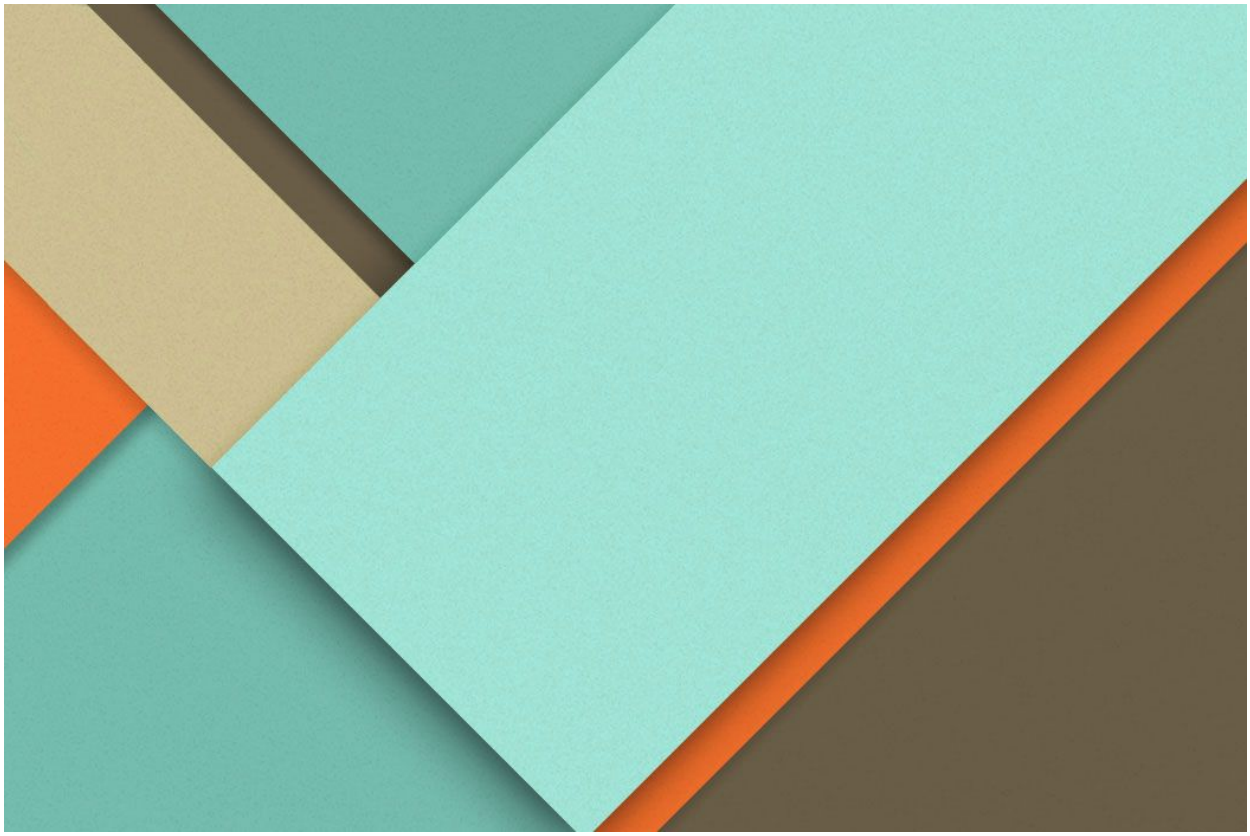

FINAL REPORT

CSE 406 : Computer Security Sessional

Attack Tools Implementation : Ping Flood Attack

Md. Mohaiminul Islam
Sec - B, Group - **03**
St ID : **1505078**
CSE, BUET

Overview

In the design proposal three attack strategies were proposed

- Regular uncontrolled ping flood
- Reflected attack using spoofed source addresses
- Amplification attack using broadcast

In this project we gradually implemented these attack strategies each of which improves on the weakness/fault of the previous strategy.

Implementation Tools

To implement this attack we developed our own version of the ping application which is by default present in most OS. This tool consists of a single **python** script named **ping.py** which takes a target IP address as command line input . After that the programme constructs an ICMP 'echo' request packet with the target address as destination ,some garbage payload and continuously pings the target IP address without waiting for a reply.

To verify and visualize the attack effects other third party applications such as- Wireshark, Iptraf & tcpdump were used. We also used virtnet and Oracle vBox to create a virtual network.

Phase One : Tool Validation and Uncontrolled Ping Flood(Demo)

To see if our tool works properly we use two cloned images of the seed ubuntu. We run the **ping.py** script from one image using the IP address(192.168.0.107) of the other.

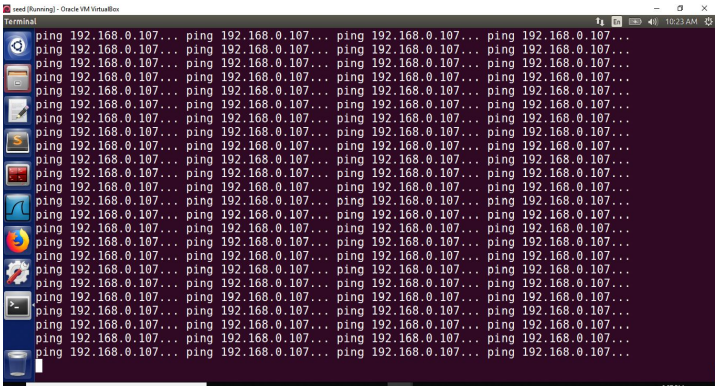

Fig1: Attacker screen after running ping.py

If we run the command in the victim machine:

$sudo tcpdump -n -t -i enp0s3 'icmp'

We can listen to all incoming and outgoing ICMP traffic at the network interface enp0s3(192.168.0.107).

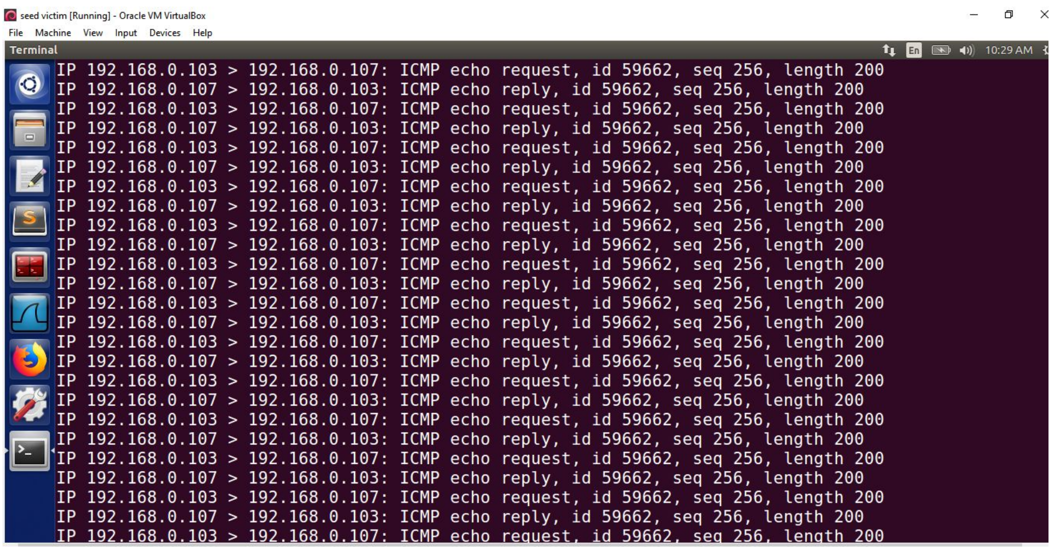

Fig2: Listening to incoming ICMP traffic at victim's network interface

After that we use wireshark to verify the packet contents on victim machine.

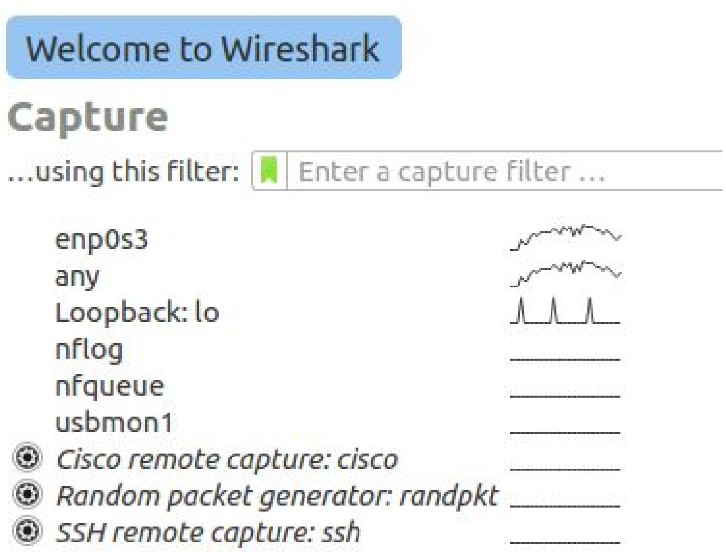

Fig3: As expected wireshark shows abnormal traffic at the enp0s3 network adapter.

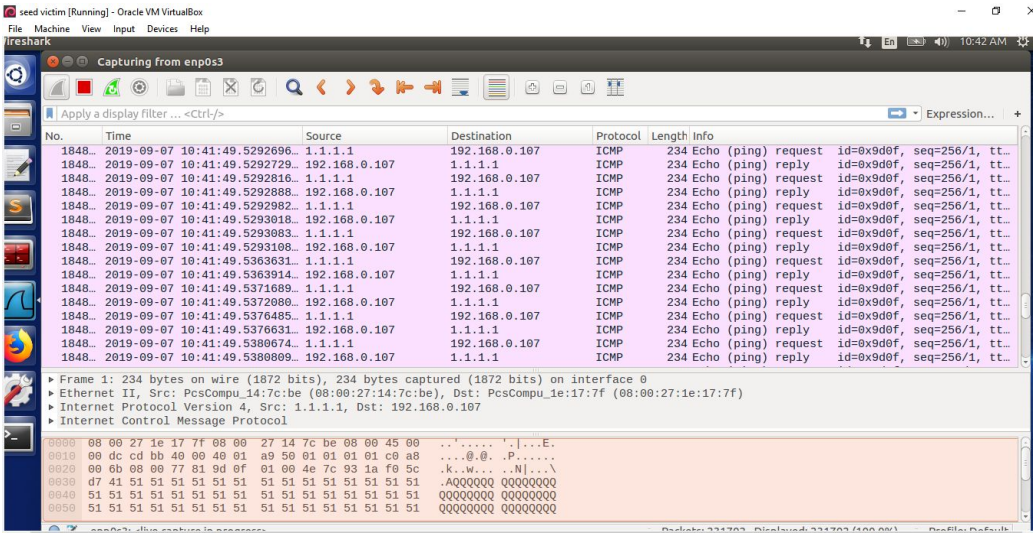

Fig4 : viewing packet contents via wireshark.

Phase Two: Experimenting on a virtual network

To test the potency of our attack we create a virtual network in Oracle virtualbox and test our tool.

Local Environment Setup :

To create this network we first created a **base** virtual machine with minimal functionalities and networking to minimize RAM consumption. After that, we clone this virtual image to create 8 different virtual machines which will act as 8 nodes in our network.

Topology:

This is the topology of our virtual network-

It has 8 nodes of which 2 are configured to be routers(nodes 2 &7) , thus creating three different subnets.

Subnet a = nodes 1,2 (address 192.168.1.0)

Subnet b = nodes 2,3,4,5,6,7 (address 192.168.2.0)

Subnet c= nodes 7,8 (address 192.168.3.0)

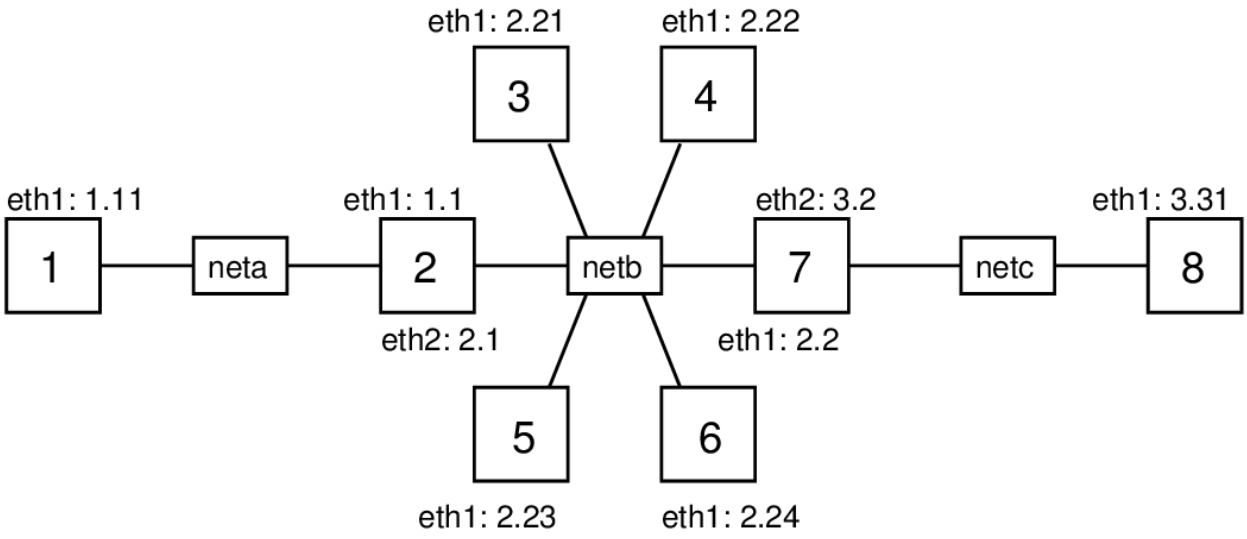

Fig5 : Virtual Network topology

To initialize this we run a topology26.cmd script from available topologies in virtnet repository.

Setup Nodes and Links :

A ping flooding DoS attack aims to overflow a link leading to the target, so data from normal users to the target will be dropped or significantly delayed. Overflowing a link means sending enough data across the link such that the full link capacity is utilized. In real-life, the link from the target network to their ISP, or a link within the target network, is that which is under attack. If the link has a high capacity (multiple Gb/s), then to utilize the capacity, thousands of computers must be sending data at a very high rate towards the target. Slaves, reflectors and amplification are usually required.

In our virtual network we only have several nodes. Therefore to overflow the link to the target (from router node 7 to target node 8), we need the capacity of that link to be quite low. VirtualBox emulates the network links, but by default doesn't set a link capacity or date rate. The speed at which two virtual nodes can exchange data across a link varies, and depends on factors such as host CPU, disk and the driver used for the virtual network interfaces. Therefore we need to explicitly set the link capacity, at least between node 7 and node 8. To do so, we will use a Linux traffic control program, **tc** . We can use **tc** to emulate link characteristics like data rate (capacity), delay, jitter and packet drops.

tc operates on the outgoing link, and therefore to set the capacity of the link in both directions between node 7 and 8, we need to apply the commands on both nodes. For the demo we will set the capacity to 100,000 bits per second (100 kb/s). This is low enough such that we can easily overflow with a few nodes sending pings. First node 7:

---

network@node7:~$ ***sudo tc qdisc add dev eth2 root handle 1:0 htb default 10***

network@node7:~$ ***sudo tc class add dev eth2 parent 1:0 classid 1:10 htb rate 100000***

And now node 8 (the only difference between the commands on the nodes is the interface, **eth2** on node 7 and **eth1** on node 8):

---

network@node7:~$ ***sudo tc qdisc add dev eth1 root handle 1:0 htb default 10***

network@node7:~$ ***sudo tc class add dev eth1 parent 1:0 classid 1:10 htb rate 100000***

Turn Off Security Features in the Linux Kernel :

The Linux kernel includes features to prevent (or at least make very difficult) ping flooding attacks. Therefore to see the attack in action, we needed to disable these security features.

1. when acting as a router, the Linux kernel does not allow packets originating from one of its subnets, but with a fake source address, to be forwarded to another subnet. The feature is called **Reverse Path Filtering** . We needed to disable this feature on the routers in the network (nodes 2 and 7). We did this by turning off the rp_filter kernel parameter for both interfaces, eth1 and eth2, on each router.

---

network@node2:~$ ***sudo sysctl net.ipv4.conf.eth1.rp_filter=0***

net.ipv4.conf.eth1.rp_filter = 0

network@node2:~$ ***sudo sysctl net.ipv4.conf.eth2.rp_filter=0***

net.ipv4.conf.eth2.rp_filter = 0

network@node2:~$ ***sudo /etc/init.d/networking restart***

Similar approach for node 7.

Next, in some attacks, We want one node to ping the broadcast address, so the ping is sent to all nodes in the subnet. However the Linux kernel is configured to ignore ping

broadcasts (i.e. not reply to Echo requests to the broadcast address). We need to accept these ping messages, at least on the nodes in the same subnet as the source. For the demo of ping broadcast, we used node 3 to broadcast to the subnet, including nodes 4, 5 and 6. Therefore on nodes 4, 5 and 6 we turned off the icmp_echo_ignore_broadcasts kernel parameter off .

---

*network@node4:~$**sudo sysctl net.ipv4.icmp_echo_ignore_broadcasts=0***

net.ipv4.icmp_echo_ignore_broadcasts = 0

Similar for nodes 3,5,6.

Phase Three : Reflector Attack Using Spoofed Source Address

Most of the routers of today will detect a malicious node directly trying to flood a link by sending ping packets and drop those packets. The victim can also analyze the packet to find out the attackers IP. So , It is useful to indirectly attack the victim by setting source IP address in the ICMP packet to the victim's address. Then, some non-malicious node accepts the packet they would send the corresponding 'ICMP echo reply' to our target victim. Using this scheme we can remain undetectable and still manage to flood the victim's inbound link.

For demonstration we spoofed the source IP Address of the constructed ICMP echo request packets and set it to the victim's(node 8) IP address(192.168.3.31) . Now set the destination address to some non malicious node such as node4(192.168.2.22). We can see when node4 receives ping it immediately sends the corresponding reply to the node8.

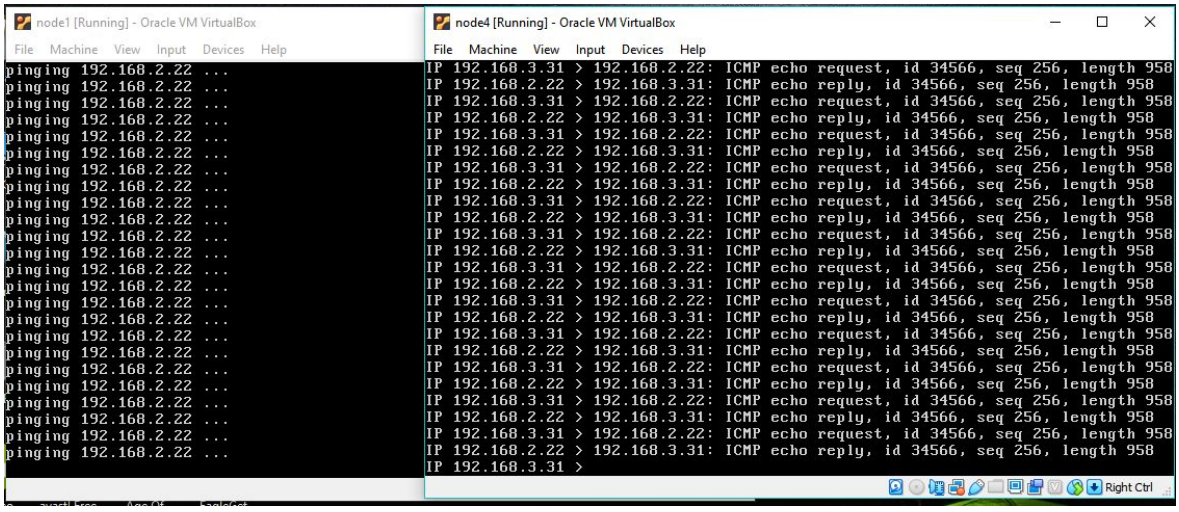

Fig6 : The Attacker(node1) and the innocent node(node4)

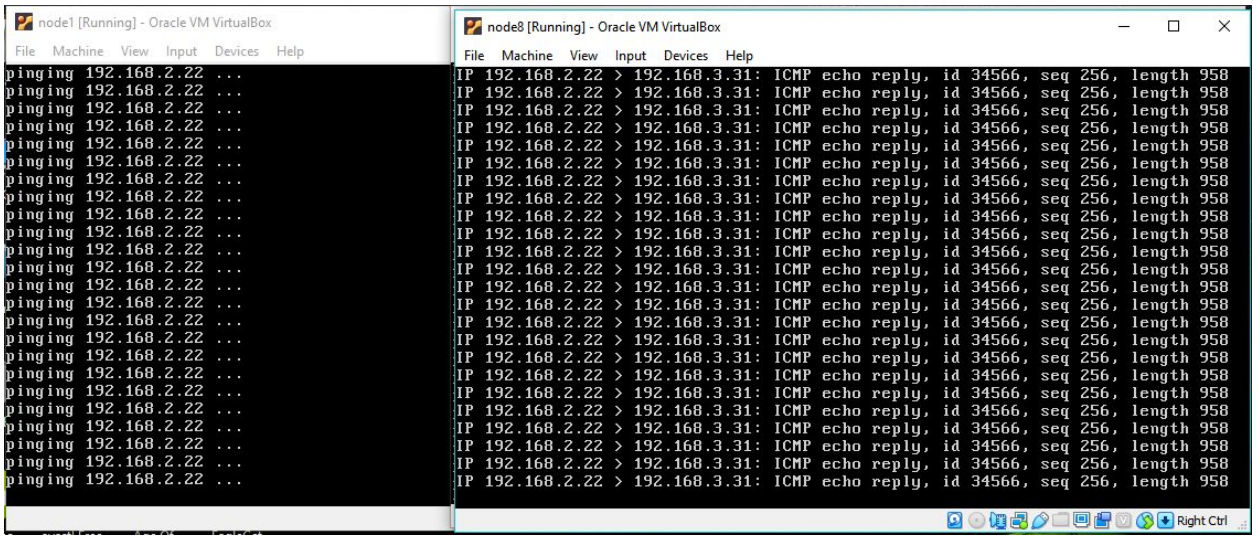

Fig7: The attacker(node1) and the victim(node7) screens

From Fig7 we clearly see that though the attacker(node1) is sending the ping requests the replies are being sent to the victim(node8) from node4.

Again we previously set the link capacity between node7 and node8 to 100Kbps. So If we monitor IP traffic at node7 we will see that it is receiving packets at great speed but can't transmit them to node8 at the same speed

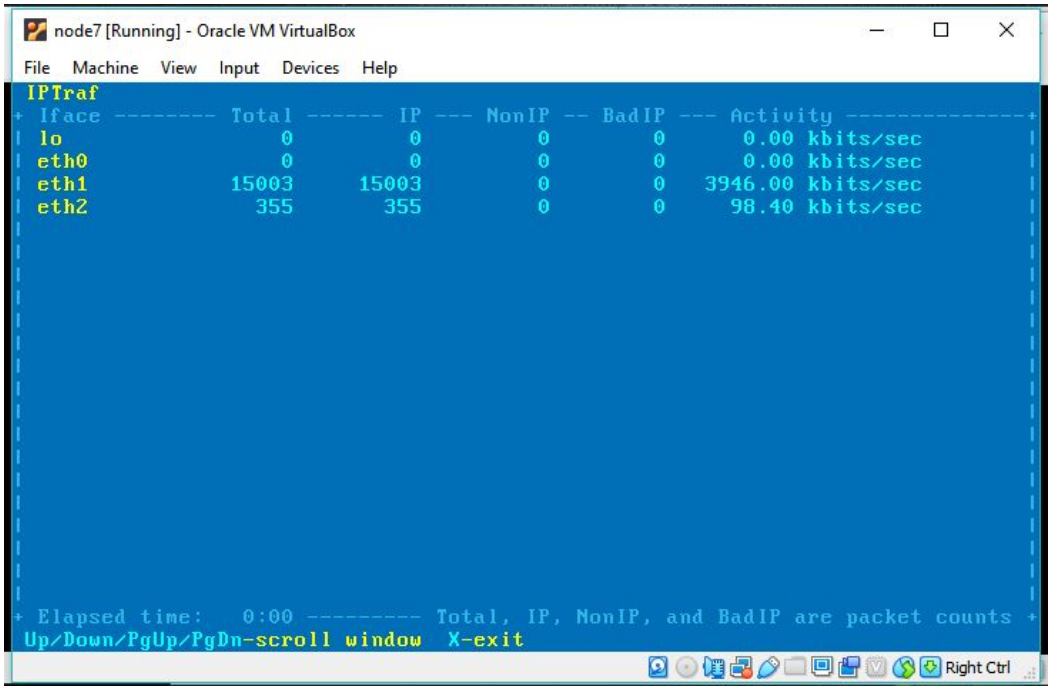

Fig8 : IP Traffic at node7 router

because the link between node7 and node8 (i.e. node 7 's eth2 network interface is linked to node 8's eth1 interface) is at maximum flow.

Node7 is a router and this means incoming packets for node8 are being dropped by the subnet's router. This is the main theme of a DoS attack and thus proves the success of our attack.

Phase Four: Amplification Attack Using Broadcasts

If we want to overflow the targets inbound link without exhausting our own resources a good way is to use amplification attacks via broadcasts. Here we broadcast a ICMP echo request message with victim's address as source address. So all the nodes in the subnet will receive this message and reply together towards victim.

For demonstration we perform broadcast amplification attack from node3(192.168.2.21). This sends ping requests to nodes 4,5,6 (192.168.2.22, 192.168.2.23, 192.168.2.24 respectively) with target's address as spoofed source address. This easily overwhelms the target's inbound link and causes packets to drop at the router.

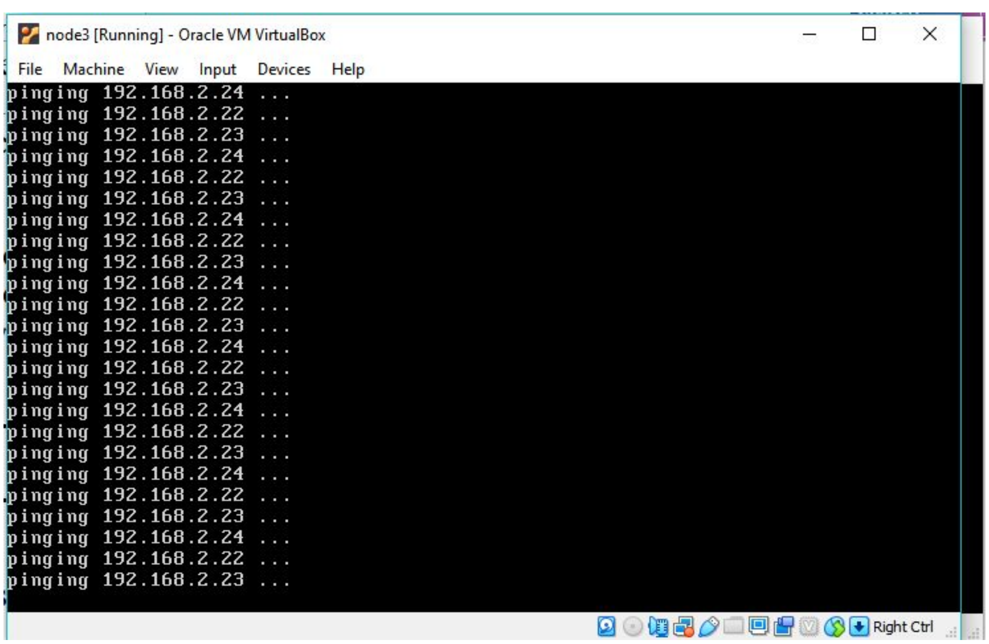

Fig 9 : broadcasting ping messages to the subnet

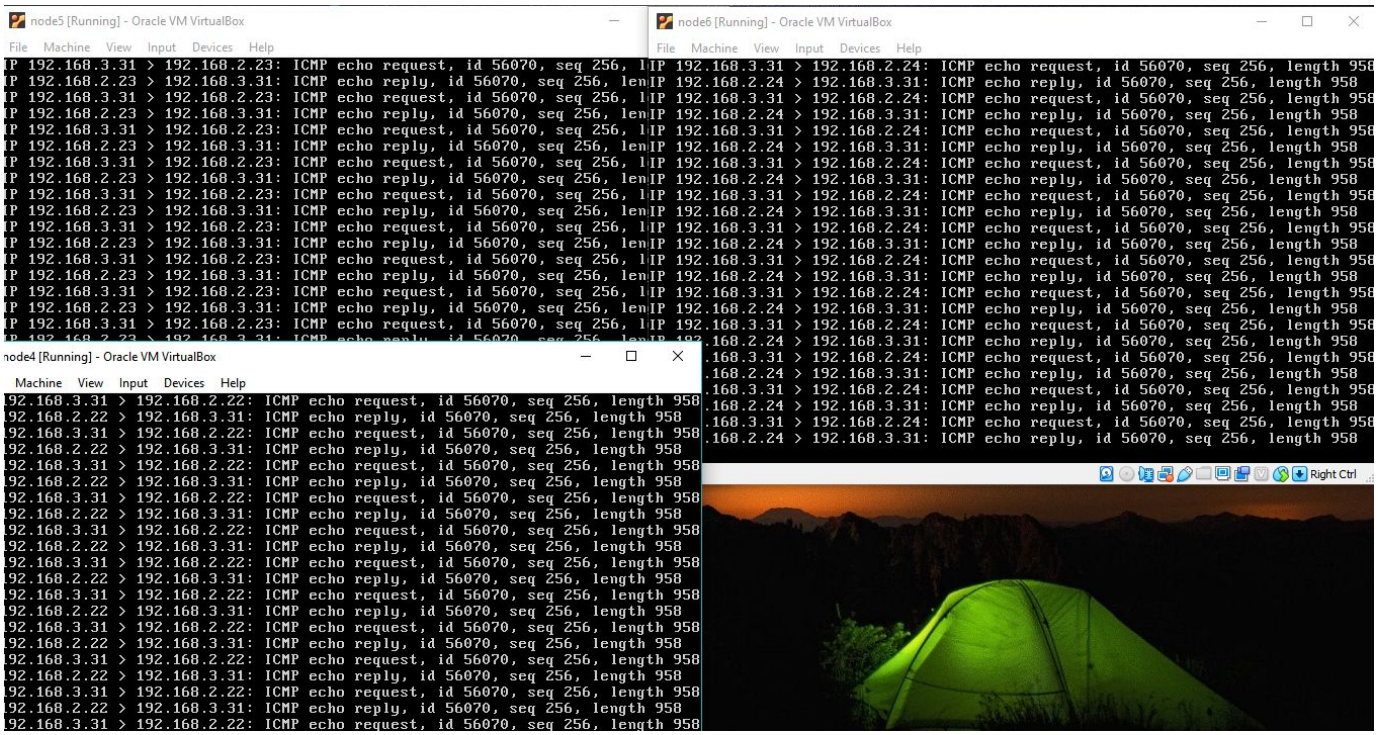

Fig10 : All other nodes in the subnet receive ping and reply to victim(node8)

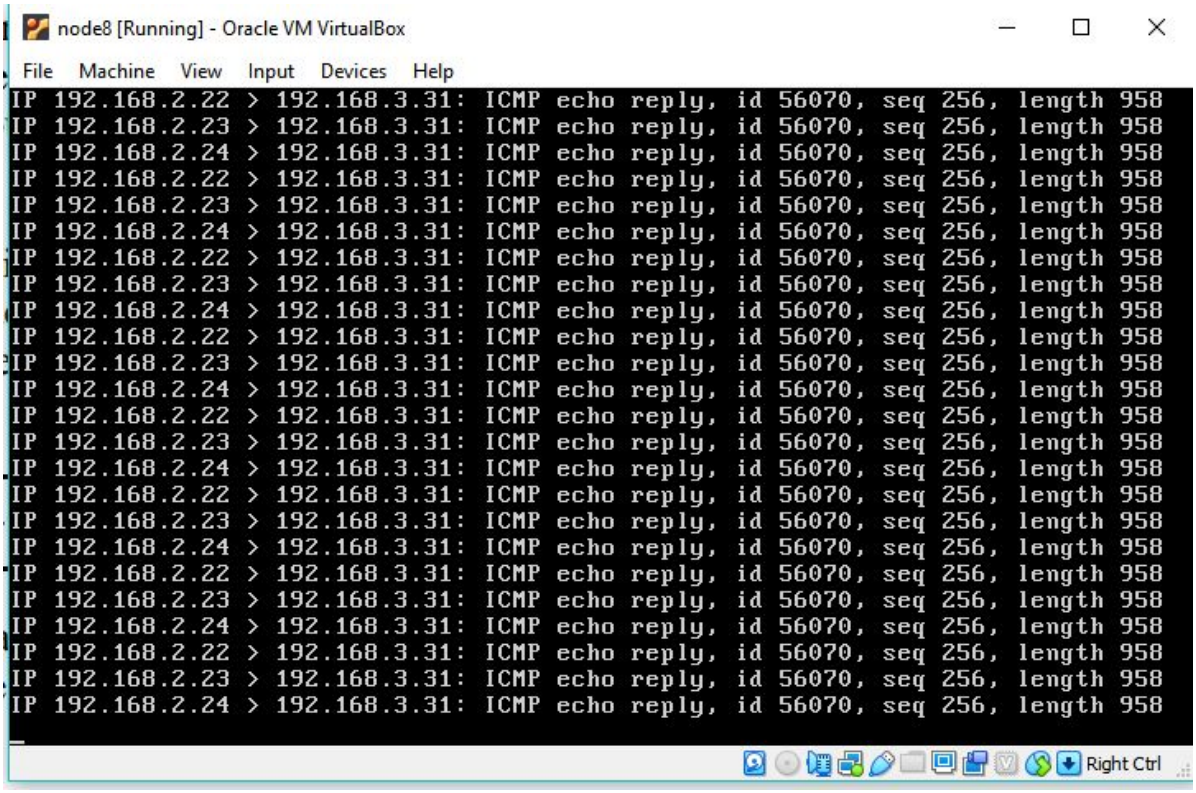

Fig11: Victim(node8) receiving ping replies from all the nodes in the subnet.

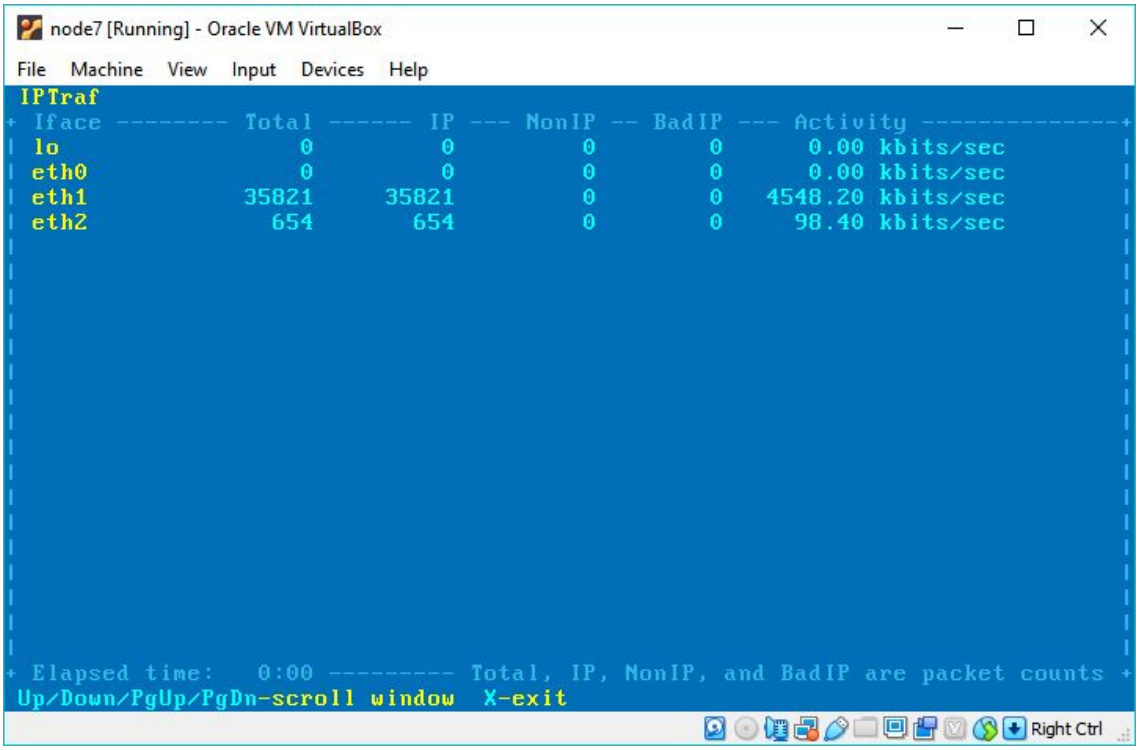

Fig 12 :As expected target's inbound link in overflowing, packets dropped at router

Conclusion

The goal of Denial of Service(DoS) attacks is to block networking services to one or more targets . As we have clearly simulated in the virtual network that our implemented tool overflows some specific target's inbound link restricting their network service . This leads us to believe our experiments were successful and the attack was implemented as proposed.