

Student Name: Shachi Shah

Email: spshah22@asu.edu

Submission Date: 18 June 2025

Class Name and Term: CSE548 Summer 2025

SDN-Based DoS Attacks and Mitigation using POX and Mininet

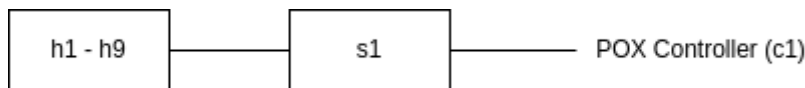
I. PROJECT OVERVIEW

This project involved simulating a Denial of Service (DoS) attack in a Software Defined Networking (SDN) environment using Mininet, POX Controller, and Open vSwitch. The goal was to demonstrate how a malicious host can flood the controller with spoofed packets and then implement mitigation using OpenFlow rules and port security techniques. The project was successfully completed by launching an attack from a compromised host and blocking it using firewall configuration.

II. NETWORK SETUP

The network was created using Mininet's single topology with 9 containernet hosts (h1 through h9) connected to a single Open vSwitch (s1) and a remote POX controller running on port 6655. IP addresses were auto-assigned by Mininet using the 10.0.0.0/8 network.

Topology Diagram:



- All hosts connect to switch s1, which in turn connects to the external POX controller on port 6655.

Initial connectivity:

- Verified using `mn --test pingall` with 0% packet loss.
- Confirmed successful reachability between hosts before launching any attacks.

III. SOFTWARE

Each software component played a critical role in enabling this simulation:

- Mininet 2.3.0d5: For simulating the SDN network environment
- POX 0.5.0 (eel): As the SDN controller to run L3 learning and firewall logic.
- Open vSwitch 2.9.5: As the SDN data plane.
- hping3 3.0.0-alpha-2: To simulate TCP SYN flood attack.
- Ubuntu 18.04: Base system for running all components.

IV. PROJECT DESCRIPTION

The steps below outline the entire process from environment validation to defense implementation and testing.

- Step 1: Environment Verification
 - Commands:
 - `python --version, python3 --version, mn --version, ovs-vsctl --version, wget --version, hping3 --version` [Screenshot 1].
Terminal showing all version checks and 0% dropped from pingall
 - `sudo mn --test pingall` [Screenshot 2].
 - This confirmed the availability of Python, Mininet, Open vSwitch, and traffic generation tools.
- Step 2: Lab Setup and Configuration
 - Purpose: Install necessary configuration files and custom firewall logic.
 - Actions:

- Downloaded lab files from GitHub.
 - Extracted the archive. [Screenshot 3].
 - Copied L3Firewall.py into the POX forwarding directory.
 - Copied l2firewall.config and l3firewall.config into the main POX directory. [Screenshot 4].
- Step 3: Start POX Controller
 - Purpose: Launch the SDN controller with learning switch behavior and custom firewall logic.
 - `cd ~/pox`
 - `sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l3_learning pox.forwarding.L3Firewall`
 - This POX terminal confirms POX is up and listening on port 6655. [Screenshot 5].
- Step 4: Launch Mininet Topology
 - Purpose: Start the SDN emulation with hosts and controllers.
 - `sudo mn --topo=single,9 --controller=remote,port=6655 --switch=ovsk --mac` [Screenshot 6].
 - This created 9 hosts, a single switch, and connected it to the POX controller. For alignment with the project rubric, the first four hosts (h1, h2, h3, h4) were configured with static IP addresses using the following commands: [Screenshot 7].
 - `h1 ifconfig h1-eth0 192.168.2.10/24`
 - `h2 ifconfig h2-eth0 192.168.2.20/24`
 - `h3 ifconfig h3-eth0 192.168.2.30/24`
 - `h4 ifconfig h4-eth0 192.168.2.40/24`
 - These IP assignments were verified through inter-host ping tests before launching any attack simulations. Mininet output confirming creation of h1 through h9, s1, controller, and manual IP configurations.
 - `sudo mn --topo=single,9 --controller=remote,port=6655 --switch=ovsk --mac`
 - This created 9 hosts, a single switch, and connected it to the POX controller. Mininet output confirming creation of h1 through h9, s1, and controller. [Screenshot 8].
- Step 5: Simulate DoS Attack
 - Purpose: Launch spoofed TCP SYN-flood from attacker (h1) to target (h2).
 - `containernet> xterm h1`
 - `hping3 10.0.0.2 -c 10000 -S --flood --rand-source -V`
 - This floods the controller with new spoofed flows, overwhelming its processing capacity. Xterm window of h1 flooding h2 with spoofed packets. [Screenshot 8].
- Step 6: Observe Switch Behavior
 - Purpose: View the switch's flow table to detect flood activity.
 - `sudo ovs-ofctl dump-flows s1`
 - Flow entries showed repeated new connections from the same MAC but different source IPs. Shows multiple flow entries from the same MAC but different spoofed IPs (DoS evidence). [Screenshot 9].
- Step 7: Impact on Other Hosts
 - Purpose: Demonstrate that the controller is too busy to handle new flows.
 - `containernet> h4 ping h9`
 - Due to the attack, ping responses failed or showed high packet loss. Failed ping or high packet loss due to controller being overwhelmed. [Screenshot 10].
- Step 8: Port Security Rule Setup
 - Purpose: Block attacker at layer 2 MAC address
 - `cat ~/pox/l2firewall.config`
 - Added rule:
 - `1,00:00:00:00:00:01,00:00:00:00:00:02`
 - This rule drops traffic from attacker MAC destined for target MAC. l2firewall.config content blocking attacker MAC. [Screenshot 11 & 12].
- Step 9: Restart POX with Mitigation
 - Purpose: Reload POX controller with updated firewall config.
 - `sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l3_learning pox.forwarding.L3Firewall`
 - The rule now prevents spoofed flows from h1 from reaching h2. POX restarted, now applying firewall rules. [Screenshot 13].
- Step 10: Retry DoS Attack
 - Purpose: Test effectiveness of firewall rule.
 - `hping3 10.0.0.2 -c 10000 -S --flood --rand-source -V`
 - Despite launching the attack, no new flow entries were created, confirming mitigation. No visible effect — attacker now blocked by firewall. [Screenshot 14].

- Step 11: Ping test After Mitigation
 - Purpose: Verify network has recovered and traffic flows as expected.
 - `containernet> h4 ping h9`
 - Ping responses were consistent with low latency and no packet loss. Successful and consistent ICMP replies from h9. [Screenshot 15].
- Step 12: Flow Table After Mitigation
 - Purpose: Confirm the firewall rule is active in the switch.
 - `ovs-ofctl dump-flows s1`
 - Output showed:
 - `priority=65535,dl_src=00:00:00:00:00:01 actions=drop`
 - Flow table with drop rule applied to attacker MAC. [Screenshot 16].

V. CONCLUSION

This project provided a comprehensive, hands-on experience in both offensive and defensive aspects of network security within a Software Defined Networking (SDN) environment. By simulating a Denial of Service (DoS) attack using spoofed TCP SYN packets, I was able to observe how an SDN controller becomes vulnerable when inundated with new, unverified flows. This scenario reflects a real-world challenge in SDN deployments, where centralized controllers are susceptible to overload without proper safeguards.

Throughout the project, I not only confirmed the ease with which such an attack can be launched but also explored its consequences, such as legitimate flow denial and degraded network performance. The use of `hping3` as a spoofing tool and Mininet as a virtual lab environment helped replicate attack conditions without needing physical hardware.

The defensive strategy—implementing MAC-based filtering using custom rules in POX—was both elegant and efficient. I learned how OpenFlow rules can be applied at the switch level to preemptively block malicious actors by monitoring MAC-IP mappings and dropping suspicious flows. This underscored the power of SDN's programmability and its potential in automated threat mitigation.

Ultimately, this project deepened my understanding of SDN controller dynamics, flow table management, and reactive security policies. It also emphasized the importance of crafting both flexible and proactive security architectures in programmable networks. I feel more confident in configuring and defending SDN environments and appreciate the crucial role that OpenFlow-based mitigation techniques can play in real-world deployments.

VI. APPENDIX B: ATTACHED FILES

Screenshots:

- 1) Screenshot 1: Setup

```

ubuntu@ubuntu:~$ V
V: command not found
ubuntu@ubuntu:~$ clear
ubuntu@ubuntu:~$ python --version
Python 2.7.17
ubuntu@ubuntu:~$ python3 --version
Python 3.6.9
ubuntu@ubuntu:~$ mn --version
2.3.0d5
ubuntu@ubuntu:~$ sudo mn --test pingall
[sudo] password for ubuntu:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.796 seconds

```

2) Screenshot 2: sudo mn --test pingall

```

ubuntu@ubuntu:~$ sudo mn --test pingall
[sudo] password for ubuntu:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 5.781 seconds
ubuntu@ubuntu:~$

```

3) Screenshot 3: Download and Setup the Firewall

```

ubuntu@ubuntu:~$ wget https://github.com/SaburH/CSE548/raw/main/lab-cs-cns-00103.zip
--2025-06-17 11:22:00-- https://github.com/SaburH/CSE548/raw/main/lab-cs-cns-00103.zip
Resolving github.com (github.com)... 140.82.116.3
Connecting to github.com (github.com)|140.82.116.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/SaburH/CSE548/main/lab-cs-cns-00103.zip
[following]
--2025-06-17 11:22:00-- https://raw.githubusercontent.com/SaburH/CSE548/main/lab-cs-cns-00103.zip
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.110.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5365 (5.2K) [application/zip]
Saving to: 'lab-cs-cns-00103.zip.1'

lab-cs-cns-00103.zip 100%[=====] 5.24K --.-KB/s in 0s

2025-06-17 11:22:01 (31.2 MB/s) - 'lab-cs-cns-00103.zip.1' saved [5365/5365]

ubuntu@ubuntu:~$ unzip lab-cs-cns-00103.zip
Archive: lab-cs-cns-00103.zip
replace lab-cs-cns-00103/l3firewall.config? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
  inflating: lab-cs-cns-00103/l3firewall.config
  inflating: __MACOSX/lab-cs-cns-00103/._l3firewall.config
  inflating: lab-cs-cns-00103/l2firewall.config
  inflating: __MACOSX/lab-cs-cns-00103/._l2firewall.config
  inflating: lab-cs-cns-00103/.DS_Store
  inflating: __MACOSX/lab-cs-cns-00103/._.DS_Store
  inflating: lab-cs-cns-00103/L3Firewall.py
  inflating: __MACOSX/lab-cs-cns-00103/._L3Firewall.py
  inflating: __MACOSX/._lab-cs-cns-00103
ubuntu@ubuntu:~$

```

4) Screenshot 4: Copied config files into POX Directory

```

ubuntu@ubuntu:~$ sudo cp lab-cs-cns-00103/L3Firewall.py ~/pox/pox/forwarding/
ubuntu@ubuntu:~$ sudo cp lab-cs-cns-00103/l2firewall.config ~/pox/
ubuntu@ubuntu:~$ sudo cp lab-cs-cns-00103/l3firewall.config ~/pox/
ubuntu@ubuntu:~$

```

5) Screenshot 5: Start POX Controller with Firewall

```

ubuntu@ubuntu:~$ cd ~/pox
ubuntu@ubuntu:~/pox$ sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l3_learning pox.forwarding.L3Firewall
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.

```

6) Screenshot 6: Launch Mininet Topology

```

ubuntu@ubuntu:~/pox$ sudo mn --topo=single,9 --controller=remote,port=6655 --switch=ovsk --mac
[sudo] password for ubuntu:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1) (h7, s1) (h8, s1) (h9, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
containernet>

```

7) Screenshot 7: Creating First Four Hosts

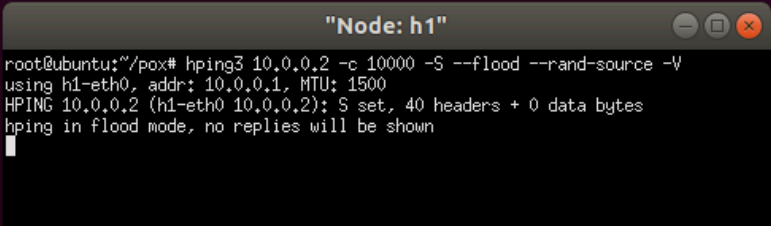
```

containernet> h1 ifconfig h1-eth0 192.168.2.10/24
containernet> h2 ifconfig h2-eth0 192.168.2.20/24
containernet> h3 ifconfig h3-eth0 192.168.2.30/24
containernet> h4 ifconfig h4-eth0 192.168.2.40/24
containernet>

```

8) Screenshot 8: Simulate DoS Attack from h1 in xterm 1

```
ubuntu@ubuntu:~/pox$ sudo mn --topo=single,9 --controller=remote,port=6655 --switch
=ovsk --mac
[sudo] password for ubuntu:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1) (h5, s1) (h6, s1) (h7, s1) (h8, s1) (h9, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
containernet> xterm h1
containernet> 
```



9) Screenshot 9: Observe Switch Flow Table (Pre-Mitigation)

```
ubuntu@ubuntu:~/pox$ sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=220.526s, table=0, n_packets=146, n_bytes=6200, priority=6553
5,dl_src=00:00:00:00:00:01 actions=drop
ubuntu@ubuntu:~/pox$ 
```

10) Screenshot 10: Verify Network Recovery

```
containernet> h4 ping h9
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
64 bytes from 10.0.0.9: icmp_seq=1 ttl=64 time=0.175 ms
64 bytes from 10.0.0.9: icmp_seq=2 ttl=64 time=0.044 ms
64 bytes from 10.0.0.9: icmp_seq=3 ttl=64 time=0.043 ms
64 bytes from 10.0.0.9: icmp_seq=4 ttl=64 time=0.040 ms
64 bytes from 10.0.0.9: icmp_seq=5 ttl=64 time=0.047 ms
64 bytes from 10.0.0.9: icmp_seq=6 ttl=64 time=0.046 ms
^C
--- 10.0.0.9 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5110ms
rtt min/avg/max/mdev = 0.040/0.065/0.175/0.049 ms
containernet> 
```

11) Screenshot 11: Mitigate Attack with Port Security_1

```

ubuntu@ubuntu:~$ cd ~/pox
ubuntu@ubuntu:~/pox$ sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l3_learning pox.forwarding.L3Firewall
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
00:00:00:00:00:01 None
Ip_packet.protocol = 1
^CINFO:core:Going down...
INFO:openflow.of_01:[00-00-00-00-00-01 2] disconnected
INFO:core:Down.
ubuntu@ubuntu:~/pox$ nano ~/pox/l2firewall.config
ubuntu@ubuntu:~/pox$

```

12) Screenshot 12: Mitigate Attack with Port Security_2

```

GNU nano 2.9.3 /home/ubuntu/pox/l2firewall.config

id,mac_0,mac_1
1,00:00:00:00:00:01,any

```

13) Screenshot 13: Restarted POX with Firewall

```

ubuntu@ubuntu:~/pox$ sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.l3_learning pox.forwarding.L3Firewall
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
00:00:00:00:00:01 None

```

14) Screenshot 14:

```

"Node: h1"
root@ubuntu:~/pox# hping3 10.0.0.2 -c 10000 -S --flood --rand-source -V
using h1-eth0, addr: 10.0.0.1, MTU: 1500
HPING 10.0.0.2 (h1-eth0 10.0.0.2): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.0.0.2 hping statistic ---
85856028 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@ubuntu:~/pox# hping3 10.0.0.2 -c 10000 -S --flood --rand-source -V
using h1-eth0, addr: 10.0.0.1, MTU: 1500
HPING 10.0.0.2 (h1-eth0 10.0.0.2): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown

```

15) Screenshot 15: Verify Network Recovery

```

containernet> h4 ping h9
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
64 bytes from 10.0.0.9: icmp_seq=2 ttl=64 time=0.224 ms
64 bytes from 10.0.0.9: icmp_seq=3 ttl=64 time=0.043 ms
64 bytes from 10.0.0.9: icmp_seq=4 ttl=64 time=0.060 ms
64 bytes from 10.0.0.9: icmp_seq=5 ttl=64 time=0.070 ms
64 bytes from 10.0.0.9: icmp_seq=6 ttl=64 time=0.053 ms
64 bytes from 10.0.0.9: icmp_seq=7 ttl=64 time=0.054 ms
64 bytes from 10.0.0.9: icmp_seq=8 ttl=64 time=0.164 ms
64 bytes from 10.0.0.9: icmp_seq=9 ttl=64 time=0.061 ms
64 bytes from 10.0.0.9: icmp_seq=10 ttl=64 time=0.033 ms
64 bytes from 10.0.0.9: icmp_seq=11 ttl=64 time=0.045 ms
64 bytes from 10.0.0.9: icmp_seq=12 ttl=64 time=0.045 ms

```

16) Screenshot 16: Final Flow Table Check

```

ubuntu@ubuntu:~/pox$ sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=243.483s, table=0, n_packets=243, n_bytes=10206, priority=655
35,dl_src=00:00:00:00:00:01 actions=drop
cookie=0x0, duration=140.191s, table=0, n_packets=102, n_bytes=9324, actions=NORMAL
ubuntu@ubuntu:~/pox$

```

VII. REFERENCES

- [1] Sabur Hossain, Github lab files, available at <https://github.com/SaburH/CSE548/>, accessed by 06/17/2025.
- [2] Open vSwitch, available at <https://www.openvswitch.org/>, accessed by 06/17/2025.
- [3] POX Controller, available at <https://github.com/noxrepo/pox>, accessed by 06/17/2025.