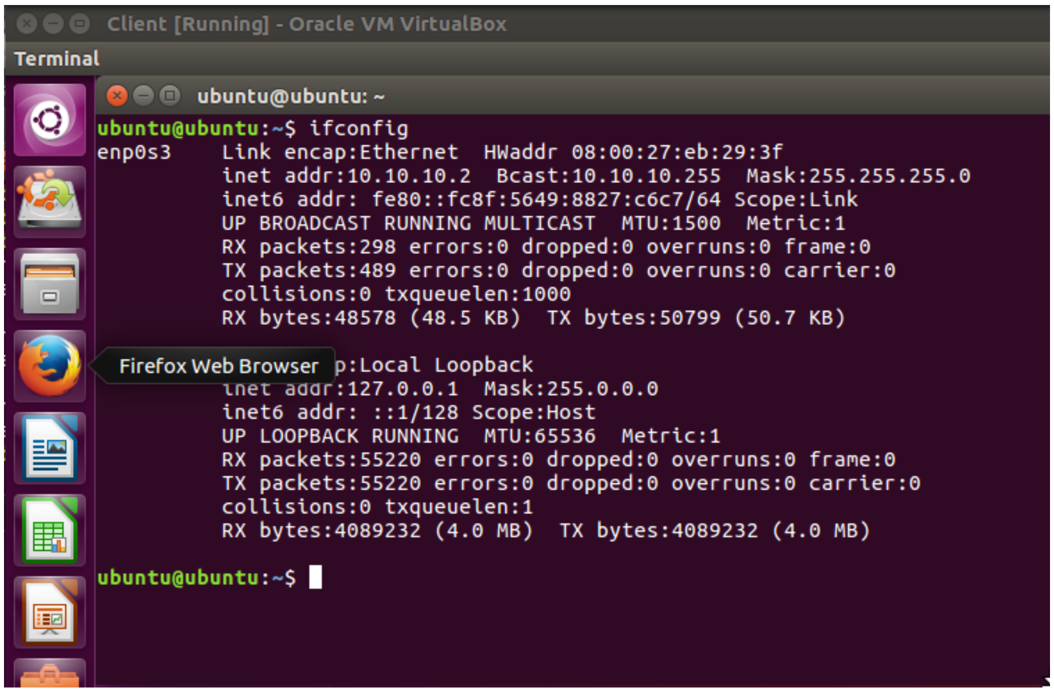
**Name: Vatsal Nagda**
**Roll: 20162008**
**SNS Assignment 5**
**IP Tables**

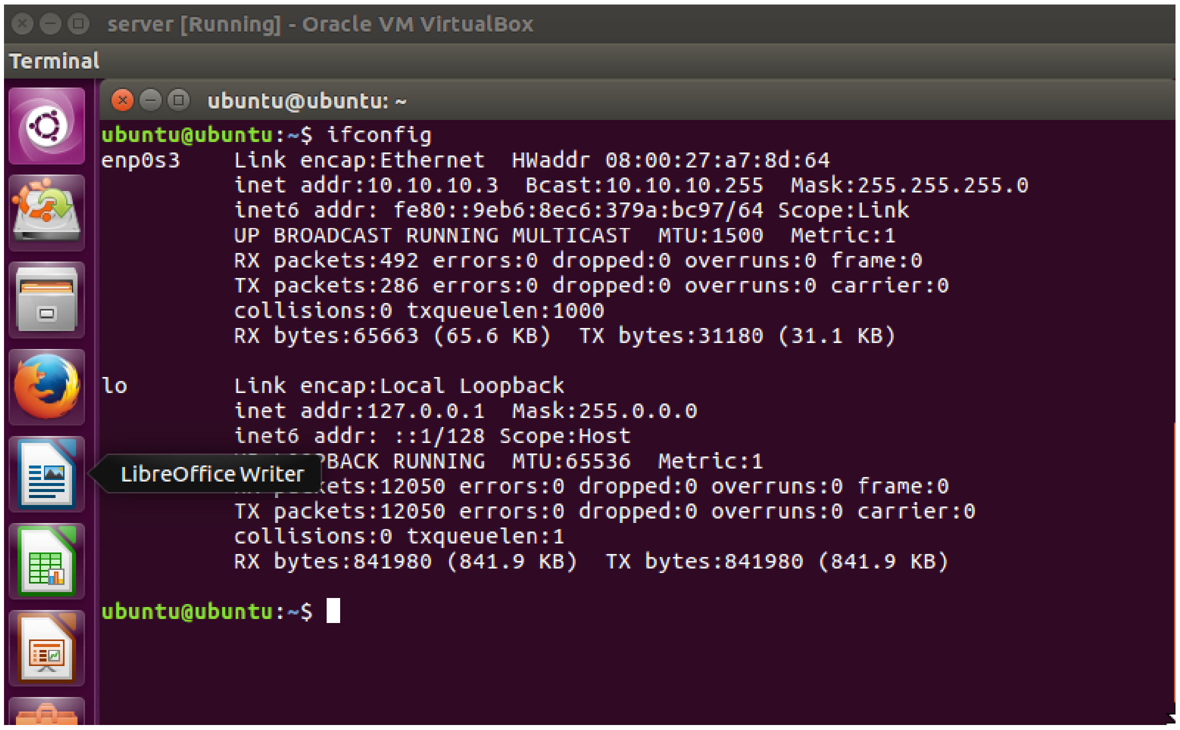
## Configuration:-

IP Address configuration of Client



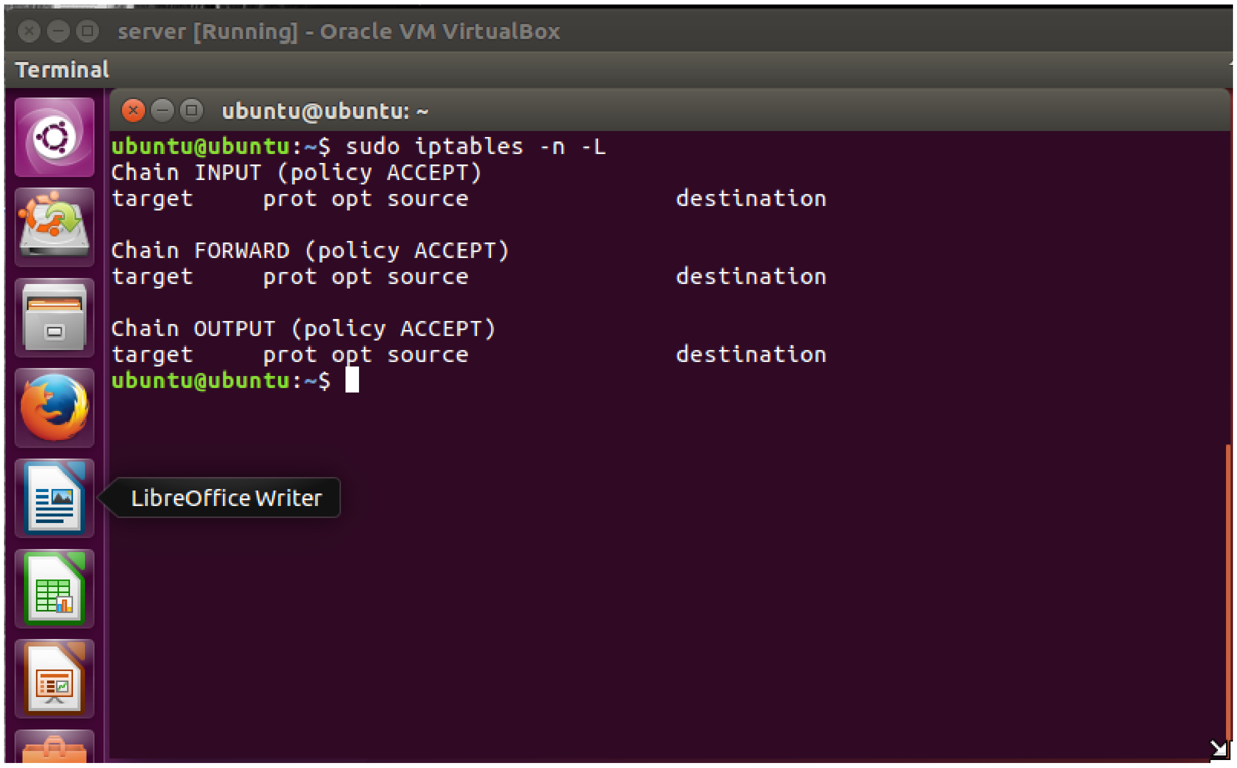
```
ubuntu@ubuntu:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:eb:29:3f
          inet addr:10.10.10.2  Bcast:10.10.10.255  Mask:255.255.255.0
          inet6 addr: fe80::fc8f:5649:8827:c6c7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:298 errors:0 dropped:0 overruns:0 frame:0
          TX packets:489 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:48578 (48.5 KB)  TX bytes:50799 (50.7 KB)

                    p:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:55220 errors:0 dropped:0 overruns:0 frame:0
          TX packets:55220 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:4089232 (4.0 MB)  TX bytes:4089232 (4.0 MB)

ubuntu@ubuntu:~$ 
```

IP Address configuration of server



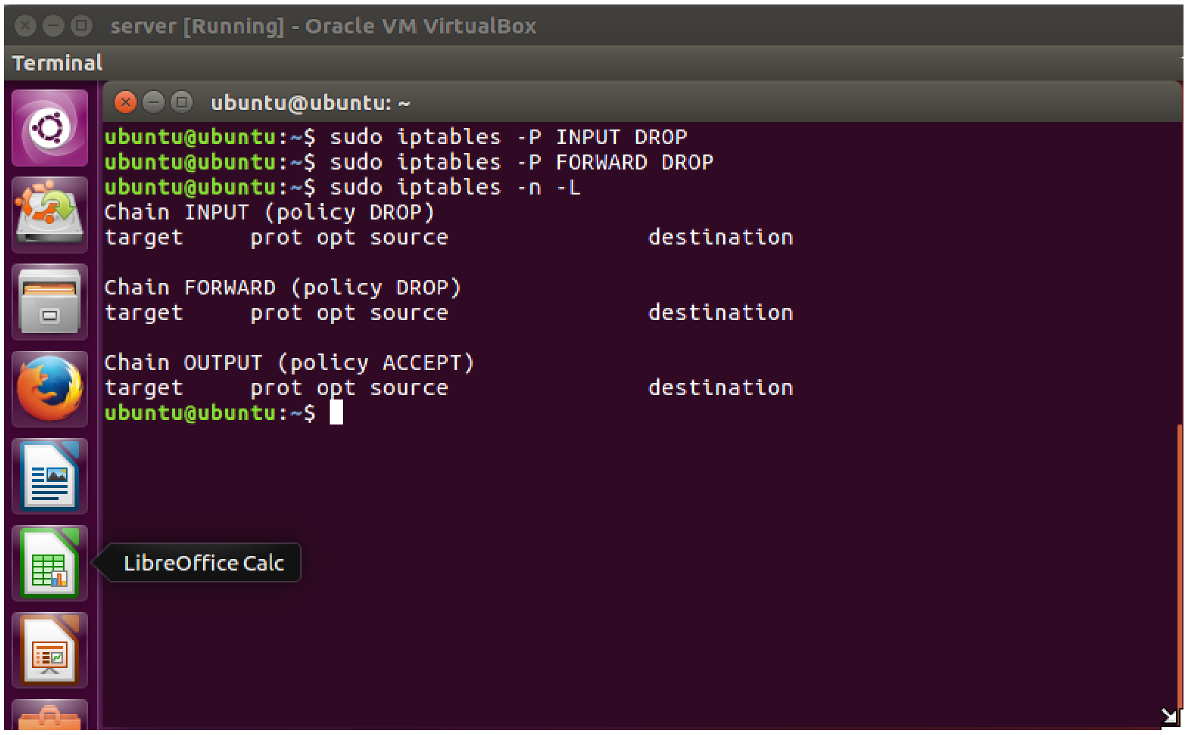
```
ubuntu@ubuntu:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:a7:8d:64
          inet addr:10.10.10.3  Bcast:10.10.10.255  Mask:255.255.255.0
          inet6 addr: fe80::9eb6:8ec6:379a:bc97/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:492 errors:0 dropped:0 overruns:0 frame:0
          TX packets:286 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:65663 (65.6 KB)  TX bytes:31180 (31.1 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          BACK RUNNING  MTU:65536  Metric:1
          ets:12050 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12050 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:841980 (841.9 KB)  TX bytes:841980 (841.9 KB)

ubuntu@ubuntu:~$
```

Default IP Tables of Server



```
ubuntu@ubuntu:~$ sudo iptables -n -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ubuntu@ubuntu:~$
```

## 1. Rules to deny all incoming traffic to the server



```
ubuntu@ubuntu:~$ sudo iptables -P INPUT DROP
ubuntu@ubuntu:~$ sudo iptables -P FORWARD DROP
ubuntu@ubuntu:~$ sudo iptables -n -L
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ubuntu@ubuntu:~$
```

In this, we want restrict all the traffic coming from outside. This can be easily achieved by adding a simple rule to the INPUT chain in iptable. We will formulate this rule as follows :
There are three main chains which have their own default policy.

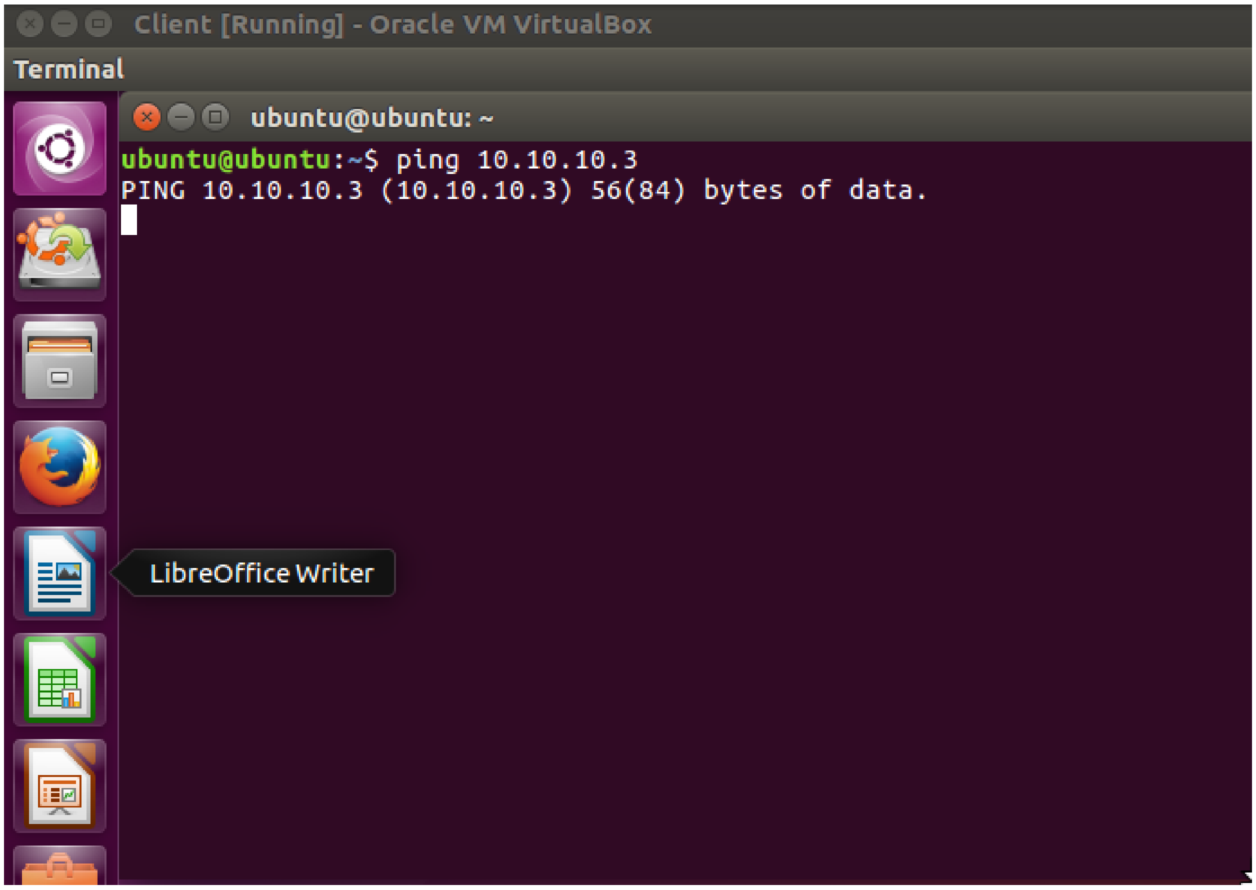
Here we can see that every chain is having default policy as ACCEPT. Now to deny incoming traffic to the server we can change the INPUT chain policy of iptables as follows:

**Sudo iptables -P INPUT DROP**

Now after doing this we should not be able to get any incoming traffic, as FORWARD chain operates way above INPUT chain we still are capable of packet forwarding but we will not be getting any packets into our system. As shown in following screen capture we have changed the INPUT chain policy. Now if we try to ping the server machine from client machine or try to SSH the system we will not be getting any response.

Verification of Incoming Traffic Denial

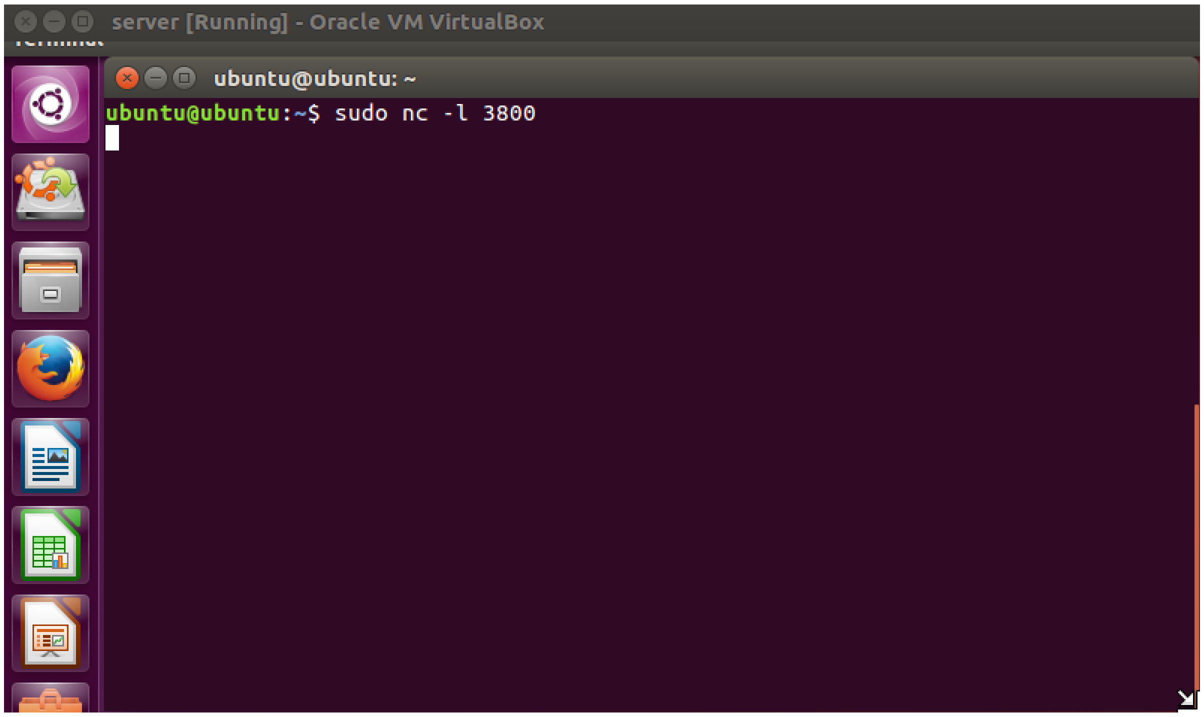
a) Ping Unsuccessful from Client



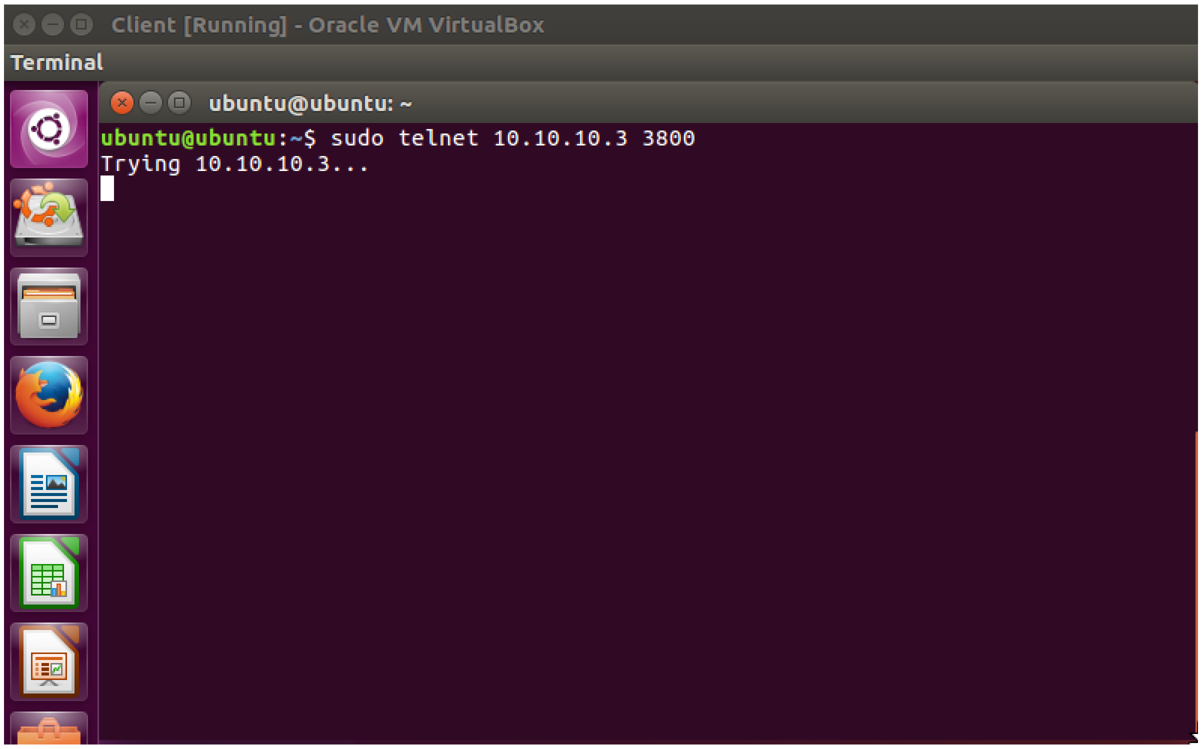
```
ubuntu@ubuntu:~$ ping 10.10.10.3
PING 10.10.10.3 (10.10.10.3) 56(84) bytes of data.
```

## b) Telnet Unsuccessful from client

Server Listening for connections on port 3800



```
ubuntu@ubuntu:~$ sudo nc -l 3800
```



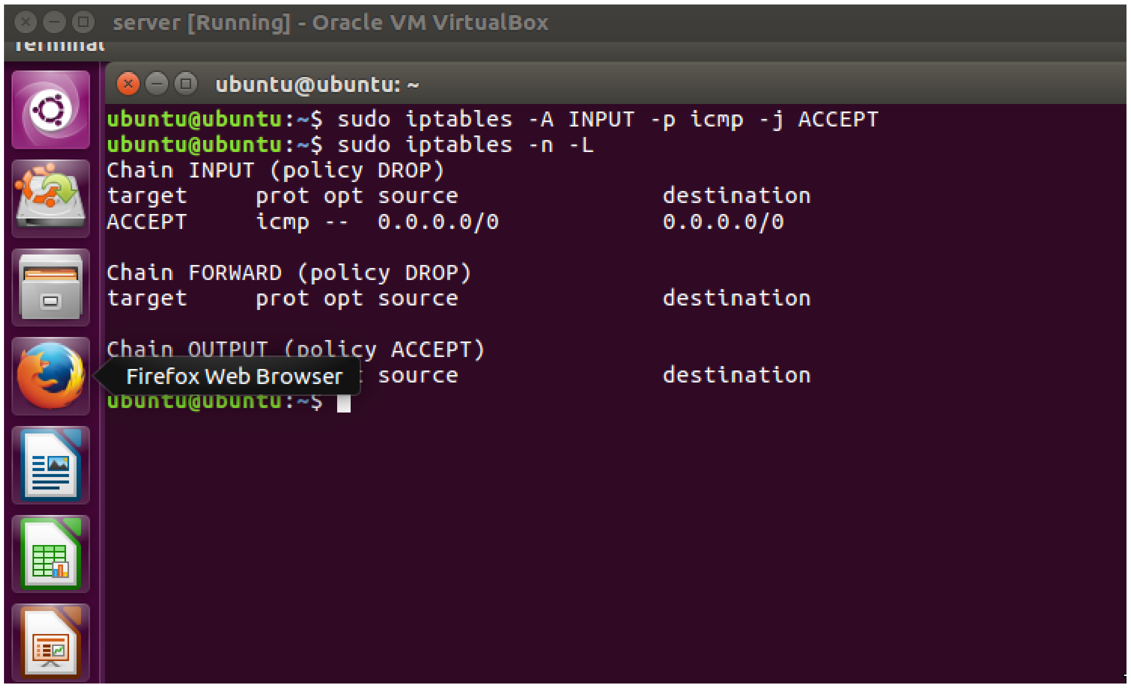
```
ubuntu@ubuntu:~$ sudo telnet 10.10.10.3 3800
Trying 10.10.10.3...
```

## 2. **Create rules to accept only incoming ping and port 23 traffic to the server.**

Now here we want to create a rule by which we will be able to accept all the ping request coming from outside. Along with this we want to accept all the packet which are coming by making use of port number 23. The packets which are coming by other sources , we will drop them.

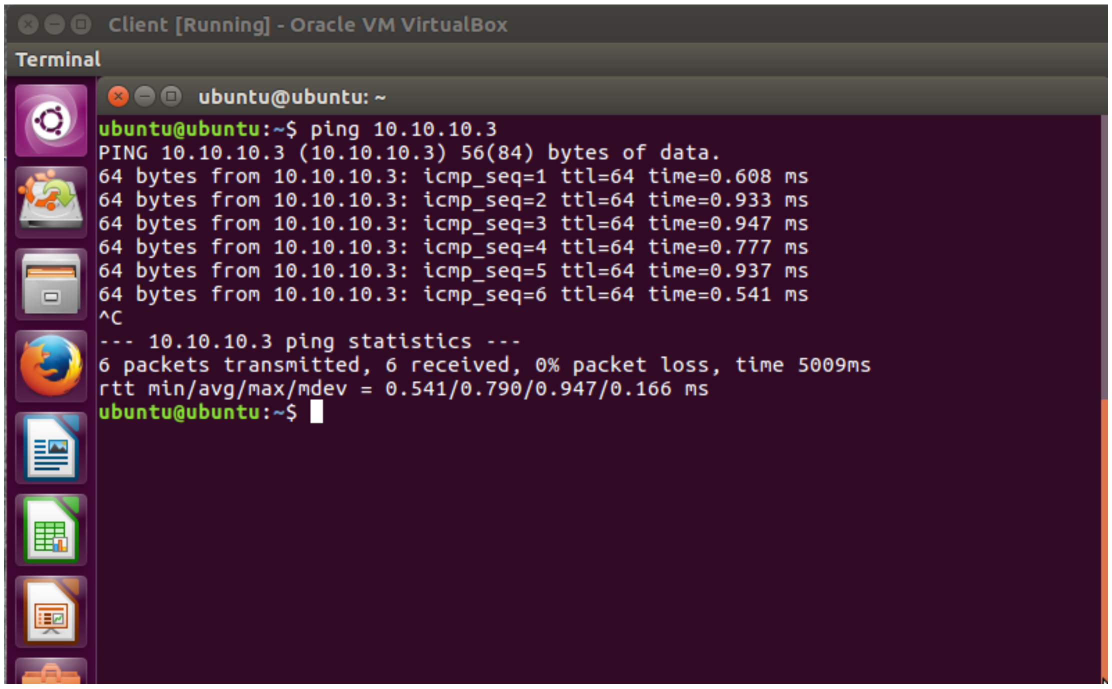
Ping is computer network administration software utility which is used to check reachability of a host. Ping operates with the help of ICMP (Internet Control Message Protocol) , in which it sends echo-request to the host and waits for an echo-reply.

Accept Incoming ping



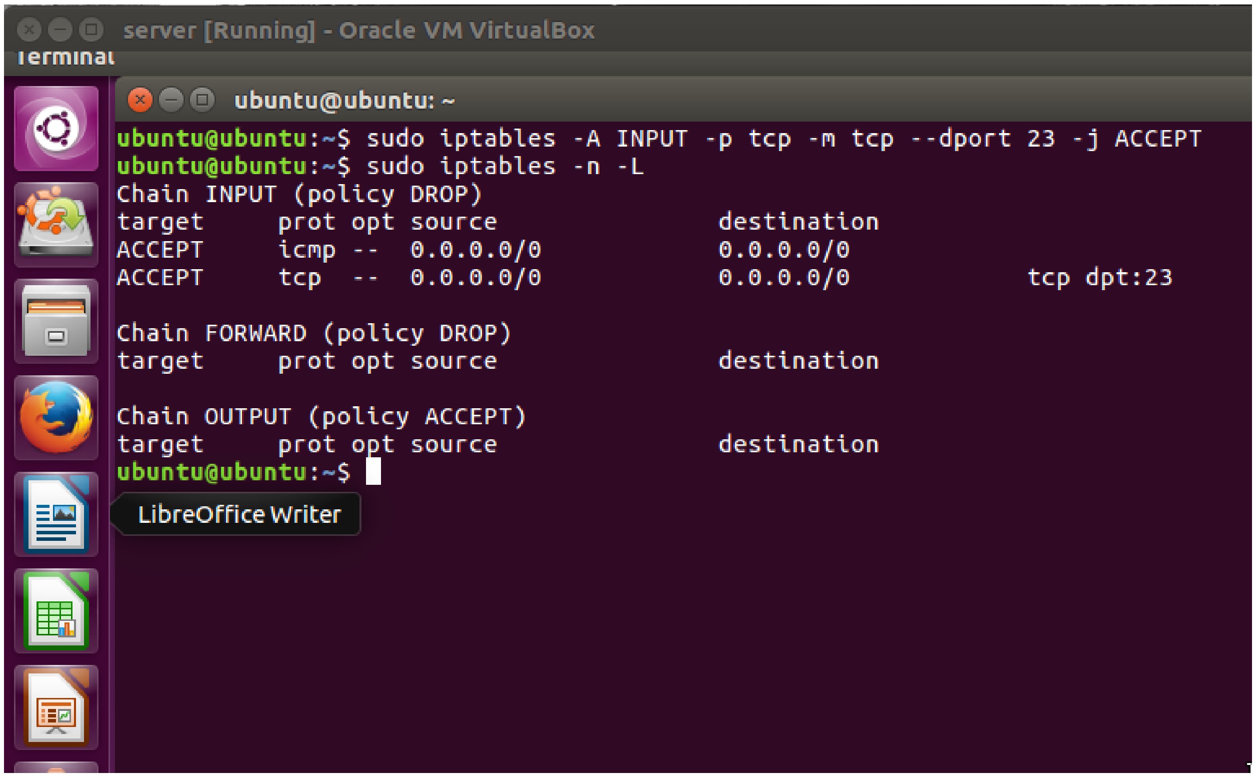
```
ubuntu@ubuntu:~$ sudo iptables -A INPUT -p icmp -j ACCEPT
ubuntu@ubuntu:~$ sudo iptables -n -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ubuntu@ubuntu:~$
```

Verification from client if ping is working



```
ubuntu@ubuntu:~$ ping 10.10.10.3
PING 10.10.10.3 (10.10.10.3) 56(84) bytes of data.
64 bytes from 10.10.10.3: icmp_seq=1 ttl=64 time=0.608 ms
64 bytes from 10.10.10.3: icmp_seq=2 ttl=64 time=0.933 ms
64 bytes from 10.10.10.3: icmp_seq=3 ttl=64 time=0.947 ms
64 bytes from 10.10.10.3: icmp_seq=4 ttl=64 time=0.777 ms
64 bytes from 10.10.10.3: icmp_seq=5 ttl=64 time=0.937 ms
64 bytes from 10.10.10.3: icmp_seq=6 ttl=64 time=0.541 ms
^C
--- 10.10.10.3 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 0.541/0.790/0.947/0.166 ms
ubuntu@ubuntu:~$
```

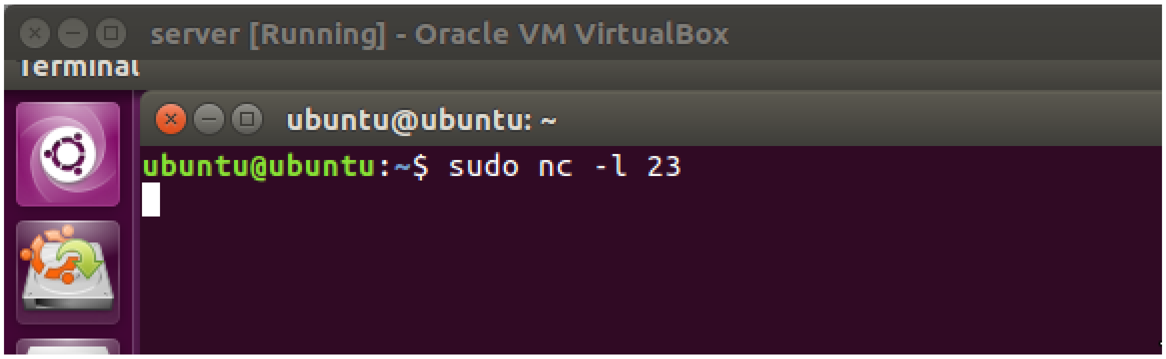
Accept port 23 traffic



```
ubuntu@ubuntu:~$ sudo iptables -A INPUT -p tcp -m tcp --dport 23 -j ACCEPT
ubuntu@ubuntu:~$ sudo iptables -n -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:23

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ubuntu@ubuntu:~$
```

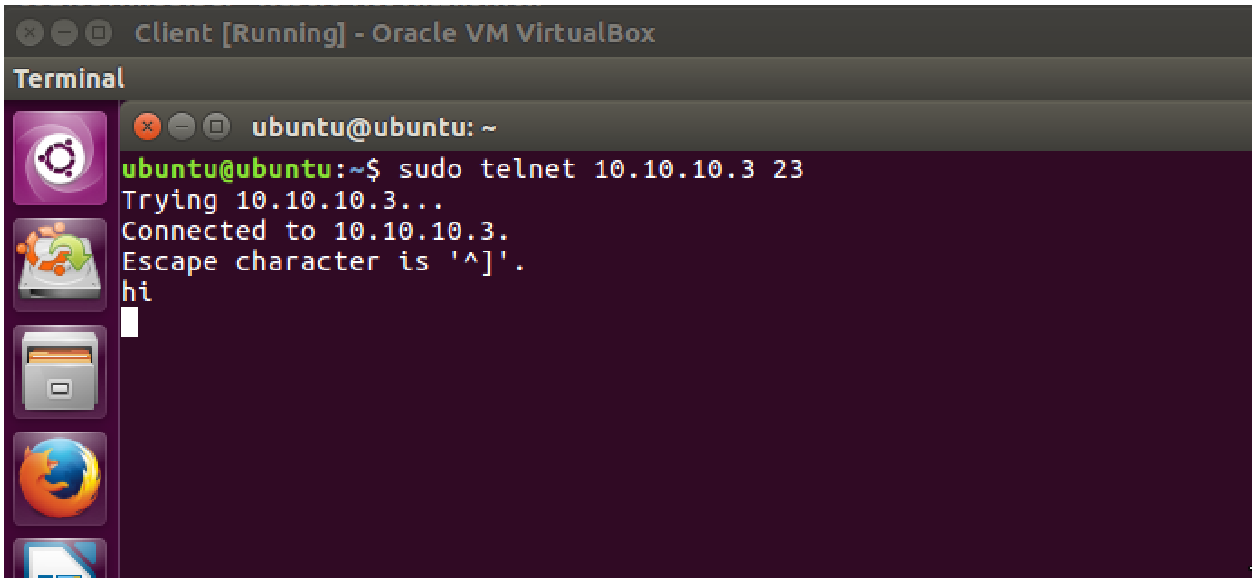
Verification if port 23 Traffic enabled

a)Listening on port 23 (server)



```
ubuntu@ubuntu:~$ sudo nc -l 23
```

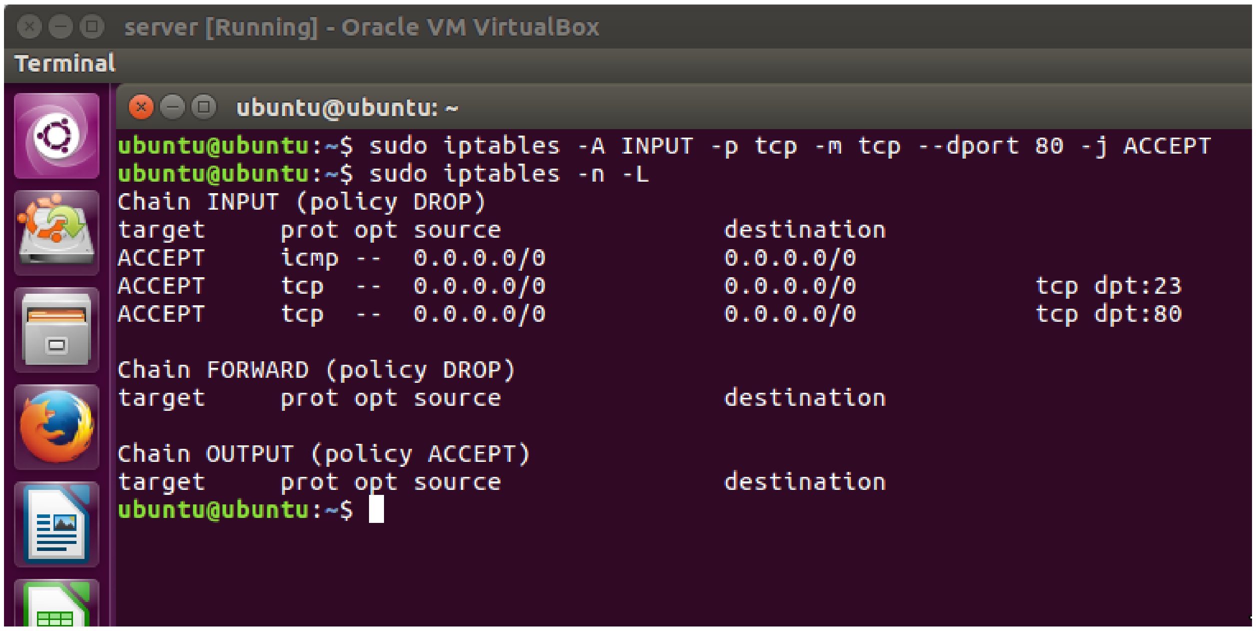
b) Telnet to port 23 from client



```
ubuntu@ubuntu:~$ sudo telnet 10.10.10.3 23
Trying 10.10.10.3...
Connected to 10.10.10.3.
Escape character is '^]'.
hi
```

## 3. Create rules to accept web connections on port 80

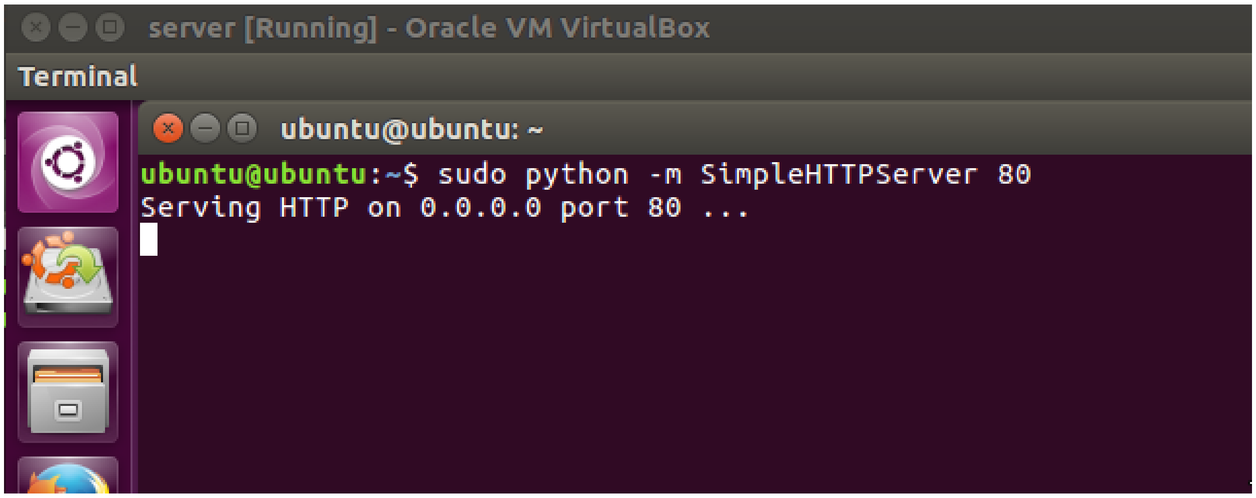
Command To Enable port 80, and the resultant IP Table



```
ubuntu@ubuntu:~$ sudo iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
ubuntu@ubuntu:~$ sudo iptables -n -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:23
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:80

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ubuntu@ubuntu:~$
```

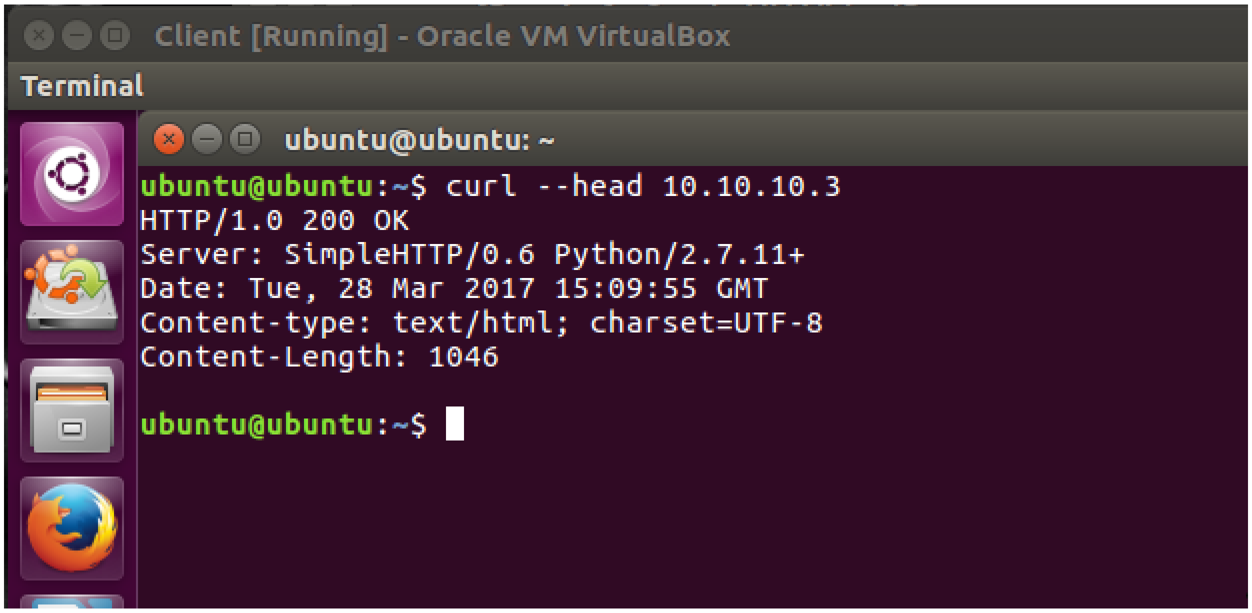
Verification if port 80 is enabled

a)Creating a simple http server at server's machine (Listening on port 80)



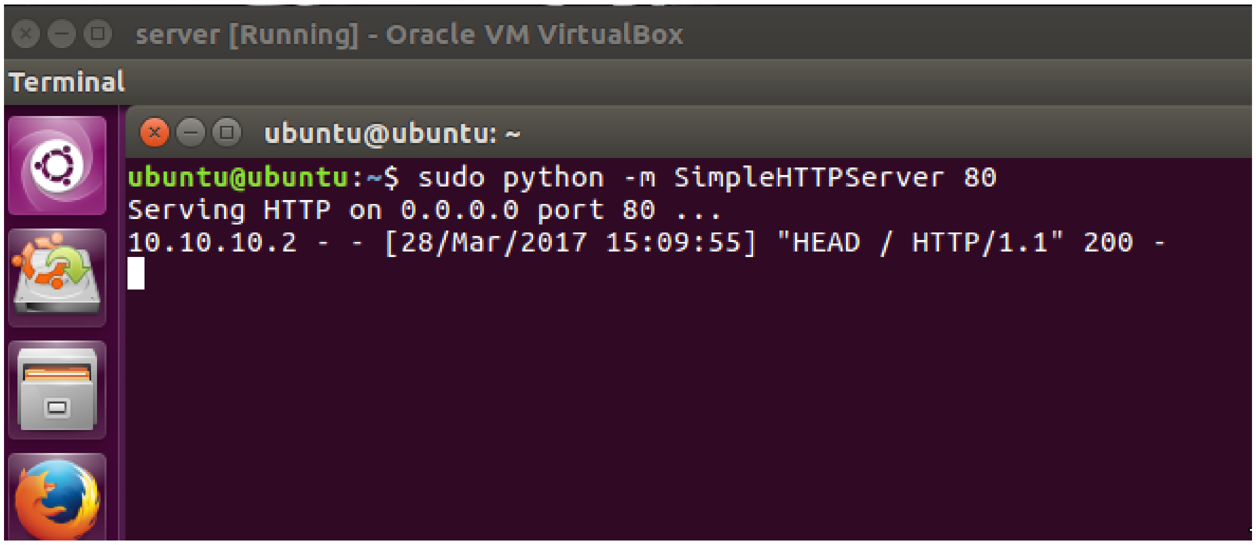
```
ubuntu@ubuntu:~$ sudo python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

b) Pinging on http server (port 80) from client's machine



```
ubuntu@ubuntu:~$ curl --head 10.10.10.3
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/2.7.11+
Date: Tue, 28 Mar 2017 15:09:55 GMT
Content-type: text/html; charset=UTF-8
Content-Length: 1046

ubuntu@ubuntu:~$
```

c) Request received on Server's machine (Ping successful)



```
ubuntu@ubuntu:~$ sudo python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.10.10.2 - - [28/Mar/2017 15:09:55] "HEAD / HTTP/1.1" 200 -
```

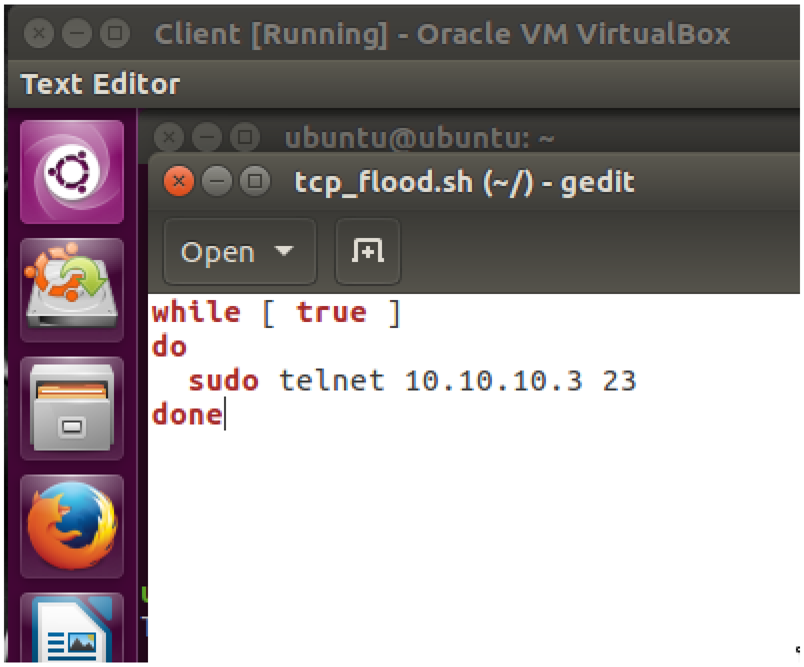
## 4. Write rules to avoid DOS attacks like TCP-SYN, PING-OF-DEATH, INVITE-OF-DEATH . Also explain the attacks briefly with ports involved and how to block them.

### TCP-SYN

SYN flood is a type of DOS (Denial Of Service) attack. SYN flooding is mainly based on the mechanism of TCP handshake

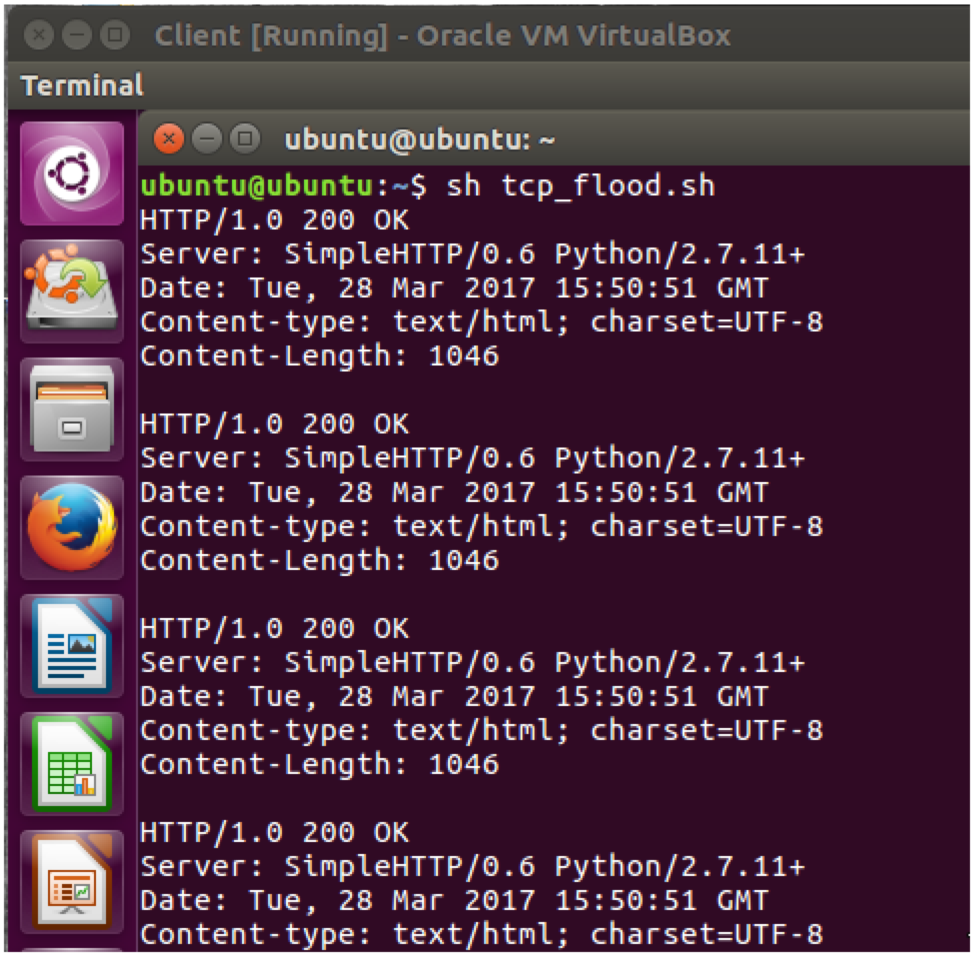
The attacker tries to create lots of SYN request packets in which source IP is changed. Because of this receiver is bound to think that these requests are coming from different sources. Now the response to SYN request packet is SYN/ACK packet after which receiver machine allocate some of its resources to complete this three way handshake. Now receiver is waiting for final ACK response from the sender machines, but never gets back ACK reply. The target machine's resources are exhausted and it stops serving any further requests from any legitimate machine. This attack and some other form of DOS/DDOS attacks can be blocked by limiting the incoming TCP connection request packets. A point to be noted here is that, we should not put a limit to requests from established connections. For avoiding this type of attack, only new connection requests need to be controlled.

a) TCP Flood Program


Client [Running] - Oracle VM VirtualBox
Text Editor
ubuntu@ubuntu: ~
tcp_flood.sh (~/) - gedit
Open

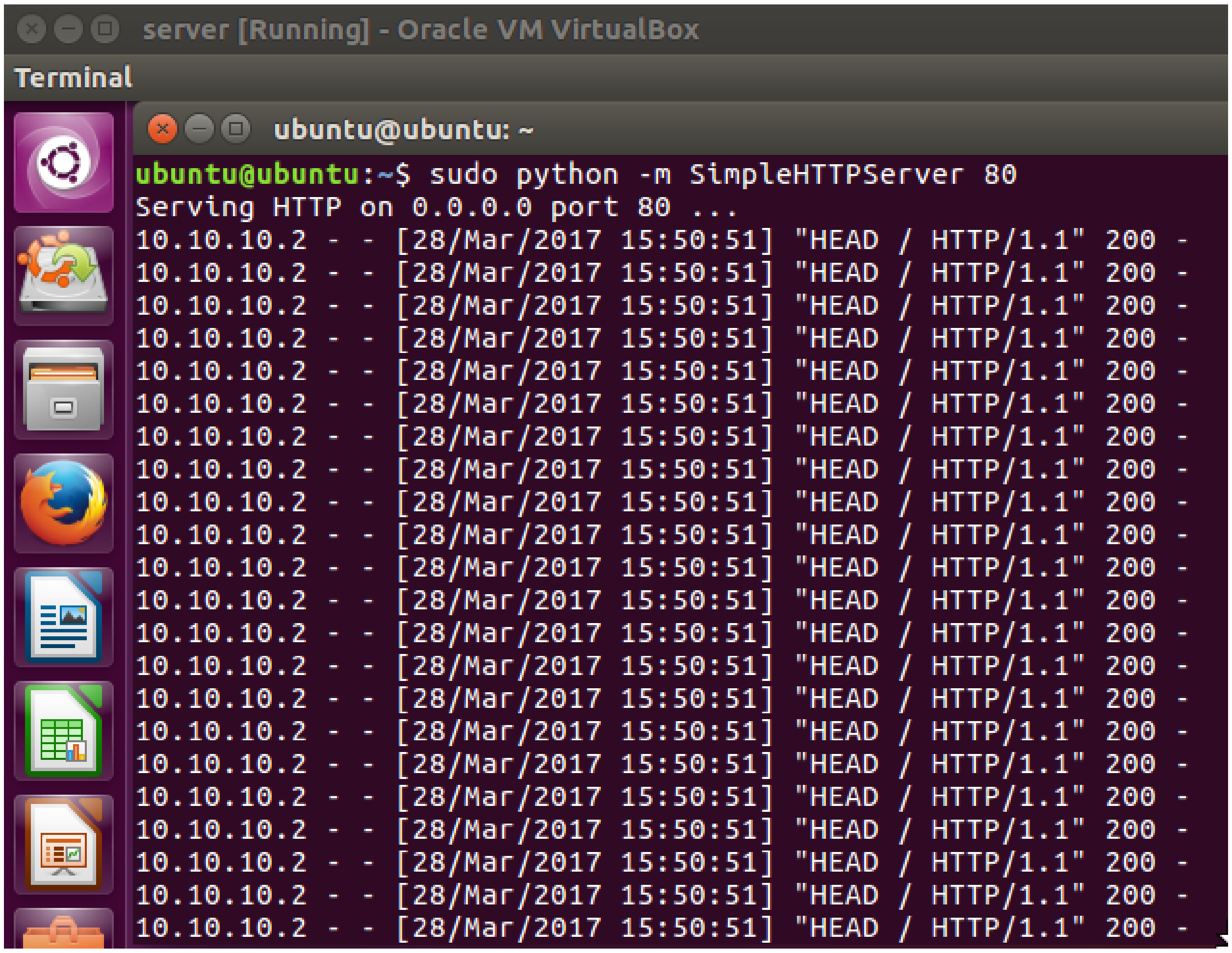
```
while [ true ]
do
  sudo telnet 10.10.10.3 23
done
```

b) Flood Request from Client



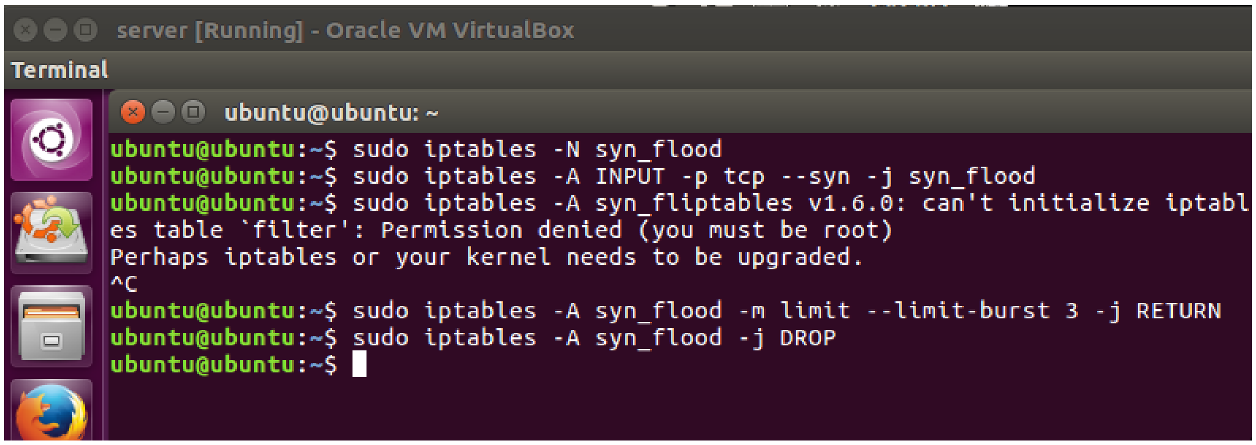
```
ubuntu@ubuntu:~$ sh tcp_flood.sh
HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/2.7.11+
Date: Tue, 28 Mar 2017 15:50:51 GMT
Content-type: text/html; charset=UTF-8
Content-Length: 1046

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/2.7.11+
Date: Tue, 28 Mar 2017 15:50:51 GMT
Content-type: text/html; charset=UTF-8
Content-Length: 1046

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/2.7.11+
Date: Tue, 28 Mar 2017 15:50:51 GMT
Content-type: text/html; charset=UTF-8
Content-Length: 1046

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/2.7.11+
Date: Tue, 28 Mar 2017 15:50:51 GMT
Content-type: text/html; charset=UTF-8
```

c) Serving TCP Flood Request at Server



```
ubuntu@ubuntu:~$ sudo python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
10.10.10.2 - - [28/Mar/2017 15:50:51] "HEAD / HTTP/1.1" 200 -
```

d) Avoiding TCP Flood at server



```
ubuntu@ubuntu:~$ sudo iptables -N syn_flood
ubuntu@ubuntu:~$ sudo iptables -A INPUT -p tcp --syn -j syn_flood
ubuntu@ubuntu:~$ sudo iptables -A syn_fliptables v1.6.0: can't initialize iptabl
es table `filter': Permission denied (you must be root)
Perhaps iptables or your kernel needs to be upgraded.
^C
ubuntu@ubuntu:~$ sudo iptables -A syn_flood -m limit --limit-burst 3 -j RETURN
ubuntu@ubuntu:~$ sudo iptables -A syn_flood -j DROP
ubuntu@ubuntu:~$
```

## PING OF DEATH

Here attacker can cause a remote system to crash by sending a single malformed IP packet. Most of the operating system are patched against this kind of attacks, basically all the operating system which are released after 1997.

In this, attacker is going to create a ICMP echo request which is commonly known as **PING** packet. Now generally this type of packets are used to check if the remote system is running or not. In practice, size of such echo request packet is very small but in theory it can be very big. Now according to the standard documentation IP packet can have length upto 65536 bytes. Here is the trick, what attacker is going to do is formulate an IP packet i.e., our echo request which is having more than 65536 bytes limit. Absolutely this violates the general rules so are the attacker is going to achieve this. Now the attacker is going to exploit the concept of fragmentation. We are talking on the level of physical abilities, every physical connection has its limit that is called as **MTU.** When we send a packet that is actually bigger than the MTU then it is bound to be fragmented. Receiver needs to join all this fragments together to form a IP packet so every fragment consist of offset and actual data. When attacker sends a packet that is bigger than the actual limits of IP then at the time of reassembly at receiver side, various internal data structure are bound to get overflow  This may result into system crash.

Obviously the solution for this type of attack is very simple. We just need to put a check on the simple parameters like

**offset + actual data <= 65536**

For each fragment that we are going to receive. When the above property is violated we can simply drop that packet.

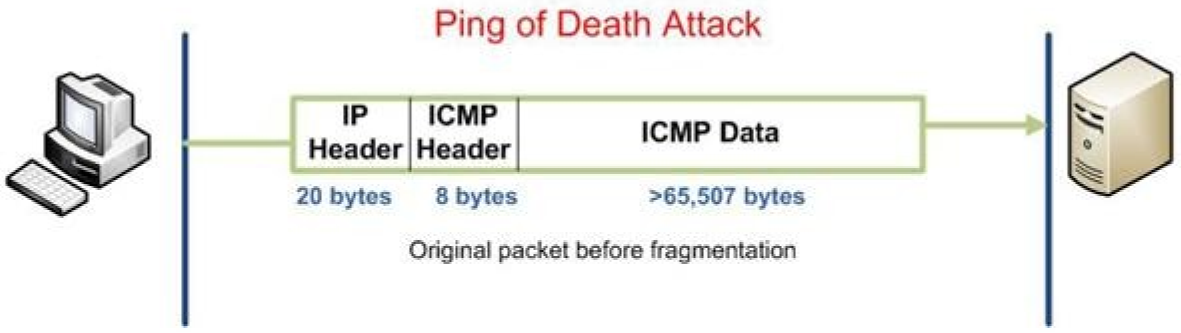
Now normally we need **ping** requests to work properly , and for generalization purpose we can say that for ping of death attack to work they require their other fragments to work. So we are trying to formulate our iptable rule as follows:

```
Sudo iptables -A INPUT -p icmp -f -j -DROP
```

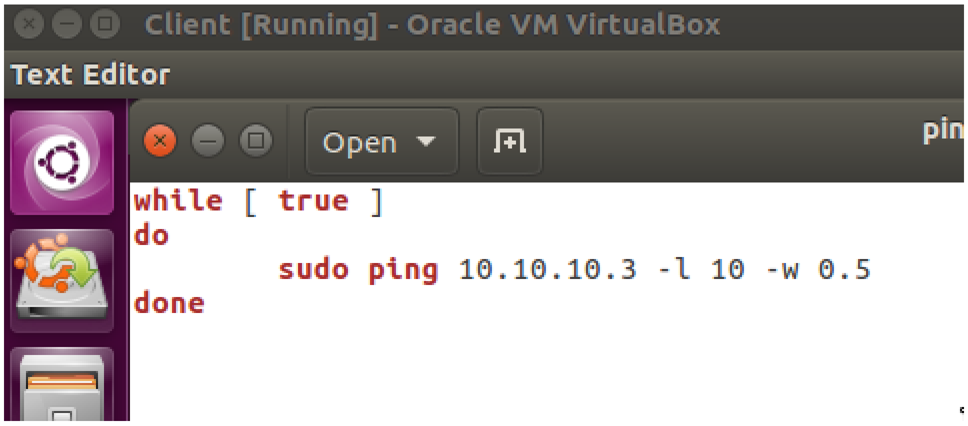
**The -f option guarantees that this rule will be applied for icmp packets which are consist of more than one fragments. Therefore**

**attacker will not be able to create IP packets which are bigger than max specified length of the IP packet.**

There is a specific ICMP echo variation that could cause a system crash. The difference of the echo request from the normal ones is the large size of IP packet it contains. RFC 791 specifies that the maximum size of an IP packet is 65,535 bytes. An ICMP echo request with more than 65,507 (65,535-20-8) bytes of data could cause a remote system to crash while reassembling the packet fragments.

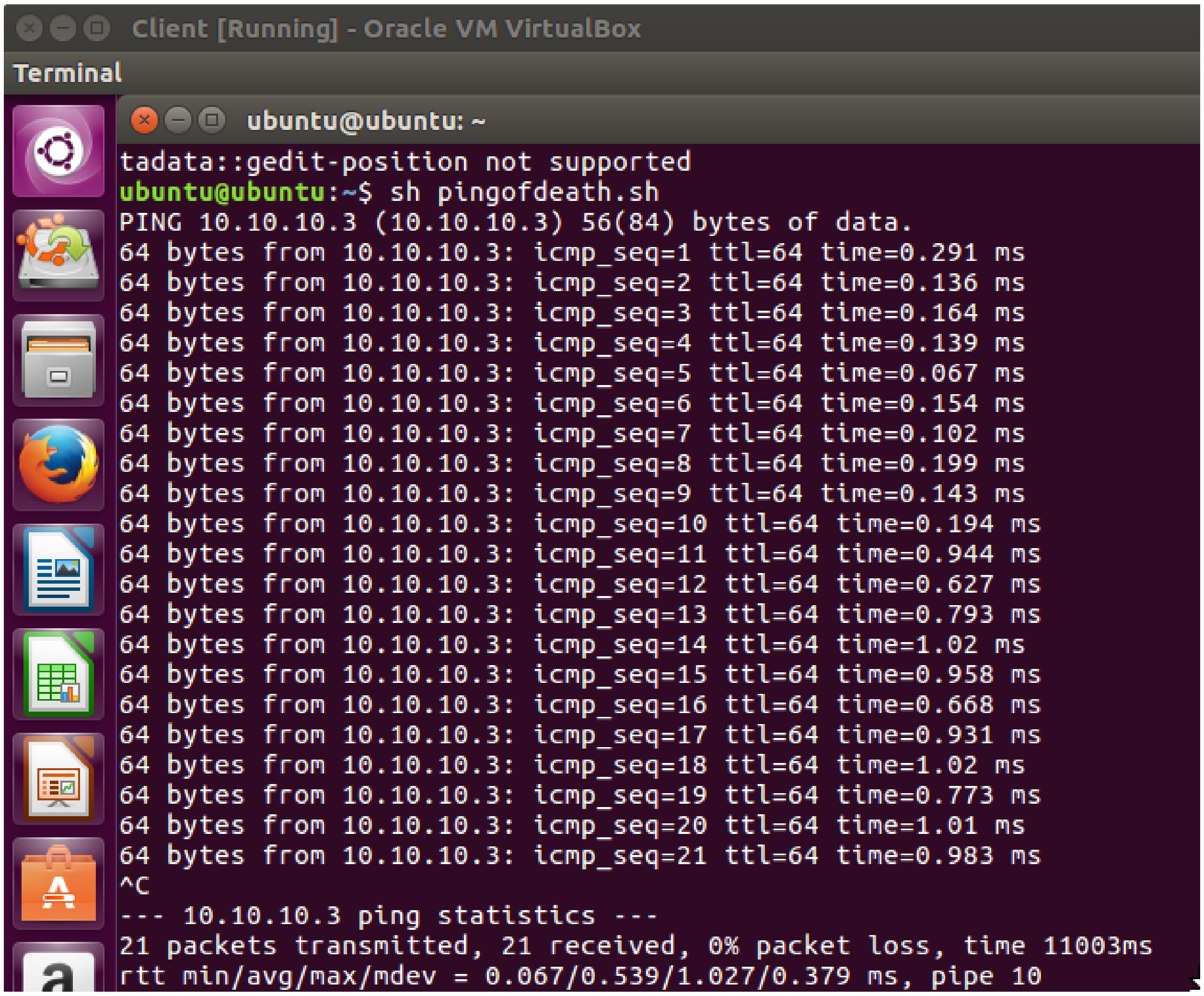


a) Ping of Death script



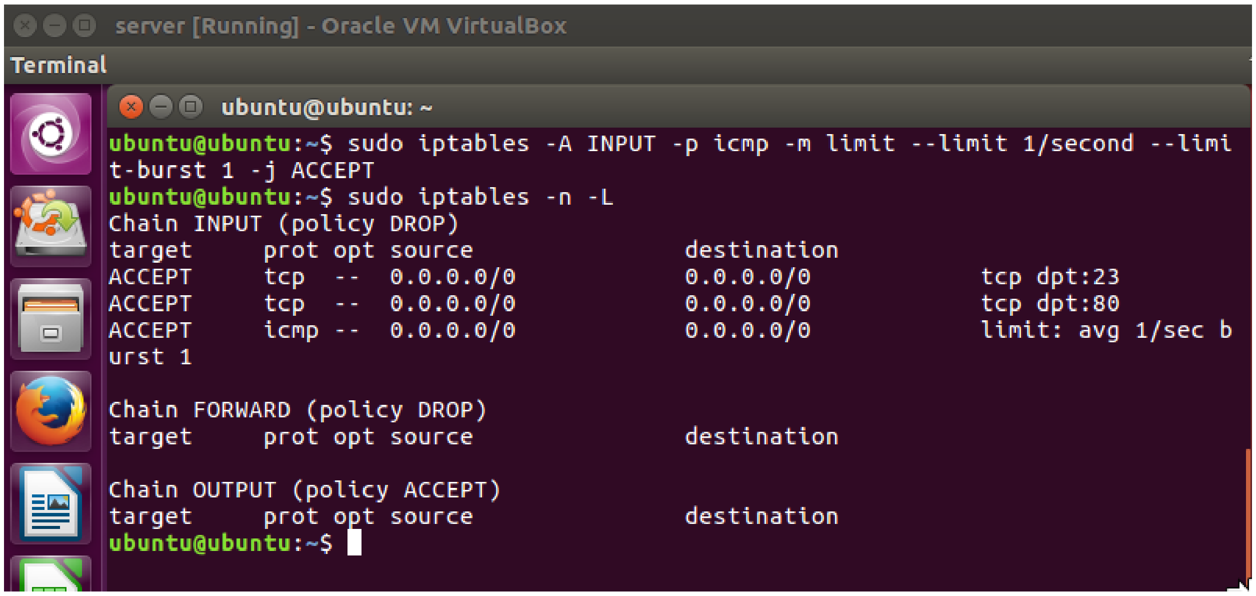
```
while [ true ]
do
        sudo ping 10.10.10.3 -l 10 -w 0.5
done
```

b) Demonstration



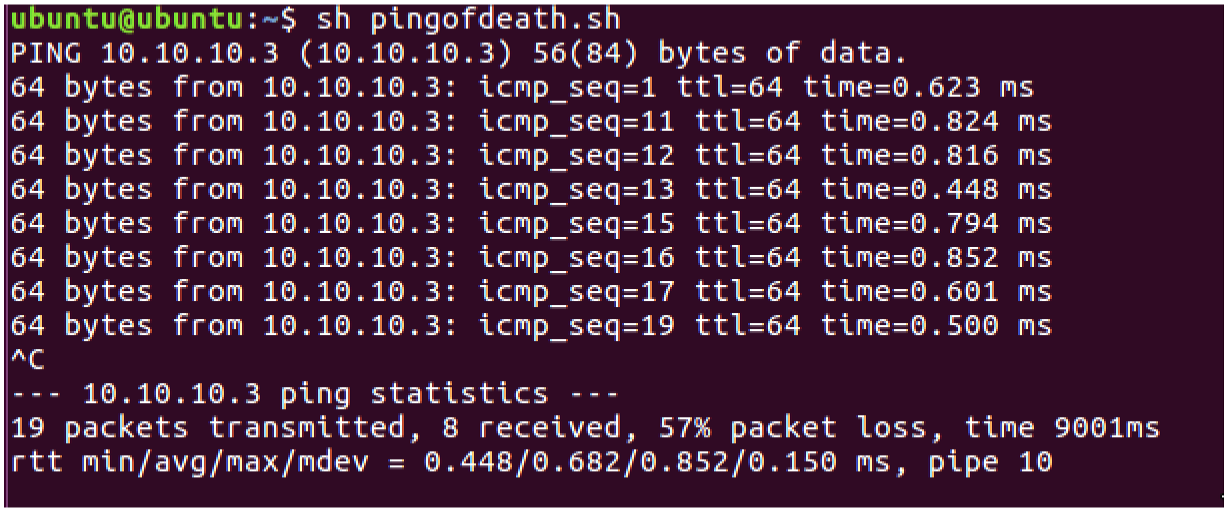
```
tadata::gedit-position not supported
ubuntu@ubuntu:~$ sh pingofdeath.sh
PING 10.10.10.3 (10.10.10.3) 56(84) bytes of data.
64 bytes from 10.10.10.3: icmp_seq=1 ttl=64 time=0.291 ms
64 bytes from 10.10.10.3: icmp_seq=2 ttl=64 time=0.136 ms
64 bytes from 10.10.10.3: icmp_seq=3 ttl=64 time=0.164 ms
64 bytes from 10.10.10.3: icmp_seq=4 ttl=64 time=0.139 ms
64 bytes from 10.10.10.3: icmp_seq=5 ttl=64 time=0.067 ms
64 bytes from 10.10.10.3: icmp_seq=6 ttl=64 time=0.154 ms
64 bytes from 10.10.10.3: icmp_seq=7 ttl=64 time=0.102 ms
64 bytes from 10.10.10.3: icmp_seq=8 ttl=64 time=0.199 ms
64 bytes from 10.10.10.3: icmp_seq=9 ttl=64 time=0.143 ms
64 bytes from 10.10.10.3: icmp_seq=10 ttl=64 time=0.194 ms
64 bytes from 10.10.10.3: icmp_seq=11 ttl=64 time=0.944 ms
64 bytes from 10.10.10.3: icmp_seq=12 ttl=64 time=0.627 ms
64 bytes from 10.10.10.3: icmp_seq=13 ttl=64 time=0.793 ms
64 bytes from 10.10.10.3: icmp_seq=14 ttl=64 time=1.02 ms
64 bytes from 10.10.10.3: icmp_seq=15 ttl=64 time=0.958 ms
64 bytes from 10.10.10.3: icmp_seq=16 ttl=64 time=0.668 ms
64 bytes from 10.10.10.3: icmp_seq=17 ttl=64 time=0.931 ms
64 bytes from 10.10.10.3: icmp_seq=18 ttl=64 time=1.02 ms
64 bytes from 10.10.10.3: icmp_seq=19 ttl=64 time=0.773 ms
64 bytes from 10.10.10.3: icmp_seq=20 ttl=64 time=1.01 ms
64 bytes from 10.10.10.3: icmp_seq=21 ttl=64 time=0.983 ms
^C
--- 10.10.10.3 ping statistics ---
21 packets transmitted, 21 received, 0% packet loss, time 11003ms
rtt min/avg/max/mdev = 0.067/0.539/1.027/0.379 ms, pipe 10
```

c)Limiting POD at server



```
ubuntu@ubuntu:~$ sudo iptables -A INPUT -p icmp -m limit --limit 1/second --limi
t-burst 1 -j ACCEPT
ubuntu@ubuntu:~$ sudo iptables -n -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:23
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp dpt:80
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            limit: avg 1/sec b
urst 1

Chain FORWARD (policy DROP)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ubuntu@ubuntu:~$
```

d) Checking correctness of command



```
ubuntu@ubuntu:~$ sh pingofdeath.sh
PING 10.10.10.3 (10.10.10.3) 56(84) bytes of data.
64 bytes from 10.10.10.3: icmp_seq=1 ttl=64 time=0.623 ms
64 bytes from 10.10.10.3: icmp_seq=11 ttl=64 time=0.824 ms
64 bytes from 10.10.10.3: icmp_seq=12 ttl=64 time=0.816 ms
64 bytes from 10.10.10.3: icmp_seq=13 ttl=64 time=0.448 ms
64 bytes from 10.10.10.3: icmp_seq=15 ttl=64 time=0.794 ms
64 bytes from 10.10.10.3: icmp_seq=16 ttl=64 time=0.852 ms
64 bytes from 10.10.10.3: icmp_seq=17 ttl=64 time=0.601 ms
64 bytes from 10.10.10.3: icmp_seq=19 ttl=64 time=0.500 ms
^C
--- 10.10.10.3 ping statistics ---
19 packets transmitted, 8 received, 57% packet loss, time 9001ms
rtt min/avg/max/mdev = 0.448/0.682/0.852/0.150 ms, pipe 10
```

We see that, after setting up ip-tables to drop flooded ping requests, there is a **drop of 57% packets** which was sent from client to server. Therefore our ip table configuration works.

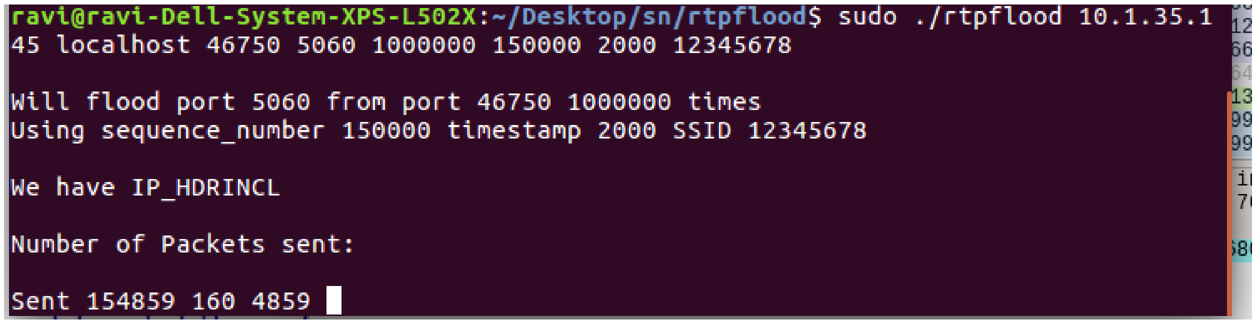
## INVITE-OF-DEATH

An INVITE of Death is a type of attack on a VOIP-system that involves sending a malformed or otherwise malicious SIP INVITE request to a telephony server, resulting in a crash of that server. Because telephony is usually a critical application, this damage causes significant disruption to the users and poses tremendous acceptance problems with VoIP. These kinds of attacks do not necessarily affect only SIP-based systems; all implementations with vulnerabilities in the VoIP area are affected. The DoS attack can also be transported in other messages than INVITE.

### Demonstration

#### a) Attack using rtpflood:-

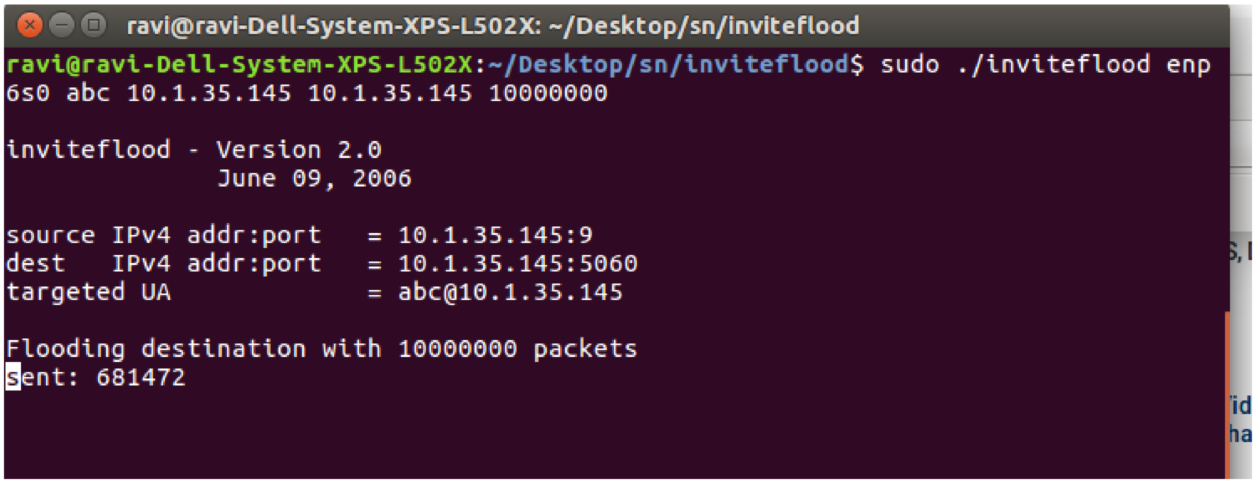
***RTPFlood*** is a command line tool used to flood any device that is processing RTP. Rtp flood is used to flood a target IP phone with a UDP packet contains a RTP data In order to launch a successful attack using rtp flood you will need know the RTP listening port on the remote device you want to attack, for example; x-lite sofphone default rtp port is 8000.



```
ravi@ravi-Dell-System-XPS-L502X:~/Desktop/sn/rtpflood$ sudo ./rtpflood 10.1.35.1
45 localhost 46750 5060 1000000 150000 2000 12345678

Will flood port 5060 from port 46750 1000000 times
Using sequence_number 150000 timestamp 2000 SSID 12345678

We have IP_HDRINCL

Number of Packets sent:

Sent 154859 160 4859
```

b) Attack using inviteflood:-

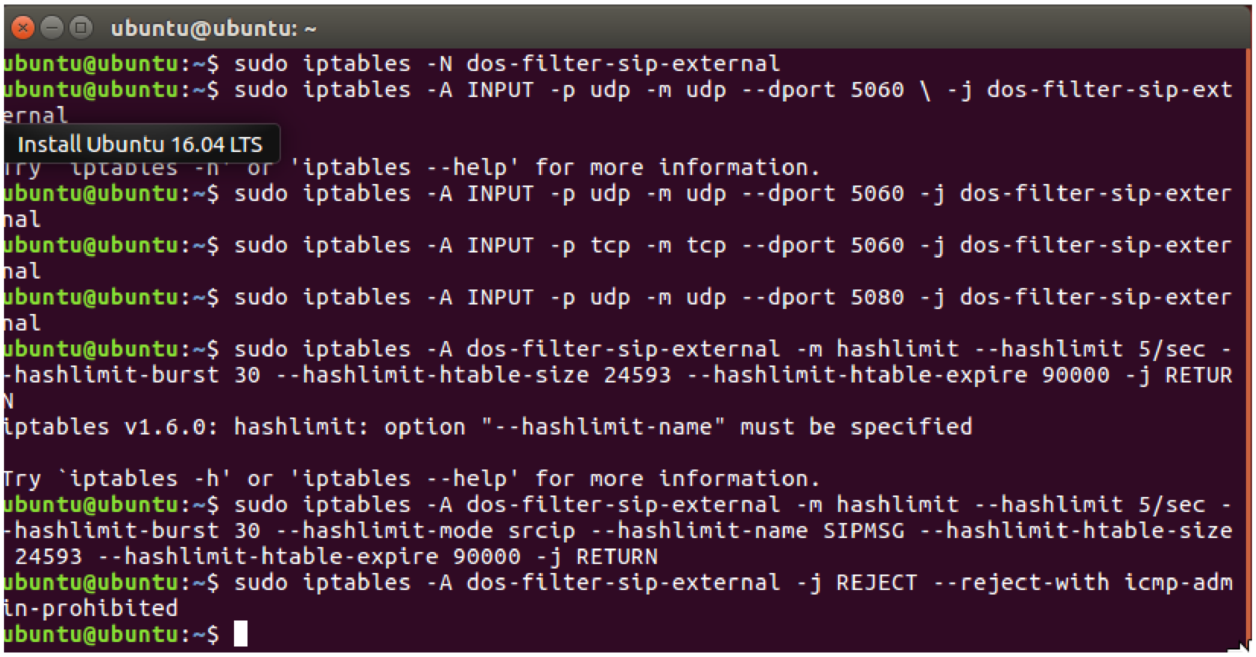
***InviteFlood*** a tool to perform SIP/SDP INVITE message flooding over UDP/IP



```
ravi@ravi-Dell-System-XPS-L502X:~/Desktop/sn/inviteflood$ sudo ./inviteflood enp
6s0 abc 10.1.35.145 10.1.35.145 10000000

inviteflood - Version 2.0
              June 09, 2006

source IPv4 addr:port   = 10.1.35.145:9
dest   IPv4 addr:port   = 10.1.35.145:5060
targeted UA             = abc@10.1.35.145

Flooding destination with 10000000 packets
sent: 681472
```

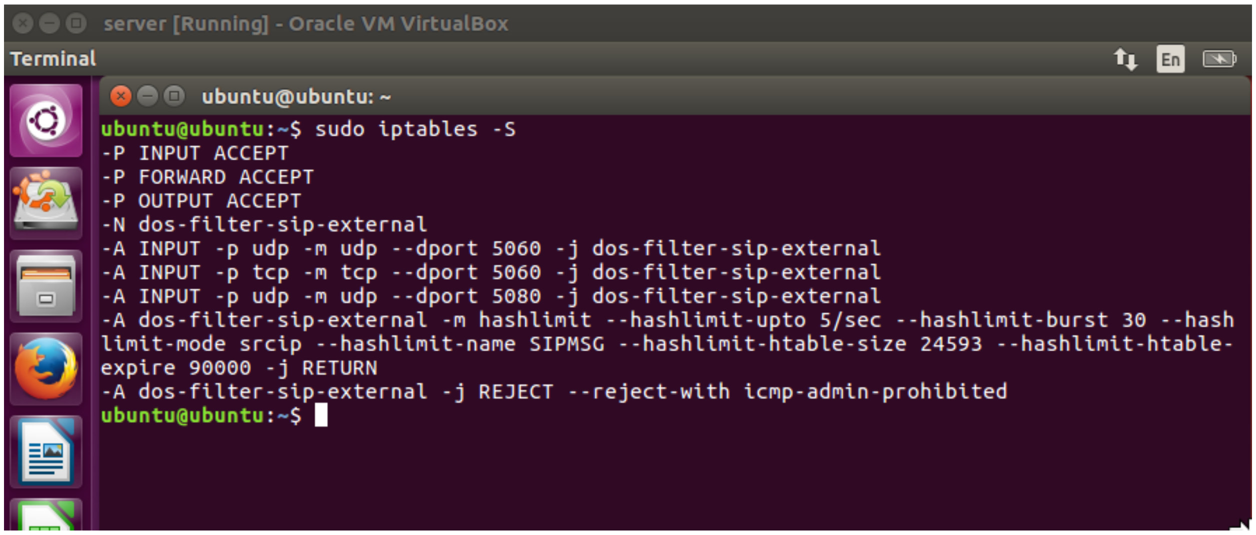
## Defence

(Assumption: Port 5060/5080 are used for VOIP applications such as Linphone)



```
ubuntu@ubuntu:~$ sudo iptables -N dos-filter-sip-external
ubuntu@ubuntu:~$ sudo iptables -A INPUT -p udp -m udp --dport 5060 \ -j dos-filter-sip-ext
ernal

Try `iptables -h' or 'iptables --help' for more information.
ubuntu@ubuntu:~$ sudo iptables -A INPUT -p udp -m udp --dport 5060 -j dos-filter-sip-exter
nal
ubuntu@ubuntu:~$ sudo iptables -A INPUT -p tcp -m tcp --dport 5060 -j dos-filter-sip-exter
nal
ubuntu@ubuntu:~$ sudo iptables -A INPUT -p udp -m udp --dport 5080 -j dos-filter-sip-exter
nal
ubuntu@ubuntu:~$ sudo iptables -A dos-filter-sip-external -m hashlimit --hashlimit 5/sec -
-hashlimit-burst 30 --hashlimit-htable-size 24593 --hashlimit-htable-expire 90000 -j RETUR
N
iptables v1.6.0: hashlimit: option "--hashlimit-name" must be specified

Try `iptables -h' or 'iptables --help' for more information.
ubuntu@ubuntu:~$ sudo iptables -A dos-filter-sip-external -m hashlimit --hashlimit 5/sec -
-hashlimit-burst 30 --hashlimit-mode srcip --hashlimit-name SIPMSG --hashlimit-htable-size
 24593 --hashlimit-htable-expire 90000 -j RETURN
ubuntu@ubuntu:~$ sudo iptables -A dos-filter-sip-external -j REJECT --reject-with icmp-adm
in-prohibited
ubuntu@ubuntu:~$ 
```

Resultant IP table



```
ubuntu@ubuntu:~$ sudo iptables -S
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
-N dos-filter-sip-external
-A INPUT -p udp -m udp --dport 5060 -j dos-filter-sip-external
-A INPUT -p tcp -m tcp --dport 5060 -j dos-filter-sip-external
-A INPUT -p udp -m udp --dport 5080 -j dos-filter-sip-external
-A dos-filter-sip-external -m hashlimit --hashlimit-upto 5/sec --hashlimit-burst 30 --hash
limit-mode srcip --hashlimit-name SIPMSG --hashlimit-htable-size 24593 --hashlimit-htable-
expire 90000 -j RETURN
-A dos-filter-sip-external -j REJECT --reject-with icmp-admin-prohibited
ubuntu@ubuntu:~$ 
```