0, n_bytes=0, idle_timeout=200
, priority=50004,tcp,nw_src=192.168.2.10,nw_dst=192.168.2.20,tp_src=1,tp_dst=1 a
ctions=drop
 cookie=0x0, duration=51.475s, table=0, n_packets=0, n_bytes=0, idle_timeout=200
, priority=50005,udp,nw_src=192.168.2.10,nw_dst=192.168.2.20,tp_src=1,tp_dst=1 a
ctions=drop
 cookie=0x0, duration=51.475s, table=0, n_packets=0, n_bytes=0, idle_timeout=200
, priority=50003,tcp,nw_src=192.168.2.20,tp_src=1,tp_dst=80 actions=drop
 cookie=0x0, duration=51.476s, table=0, n_packets=19, n_bytes=1862, idle_timeout
=200, priority=50001,icmp,nw_src=192.168.2.10,nw_dst=192.168.2.30 actions=drop
 cookie=0x0, duration=51.475s, table=0, n_packets=0, n_bytes=0, idle_timeout=200
, priority=50002,icmp,nw_src=192.168.2.20,nw_dst=192.168.2.40 actions=drop
 cookie=0x0, duration=51.475s, table=0, n_packets=5, n_bytes=266, actions=NORMAL
ubuntu@ubuntu:~$

```

IV-15 dumpflows detail

Here the ICMP traffic from h1 to h3 has been blocked.

Lab Assessment 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.

The operation of this question is almost the same as the previous one. First test the ICMP traffic before the rule is added. Then we add the rule 2 in layer 3 level.

```

containernet> h2 ping h4
PING 192.168.2.40 (192.168.2.40) 56(84) bytes of data.
64 bytes from 192.168.2.40: icmp_seq=1 ttl=64 time=57.9 ms
64 bytes from 192.168.2.40: icmp_seq=2 ttl=64 time=0.041 ms
64 bytes from 192.168.2.40: icmp_seq=3 ttl=64 time=0.042 ms
64 bytes from 192.168.2.40: icmp_seq=4 ttl=64 time=0.042 ms
64 bytes from 192.168.2.40: icmp_seq=5 ttl=64 time=0.040 ms

```

IV-16 Traffic status before rule 2 added

```

Open [icon] *l3firewall.config ~/pox
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

```

IV-17 Rule 2 be added

Retest h2 ping h4 and its traffic after activating the rule and find that the rule is enabled successfully.

```

--- 192.168.2.40 ping statistics ---
57 packets transmitted, 6 received, 89% packet loss,
rtt min/avg/max/mdev = 0.040/25.727/96.300/37.980 ms
containernet> h2 ping h4
PING 192.168.2.40 (192.168.2.40) 56(84) bytes of data
^C
--- 192.168.2.40 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss,
containernet>
 cookie=0x0, duration=51.475s, table=0, n_packets=5, n_bytes=266, actions=NORMAL
ubuntu@ubuntu:~$ sudo ovs-ofctl dump-flows s1
 cookie=0x0, duration=42.471s, table=0, n_packets=23, n_bytes=1750, priority=655
35,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04 actions=drop
 cookie=0x0, duration=33.891s, table=0, n_packets=0, n_bytes=0, idle_timeout=200
, priority=50001,icmp,nw_src=192.168.2.10,nw_dst=192.168.2.30 actions=drop
 cookie=0x0, duration=33.847s, table=0, n_packets=3, n_bytes=294, idle_timeout=2
00, priority=50002,icmp,nw_src=192.168.2.20,nw_dst=192.168.2.40 actions=drop
 cookie=0x0, duration=33.847s, table=0, n_packets=4, n_bytes=224, actions=NORMAL
ubuntu@ubuntu:~$

```

IV-18 ICMP traffic and flow of rule 2

Lab Assessment 5):

Add new rule to l3config file for blocking HTTP traffic from source IP 192.168.2.20.

This question requires us to block all http traffic from the source address 192.168.2.20. This means we need to block the host with source IP 192.168.2.20 from accessing all other hosts on port 80 of the tcp protocol (HTTP). So let's test h2 and other machines for http traffic at the beginning as usual.

Here we have created a python-based http service in h1 and opened its port 80. Another way to test this is to use the hping3 command in a mininet window. The result is the same. Here you can see that h2 can access the http service of h1.

```

57 packets transmitted, 6 received, 89% packet loss, time 57321ms
rtt min/avg/max/mdev = 0.040/25.727/96.300/37.980 ms
containernet> h2 ping h4
Node: h1
root@ubuntu:~# python -v
Unknown option: -v
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Try 'python -h' for more information.
root@ubuntu:~# python --version
Python 2.7.12
root@ubuntu:~# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
192.168.2.40 - - [05/Mar/2022 04:28:05] "GET / HTTP/1.1" 200 -
192.168.2.20 - - [05/Mar/2022 04:28:14] "GET / HTTP/1.1" 200 -
Node: h4
rtt min/avg/max/mdev = 0.037/0.116/0.195/0.079 ms
root@ubuntu:~# curl 192.168.2.10
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href=".ansible/">.ansible/</a>
<li><a href=".bash_history">.bash_history</a>
<li><a href=".bash_logout">.bash_logout</a>
<li><a href=".bashrc">.bashrc</a>
<li><a href=".cache/">.cache/</a>
<li><a href=".config/">.config/</a>
<li><a href=".gnome/">.gnome/</a>
<li><a href=".gnupg/">.gnupg/</a>
<li><a href=".ICEauthority">.ICEauthority</a>
<li><a href=".local/">.local/</a>
<li><a href=".mininet_history">.mininet_history</a>
<li><a href=".mozilla/">.mozilla/</a>
<li><a href=".pki/">.pki/</a>
<li><a href=".profile">.profile</a>
<li><a href=".python_history">.python_history</a>
<li><a href=".ssh/">.ssh/</a>
</ul>
</body>
</html>
Node: h2
<li><a href=".wget-hsts">.wget-hsts</a>
<li><a href=".wireshark/">.wireshark/</a>
<li><a href=".MacOSX/">.MacOSX/</a>
<li><a href=".containernet/">.containernet/</a>
<li><a href=".Desktop/">.Desktop/</a>
<li><a href=".Documents/">.Documents/</a>
<li><a href=".Downloads/">.Downloads/</a>
<li><a href=".examples.desktop">.examples.desktop</a>
<li><a href=".lab-cs-cns-00101/">.lab-cs-cns-00101/</a>
<li><a href=".lab-cs-cns-00101.zip">.lab-cs-cns-00101.zip</a>
<li><a href=".Music/">.Music/</a>
<li><a href=".oflops/">.oflops/</a>
<li><a href=".oftest/">.oftest/</a>
<li><a href=".openflow/">.openflow/</a>
<li><a href=".Pictures/">.Pictures/</a>
<li><a href=".pox/">.pox/</a>
<li><a href=".Public/">.Public/</a>
<li><a href=".Templates/">.Templates/</a>
<li><a href=".Videos/">.Videos/</a>
</ul>
</body>
</html>
root@ubuntu:~#
--- 192.168.2.30 ping statistics ---
2 pings: 0.000 ms, 0.000 ms, 0.000 ms
Node: h3
ip,src_port,dst_port,nw_proto
1,1,icmp
t=6655 pox.forwarding.l2_l
l.config" --l3config="l3fi

```

IV-19 HTTP traffic test before rule added

Then we start a new rule: `3,any,any,192.168.2.20,any,any,80,tcp`, after the new rule is started we first test the default tcp traffic for h2 and h3 and find that it is successful, then we test the traffic for h2 and h3 based on port 80 (http) and find that it fails. This means that our new rule was successfully used to block HTTP traffic from h2.

```

containernet> h2 hping3 -c 5 h3 -V --tcp-timestamp
using h2-eth0, addr: 192.168.2.20, MTU: 1500
HPING 192.168.2.30 (h2-eth0 192.168.2.30): NO FLAGS are set, 40 headers + 0 data bytes
len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
sport=0 flags=RA seq=0 win=0 rtt=4.9 ms
seq=0 ack=4754081 sum=3f94 urp=0

len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
sport=0 flags=RA seq=1 win=0 rtt=9.8 ms
seq=0 ack=609912122 sum=9375 urp=0

len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
sport=0 flags=RA seq=2 win=0 rtt=9.8 ms
seq=0 ack=1073496225 sum=899e urp=0

len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
sport=0 flags=RA seq=3 win=0 rtt=2.0 ms
seq=0 ack=1343786963 sum=3a3f urp=0

^C
--- 192.168.2.30 hping statistic ---
4 packets transmitted, 4 packets received, 0% packet loss

```

IV-20 Other port traffic was not blocked

```

containernetwork> h2 hping3 -c 5 --destport 80 h3 -V
using h2-eth0, addr: 192.168.2.20, MTU: 1500
HPING 192.168.2.30 (h2-eth0 192.168.2.30): NO FLAGS are set, 40 headers + 0 data bytes
len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
sport=80 flags=RA seq=0 win=0 rtt=3.8 ms
seq=0 ack=713318352 sum=bf94 urp=0

len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
sport=80 flags=RA seq=1 win=0 rtt=6.9 ms
seq=0 ack=1980266918 sum=642f urp=0

len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
sport=80 flags=RA seq=2 win=0 rtt=6.6 ms
seq=0 ack=1001713094 sum=96f5 urp=0

^C
--- 192.168.2.30 hping statistic ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 3.8/5.7/6.9 ms
containernetwork> h2 hping3 --destport 80 -c 5 h3 -V
using h2-eth0, addr: 192.168.2.20, MTU: 1500
HPING 192.168.2.30 (h2-eth0 192.168.2.30): NO FLAGS are set, 40 headers + 0 data bytes
len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
sport=80 flags=RA seq=0 win=0 rtt=12.6 ms
seq=0 ack=243681397 sum=fdce urp=0

--- 192.168.2.30 hping statistic ---
5 packets transmitted, 1 packets received, 80% packet loss
containernetwork> h2 hping3 --destport 80 -c 5 h1 -V
using h2-eth0, addr: 192.168.2.20, MTU: 1500
HPING 192.168.2.10 (h2-eth0 192.168.2.10): NO FLAGS are set, 40 headers + 0 data bytes

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

```

IV-21 Only HTTP traffic from h2 has been blocked

Lab Assessment 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`.

To complete this question, I need to create a new mac address-based policy in the layer 2 level. From the question we can see that we need to block all information from h2 to h4. So we can test the ICMP, TCP and UDP traffic after the new rule is implemented. The rule in layer 2 can be: `1,00:00:00:00:00:02,00:00:00:00:00:04`

```

containernetwork> h2 ping h4
PING 192.168.2.40 (192.168.2.40) 56(84) bytes of data.
^C
--- 192.168.2.40 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5147ms

containernetwork> h2 hping3 --udp -c 5 h4 -V
using h2-eth0, addr: 192.168.2.20, MTU: 1500
HPING 192.168.2.40 (h2-eth0 192.168.2.40): udp mode set, 28 headers + 0 data bytes

--- 192.168.2.40 hping statistic ---
5 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
containernetwork> h2 hping3 -c 5 h4 -V --tcp-timestamp
using h2-eth0, addr: 192.168.2.20, MTU: 1500
HPING 192.168.2.40 (h2-eth0 192.168.2.40): NO FLAGS are set, 40 headers + 0 data bytes

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

```



```

2 root@ubuntu:~# hping3 --udp 192.168.2.20
HPING 192.168.2.20 (h2-eth0 192.168.2.20): udp mode set, 28 headers + 0 data byt
4es
^C
SI --- 192.168.2.20 hping statistic ---
/ 7 packets transmitted, 0 packets received, 100% packet loss
x round-trip min/avg/max = 0.0/0.0/0.0 ms
root@ubuntu:~# hping3 --udp 192.168.2.40
HPING 192.168.2.40 (h2-eth0 192.168.2.40): udp mode set, 28 headers + 0 data byt
4es
^C
SI --- 192.168.2.40 hping statistic ---
/ 6 packets transmitted, 0 packets received, 100% packet loss
x round-trip min/avg/max = 0.0/0.0/0.0 ms
root@ubuntu:~# hping3 --udp 192.168.2.30
HPING 192.168.2.30 (h2-eth0 192.168.2.30): udp mode set, 28 headers + 0 data byt
4es
hies
ICMP Port Unreachable from ip=192.168.2.30 name=UNKNOWN
status=0 port=2111 seq=0
2 ICMP Port Unreachable from ip=192.168.2.30 name=UNKNOWN
status=0 port=2112 seq=1
4 ICMP Port Unreachable from ip=192.168.2.30 name=UNKNOWN
status=0 port=2113 seq=2
SI ^C

```

IV-22 ICMP,UDP,TCP traffic are all be blocked from h2 to h4

Lab Assessment 7):

Add new rule to *l3config* file for blocking tcp traffic from 192.168.2.10 to 192.168.2.20.

Here we need to block all TCP traffic from h1 to h2, so it is all ports in the new rule.

New rule is: 4,any,any,192.168.2.10,192.168.2.20,any,any,tcp

Then we test traffic by command: *h1 hping3 -c 5 h2 -V --tcp-timestamp*. The test shows that h1 can go to h3 for tcp traffic but tcp traffic with h2 is blocked.

```

containernet> h1 hping3 -c 5 h3 -V --tcp-timestamp
using h1-eth0, addr: 192.168.2.10, MTU: 1500
HPING 192.168.2.30 (h1-eth0 192.168.2.30): NO FLAGS are set, 40 headers + 0 data bytes
len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
sport=0 flags=RA seq=0 win=0 rtt=8.6 ms
seq=0 ack=2009947681 sum=924a urp=0

len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
sport=0 flags=RA seq=1 win=0 rtt=18.5 ms
seq=0 ack=1179259001 sum=bcc9 urp=0

len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
sport=0 flags=RA seq=2 win=0 rtt=8.7 ms
seq=0 ack=971008917 sum=97b urp=0

ef len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
ef sport=0 flags=RA seq=3 win=0 rtt=8.8 ms
ef seq=0 ack=1397775461 sum=c870 urp=0
ef
ef len=40 ip=192.168.2.30 ttl=64 DF id=0 tos=0 iplen=40
ef sport=0 flags=RA seq=4 win=0 rtt=10.6 ms
ef seq=0 ack=603000335 sum=7cee urp=0
ef
ef --- 192.168.2.30 hping statistic ---
ef 5 packets transmitted, 5 packets received, 0% packet loss
ef round-trip min/avg/max = 8.6/11.0/18.5 ms
ef containernet> h1 hping3 -c 5 h2 -V --tcp-timestamp
using h1-eth0, addr: 192.168.2.10, MTU: 1500
HPING 192.168.2.20 (h1-eth0 192.168.2.20): NO FLAGS are set, 40 headers + 0 data bytes
tu --- 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
containernet>

```

IV-23 TCP traffic from h1 to h3 and h1 to h2

Lab Assessment 8):

Add new rule to *l3config* file for blocking udp traffic from 192.168.2.10 to 192.168.2.20.

Here we need to block all UDP traffic from h1 to h2, so it is all ports in the new rule.

New rule is: 5,any,any,192.168.2.10,192.168.2.20,any,any,udp

Then we test traffic by command: *h1 hping3 --udp -c 5 h2 -V*. The test shows that h1 can go to h3 for UDP traffic but UDP traffic with h2 is blocked.

```

containernet> h1 hping3 --udp -c 5 h3 -V
using h1-eth0, addr: 192.168.2.10, MTU: 1500
HPING 192.168.2.30 (h1-eth0 192.168.2.30): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from ip=192.168.2.30 name=UNKNOWN
status=0 port=1200 seq=0
ICMP Port Unreachable from ip=192.168.2.30 name=UNKNOWN
status=0 port=1201 seq=1
ICMP Port Unreachable from ip=192.168.2.30 name=UNKNOWN
status=0 port=1202 seq=2
ICMP Port Unreachable from ip=192.168.2.30 name=UNKNOWN
status=0 port=1203 seq=3
ICMP Port Unreachable from ip=192.168.2.30 name=UNKNOWN
status=0 port=1204 seq=4

--- 192.168.2.30 hping statistic ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 6.0/15.4/28.0 ms
containernet> h1 hping3 --udp -c 5 h2 -V
using h1-eth0, addr: 192.168.2.10, MTU: 1500
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
containernet>

```

IV-24 UDP traffic from h1 to h3 and h1 to h2

V. CONCLUSION

In this experiment, from the installation and setup of a series of software such as mininet, pox and ovs, to the setup of SDN-based openflow firewall rules. I started to build and learn step by step, and initially mastered and understood the protection for computer security in SDN environment. I can set up rules to allow, deny and forward data flow traffic. By setting these rules, we can effectively protect the security of the computer. And in which I also found that when waking up the pox rules, there is often the problem that the rules fail when operating in mininet. At this point I need to restart the rule to take effect. Guess it may be due to a request timeout at some level of the pox call source code. Adding an error report to the pox code at this point can effectively indicate that status.

VI. APPENDIX B: ATTACHED FILES

l2firewall.config	https://github.com/MatthewLLLL/Adv-Network-Security/blob/main/Project%20l2firewall.config
l3firewall.config	https://github.com/MatthewLLLL/Adv-Network-Security/blob/main/Project%20l3firewall.config
L3Firewall.py	https://github.com/MatthewLLLL/Adv-Network-Security/blob/main/Project%20L3Firewall.py

VII. REFERENCES

- [1] Hping3 guide book, available at <https://linux.die.net/man/8/hping3/>.
- [2] Pox at <https://noxrepo.github.io/pox-doc/html/>.
- [3] Mininet at <http://mininet.org/walkthrough/>