

## Packet Sniffing and Spoofing Lab

### Task 1.1 A: Sniffing Packets

The screenshot shows a Linux desktop environment with several windows open:

- Sublime Text (UNREGISTERED)**: The file `sniffer.py` is open, containing the following Python code:

```
#!/usr/bin/python
from scapy.all import *
def print_pkt(pkt):
    pkt.show()
    pkt = sniff(filter='icmp', prn=print_pkt)
```
- Terminal 1**: Shows the output of running the script:

```
[02/01/2019 15:22] seed@VM:~/sniffing$ [02/01/2019 15:22] seed@VM:~/sniffing$ [02/01/2019 15:22] seed@VM:~/sniffing$ sudo python sniffer.py WARNING: No route found for IPv6 destination :: (no default route?)
```

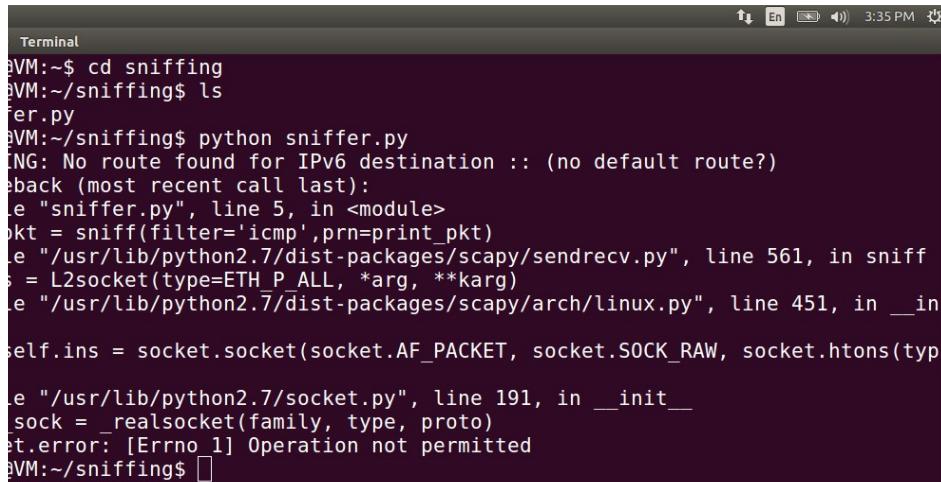
Details of the captured ICMP echo-request packet:

```
###[ Ethernet ]###
dst      = 00:00:00:00:00:00
src      = 00:00:00:00:00:00
type     = 0x800
###[ IP ]###
version  = 4L
ihl      = 5L
tos      = 0x0
len      = 84
id       = 49826
flags    = DF
frag    = 0L
ttl      = 64
proto   = icmp
chksum   = 0x5fe9
src      = 10.0.2.15
dst      = 10.0.2.15
'options' \
###[ ICMP ]###
type     = echo-request
code    = 0
checksum = 0x1cf1
id      = 0x6f88
seq     = 0x1
###[ Raw ]###
load    = '\xbcc\xaaT\\h{\x07\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11&\'()*+, -./01234567'
```
- Terminal 2**: Shows the output of a ping command to the local host:

```
VM:~/sniffing$ ping 10.0.2.15
10.0.2.15 (10.0.2.15) 56(84) bytes of data.
tes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.034 ms
tes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.093 ms
tes from 10.0.2.15: icmp_seq=3 ttl=64 time=0.149 ms

0.0.2.15 ping statistics ---
kets transmitted, 3 received, 0% packet loss, time 2009ms
in/avg/max/mdev = 0.034/0.092/0.149/0.046 ms
VM:~/sniffing$
```

Observation: In task 1.1 A we first copied the code to sniffer.py. After that we executed it. Then we pinged the IP as seen from the above screenshot. We can see that we are able to get the details of the ICMP packets and the statistics of the same. As seen from the above screen shot we need to use sudo in order to run the code

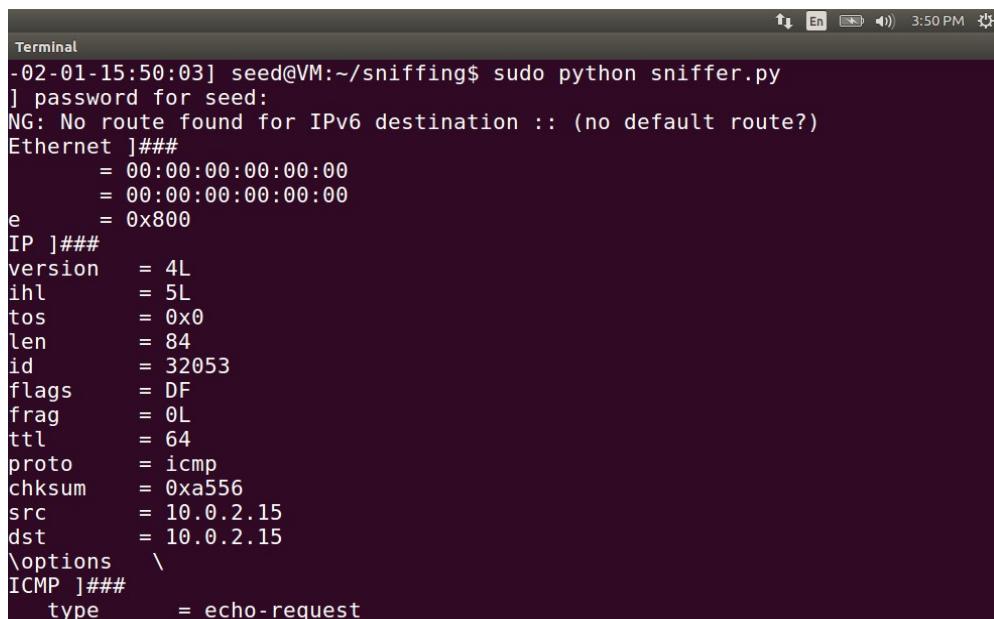


```
Terminal
VM:~$ cd sniffing
VM:~/sniffing$ ls
er.py
VM:~/sniffing$ python sniffer.py
[NG: No route found for IPv6 destination :: (no default route?)]
back (most recent call last):
  in "sniffer.py", line 5, in <module>
    pkt = sniff(filter='icmp', prn=print_pkt)
  in "/usr/lib/python2.7/dist-packages/scapy/sendrecv.py", line 561, in sniff
    s = L2socket(type=ETH_P_ALL, *arg, **karg)
  in "/usr/lib/python2.7/dist-packages/scapy/arch/linux.py", line 451, in __in
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(typ
  in "/usr/lib/python2.7/socket.py", line 191, in __init__
    sock = _realsocket(family, type, proto)
  et.error: [Errno 1] Operation not permitted
VM:~/sniffing$ 
```

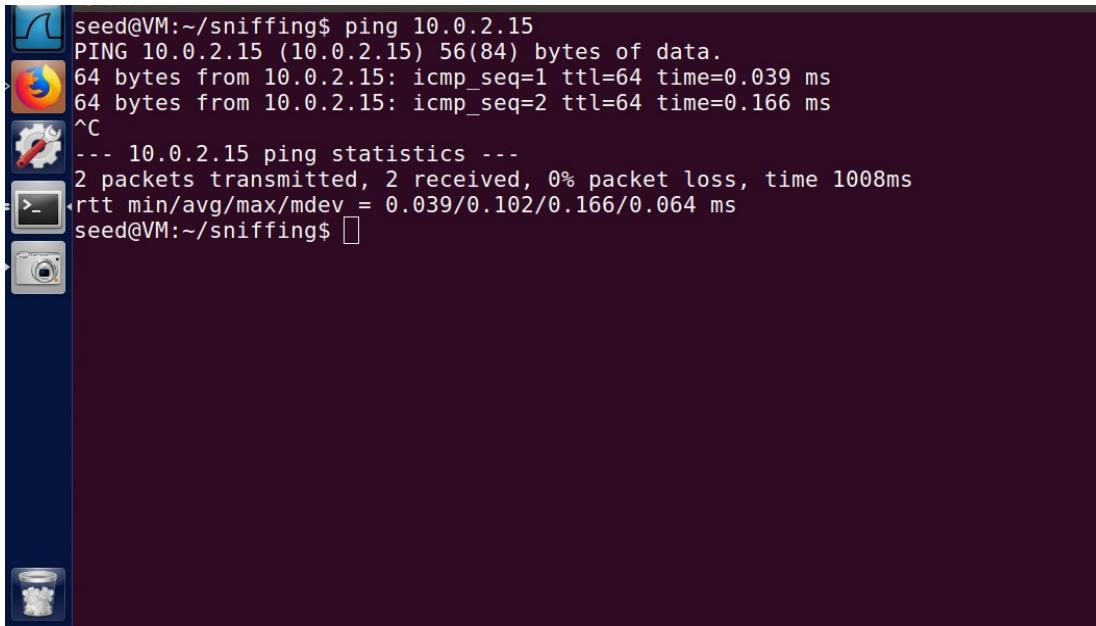
Observation: Similarly we tried to do the same but without using the root privileges and we can see that we are not allowed to execute the code as the operation is not permitted meaning that we need the root privileges to sniff the packets.

Explanation : As network interface card (NIC) accepts packets & only root can access it and the permission is given to pcap which is used in the code. Therefore, we need root privileges to execute the code.

### Task 1.1 B - Capture only the ICMP packet



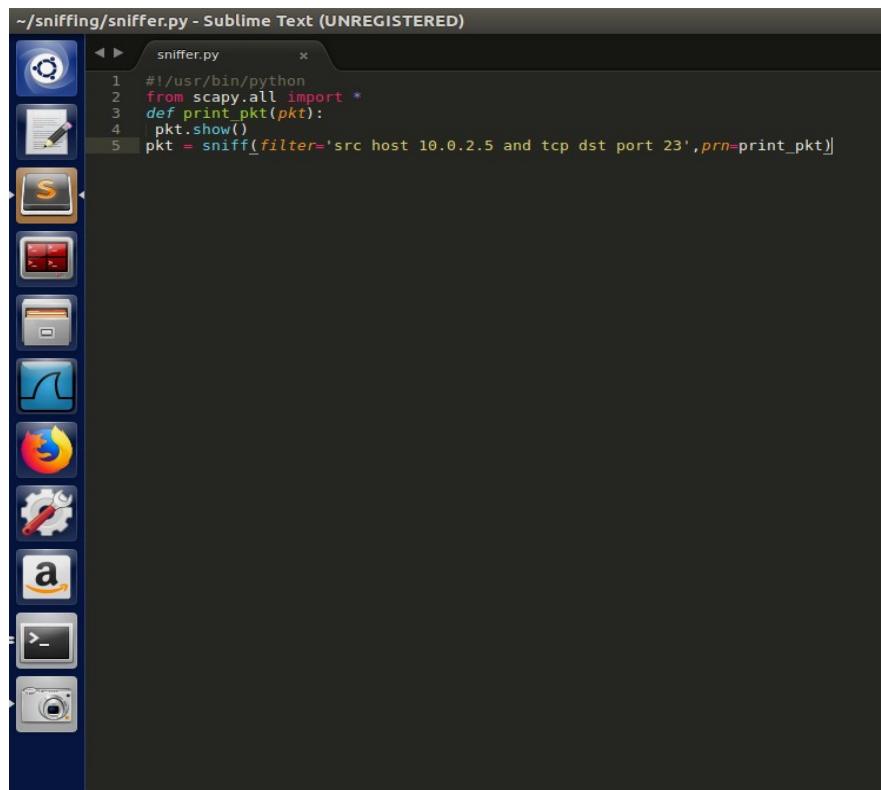
```
Terminal
[02-01-15:50:03] seed@VM:~/sniffing$ sudo python sniffer.py
] password for seed:
[NG: No route found for IPv6 destination :: (no default route?)]
Ethernet ]###
  = 00:00:00:00:00:00
  = 00:00:00:00:00:00
  = 0x800
IP ]###
  version   = 4L
  ihl      = 5L
  tos      = 0x0
  len      = 84
  id       = 32053
  flags     = DF
  frag     = 0L
  ttl      = 64
  proto    = icmp
  checksum = 0xa556
  src      = 10.0.2.15
  dst      = 10.0.2.15
  \options  \
  ICMP ]###
    type     = echo-request
```



```
seed@VM:~/sniffing$ ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.039 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.166 ms
^C
--- 10.0.2.15 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1008ms
rtt min/avg/max/mdev = 0.039/0.102/0.166/0.064 ms
seed@VM:~/sniffing$
```

This task is similar to the task 1.1 A where we captured the ICMP packet.

### **Task 1.1 B -Capture any TCP packet that comes from a particular IP and with a destination port number 23.**



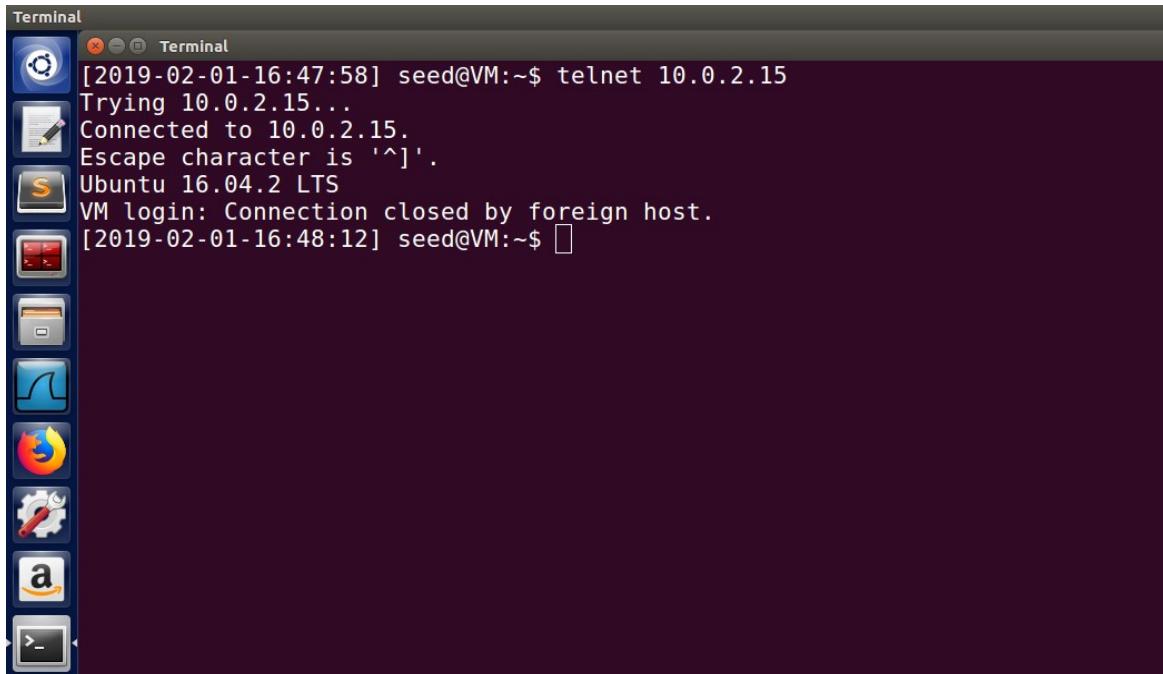
```
~/sniffing/sniffer.py - Sublime Text (UNREGISTERED)
sniffer.py
1 #!/usr/bin/python
2 from scapy.all import *
3 def print_pkt(pkt):
4     pkt.show()
5     pkt = sniff(filter='src host 10.0.2.5 and tcp dst port 23',prn=print_pkt)
```

```
Terminal
seed@VM:~/sniffing$ ifconfig
enp0s3 Link encap:Ethernet HWaddr 08:00:27:3a:58:ce
      inet addr:10.0.2.15 Bcast:10.0.2.255 Mask:255.255.255.0
      inet6 addr: fe80::ae83:9237%enp0s3 Brd:fe80::ff:fe27:3a58:ce Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:151 errors:0 dropped:0 overruns:0 frame:0
          TX packets:186 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:20985 (20.9 KB) TX bytes:18965 (18.9 KB)

lo Link encap: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:162 errors:0 dropped:0 overruns:0 frame:0
          TX packets:162 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:27439 (27.4 KB) TX bytes:27439 (27.4 KB)

seed@VM:~/sniffing$ 
```

```
Terminal
seed@VM:~/sniffing$ sudo python sniffer.py
WARNING: No route found for IPv6 destination :: (no default route?)
###[ Ethernet ]###
    dst      = 08:00:27:3a:58:ce
    src      = 08:00:27:e2:82:f3
    type     = 0x800
###[ IP ]###
    version   = 4L
    ihl       = 5L
    tos       = 0x10
    len       = 53
    id        = 17847
    flags     = DF
    frag      = 0L
    ttl       = 64
    proto     = tcp
    checksum  = 0xdce8
    src       = 10.0.2.5
    dst       = 10.0.2.15
    \options  \
###[ TCP ]###
    sport     = 39146
    dport     = telnet
    seq       = 908903226
    ack       = 1406934347
    dataofs   = 8L
    reserved  = 0L
    flags     = PA
    window    = 229
    checksum  = 0x68f4
    urgptr   = 0
    options   = [('NOP', None), ('NOP', None), ('Timestamp', (175619, 147504))]
###[ Raw ]###
    load     = '\x03'
###[ Ethernet ]###
    dst      = 08:00:27:3a:58:ce
```

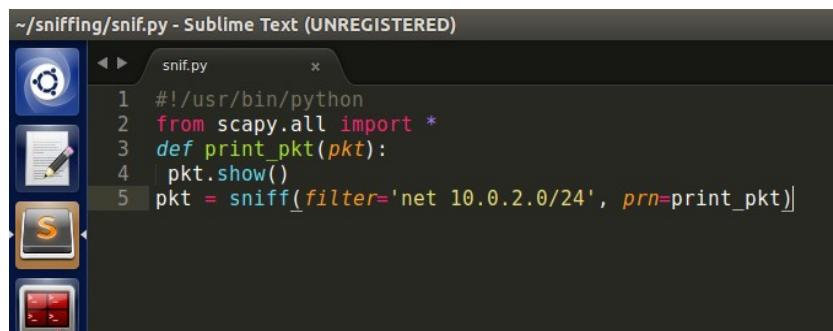


```
[Terminal]
[2019-02-01-16:47:58] seed@VM:~$ telnet 10.0.2.15
Trying 10.0.2.15...
Connected to 10.0.2.15.
Escape character is '^].
Ubuntu 16.04.2 LTS
VM login: Connection closed by foreign host.
[2019-02-01-16:48:12] seed@VM:~$
```

In this task we use the code `sniffer.py`. In that we filter the source IP to 10.0.2.5 which means that only packets from that IP will be allowed to enter. And we filter the TCP destination port to 23 meaning that packets will only be sent to port 23 of the destination IP. Once, we execute the code we sent the packets from 10.0.2.5 to 10.0.2.15 and we can see that only the TCP packets are seen and no other packets which states that all the TCP packets are being captured.

Explanation: We executed from the IP 10.0.2.5 and connected to IP 10.0.2.15 using telnet. Since, telnet uses the port 23 we can see that we get the details of the packet

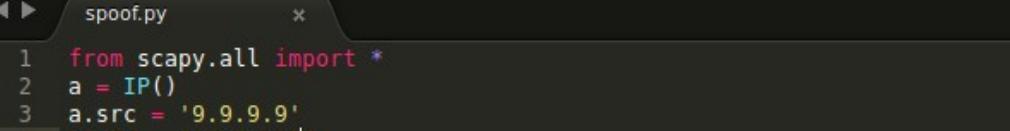
### **Task 1.1 - Capture packets comes from or to go to a particular subnet.**



```
~/sniffing/snif.py - Sublime Text (UNREGISTERED)
snif.py
1 #!/usr/bin/python
2 from scapy.all import *
3 def print_pkt(pkt):
4     pkt.show()
5     pkt = sniff(filter='net 10.0.2.0/24', prn=print_pkt)
```

Here we used the filter = 'net 10.0.2.0/24'

## Task 1.2: Spoofing ICMP Packets



The screenshot shows a Linux desktop environment with the Unity interface. A Sublime Text 3 window is open, titled "spoof.py - Sublime Text (UNREGISTERED)". The code in the editor is as follows:

```
from scapy.all import *
a = IP()
a.src = '9.9.9.9'
a.dst = '10.0.2.6'
b = ICMP()
p = a/b
send(p)
```

The screenshot shows a Linux desktop environment with a dark theme. A terminal window titled "Terminal" is open, displaying the following command-line session:

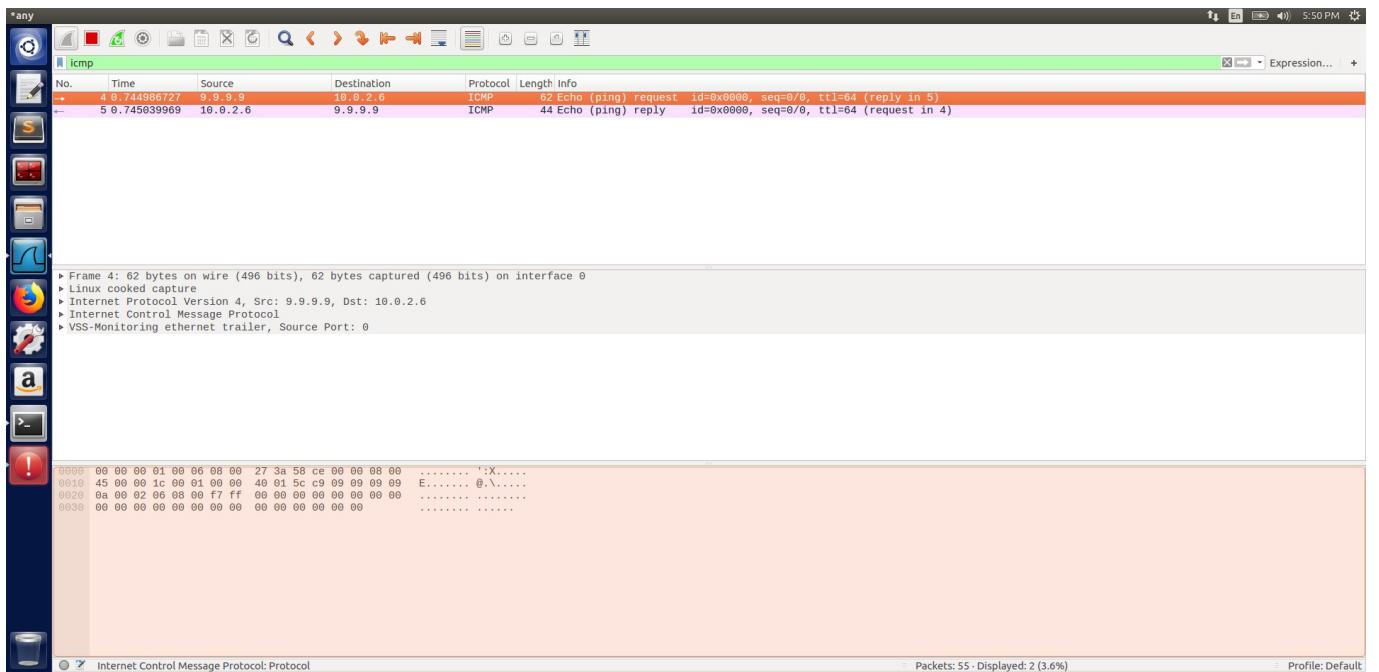
```
[2019-02-01-17:58:22] seed@VM:~/sniffing$ sudo python spoof.py
WARNING: No route found for IPv6 destination :: (no default route?)
.
Sent 1 packets.
[2019-02-01-17:58:24] seed@VM:~/sniffing$
```

The desktop interface includes a vertical dock on the left containing icons for various applications such as a terminal, file manager, browser, and system tools.

```
[2019-02-01-17:46:56] seed@VM:~$ ifconfig
enp0s3      Link encap:Ethernet HWaddr 08:00:27:ae:fd:1e
             inet addr:10.0.2.6 Bcast:10.0.2.255 Mask:255.255.255.0
             inet6 addr: fe80::1df9:d350:6c7e:1b0e/64 Scope:Link
                   UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
                   RX packets:27 errors:0 dropped:0 overruns:0 frame:0
                   TX packets:90 errors:0 dropped:0 overruns:0 carrier:0
                   collisions:0 txqueuelen:1000
                   RX bytes:5134 (5.1 KB)  TX bytes:9866 (9.8 KB)

lo          Link encap: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:180 errors:0 dropped:0 overruns:0 frame:0
                   TX packets:180 errors:0 dropped:0 overruns:0 carrier:0
                   collisions:0 txqueuelen:1
                   RX bytes:27053 (27.0 KB)  TX bytes:27053 (27.0 KB)

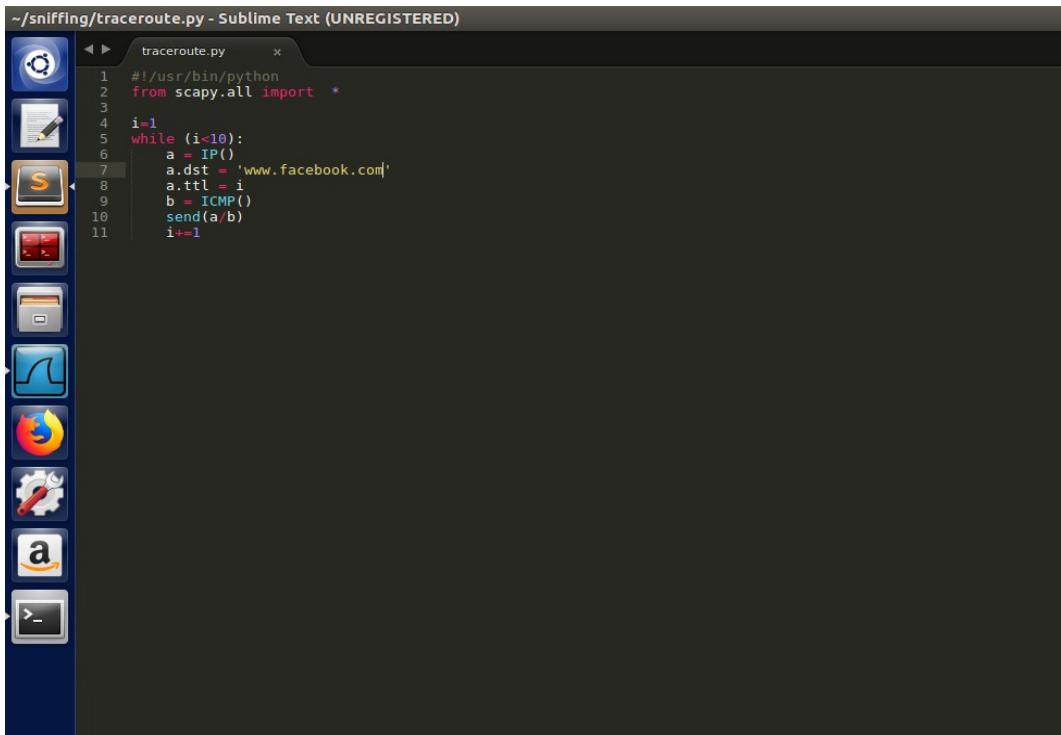
[2019-02-01-17:48:39] seed@VM:~$ 
```



**Observation -** In this task we keep the source ip as 9.9.9.9 and destination ip as one of the VM's ip that is 10.0.2.6. After that we execute the code and the destination ip assumes that the packets are coming from the source ip 9.9.9.9.

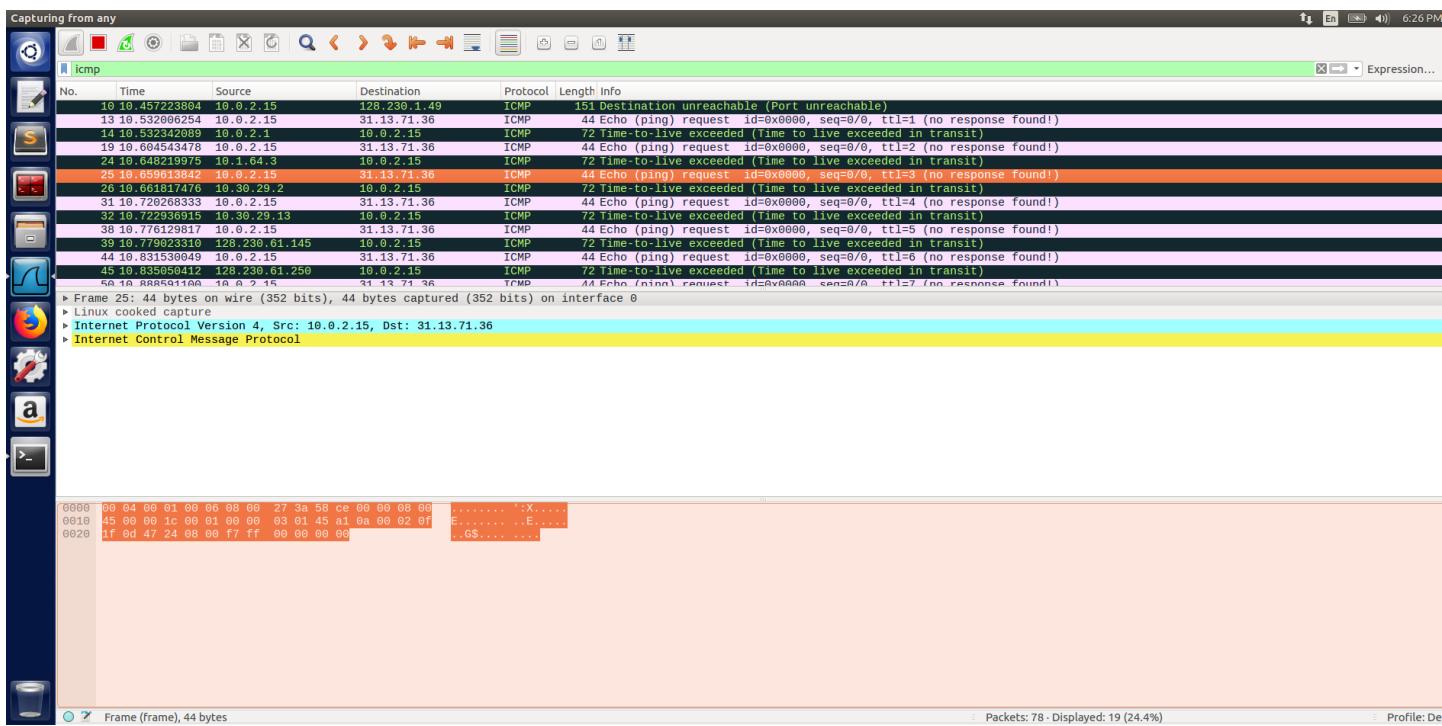
**Explanation-** We can see on wireshark that we are able to read both the packets as above. We are using arbitrary source ip and keeping one of our VM as destination ip – 10.0.2.6. Wireshark is able to capture the spoofed packets as seen from the screenshot.

## Task 1.3: Traceroute



A screenshot of the Sublime Text editor showing a Python script named `traceroute.py`. The code uses the `scapy.all` module to send ICMP echo requests to `'www.facebook.com'` with increasing TTL values. The script is as follows:

```
#!/usr/bin/python
from scapy.all import *
i=1
while (i<10):
    a = IP()
    a.dst = 'www.facebook.com'
    a.ttl = i
    b = ICMP()
    send(a/b)
    i+=1
```



A screenshot of Wireshark capturing ICMP traffic. The packet list shows multiple ICMP echo requests (Type 8) and echo replies (Type 0) between the source IP `10.0.2.15` and destination IP `31.13.71.36`. Many of the requests result in "Destination unreachable (Port unreachable)" errors, indicating routers along the path are dropping the packets. The details and bytes panes show the structure of the captured ICMP frames.

Capturing from any

icmp

| No. | Time         | Source         | Destination  | Protocol | Length | Info   |
|-----|--------------|----------------|--------------|----------|--------|--|
| 10  | 18.457223884 | 10.0.2.15      | 128.230.1.49 | ICMP     | 151    | Destination unreachable (Port unreachable)                         |
| 13  | 18.532986254 | 10.0.2.15      | 31.13.71.36  | ICMP     | 44     | Echo (ping) request id=0x0000, seq=0/0, ttl=1 (no response found!) |
| 14  | 18.532342089 | 10.0.2.1       | 10.0.2.15    | ICMP     | 72     | Time-to-live exceeded (Time to live exceeded in transit)           |
| 19  | 18.604543478 | 10.0.2.15      | 31.13.71.36  | ICMP     | 44     | Echo (ping) request id=0x0000, seq=0/0, ttl=2 (no response found!) |
| 24  | 18.648219975 | 10.1.64.3      | 10.0.2.15    | ICMP     | 72     | Time-to-live exceeded (Time to live exceeded in transit)           |
| 25  | 18.659613842 | 10.0.2.15      | 31.13.71.36  | ICMP     | 44     | Echo (ping) request id=0x0000, seq=0/0, ttl=3 (no response found!) |
| 26  | 18.661817476 | 10.30.29.2     | 10.0.2.15    | ICMP     | 72     | Time-to-live exceeded (Time to live exceeded in transit)           |
| 31  | 18.720268333 | 10.0.2.15      | 31.13.71.36  | ICMP     | 44     | Echo (ping) request id=0x0000, seq=0/0, ttl=4 (no response found!) |
| 32  | 18.722936915 | 10.30.29.13    | 10.0.2.15    | ICMP     | 72     | Time-to-live exceeded (Time to live exceeded in transit)           |
| 38  | 18.776129817 | 10.0.2.15      | 31.13.71.36  | ICMP     | 44     | Echo (ping) request id=0x0000, seq=0/0, ttl=5 (no response found!) |
| 39  | 18.779023316 | 128.230.61.145 | 10.0.2.15    | ICMP     | 72     | Time-to-live exceeded (Time to live exceeded in transit)           |
| 44  | 18.831530049 | 10.0.2.15      | 31.13.71.36  | ICMP     | 44     | Echo (ping) request id=0x0000, seq=0/0, ttl=6 (no response found!) |
| 45  | 18.835050412 | 128.230.61.256 | 10.0.2.15    | ICMP     | 72     | Time-to-live exceeded (Time to live exceeded in transit)           |
| 50  | 18.888501160 | 10.0.2.15      | 31.13.71.36  | ICMP     | 44     | Echo (ping) request id=0x0000, seq=0/0, ttl=7 (no response found!) |

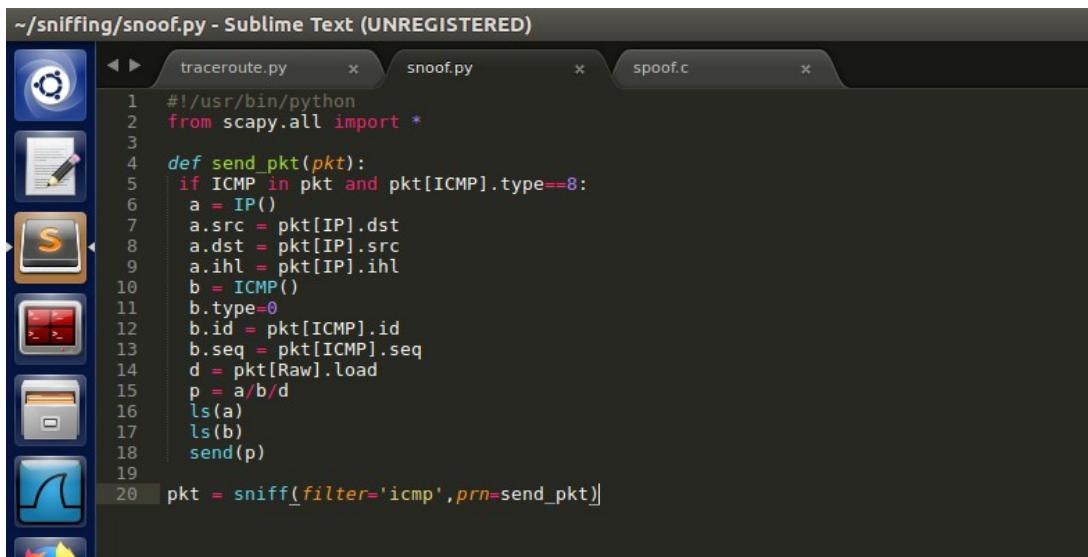
Packets: 78 - Displayed: 19 (24.4%)

Profile: Default

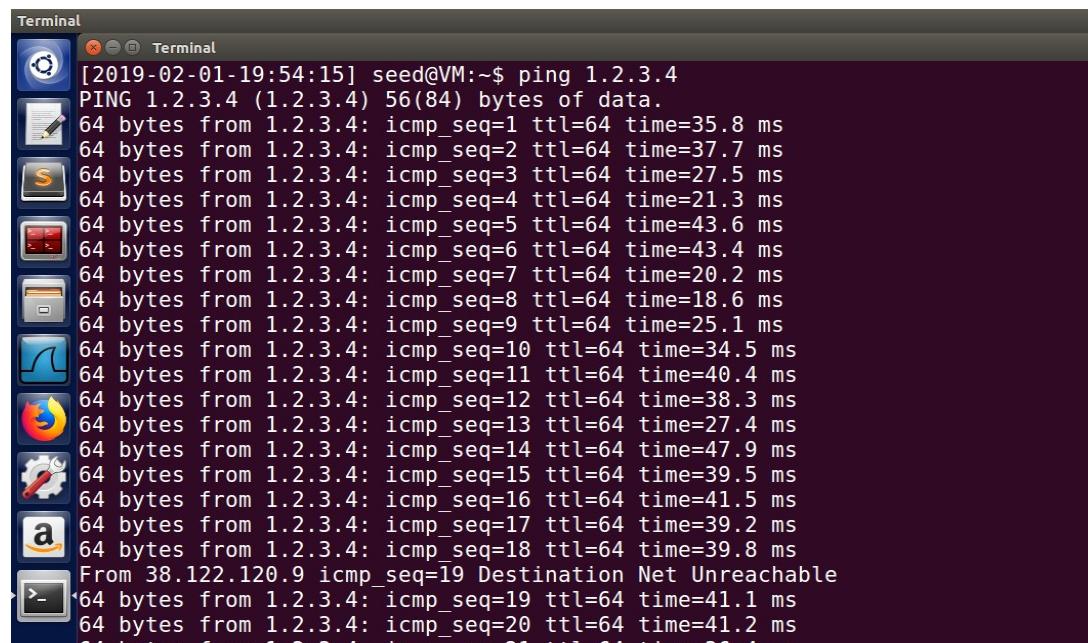
**Observation** – In this task we compile and execute the code traceroute.py and are pinging to ‘www.facebook.com’ and increase our time to live from 1 to 10.

**Explanation** - In the wireshark screenshot we can see that if the time to live is short then the packet does not reach the destination and will be dropped by the first router and will send us an ICMP error that will tell us that time to live has exceeded and we will get the IP address of the first router and will keep repeating until the packet reaches the destination. We are able to see numerous IP indicating the hops that are occurring.

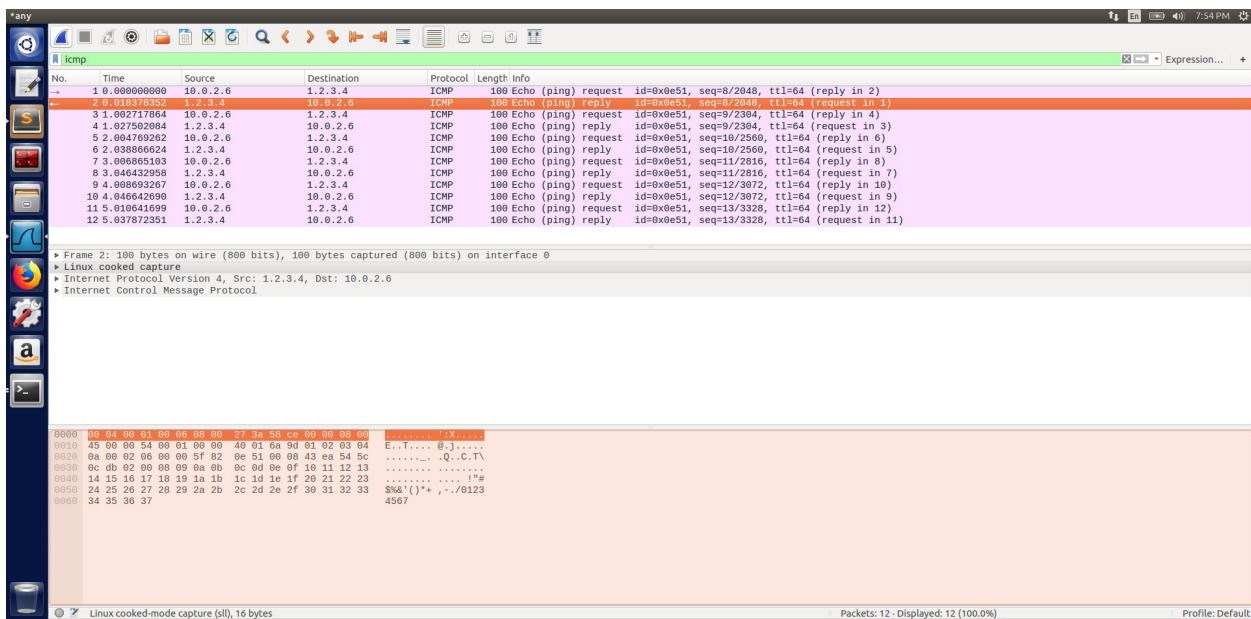
### Task 1.4: Sniffing and-then Spoofing



```
~/sniffing/snoof.py - Sublime Text (UNREGISTERED)
traceroute.py      snoof.py      spoof.c
1  #!/usr/bin/python
2  from scapy.all import *
3
4  def send_pkt(pkt):
5      if ICMP in pkt and pkt[ICMP].type==8:
6          a = IP()
7          a.src = pkt[IP].dst
8          a.dst = pkt[IP].src
9          a.ihl = pkt[IP].ihl
10         b = ICMP()
11         b.type=0
12         b.id = pkt[ICMP].id
13         b.seq = pkt[ICMP].seq
14         d = pkt[Raw].load
15         p = a/b/d
16         ls(a)
17         ls(b)
18         send(p)
19
20     pkt = sniff(filter='icmp',prn=send_pkt)
```



```
[2019-02-01-19:54:15] seed@VM:~$ ping 1.2.3.4
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=35.8 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=37.7 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=27.5 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=21.3 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=43.6 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=43.4 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=20.2 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=18.6 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=25.1 ms
64 bytes from 1.2.3.4: icmp_seq=10 ttl=64 time=34.5 ms
64 bytes from 1.2.3.4: icmp_seq=11 ttl=64 time=40.4 ms
64 bytes from 1.2.3.4: icmp_seq=12 ttl=64 time=38.3 ms
64 bytes from 1.2.3.4: icmp_seq=13 ttl=64 time=27.4 ms
64 bytes from 1.2.3.4: icmp_seq=14 ttl=64 time=47.9 ms
64 bytes from 1.2.3.4: icmp_seq=15 ttl=64 time=39.5 ms
64 bytes from 1.2.3.4: icmp_seq=16 ttl=64 time=41.5 ms
64 bytes from 1.2.3.4: icmp_seq=17 ttl=64 time=39.2 ms
64 bytes from 1.2.3.4: icmp_seq=18 ttl=64 time=39.8 ms
From 38.122.120.9 icmp_seq=19 Destination Net Unreachable
64 bytes from 1.2.3.4: icmp_seq=19 ttl=64 time=41.1 ms
64 bytes from 1.2.3.4: icmp_seq=20 ttl=64 time=41.2 ms
```



**Observation -** In this task we compile and execute the snoof.py code. We are pinging any arbitrary ip - 1.2.3.4.

**Explanation-** Even after that ip not being a valid one it is getting a response from our IP that is 10.0.2.6. Thus, it indicates that whether the destination machine is alive or not our machine sends out the echo reply using the packet spoofing technique.

## Task 2.1 A: Writing Packet Sniffing Program

```
~/sniffing/sniff.c - Sublime Text (UNREGISTERED)
 1  ****
 2  * Listing 12.3: Packet Capturing using raw libpcap
 3  ****
 4
 5  #include <pcap.h>
 6  #include <stdio.h>
 7
 8  void got_packet(u_char *args, const struct pcap_pkthdr *header,
 9  |     const u_char *packet)
10 {
11     printf("Got a packet\n");
12 }
13
14 int main()
15 {
16     pcap_t *handle;
17     char errbuf[PCAP_ERRBUF_SIZE];
18     struct bpf_program fp;
19     char filter_exp[] = "ip proto icmp";
20     bpf_u_int32 net;
21
22     // Step 1: Open live pcap session on NIC with name eth3
23     handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
24
25     // Step 2: Compile filter_exp into BPF pseudo-code
26     pcap_compile(handle, &fp, filter_exp, 0, net);
27     pcap_setfilter(handle, &fp);
28
29     // Step 3: Capture packets
30     pcap_loop(handle, -1, got_packet, NULL);
31
32     pcap_close(handle); //Close the handle
33
34     return 0;
35 }
```

**Question 1 : Please use your own words to describe the sequence of the library calls that are essential for sniffer programs**

- (a) Setting up device : We first have to set up the device which pcap does by itself. If there are any errors then it is saved in the errbuf.
  - (b) Open sniffing : after the previous setup we open the device for the sniffing using the command - pcap\_open\_live()
  - (c) Traffic Filter : pcap\_compile() is used to compile the filter and pcap\_setfilter() is used to tell what the program will sniff.
  - (d) Sniffing : We capture packets when it arrives at the NIC.
  - (e) Close Sniffing : Once all the previous task are done we close the session.

**Question 2. Why do you need the root privilege to run a sniffer program? Where does the program fail if it is executed without the root privilege?**

Root permissions are needed in order to use pcap since it can access network interface card that accepts packets that only root can access. The pcap lookupdev() will fail if the root privileges is not present.

**Question 3. Please turn on and turn off the promiscuous mode in your sniffer program. Can you demonstrate the difference when this mode is on and off? Please describe how you can demonstrate this**

## Promiscuous Mode

~/sniffing/sniff.c - Sublime Text (UNREGISTERED)

```
1 //*****
2 * Listing 12.3: Packet Capturing using raw libpcap
3 *****/
4
5 #include <pcap.h>
6 #include <stdio.h>
7
8 void got_packet(u_char *args, const struct pcap_pkthdr *header,
9                 const u_char *packet)
10 {
11     printf("Got a packet\n");
12 }
13
14
15 int main()
16 {
17     pcap_t *handle;
18     char errbuf[PCAP_ERRBUF_SIZE];
19     struct bpf_program fp;
20     char filter_exp[] = "ip proto icmp";
21     bpf_u_int32 net;
22
23     // Step 1: Open live pcap session on NIC with name eth3
24     handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
25
26     // Step 2: Compile filter exp into BPF pseudo-code
27     pcap_compile(handle, &fp, filter_exp, 0, net);
28     pcap_setfilter(handle, &fp);
29
30     // Step 3: Capture packets
31     pcap_loop(handle, -1, got_packet, NULL);
32
33     pcap_close(handle); //Close the handle
34     return 0;
35 }
36
37
38
```

Line 24, Column 46

Terminal

```
10.5 ms
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_seq=7 ttl=52 time=
13.1 ms
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_seq=8 ttl=52 time=
SublimeText
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_seq=9 ttl=52 time=
10.8 ms
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_seq=10 ttl=52 time=
15.5 ms
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_seq=11 ttl=52 time=
10.7 ms
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_seq=seed@VM:~/sniffing$ ifconfig
=11.1 ms
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_
=8.85 ms
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_
=10.9 ms
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_
=18.4 ms
^C
--- gmail.com ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 1402ms
rtt min/avg/max/mdev = 8.850/12.128/18.432/2.705 ms
seed@VM:~/sniffing$ 
```

seed@VM:~/sniffing\$ ifconfig

```
Link encap:Ethernet HWaddr 08:00:27:ae:fd:1e
inet addr:10.0.2.6 Bcast:10.0.2.255 Mask:255.255.255.0
inet6 addr: fe80::1d9:350:6c7e:1b0e/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:9509 errors:0 dropped:0 overruns:0 frame:0
TX packets:3258 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:7186489 (7.1 MB) TX bytes:332484 (332.4 KB)

Link encap: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:1818 errors:0 dropped:0 overruns:0 frame:0
TX packets:1818 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:163929 (163.9 KB) TX bytes:163929 (163.9 KB)
```

seed@VM:~/sniffing\$

When we turn on the promiscuous mode by changing the value to 1 and then we compile and execute the code after that if we ping `gmail.com` we can see that the other VM from where we ran the code is able to capture the packets.

### **Non - Promiscuous Mode**

-sniffing/sniff.c - Sublime Text (UNREGISTERED)

```
1 //*****  
2 /* Listing 12.3: Packet Capturing using raw libpcap  
3 *****/  
4  
5 #include <pcap.h>  
6 #include <stdio.h>  
7  
8 void got_packet(u_char *args, const struct pcap_pkthdr *header,  
9                  const u_char *packet)  
10 {  
11     printf("Got a packet\n");  
12 }  
13  
14 int main()  
15 {  
16     pcap_t *handle;  
17     char errbuf[PCAP_ERRBUF_SIZE];  
18     struct bpf_program fp;  
19     char filter_exp[] = "ip proto icmp";  
20     bpf_u_int32 net;  
21  
22     // Step 1: Open live pcap session on NIC with name eth3  
23     handle = pcap_open_live("enp0s3", BUFSIZ, 0, 1000, errbuf);  
24  
25     // Step 2: Compile filter_exp into BPF pseudo-code  
26     pcap_compile(handle, &fp, filter_exp, 0, net);  
27     pcap_setfilter(handle, &fp);  
28  
29     // Step 3: Capture packets  
30     pcap_loop(handle, -1, got_packet, NULL);  
31  
32     pcap_close(handle); //Close the handle  
33     return 0;  
34 }  
35  
36  
37  
38
```

```

Terminal
seed@VM:~/sniffing$ ping gmail.com
PING gmail.com (172.217.10.133) 56(84) bytes of data.
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_seq=1 ttl=52 time=8.93 ms
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_seq=2 ttl=52 time=9.07 ms
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_seq=3 ttl=52 time=8.81 ms
64 bytes from lga34s16-in-f5.1e100.net (172.217.10.133): icmp_seq=4 ttl=52 time=10.3 ms
^C
--- gmail.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3016ms
rtt min/avg/max/mdev = 8.816/9.432/10.308/0.606 ms
seed@VM:~/sniffing$ 

ifconfig
p:Ethernet HWaddr 08:00:27:ae:fd:1e
:10.0.2.6 Bcast:10.0.2.255 Mask:255.255.255.0
r: fe80::1d9:6c7e:1b0e/64 Scope:Link
    MTU:1500 Metric:1
    RX packets:9509 errors:0 dropped:0 overruns:0 frame:0
    TX packets:3258 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:1000
    7186489 (7.1 MB) TX bytes:332484 (332.4 KB)

p:Local Loopback
:127.0.0.1 Mask:255.0.0.0
inet brd ::1/128 Scope:Host
    UP LOOPBACK RUNNING MTU:65536 Metric:1
    RX packets:1818 errors:0 dropped:0 overruns:0 frame:0
    TX packets:1818 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:1
    RX bytes:163929 (163.9 KB) TX bytes:163929 (163.9 KB)

seed@VM:~/sniffing$ 

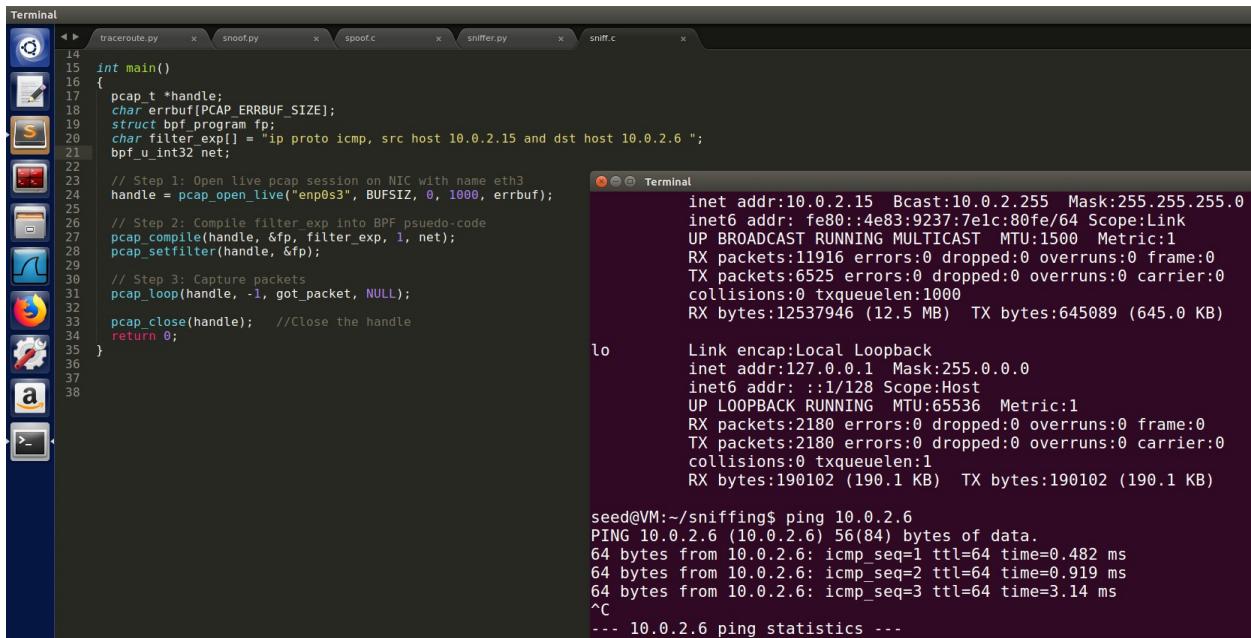
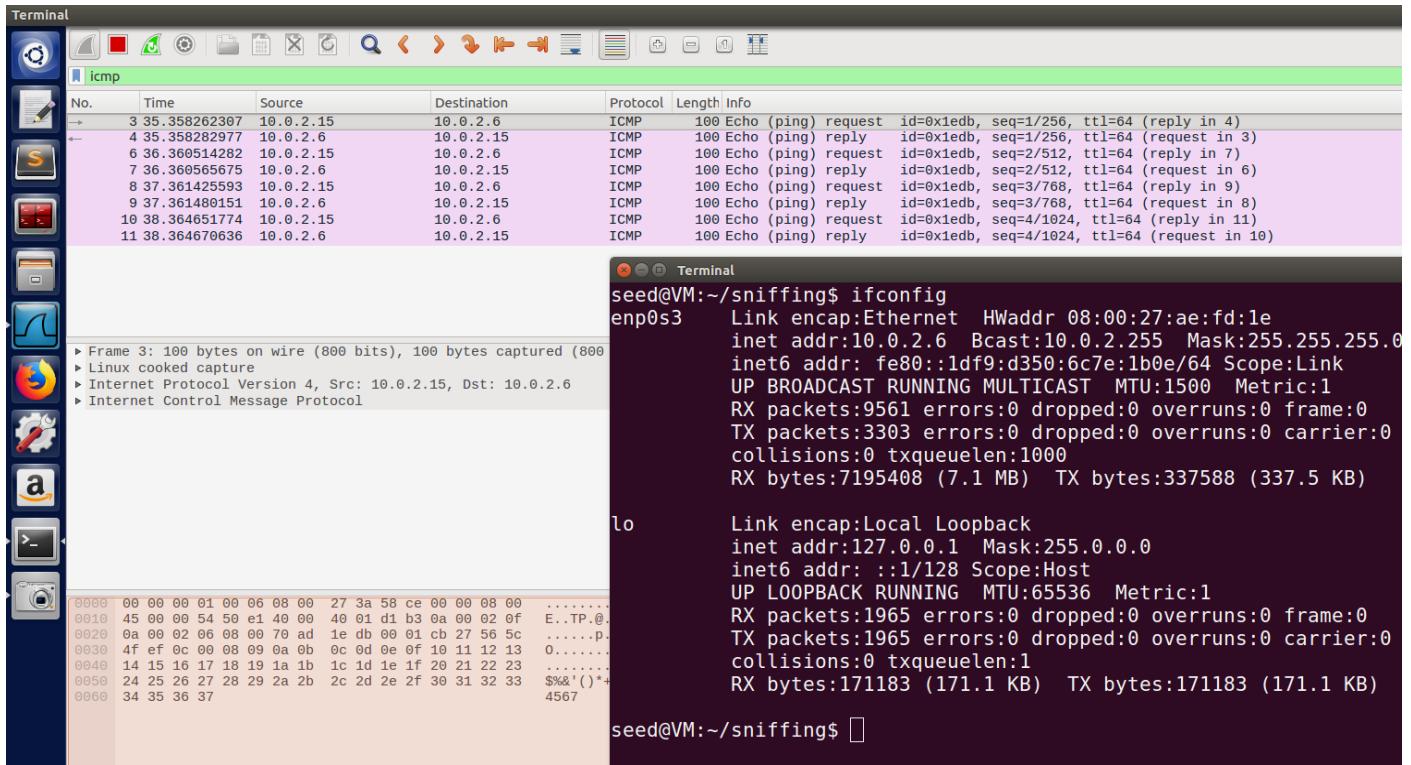
Terminal
traceroute.py x snoof.py x spoofer.py x sniffer.py x sniff.c x untitled x
14
15 int main()
16 {
17     pcap_t *handle;
18     char errbuf[PCAP_ERRBUF_SIZE];
19     struct bpf_program fp;
20     char filter_exp[] = "ip proto tcp";
21     bpf_u_int32 net;
22
23     // Step 1: Open live pcap session on NIC with name eth3
24     handle = pcap_open_live("enp0s3", BUFSIZ, 0, 1000, errbuf);
25
26     // Step 2: Compile filter_exp into BPF pseudo-code
27     pcap_compile(handle, &fp, filter_exp, 1, net);
28     pcap_setfilter(handle, &fp);
29
30     // Step 3: Capture packets
31     pcap_loop(handle, -1, got_packet, NULL);
32
33     pcap_close(handle); //Close the handle
34     return 0;
35
36
37
38
Line 24, Column 46

```

When we turn off the promiscuous mode by changing the value to 0 and then we compile and execute the code after that if we ping gmail.com we can see that the other VM from where we ran the code is not able to capture the packets.

## Task 2.1B: Writing Filters

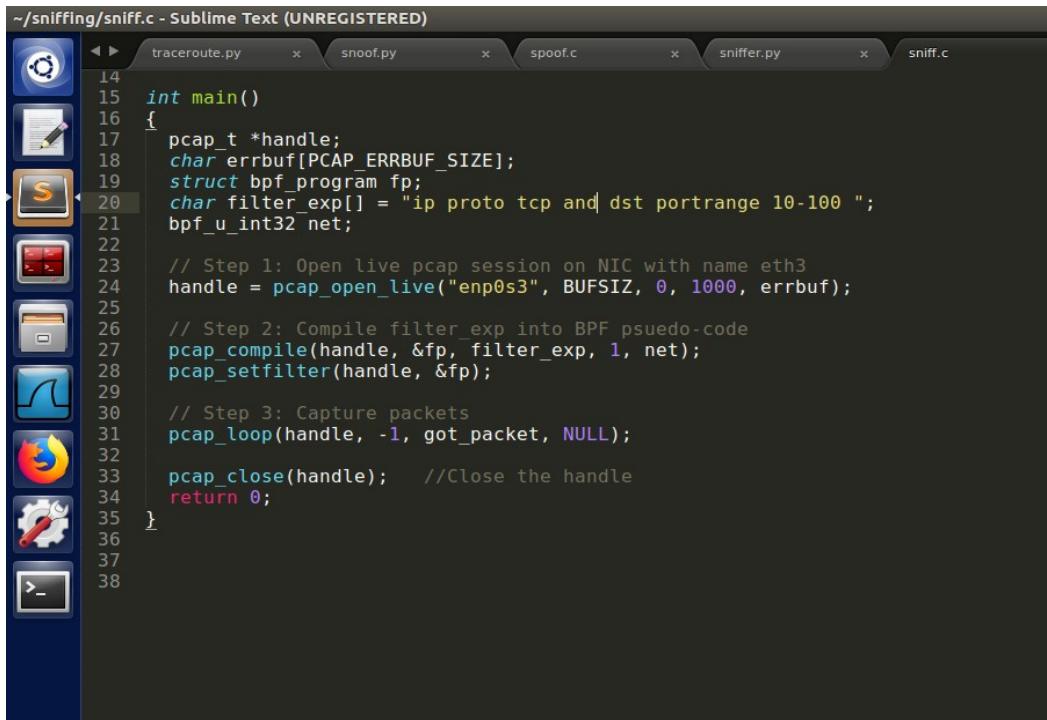
**Capture the ICMP packets between two specific hosts.**



**Observation :** In the code above we change the filter to ' ip proto icmp, src host 10.0.2.15 and dst host 10.0.2.6'

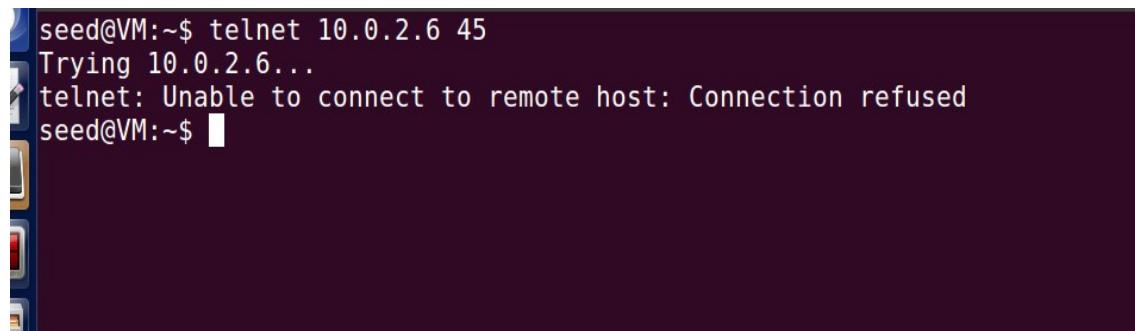
**Explanation :** The filter that we changed means that only packets that are going from 10.0.2.15 and having its destination 10.0.2.6 are captured which can be seen in the wireshark.

**Capture the TCP packets with a destination port number in the range from 10 to 100.**



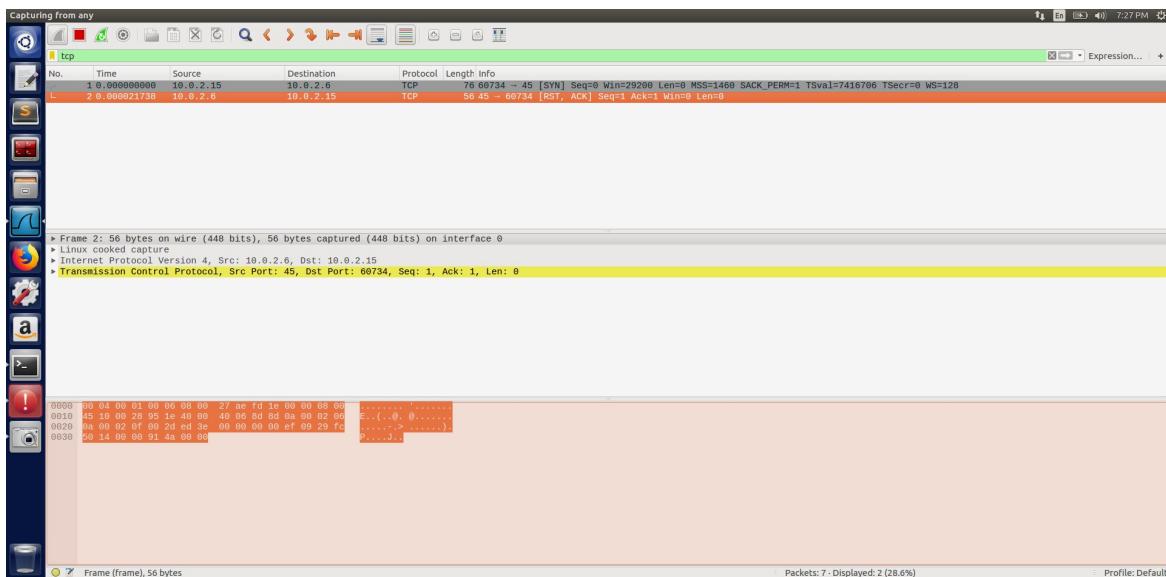
The screenshot shows a Sublime Text window with multiple tabs open. The active tab is titled 'sniff.c' and contains the following C code:

```
~/sniffing/sniff.c - Sublime Text (UNREGISTERED)
14
15 int main()
16 {
17     pcap_t *handle;
18     char errbuf[PCAP_ERRBUF_SIZE];
19     struct bpf_program fp;
20     char filter_exp[] = "ip proto tcp and dst portrange 10-100 ";
21     bpf_u_int32 net;
22
23     // Step 1: Open live pcap session on NIC with name eth3
24     handle = pcap_open_live("enp0s3", BUFSIZ, 0, 1000, errbuf);
25
26     // Step 2: Compile filter_exp into BPF psuedo-code
27     pcap_compile(handle, &fp, filter_exp, 1, net);
28     pcap_setfilter(handle, &fp);
29
30     // Step 3: Capture packets
31     pcap_loop(handle, -1, got_packet, NULL);
32
33     pcap_close(handle);    //Close the handle
34
35     return 0;
36
37
38 }
```



The screenshot shows a terminal window with the following output:

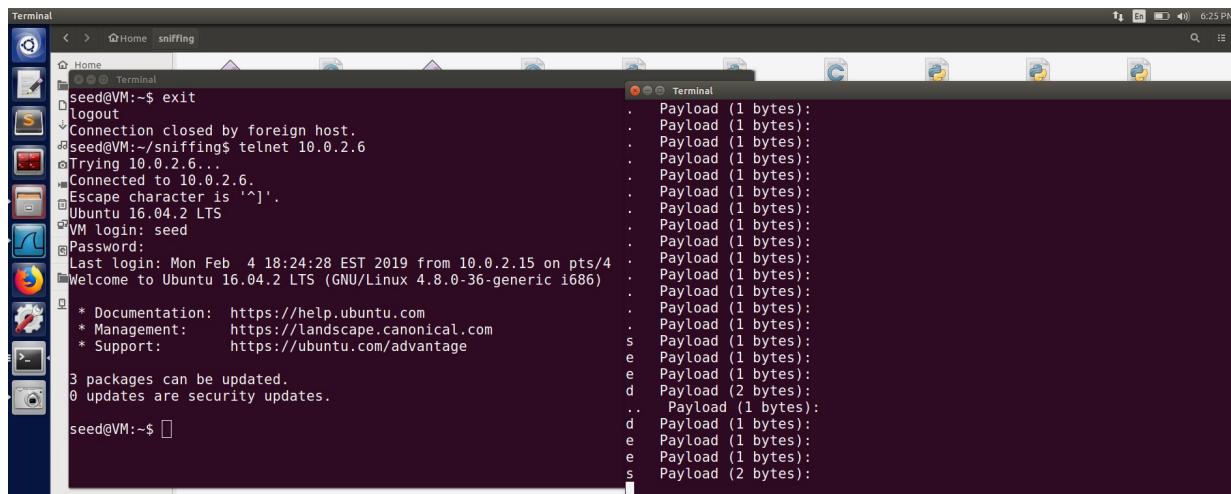
```
seed@VM:~$ telnet 10.0.2.6 45
Trying 10.0.2.6...
telnet: Unable to connect to remote host: Connection refused
seed@VM:~$
```



**Observation:** In this task we change the filter to ip proto tcp and dst portrange 10-100

**Explanation:** we use the command ‘telnet 10.0.2.6 45’ which specifies the ip address and the port number. We can see in wireshark that the packet was sent to port 45

### Task 2.1C: Sniffing Passwords



\*any

telnet

| No. | Time          | Source    | Destination | Protocol | Length | Info            |
|-----|---------------|-----------|-------------|----------|--------|-----------------|
| 125 | 21.181682999  | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 127 | 21.182623962  | 10.0.2.15 | 10.0.2.15   | TELNET   | 69     | Telnet Data ... |
| 129 | 1491808230    | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 130 | 21.492045591  | 10.0.2.6  | 10.0.2.15   | TELNET   | 69     | Telnet Data ... |
| 132 | 21.745779537  | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 133 | 21.746042425  | 10.0.2.6  | 10.0.2.15   | TELNET   | 69     | Telnet Data ... |
| 135 | 22.063162939  | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 136 | 22.0653162939 | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 138 | 22.256888329  | 10.0.2.15 | 10.0.2.6    | TELNET   | 70     | Telnet Data ... |
| 141 | 22.2576011898 | 10.0.2.6  | 10.0.2.15   | TELNET   | 78     | Telnet Data ... |
| 143 | 23.232853962  | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 145 | 23.398596935  | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 146 | 23.475985935  | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |

Frame 141: 79 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0

- Linux cooked capture
- Internet Protocol Version 4, Src: 10.0.2.6, Dst: 10.0.2.15
- Transmission Control Protocol, Src Port: 23, Dst Port: 42712, Seq: 126, Ack: 139, Len: 10

▼ Telnet

Data: Password:

```
0000  00 04 00 01 00 06 08 00 27 ae fd 1e 00 00 08 00 .E..>..@.^.  
0010  45 10 00 3e c3 b1 40 00 48 06 5e e4 0a 00 02 06 E..>..@.^.  
0020  0a 00 02 0f 00 17 a6 d8 ee fe 9a 3b 9f ea b1 1e .....;....  
0030  80 18 00 e3 18 45 00 00 b1 01 08 0a 00 0a 62 63 ....E.....bc  
0040  00 0a 1b 3c 0d 01 73 73 77 0f 72 64 38 29 ...pass word!
```

Data (telnet.data), 10 bytes

Packets: 171 · Displayed: 90 (52.6%)

Profile: Default

\*any

telnet

| No. | Time          | Source    | Destination | Protocol | Length | Info            |
|-----|---------------|-----------|-------------|----------|--------|-----------------|
| 125 | 21.181682999  | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 127 | 21.182623962  | 10.0.2.15 | 10.0.2.15   | TELNET   | 69     | Telnet Data ... |
| 129 | 1491808230    | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 130 | 21.492045591  | 10.0.2.6  | 10.0.2.15   | TELNET   | 69     | Telnet Data ... |
| 132 | 21.745779537  | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 133 | 21.746042425  | 10.0.2.6  | 10.0.2.15   | TELNET   | 69     | Telnet Data ... |
| 135 | 22.063162939  | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 136 | 22.0653162939 | 10.0.2.15 | 10.0.2.15   | TELNET   | 69     | Telnet Data ... |
| 138 | 22.256888329  | 10.0.2.15 | 10.0.2.15   | TELNET   | 70     | Telnet Data ... |
| 141 | 22.2576011898 | 10.0.2.6  | 10.0.2.15   | TELNET   | 78     | Telnet Data ... |
| 143 | 23.232853962  | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 145 | 23.398596935  | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |
| 146 | 23.475985935  | 10.0.2.15 | 10.0.2.6    | TELNET   | 69     | Telnet Data ... |

Frame 143: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface 0

- Linux cooked capture
- Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.6
- Transmission Control Protocol, Src Port: 42712, Dst Port: 23, Seq: 139, Ack: 136, Len: 1

▼ Telnet

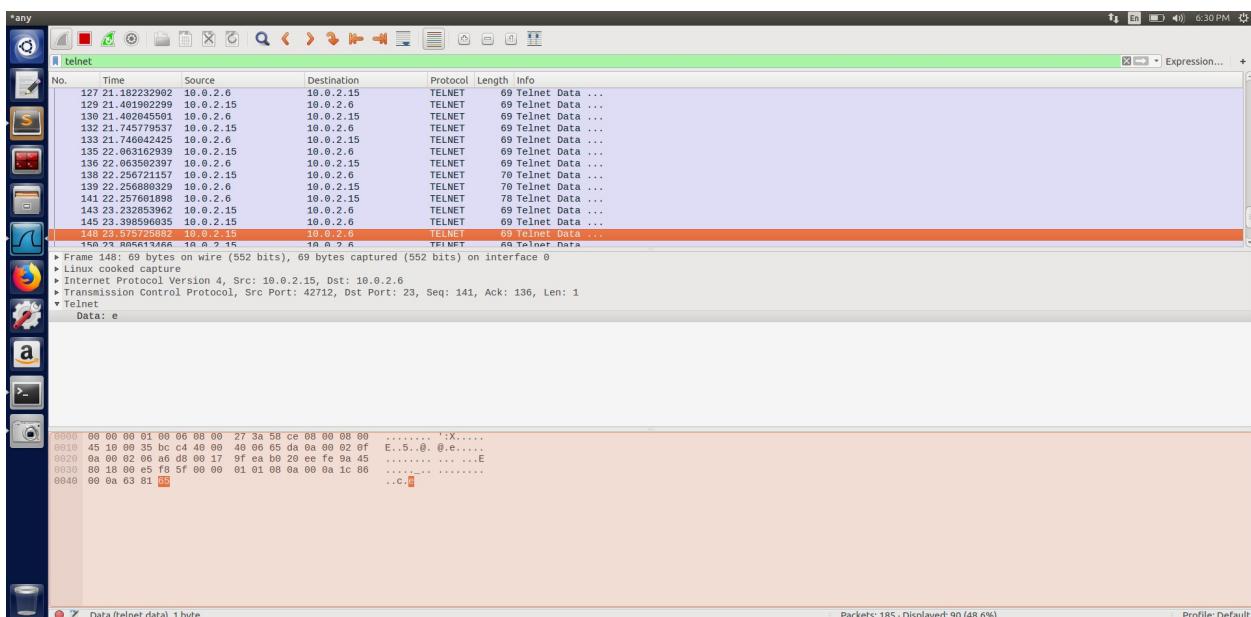
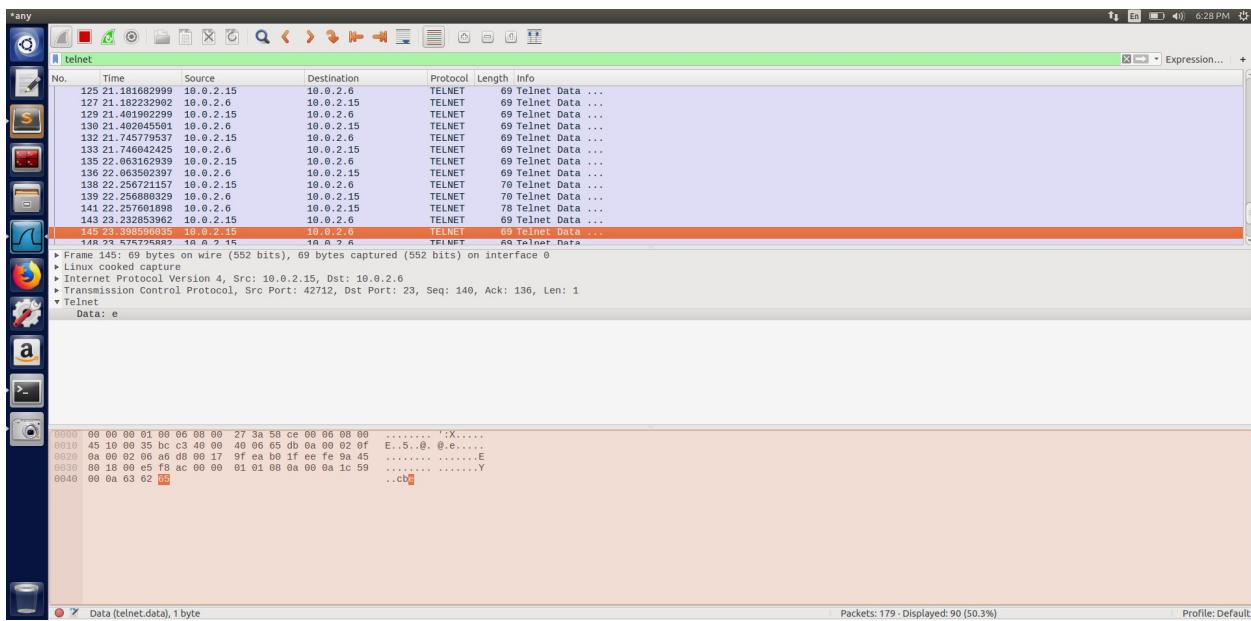
Data: d

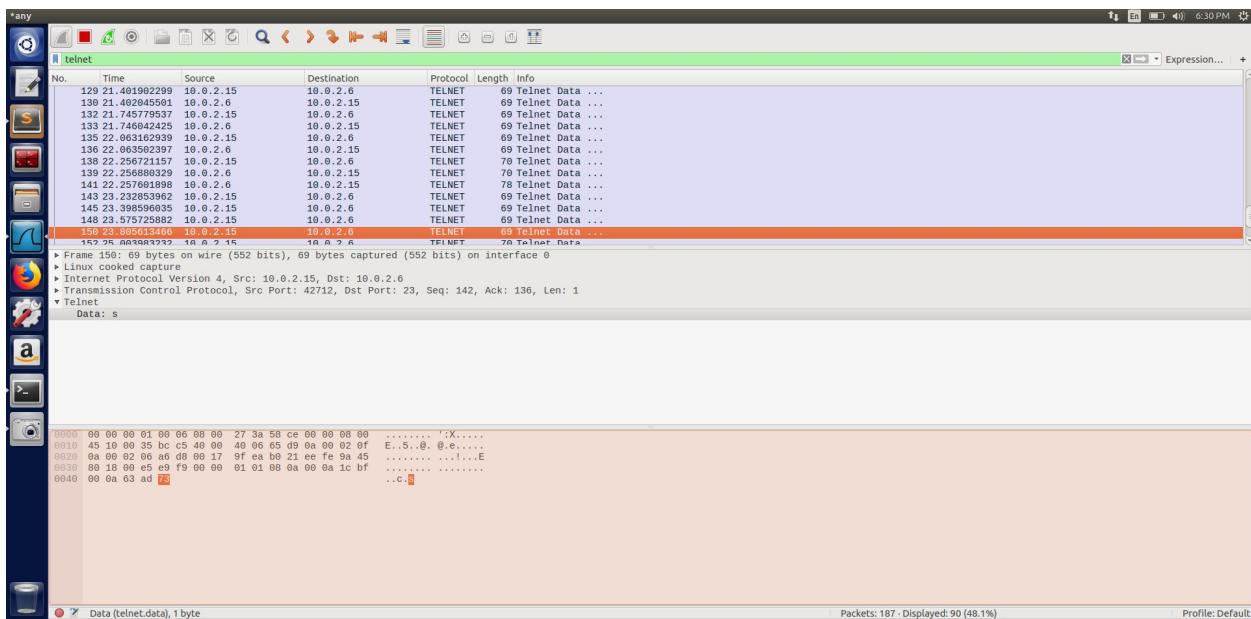
```
0000  05 00 00 01 00 06 08 00 27 3a 58 ce 00 00 08 00 .X.....  
0010  45 10 00 35 bc c2 40 00 48 06 65 dc 0a 00 02 0f E..5..@.^.  
0020  0a 00 02 06 a0 d8 00 17 9f ea b0 1e ee fe 9a 45 .....;....E  
0030  80 18 00 e5 fa d5 00 00 b1 01 08 0a 00 0a 1c 30 .....0  
0040  00 0a 02 03 ..bc
```

Data (telnet.data), 1 byte

Packets: 174 · Displayed: 90 (51.7%)

Profile: Default





As seen from the above screenshots we are using telnet to connect to 10.0.2.6 and when it asks for username and password we can see from the other VM that we are able to get the password character by character.

## Task 2.2A

```

Terminal
seed@VM:~/sniffing$ gcc -o spoof spoof.c -lpcap
seed@VM:~/sniffing$ sudo ./spoof
Sending spoofed IP packet...
    From: 10.0.2.6
    To: 10.0.2.15
seed@VM:~/sniffing$ 
```

```
~\sniffing\spooftest.c - Sublime Text (UNREGISTERED)
```

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <sys/types.h>
4 #include <arpa/inet.h>
5 #include <pcap.h>
6 #include <sys/socket.h>
7 #include <netinet/in.h>
8
9 #define SRC_IP "192.168.2.0"
10#define DST_IP "192.168.2.15"
11#define ETHER_ADDR_LEN 6
12#define PACKETLEN 1500
13#define SIZE_ETHERNET 14
14
15 struct ipheader {
16     unsigned char iph_ihl4; //IP header length
17     unsigned char iph_ver4; //IP version
18     unsigned char iph_tos; //Type of service
19     unsigned short int iph_tlen; //IP Total Length (data + header)
20     unsigned short int iph_id; //Identification
21     unsigned short int iph_flag3; //Fragmentation flags
22     unsigned short int iph_offset13; //Flags offset
23     unsigned char iph_ttl; //Time to live
24     unsigned char iph_protocol; //Protocol type
25     unsigned short int iph_cksum; //IP datagram checksum
26     struct in_addr iph_sourcelp; //Source IP address
27     struct in_addr iph_destip; //Destination IP address
28 };
29
30 struct ethheader {
31     u_char ether_dhost[ETHER_ADDR_LEN]; // destination host address *
32     u_char ether_shost[ETHER_ADDR_LEN]; //source host address */
33     u_short ether_type; //IP? ARP? RARP? Etc */
34 };
35
36 /* ICMP Header */
37 struct icmpheader {
38     unsigned char icmp_type; //ICMP message type
39     unsigned short icmp_cksum; //Checksum for ICMP Header and data
40     unsigned short icmp_seq; //Used in echo request/reply to identify request
41     unsigned short icmp_id; //Identifies the sequence of echo messages,
42     /*#if more than one is sent.
43     */;
44 };
45
46 /* TCP Header */
47 struct tcphandler {
48     u_short tcp_sport; /* source port */
49     u_short tcp_dport; /* destination port */
50     u_int tcp_seq; /* sequence number */
51     u_int tcp_ack; /* acknowledgement number */
52     u_short tcp_rfc2; /* don't know, read */
53     u_char tcp_flags;
54     u_short tcp_win; /* window */
55 }
```

-/sniffing/sniff.c - Sublime Text (UNREGISTERED)

```
46 /* TCP Header */
47 struct tcphdr {
48     u_short tcp_sport;           /* source port */
49     u_short tcp_dport;          /* destination port */
50     u_int   tcp_seq;            /* sequence number */
51     u_int   tcp_ack;            /* acknowledgement number */
52     u_char  tcp_offx2;          /* data offset, rsvd */
53     u_char  tcp_flags;
54     u_short tcp_wnd;            /* window */
55     u_short tcp_urp;            /* checksum */
56     u_short tcp_urgp;           /* urgent pointer */
57 };
58 }
59
60 /* UDP Header */
61 struct udphdr {
62     u_short udp_sport;           /* source port */
63     u_short udp_dport;          /* destination port */
64     u_int16_t udp_len;           /* total length */
65     u_int16_t udp_ulen;          /* user data len */
66     u_int16_t udp_sum;           /* udp checksum */
67 };
68 }
69 void send_raw_ip_packet(struct ipheader* ip);
70
71
72
73 void main()
74 {
75     char buff[PACKET_LEN];
76     struct ipheader *ip = (struct ipheader *)buffer;
77     struct udphdr *udp = (struct udphdr *)buffer + sizeof(struct ipheader);
78
79     /*Fill UDP data*/
80     char *data = buffer + sizeof(struct ipheader) + sizeof(struct udphdr);
81     char *msg="Hello Server.\n";
82
83     int data_len=strlen(msg);
84     strcpy(data,msg,data_len);
85
86     /*Fill IP header*/
87     udp->udp_sport = htons(9999);
88     udp->udp_dport = htons(8888);
89     udp->udp_ulen = htons(sizeof(struct udphdr)+data_len);
90     udp->udp_sum = 0;
91
92
93     /*Fill IP header*/
94     ip->iph_ver4;
95     ip->iph_tos;
96     ip->iph_ttl=255;
97     ip->iph_sourceip.s_addr=inet_addr(SRC_IP);
98     ip->iph_destip.s_addr=inet_addr(DST_IP);
99     ip->iph_protocol=IPPROTO_UDP;
100    ip->iph_lenth htons(sizeof(struct ipheader) + sizeof(struct udphdr) + data_len);
```

The figure shows the Wireshark interface. At the top, there's a toolbar with various icons for file operations, search, and selection. Below the toolbar is a search bar containing the text "Apply a display filter ... <Ctrl-/>". The main area displays a table of captured network packets. The columns are labeled: No., Time, Source, Destination, Protocol, Length, and Info. There are two rows of data:

| No. | Time         | Source    | Destination | Protocol | Length | Info                                       |
|-----|--------------|-----------|-------------|----------|--------|--|
| 1   | 0.000000000  | 10.0.2.6  | 10.0.2.15   | UDP      | 62     | 9999 → 8888 Len=14                         |
| 2   | 0.0000031052 | 10.0.2.15 | 10.0.2.6    | ICMP     | 86     | Destination unreachable (Port unreachable) |

From the above screenshot we can see that we are able to send spoofed packets from 10.0.2.6 to 10.0.2.15

## Task 2.2 B

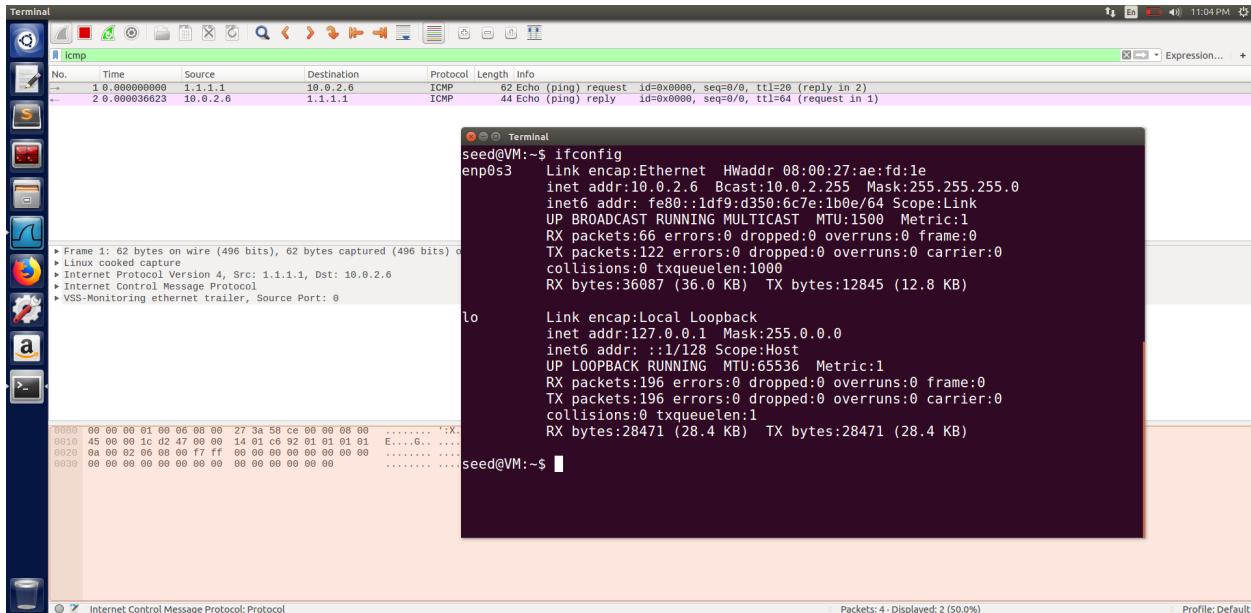
```
seed@VM:~/sniffing$ ifconfig
enp0s3      Link encap:Ethernet HWaddr 08:00:27:3a:58:ce
            inet addr:10.0.2.15 Bcast:10.0.2.255 Mask:255.255.255.0
            inet6 addr: fe80::4e83:9237%7:elc:80fe/64 Scope:Link
              UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
              RX packets:20280 errors:0 dropped:0 overruns:0 frame:0
              TX packets:9708 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:1000
              RX bytes:26140990 (26.1 MB)  TX bytes:1021579 (1.0 MB)

lo          Link encap: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:881 errors:0 dropped:0 overruns:0 frame:0
              TX packets:881 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:1
              RX bytes:93032 (93.0 KB)  TX bytes:93032 (93.0 KB)

seed@VM:~/sniffing$ gcc -o icmpecho icmpecho.c -lpcap
seed@VM:~/sniffing$ sudo ./icmpecho
seed@VM:~/sniffing$ sudo ./icmpecho
seed@VM:~/sniffing$
```

```
~/sniffing/icmpecho.c - Sublime Text (UNREGISTERED)
49     sum = (sum >> 16) + (sum & 0xffff); // add hi 16 to low 16
50     sum += (sum >> 16); // add carry
51     return (unsigned short)(~sum);
52 }
53 void send_raw_ip_packet(struct ipheader* ip)
54 {
55     struct sockaddr_in dest_info;
56     int enable = 1;
57
58     int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
59
60     setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
61                 &enable, sizeof(enable));
62     dest_info.sin_family = AF_INET;
63     dest_info.sin_addr.s_addr = ip->iph_destip;
64     sendto(sock, ip, htons(ip->iph_len), 0,
65             (struct sockaddr *)&dest_info, sizeof(dest_info));
66 }
67
68 int main()
69 {
70     char buffer[1500];
71
72     memset(buffer, 0, 1500);
73
74     struct ifaddrs *id;
75     int val;
76     val = getifaddrs(&id); // to get the IP address of the machine
77
78     struct icmpheader *icmp = (struct icmpheader *)
79         (buffer + sizeof(struct ipheader));
80     icmp->icmp_type = 8; // ICMP type: 8 is request, 0 is reply.
81     icmp->icmp_cksum = 0;
82     icmp->icmp_cksum = in_cksum((unsigned short *)icmp,
83                                   sizeof(struct icmpheader));
84     struct ipheader *ip = (struct ipheader *) buffer;
85     ip->iph_ver = 4;
86     ip->iph_ihl = 5;
87     ip->iph_ttl = 20;
88     ip->iph_src.s_addr = inet_addr("1.1.1.1");
89     ip->iph_destip.s_addr = inet_addr("10.0.2.6");
90     ip->iph_protocol = IPPROTO_ICMP;
91     ip->iph_len = htons(sizeof(struct ipheader) +
92                           sizeof(struct icmpheader));
93
94     send_raw_ip_packet (ip);

```



**Observation :** As seen from the above screenshot we are able to send spoofed ICMP echo request to 1.1.1.1 and also get ICMP reply

**Explanation :** We send ICMP echo from 10.0.2.15 and the ICMP echo reply is sent to 10.0.2.6 from where it assumes the request is made. We can see that on Wireshark as seen from the screenshot above

#### **Question 4 Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is?**

If we set the length to any arbitrary value then the IP packet will not be formed properly and when we will send the packet then the size will be truncated because it being too large

#### **Question 5 Using the raw socket programming, do you have to calculate the checksum for the IP header?**

OS calculates the checksum before sending the packet therefore, we don't need to calculate the checksum

#### **Question 6 Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?**

If we do not run the program using the root privileges then it will show an error because the program will need to access the NIC and when we use raw socket it gives the user root privileges.

## Task 2.3

Screenshot of Wireshark showing captured ICMP traffic:

| No. | Time        | Source         | Destination    | Protocol | Length | Info  |
|-----|-------------|----------------|----------------|----------|--------|---|
| 6   | 1.618829952 | 10.0.2.6       | 172.217.12.164 | ICMP     | 100    | Echo (ping) request id=0x0e4e, seq=1/256, ttl=64 (no response found!) |
| 7   | 1.618831373 | 172.217.12.164 | 10.0.2.6       | ICMP     | 100    | Echo (ping) reply id=0x0e4e, seq=56/256, ttl=50                       |
| 10  | 2.259816333 | 10.0.2.6       | 172.217.12.164 | ICMP     | 100    | Echo (ping) reply id=0x0e4e, seq=512/2, ttl=50                        |
| 11  | 3.284091422 | 172.217.12.164 | 10.0.2.6       | ICMP     | 100    | Echo (ping) reply id=0x0e4e, seq=768/3, ttl=50                        |
| 12  | 4.311418925 | 10.0.2.6       | 172.217.12.164 | ICMP     | 100    | Echo (ping) reply id=0x0e4e, seq=1024/4, ttl=50                       |
| 13  | 5.331972995 | 172.217.12.164 | 10.0.2.6       | ICMP     | 100    | Echo (ping) reply id=0x0e4e, seq=1280/5, ttl=50                       |

Details for the selected ICMP reply packet (Frame 7):

- Frame 7: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0
- Linux cooked capture
- Internet Protocol Version 4, Src: 172.217.12.164, Dst: 10.0.2.6
- Internet Control Message Protocol (Echo (ping) reply)
- Checksum: **0xf1b0 [incorrect, should be 0xc89]** [Checksum Status: Bad]
- Identifier (BE): 3662 (0x0e4e)
- Identifier (LE): 2562 (0xe4e)
- Sequence number (BE): 56 (0x0100)
- Sequence number (LE): 1 (0x0001)
- Timestamp from icmp data: Feb 4, 2019 19:20:57.586414000 EST
- [Timestamp from icmp data (relative): 3.074319182 seconds]
- Data (48 bytes):  
0000: 00 04 00 01 00 06 08 00 27 3a 58 ce 00 00 08 00 ..... :X.....  
0010: 45 00 00 54 ed 03 40 00 32 01 95 c2 ac d9 0c a4 E..T.CB. 2.....  
0020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....N.....A  
0030: 00 12 39 00 00 00 00 00 00 00 00 00 00 00 00 00 .....:.....!#  
0040: 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 .....:.....!#  
0050: 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 S\&(')+ ,./0123  
0060: 34 35 36 37 456/

Screenshot of a terminal window showing the process of sending a spoofed ICMP packet:

```
Received packet
From:10.0.2.6
To:172.217.12.164
Protocol: ICMP

Sending spoofed IP packet...
From:172.217.12.164
To:10.0.2.6

^C
seed@VM:~/sniffing$ ifconfig
enp0s3      Link encap:Ethernet  HWaddr 08:00:27:3a:58:ce
              inet  addr:10.0.2.15   Bcast:10.0.2.255  Mask:255.255.255.0
              inet6 addr: fe80::4e83:9237%enp0s3/64 Scope:Link
```

```
Terminal
eed@VM:~/sniffing$ ping www.google.com
PING www.google.com (172.217.12.164) 56(84) bytes of data.
4 bytes from lga25s62-in-f4.1e100.net (172.217.12.164): icmp_seq=256 ttl=50
=226 ms
C
-- www.google.com ping statistics ---
  packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 226.395/226.395/226.395/0.000 ms
eed@VM:~/sniffing$ ifconfig
np0s3    Link encap:Ethernet  HWaddr 08:00:27:ae:fd:1e
          inet addr:10.0.2.6  Bcast:10.0.2.255  Mask:255.255.255.0
            inet6 addr: fe80::1df9:d350:6c7e:1b0e/64 Scope:Link
              UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
              RX packets:655 errors:0 dropped:0 overruns:0 frame:0
              TX packets:354 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:1000
              RX bytes:80189 (80.1 KB)  TX bytes:35504 (35.5 KB)

lo      Link encap: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:515 errors:0 dropped:0 overruns:0 frame:0
              TX packets:515 errors:0 dropped:0 overruns:0 carrier:0
```

We can see from the above screenshot that we are able to sniff and spoof i.e., we are able to spoof ICMP echo reply.