Introduction

Distributed Denial-of-Service attack (DDoS attack) is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. Denial of service is typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled.

In a DDoS attack, the incoming traffic flooding the victim originates from many different sources. This effectively makes it impossible to stop the attack simply by blocking a single source.

A DDoS attack is analogous to a group of people crowding the entry door of a shop, making it hard for legitimate customers to enter, thus disrupting trade.

The **objective** is to emulate a network with multiple hosts and switches, create traffic in that network, launch DDoS attack and it's early detection so that it can be prevented.

Implementation

1. Network Creation:

The network emulator used is Mininet for creation of network topology as shown below.

Download and install Mininet from here.

Use the following command to create a network of 9 switches and 64 hosts.

sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --controller=remote,ip=127.0.0.1,port=6633

```
apoorva@apoorva-VirtualBox:~/Projects/SecureCoding/mininet/mininet$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --controller=remote,ip=127.0.0.1,port=6633
[sudo] password for apoorva:
Sorry, try again.
[sudo] password for apoorva:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h
54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s1, s6) (s1, s7) (s1, s8) (s1, s9) (s2, h1) (s2, h2) (s2, h3) (s2, h4) (s2, h5) (s2, h6) (s2, h7) (s2, h8) (s3, h9) (s3, h10) (s3, h11) (s3, h12) (s3, h13) (s3, h14) (
s3, h15) (s3, h16) (s4, h17) (s4, h18) (s4, h19) (s4, h20) (s4, h21) (s4, h22) (s4, h23) (s4, h24) (s5, h25) (s5, h26) (s5, h27) (s5, h28) (s5, h29) (s5, h30) (s5, h31) (s5, h32) (s6, h33) (s6, h34) (s6,
h35) (s6, h36) (s6, h37) (s6, h38) (s6, h39) (s6, h40) (s7, h41) (s7, h42) (s7, h43) (s7, h44) (s7, h45) (s7, h46) (s7, h47) (s7, h48) (s8, h49) (s8, h50) (s8, h51) (s8, h52) (s8, h53) (s8, h54) (s8, h55)
 (s8, h56) (s9, h57) (s9, h58) (s9, h59) (s9, h60) (s9, h61) (s9, h62) (s9, h63) (s9, h64)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h49 h50 h51 h52 h53 h
54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64
*** Starting controller
c0
*** Starting 9 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 ...
*** Starting CLI:
mininet>
```

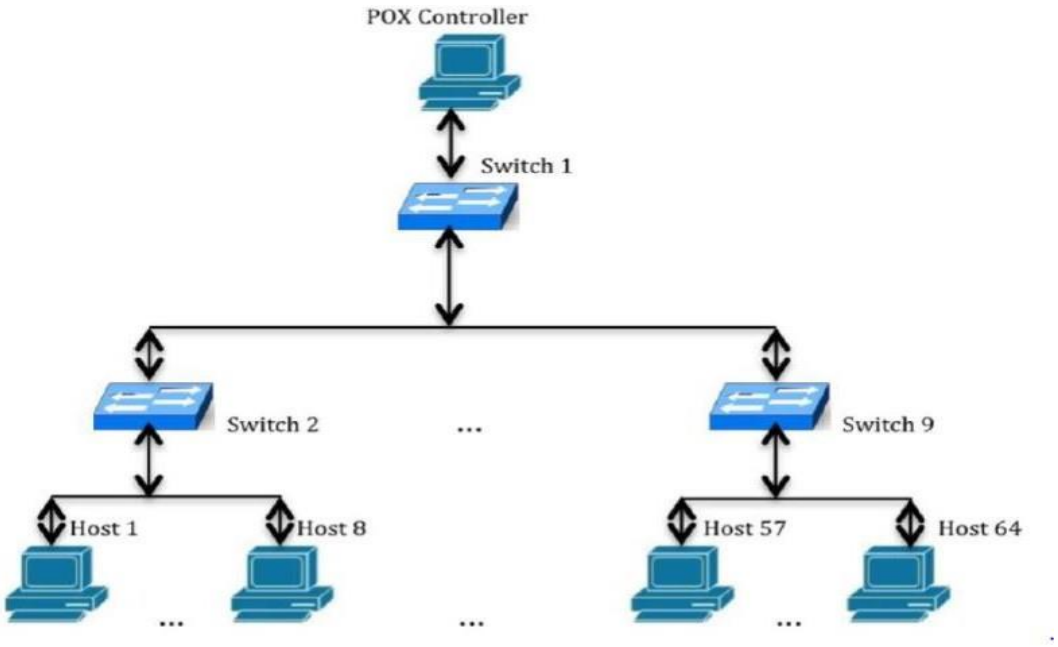


Fig 1: Network Topology

2. Network Monitoring:

POX SDN controller is used to monitor the network traffic and it's entropy. It is present as part of Mininet. POX provides a framework for communicating with SDN (Software defined networks) switches using either the OpenFlow or OVSDB protocol. Developers can use POX to create an SDN controller using the Python programming language.

I have modified the POX controller as per the needs.

Start the POX controller as follows:

cp l3_learning_edit.py <POX_dir>/pox/forwarding
cp detection.py <POX_dir>/pox/forwarding
cd <POX_dir>
python pox.py pox.forwarding.l3_learning_edit

```
apoorva@apoorva-VirtualBox:~/Projects/SecureCoding/pox$ python pox.py pox.forwarding.editted_l3_pox_controller
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
INFO:openflow.of_01:[00-00-00-00-00-06 2] connected
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:openflow.of_01:[00-00-00-00-00-04 8] connected
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
INFO:openflow.of_01:[00-00-00-00-00-08 4] connected
INFO:openflow.of_01:[00-00-00-00-00-07 6] connected
INFO:openflow.of_01:[00-00-00-00-00-09 7] connected
INFO:openflow.of_01:[00-00-00-00-00-05 9] connected
```

l3_learning_edit.py is modified to collect the statistics of the network and calculate it's Entropy to take decision regarding DDoS prevention.

Explanation:

In DDoS, if the source addresses of incoming are spoofed, then the switch would not find a match and the packet needs to be forwarded to the controller. The collection of DDoS spoofed packets and legitimate packets can bind the controller into continuous processing, which exhausts them. Due to this, the controller is unreachable for the new incoming legitimate packets. This will bring the controller down causing loss to the SDN architecture. For a backup controller, the same challenge is to be faced.

Such kind of attacks can be detected in the early stage by monitoring few hundred of packets considering changes in entropy. The early detection of DDoS attacks stops the controller from going down. The term 'early' is related to tolerance level and traffic being handled by the controller. Due to this, the impact of malicious packets flooding can be controlled. Such a mechanism needs to be lightweight and high response time. The high response time saves the controller during the attack period for regaining the control by terminating the DDoS attack.

Why entropy?

The main reason for considering entropy is its ability for measuring randomness in a network. The higher the randomness the lower is the entropy and vice versa. So, whenever the entropy is less than a threshold value, we can say that a DDoS attack has occurred.

3. Traffic Generation

Traffic generation is done with the help of **Scapy**. It is used for generation of packets, sniffing, scanning, forging of packet and attacking. Scapy is used for generation of UDP packets and spoofing the source IP address of the packets.

Install Scapy:

sudo apt-get install python-scapy

Run the traffic generation script from one of the host which generates random source IP and send the packets to random destinations. Observe the entropy in POX controller terminal.

cp Trafficlauch.py <mininet_dir>/custom

cp attacklaunch.py <mininet_dir>/custom

Open xterm for a host by typing the following command in mininet terminal:

mininet>xterm h1

In xterm window of host h1, run the following command:

cd <mininet_dir>/custom

python launchTraffic.py –s 2 –e 65

```
Sent 1 packets.
HELOOOOO
<Ether  type=0x800 |<IP  frag=0 proto=udp src=197.77.157.122 dst=10.0.0.49 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
HELOOOOO
<Ether  type=0x800 |<IP  frag=0 proto=udp src=98.189.114.16 dst=10.0.0.39 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
HELOOOOO
<Ether  type=0x800 |<IP  frag=0 proto=udp src=57.175.144.111 dst=10.0.0.61 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
HELOOOOO
<Ether  type=0x800 |<IP  frag=0 proto=udp src=193.61.2.30 dst=10.0.0.40 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
HELOOOOO
<Ether  type=0x800 |<IP  frag=0 proto=udp src=119.41.35.57 dst=10.0.0.18 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
HELOOOOO
<Ether  type=0x800 |<IP  frag=0 proto=udp src=132.91.234.196 dst=10.0.0.32 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
HELOOOOO
<Ether  type=0x800 |<IP  frag=0 proto=udp src=154.20.232.111 dst=10.0.0.40 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
HELOOOOO
<Ether  type=0x800 |<IP  frag=0 proto=udp src=22.126.63.41 dst=10.0.0.11 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
HELOOOOO
<Ether  type=0x800 |<IP  frag=0 proto=udp src=183.172.143.178 dst=10.0.0.7 |<UDP  sport=2 dport=http |>>>
.
Sent 1 packets.
HELOOOOO
<Ether  type=0x800 |<IP  frag=0 proto=udp src=228.134.205.127 dst=10.0.0.58 |<UDP  sport=2 dport=http |>>>
.
```

4. Launch Attack

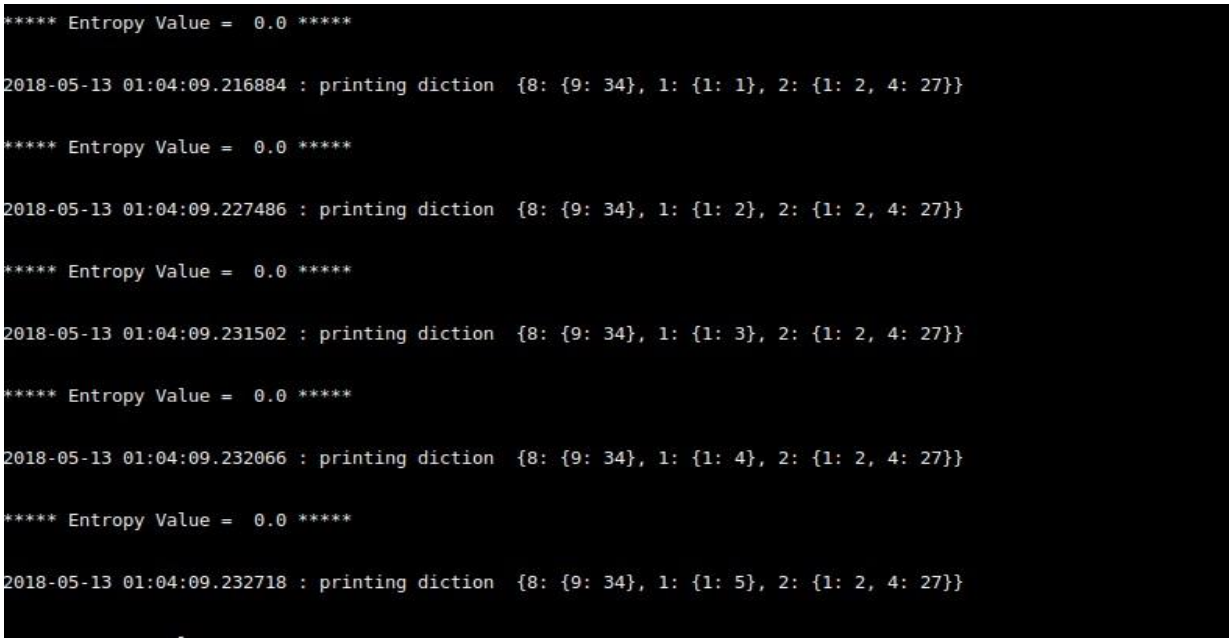
Now repeat above step on h1 and parallelly enter the following command to run the attack traffic from h4 and h6 xterm windows to attack on h56.

python launchAttack.py 10.0.0.56

Entropy value before the DDoS attack:

```
INFO:forwarding.detection:0.66219900103
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.735509926007
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.808820850984
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.842800251071
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.916111176048
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:0.989422101025
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.0999203515
INFO:forwarding.detection:Entropy =
INFO:forwarding.detection:1.17323127648
INFO:forwarding.detection:{IPAddr('10.0.0.7'): 2, IPAddr('10.0.0.52'): 3, IPAddr('10.0.0.64'): 3, IPAddr('10.0.0.
11'): 3, IPAddr('10.0.0.34'): 3, IPAddr('10.0.0.40'): 6, IPAddr('10.0.0.54'): 3, IPAddr('10.0.0.39'): 3, IPAddr('
10.0.0.37'): 2, IPAddr('10.0.0.10'): 3, IPAddr('10.0.0.12'): 3, IPAddr('10.0.0.8'): 1, IPAddr('10.0.0.20'): 3, IP
Addr('10.0.0.27'): 3, IPAddr('10.0.0.32'): 6, IPAddr('10.0.0.25'): 3}

***** Entropy Value =  1.17323127648 *****


***** Entropy Value =  1.17323127648 *****


***** Entropy Value =  1.17323127648 *****


***** Entropy Value =  1.17323127648 *****


***** Entropy Value =  1.17323127648 *****
```

Entropy value after DDoS attack:



Observing the entropy values in the POX controller. The value decreases below the threshold value (which is equal to 0.5 here) for normal traffic. Thus, we can detect the attack within the first 250 packets of malicious type of traffic attacking a host in the SDN network.

After the hosts stop sending attack packets, the switches are started again by the POX controller after shutting them down during the attack.

5. DDoS Detection

First, make a count of 50 packets in a window and then calculate the entropy and compare it with threshold and make a count of consecutive entropy value lower than threshold. If this count reaches 5 then DDoS had occurred otherwise not.

For this, a detection script is created and some changes in the l3_learning module of the pox controller are also done so that it can detect the DDoS. These changes are explained below.

Entropy formula

Entropy is detected with the help of 2 factors:

1. Destination IP
2. No. of times it repeated.

Window size is kept as 50 and probability of a destination IP occurred in the window is given as pi

pi=(xi)/n where x is no. of event in the set and n to be the window size.

Now,

entropy H= - sum of all (pi)log(pi) where i is from o to n

The sequence of steps we use in the detection code is given as a flow below.