Mohammed VI Polytechnic University
School of Computer Science

Advanced Network Programming
Lab 3 – Practical
Building a Firewall

Instructions:

For this lab activity you will create a simple firewall architecture using an SDN controller and OpenFlow-enabled switches. A firewall provides security by not letting specified traffic pass through it. This feature is good for minimizing attack vectors and limiting the network "surface" exposed to attackers.

SDN Controller:
For this lab, you can use any type of SDN controller (POX, RYU, NOX etc.), to create Firewall application. We have used POX controller in our examples and instructions.

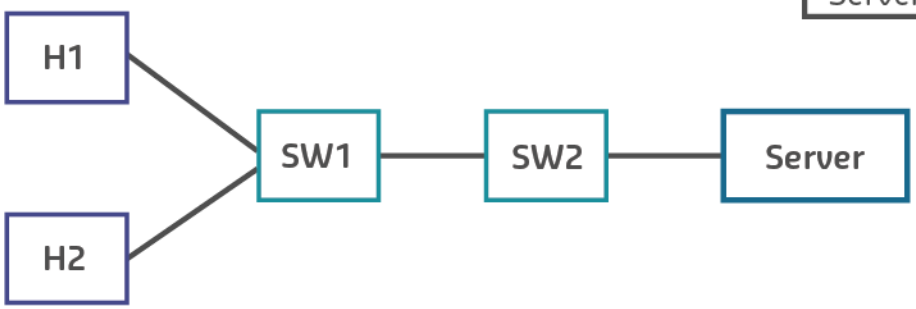
More details of the POX controller can be found here: https://noxrepo.github.io/pox-doc/html/

OpenFlow:
In this lab, we will explore how OpenFlow rules can be installed in switches to control the network traffic.

More details of the OpenFlow protocol, can be found here:
https://opennetworking.org/software-defined-standards/specifications/

Topology:

H1	10.0.0.1
H2	10.0.0.2
Server	10.0.0.3



Task 1: Create the Firewall Application with the Firewall Rules:

The rules that you will need to implement in OpenFlow switches for this task are:

Source IP	Destination IP	Protocol	Action
H1	Server	TCP	Accept
H2	Server	TCP	Accept
Any	Any	ARP	Accept
Any IPv4	Any IPv4	-	Drop

Basically, your Firewall application should allow all ARP and TCP traffic to pass. However, any other type of traffic should be dropped.

Be careful! Flow tables match the rule with highest priority first, where priority is established based on the order rules are placed in the table.

In this lab, we will try the **Proactive Controller approach** – All OpenFlow rules should be installed at the beginning of network initialization. You can use any type of SDN controller (POX, RYU, NOX etc.) we have given instructions for POX controller.

Create "Firewall" application with necessary codes to install OpenFlow rules and run with the selected controller to install all OpenFlow rules.

1. Setup the Mininet VM

This has been done in Lab 0.

2. Install POX controller in Mininet VM

The Mininet VM comes with POX installed.

3. Create Firewall.py application to install OpenFlow rules inside the pox directory.

The file will be provided by the instructor or you can use the file from Lab 2 and modify it as per the code shown below. Rename it appropriately.

More details of the OpenFlow rules, can be found at https://opennetworking.org/software-defined-standards/specifications/

OpenFlow rules for Switch 1:

Allow all ARP traffic

Allow only TCP traffic

```
#flood ARP packets
msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x806),action=of.ofp_action_output(port=of.OFPP_ALL))
event.connection.send(msg)

msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.1",nw_dst="10.0.0.3",nw_proto=6),action=of.ofp_action_output(port=3))
event.connection.send(msg)

msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.2",nw_dst="10.0.0.3",nw_proto=6),action=of.ofp_action_output(port=3))
event.connection.send(msg)

msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.3",nw_dst="10.0.0.1",nw_proto=6),action=of.ofp_action_output(port=1))
event.connection.send(msg)

msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.3",nw_dst="10.0.0.2",nw_proto=6),action=of.ofp_action_output(port=2))
event.connection.send(msg)
```

OpenFlow rules for Switch 2:

```
#flood ARP packets
msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x806),action=of.ofp_action_output(port=of.OFPP_ALL))
event.connection.send(msg)

msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.1",nw_dst="10.0.0.3",nw_proto=6),action=of.ofp_action_output(port=2)
event.connection.send(msg)

msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.2",nw_dst="10.0.0.3",nw_proto=6),action=of.ofp_action_output(port=2)
event.connection.send(msg)

msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.3",nw_dst="10.0.0.1",nw_proto=6),action=of.ofp_action_output(port=1)
event.connection.send(msg)

msg = of.ofp_flow_mod(match=of.ofp_match(dl_type=0x800,nw_src="10.0.0.3",nw_dst="10.0.0.2",nw_proto=6),action=of.ofp_action_output(port=1)
event.connection.send(msg)
```

3. Run your Firewall.py application with the controller (pox).

mininet@mininet-vm:~/pox$ sudo python ./pox.py Firewall

```
mininet@mininet-vm:~/pox$ sudo python ./pox.py Firewall
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
```

Task 2: Create the given topology in Mininet:

1. Create a topology application (FirewallTopo.py) according to the given topology diagram.

mininet@mininet-vm:~$ cd SDN

The file will be provided by the instructor or you can use the file from Lab 2 and modify it as per the code shown below. Rename it appropriately.

```python
def emptyNet():

        net = Mininet(controller=RemoteController, switch=OVSKernelSwitch)

        c1 = net.addController('c1', controller=RemoteController, ip="127.0.0.1", port=6633)

        h1 = net.addHost( 'h1', ip='10.0.0.1' )
        h2 = net.addHost( 'h2', ip='10.0.0.2' )
        h3 = net.addHost( 'Server', ip='10.0.0.3' )

        s1 = net.addSwitch( 's1' )
        s2 = net.addSwitch( 's2' )

        s1.linkTo( h1 )
        s1.linkTo( h2 )
        s1.linkTo( s2 )

        s2.linkTo( h3 )

        net.build()
        c1.start()
        s1.start([c1])
        s2.start([c1])

        CLI( net )
        net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    emptyNet()
```

2. Run your topology by using the command below.

mininet@mininet-vm:~/SDN$ sudo python ./FirewallTopo.py

```
mininet@mininet-vm:~/mininet$ sudo python FirewallTopo.py
*** Configuring hosts
h1 h2 Server
*** Starting CLI:
mininet>
```

Task 3: Testing

1. Use “net” command in mininet to verify the topology.

```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
Server Server-eth0:s2-eth2
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0 s1-eth3:s2-eth1
s2 lo:  s2-eth1:s1-eth3 s2-eth2:Server-eth0
c1
mininet>
```

2. Check the controller prompt to verify switch connections to the controller.

```
mininet@mininet-vm:~/pox$ sudo python ./pox.py Firewall
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
```

3. Use “sudo ovs-ofctl dump-flows s1” to verify the commands installed in the switch **s1** (Note: Since we are implementing a proactive controller, all rules should be installed in the switches before initiation of any communication).

More details of the OpenVSwitch based commands, can be found here:
http://www.openvswitch.org/support/dist-docs/ovs-ofctl.8.html

```
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=320.91s, table=0, n_packets=0, n_bytes=0, idle_age=320, arp actions=ALL
 cookie=0x0, duration=320.87s, table=0, n_packets=0, n_bytes=0, idle_age=320, tcp,nw_src=10.0.0.2,nw_dst=10.0
.0.3 actions=output:3
 cookie=0x0, duration=320.87s, table=0, n_packets=0, n_bytes=0, idle_age=320, tcp,nw_src=10.0.0.1,nw_dst=10.0
.0.3 actions=output:3
 cookie=0x0, duration=320.87s, table=0, n_packets=0, n_bytes=0, idle_age=320, tcp,nw_src=10.0.0.3,nw_dst=10.0
.0.2 actions=output:2
 cookie=0x0, duration=320.87s, table=0, n_packets=0, n_bytes=0, idle_age=320, tcp,nw_src=10.0.0.3,nw_dst=10.0
.0.1 actions=output:1
mininet@mininet-vm:~$
```

4. Try to ping from h1 to server: This should fail, since only TCP traffic is allowed (ICMP traffic is blocked)

```
mininet> h1 ping -c2 10.0.0.3
```

5. Use “iperf” to ensure TCP traffic is allowed from h1 and h2 to server. More details on the iperf command, can be found here: https://iperf.fr/

   I. Start iperf server in Server machine
   #iperf -s

   II. Start iperf client in H1 and verify the data transfer
   #iperf -c 10.0.0.3