CS255 Computer Security Lab -4

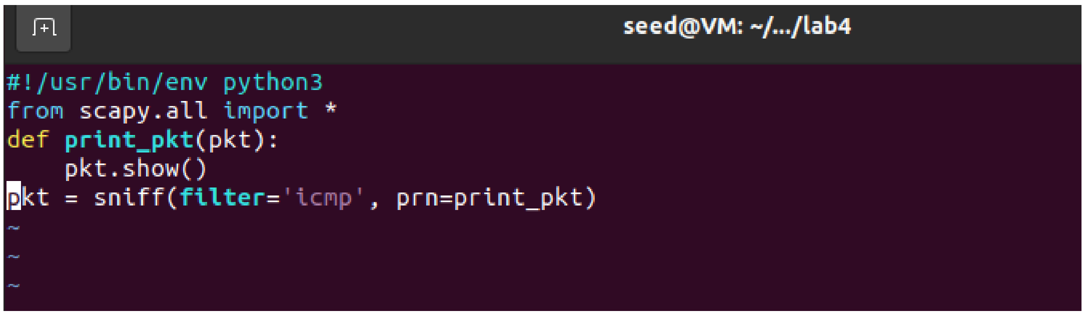
Name: Prince Choudhary

SID: 862254827

1.1 Sniffing Packets

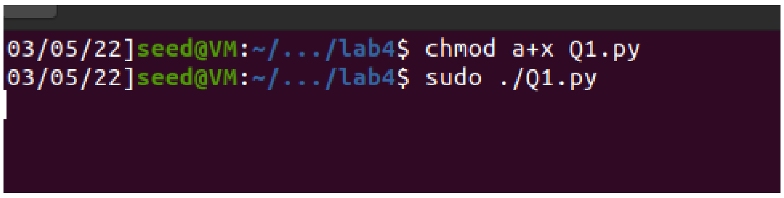
1.1A

We will first install Scapy package.
Now we will run the sniff script in python using the Scapy method sniff.
This program sniffs icmp messages and print in the terminal.



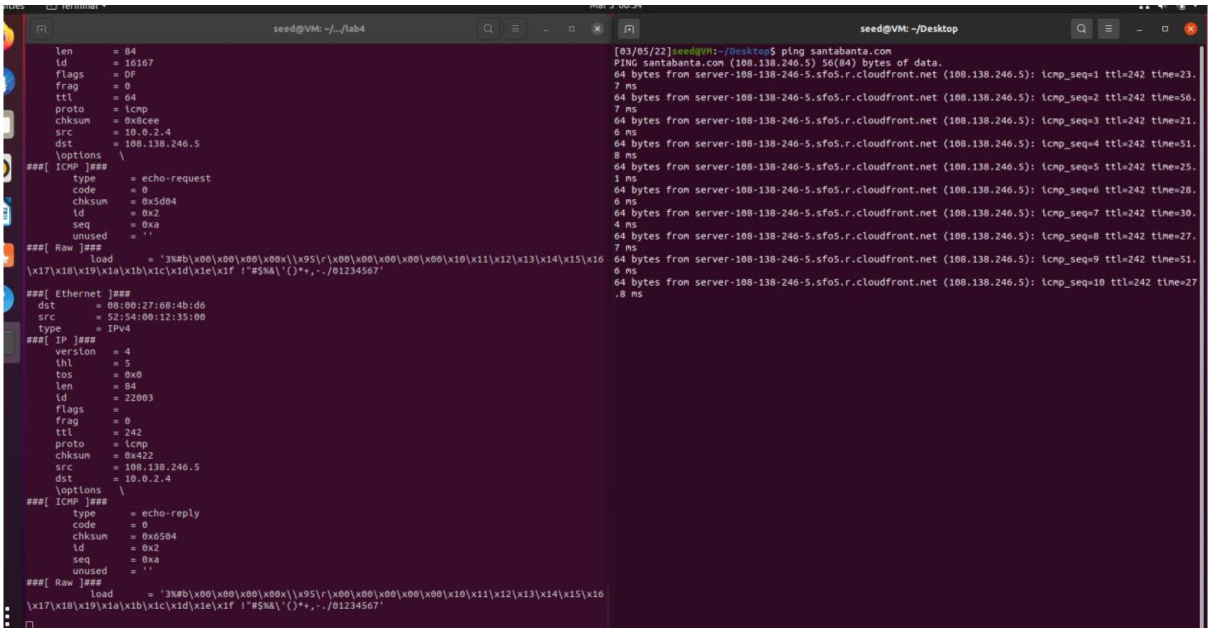
```
seed@VM: ~/.../lab4

#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
    pkt.show()
pkt = sniff(filter='icmp', prn=print_pkt)
~
~
~
```

Now we will give execution privileges to the file and run it in root privilege mode.



```
03/05/22]seed@VM:~/.../lab4$ chmod a+x Q1.py
03/05/22]seed@VM:~/.../lab4$ sudo ./Q1.py
```

Now we will open another terminal and send a ping

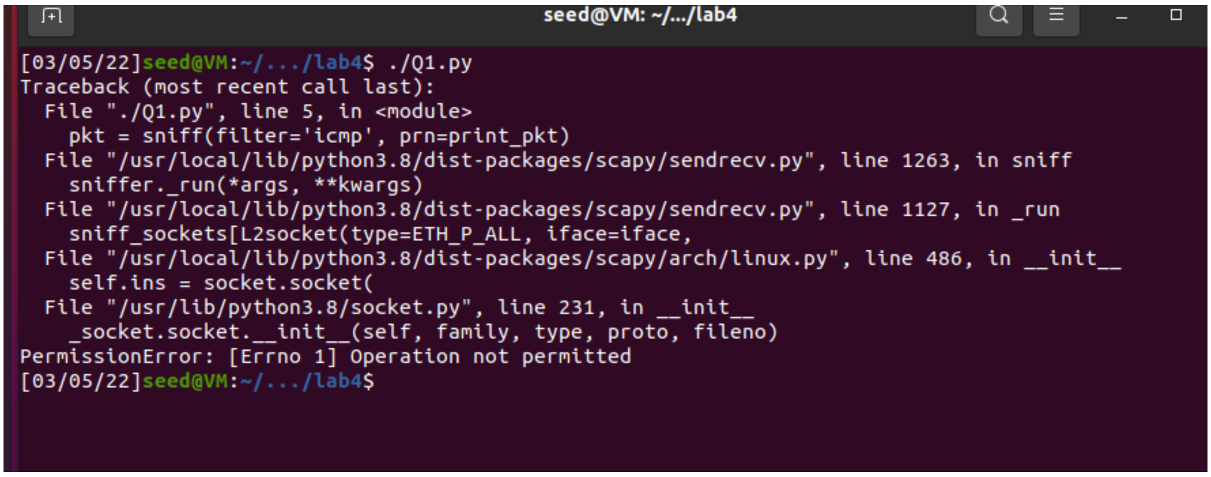


```
seed@VM: ~/.../lab4

     len       = 84
     id        = 16167
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x8cee
     src       = 10.0.2.4
     dst       = 108.138.246.5
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x5d04
        id        = 0x2
        seq       = 0xa
        unused    = ''
###[ Raw ]###
           load      = '3%#b\x00\x00\x00\x00x\\x95\r\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16
\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 08:00:27:68:4b:d6
  src       = 52:54:00:12:35:00
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 22003
     flags     =
     frag      = 0
     ttl       = 242
     proto     = icmp
     chksum    = 0x422
     src       = 108.138.246.5
     dst       = 10.0.2.4
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x6504
        id        = 0x2
        seq       = 0xa
        unused    = ''
###[ Raw ]###
           load      = '3%#b\x00\x00\x00\x00x\\x95\r\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16
\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

```
seed@VM: ~/Desktop

[03/05/22]seed@VM:~/Desktop$ ping santabanta.com
PING santabanta.com (108.138.246.5) 56(84) bytes of data.
64 bytes from server-108-138-246-5.sfo5.r.cloudfront.net (108.138.246.5): icmp_seq=1 ttl=242 time=23.
7 ms
64 bytes from server-108-138-246-5.sfo5.r.cloudfront.net (108.138.246.5): icmp_seq=2 ttl=242 time=56.
7 ms
64 bytes from server-108-138-246-5.sfo5.r.cloudfront.net (108.138.246.5): icmp_seq=3 ttl=242 time=21.
6 ms
64 bytes from server-108-138-246-5.sfo5.r.cloudfront.net (108.138.246.5): icmp_seq=4 ttl=242 time=51.
8 ms
64 bytes from server-108-138-246-5.sfo5.r.cloudfront.net (108.138.246.5): icmp_seq=5 ttl=242 time=25.
1 ms
64 bytes from server-108-138-246-5.sfo5.r.cloudfront.net (108.138.246.5): icmp_seq=6 ttl=242 time=28.
6 ms
64 bytes from server-108-138-246-5.sfo5.r.cloudfront.net (108.138.246.5): icmp_seq=7 ttl=242 time=30.
4 ms
64 bytes from server-108-138-246-5.sfo5.r.cloudfront.net (108.138.246.5): icmp_seq=8 ttl=242 time=27.
7 ms
64 bytes from server-108-138-246-5.sfo5.r.cloudfront.net (108.138.246.5): icmp_seq=9 ttl=242 time=51.
6 ms
64 bytes from server-108-138-246-5.sfo5.r.cloudfront.net (108.138.246.5): icmp_seq=10 ttl=242 time=27
.8 ms
```

We can see from the screenshot that we are sending echo request packet and getting the echo-reply from santabanta.com.

If we attempt to run the program without using the root privileges, we will get permission error message as shown in the figure.



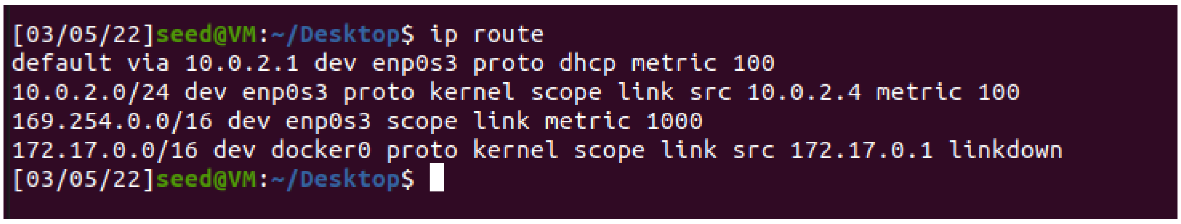
```
[03/05/22]seed@VM:~/.../lab4$ ./Q1.py
Traceback (most recent call last):
  File "./Q1.py", line 5, in <module>
    pkt = sniff(filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1263, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1127, in _run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 486, in __init__
    self.ins = socket.socket(
  File "/usr/lib/python3.8/socket.py", line 231, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
[03/05/22]seed@VM:~/.../lab4$
```

1.1B

In this part of the problem we need to set 3 filters and then demonstrate the sniffer program.
The filters are:
i) Capture only ICMP packet.
ii) Capture any TCP packet that comes from a particular IP and with a destination port number 23.
iii) Capture packets comes from or to go to a particular subnet. Pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.

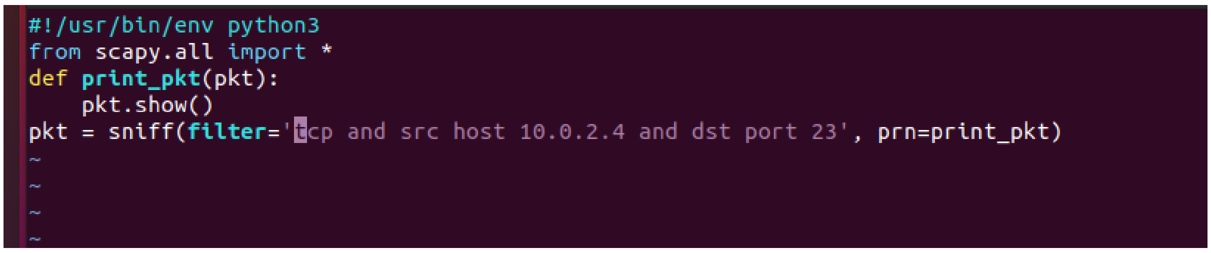
i) This one require us to filter ICMP packet which we just did in 1.1A, we can refer it again for this solution

ii) First we need to know the IP address. My IP address is 10.0.2.4 as it can be seen from screenshot.



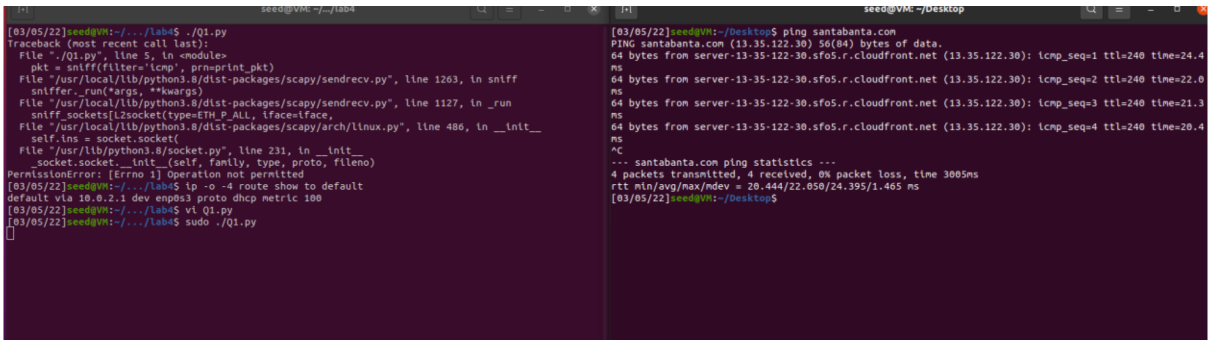
```
[03/05/22]seed@VM:~/Desktop$ ip route
default via 10.0.2.1 dev enp0s3 proto dhcp metric 100
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.4 metric 100
169.254.0.0/16 dev enp0s3 scope link metric 1000
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1 linkdown
[03/05/22]seed@VM:~/Desktop$
```

Now we will make changes in our python file to filter only tcp packets from the host IP 10.0.2.4 and going to port 23. Below is the code after the changes made.



```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
    pkt.show()
pkt = sniff(filter='tcp and src host 10.0.2.4 and dst port 23', prn=print_pkt)
~
~
~
~
```

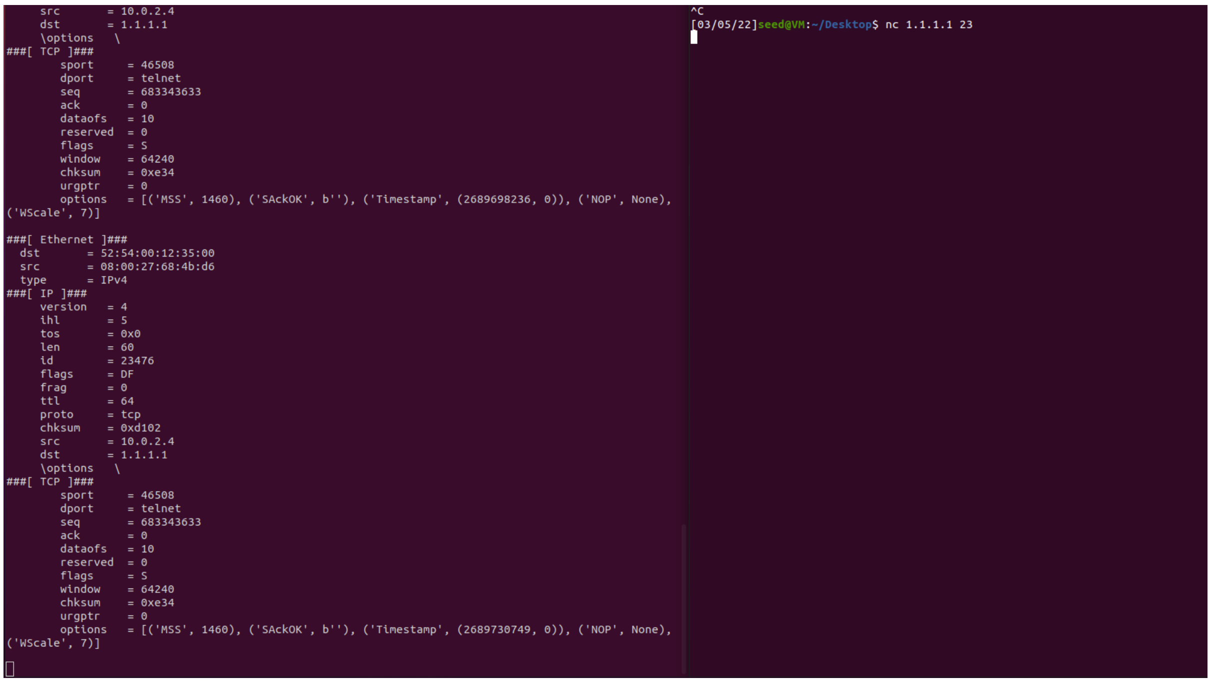
Now we will run the file with sudo privileges. If we try to ping any random host it won't be sniffed by the code. As we have set the host IP.



```
[03/05/22]seed@VM:~/.../lab4$ ./Q1.py
Traceback (most recent call last):
  File "./Q1.py", line 5, in <module>
    pkt = sniff(filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1263, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1127, in _run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 486, in __init__
    self.ins = socket.socket(
  File "/usr/lib/python3.8/socket.py", line 231, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
[03/05/22]seed@VM:~/.../lab4$ ip -o -4 route show to default
default via 10.0.2.1 dev enp0s3 proto dhcp metric 100
[03/05/22]seed@VM:~/.../lab4$ vi Q1.py
[03/05/22]seed@VM:~/.../lab4$ sudo ./Q1.py
```

```
[03/05/22]seed@VM:~/Desktop$ ping santabanta.com
PING santabanta.com (13.35.122.30) 56(84) bytes of data.
64 bytes from server-13-35-122-30.sfo5.r.cloudfront.net (13.35.122.30): icmp_seq=1 ttl=240 time=24.4
ms
64 bytes from server-13-35-122-30.sfo5.r.cloudfront.net (13.35.122.30): icmp_seq=2 ttl=240 time=22.0
ms
64 bytes from server-13-35-122-30.sfo5.r.cloudfront.net (13.35.122.30): icmp_seq=3 ttl=240 time=21.3
ms
64 bytes from server-13-35-122-30.sfo5.r.cloudfront.net (13.35.122.30): icmp_seq=4 ttl=240 time=20.4
ms
^C
--- santabanta.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 20.444/22.050/24.395/1.465 ms
[03/05/22]seed@VM:~/Desktop$
```

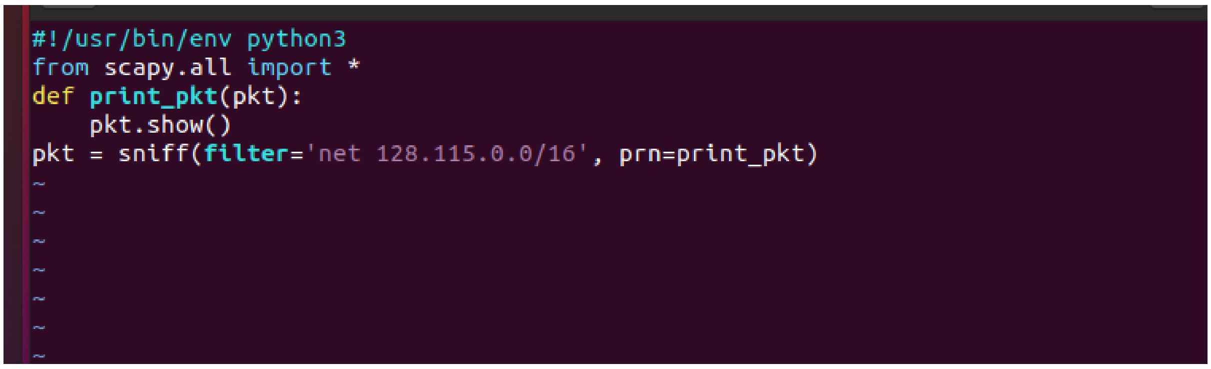
Now we will use netcat command and port 23, the packets would be sniffed by our program as the below screenshot.



```
     src       = 10.0.2.4
     dst       = 1.1.1.1
     \options   \
###[ TCP ]###
        sport     = 46508
        dport     = telnet
        seq       = 683343633
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0xe34
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (2689698236, 0)), ('NOP', None),
('WScale', 7)]

###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:68:4b:d6
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 60
     id        = 23476
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xd102
     src       = 10.0.2.4
     dst       = 1.1.1.1
     \options   \
###[ TCP ]###
        sport     = 46508
        dport     = telnet
        seq       = 683343633
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0xe34
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (2689730749, 0)), ('NOP', None),
('WScale', 7)]
```

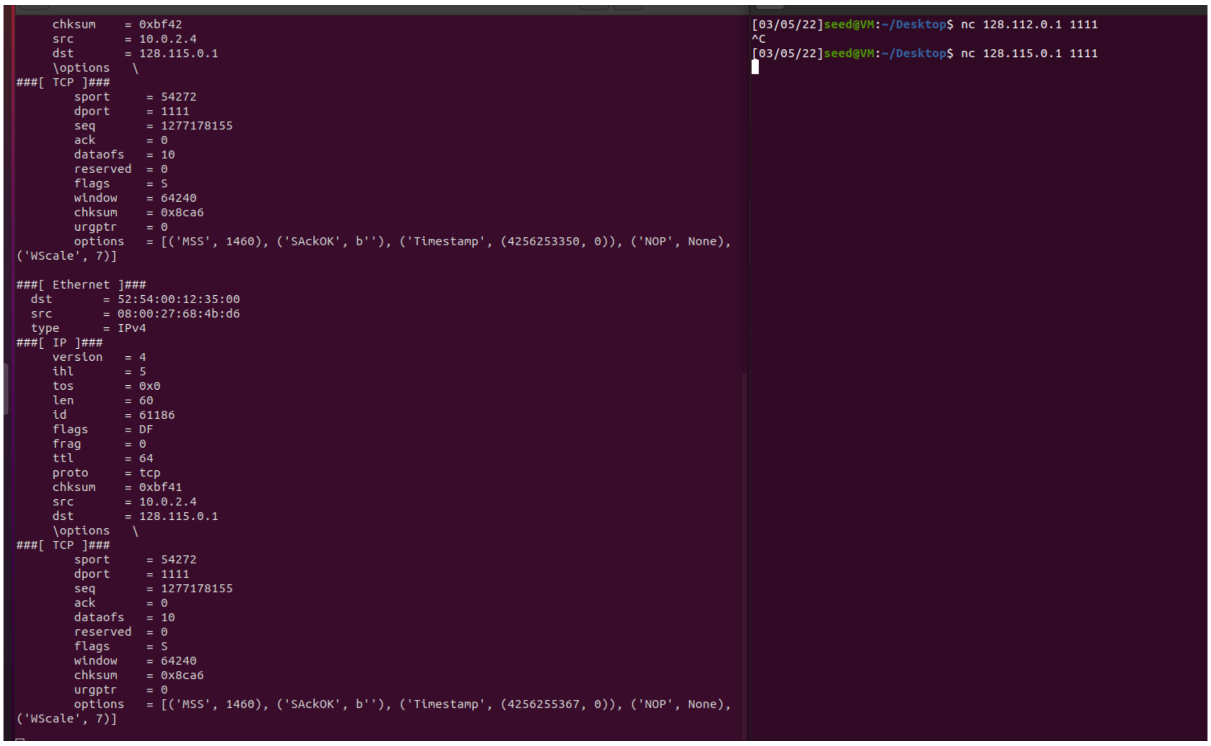
```
^C
[03/05/22]seed@VM:~/Desktop$ nc 1.1.1.1 23
```

iii) Now in our we move to our third filter. Here we need to sniff the packets coming and going to specific subnet. The modified code would be as shown in the screenshot.



```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
        pkt.show()
pkt = sniff(filter='net 128.115.0.0/16', prn=print_pkt)
~
~
~
~
~
~
~
```

Now if we will run the code and send TCP packet to IP which is not part of subnet and we observe that it won’t be cached and when we will send to IP which is part of subnet it would be caught as shown in the screenshot.

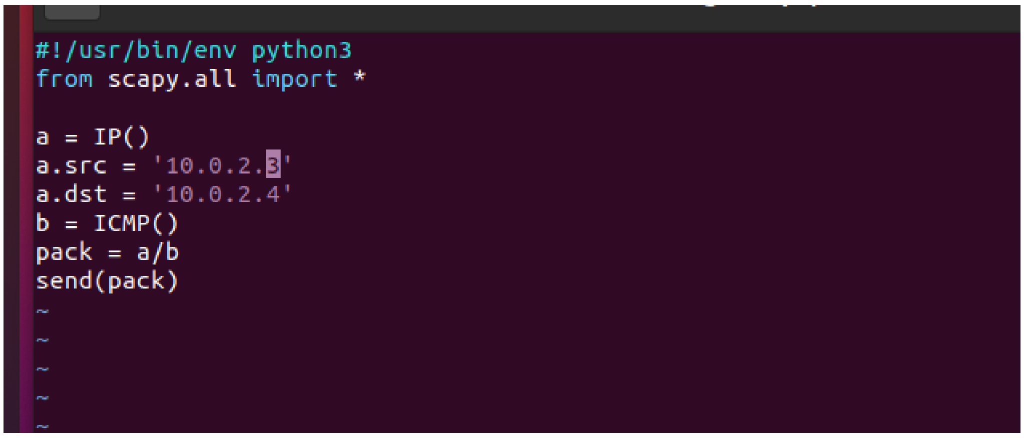


```
    chksum    = 0xbf42
    src       = 10.0.2.4
    dst       = 128.115.0.1
    \options   \
###[ TCP ]###
       sport     = 54272
       dport     = 1111
       seq       = 1277178155
       ack       = 0
       dataofs   = 10
       reserved  = 0
       flags     = S
       window    = 64240
       chksum    = 0x8ca6
       urgptr    = 0
       options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (4256253350, 0)), ('NOP', None),
('WScale', 7)]

###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:68:4b:d6
  type      = IPv4
###[ IP ]###
    version   = 4
    ihl       = 5
    tos       = 0x0
    len       = 60
    id        = 61186
    flags     = DF
    frag      = 0
    ttl       = 64
    proto     = tcp
    chksum    = 0xbf41
    src       = 10.0.2.4
    dst       = 128.115.0.1
    \options   \
###[ TCP ]###
       sport     = 54272
       dport     = 1111
       seq       = 1277178155
       ack       = 0
       dataofs   = 10
       reserved  = 0
       flags     = S
       window    = 64240
       chksum    = 0x8ca6
       urgptr    = 0
       options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (4256255367, 0)), ('NOP', None),
('WScale', 7)]
```

```
[03/05/22]seed@VM:~/Desktop$ nc 128.112.0.1 1111
^C
[03/05/22]seed@VM:~/Desktop$ nc 128.115.0.1 1111
```

1.2 Spoofing ICMP Packets

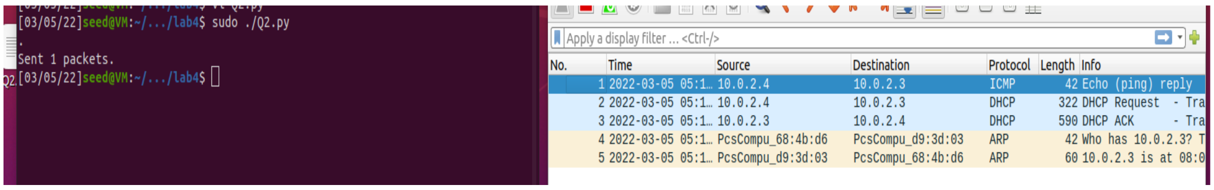
In this problem we need to write code to create spoof ICMP request packet with random source IP address and send it to particular machine in our network. Suppose our random IP address is 10.0.2.3 and destination would be my IP address 10.0.2.4. Below is the screenshot of my code using Scapy in python.



```
#!/usr/bin/env python3
from scapy.all import *

a = IP()
a.src = '10.0.2.3'
a.dst = '10.0.2.4'
b = ICMP()
pack = a/b
send(pack)
~
~
~
~
~
```

Here, we are creating an IP object and setting source and destination IP addresses. Now we will create an ICMP object. We will use overloaded operator '/' to create packet and then we will send it.

Now we will run this program in sudo privilege and check the packets in Wireshark. We can see that a request is sent from 10.0.2.3 and reply is sent back from the destination or our IP to the random/fake IP.



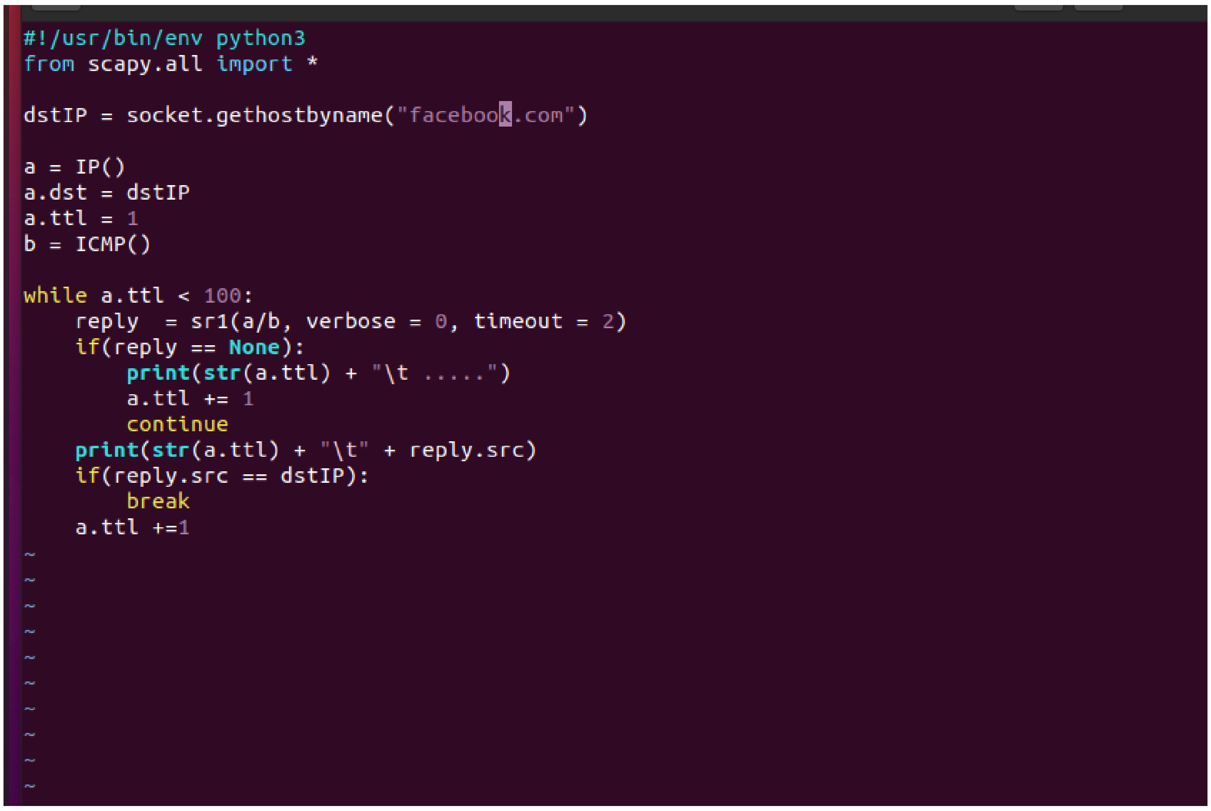
```
[03/05/22]seed@VM:~/.../lab4$ sudo ./Q2.py
.
Sent 1 packets.
[03/05/22]seed@VM:~/.../lab4$
```

```
No.  Time                Source              Destination         Protocol Length Info
   1 2022-03-05 05:1…    10.0.2.4            10.0.2.3            ICMP       42 Echo (ping) reply
   2 2022-03-05 05:1…    10.0.2.4            10.0.2.3            DHCP      322 DHCP Request  - Tra
   3 2022-03-05 05:1…    10.0.2.3            10.0.2.4            DHCP      590 DHCP ACK      - Tra
   4 2022-03-05 05:1…    PcsCompu_68:4b:d6   PcsCompu_d9:3d:03   ARP        42 Who has 10.0.2.3? T
   5 2022-03-05 05:1…    PcsCompu_d9:3d:03   PcsCompu_68:4b:d6   ARP        60 10.0.2.3 is at 08:0
```

This depicts that our program was successful in spoofing ICMP packet.

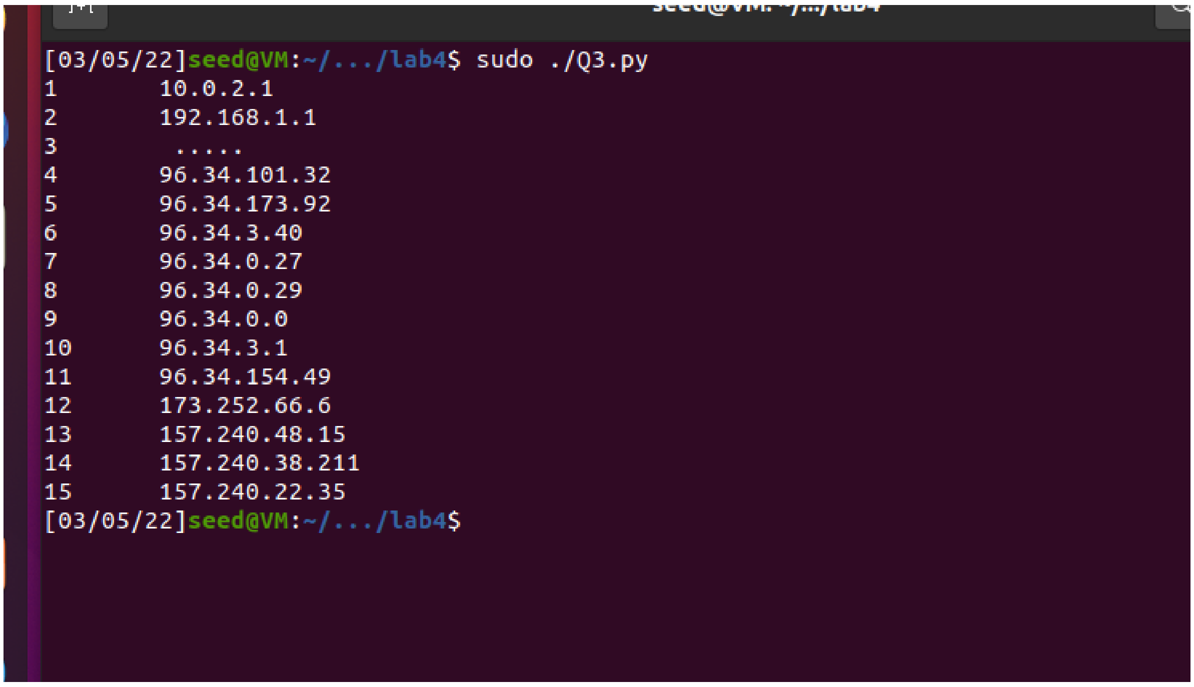
1.3 Traceroute

In this problem we need to create a traceroute send packets with changing the value of time to live or ttl. Depending on the ttl value either it will give time exceeded message or it will reach the destination. We will keep increasing the value of TTL and send packets until the destination is reached. We will be able to check this when we get back the Echo-Reply. The IP addresses would match with the address sent and reply. We need to set the maximum ttl allowed, I will set it hundred. Below is the screenshot of my code in python using Scapy.



```python
#!/usr/bin/env python3
from scapy.all import *

dstIP = socket.gethostbyname("facebook.com")

a = IP()
a.dst = dstIP
a.ttl = 1
b = ICMP()

while a.ttl < 100:
    reply  = sr1(a/b, verbose = 0, timeout = 2)
    if(reply == None):
        print(str(a.ttl) + "\t .....")
        a.ttl += 1
        continue
    print(str(a.ttl) + "\t" + reply.src)
    if(reply.src == dstIP):
        break
    a.ttl +=1
~
~
~
~
~
~
~
~
~
~
```

The code takes the IP of facebook.com and set initial ttl to 1, then in the loop it keep checking for reply and when reach the ttl value where it is able to get the reply it prints the trace it out. Now we will run program with sudo privilege. Below is the output we will get.



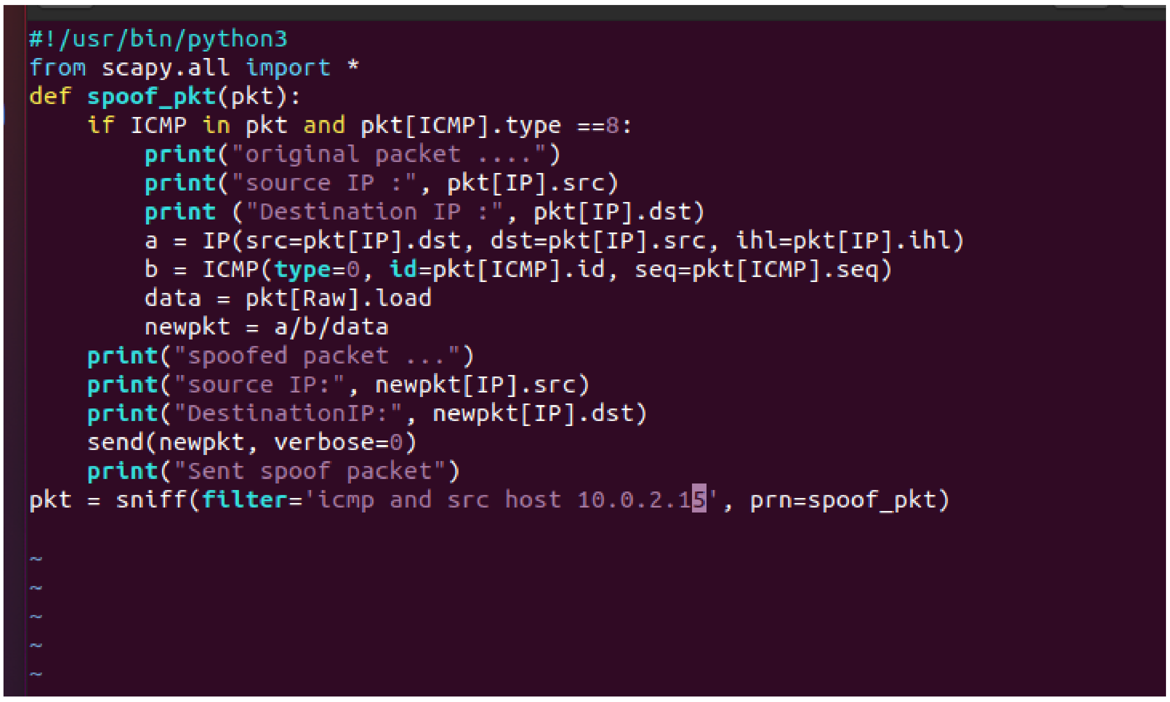
```
[03/05/22]seed@VM:~/.../lab4$ sudo ./Q3.py
1        10.0.2.1
2        192.168.1.1
3         .....
4        96.34.101.32
5        96.34.173.92
6        96.34.3.40
7        96.34.0.27
8        96.34.0.29
9        96.34.0.0
10       96.34.3.1
11       96.34.154.49
12       173.252.66.6
13       157.240.48.15
14       157.240.38.211
15       157.240.22.35
[03/05/22]seed@VM:~/.../lab4$
```

From the output we can observe that facebook.com can be reached with 15 hops, and we can see the trace of path that it took to reach the destination.

1.4 Sniffing and-then Spoofing

In this we are using 2 VMs one for attacking and another which will be attacked. Our program would sniff for ICMP for our host/ victim machine which is 10.0.2.15. We would spoof and sent reply to the victim machine and even though the machine doesn't exist the victim will get the reply and think that the machine is existing.

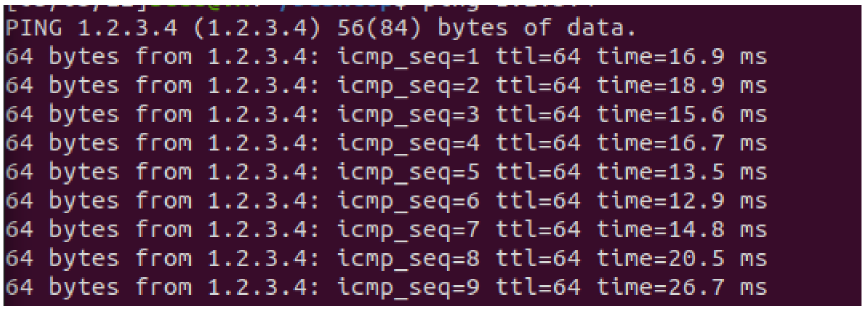
Below is the program screenshot.



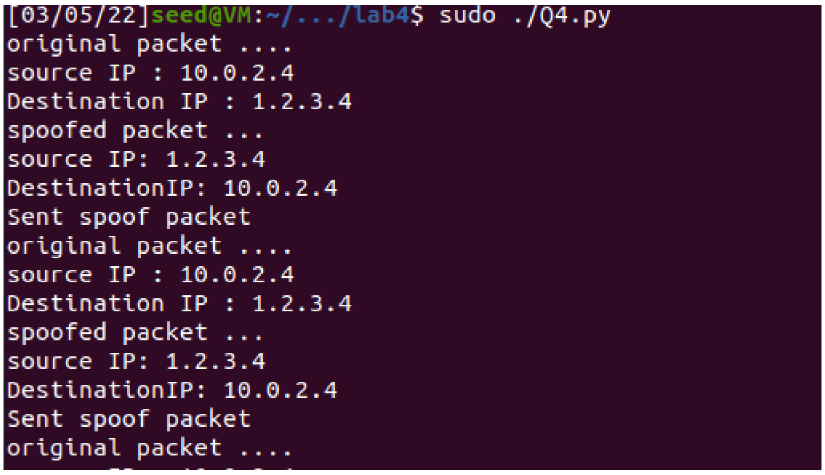
```
#!/usr/bin/python3
from scapy.all import *
def spoof_pkt(pkt):
    if ICMP in pkt and pkt[ICMP].type ==8:
        print("original packet ....")
        print("source IP :", pkt[IP].src)
        print ("Destination IP :", pkt[IP].dst)
        a = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        b = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load
        newpkt = a/b/data
    print("spoofed packet ...")
    print("source IP:", newpkt[IP].src)
    print("DestinationIP:", newpkt[IP].dst)
    send(newpkt, verbose=0)
    print("Sent spoof packet")
pkt = sniff(filter='icmp and src host 10.0.2.15', prn=spoof_pkt)

~
~
~
~
~
```

Now we will run the program with sudo privilege. We will get the following output

i) When we ping 1.2.3.4. We can see from the screenshots that the victim is able to get reply from 1.2.3.4 even when it doesn't exist.

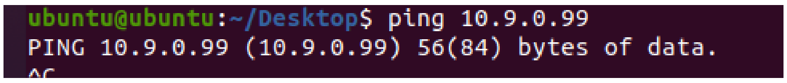


```
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=16.9 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=18.9 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=15.6 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=16.7 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=13.5 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=12.9 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=14.8 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=20.5 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=26.7 ms
```



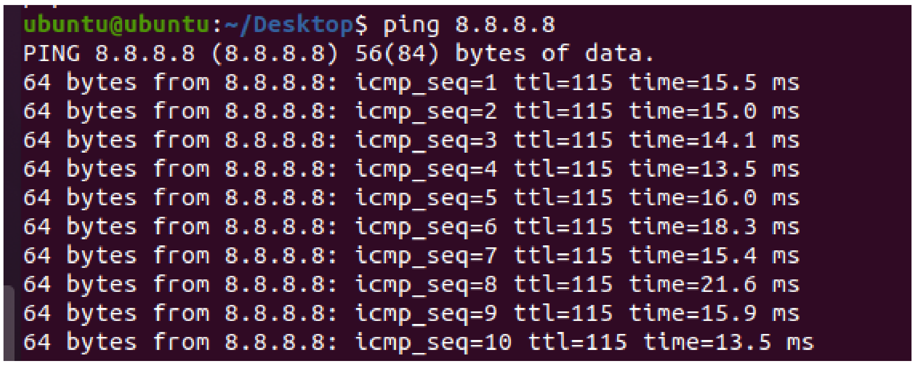
```
[03/05/22]seed@VM:~/.../lab4$ sudo ./Q4.py
original packet ....
source IP : 10.0.2.4
Destination IP : 1.2.3.4
spoofed packet ...
source IP: 1.2.3.4
DestinationIP: 10.0.2.4
Sent spoof packet
original packet ....
source IP : 10.0.2.4
Destination IP : 1.2.3.4
spoofed packet ...
source IP: 1.2.3.4
DestinationIP: 10.0.2.4
Sent spoof packet
original packet ....
```

ii)When ping 10.9.0.99, we won't get any response, as it is owned by private network and blocked by firewall.



```
ubuntu@ubuntu:~/Desktop$ ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
```

iii) When we ping 8.8.8.8, we will get duplicates responses. As this is a public DNS provided by google and it is broadcasted.



```
ubuntu@ubuntu:~/Desktop$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=15.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=15.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=14.1 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=13.5 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=16.0 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=115 time=18.3 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=115 time=15.4 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=115 time=21.6 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=115 time=15.9 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=115 time=13.5 ms
```