

PROJECT MANUAL

1. SOFTWARE INSTALLATION

This section lists all the essential software that must to be installed prior to the start of this project.

1.1. LINUX OS

- Dual boot or Install the latest version of Ubuntu (32bit/64bit) machine available online based on the system requirements.
- Install any virtualization platform such as VMware Workstation or Oracle VirtualBox.

1.2. MININET (using VM)

- Download the Mininet VM from below location and import it into the virtual machine.
<https://github.com/mininet/mininet/wiki/Mininet-VM-Images>
- Set the virtual machine settings to allow X11 forwarding. This is essential to run the XTerm application that is required during this project.
- Mininet comes pre-installed with the latest version of Open vSwitch.

1.2. POX Controller

- Once the Mininet has been successfully installed within the VM, restart the VM. Install the POX controller within the home directory in the Mininet.
- The latest and stable release of POX controller can be obtained from its git repository. It can be cloned into the Mininet VM using the below link.
\$ git clone http://github.com/noxrepo/pox

2. COMPONENTS

Once different software that is required for this project are installed successfully, there are certain configuration settings that must be performed before starting this project. This section provides the details regarding various components involved in this project.

2.1. Network Topology

- The network topology for this project is available in custom_network.py file. It consists of 2 switches, 6 hosts and 1 HTTP server.

S1 and S2 = Switch 1 and 2.

h1 = Bot master

h2 - h5 = slave Botnet Hosts

h6 = HTTP Server

h7 = Legitimate host

- Save the custom_network.py file within a separate folder in the Mininet.
/home/mininet/new_folder

- In addition, save below files within the same directory.

basic_server.py = General HTTP Server, handles page requests from clients.
server.py = HTTP Server where CAPTCHA mode and Honey-token features are enabled.
client.py = General HTTP Client to request pages from the server
hclient.py = HTTP Client that requests for the HONEYTOKEN page – ‘classified.txt’.
master.py = Bot master program to control the botnet hosts.
slave.py = Slave program for Botnet hosts that initiate HTTP Flood attack.
asdf.txt = Legitimate HTTP file.
classified.txt = Honey-token files.

2.2. DDoS Blocking Application on POX

- The DDoS Blocking application (DBA) is the POX component that mitigates the DDoS attack on the HTTP server in this project. This component is defined in **blocking_app.py** file.
- Save **blocking_app.py** file within the *pox/ext* folder within the Mininet VM. POX automatically fetches this component from this folder when initiated.

3. IMPLEMENTATION

This section provides a detailed explanation regarding how this project can be performed. Each step is carefully explained and must be followed in the same sequence as listed here. In addition, screenshots are also provided in suitable locations to aid in understanding the procedure.

3.1. ATTACK Mode

In this mode, the Botnet based DDoS attack scenario is demonstrated.

- Start the Mininet VM.
- SSH into the Mininet using terminal on the Ubuntu machine using below command-
`ssh -X mininet@<IP address>`
Enter the IP address of the Ethernet port on the Mininet VM in the above command.
- Upon success, the Mininet prompt must be visible on the Ubuntu terminal.
- Establish two SSH sessions in two different Ubuntu terminals using above command.
- In one session, navigate to the *new_folder* created in section 2.1. This is the session where the network topology will be initiated.

```
gouthamp@gouthamp:~$ ssh -X mininet@192.168.56.101
mininet@192.168.56.101's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

218 packages can be updated.
101 updates are security updates.

Last login: Wed May 13 02:40:39 2015 from 192.168.56.1
mininet@mininet-vm:~$ cd new_folder
mininet@mininet-vm:~/new_folder$ pwd
/home/mininet/new_folder
mininet@mininet-vm:~/new_folder$ ls
asdf.txt          classified.txt    custom_network.py  master.py  slave.py
basic_server.py   client.py        hclient.py         server.py
mininet@mininet-vm:~/new_folder$
```

Fig. 1. Save all files in new_folder.

- In the other session, navigate to the `/pox` folder from the section 2.3. POX controller will be initiated in this terminal.

```
mininet@mininet-vm:~/pox$ pwd
/home/mininet/pox
```

Fig. 2. POX Directory.

- Within the terminal in `/pox` directory, run below command –

```
./pox.py --verbose forwarding.l2_learning
```

This starts the POX controller as a basic Layer 2 switch. The 'l2_learning' POX controller component is used during this mode to allow the underlying switches to learn basic MAC address and to provide Layer 2 forwarding capabilities.

- In the network topology terminal, within `home/mininet/new_folder` directory, run the below command –

```
sudo python custom_network.py
```

This will build the network topology as provided below and start all the network nodes. The switches, s1 and s2, connect with the POX controller currently running in the other terminal and allow Layer 2 forwarding.

- If the network topology is initiated successfully, below output is generated on the topology terminal and the POX terminal simultaneously.

```
mininet@mininet-vm:~/new_folder$ sudo python custom_network.py
***Adding Controller
***Adding Switches
***Adding Hosts
***Creating Links
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
***Starting network
*** Starting controller
c0
*** Starting 2 switches
s1 s2
***Entering command prompt
*** Starting CLI:
mininet>
```

Fig. 3. Network Topology Terminal

```
mininet@mininet-vm:~/pox$ pwd
/home/mininet/pox
mininet@mininet-vm:~/pox$ ./pox.py --verbose forwarding.l2_learning
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.1.0 (betta) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.1.0 (betta) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
```

Fig. 4. POX Controller Terminal

- Check the connectivity of all the nodes by using the *pingall* command in the topology terminal. Ensure no packets are being dropped.
- DDoS attack can be initiated by accessing all the hosts through their console. To achieve this, run below command on the topology terminal.

xterm h1 h2 h3 h4 h5 h6 h7

This will open 7 Xterm terminals that provide access to each of the 7 hosts mentioned in the command. This is shown in the figure below.

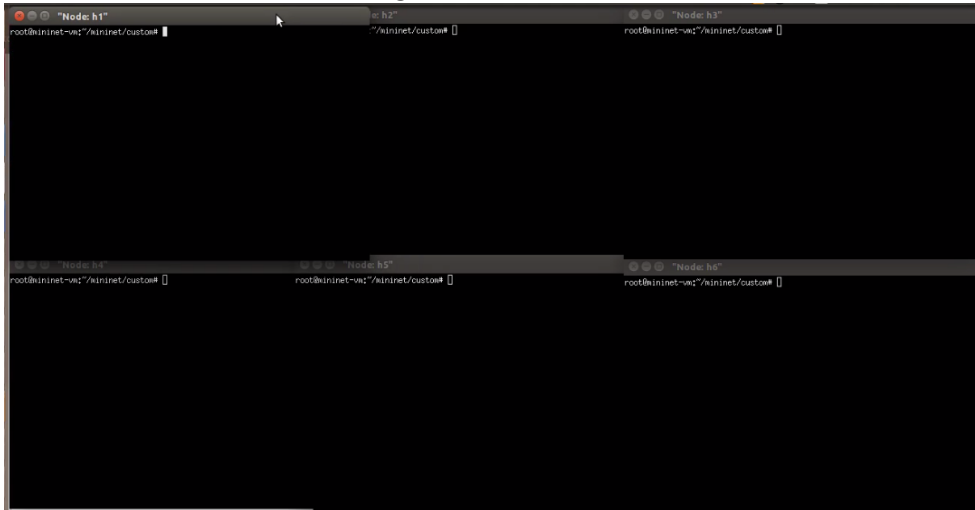


Fig. 5. XTerm Session for Hosts 'h1' to 'h7'.

- In this configuration, 'h6' is the designated HTTP server, 'h1' is the bot master, 'h2-h5' are the botnet hosts and 'h7' is the legitimate client.
- Start the HTTP server by executing the below command in the 'h6' terminal.

python basic_server.py

This initiates the HTTP server.

• **TEST CASE 1:**

To validate the server-client action, request an HTTP page from the Host 'h7' using the below command-

python client.py

Upon success, Host 'h7' request 'asdf.txt' file from the server and below output is generated.

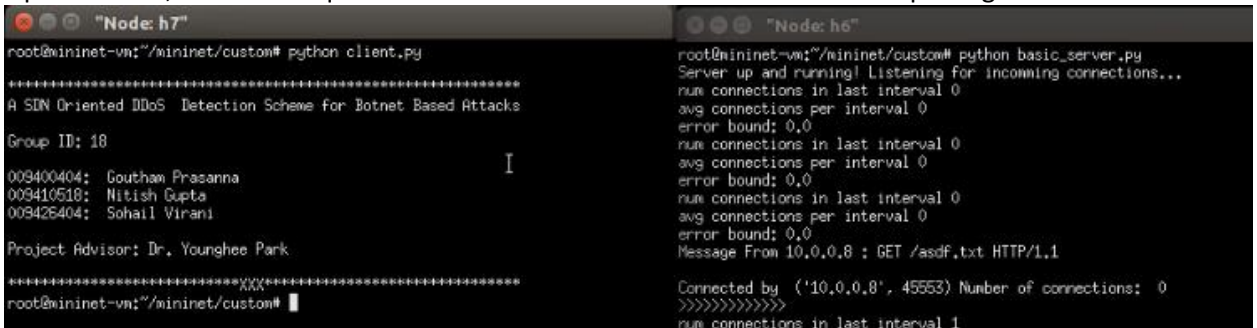


Fig. 6. HTTP Server-Client Interaction

- Once the server-client action is validated, DDoS attack can now be tested. To initiate this attack a series of commands must be initiated in hosts 'h1' to 'h5'.
- Start the bot master by running below command in Host 'h1'-

python master.py

This starts the bot master application on the Host 'h1'.

- Following this, run the below command in Hosts 'h2' to 'h7 in sequence.

python client.py

After executing above command in each host, the corresponding host connects to the bot master. This is verified by recording the host entry in the bot master 'h1' that is displayed with the current timestamp.

- Once the above command is executed in Hosts 'h2' –'h5', the bot master 'h1' automatically triggers the botnets to initiate the HTTP GET FLOOD attack. This can be observed in the botnet hosts by the displayed message- "DDoS Attack Activated! ", as shown below.

```

Node: h1
root@mininet-vn:/mininet/custom# python master.py
Listening for connections
Accepting connection ('10.0.0.3', 53821)
Slave 1 connected at: Mon Apr 27 16:54:42 2015
Accepting connection ('10.0.0.4', 51194)
Slave 2 connected at: Mon Apr 27 16:54:49 2015
Accepting connection ('10.0.0.5', 51959)
Slave 3 connected at: Mon Apr 27 16:54:57 2015
Accepting connection ('10.0.0.6', 45897)
Slave 4 connected at: Mon Apr 27 16:55:10 2015
All Slaves ready to ATTACK!!!
root@mininet-vn:/mininet/custom#

Node: h2
root@mininet-vn:/mininet/custom# python slave.py
DDoS mode loaded
Connected to Master at: Mon Apr 27 16:54:37 2015
ATTACK 10.0.0.7 8080 1430178911.03
I[DDoS Attack Activated.....!!!!] I
I[DDoS Attack Activated.....!!!!] I
I[DDoS Attack Activated.....!!!!] I
I[DDoS Attack Activated.....!!!!] I
I[DDoS Attack Activated.....!!!!] I
I[DDoS Attack Activated.....!!!!] I
I[DDoS Attack Activated.....!!!!] I
I[DDoS Attack Activated.....!!!!] I
I[DDoS Attack Activated.....!!!!] I
I[DDoS Attack Activated.....!!!!] I

```

Fig. 7. DDoS Attack Activated.

- The HTTP server is designed to monitor the HTTP requests received within a particular time interval (set to 2.0 seconds for this project). Upon HTTP GET FLOOD attack, the requests at the server increase drastically during this duration. The server is designed to trigger a "DDoS DETECTED" and "ALERT" message as shown in the figure below.

```

Connected by ('10.0.0.5', 50797) Number of connections: 150
>>>>>>>>>>>>
rpm connections in last interval 85
avg connections per interval 4
DDOS DETECTED! ERROR: 1
error bound: 0.740810367098
num connections in last interval 0
avg connections per interval 3
ALERT!!! ALERT!!! SERVER UNDER ATTACK...
SERVER SHUTTING DOWN...
root@mininet-vn:/mininet/custom#

```

Fig. 8. DDoS Detected Mode and ALERT Message.

- Due to the tremendous load on the server, it shuts down and the application is terminated.

- Within the network topology terminal running in the background, terminate all the xterm sessions running for the hosts 'h1' to 'h7' by running 'exit' command. This stops the Mininet instance that is running the network nodes and the switches get disconnected from the POX controller.

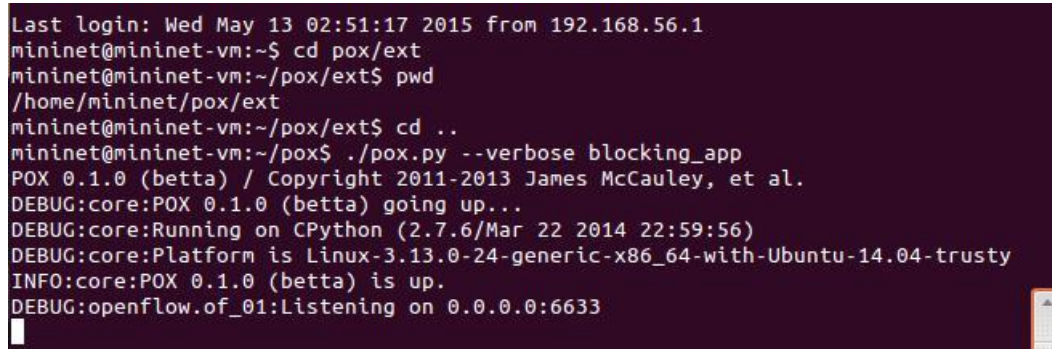
3.2. DEFENSE Mode

In this mode, the detection and mitigation techniques for the DDoS attack is demonstrated.

- In the POX terminal, for the defense mode, we initiate the DDoS Blocking Application using **blocking_app.py** component. To begin POX controller, run the following command in *home/mininet/pox/* folder-

```
./pox.py --verbose blocking_app
```

This component is a combination of Layer 2 learning switch and DDoS Blocking application.



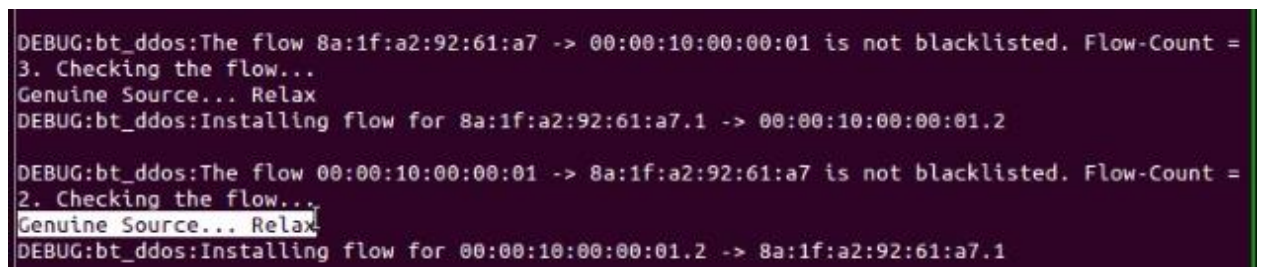
```
Last login: Wed May 13 02:51:17 2015 from 192.168.56.1
mininet@mininet-vm:~$ cd pox/ext
mininet@mininet-vm:~/pox/ext$ pwd
/home/mininet/pox/ext
mininet@mininet-vm:~/pox/ext$ cd ..
mininet@mininet-vm:~/pox$ ./pox.py --verbose blocking_app
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.1.0 (betta) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.1.0 (betta) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
```

Fig. 9. Initiating the Blocking_app component on POX Controller.

- In the network topology terminal, rebuild the network topology using below commands and check for connectivity between the hosts.

```
sudo python custom_network.py
pingall
```

The flows between different hosts are installed within the POX controller and its **blocking_app** component keeps a track of all these flows.



```
DEBUG:bt_ddos:The flow 8a:1f:a2:92:61:a7 -> 00:00:10:00:00:01 is not blacklisted. Flow-Count = 3. Checking the flow...
Genuine Source... Relax
DEBUG:bt_ddos:Installing flow for 8a:1f:a2:92:61:a7.1 -> 00:00:10:00:00:01.2

DEBUG:bt_ddos:The flow 00:00:10:00:00:01 -> 8a:1f:a2:92:61:a7 is not blacklisted. Flow-Count = 2. Checking the flow...
Genuine Source... Relax
DEBUG:bt_ddos:Installing flow for 00:00:10:00:00:01.2 -> 8a:1f:a2:92:61:a7.1
```

Fig. 10. Blocking_app component of POX controller monitoring the flows within the SDN.

- Rerun the xterm terminal to access the console for all the hosts within the network.
xterm h1 h2 h3 h4 h5 h6 h7

- In the defense mode, we initiate the modified HTTP server with CAPTHCHA and HONEYTOKEN features enabled. To start this server, within the xterm terminal for Host 'h6' run following command-
python server.py
- **TEST CASE 2:**
Once the server is running, verify the initial server-client interaction using following command in the xterm terminal for Host 'h7'-
python client.py

```

"Node: h7"
root@mininet-vni:/mininet/custom# python client.py
*****
A SDN Oriented DDoS Detection Scheme for Botnet Based Attacks
*****
Group ID: 18
009400404: Goutham Prasanna
009410518: Nitish Gupta
009426404: Sohail Virani
Project Advisor: Dr. Youngee Park
*****
root@mininet-vni:/mininet/custom#

"Node: h6"
root@mininet-vni:/mininet/custom# python server.py
Server up and running! Listening for incoming connections...
num connections in last interval 0
avg connections per interval 0

num connections in last interval 0
avg connections per interval 0

num connections in last interval 0
avg connections per interval 0

Message From 10.0.0.8 : GET /asdf.txt HTTP/1.1

Connected by ('10.0.0.8', 45735) Number of connections: 1
>>>>>>>>>>
num connections in last interval 1
avg connections per interval 0

```

Fig. 11. Test Case 2 Result.

- **TEST CASE 3:**
In this test, the HONEYTOKEN feature at the server end is tested. The clients 'h2' to 'h7' can access any file within the server. To access another file run the below command on Host 'h7'-

python h_client.py

Upon running this application, it now requests for the honey-token page, 'classified.txt', instead of the legitimate 'asdf.txt' page. Even though the client can access this Honey-token file from the server, the server blacklists the client as seen below.

```

"Node: h7"
root@mininet-vni:/new_folder# python hclient.py
*****CLASSIFIED*****
*****
EMPLOYEE SOCIAL SECURITY AND FINANCIAL RECORDS
*****
WHITE CASTLE Inc.
San Jose, CA,95113
USA

SOCIAL SECURITY #      NAME      CREDIT CARD      ISSUED BY
517-07-1774      Tanika Bradshaw      455605360755056      VISA
422-45-9214      Jerry Bass      4533857512317491      VISA
470-16-9134      Marti Pearson      5556307961573058      MASTERCARD
239-36-8132      Maxine Khan      370601901275716      AMERICAN EXPRESS
454-73-0153      Sylvie North      4024007153721736      VISA
041-14-0139      Reulene Hoperson      601103914948804      DISCOVER
518-17-5208      Marybeth Moarthur      5548410742755134      MASTERCARD
673-32-3419      Kristen Bell      4529474288996405      VISA
304-78-9689      Jessica Deacon      4539291190747516      VISA
222-58-2379      Michael Jordan      4539187318199842      VISA

NOTE: ONLY FOR AUTHORIZED USERS

*****END OF CLASSIFIED DOCUMENT*****
root@mininet-vni:/new_folder#
root@mininet-vni:/new_folder#

```

Fig. 12. Honey-token file accessed by the Client 'h7'.

Host 'h1':
`python master.py`

Host 'h2' to 'h5':
`python slave.py`

Upon successful execution, the bot master records each botnet host and triggers the DDoS attack.

- Observe that the DDoS attack is interrupted and 'Connection Failed' message is received in Hosts 'h2' to 'h5'. This is seen in the figure below.

```
[DDoS Attack Activated....!!!!] |
[DDoS Attack Activated....!!!!] |
[DDoS Attack Activated....!!!!] |
[DDoS Attack Activated....!!!!] |
[Connection Failed] | 1
[Connection Failed] | 2
[Connection Failed] | 3
[Connection Failed] | 4
[Connection Failed] | 5
[Connection Failed] | 6
[Connection Failed] | 7
```

Fig. 15. DDoS Attack blocked by the POX Controller.

The '**blocking_app**' component running on the POX controller interrupts the DDoS attack and blacklists all the botnet hosts from accessing the HTTP server. This can be seen in the POX controller terminal by the "DELETING" flows and the "DROPPING" packets from these flows as seen below.

```
DEBUG:bt_ddos:The flow 32:c9:67:c4:19:b0 -> 00:00:10:00:00:01 is not blacklisted. Flow-Count =
39. Checking the flow...
Warning. The High inflow of packets from 32:c9:67:c4:19:b0 -> 00:00:10:00:00:01
DEBUG:bt_ddos:Installing flow for 32:c9:67:c4:19:b0.1 -> 00:00:10:00:00:01.2

DEBUG:bt_ddos:The flow 00:00:10:00:00:01 -> 32:c9:67:c4:19:b0 is not blacklisted. Flow-Count =
40. Checking the flow...
Attention: DDoS attack from 00:00:10:00:00:01 -> 32:c9:67:c4:19:b0
DEBUG:bt_ddos:Deleting flow for 00:00:10:00:00:01.2 -> 32:c9:67:c4:19:b0.1

The flow 00:00:10:00:00:01 -> 2e:40:06:38:0e:89 is blacklisted. Access denied.
DEBUG:bt_ddos:Dropping packets for blacklisted flow- 00:00:10:00:00:01.2 -> 2e:40:06:38:0e:89.1
```

Fig. 16. POX Controller filtering flows by deleting affected flows and dropping packets.

- Simultaneously, the HTTP server records the sudden increase in the HTTP GET requests and enters the 'DDoS Warning' mode. Rate-limiting the flows at the POX controller safely prevents the server from entering the 'DDoS Detected' mode. However, the CAPTCHA mode is enabled at the server and it requests a new CAPTCHA for every new HTTP GET request.

- **TEST CASE 5:**

In this test, the CAPTCHA mode in the server is tested. From the host 'h7', run the following command-

`python client.py`

The client now will be prompted to provide the CAPTCHA before it can access the requested file. In this scenario, the client provides the wrong CAPTCHA. The server denies the client page request by replying with an "AUTHENTICATION FAILED" message.

- **TEST CASE 6:**

In this test, rerun the same command from above test case. During this test, the client must provide the exact same CAPTCHA as prompted by the server. The server now authenticates the client and provides the requested page. This is shown below.

- **TEST CASE 7:**

In this test, the botnet connectivity towards the HTTP server is tested. After the POX controller blacklists the flows from the botnets towards the HTTP server, they will not be able to PING the server ('h6', '10.0.0.7') as seen below.

```
root@mininet-vx:~/mininet/custom# ping 10.0.0.7
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data:
From 10.0.0.3 icmp_seq=1 Destination Host Unreachable
From 10.0.0.3 icmp_seq=2 Destination Host Unreachable
From 10.0.0.3 icmp_seq=3 Destination Host Unreachable
From 10.0.0.3 icmp_seq=4 Destination Host Unreachable
From 10.0.0.3 icmp_seq=5 Destination Host Unreachable
From 10.0.0.3 icmp_seq=6 Destination Host Unreachable
^C
--- 10.0.0.7 ping statistics ---
7 packets transmitted, 0 received, +6 errors, 100% packet loss, time 6017ns
pipe 3
root@mininet-vx:~/mininet/custom#
```

Fig. 17. Failed PING requests from Botnet host 'h2' towards HTTP Server 'h6'.

However, the botnets will still be able to access all the other nodes present within the network. This is shown in the figure below where the PING command from the host 'h2' (10.0.0.3) towards host 'h3' (10.0.0.4) is successful.

```
root@mininet-vx:~/mininet/custom# ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data:
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=42.9 ns
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.348 ns
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.451 ns
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.035 ns
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.021 ns
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=0.047 ns
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=0.398 ns
^C
--- 10.0.0.4 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6004ns
rtt min/avg/max/mdev = 0.021/6.322/42.959/14.957 ns
root@mininet-vx:~/mininet/custom#
```

Fig. 18. Successful PING requests from Botnet Hosts 'h2' and 'h3'.