Openflow Firewall

Dimitra Zuccarelli, 20072495

January 2019

1 Introduction

This report describes the steps I followed in attempting to create an Openflow firewall controller in Python using POX. The controller parses a list of rules from a provided csv file and installs these as proactive flows on the switch upon start up. Reactive flows are installed subsequently upon communication between hosts in the topology, but the proactive rules always take precedence.

2 Network Topology

For testing the controller, a very simple custom topology was used with 6 hosts and a single Openflow switch. The code for the topology is below.

```
from mininet.topo import Topo

class CloudTopo(Topo):
    """
    Simple topology with one switch and six
    hosts
    """

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add switch
        s1 = self.addSwitch('s1')

        # Add hosts
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')
        h5 = self.addHost('h5')
        h6 = self.addHost('h6')

        # Add links
        self.addLink(s1, h1)
        self.addLink(s1, h2)
        self.addLink(s1, h3)
        self.addLink(s1, h4)
        self.addLink(s1, h5)
        self.addLink(s1, h6)

topos = { 'cloudtopo': ( lambda: CloudTopo() ) }
```

3 Proactive Firewall Rules

Next, a new controller was created based on the `of_tutorial.py` and the `l2_learning.py` code.

The provided rules are read in from `firewall.csv` and appended to an array of tuples called `block_rules`.

```
with open('firewall.csv', 'rb') as rules:
    reader = csv.DictReader(rules)
    for rule in reader:
        self.block_rules.append(
            (rule['id'], rule['src'], rule['dst'], rule['dstport']))
```

The `block_rules` array contains the following rules:

```
# Type, Src, Dst, Port
[
  ('mac', '00:00:00:00:00:01', '00:00:00:00:00:02', ''),
  ('mac', '00:00:00:00:00:01', '00:00:00:00:00:04', ''),
  ('ip', '10.0.0.4', '10.0.0.6', '80'),
  ('ip', '10.0.0.5', '10.0.0.4', '*')
]
```

A `ConnectionUp` event is triggered when a connection is first established between the switch and the controller. This event can be handled in the `_handle_ConnectionUp` function, and the proactive (firewall) rules that were read in earlier can be installed as flows:

```
for (rule_type, src, dst, port) in self.block:
    if rule_type == "mac":
        match = of.ofp_match(dl_src=EthAddr(src), dl_dst=EthAddr(dst))
    else:
        tcp_port = None if port == "*" else int(port)
        match = of.ofp_match(dl_type=0x0800, nw_proto=6,
                             nw_src=IPAddr(src), nw_dst=IPAddr(dst), tp_dst=tcp_port)

    # Install the proactive rule
    msg = of.ofp_flow_mod(match=match)
    event.connection.send(msg)
    log.debug("%s firewall rule installed on %s",
              rule_type.upper(), dpidToStr(event.dpid))
```

1. If the rule is a MAC rule, the match is done based on the SRC and DST MAC addresses. Ethernet address strings have to be converted to `EthAddr` objects.
2. If the rule is an IP rule, the match is done based on the SRC and DST IP addresses and the TCP port. IP addresses have to be converted to `IPAddr` objects, and TCP ports have to be converted to `Int`s, or `None` for the wildcard.
3. Once the match is created, the flow can be modified and sent to the switch.
4. These rules will always have precedence over any reactive rules installed after them. If the csv file has been modified or updated with additional rules, the controller will have to be restarted altogether.

4 Reactive Learning Rules

The _handle_PacketIn function is triggered when the controller receives an OpenFlow packet-in message from a switch. This portion of the code acts exactly like the layer 2 learning switch (in fact it is taken from the act_like_switch code). The reactive flows are installed here.

```
    def _handle_PacketIn(self, event):
        """
        Handles packet in messages from the switch.
        """

        packet = event.parsed # This is the parsed packet data.
        if not packet.parsed:
            log.warning("Ignoring incomplete packet")
            return

        log.debug("A packet just came in")
        packet_in = event.ofp

        # Learn the port to mac mapping
        self.mac_to_port[packet.src] = packet_in.in_port

        # Check to see if the destination port is known
        if packet.dst in self.mac_to_port:
            dst_port = self.mac_to_port[packet.dst]

            # Install the flow
            log.debug("Modifying flow... \n\t%s (port %i) => %s (port %i)" % (packet.src,
                                                                    packet_in.in_port,
                                                                        packet.dst,
                                                                        dst_port))

            msg = of.ofp_flow_mod() # Start Modifying the flow
            msg.match = of.ofp_match.from_packet(packet)
            msg.idle_timeout = 10
            msg.hard_timeout = 30
            msg.actions.append(of.ofp_action_output(port=dst_port))
            msg.buffer_id = packet_in.buffer_id

            # Send message to switch
            self.connection.send(msg)
        else:
            # Flood the packet out everything but the input port
            msg = of.ofp_packet_out(data=packet_in)
            action = of.ofp_action_output(port=of.OFPP_ALL)
            msg.actions.append(action)

            # Send message to switch
            self.connection.send(msg)
```

A few things to note:

1. These rules will expire if they are not matched after 10 seconds
2. These rules have a hard timeout of 30 seconds, so they will expire after 30 seconds regardless of a match in that time
3. These rules will never have precedence over the firewall rules, and therefore will never overwrite the proactive rules installed in the _handle_ConnectionUp function.

5 Testing

To start the controller and the network, I ran the following commands:

```
# Start the controller
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.of_f2

# In a separate terminal, create the network:
sudo mn --custom ~/mininet/custom/cloud_topo.py --mac --topo cloudtopo --controller remote
```

At this point, the controller displays confirmation that the firewall rules have been installed on h1:

```
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:misc.of_f2:Connection [00-00-00-00-00-01 2]
DEBUG:misc.of_f2:MAC firewall rule installed on 00-00-00-00-00-01
DEBUG:misc.of_f2:MAC firewall rule installed on 00-00-00-00-00-01
DEBUG:misc.of_f2:IP firewall rule installed on 00-00-00-00-00-01
DEBUG:misc.of_f2:IP firewall rule installed on 00-00-00-00-00-01
```

The following scenarios were tested with outcomes as expected:

1. Sent pings between host 1 and host 2 (`h1 ping h2` and viceversa). These pings did not go through because of the MAC firewall rules.

2. Started a python server on host 1 (`h1 python -m SimpleHTTPServer 80 &`), and tried to connect to it from host 2 `h2 wget -O - h1`, this did not work since the hosts are still blocked at the layer 2 level.

3. Started an HTTP server on host 3 and ran wget from other hosts. As expected, this was successful since there are no rules blocking this connection:

   ```
   mininet> h4 wget h3

   --2019-02-02 17:21:10-- http://10.0.0.3/
   Connecting to 10.0.0.3:80... connected.
   HTTP request sent, awaiting response... 200 OK
   Length: 880 [text/html]
   Saving to:  index .html.3

   100%[======================================>] 880    --.-K/s  in 0s

   2019-02-02 17:21:10 (117 MB/s) - index.html.3  saved [880/880]
   ```

4. Ran a few pings between hosts and waited 10 seconds after the reactive flows were added to the flow table. As expected, these disappeared after 10 seconds because they were not being used. Only the original firewall rules are displayed in `s1`'s flow table:

   ```
   sudo ovs-ofctl dump-flows s1
   NXST_FLOW reply (xid=0x4):
    cookie=0x0, duration=73.214s, table=0, n_packets=0, n_bytes=0, idle_age=73,
        dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02 actions=drop
    cookie=0x0, duration=73.176s, table=0, n_packets=0, n_bytes=0, idle_age=73,
        dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04 actions=drop
    cookie=0x0, duration=73.176s, table=0, n_packets=0, n_bytes=0, idle_age=73,
        tcp,nw_src=10.0.0.5,nw_dst=10.0.0.4 actions=drop
    cookie=0x0, duration=73.176s, table=0, n_packets=0, n_bytes=0, idle_age=73,
        tcp,nw_src=10.0.0.4,nw_dst=10.0.0.6,tp_dst=80 actions=drop
   ```

5. Ran a few pings and wgets between hosts and waited 30 seconds. As expected, these flows no longer showed up in the flow table after running a `sudo ovs-ofctl dump-flows s1`.

6. Ran pings between `h1` and `h4`. As expected these hosts were not able to communicate.

7. Installed HTTP servers on `h4`, `h6` and `h5`. As expected, host 4 could not access host 6's server, and host 5 could not access host 4's server. (For reference the firewall rules are shown again below):

```
# Type, Src, Dst, Port
[
  ('mac', '00:00:00:00:00:01', '00:00:00:00:00:02', ''),
  ('mac', '00:00:00:00:00:01', '00:00:00:00:00:04', ''),
  ('ip', '10.0.0.4', '10.0.0.6', '80'),
  ('ip', '10.0.0.5', '10.0.0.4', '*')
]
```