Application Oriented Stateful Firewall Mechanism for Software Defined Network

ABHISHEK RAJ CHAUHAN-20BCI0161,
*(Computer Science Engineering with Specialization in Information Security, Vellore Institute of Technology, Vellore
Email:abhishekraj.chauhan2020@vitstudent.ac.in)
KANUGO KRISHNA GANESH-20BCI0149,
*(Computer Science Engineering with Specialization in Information Security, Vellore Institute of Technology, Vellore
Email: kanugokrishna.ganesh2020@vitstudent.ac.in)

BATCH NUMBER: 15

Report submitted for the Final Project Review of

CSE2008 – Network Security

Professor: Dr. Karthik G M

B.Tech. in

Computer Science and Engineering with Specialization in Information Security

School of Computer Science & Engineering

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Abstract:

Software Defined Network is a networking three layered architecture which has application, data, and control planes. The routing and forwarding functions are implemented on the control plane. Application Layer which consists of networking applications like monitoring, traffic control, network analysis, and security. A firewall is a very critical application for any network. It acts as the first/last line of defence against any unauthorized user trying to access or exploit the network. SDN provides more flexible, programmable, and innovative network's architecture, which may pose new vulnerabilities and may lead to new security problems. Hence, we're proposing application aware firewall mechanism for SDC.The Stateful firewall is an application that is implemented on the control plane. The stateful firewall keeps track of the TCP streams so this type of firewall is implemented in the control plane[1]. The firewall will filter the TCP packets or communications according to the network policies. To provide more control and visibility in applications running over the network, the system detects network applications that may at some point affect network's performance and dynamically enforces constraint rules on those applications. The proposed mechanism checks the network traffic at the network, transport, and application levels, and installs appropriate security instructions down into the network[4]. Proposed solution features are implemented using POX SDN controller, which is an open, software-defined networking (SDN) Controller designed to increase the agility of the network by making it easy to manage and adapt how traffic is handled. OpenFlow which is a protocol that allows a server to tell network switches where to send packets and Mininet emulation tool. which is an emulator of a network, and it is used to visualize the switches and application of software-defined networks in a virtualized environment.

Keywords **— SDN, STATEFUL FIREWALL, OPENFLOW, POX CONTROLLER, MININET, CONTROL PLANE.**

I. PROJECT OBJECTIVE

- Implement the stateful firewall in the SDN application.
- To build a small network on the virtual machine using the "Mininet". Which consists of POX controller, a switch with 3 different hosts.
- Discarding the invalid TCP sessions.

II. INTRODUCTION

In today's world, there are thousands of applications that use the Internet to provide different services, such as online shopping, data transfer, instant messaging, or video conferencing. They use a wide range of different communication protocols. The key feature of the SDN model is the physical separation between the control plane and the data plane. The control plane of all network devices is migrated from the forwarding elements and grouped into a software-based device known as a "controller". Typically, SDN architecture encompasses three distinct planes: the data plane, the control plane, and the application plane. The data plane involves network devices connected to each other. This plane is responsible for data forwarding as well as gathering local information and statistics. The control plane contains one or more connected controllers[13]. It strives to behave like a network operating system. It represents the brain of the network in which the network intelligence is centralized. The controller's comprehensive view of the entire network helps to make forwarding decisions for the underlying network elements. A firewall is a crucial application in any network. It serves as the first or final line of protection against anyone trying to access or use the network in an unlawful way. By adding or modifying flow entries to cause misconfigurations, launching DoS assaults, or even sniffing surreptitiously vital network information, these individuals can destroy networks in several ways. The goal of this project is to use python code to construct a straightforward Stateful firewall on the POX controller and learn how rules work to carry out specific actions on packets that match them[5].

Understanding this lab's concepts should serve as a basis for knowledge of upper layer firewalls and how software may govern the network. This fundamental firewall can eventually be improved with new features. POX is a component-based software defined networking framework. POX provides software components with well-defined API that make it easy for developers to create new network management and control applications. With a single command, Mininet quickly builds a realistic virtual network running genuine kernel, switch, and application code on a single computer (VM, cloud, or native)[9]. Mininet is helpful for development, teaching, and research since you can quickly interact with your network using the Mininet CLI (and API), change it, share it with others, or deploy it on actual hardware. OpenFlow and Software-Defined Networking solutions may be developed, shared, and tested in Mininet[4]. Mininet is available under a permissive BSD Open-Source licence and is actively developed and supported. The firewall we are implementing is stateful firewall which intercepts the packets at the network level and then derive, analyse the data from communication layer. It basically individually inspects the packets that are traversing through it. As the primary functionality of this project is to add a firewall on the POX controller, the firewall algorithm is started along with the POX controller, which works in accordance with the firewall policies that are defined. So when a packet is received by the switch from any address, it is sent to the POX, which will comply with the firewall module and push down a 'drop' flow entry to the switch for the packets to be blocked. Hence the switch will block any further packets from that source. The rest of the packets are simply forwarded and corresponding flow-entries are added by the POX controller in the flow-tables of the switches, so that any further packet is managed solely by the switches, without the need to send the packets to the controllers, thus complying with the principles of software defined network.

III. AREA OF THE PROJECT AND OBJCTVES

- Implement the stateful firewall in the SDN application.
- To build a small network on the virtual machine using the "Mininet". Which consists of POX controller, a switch with 6 different hosts.
- Discarding the invalid TCP sessions.
- Area of project includes applications that use the Internet to provide different services, such as online shopping, data transfer, instant messaging, or video conferencing.

IV. LITERATURE SURVEY

Software Defined Networking (SDN decouples the network in two layers; the control plane and the data plane, which provide centralized control of the network. Unlike traditional networks, both the control and data planes are integrated into a single physical device. OpenFlow networks have no security. The aim of this study is to implement a system to improve the security and performance of in SDN by specifying the firewall module in OpenFlow. Two scenarios were implemented, first the network connection was checked before the firewall was implemented. In the second scenario, the firewall has been activated. According to the results, packet analysis and matching rules improve security and do not negatively impact performance. This study recommends implementing distributed firewalls to reduce overhead on the controller, using L3_learning switch to perform the routing functionality[1].

In this paper, To simplify network forensics, security policy modification, and security service insertion, the SDN architecture equips networks with the ability to actively monitor traffic and identify problems. However, the separation of the control and data planes presents new security risks, including saturation, DoS, and man-in-the-middle attacks. In this article, we examine security risks to the data, control, and application planes of SDN. Various security techniques for network-wide security in SDN are discussed after the security platforms that safeguard each of the planes. It is very likely that new security risks may materialise as a result of the introduction of SDN technology gradually. Likewise, Given the current state of security concerns, threat space will certainly expand will spread on traditional networks with SDN-specific security difficulties. However, SDN intends to innovate communication networks, therefore quick anomaly detection automated security procedures will be implemented and prompt action for defence. The available research on SDN's network security supports the idea that it will make possible swift provision of affordable security services.[2]

In SDN, decoupling the control plane from the data plane presents several challenges, including security, reliability, load balancing, and traffic engineering. The biggest security challenges with SDN are Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks. For example, in SDNs, DoS/DDoS attacks could swamp the control plane, data plane, or communication channel. In this white paper, they reviewed and systematized state-of-the-art solutions that address DoS attacks and DDoS attacks in SDN through intrinsic and extrinsic approaches. , prevention or dignified disparagement. In addition, a study of the different approaches and tools used to implement the verified solutions will be carried out. Finally, potential future research directions to counter DoS/DDoS attacks on SDN are highlighted.[3]

In this study, SDN controllers were programmed with a stateful firewall appliance to provide firewall functionality without the support of compromised hardware. A stateful firewall filtered traffic based on the full context of incoming packets; continuously evaluated the overall context of traffic flows, looking for network inputs rather than specific traffic flows. In addition, a flow-based scheduling engine has been implemented in the distributed controllers to improve network scalability. A network cluster with three distributed controllers was set up and we experimented with three independent network topologies. Performance test results show that distributed SDN controllers perform better than a centralized controller.[6]

In this study, The authors have suggested leveraging Open Switch to create the SDN stateful firewall in the data plane. The OpenFlow protocol version 1.5.0, which supports TCP flag matching, and Open Switch with learn action, which may change the rules in the SDN switch, are both used in our implementation. With the help of our implementation, the SDN stateful firewall may be used with the present Open Flow protocol without the need for further control plane and data plane modifications. Further examination could be on assessing the overall performance of the SDN stateful firewall with extra complicated community topology and extra safety assault instances together with SYN flood. In addition, the test that packets despatched from specific reasserts to specific locations concurrently ought to be accomplished to assess the effectiveness of the SDN transfer and how it could cope with the rule updating concurrently.[7]

They have suggested an application-aware firewall strategy for protecting OpenFlow-based SDN networks in this study. Regarding design and behaviour, the solution does not impose any changes on the SDN paradigm. Additionally, it aims to improve security on programmable networks and streamline security administration. By identifying programmes like FTP or limiting streams in accordance with the present policy, this suggested solution aims to secure the small, localised network. The firewall programme above the controller inspects traffic flows and enforces the necessary policy, and the switches function as a distributed checkpoint that carries out the controller's orders. The security policy is centralised on the controller side. Rather than using a gateway firewall with a low performance and a single point of failure, multi plane-like firewall creates layers of defence; “defence-in-depth” can be used successfully for networks with different levels of trust. The future work could increase the level of security provided to the network by building a system that can examine the contents of the packets to prevent any malicious or somehow undesirable content.[10]

The use of hardware load balancers is a critical issue in traditional IP networks that has many negative effects such as: Affordability, customization of features and availability. Also, the existing load balancing algorithm does not take into account the size of the data flow generated by the client nodes. Also, streams are not ranked based on the dynamic stream size threshold. The paper proposes to compare the performance of two load balancing algorithms such as the flow-based load balancing algorithm and the traffic pattern-based load balancing algorithm with distributed controller architecture.[11]

SDN offers simple and effective administration options for virtual machines with software-programmed middleboxes. SDN with a centralised controller also encounters issues with scalability, network congestion, and single point failure. In this study, a distributed SDN-controlled network uses a stateful inspection firewall as a middlebox. Stateful firewalls on SDN architecture are designed to safeguard the network by keeping track of active connections and maintaining their state data till the connection is active. This article measures, compares, and analyses the performance of firewall enabled SDN with centralised and distributed controllers. The trials are carried out with a POX controller, and the Mininet network emulation programme verifies the outcomes. The controller is programmed with a failure detection and recovery mechanism to provide reliability and redundancy and enhance the overall performance of the network.[12]

The goal of this research was to develop a secure OpenFlow called Secured-OF. A stateful firewall was used to store state information for later analysis. Dynamic Bayesian Network (DBN) was used to learn denial of service attacks and distributed denial of service attacks. It analyses a packet's states to determine the nature of an attack and adds this information to the flow table entry. The Secured-OF model proposed in the Ryu controller has been tested against various performance metrics. Analytical evaluation of the proposed Secured-OF scheme was performed on an emulated network. In summary, the security of the OpenFlow controller has been drastically improved with a minor performance drop compared to an SDN with no security implementation.[14]

Security Results in conventional networks are complex and expensive because of the lack of abstraction, the severity, and the diversity of the network armature. still, in Software Defined Networking, flexible, reprogrammable, robust, and cost-effective security results can be erected over the armature. In this environment, we propose a SDN visionary stateful Firewall. Our result is fully integrated into the SDN terrain and it's biddable with the OpenFlow protocol. The proposed Firewall is the first enforced stateful SDN Firewall. It uses a visionary sense to alleviate some characteristic and DoS attacks either, an Orchestrator subcaste is integrated in the Firewall in order to manage the deployment of the Firewall operations. This integration empowers the relations with the director and the data aeroplane rudiments. We conduct two tests to prove the validity of our conception and to show that the proposed Firewall is effective and performant.[15]

The first SDN reactive firewall that have been implemented and the advantages of the system flexibility, performance, security enforcement and have been discussed about the conceptual foundations based on a general firewall algorithm and the deployment of the SDN in the virtual environment and to manage the network according to a holistic policy in this paper. The future enhancement to this paper is to first improvement will focus on the evaluation part. We will deploy the SDN test bed in a real environment. All the SDN elements will be hosted in dedicated powerful machines. We will push the firewall capabilities to their limits to measure the maximum number of connections that it can handle and the impacts on the network performances. In the second enhancement, we propose to develop a meta-firewall on the management plane.[16]

V. PROBLEM DEFINITION

SDN based network supports rapid updating and provides other opportunities to enhance and protect the network, but on the other hand these features bring new vulnerabilities related to security, scalability and flexibility. Since the primary design of the SDN architecture does not sufficiently take into account security requirements what makes security issues a real challenge. Hence a mechanism with application aware capabilities to inspect the network traffic, which can have a plane-like firewall that creates additional boundaries within the network is needed.

VI. PROPOSED WORK

- A small network is set up on the virtual machine, with Mininet installed. This network contains a switch and 3 hosts.
- A python code that instantiates a virtual network(Mininet) independently on any system and in no time.
- Mininet is a network emulator that creates a realistic virtual network, running real kernel, switch and application code, on a single machine.
- The algorithm is started along with the POX controller, which works in accordance with the firewall policies table. So when a packet is received by the switch from a certain IP address, it is sent to the POX, which will comply with the firewall module and push down a 'drop' flow entry to the switch for the packets to be blocked.
- Hence the switch will block any further packets from that source. The rest of the packets are simply forwarded and corresponding flow-entries are added by the POX controller in the flow-tables of the switches, so that any further packet is managed solely by the switches, without the need to send the packets to the controllers, thus complying with the principles of software defined network.

VII. METHODOLOGY

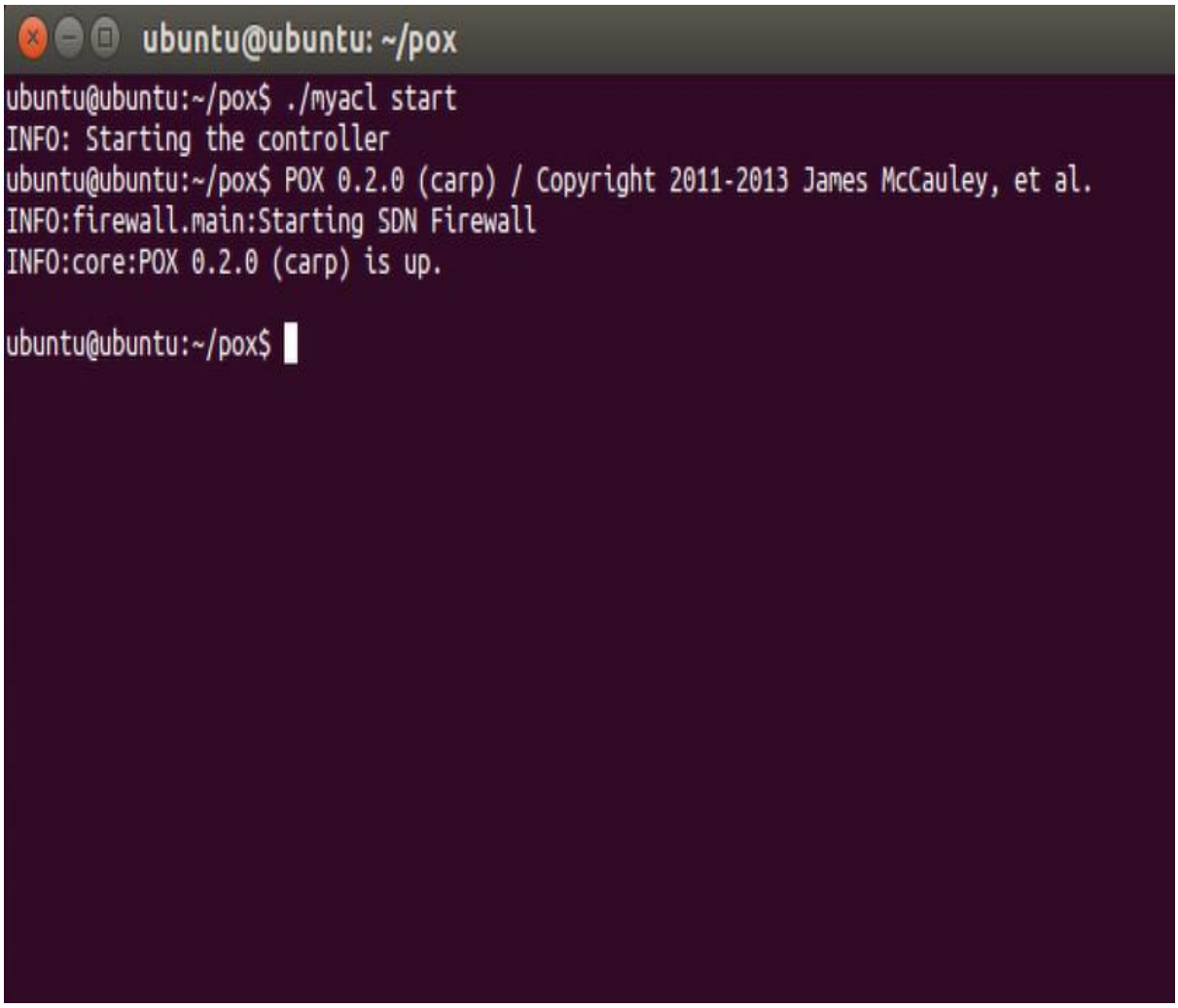
Using “mininet” a small virtual network of one switch and hosts are created, and they are connected to the POX controller. Now, when the connectivity is checked among all hosts, using the command “pingall”, the host having certain ip address will not be able to reach host with another ip address and vice versa as specified in the policies. Hence, the firewall module is working appropriately on the controller. The firewall policies specify which IP addresses must be blocked. This file is being called and used by firewall code, which is a python code that recognizes the IP addresses to be blocked from certain other ip addresses and also directs the controller to add specific flow-entries in the flow tables of the switches, so that packets from these blocked IP addresses are independently handled by the switches in future. The learning algorithm runs along with the POX controller that forces the switches to behave as normal switches that have learning capabilities. When a new packet reaches a switch, the switch acts according to the OpenFlow protocol, in which it sends the packet to the controller, as the switch is unaware of the action, it needs to perform. The controller informs the switch the required action and hence the switch 'learns' the source address and its corresponding action. The specific flow entries are added in the flow tables of the switches. When a new packet reaches a switch, the switch acts according to the OpenFlow protocol that has been defined in the polices Based on the threshold defined in the polices if any host sends more requests to another host in a certain period the firewall drops the packets. The controller informs the switch the required action and hence the switch 'learns' the source address and its corresponding action. The specific flow entries are added in the flow tables of the switches.

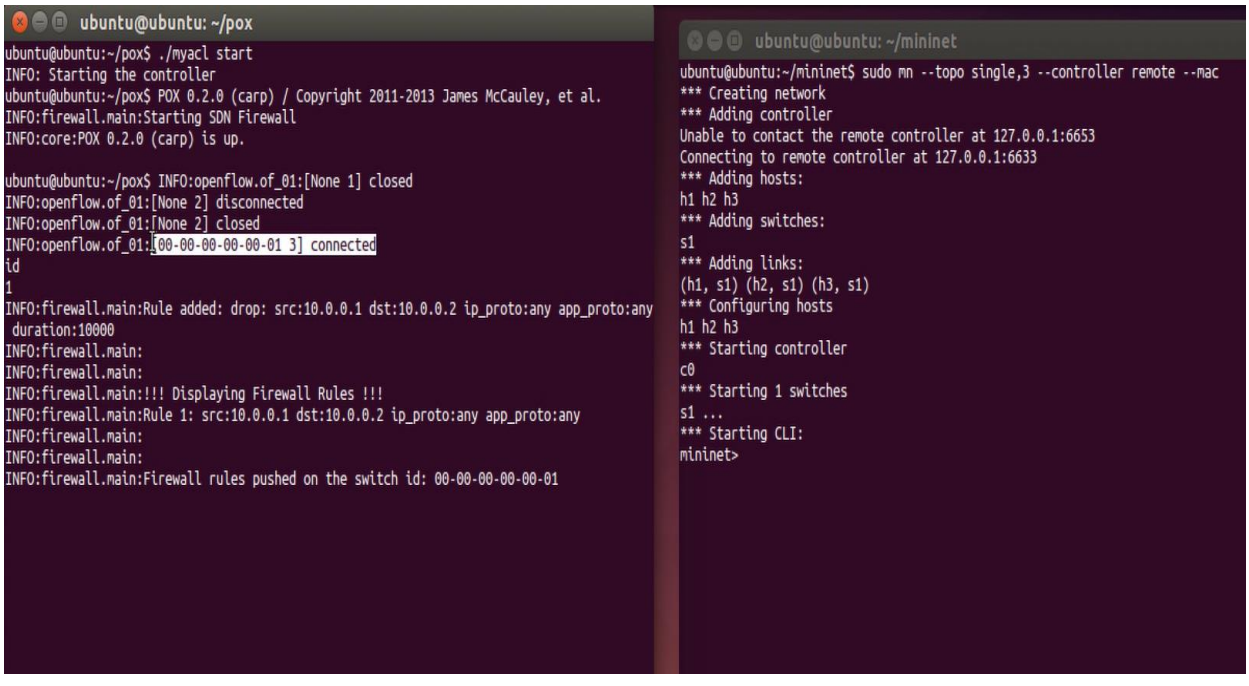
VIII. PROCEDURE

- A csv file is created which has the firewall configurations and the threshold limits in the network, so this file must be run using POX controller in order to channel the packets in the network.
- Start the Pox Controller along with the python firewall program program.
- Create a simple Mininet topology, preferably using one switch and few hosts.
- The configuration file is configured to allow TCP traffic between HTTP port of hosts.
- The program can be tested by any server on the host1 or by sending any request to the blocked destination to check if the firewall can detect and drop the packets.
- Using hping command we can flood a host from the another host so by this if the hosts sends more tcp packets than the threshold limit the firewall drops the packets from that particular host.
- In this way the firewall blocks all the packets from the defined blocked network and detects the malicious host based on the number of requests it sends and drops the packets from that host.

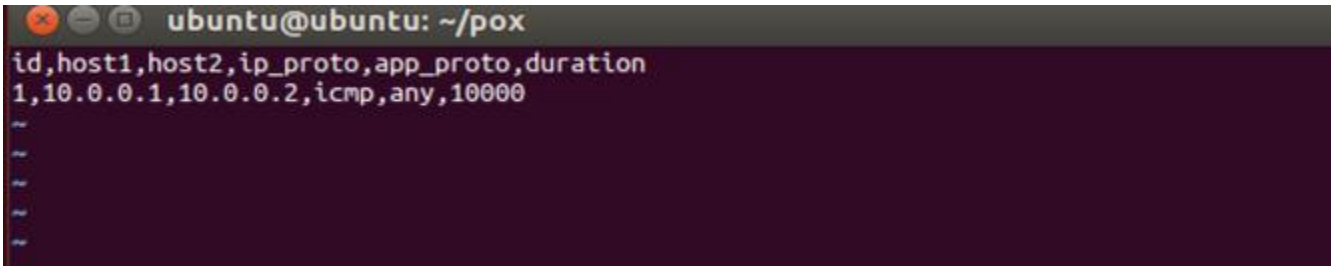
IX. EXPERIMENTAL RESULTS

Output of starting the POX controller and creating a virtual network:

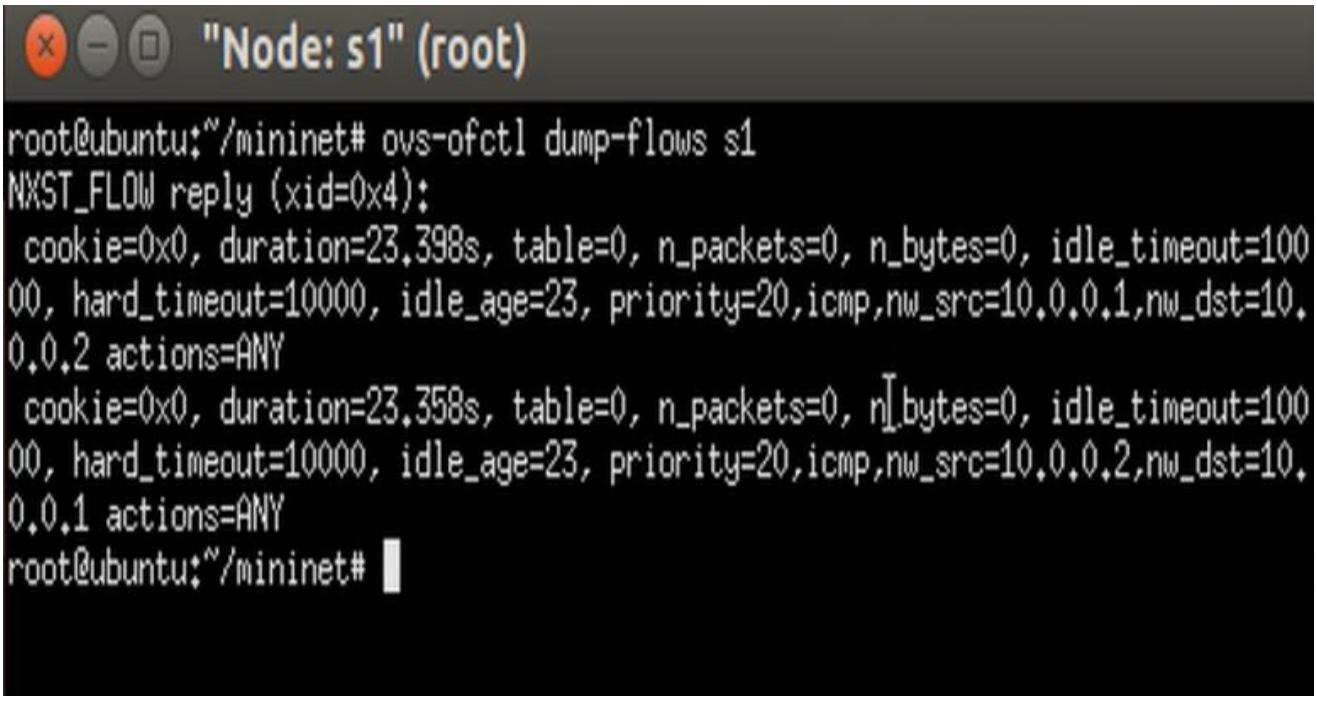




Output of changing the IP rules in IP table:




```
ubuntu@ubuntu:~/pox$ ./myacl restart
INFO: Stopping the controller
INFO: Starting the controller
ubuntu@ubuntu:~/pox$ POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:firewall.main:Starting SDN Firewall
INFO:core:POX 0.2.0 (carp) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
id
1
INFO:firewall.main:Rule added: drop: src:10.0.0.1 dst:10.0.0.2 ip_proto:icmp app_proto:a
y duration:10000
INFO:firewall.main:
INFO:firewall.main:
INFO:firewall.main:!!! Displaying Firewall Rules !!!
INFO:firewall.main:Rule 1: src:10.0.0.1 dst:10.0.0.2 ip_proto:icmp app_proto:any
INFO:firewall.main:
INFO:firewall.main:
INFO:firewall.main:Firewall rules pushed on the switch id: 00-00-00-00-00-01
```



Output of reachability from every host in the network

```
mininet> xterm s1
mininet> pingall
*** Ping: testing ping reachability
h1 -> X h3
h2 -> X h3
h3 -> h1 h2
*** Results: 33% dropped (4/6 received)
mininet>
```

Output of flooding a host from another

```
File  Edit  View  Search  Terminal  Help
root@Hacker:~#
root@Hacker:~# hping3 -SA --flood --rand-source -p 80 10.10.1.112
HPING 10.10.1.112 (eth0 10.10.1.112): SA set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.10.1.112 hping statistic ---
256041 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@Hacker:~#
```

```
description "ETH0 TO ETH1"
enable-default-log
rule 200 {
    action accept
    description STATEFUL_WEB_ACCESS
    destination {
        address 10.10.1.112
        port 80,443
    }
    protocol tcp
    state {
        new enable
    }
}
```

```
----------------------------
Rulesets Information
----------------------------
------------------------------------------------------------------------
IPv4 Firewall "WAN_TO_DMZ":

 Active on (eth0,IN)

rule  packets   bytes     action  source           destination
----  -------   -----     ------  ------           -----------
200   0         0         ACCEPT  0.0.0.0/0        10.10.1.112
10000 235.05K   9.40M     DROP    0.0.0.0/0        0.0.0.0/0
```

X. CONCLUSION:

- SDN provides more flexible, programmable, and innovative network's architecture, which may pose new vulnerabilities and may lead to new security problems.
- As the security requirements are not sufficiently taken into account in the primary design of the SDN architecture which makes security issues a real challenge.
- This mechanism has application aware capabilities to inspect the network traffic, which can have a plane-like firewall that creates additional boundaries within the network by blocking all the packets from the defined blocked network, which can also detect the malicious host based on the number of requests it sends and drops the packets from that host.
- SDN is emulated on Mininet and firewall is implemented at the switch and all packets will traverse only after the switch configures them.

XI. FUTURE ENCHANCEMENT:

- Further improvements can be made on improvements on firewall algorithm to push the firewall capabilities to their limits to measure the maximum number of connections that it can handle and the impacts on the network performances.
- The future work could increase the level of security provided to the network by building a system that can examine the contents of the packets to prevent any malicious or somehow undesirable content.

XII. SAMPLE CODE:

fire.py > Firewall

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
from pox.lib.util import dpidToStr
import pox.lib.packet as pkt
from pox.lib.addresses import EthAddr, IPAddr
import os
import csv


log = core.getLogger()
aclSrc = "%s/pox/pox/firewall/nw_policies.csv" % os.environ[ 'HOME' ]


class Firewall (EventMixin):
    def _init_ (self):
        self.listenTo(core.openflow)
        self.firewall = {}
        log.info("Starting SDN Firewall")
        self.FTP_PORT      = 21
        self.HTTP_PORT     = 80
        self.TELNET_PORT   = 23
        self.SMTP_PORT     = 25

    def pushRuleToSwitch (self, src, dst, ip_proto, app_proto, duration):
        msg = of.ofp_flow_mod()
        msg.priority = 20
        msg.actions.append(of.ofp_action_output(port=of.OFPP_NONE))
        match = of.ofp_match()
        match.dl_type = 0x800;
        d = int(duration)
        if d == 0:
            action = "del"
        else:
            action = "add"
        if not isinstance(d, tuple):
            d = (d,d)
        msg.idle_timeout = d[0]
```

```
110
111     def _handle_ConnectionUp (self, event):
112         acl  = open(aclSrc, "rb")
113
114         self.connection = event.connection
115
116         rulesIterator = csv.reader(acl)
117         for rule in rulesIterator:
118             print(rule[0])
119             if rule[0] != "id" :
120                 self.addFirewallRule(rule[1], rule[2], rule[3], rule[4], rule[5])
121
122         self.showFirewallRules()
123         log.info("")
124         log.info("")
125         log.info("Firewall rules pushed on the switch id: %s", dpidToStr(event.dpid))
126
127
128     def firewall(pkt):
129         sca = IP(pkt.get_payload())
130
131         if(sca.src in ListOfBannedIpAddr):
132             print(sca.src, "is a incoming IP address that is banned by the firewall.")
133             pkt.drop()
134             return
135
136         if(sca.haslayer(TCP)):
137             t = sca.getlayer(TCP)
138             if(t.dport in ListOfBannedPorts):
139                 print(t.dport, "is a destination port that is blocked by the firewall.")
140                 pkt.drop()
141                 return
142
143         if(sca.haslayer(UDP)):
144             t = sca.getlayer(UDP)
145             if(t.dport in ListOfBannedPorts):
```

```
39          msg.hard_timeout = d[1]
40          if ip_proto == "tcp":
41              match.nw_proto = pkt.ipv4.TCP_PROTOCOL
42          if ip_proto == "udp":
43              match.nw_proto = pkt.ipv4.UDP_PROTOCOL
44          elif ip_proto == "icmp":
45              match.nw_proto = pkt.ipv4.ICMP_PROTOCOL
46          elif ip_proto == "igmp":
47              match.nw_proto = pkt.ipv4.IGMP_PROTOCOL
48
49          if app_proto == "ftp":
50              match.tp_dst = self.FTP_PORT
51          elif app_proto == "http":
52              match.tp_dst = self.HTTP_PORT
53          elif app_proto == "telnet":
54              match.tp_dst = self.TELNET_PORT
55          elif app_proto == "smtp":
56              match.tp_dst = self.SMTP_PORT
57
58          if src != "any":
59
60           if dst != "any":
61              match.nw_dst = IPAddr(dst)
62          msg.match = match
63
64          if action == "del":
65                  msg.command=of.OFPFC_DELETE
66                  msg.flags = of.OFPFF_SEND_FLOW_REM
67                  self.connection.send(msg)
68          elif action == "add":
69                  self.connection.send(msg)
70
71          if dst != "any":
72              match.nw_src = IPAddr(dst)
73          if src != "any":
74              match.nw_dst = IPAddr(src)
75          msg.match = match
```

```
76
77          if action == "delete":
78                  msg.command=of.OFPFC_DELETE
79                  msg.flags = of.OFPFF_SEND_FLOW_REM
80                  self.connection.send(msg)
81          elif action == "add":
82                  self.connection.send(msg)
83
84      def addFirewallRule (self, src=0, dst=0, ip_proto=0, app_proto=0, duration = 0, value=True):
85          if (src, dst, ip_proto, app_proto, duration) in self.firewall:
86              log.warning("Rule exists: drop: src:%s dst:%s ip_proto:%s app_proto:%s duration:%s", src, dst, ip_proto, app_proto,
                duration)
87          else:
88              self.firewall[(src, dst, ip_proto, app_proto, duration)]=value
89              self.pushRuleToSwitch(src, dst, ip_proto, app_proto, duration)
90              log.info("Rule added: drop: src:%s dst:%s ip_proto:%s app_proto:%s duration:%s", src, dst, ip_proto, app_proto,
                duration)
91
92      def delFirewallRule (self, src=0, dst=0, ip_proto=0, app_proto=0, duration = 0, value=True):
93          if (src, dst, ip_proto, app_proto) in self.firewall:
94              del self.firewall[(src, dst, ip_proto, app_proto)]
95              self.pushRuleToSwitch(src, dst, ip_proto, app_proto, duration)
96              log.info("Rule Deleted: drop: src:%s dst:%s ip_proto:%s app_proto:%s", src, dst, ip_proto, app_proto)
97          else:
98              log.error("Rule doesn't exist: drop: src:%s dst:%s ip_proto:%s app_proto:%s", src, dst, ip_proto, app_proto)
99
100
101     def showFirewallRules (self):
102         log.info("")
103         log.info("")
104         log.info("!!! Displaying Firewall Rules !!!")
105         rule_num = 1
106         for item in self.firewall:
107              if item[4] != "0":
108                 log.info("Rule %s: src:%s dst:%s ip_proto:%s app_proto:%s", rule_num, item[0], item[1], item[2], item[3])
109              rule_num += 1
110
```

```
146                 print(t.dport, "is a destination port that is blocked by the firewall.")
147                 pkt.drop()
148                 return
149
150         if(True in [sca.src.find(suff)==0 for suff in ListOfBannedPrefixes]):
151             print("Prefix of " + sca.src + " is banned by the firewall.")
152             pkt.drop()
153             return
154
155         if(BlockPingAttacks and sca.haslayer(ICMP)):
156             t = sca.getlayer(ICMP)
157             if(t.code==0):
158                 if(sca.src in DictOfPackets):
159                     temptime = list(DictOfPackets[sca.src])
160                     if(len(DictOfPackets[sca.src]) >= PacketThreshold):
161                         if(time.time()-DictOfPackets[sca.src][0] <= TimeThreshold):
162                             print("Ping by %s blocked by the firewall (too many requests in short span of time)." %(sca.src))
163                             pkt.drop()
164                             return
165                         else:
166                             DictOfPackets[sca.src].pop(0)
167                             DictOfPackets[sca.src].append(time.time())
168                     else:
169                         DictOfPackets[sca.src].append(time.time())
170                 else:
171                     DictOfPackets[sca.src] = [time.time()]
172
173
174             pkt.accept()
175             return
176         pkt.accept()
177
178     nfqueue = NetfilterQueue()
179     nfqueue.bind(1,firewall)
180
181 def launch ():
182     core.registerNew(Firewall)
```

Output of policies and topology:

firewallpolicies.csv

```
id,ip_0,ip_1
1,10.0.0.1,10.0.0.6
2,10.0.0.2,10.0.0.5
3,10.0.0.3,10.0.0.4
5,10.0.0.4,10.0.0.5
```

topology.py > {} Topo

```python
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example created by me."

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

        # 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' )
        FifthHost = self.addHost( 'h5' )
        SixthHost = self.addHost( 'h6' )
        firstSwitch = self.addSwitch( 's1' )
        secondSwitch = self.addSwitch( 's2' )
        thirdSwitch = self.addSwitch( 's3' )
        fourthSwitch = self.addSwitch( 's4' )
        fifthSwitch = self.addSwitch( 's5' )
        sixthSwitch = self.addSwitch( 's6' )

        # Add links
        self.addLink( firstSwitch, FirstHost )
        self.addLink( secondSwitch, SecondHost )
        self.addLink( thirdSwitch, ThirdHost )
        self.addLink( fourthSwitch, FourthHost )
        self.addLink( fifthSwitch, FifthHost )
        self.addLink( sixthSwitch, SixthHost )
        self.addLink( firstSwitch, secondSwitch  )
        self.addLink( secondSwitch, thirdSwitch )
```

```
        self.addLink( secondSwitch, thirdSwitch )
        self.addLink( thirdSwitch, sixthSwitch )
        self.addLink( fourthSwitch, sixthSwitch )
        self.addLink( fourthSwitch, thirdSwitch )
        self.addLink( firstSwitch, fourthSwitch )
        self.addLink( firstSwitch, fifthSwitch )
        self.addLink( secondSwitch, fifthSwitch )
        self.addLink( secondSwitch, sixthSwitch )
        self.addLink( fifthSwitch, thirdSwitch )

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

XIII. REFERENCES:

1.Adam, Saba Bashir Ahmed. "Development a Firewall-Based Mechanism to Enhance Security in Software Defined Networking." PhD diss., University of Gezira, 2021.

2.Ahmad, Ijaz, Suneth Namal, Mika Ylianttila, and Andrei Gurtov. "Security in software defined networks: A survey." IEEE Communications Surveys & Tutorials 17, no. 4 (2015): 2317-2346.

3.Eliyan, Lubna Fayez, and Roberto Di Pietro. "DoS and DDoS attacks in Software Defined Networks: A survey of existing solutions and research challenges." *Future Generation Computer Systems* 122 (2021): 149-171.

4.https://www.youtube.com/results?search_query=firewall+implementation+for+software+defined+networks

5.Javid, Tariq, Tehseen Riaz, and Asad Rasheed. "A layer2 firewall for software defined network." In *2014 Conference on Information Assurance and Cyber Security (CIACS)*, pp. 39-42. IEEE, 2014.

6.Kavin, Balasubramanian Prabhu, S. R. Srividhya, and Wen-Cheng Lai. "Performance Evaluation of Stateful Firewall-Enabled SDN with Flow-Based Scheduling for Distributed Controllers." *Electronics* 11, no. 19 (2022): 3000.

7.Krongbaramee, Pakapol, and Yuthapong Somchit. "Implementation of SDN stateful firewall on data plane using open vSwitch." *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. IEEE, 2018.

8.Maldonado-Lopez, Ferney A., Eusebi Calle, and Yezid Donoso. "Detection and prevention of firewall-rule conflicts on software-defined networking." In *2015 7th international workshop on reliable networks design and modeling (rndm)*, pp. 259-265. IEEE, 2015.

9.Nife, Fahad, and Zbigniew Kotulski. "Multi-level stateful firewall mechanism for software defined networks." In *International Conference on Computer Networks*, pp. 271-286. Springer, Cham, 2017.

10.Nife, Fahad N., and Zbigniew Kotulski. "Application-aware firewall mechanism for software defined networks." *Journal of Network and Systems Management* 28, no. 3 (2020): 605-626.

11.Prabakaran, Senthil, and Ramalakshmi Ramar. "Software defined network: load balancing algorithm design and analysis." *Int. Arab J. Inf. Technol.* 18, no. 3 (2021): 312-318.

12.Prabakaran, Senthil, and Ramalakshmi Ramar. "Stateful firewall-enabled software-defined network with distributed controllers: A network performance study." *International Journal of Communication Systems* 32, no. 17 (2019): e4237.

13.Shin, Seungwon, Lei Xu, Sungmin Hong, and Guofei Gu. "Enhancing network security through software defined networking (SDN)." In *2016 25th international conference on computer communication and networks (ICCCN)*, pp. 1-9. IEEE, 2016.

14.Sophakan, Natnaree, and Chanboon Sathitwiriyawong. "A Secured OpenFlow-Based Software Defined Networking Using Dynamic Bayesian Network." In *2019 19th International Conference on Control, Automation and Systems (ICCAS)*, pp. 1517-1522. IEEE, 2019.

15.Zerkane, Salaheddine, David Espes, Philippe Le Parc, and Frederic Cuppens. "A proactive stateful firewall for software defined networking." In International Conference on Risks and Security of Internet and Systems, pp. 123-138. Springer, Cham, 2018.

16. Zerkane, Salaheddine, David Espes, Philippe Le Parc, and Frederic Cuppens. "Software defined networking reactive stateful firewall." In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pp. 119-132. Springer, Cham, 2016.