CO515: Advances in Computer Networks: Selected Topics – Lab08 (Mininet Lab 05)

E/19/446: Wijerathna I.M.K.D.I.

30/05/2024

Implementing a Firewall with ACLs in OpenFlow Using Mininet OpenFlow

Install Mininet and Pox (Using Pox SDN Controller to implement the Firewall)

```
vagrant@sdn-box:~$ sudo apt-get update
Ign http://archive.ubuntu.com trusty InRelease
Get:1 http://security.ubuntu.com trusty-security InRelease [56.4 kB]
Get:2 http://archive.ubuntu.com trusty-updates InRelease [56.5 kB]
Hit http://archive.ubuntu.com trusty Release.gpg
Get:3 http://security.ubuntu.com trusty-security/main Sources [254 kB]
Get:4 http://archive.ubuntu.com trusty-updates/main Sources [667 kB]
Get:5 http://security.ubuntu.com trusty-security/universe Sources [154 kB]
Get:6 http://archive.ubuntu.com trusty-updates/universe Sources [356 kB]
Get:7 http://security.ubuntu.com trusty-security/main amd64 Packages [702 kB]
Get:8 http://archive.ubuntu.com trusty-updates/main amd64 Packages [1,172 kB]
```

Figure 1: Updating the OS

```
vagrant@sdn-box:~$ sudo apt-get install -y mininet
Reading package lists... Done
Building dependency tree
Reading state information... Done
Recommended packages:
  openvswitch-controller
The following NEW packages will be installed:
  mininet
```

Figure 2: Installation of Mininet

```
vagrant@sdn-box:~$ cd pox
vagrant@sdn-box:~/pox$ ls
debug-pox.py  ext  LICENSE  NOTICE  pox  pox.py  README  setup.cfg  tests  tools
vagrant@sdn-box:~/pox$
```

Figure 3: Ensuring the Download of the Pox Repository

Lab Environment Setup

I used the same setup as the previous labs (Mininet Lab1,2,3,4) which is a lightweight ubuntu os setup which runs over vagrant+virtual box. The following is the lab environment structure.

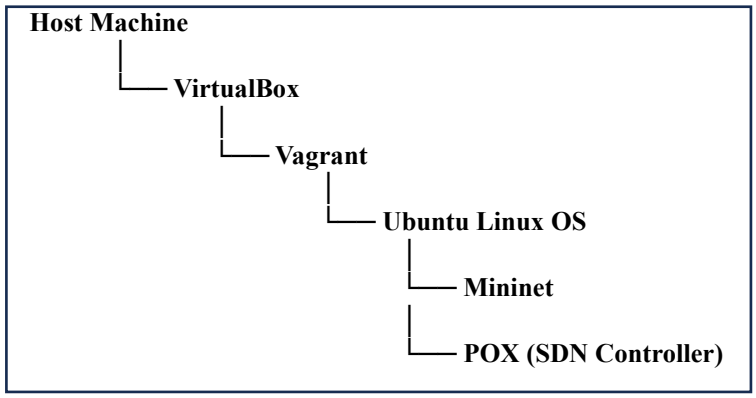

Figure 4: Lab Environment Setup

Create the Basic Mininet Topology (1 switch and 3 hosts)

Even though, it was given to try with 2-hosts topology, I have implemented a 3-host topology in order to demonstrate different ACL rules been applied to the OpenFlow switch.

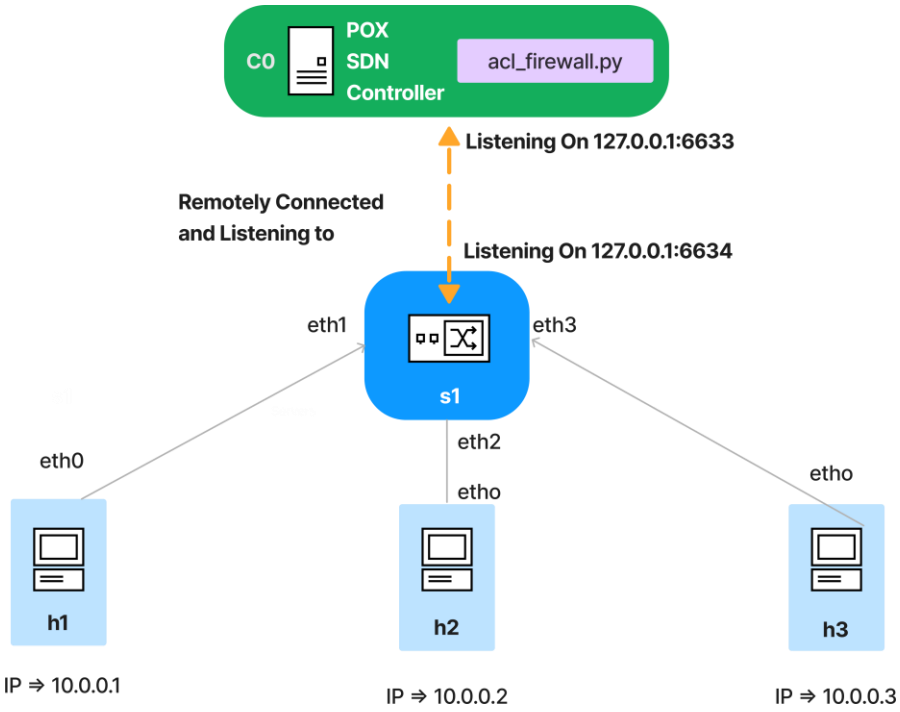

Figure 5: The mininet Topology

Create the Mininet Topology

Creating the topology and initially testing pingall without connecting the Pox SDN controller.

```
vagrant@sdn-box:~$ sudo mn --topo single,3 --mac --switch ovsk,protocols=OpenFlow10 --controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X
h2 -> X X
h3 -> X X
*** Results: 100% dropped (0/6 received)
```

Figure 6: Staring the mininet topology and pingall without connecting the Pox Controller

Configure ACLs to Allow or Block Traffic Based on Source IP address

Python code for acl_firewall.py

- *This configuration sets the ACL rules to deny traffic from h2 and allow traffic from other hosts h1,h3.*

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.packet.ethernet import ethernet
from pox.lib.packet.ipv4 import ipv4
from pox.lib.addresses import IPAddr

# Logger for outputting information and debug messages
log = core.getLogger()

# Define Access Control List (ACL) rules
acl_rules = [
    {'src_ip': IPAddr("10.0.0.1"), 'action': 'permit'},  # Allow traffic from h1
    {'src_ip': IPAddr("10.0.0.2"), 'action': 'deny'},    # Deny traffic from h2
    {'src_ip': IPAddr("10.0.0.3"), 'action': 'permit'},  # Allow traffic from h3
]

class ACLFirewall(object):
    def __init__(self):
        # Add listeners for OpenFlow events
        core.openflow.addListeners(self)
        # Dictionary to store MAC to port mappings for each switch
        self.mac_to_port = {}
        # Add listener for switch features and packet-in events
        core.openflow.addListenerByName("ConnectionUp", self.switch_features_handler)
        core.openflow.addListenerByName("PacketIn", self.packet_in_handler)

    def switch_features_handler(self, event):
        # Log when a switch connects to the controller
        log.info("Switch %s has connected", event.dpid)
        # Set the table-miss flow entry to send unknown packets to the controller
        msg = of.ofp_flow_mod()
        msg.priority = 0  # Lowest priority for table-miss flow
        msg.match = of.ofp_match()  # Match all packets
        # Action to send packets to the controller
        msg.actions.append(of.ofp_action_output(port=of.OFPP_CONTROLLER))
        event.connection.send(msg)
        log.info("Set table-miss flow on switch %s", event.dpid)

    def packet_in_handler(self, event):
        packet = event.parsed
        if not packet.parsed:
            # Log and ignore incomplete packets
            log.warning("Ignoring incomplete packet")
            return
```

```
        # Update MAC to port mapping for the switch
        self.mac_to_port[event.dpid] = self.mac_to_port.get(event.dpid, {})
        self.mac_to_port[event.dpid][packet.src] = event.port

        # Check if the packet is an IPv4 packet
        ipv4_packet = packet.find('ipv4')
        if ipv4_packet:
            # Apply ACL rules to the IPv4 packet
            if self.apply_acl_rules(ipv4_packet):
                log.info("Dropping packet from %s", ipv4_packet.srcip)
                return

        # If the packet is not dropped, flood it to all ports
        self.flood_packet(event, packet)

    def apply_acl_rules(self, ipv4_packet):
        # Check if the IPv4 packet matches any deny rules in the ACL
        for rule in acl_rules:
            if ipv4_packet.srcip == rule['src_ip'] and rule['action'] == 'deny':
                return True
        # Permit by default if no deny rule matches
        return False

    def flood_packet(self, event, packet):
        # Create a packet out message to flood the packet to all ports
        msg = of.ofp_packet_out()
        msg.data = event.ofp  # Use the original packet data
        # Add action to flood the packet
        msg.actions.append(of.ofp_action_output(port=of.OFPP_FLOOD))
        event.connection.send(msg)
        log.info("Flooding packet from %s", packet.src)

def launch():
    # Register the ACLFirewall class with the POX core
    core.registerNew(ACLFirewall)
    log.info("ACL Firewall module running")
```

Tasks

1. **Test packet switching again by using ping or other connectivity checks.**
2. **Verify that traffic from IP_x to h2 is blocked**

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3
h2 -> X X
h3 -> h1 X
*** Results: 66% dropped (2/6 received)
mininet>
```

Figure 7: pingall with the POX controller + acl_firewall.py

The above pingall results verify that h2 is blocked from any IP_X while other hosts can communicate within the network without any issue. The following output is the debug terminal of the pox controller and it also clearly highlights the packets from h2(10.0.0.2) been dropped.

```
vagrant@sdn-box:~/pox$ sudo vim ./pox/forwarding/acl_firewall.py
vagrant@sdn-box:~/pox$ ./pox.py log.level --DEBUG forwarding.acl_firewall
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:forwarding.acl_firewall:ACL Firewall module running
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-49-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:forwarding.acl_firewall:Switch 1 has connected
INFO:forwarding.acl_firewall:Set table-miss flow on switch 1
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:01
INFO:forwarding.acl_firewall:Dropping packet from 10.0.0.2
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:01
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:02
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:01
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:03
INFO:forwarding.acl_firewall:Dropping packet from 10.0.0.2
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:01
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:03
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:02
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:01
INFO:forwarding.acl_firewall:Dropping packet from 10.0.0.2
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:02
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:03
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:03
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:01
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:03
INFO:forwarding.acl_firewall:Dropping packet from 10.0.0.2
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:03
INFO:forwarding.acl_firewall:Flooding packet from 00:00:00:00:00:01
```

Figure 8: Debug terminal output of the Pox controller

To Elaborate, the following are the wireshark captured outputs taken while the pingall

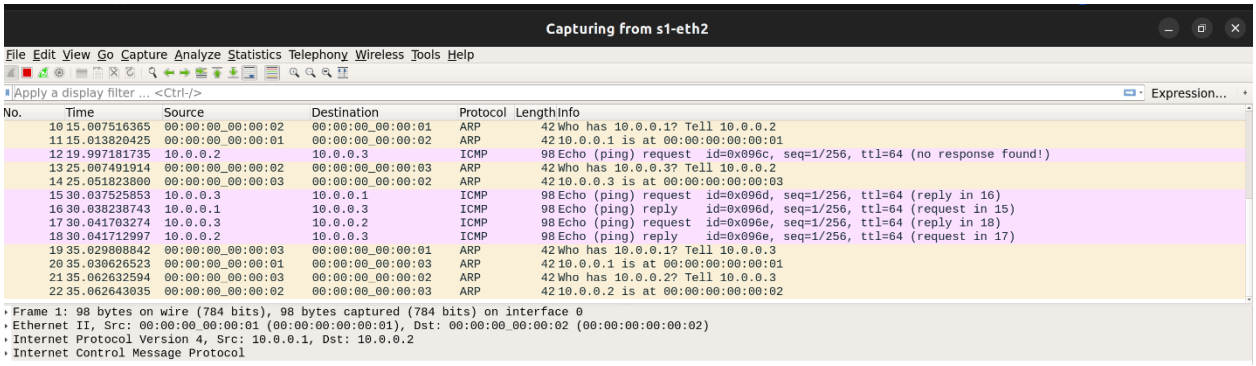

Figure 9: Wireshark output at OpenFlow switch's port-2 (port connected to host2)

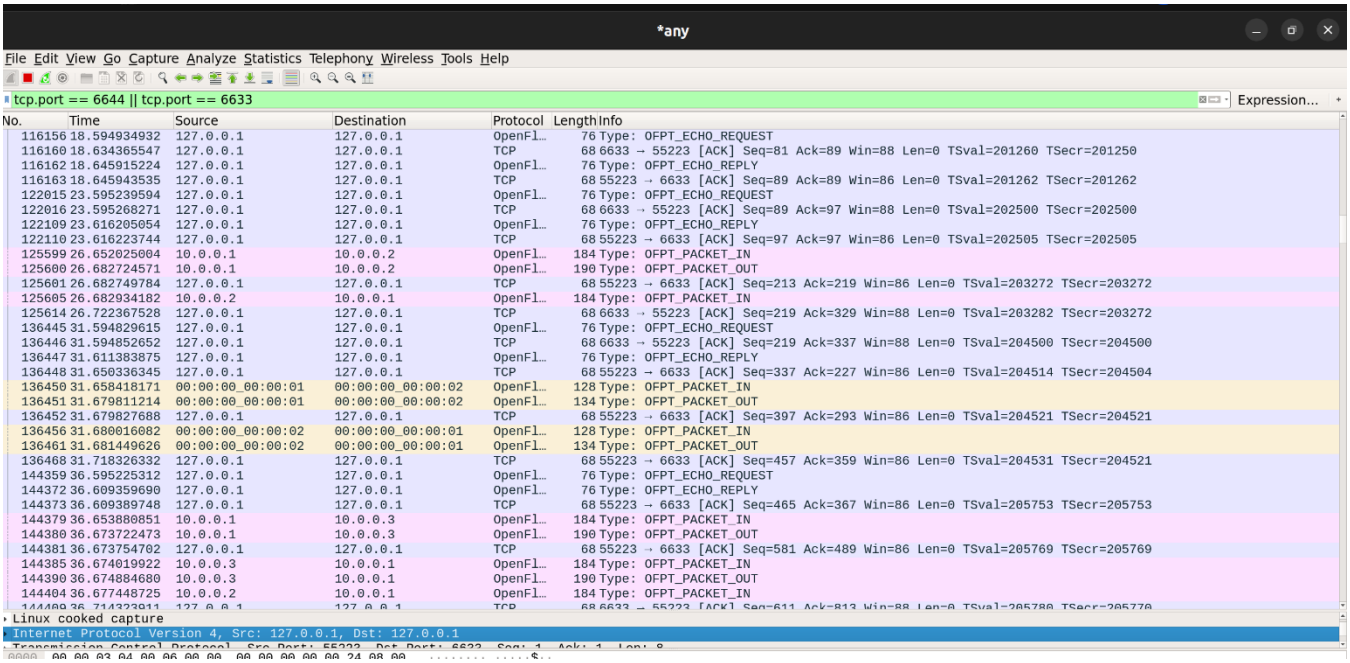

Figure 10: Wireshark output capatured at the Openflow Switch and Pox Controller's listening ports

Observation

- *The acl_firewall configuration details are not directly visible in the output, however the presence of OFPT_PACKET_IN and OFPT_PACKET_OUT messages imply that the controller is actively filtering and modifying traffic based on the configured access control lists (ACLs).*
- *OFPT_PACKET_OUT messages indicate the instances of controller receiving packets from the network devices, likely due to no matching flow entries (since this has been configured in the acl_firewall.py file)*
- *The output also includes various OpenFlow packet types such as OFPT_ECHO_REQUEST, OFPT_ECHO_REPLY, OFPT_PACKET_IN, and OFPT_PACKET_OUT. These are control messages exchanged between the controller and the network devices.*

3. You are required to submit the ACLfirewall.py written by yourself and the test results with screen recording.

This has been attached with the submitted zip file.