Received: March 1, 2017

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

FlowAgent: Software Defined Firewall Rule Generator for Network Intrusion Detection System

Nithya Muthukumaran[1]* **Jayakumar Chinnappan**[2]

[1] *Vellore Institute of Technology University, Vellore, India*
[2] *Sri Venkateswara College of Engineering, Chennai, India*
* Corresponding author's Email: nithya.s@vit.ac.in

Abstract: Networking suffers from various security vulnerabilities in the present paradigm due to the technical difficulties like static architectural design, provider-dependent and it is economically costly. To overcome these challenges a novel paradigm concept is introduced based on dynamic control in the network as Software Defined Firewall. Software Defined Network is an emerging reprogrammable technology which helps the administrator to control the overall network, on the other hand Firewall act as a single point perimeter to protect the network from network attacks. Combination of different paradigms led to reshaping of future. Software Defined Firewall is a new network platform to manage the networks by segregating the control plane from the data plane. This separation provides a programmatic control over the network traffic by writing rules, which act as a network attack defence. Hereby the firewall policy rules are written by analysing the traffic and packet log are mined using Association rule mining techniques. Analysis of these policy rules costs for generation of efficient rule set and multiple minimum support with probability will minimize the number of rule set which will result in effective network policy management. Software Defined Firewall provides a better solution to reduce ratio of attack traffic.

Keywords: Software defined network, Firewall, Firewall rules, Controller, Mininet, POX.

1. Introduction

In today's network security for the global world of internet technologies de facto core is firewall which also first in line for defence against external threats and network attacks. Firewall is an integrated collection of security measures designed to prevent from unauthorized access .It may be a software or hardware designed to block unauthorized access. Network firewalls are used to protect large private networks and individual machines from the danger of the internet, a firewall can be employed to filter incoming or outgoing traffic based on a predefined set of rules called firewall policies.

Firewall is essential for governing network access and by denying or allowing network traffic flow according to its policy rules. These rules are explicitly managed and written for the purpose of filtering any unwanted information from the secured network. As the number of filtering rules drastically increases beyond reasonable scope and scale of manual process , furthermore making the task of manually managing the firewall policy rules very time consuming and difficult. Thus the effective management of firewall security along with policy management techniques can be implemented using Associated Rule Mining (ARM) technique.

Software Defined Network (SDN) is a programmable network technology which helps the administrator to deal with network traffic, flow of packets and so on. The vendor can manage and administrate their network according to their network usage [1]. It can be reconfigured from time to time by the Network Administrator to suite the network traffic and various applications. According to the rule set instruction in control plane, the switch which will act as a forwarding media.

Software Defined Firewall (SDF) is an effective rule generator using a data mining technique named associated rule mining for multiple minimum support.

These rule sets are defined with respect to the flow entries in the flow table. Firewall policies are ordered filtering rules that define the actions performed on matching packets. The frequent rules are mined and the firewall policies are generated to detect the anomalies.

This paper is organized into the following sections. In Section 2, summary of related work is presented. In Section 3, the process of analysing and managing firewall policy rules including data mining for policy management is explained. Section 4 consists of the result of the experimental design and implementation using mininet with POX controller is described. In Section 5, conclusion of this research and potential areas for future work is presented.

2. Related work

2.1 SDN architectural component

SDN is a revolutionary paradigm which provides the transition from system standards to component standards. SDN is highly dynamic and presents configurable topologies which support easy scalability, centralized control, target flexibility and reduced cost [1]. Considering the layer independent, it slices the layers into three layered architecture namely

- Control Layer
- Infrastructure Layer
- Application Layer

Figure 1 describes about the layers and various interfaces in SDN.

2.2.1. Control plane

The control plane of SDN comprises of Controllers, these controllers determine the flow path of packets around the network. It acts as a centralized entity in charge of (i) controlling the path flow of the packets from SDN Data plane, according to the requirements to the SDN application layer (ii) providing the SDN applications with an abstract view of each and every event of the network (iii) maintains the flow entries for security purpose. An administrator can shape the traffic and make decision about their traffic from a centralized control console without modifying their individual switches.

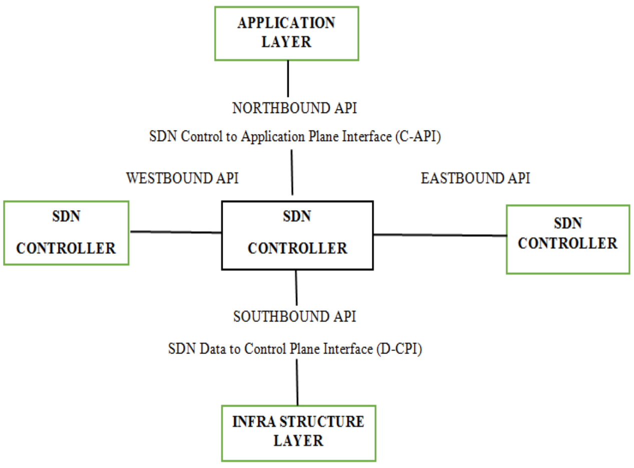


Figure.1 Layered architecture & interfaces of SDN

2.1.2. Data plane

SDN Data plane as an infrastructure layer comprises of the Forwarding Elements (FEs) which are physical and virtual switches. It functions based on control instruction from the controller plane via southbound interfaces. A data plane may include the necessary minimum subset of control and management functions. One or more SDN Data plane can be configured in a physical network element—an integrated physical combination of communications resources, managed as a unit. A unit can be controlled by a SDN controller by programming corresponding instruction with respect to network state.

2.1.3. Application plane

Application plane comprises of one or more end user applications by closely binding the interaction among network services and devices. Applications are programs that explicitly, directly, and programmatically communicate their network device to the SDN Controller via interface. The SDN application plane interacts with the controllers to achieve a desired network function in order to fulfil the network operator needs. The application can be categorize according to the network functionality like security, quality of service (QoS), traffic engineering (TE), access control lists (ACL) management, and load balancing [2].

2.1.4. Interface drivers & agents

Interfaces play a major role for effective communication between the planes. Each interface is implemented by a driver-agent, the agent representing the southern and northern bound.

2.2 Firewall

Firewall is a perimeter to protect the network from internet based attacks and to provide a single point where security and auditing can be imposed. Packets flowing through the firewall can either be accepted, or rejected, or dropped. The firewall has two network interfaces. One is used for the external side of the network and the other one is used for the internal side of the network. The main purpose is to control the traffic which is traverse from one side to other. Sometimes, firewall can block the traffic which is intended for the particular IP addresses or server ports.

2.2.1. Packet-filtering firewall

Packet-Filtering Firewall is also called as stateless firewalls. It operates at the router and compares each packet with the set of criteria such as port number, packet type, and IP address before the packet being dropped or forwarded [3].

2.2.2. Circuit Level Gateway

Circuit Level Gateway monitors the TCP handshaking between the local and remote host to determine whether the remote system is considered as trusted.

2.2.3. Stateful inspection firewall

Stateful Inspection Firewall maintains state information record of all packets passing through the network. It also able to indicate whether the packet is from trusted network. These firewalls are more secure than the packet filtering and the circuit level gateways [5].

2.2.4. Application level gateway

It works as proxy while combining the attribute of packet filtering firewalls with those of circuit level gateways. It may check the contents of the traffic and block those which are view as inappropriate content.

2.2.5. Multilayer inspection firewall

Multilayer Inspection Firewall enables the direct connection between local and remote hosts that are transparent to the network. This is done by relying on the algorithms to recognize which service is requested. It retains the status assigned to a packet by each firewall. This allows user to maximum control over the packets which are allowed to reach the destination. But it affects the network performance [3].

3. Software Defined Firewall (SDF)

SDN is the ideal solution which implements centralized management, plug and play, configuring multiple devices, channels the resources accordingly. SDN directly controls the network’s data-plane elements via a well-defined Application Programming Interface (API). Then firewall provides an additional layer of defence, insulating the internal systems from external networks. The network administrator can change any switch rules whenever it is necessary, the rule may be prioritizing, de-prioritizing or even blocking specific types of packets, access control lists management. The major control plane functions include the system configuration, management, controlling the switch and exchange/updating of table information [4, 5]. The heterogeneous flow table help to form association rules.

3.1 Controller design

Controller is a core part of SDN it acts as a single, logically centralized network operating system in terms of both scheduling and resolving resource which learns which paths the packets moves. It enables multi-layer programmatic access to make network administration much more flexible and the controller provides a single interface to configure all the network elements on the network. In control plane, the instruction logic, that decides where to forward the packets, which is separated from the network elements and hence the forwarding decisions are made only at SDN Controller [5]. The remote controllers are used to manage the traffic, provide the firewall policies and secure the network by forwarding the packets for further decision with help of flow table.

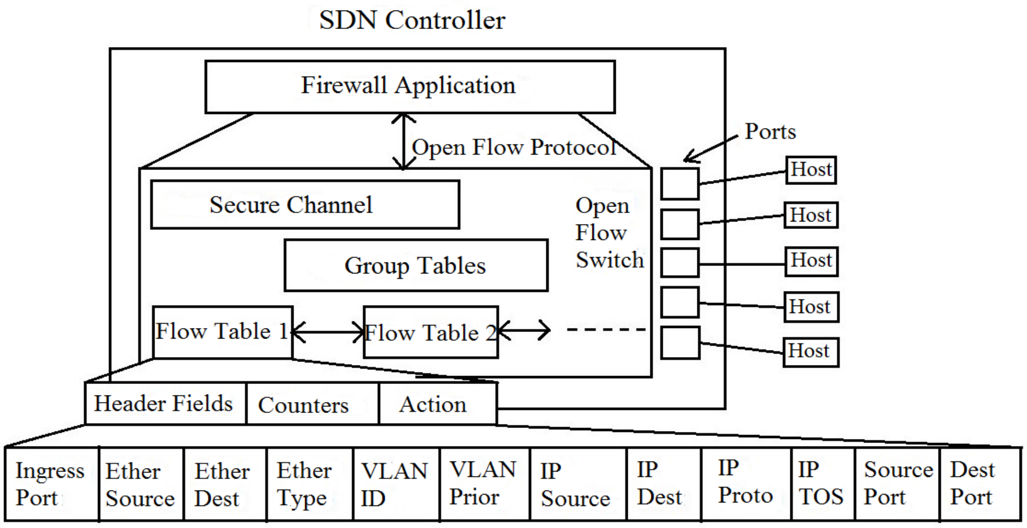


Figure.2 Software defined firewall

3.2 Openflow switch

An OpenFlow switch has one or more tables of packet-handling rules. Each rule matches a subset of traffic and performs certain actions on the traffic that matches a rule; actions include dropping, forwarding, or flooding. Depending on the rules installed by a controller application, an OpenFlow switch can behave like a router, switch, firewall and intrusion detection system [6]. An Openflow switch consists of three main elements:

3.2.1. Flow table

Flow Table consists of an action field along with the flow according to the rules specified. Flow entries are entered in flow table and this defines how the switch is handling the flow. Flow table are filled by the controller. The group entries are found in group table which allows expressing additional methods for flow forwarding [7].Table 1 describes about the tuple and its corresponding action values in the flow table.

3.2.2. Secure channel

A secure channel is a control path between the switch and the controller for commanding, forwarding and for programming purposes. Communicated messages may include information on received packets, sent packets, statistics collection and actions to be performed on specific flows.

3.2.3. Openflow protocol

OpenFlow protocol is a programmable interface between Control Plane and Data Plane, which allows switches to perform flows-level control. An OpenFlow switch consists of flow tables and group table which executes the task of packet look ups and forwarding.

Table.1 Flow Table tuple description

Header	Format	Description
Ingress Port	Short	Port number of input switch
Ethernet Source Address	xx:xx:xx:xx:xx:xx	Source MAC address
Ethernet Destination Address	xx:xx:xx:xx:xx:xx	Destination MAC address
Ethernet Type	ARP/IPv4	Protocol
VLAN ID	xx:xx:xx:xx:xx:xx:xx:xx	Switch identification number
VLAN Priority	Int	Priority of the rule
IP Source Address	A.B.C.D/M	Source IP address
IP Destination Address	A.B.C.D/M	Destination IP address
IP Protocol	TCP or UDP or ICMP	Protocol
IP ToS Bits	Short	Type of Service
Source Port	Short	Source Port Number
Destination Port	Short	Destination Port Number
Action	Allow or deny	Allow or deny the network flows that match the rules
Counter	Statistics	Maintained per entry

Table 2. Firewall Policy Rule Sets

Src IP	Dest IP	Service	Interface	Direction	Action	Comment
10.0.0.2	Any	SSH or Any	Outside	Outbound or any	Deny	Anti-spoofing rule
Firewall	10.0.0.1	DNS	Loopback	Inbound	Accept	Firewall uses one of the machine

3.2.4. Firewall Log

Firewall provides huge amount of log audit data in entire network environment. Firewall logs can be collected and analysed to determine what types of traffic to be allowed or denied. Therefore, the analysis of firewall lo enables the network administrators to get insight into their network security state.

4. Mining Firewall Log File

Each firewall object has several sets of rules associated with policy rules. Firewall policy rules filter traffic, controlling access to and from the firewall machine and the machines behind it. NAT rules describe address and port transformations that the firewall should make to packets flowing through it. For example, some firewalls perform address and port transformations first and then apply policy rules, while some others do it the other way around.

As the number of filtering rules drastically increases, managing the firewall policy rules become very time consuming and difficult. Thus the effective management of firewall security along with policy management techniques and tools is needed for the enormous task thus enabling network administrators with ease to automatically optimize and validate firewall rules [8].

4.1 Set up a firewall policy rule

Association rule mining is the process of discovering correlations among the entities in a dataset. Two statistical measures which govern association rule mining are *frequency* and *threshold.* Frequent Pattern Growth algorithm is an efficient method of mining all frequent item sets. Since the log files are growing drastically the rules should be mined with multiple support instead of single support. To bridge the gap between the observation in the network and the rules written in the firewall policy, analysing the traffic and log the packets these effective mining techniques are used. With the help of these techniques the traditional research is extending in the management and generation of efficient policies, minimize the number of rules, reflection of up to date traffic trend and providing a real time policy update capability from the application.

4.2 Algorithm

```
Start
Calculate Threshold (TD), Log File (L),MaxItem Pattern Difference (MPD)
Generate Policy Rule PR1;
Calculate Frequency
    Policy Rule (PR1)= Frequency(F1) / L;
Calculate Min Frequency for each Log File in P1;
        MF= (PR1, Frequency);
        Max_value = max (Policy Rule) – MPD
        If Max_value > min_mis then
                min_1=Max_value;
          Else min_PR=min_mis;
Generate frequent 1-itemset
     L1= {<i> | i ε c1, Policy Rule(i) ≥ min_mis(i)};
        L1= sort(L1,MF);
Calculate MF and IPD;
        for k=2; Lk-1≠φ;k++ do
        // repeats till no frequent item occurs
        PR1=candidate-gen(Lk-1);
        PL=subset(PR1,L);
        for each policy p ε Ct do
        temp_prob(j)= Policy Rule(PR)
          if min_mis >MF(PR)
        then
             min_min=MF(PR)
                j++;
        end for
      Policy Rule (PR1) = a * min(temp_prob) + ( b
* min(temp_prob) * max(temp_prob));
        Max_value=max(temp_prob);
        if max_value >= min_mis
        then
          min_sup=max_value
        else  min_sup= min_mis
        IPD=max_value – prob_sup(ck);
        Lk ={ c ε C | IPD<=MPD & Policy Rule(PR1)
        ≥ min_mis &  Policy Rule(PR1)>=min_sup| for
        all i ε c) }
                end for
        return U Lk;
        k,
Generate Association Rules,
End.
```

Figure.3 Pseudo code for MMSP algorithm

Received: March 1, 2017

4.3 Mining firewall log using frequency

A large number of rules which may turn out to be impractical or useless analysis might be a result of an Apriori algorithm [9]. Rules which do not yield in any action or repeated rules have no impact on firewall. Thus these rules are filtered for further analysis. Generalizing specific and unique rules to more general rules is the major objective. Thus identification of frequent patterns that occur in original form throughout the database typically is done by Frequent Pattern mining algorithm. Only the database entries of the exact same match for the candidate patterns may contribute to the support of the candidate pattern is one of the limitations of many Association Rule Mining algorithms like Apriori algorithm.

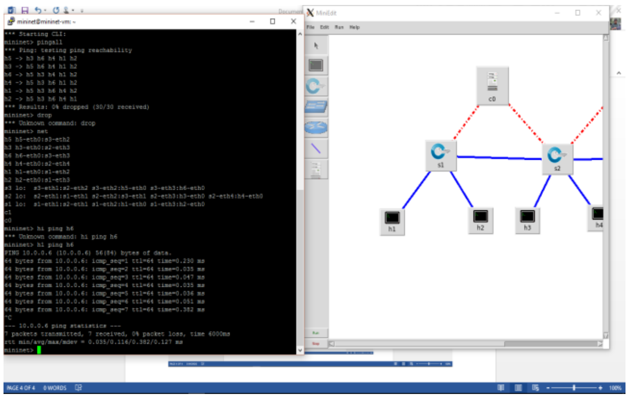
A set of techniques and algorithm to analyse and manage firewall policy rules is presented in this paper such as:

- Instead of using single minimum support, multiple minimum supports with probability (MMSP) [10].
- Frequent pattern growth algorithms are used in firewall log Fig.3.
- Deduction of efficient firewall policy rules by mining the network traffic log based on the frequency.
- Filtering Rule Generalization (FRG) for the reduction of the number of policy rules by generalization.

In this paper the technique provides an automated tool for the discovery of frequent traffic patterns and filtering rules, using Filtering Rule Generalization and Log Mining Frequency, thus an effective and anomaly free firewall policy rules can be generated and discovered. Filtering may be an important measure to eliminate false positives or undesired alarms in general. A typical example is the filtering based on network addresses if a given host is known to cause many false alarms, even though the host itself is known to be harmless.

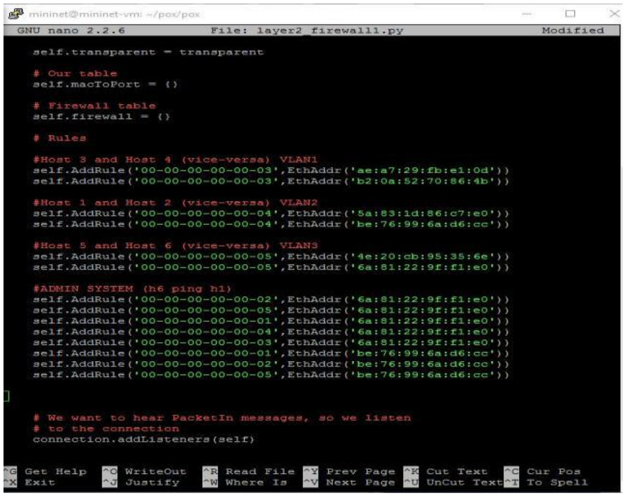
For detecting a massive incoming of spam mails from a host to the mail server. [11]An example rule for firewall policy rule

In_port=2,eth_src=06:e3:f7:80:ed:84, dst=8e:d9:68:d5:6c:a4),eth_type(0x0800),ipv4 src=10.0.0.2, dst=10.0.0.1, proto=1, tos=0, ttl=64, frag=no, icmp(type=0, code=0), packets:19, bytes:1862, used:0.648s, actions: Drop



```
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h5 -> h3 h6 h4 h1 h2
h3 -> h5 h6 h4 h1 h2
h6 -> h5 h3 h4 h1 h2
h4 -> h5 h3 h6 h1 h2
h1 -> h5 h3 h6 h4 h2
h2 -> h5 h3 h6 h4 h1
*** Results: 0% dropped (30/30 received)
mininet> drop
*** Unknown command: drop
mininet> net
h5 h5-eth0:s3-eth2
h3 h3-eth0:s2-eth3
h6 h6-eth0:s3-eth3
h4 h4-eth0:s2-eth4
h1 h1-eth0:s1-eth2
h2 h2-eth0:s1-eth3
s3 lo:  s3-eth1:s2-eth2 s3-eth2:h5-eth0 s3-eth3:h6-eth0
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1 s2-eth3:h3-eth0 s2-eth4:h4-eth0
s1 lo:  s1-eth1:s2-eth1 s1-eth2:h1-eth0 s1-eth3:h2-eth0
c1
c0
mininet> hi ping h6
*** Unknown command: hi ping h6
mininet> h1 ping h6
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_seq=1 ttl=64 time=0.230 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=64 time=0.035 ms
64 bytes from 10.0.0.6: icmp_seq=3 ttl=64 time=0.047 ms
64 bytes from 10.0.0.6: icmp_seq=4 ttl=64 time=0.035 ms
64 bytes from 10.0.0.6: icmp_seq=5 ttl=64 time=0.036 ms
64 bytes from 10.0.0.6: icmp_seq=6 ttl=64 time=0.051 ms
64 bytes from 10.0.0.6: icmp_seq=7 ttl=64 time=0.382 ms
^C
--- 10.0.0.6 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6000ms
rtt min/avg/max/mdev = 0.035/0.116/0.382/0.127 ms
mininet>
```

Figure.4 Topology creation in miniedit



```
  GNU nano 2.2.6              File: layer2_firewall1.py                    Modified

    self.transparent = transparent

    # Our table
    self.macToPort = {}

    # Firewall table
    self.firewall = {}

    # Rules

    #Host 3 and Host 4 (vice-versa) VLAN1
    self.AddRule('00-00-00-00-00-03',EthAddr('ae:a7:29:fb:e1:0d'))
    self.AddRule('00-00-00-00-00-03',EthAddr('b2:0a:52:70:86:4b'))

    #Host 1 and Host 2 (vice-versa) VLAN2
    self.AddRule('00-00-00-00-00-04',EthAddr('5a:83:1d:86:c7:e0'))
    self.AddRule('00-00-00-00-00-04',EthAddr('be:76:99:6a:d6:cc'))

    #Host 5 and Host 6 (vice-versa) VLAN3
    self.AddRule('00-00-00-00-00-05',EthAddr('4e:20:cb:95:35:6e'))
    self.AddRule('00-00-00-00-00-05',EthAddr('6a:81:22:9f:f1:e0'))

    #ADMIN SYSTEM (h6 ping h1)
    self.AddRule('00-00-00-00-00-02',EthAddr('6a:81:22:9f:f1:e0'))
    self.AddRule('00-00-00-00-00-05',EthAddr('6a:81:22:9f:f1:e0'))
    self.AddRule('00-00-00-00-00-01',EthAddr('6a:81:22:9f:f1:e0'))
    self.AddRule('00-00-00-00-00-04',EthAddr('6a:81:22:9f:f1:e0'))
    self.AddRule('00-00-00-00-00-03',EthAddr('6a:81:22:9f:f1:e0'))
    self.AddRule('00-00-00-00-00-01',EthAddr('be:76:99:6a:d6:cc'))
    self.AddRule('00-00-00-00-00-02',EthAddr('be:76:99:6a:d6:cc'))
    self.AddRule('00-00-00-00-00-05',EthAddr('be:76:99:6a:d6:cc'))



    # We want to hear PacketIn messages, so we listen
    # to the connection
    connection.addListeners(self)

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text^T To Spell
```

Figure.5 Firewall policy rules

5. Experimental Result and Discussion

Mininet is a Software Network Emulator that allows virtual network with networking devices like hosts, switches and an SDN controller all with a single command. It makes a Single system looks as networks of the computer, running the same kernel, system and user code through lightweight virtualization [11].

In Fig.4, a Customized Mininet Topology with 3 Open-flow Switches, 6 hosts and 2 external controller (POX) with an IP address of 127.0.0.1 using port 6633 was created.

Then, start the POX Controller launching the Port Blocker Application and without it, Configure the Host 1 (10.0.0.1) to serve as an HTTP server and the Host 6 (10.0.0.6) as the client machine which will request HTTP service from the server using the CURL command. Firewall policy rule can be added.

In Fig. 6, a Customized Mininet Topology with 1 Open-flow Switches, 3 hosts and an external controller (POX) with an IP address of 127.0.0.1 using port 6633 was created.

ubuntu@sdnhubvm:Sudo mn –-topo single,3 –mac –switch ovsk –controller=remote

mininet>xterm h1 h2 h3

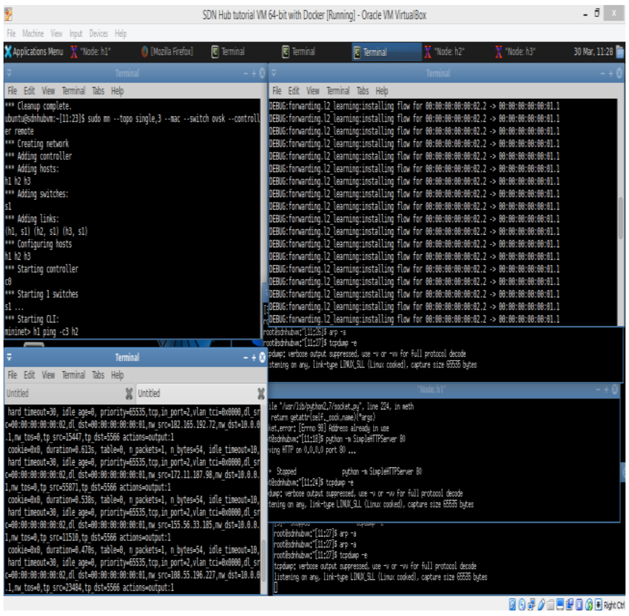
By using xterm the hosts h1,h2,h3 and POX terminal were opened.

ubuntu@sdnhubvm: sudo ovs-oftcl dump-flows s1

ubuntu@sdnhubvm:tcpdump -e

For mining the associated rules with multiple minimum support the POX controller is trained to learn the forwarding packets and log files. The flow in the switch is displayed using dump flow command.

Customizing the control with flow rule ends up with a new firewall policy. By using source address and switch port the updating can be made for flow entries. The firewall policies were stored in .csv file and it can be given as a input to the firewall rule mining .Check the source MAC address against firewall rules and install flow table entry in the switch so that this flow goes out the appropriate port will be forwarded or dropped.



```
*** Cleanup complete.
ubuntu@sdnhubvm:~[11:23]$ sudo mn --topo single,3 --mac --switch ovsk --controll
er remote
*** Creating network
*** Adding controller
*** 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> h1 ping -c3 h2
```

```
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.2 -> 00:00:00:00:00:01.1
```

```
root@sdnhubvm:~[11:26]$ arp -a
root@sdnhubvm:~[11:27]$ tcpdump -e
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 65535 bytes
```

```
 hard_timeout=30, idle_age=0, priority=65535,tcp,in_port=2,vlan_tci=0x0000,dl_sr
c=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw_src=182.165.192.72,nw_dst=10.0.0
.1,nw_tos=0,tp_src=15447,tp_dst=5566 actions=output:1
 cookie=0x0, duration=0.613s, table=0, n_packets=1, n_bytes=54, idle_timeout=10,
 hard_timeout=30, idle_age=0, priority=65535,tcp,in_port=2,vlan_tci=0x0000,dl_sr
c=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw_src=172.11.187.98,nw_dst=10.0.0.
1,nw_tos=0,tp_src=55871,tp_dst=5566 actions=output:1
 cookie=0x0, duration=0.538s, table=0, n_packets=1, n_bytes=54, idle_timeout=10,
 hard_timeout=30, idle_age=0, priority=65535,tcp,in_port=2,vlan_tci=0x0000,dl_sr
c=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw_src=155.56.33.185,nw_dst=10.0.0.
1,nw_tos=0,tp_src=11510,tp_dst=5566 actions=output:1
 cookie=0x0, duration=0.470s, table=0, n_packets=1, n_bytes=54, idle_timeout=10,
 hard_timeout=30, idle_age=0, priority=65535,tcp,in_port=2,vlan_tci=0x0000,dl_sr
c=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01,nw_src=108.55.196.227,nw_dst=10.0.0
.1,nw_tos=0,tp_src=23484,tp_dst=5566 actions=output:1
```

```
ile "/usr/lib/python2.7/socket.py", line 224, in meth
 return getattr(self._sock,name)(*args)
ket.error: [Errno 98] Address already in use
t@sdnhubvm:~[11:18]$ python -m SimpleHTTPServer 80
ving HTTP on 0.0.0.0 port 80 ...

+  Stopped                python -m SimpleHTTPServer 80
t@sdnhubvm:~[11:24]$ tcpdump -e
dump: verbose output suppressed, use -v or -vv for full protocol decode
tening on any, link-type LINUX_SLL (Linux cooked), capture size 65535 bytes
```

```
root@sdnhubvm:~[11:27]$ arp -a
root@sdnhubvm:~[11:27]$ arp -a
root@sdnhubvm:~[11:27]$ tcpdump -e
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 65535 bytes
[]
```

Figure.6 Firewall policy mining

6. Conclusion

In this paper, the association firewall rule generator for SDN environment towards the detection of anomalies has been discussed. The implementation of the firewall rules in Mininet with POX Controller proves the ability to generate policies by mining the logs. This work can be extended to derive the firewall policy with respect to attack detection using ensemble algorithm and mine the associated firewall rules whereas switch itself will act as an Intrusion Detection System [12].

References

[1] Y. Jarraya, T. Madi and M. Debbabi, “A Survey and a Layered Taxonomy of Software-Defined Networking”, *IEEE Communication Surveys & Tutorials*, Vol. 16, No. 4, pp.1955-1980, 2014.

[2] S. Sezer, S. Scott-Hayward, P.K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, “Are we ready for SDN? Implementation challenges for software-defined networks”, *IEEE Communications Magazine*, Vol. 51, No. 7, pp 36-43, 2013.

[3] A.D. Wankhade and P.N. Chatur, Comparison of firewall and intrusion detection system. *International Journal of Computer Science and Information Technologies*, Vol.5, No. 1, pp. 674-678, 2014.

[4] S. Nithya, M. AshaJerlin and C. Jayakumar, “A Survey of Software Defined Network”, *International Journal of Pharmacy and Technology*, Vol.8, No.4, pp 25944-25958, 2016.

[5] T. Vinh Tran and H. Ahn, "Flowtracker: A SDN Stateful Firewall Solution with Adaptive Connection Tracking and Minimized Controller Processing”, In: *Proc. of the 2016 International Conference on Software Networking*, pp. 1-5, 2016.

[6] S. Morzhov, I. Alekseev, and M. Nikitinskiy, “Firewall application for Floodlight SDN controller”, In: *Proc. of the International Siberian Conference on Control and Communications*, pp. 1-5,2016.

[7] J. F. Huang, G. Y. Chang, C. F. Wang, and C. H. Lin, “Heterogeneous Flow Table Distribution in Software-Defined Networks”, *IEEE Trans. on Emerging Topics in Computing*, Vol. 4, No. 2, pp. 252-261, 2016.

[8] D. Satasiya and D. Raviya Rupal, “Analysis of Software Defined Network firewall (SDF)”, In: *Proc. of the 2016 International Conference on Wireless Communications, Signal Processing and Networking*, pp. 228-231, 2016.

[9] A.L. Buczak and E. Guven, “A survey of data mining and machine learning methods for cybersecurity intrusion detection”, *IEEE Communications Surveys & Tutorials*, Vol 18, No.2, pp 1153-1176, 2015.

[10] S. Nithya and C. Jayakumar, "Automatic Firewall Rule Generator for Network Intrusion Detection System based on Multiple Minimum Support", *Indian Journal of Science and Technology*, Vol 9, No.41, pp 1-4, 2016.

[11] F. Keti and S. Askar, "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments", In: *Proc. of the 6th International Conference on Intelligent Systems*, Modelling and Simulation, pp. 205-210, 2015.

[12] S. T. Ali, V. Sivaraman, A. Radford, and S. Jha, "A Survey of Securing Networks Using Software Defined Networking", *IEEE Trans. on Reliability*, Vol. 64, No. 3, pp. 1086-1097, 2015.