SDN based firewall implementation using POX and Mininet

Varun Subramanya

Akash S Raghavan

Hours Spent :20

***Abstract*:** Software defined networking (SDN) is the new network technology. It emphasises the separation of the network and the control plane. Responsibility is divided between both the planes. The forward plane is only responsible for packet forwarding in the network. The control plane is responsible for policy creation and its implementation, based on predefined rules. It acts as the entry point of the system and can replace the conventional router. Any packet coming into the network is examined by the control plane and a decision is taken on whether to drop the packet or forward it to the next host. It can also update the IP table entry.

Keywords: Firewall, SDN, Control Plane, Data Plane.

I. Introduction

SDN is the physical separation of the control plane from the data plane. So, instead of each networking device independently forwarding packets to the next hop, the controls are centralised on SDN controllers. SDN is an engaging platform for network virtualisation, since each occupants control logic can run on a controller rather than on physical switches. It is an approach to computer networking that allows network administrators to manage network services through the abstraction of higher level functionality.

The characteristics of SDN

- Directly programmable
- Agile
- Centrally managed
- Programmatically configured
- Experimenting and research is not expensive

II. Firewall

A firewall is a system that secures incoming network packets, which come from various sources, as well as outgoing network packets. It can monitor and control the flow of data which comes into the network from different sources, and works on the basis of predefined rules.

Firewalls typically maintain a barricade between a confidential, protected internal network and another outside network, such as the Internet, which is assumed not to be secure or trusted. They can be categorised as either hardware or software firewalls. Network firewalls are software programs running on different hardware appliances in the network [2].

SDN is an engaging platform for network virtualisation, since each occupants control logic can run on a controller rather than on physical switches. It is an approach to computer networking that allows network administrators to manage network services through the abstraction of higher level functionality.

III. Traditional vs SDN based firewalls

Internal traffic is not seen and cannot be filtered by a traditional firewall. An SDN based firewall works both as a packet filter and a policy checker. The first packet goes through the controller and is filtered by the SDN firewall. The subsequent packets of the flow directly match the flow policy defined in the controller. The firewall policy is centrally defined and enforced at the controller [3]

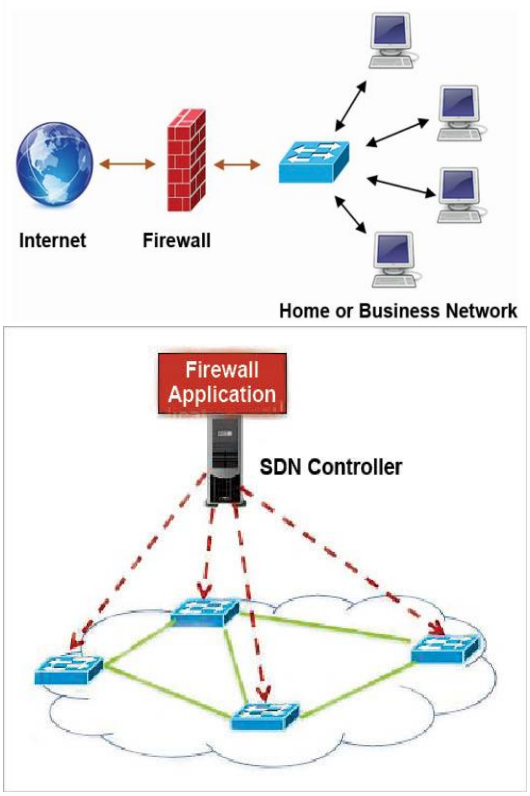

Fig: Traditional and SDN based firewalls

Survey on SDN based firewalls

Security in SDN has become a popular research topic these days. However, this focus arose almost a decade after the inception of Software Defined Networking. The very first work on security of SDN [5] focused only on implementing access control lists within the SDN controller. Moreover, every work on SDN-based firewall focuses on certain aspect of SDN security but misses other important measures Apart from the slow progress in SDN-security, there are other factors responsible for the absence of a comprehensive SDN-based firewall solution. First, there are no operation and design standards for SDN controllers. Since an SDN-based firewall is a software application running inside the controller, unless there exist norms for the design of an SDN controller, an agnostic SDN-based firewall is a distant dream. On the contrary, there exists a well-maintained standard for the OpenFlow protocol [4]. It is intensely adapting to the new requirements.

IV. Approach to firewall implementation

We created 3 files for the project implementation. The CSV file where the rules for the firewall is implemented. A custom topology was created which consists of 4 hosts and 1 switch. The firewall.py file takes in the rules defined in the firewallpolicies.csv file (Mac address are used) and then applies the firewall rules to the same.

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
from pox.lib.util import dpidToStr
from pox.lib.addresses import EthAddr
from collections import namedtuple
import os
from csv import DictReader

log = core.getLogger()
policyFile = "%s/pox/pox/misc/firewallpolicies.csv" % os.environ['HOME']

# Add the global variables here...

# Note: Policy is data structure which contains a single source-destination flow to be blocked on the controller.

Policy = namedtuple('Policy', ('dl_src','dl_dst'))

class Firewall (EventMixin):

        def __init__ (self):
                self.listenTo(core.openflow)
                log.debug("Enabling Firewall Module")

        def read_policies (self,file):
                with open(file, 'r') as f:
                        reader = DictReader(f, delimiter = ",")
```

```
class Firewall (EventMixin):

        def __init__ (self):
                self.listenTo(core.openflow)
                log.debug("Enabling Firewall Module")

        def read_policies (self,file):
                with open(file, 'r') as f:
                        reader = DictReader(f, delimiter = ",")
                        policies = {}
                        for row in reader:
                                policies[row['id']] = Policy(EthAddr(row['mac_0']), EthAddr(row['mac_1']))
                return policies

        def _handle_ConnectionUp (self,event):
                policies = self.read_policies(policyFile)
                for policy in policies.itervalues():
                #TODO : implement the code to add a rule to block the flow
                #between the source and destination specified in each policy


                        log.debug("--> Source Mac is %s", policy.dl_src)
                        log.debug("--> Destination Mac is %s", policy.dl_dst)

                        match = of.ofp_match(dl_src = policy.dl_src, dl_dst = policy.dl_dst)
                        fm = of.ofp_flow_mod()
                        fm.priority = 20
                        fm.match = match
                        event.connection.send(fm)
```

Fig : Firewall.py implementation code

```
from mininet.topo import Topo

class MyTopo( Topo ):
        " simple topology "

        def __init__( self ):
#Initialize topology

                Topo.__init__( self )

#Add hosts and switches

                FirstHost = self.addHost( 'h1' )
                SecondHost = self.addHost( 'h2' )
                ThirdHost = self.addHost( 'h3' )
                FourthHost = self.addHost( 'h4' )

                firstSwitch = self.addSwitch( 's1' )

#Add Links
                self.addLink( firstSwitch,FirstHost)
                self.addLink( firstSwitch,SecondHost)
                self.addLink( firstSwitch,SecondHost)
                self.addLink( firstSwitch,ThirdHost)
                self.addLink( firstSwitch,FourthHost)

topos = { 'mytopo': (lambda: MyTopo() ) }
```

Fig: Custom topology consisting of 4 hosts and 1 switch.

```
id,mac_0,mac_1
1,00:00:00:00:00:02,00:00:00:00:00:03
```

Fig : The rule in the CSV file.

V. Results and Screenshots

The pox controller is setup.The command for the pox controller setup is shown in the screenshot.

```
ubuntu@sdnhubvm:~[19:02]$ ./pox.py log.level --DEBUG forwarding.tutorial_l2_hub
pox.misc.firewall
bash: ./pox.py: No such file or directory
ubuntu@sdnhubvm:~[20:09]$ cd pox
ubuntu@sdnhubvm:~/pox[20:09] (eel)$ ./pox.py log.level --DEBUG forwarding.tutori
al_l2_hub pox.misc.firewall
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
DEBUG:misc.firewall:Enabling Firewall Module
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.6/Nov 23 2017 15:49:48)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.5.0 (eel) is up.
```

Fig: Pox controller is setup

The mininet is setup. The command for the mininet setup is shown in the screenshot.

```
mytopo --mac --controller=remote,ip=127.0.0.1,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
```

Fig: Mininet setup

The Pox is running and the rule is applied on mac_0. Connection is aborted once the rule is applied on the particular host.

```
ubuntu@sdnhubvm:~[19:02]$ ./pox.py log.level --DEBUG forwarding.tutorial_l2_hub
pox.misc.firewall
bash: ./pox.py: No such file or directory
ubuntu@sdnhubvm:~[20:09]$ cd pox
ubuntu@sdnhubvm:~/pox[20:09] (eel)$ ./pox.py log.level --DEBUG forwarding.tutori
al_l2_hub pox.misc.firewall
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
DEBUG:misc.firewall:Enabling Firewall Module
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.6/Nov 23 2017 15:49:48)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:misc.firewall:--> Source Mac is 00:00:00:00:00:02
DEBUG:misc.firewall:--> Destination Mac is 00:00:00:00:00:03
DEBUG:misc.firewall:Firewall rules installed on 00-00-00-00-00-01
DEBUG:openflow.of_01:1 connection aborted
^CINFO:core:Going down...
INFO:openflow.of_01:[00-00-00-00-00-01 2] disconnected
INFO:core:Down.
ubuntu@sdnhubvm:~/pox[20:29] (eel)$
```

Fig: Screenshot showing the firewall rule successfully applied and the connection abortion.

The Pingall command is used to ping all the hosts from every other host. Since the rules are applied on h2 and h3, the ‘X’ mark is shown when h2 pings h3 and vice versa.

```
mytopo --mac --controller=remote,ip=127.0.0.1,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 X h4
h3 -> h1 X h4
h4 -> h1 h2 h3
*** Results: 16% dropped (10/12 received)
mininet>
```

Fig: The firewall implementation successful.

VI. Conclusion

We have implemented a simple firewall with four hosts and one switch successfully. During the project we learnt a lot about firewalls, mininet and implementation of custom topologies.

VII. Reference

[1] Recommendation ITU-T Y.3300, “Framework of Software-Defined Networking,” ITU-T, Jun. 2014.

[2] Open Networking Foundation, “SDN Architecture,” ONF, Jun. 2014.

[3] H. Kim and N. Feamster, “Improving Network Management with Software Defined Networking,” IEEE Communications Magazine, vol. 51, no. 2, pp. 114–119, Feb. 2013.

[4] OpenFlow Switch Speci□cation 1.4.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-speci□cations/open□ow/open□ow-spec-v1.4.0.pdf

[5] M. Casado, M. J. Freedman, J. Pettit, J. Luo, and N. McKeown. 2007. Ethane: Taking Control of the Enterprise. In Proceedings of the ACM SIGCOMM, August, 2007, Kyoto, Japan. (2007)