



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

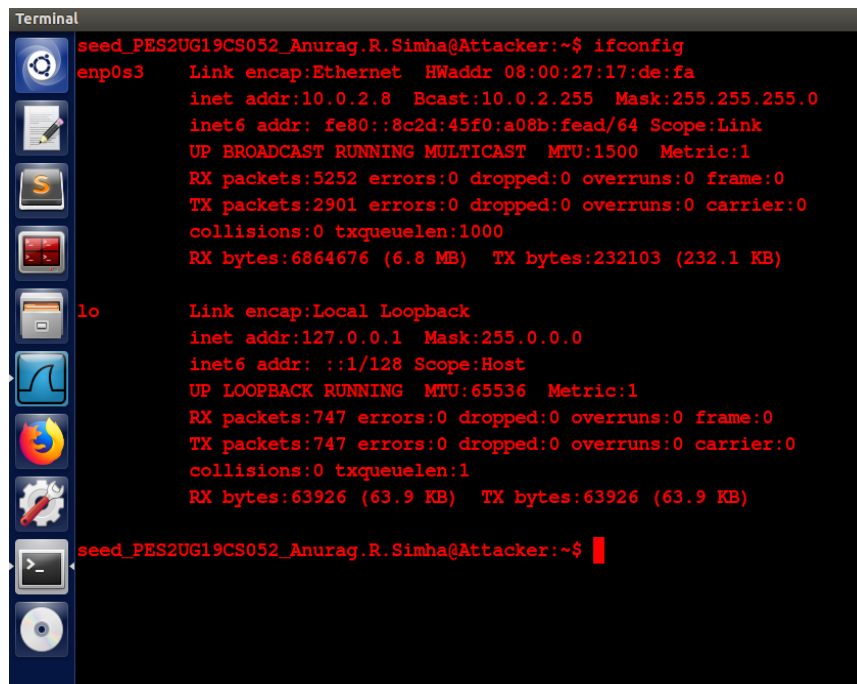
The Table of Contents

1. The Virtual Machines	2
2. Using Tools to Sniff and Spoof Packets using Scapy.....	3
Task 1: Sniffing Packets	3
Task 1.1 Sniff IP packets using Scapy	3
Task 1.2 Capturing ICMP, TCP packet and Subnet.....	7
a) Capture only the ICMP packet	7
b) Capture any TCP packet that comes from a particular IP and with a destination port number 23	12
c) Capture packets comes from or go to a particular subnet	14
Task 2: Spoofing	16
Task 3: Traceroute	19
Task 4: Sniffing and then Spoofing	21

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

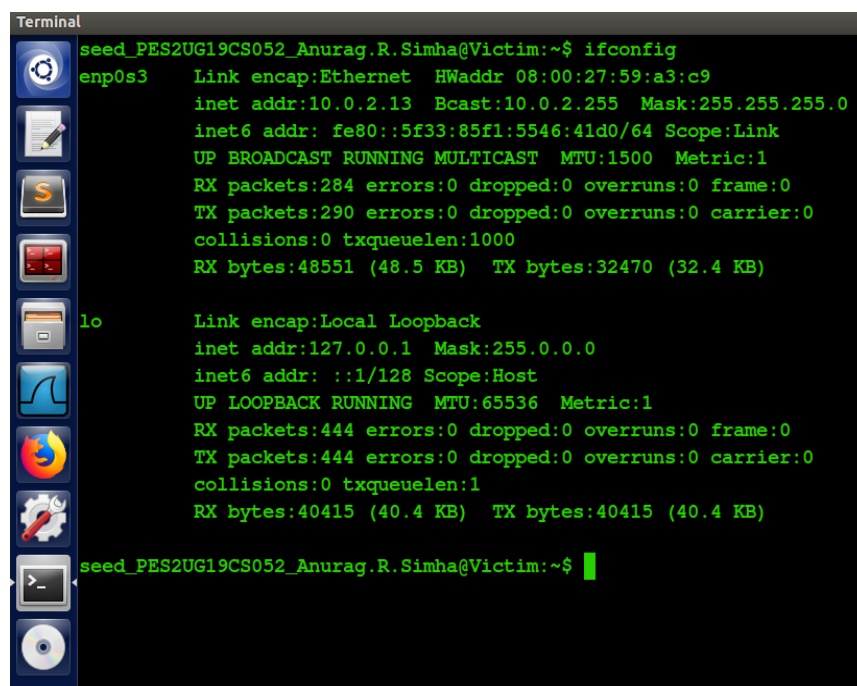


```
Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ ifconfig
enp0s3  Link encap:Ethernet  HWaddr 08:00:27:17:de:fa
        inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
        inet6 addr: fe80::8c2d:45f0:a08b:fead/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:5252 errors:0 dropped:0 overruns:0 frame:0
        TX packets:2901 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:6864676 (6.8 MB)  TX bytes:232103 (232.1 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:747 errors:0 dropped:0 overruns:0 frame:0
        TX packets:747 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:63926 (63.9 KB)  TX bytes:63926 (63.9 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$
```

Victim: 10.0.2.13



```
Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ ifconfig
enp0s3  Link encap:Ethernet  HWaddr 08:00:27:59:a3:c9
        inet addr:10.0.2.13  Bcast:10.0.2.255  Mask:255.255.255.0
        inet6 addr: fe80::5f33:85f1:5546:41d0/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:284 errors:0 dropped:0 overruns:0 frame:0
        TX packets:290 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:48551 (48.5 KB)  TX bytes:32470 (32.4 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:444 errors:0 dropped:0 overruns:0 frame:0
        TX packets:444 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:40415 (40.4 KB)  TX bytes:40415 (40.4 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$
```

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`.

```
Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ sudo apt-get install scapy
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'python-scapy' instead of 'scapy'
python-scapy is already the newest version (2.2.0-1).
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$
```

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`

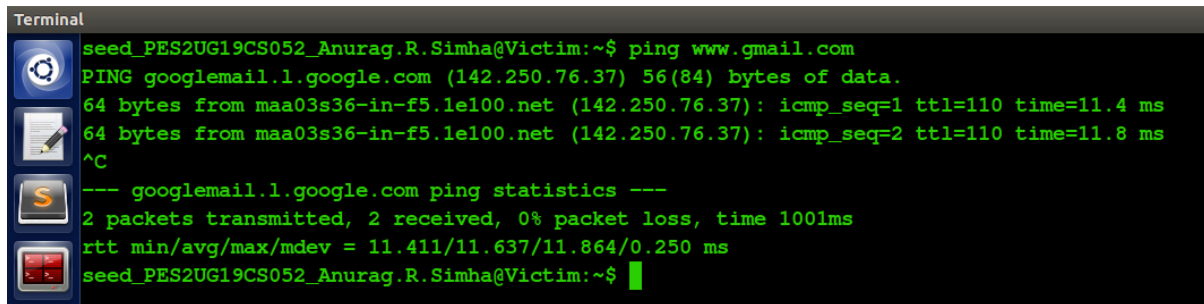
Q. Explain on which VM you ran this command and why? Provide a screenshot of your observations.

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`



```

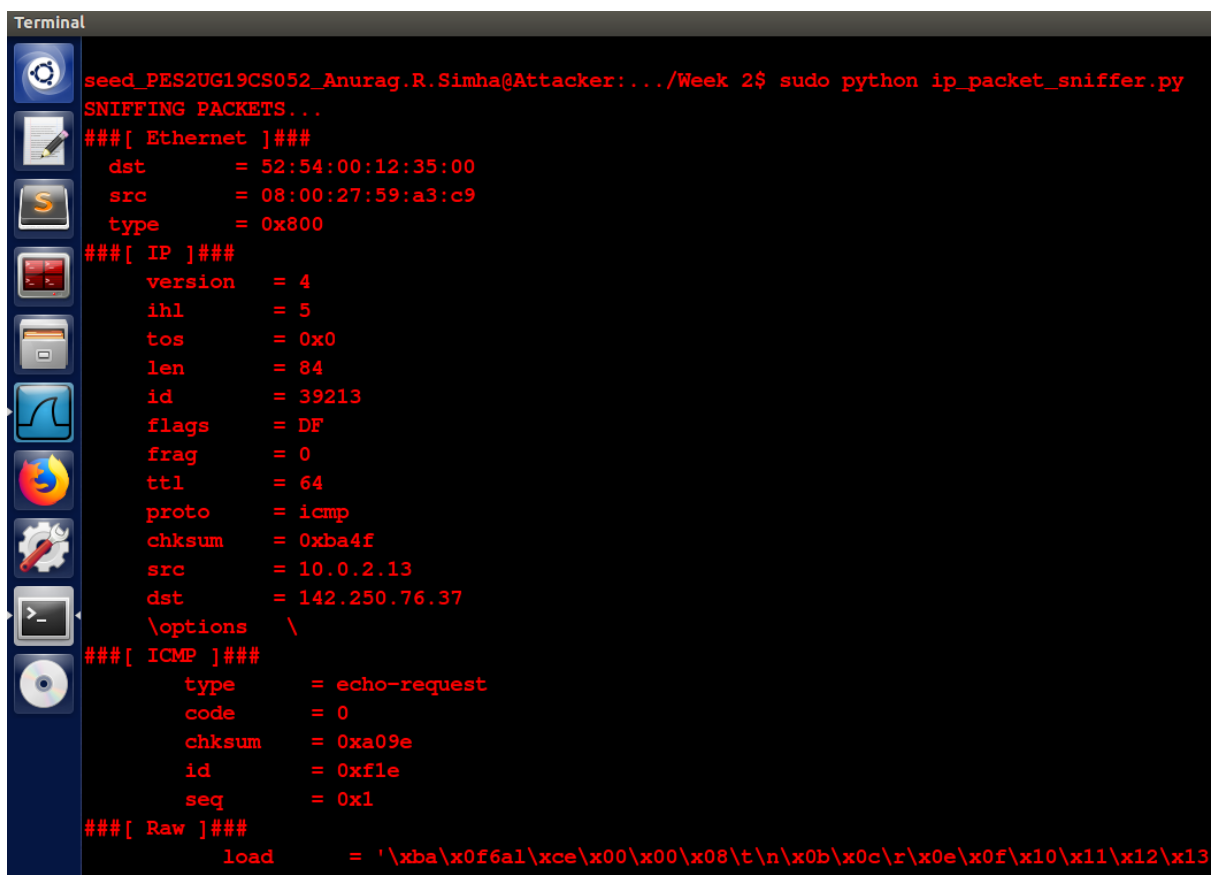
Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ ping www.gmail.com
PING googlemail.l.google.com (142.250.76.37) 56(84) bytes of data.
64 bytes from maa03s36-in-f5.1e100.net (142.250.76.37): icmp_seq=1 ttl=110 time=11.4 ms
64 bytes from maa03s36-in-f5.1e100.net (142.250.76.37): icmp_seq=2 ttl=110 time=11.8 ms
^C
--- googlemail.l.google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 11.411/11.637/11.864/0.250 ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$

```

The following observations were recorded on the attacker VM.

(10.0.2.8 – The attacker machine)

Command: `sudo python ip_packet_sniffer.py`



```

Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$ sudo python ip_packet_sniffer.py
SNIFFING PACKETS...
###[ 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       = 39213
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xba4f
src      = 10.0.2.13
dst      = 142.250.76.37
\options \
###[ ICMP ]###
type     = echo-request
code     = 0
chksum   = 0xa09e
id       = 0xf1e
seq      = 0x1
###[ Raw ]###
load     = '\xba\x0f6a1\xce\x00\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13

```

(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

```

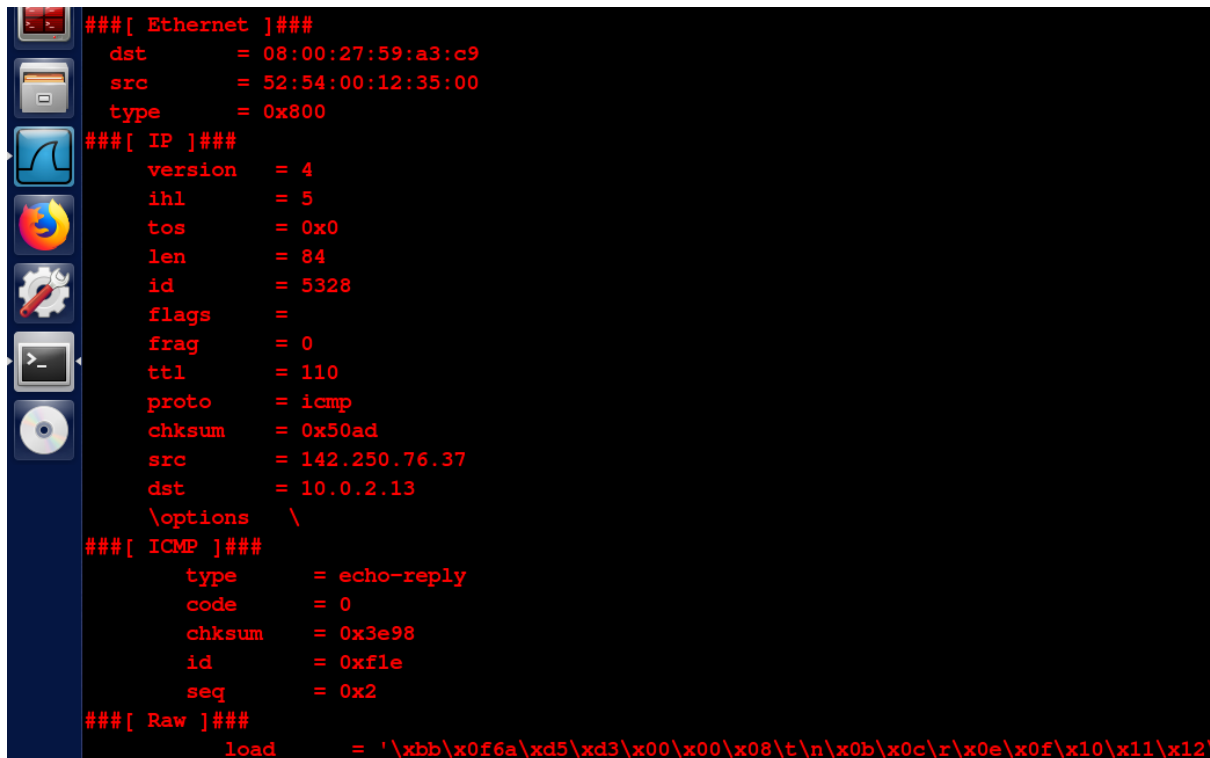
(ii)

```

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)



```

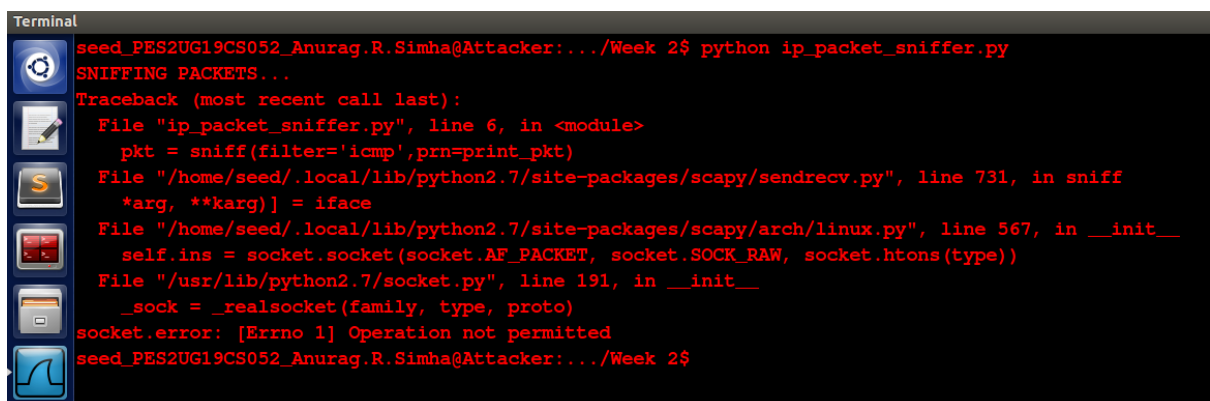
###[ Ethernet ]###
  dst      = 08:00:27:59:a3:c9
  src      = 52:54:00:12:35:00
  type     = 0x800
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 5328
  flags    =
  frag     = 0
  ttl      = 110
  proto    = icmp
  checksum = 0x50ad
  src      = 142.250.76.37
  dst      = 10.0.2.13
  \options \
###[ ICMP ]###
  type     = echo-reply
  code     = 0
  checksum = 0x3e98
  id       = 0xf1e
  seq      = 0x2
###[ Raw ]###
  load     = '\xbb\x0f6a\xd5\xd3\x00\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12

```

(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.



```

Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$ python ip_packet_sniffer.py
SNIFFING PACKETS...
Traceback (most recent call last):
  File "ip_packet_sniffer.py", line 6, in <module>
    pkt = sniff(filter='icmp', prn=print_pkt)
  File "/home/seed/.local/lib/python2.7/site-packages/scapy/sendrecv.py", line 731, in sniff
    *arg, **karg]] = iface
  File "/home/seed/.local/lib/python2.7/site-packages/scapy/arch/linux.py", line 567, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type))
  File "/usr/lib/python2.7/socket.py", line 191, in __init__
    _sock = _realsocket(family, type, proto)
socket.error: [Errno 1] Operation not permitted
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$

```

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.

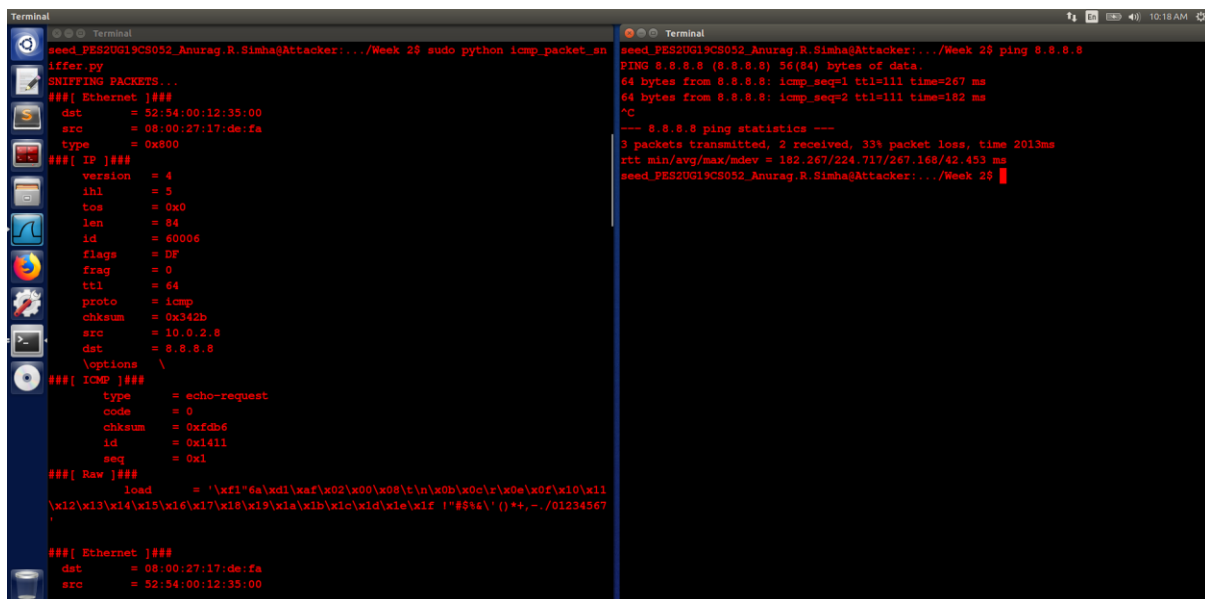
```

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`

```

Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$ 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=111 time=267 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=111 time=182 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 2 received, 33% packet loss, time 2013ms
rtt min/avg/max/mdev = 182.267/224.717/267.168/42.453 ms
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$

```


The sniffer window:

Command: `sudo python icmp_packet_sniffer.py`

```

Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$ sudo python icmp_packet_sniffer.py
SNIFFING PACKETS...
#### 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       = 60006
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  checksum = 0x342b
  src      = 10.0.2.8
  dst      = 8.8.8.8
  \options \
#### ICMP ####
  type     = echo-request
  code     = 0
  checksum = 0xfdb6
  id       = 0x1411
  seq      = 0x1
#### Raw ####
  load     = '\xf1"6a\xd1\xaf\x02\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&'()*+,-./01234567'

#### Ethernet ####
  dst      = 08:00:27:17:de:fa
  src      = 52:54:00:12:35:00

```

(i)

```

#### 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       = 5329
  flags    = 
  frag     = 0
  ttl      = 111
  proto    = icmp
  checksum = 0x1aa9
  src      = 8.8.8.8
  dst      = 10.0.2.8
  \options \
#### ICMP ####
  type     = echo-reply
  code     = 0
  checksum = 0x5b7
  id       = 0x1411
  seq      = 0x1
#### Raw ####
  load     = '\xf1"6a\xd1\xaf\x02\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12'

```

(ii)

```

###[ Ethernet ]###
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:17: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)

```

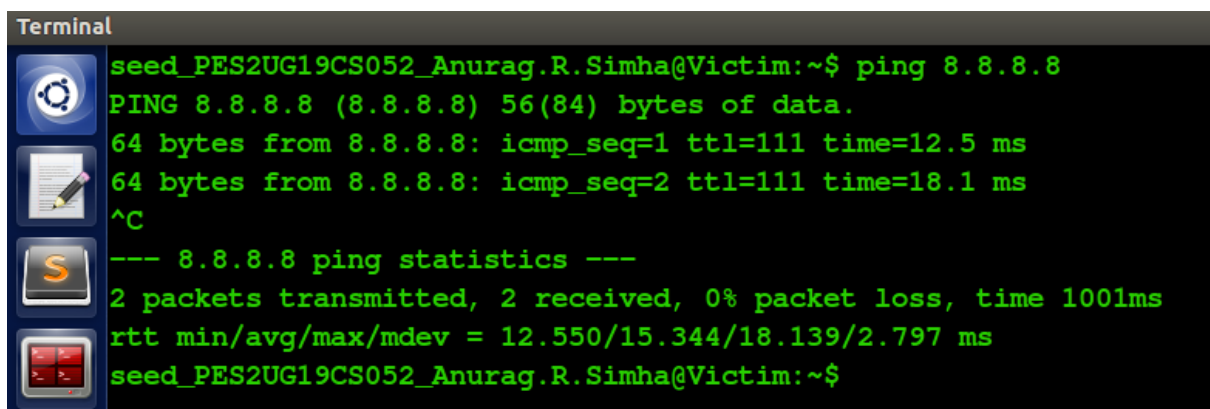
###[ Ethernet ]###
  dst      = 08:00:27:17:de:fa
  src      = 52:54:00:12:35:00
  type     = 0x800
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x18
  len      = 84
  id       = 5331
  flags    =
  frag     = 0
  ttl      = 111
  proto    = icmp
  chksum   = 0x1aa7
  src      = 8.8.8.8
  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)

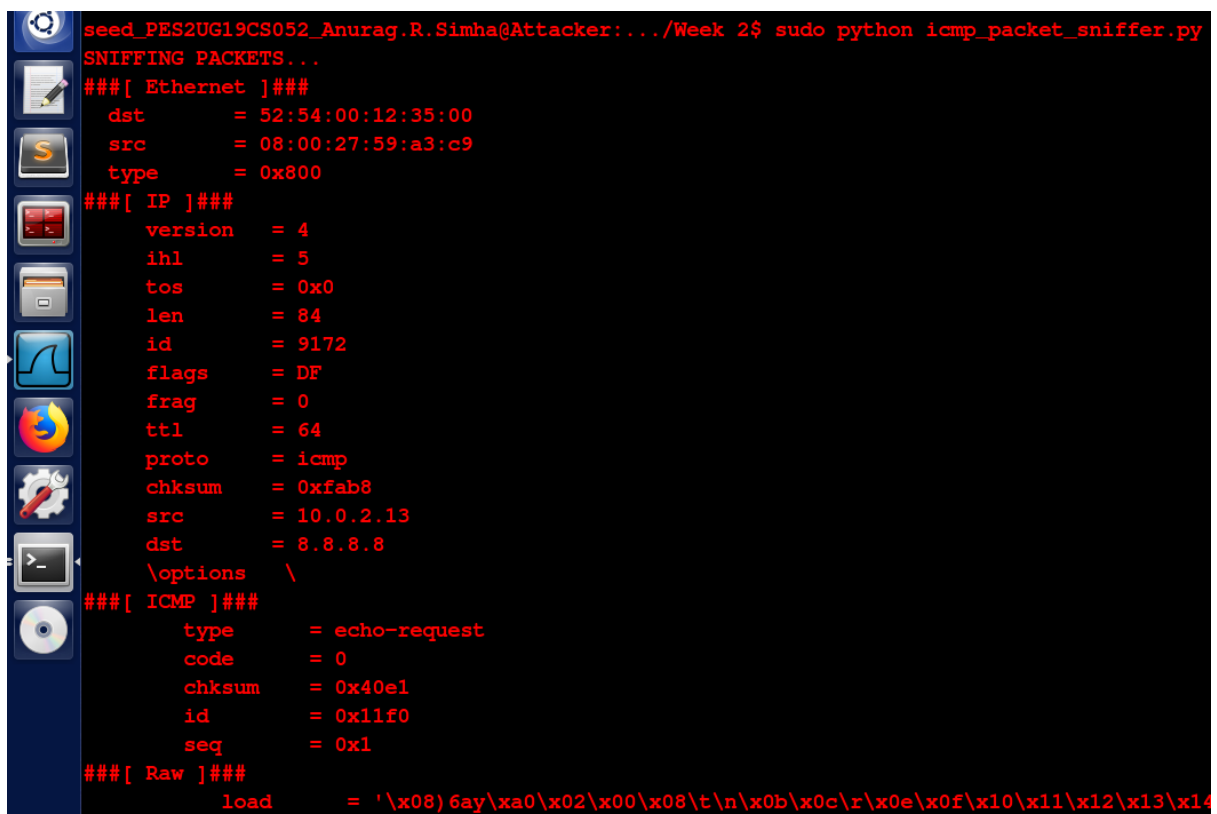
If the command is executed on the victim machine, 10.0.2.8 alters to 10.0.2.13 (IP address of the victim machine).



```

Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ 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=111 time=12.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=111 time=18.1 ms
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 12.550/15.344/18.139/2.797 ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$

```



```

seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$ sudo python icmp_packet_sniffer.py
SNIFFING PACKETS...
#### 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       = 9172
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xfab8
src      = 10.0.2.13
dst      = 8.8.8.8
\options \
#### ICMP ####
type     = echo-request
code     = 0
chksum   = 0x40e1
id       = 0x11f0
seq      = 0x1
#### Raw ####
load     = '\x08)6ay\xa0\x02\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14

```

(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\

```

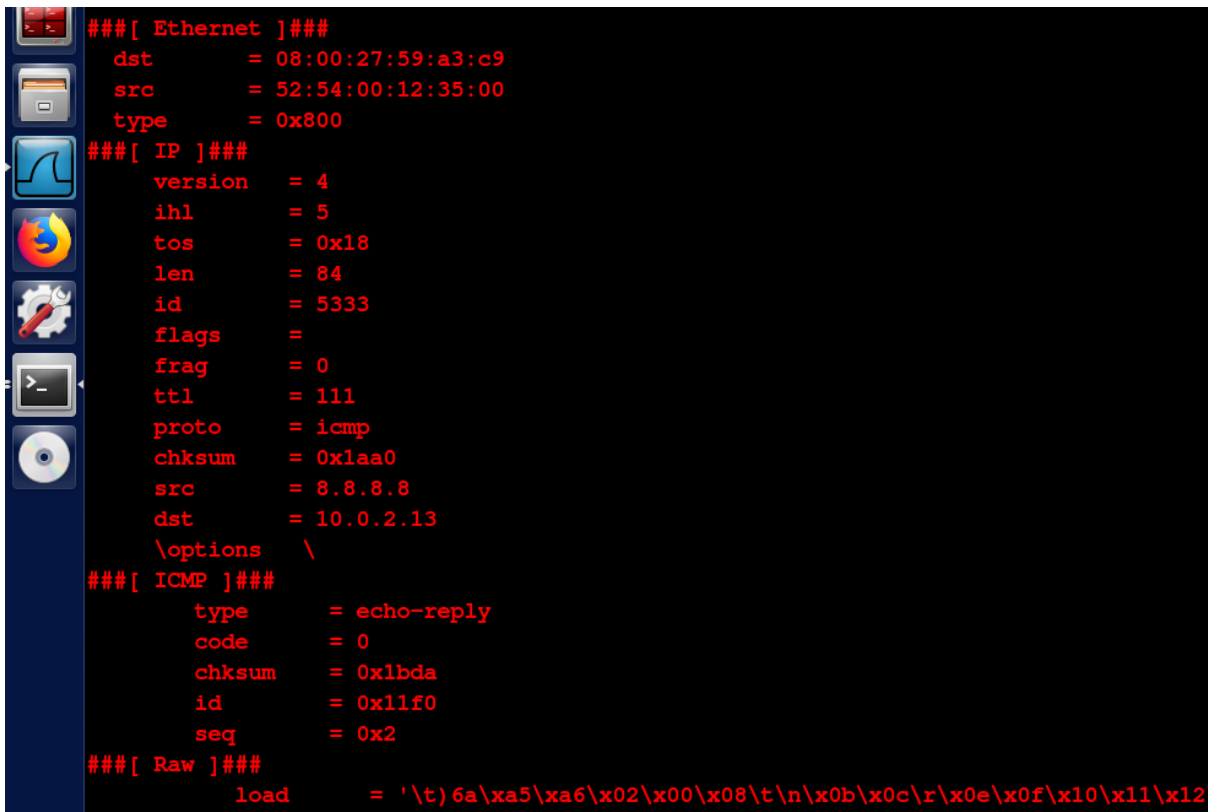
(ii)

```

###[ 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)



```

###[ 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       = 5333
  flags    =
  frag     = 0
  ttl      = 111
  proto    = icmp
  chksum   = 0x1aa0
  src      = 8.8.8.8
  dst      = 10.0.2.13
  \options \
###[ ICMP ]###
  type     = echo-reply
  code     = 0
  chksum   = 0x1bda
  id       = 0x11f0
  seq      = 0x2
###[ Raw ]###
  load     = '\t) 6a\xa5\xa6\x02\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12

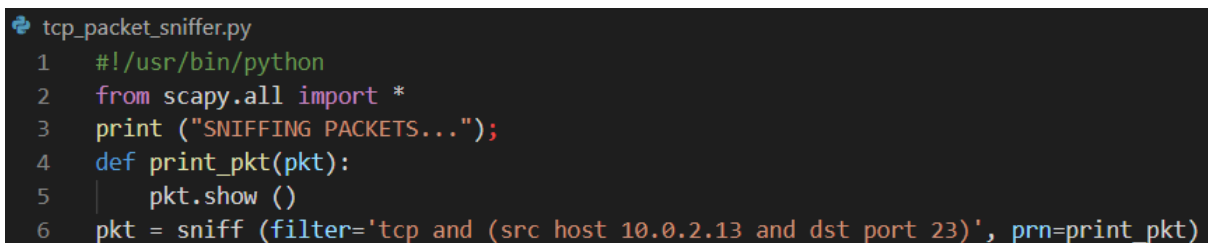
```

(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:



```

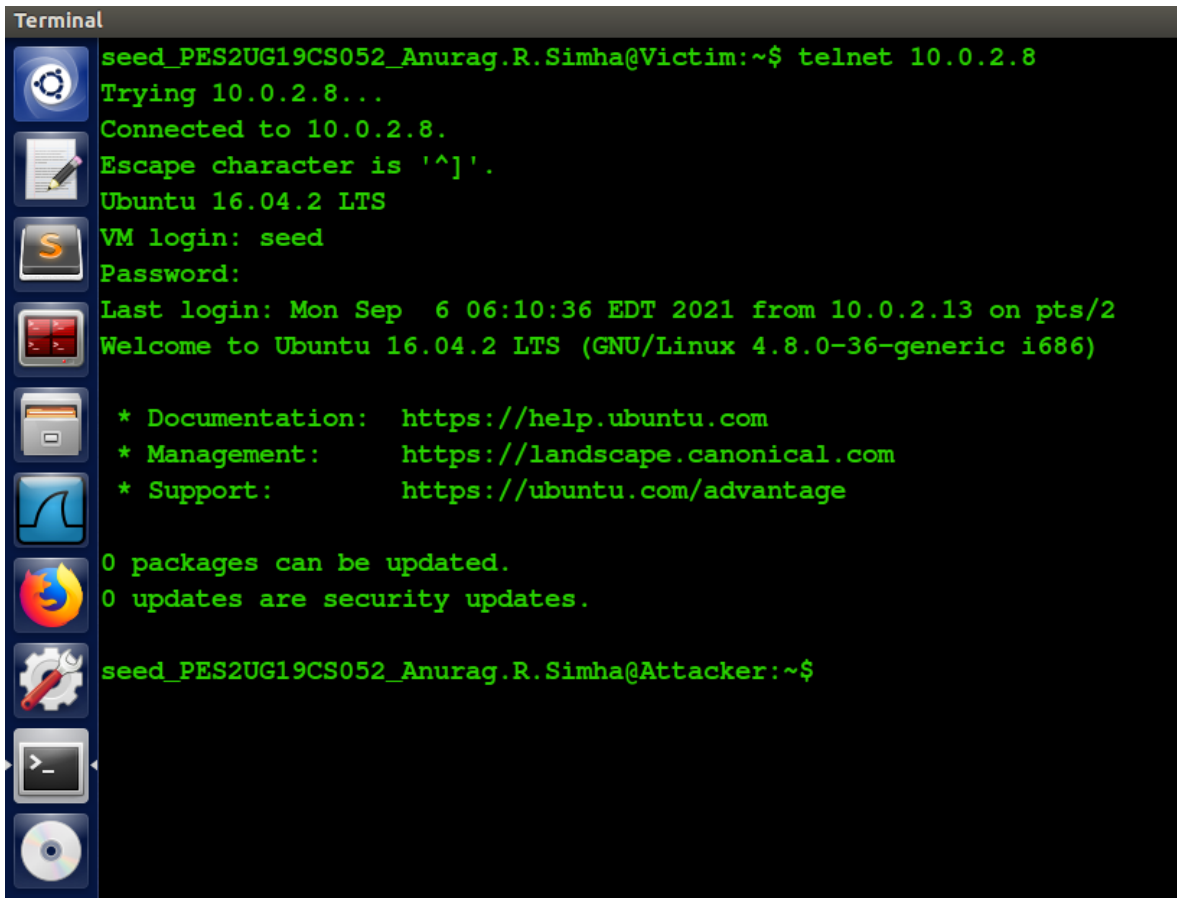
tcp_packet_sniffer.py
1  #!/usr/bin/python
2  from scapy.all import *
3  print ("SNIFFING PACKETS...");
4  def print_pkt(pkt):
5      |   pkt.show ()
6  pkt = sniff (filter='tcp and (src host 10.0.2.13 and dst port 23)', prn=print_pkt)

```

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



```

Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ telnet 10.0.2.8
Trying 10.0.2.8...
Connected to 10.0.2.8.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Sep  6 06:10:36 EDT 2021 from 10.0.2.13 on pts/2
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

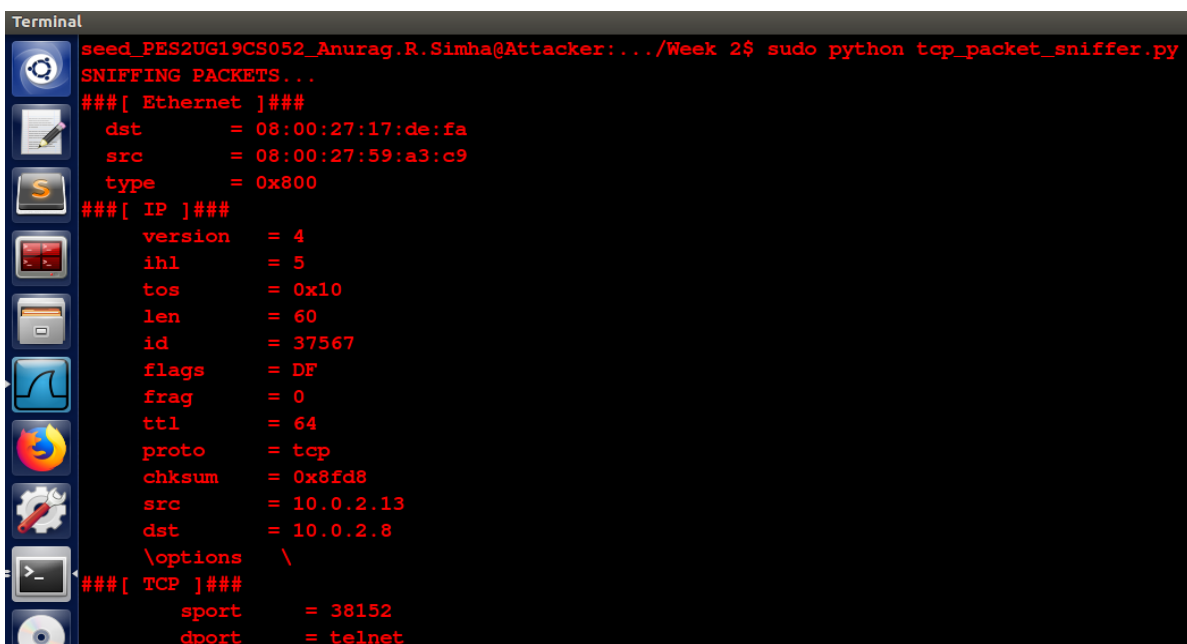
0 packages can be updated.
0 updates are security updates.

seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$

```

On the attacker machine:

Command: sudo python tcp_packet_sniffer.py



```

Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$ sudo python tcp_packet_sniffer.py
SNIFFING PACKETS...
###[ Ethernet ]###
  dst      = 08:00:27:17:de:fa
  src      = 08:00:27:59:a3:c9
  type     = 0x800
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x10
  len      = 60
  id       = 37567
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  checksum = 0x8fd8
  src      = 10.0.2.13
  dst      = 10.0.2.8
  \options \
###[ TCP ]###
  sport    = 38152
  dport    = telnet

```

```

seq      = 49878336
ack      = 0
dataoffs = 10
reserved = 0
flags    = S
window   = 29200
chksum   = 0x8840
urgptr   = 0
options  = [('MSS', 1460), ('SackOK', ''), ('Timestamp', (4425727, 0)), ('NOP', None), ('WScale', 7)]

###[ Ethernet ]###
dst      = 08:00:27:17:de:fa
src      = 08:00:27:59:a3:c9

```

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`

```

Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$ ping 192.168.29.153
PING 192.168.29.153 (192.168.29.153) 56(84) bytes of data.
64 bytes from 192.168.29.153: icmp_seq=1 ttl=127 time=1.89 ms
64 bytes from 192.168.29.153: icmp_seq=2 ttl=127 time=0.793 ms
^C
--- 192.168.29.153 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.793/1.341/1.890/0.549 ms

```

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)

```

Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$ sudo python subnet_sniffer.py
SNIFFING PACKETS...
###[ 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       = 5349
flags    =
frag     = 0
ttl      = 127
proto    = icmp
chksum   = 0x3c76
src      = 192.168.29.153
dst      = 10.0.2.13
\options \
###[ ICMP ]###
type     = echo-reply
code     = 0
chksum   = 0x7472
id       = 0x14a3
seq      = 0x1
###[ Raw ]###
load     = '\xfd>6aWF\x01\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\b8\b9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\x

```



```

###[ 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       = 5350
  flags    =
  frag     = 0
  ttl      = 127
  proto    = icmp
  checksum = 0x3c75
  src      = 192.168.29.153
  dst      = 10.0.2.13
  \options \
###[ ICMP ]###
  type     = echo-reply
  code     = 0
  checksum = 0xd6f8
  id       = 0x14a3
  seq      = 0x2
###[ Raw ]###
  load     = '\xfe>6a\xf3\xbe\x01\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\'

```

Task 2: Spoofing

The python programme:

```

❏ icmp_spoof.py
1  #!/usr/bin/python
2  from scapy.all import *
3  print ("SENDING SPOOFED ICMP PACKET...");
4  IPLayer = IP()
5  IPLayer.src="10.0.2.13"
6  IPLayer.dst="192.168.29.153"
7  ICMPpkt = ICMP()
8  pkt = IPLayer/ICMPpkt
9  pkt.show ()
10 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 from a spoofed source.

```

Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$ sudo python icmp_spoof.py
SENDING SPOOFED ICMP PACKET...
###[ IP ]###
version    = 4
ihl        = None
tos        = 0x0
len        = None
id         = 1
flags      = 
frag       = 0
ttl        = 64
proto      = icmp
checksum   = None
src        = 10.0.2.13
dst        = 192.168.29.153
\options   \
###[ ICMP ]###
type       = echo-request
code       = 0
checksum   = None
id         = 0x0
seq        = 0x0
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$
  
```

The Wireshark packet capture result on the victim:

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

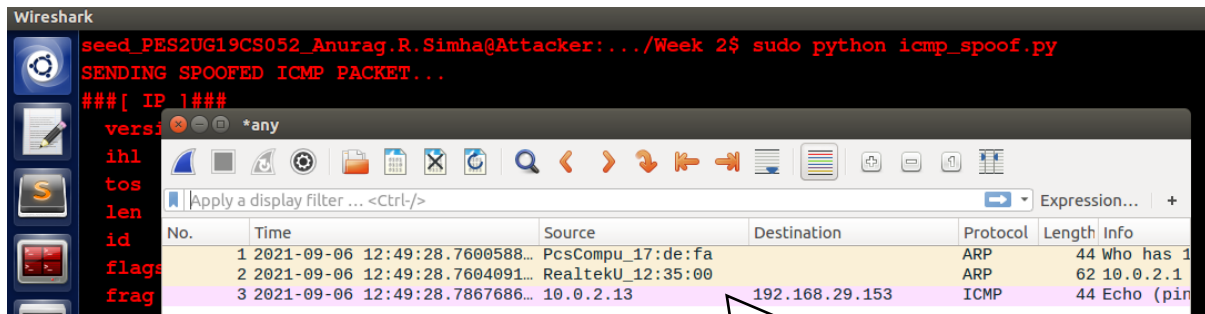
A.

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-09-06 12:35:38.2516463...	PcsCompu_17:de:fa	10.0.2.13	ARP	62	Who has 10.0.2.1? Tell 10.0.2.8
2	2021-09-06 12:35:38.2746319...	192.168.29.153	10.0.2.13	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, tt...
3	2021-09-06 12:35:38.2747555...	::1	::1	UDP	64	32975 → 37617 Len=0

Source	Destination	Proto
PcsCompu_17:de:fa		ARP
192.168.29.153	10.0.2.13	ICMP
::1	::1	UDP

The spoofed packet is hence delivered.

[This was done on the attacker machine:



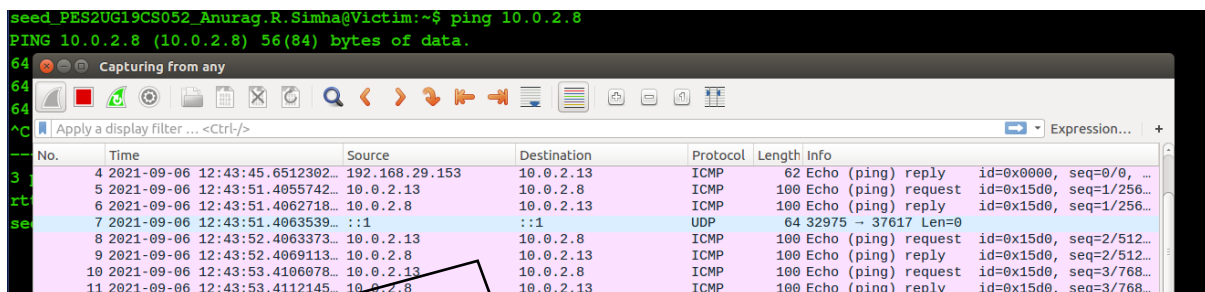
	Source	Destination	Protocol
12:49:28.7600588...	PcsCompu_17:de:fa	10.0.2.13	ARP
12:49:28.7604091...	RealtekU_12:35:00	10.0.2.13	ARP
12:49:28.7867686...	10.0.2.13	192.168.29.153	ICMP

]

Command (the victim): ping 10.0.2.8

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

A.



Source	Destination	Protocol
192.168.29.153	10.0.2.13	ICMP
10.0.2.13	10.0.2.8	ICMP
10.0.2.8	10.0.2.13	ICMP
:::1	:::1	UDP
10.0.2.13	10.0.2.8	ICMP
10.0.2.8	10.0.2.13	ICMP
10.0.2.13	10.0.2.8	ICMP
10.0.2.8	10.0.2.13	ICMP

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:

```

tracert.py
1  from scapy.all import *
2  '''Usage: ./traceroute.py " hostname or ip address"'''
3  host=sys.argv[1]
4  print("Traceroute "+ host)
5  ttl=1
6  while 1:
7      IPlayer=IP ()
8      IPlayer.dst=host
9      IPlayer.ttl=ttl
10     ICMPpkt=ICMP()
11     pkt=IPlayer/ICMPpkt
12     replypkt = sr1(pkt,verbose=0)
13     if replypkt is None:
14         break
15     elif replypkt [ICMP].type==0:
16         print "%d hops away: "%ttl, replypkt [IP].src
17         print "Done", replypkt [IP].src
18         break
19     else:
20         print "%d hops away: "%ttl, replypkt [IP].src
21         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`

```

Terminal
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$ sudo python traceroute.py 192.168.29.153
Traceroute 192.168.29.153
1 hops away: 10.0.2.1
2 hops away: 192.168.29.153
Done 192.168.29.153
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~/Week 2$

```

```

Traceroute 192.168.29.153
1 hops away: 10.0.2.1
2 hops away: 192.168.29.153
Done 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.

Source	Destination	Protocol	Length	Info
10.0.2.8	192.168.29.153	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=1 (no response found!)
10.0.2.1	10.0.2.8	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
10.0.2.8	192.168.29.153	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=2 (reply in 7)
192.168.29.153	10.0.2.8	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=127 (request in 6)

Source	Destination	Protocol
10.0.2.8	192.168.29.153	ICMP
10.0.2.1	10.0.2.8	ICMP
10.0.2.8	192.168.29.153	ICMP
192.168.29.153	10.0.2.8	ICMP

gth	Info
44	Echo (ping) request id=0x0000, seq=0/0, ttl=1 (no response found!)
72	Time-to-live exceeded (Time to live exceeded in transit)
44	Echo (ping) request id=0x0000, seq=0/0, ttl=2 (reply in 7)
62	Echo (ping) reply id=0x0000, seq=0/0, ttl=127 (request in 6)

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:

```

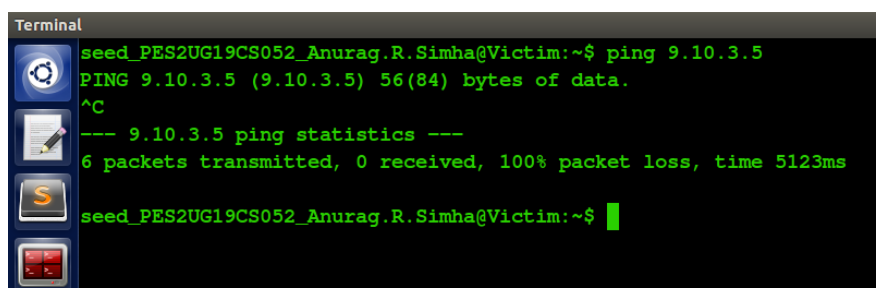
sniff+spoof.py
1  from scapy.all import *
2  def spoof_pkt(pkt):
3      newseq=0
4      if ICMP in pkt:
5          print("original packet.....")
6          print("source IP :", pkt[IP].src)
7          print("Destination IP :", pkt[IP].dst)
8          srcip = pkt[IP].dst
9          dstip = pkt[IP].src
10         newihl = pkt[IP].ihl
11         newtype = 0
12         newid = pkt[ICMP].id
13         newseq = pkt[ICMP].seq
14         data = pkt[Raw].load
15         IPLayer = IP (src=srcip, dst=dstip, ihl=newihl)
16         ICMPpkt = ICMP (type=newtype, id=newid, seq=newseq)
17         newpkt = IPLayer/ICMPpkt/data
18         print ("spoofed packet.....")
19         print ("Source IP:", newpkt[IP].src)
20         print ("Destination IP:", newpkt[IP].dst)
21         send (newpkt, verbose=0)
22  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

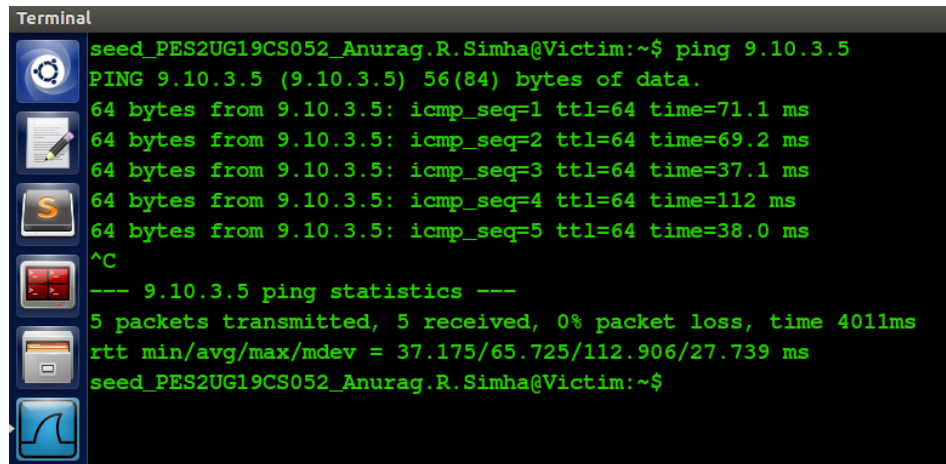


```

Terminal
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.
^C
--- 9.10.3.5 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5123ms
seed_PES2UG19CS052_Anurag.R.Simha@Victim:~$

```

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



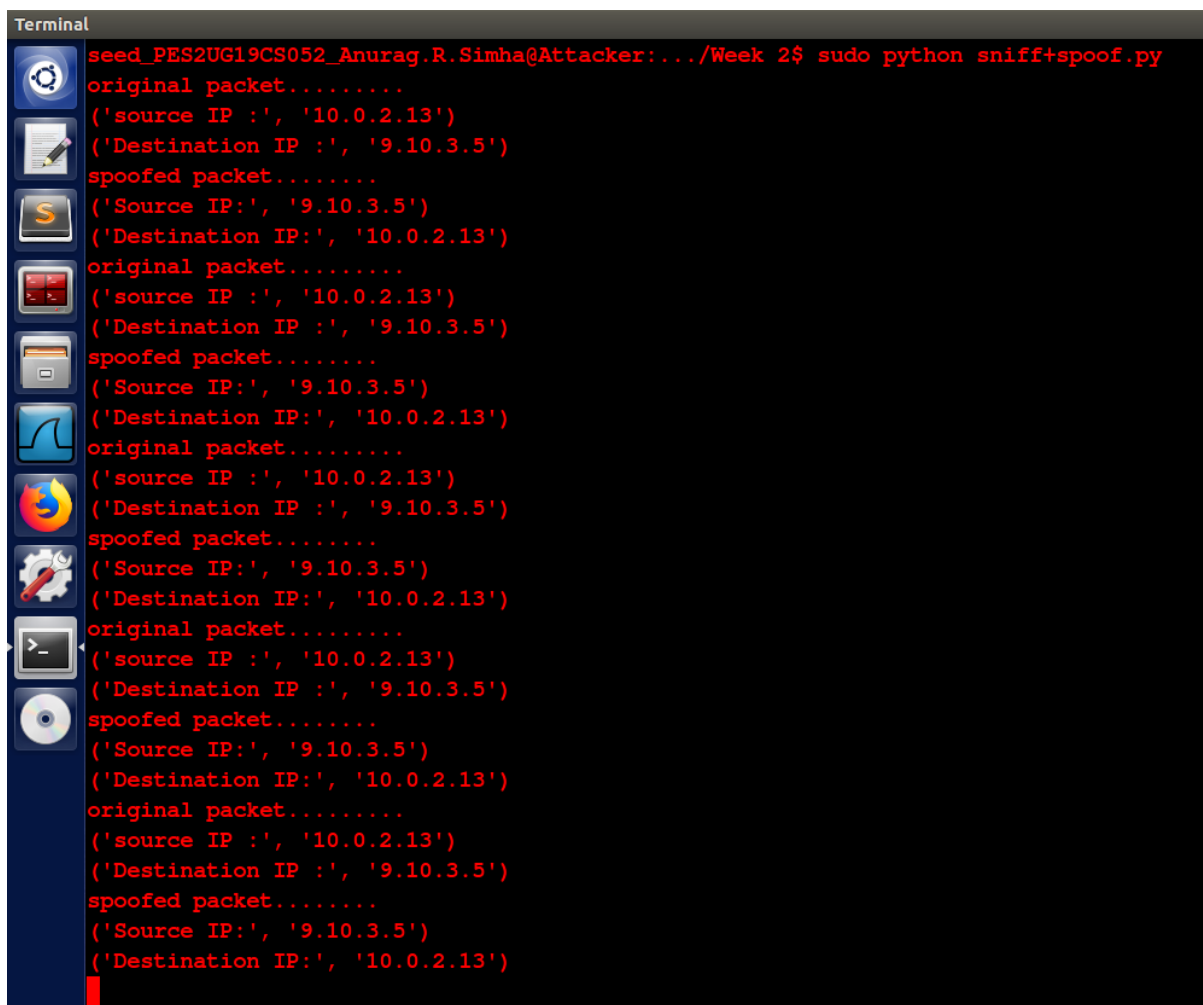
```

Terminal
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`



```

Terminal
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')
```


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
