

# **COMPUTER NETWORK**

## **SECURITY**

### **LAB REPORT**

#### **SNIFFING AND SPOOFING**

**NAME: VISHWAS M**

**SRN : PES2UG20CS390**

**SEC : F**

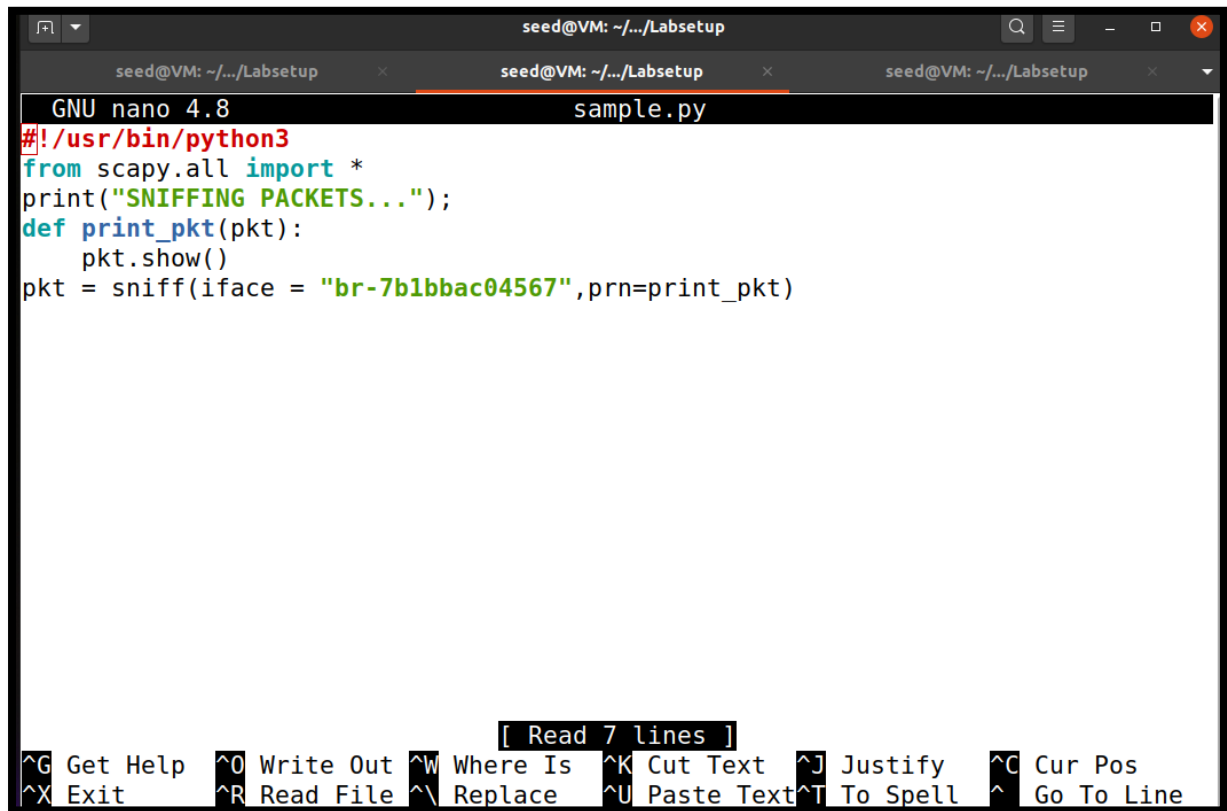
**DATE : 26/08/2022**

**LAB NO : 1**

## Task 1.1: Sniffing Packets

The objective of this task is to learn how to use Scapy to do packet sniffing in Python programs.

### Task 1.1A: Sniff IP packets using Scapy



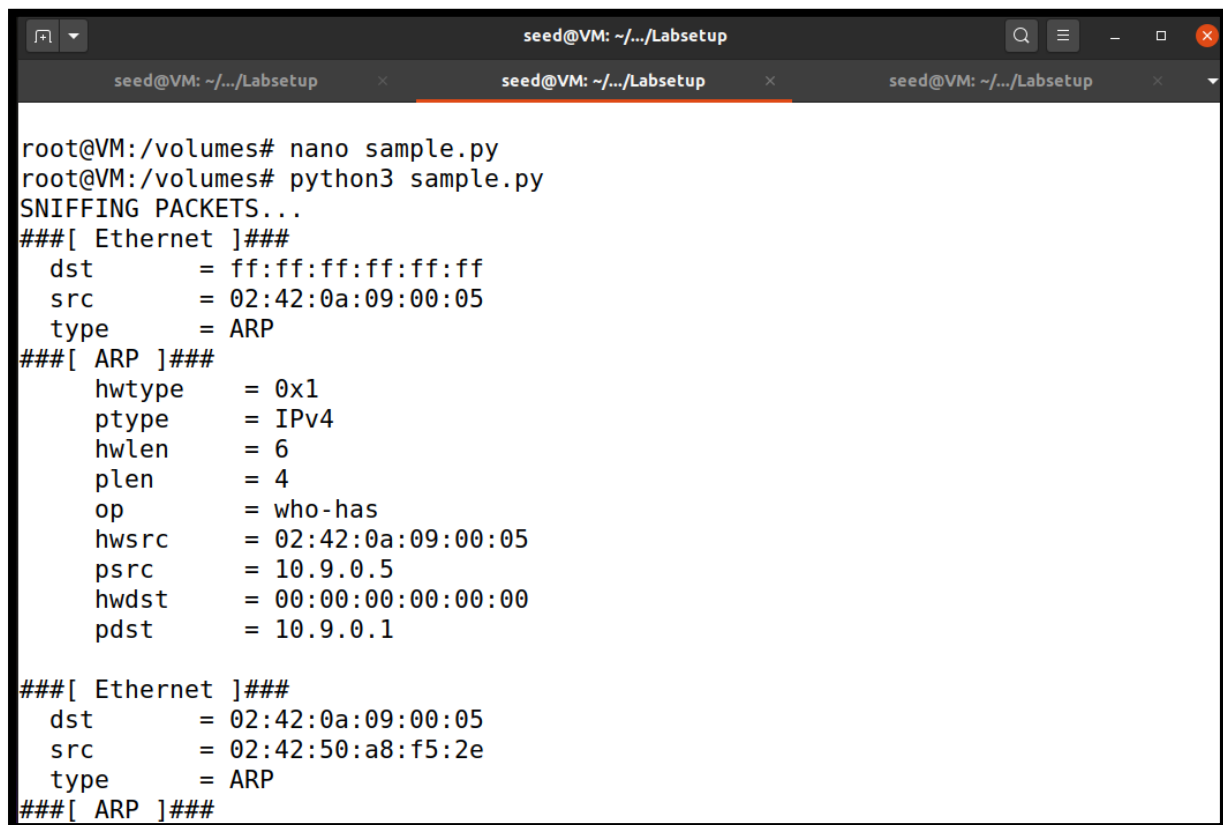
The screenshot shows a terminal window titled 'seed@VM: ~/.../Labsetup' with three tabs. The active tab is 'sample.py'. The terminal displays the GNU nano 4.8 editor with the following Python code:

```
#!/usr/bin/python3
from scapy.all import *
print("SNIFFING PACKETS...");
def print_pkt(pkt):
    pkt.show()
pkt = sniff(iface = "br-7b1bbac04567",prn=print_pkt)
```

At the bottom of the terminal, there is a status bar with the text '[ Read 7 lines ]' and a list of keyboard shortcuts: ^G Get Help, ^O Write Out, ^W Where Is, ^K Cut Text, ^J Justify, ^C Cur Pos, ^X Exit, ^R Read File, ^\ Replace, ^U Paste Text, ^T To Spell, and ^\_ Go To Line.

The program ,for each captured packet, the call-back function print\_pkt() will be invoked and this function will print out some of the information about the packet .

## Attacker Terminal:

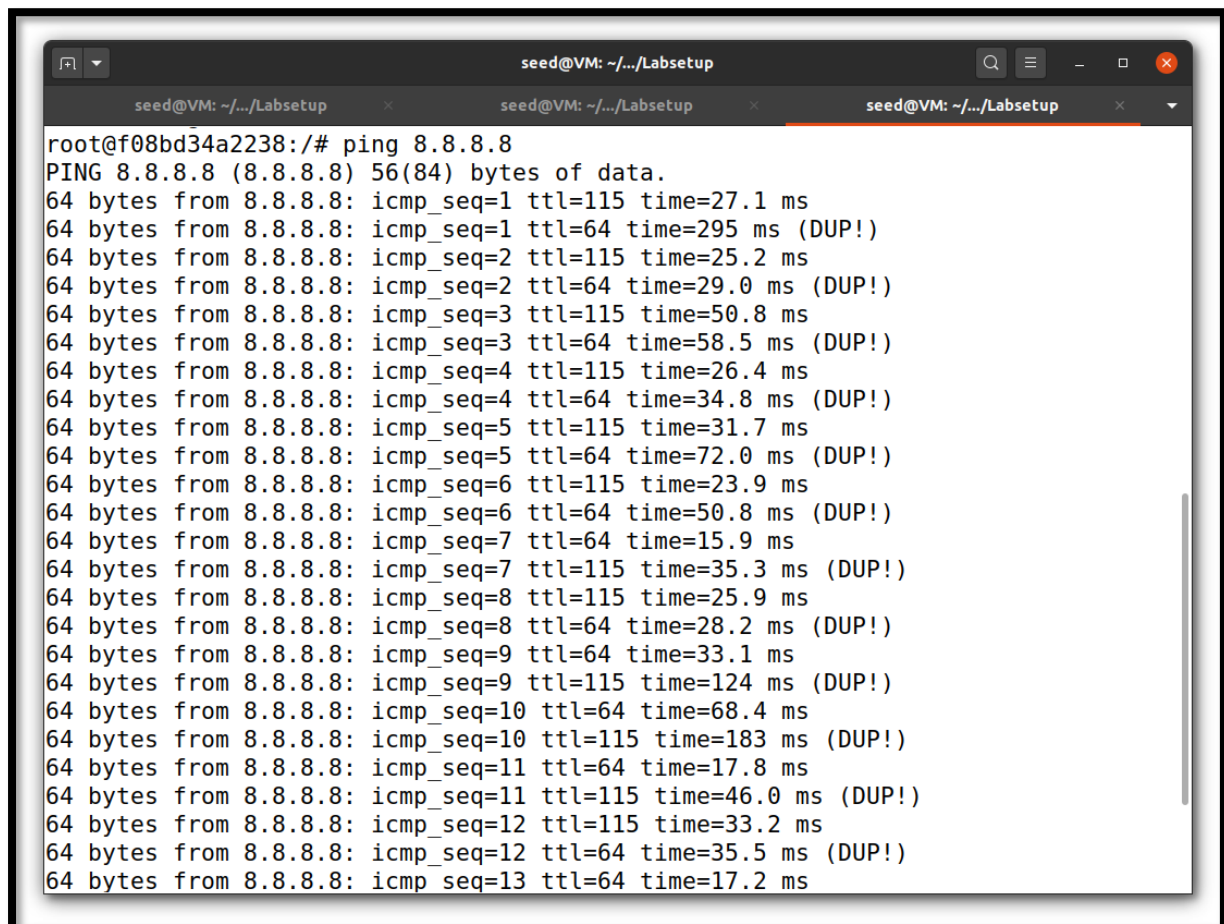
A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows a user running 'nano sample.py' and 'python3 sample.py'. The output displays sniffed network packets, including Ethernet and ARP details. The first packet is an ARP request from 10.9.0.5 to 10.9.0.1. The second packet is an ARP request from 02:42:50:a8:f5:2e to 02:42:0a:09:00:05.

```
seed@VM: ~/.../Labsetup
root@VM:/volumes# nano sample.py
root@VM:/volumes# python3 sample.py
SNIFFING PACKETS...
###[ Ethernet ]###
dst      = ff:ff:ff:ff:ff:ff
src      = 02:42:0a:09:00:05
type     = ARP
###[ ARP ]###
hwtype   = 0x1
ptype    = IPv4
hwlen    = 6
plen     = 4
op       = who-has
hwsrc    = 02:42:0a:09:00:05
psrc     = 10.9.0.5
hwdst    = 00:00:00:00:00:00
pdst     = 10.9.0.1

###[ Ethernet ]###
dst      = 02:42:0a:09:00:05
src      = 02:42:50:a8:f5:2e
type     = ARP
###[ ARP ]###
```

In the attacker terminal when we run the above-mentioned python code, it will print information about each and every packet it has sniffed when the host executed the command #ping 8.8.8.8.

## Host A Terminal:

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows a root user at a host with IP f08bd34a2238 running a 'ping 8.8.8.8' command. The output displays 13 ICMP echo requests, each with a sequence number, TTL, and round-trip time. Several responses are marked as 'DUP!' (duplicate).

```
root@f08bd34a2238:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=27.1 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=295 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=25.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=29.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=50.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=58.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=26.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=34.8 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=31.7 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=72.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=6 ttl=115 time=23.9 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=64 time=50.8 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=7 ttl=64 time=15.9 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=115 time=35.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=8 ttl=115 time=25.9 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=64 time=28.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=9 ttl=64 time=33.1 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=115 time=124 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=10 ttl=64 time=68.4 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=115 time=183 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=11 ttl=64 time=17.8 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=115 time=46.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=12 ttl=115 time=33.2 ms
64 bytes from 8.8.8.8: icmp_seq=12 ttl=64 time=35.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=13 ttl=64 time=17.2 ms
```

Here the host hanged pinged into a random ip address and the ICMP packets are created. These packets will get sniffed in the attacker's terminal and some information about these packets get printed in the terminal.

What happens if we run these commands without root privileges?

Ans: **The ICMP packets won't get sniffed in the attacker's terminal and some error occurs and the sniffing fails.**

```
seed@VM: ~/.../Labsetup
load      = '\x96\x96\x08c\x00\x00\x00\x00p\xfd\x08\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./01234567'

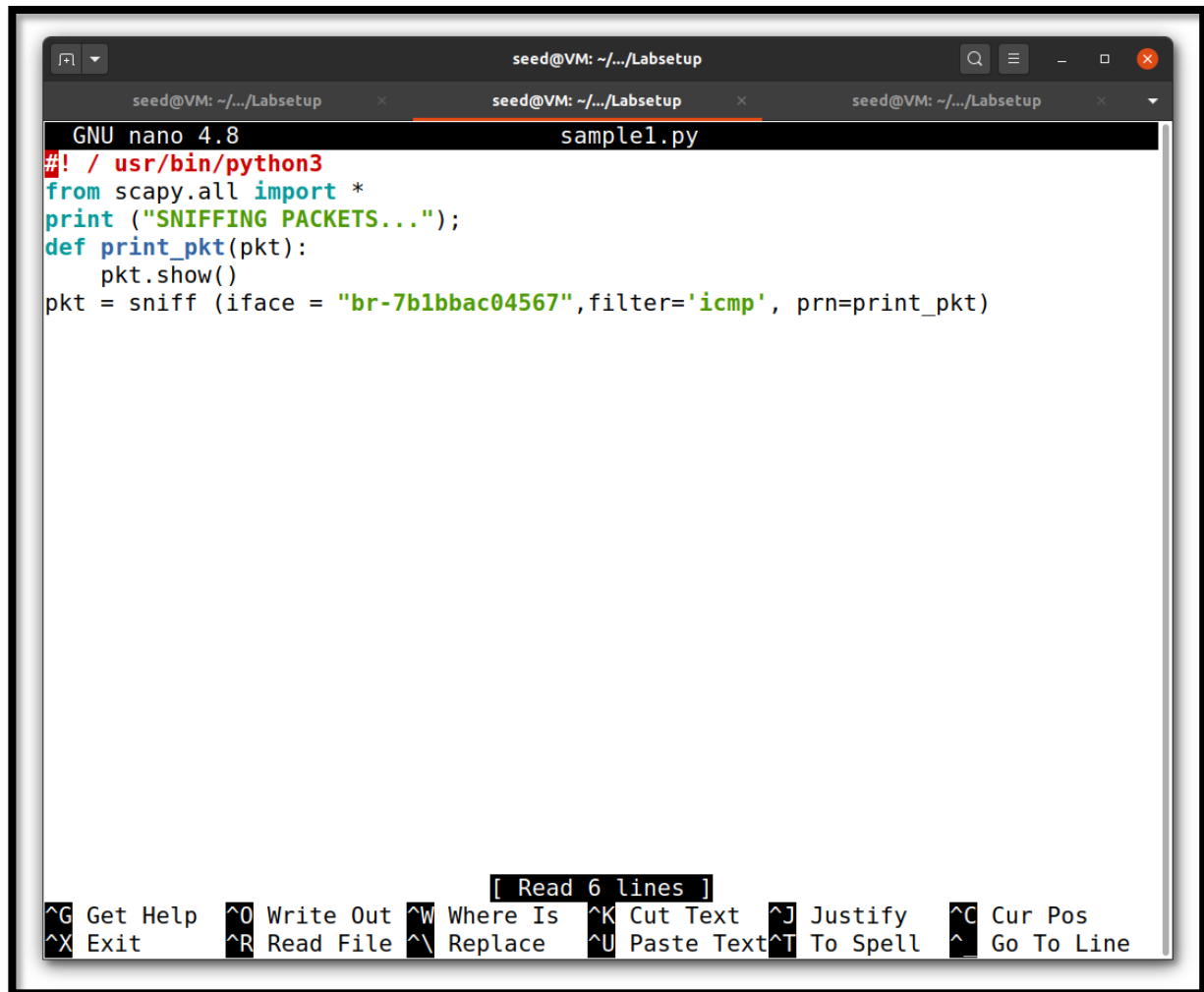
^Croot@VM:/volumes# su seed
seed@VM:/volumes$ python3 sample.py
SNIFFING PACKETS...
Traceback (most recent call last):
  File "sample.py", line 6, in <module>
    pkt = sniff(iface = "br-7b1bbac04567",prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 906, in run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type)) # noqa: E501
  File "/usr/lib/python3.8/socket.py", line 231, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
seed@VM:/volumes$
```

## Task 1.1B: Capturing ICMP, TCP packet and Subnet

### ICMP:

The ICMP packets are captured by the Scapy sniffer program. Hence, when some machine on the same network sends ping requests, the packets get captured by the sniffer.

## Python code:

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows the GNU nano 4.8 editor editing a file named 'sample1.py'. The code in the file is a Python script that imports scapy, prints a message, and sets up a sniff on the 'br-7b1bbac04567' interface for ICMP traffic. The script uses a custom 'print\_pkt' function to display packet details. The bottom of the terminal shows the nano editor's status bar with various keyboard shortcuts like '^G Get Help', '^O Write Out', etc.

```
seed@VM: ~/.../Labsetup
GNU nano 4.8 sample1.py
#!/usr/bin/python3
from scapy.all import *
print ("SNIFFING PACKETS...");
def print_pkt(pkt):
    pkt.show()
pkt = sniff (iface = "br-7b1bbac04567",filter='icmp', prn=print_pkt)
```

We provided the filter as ICMP inside the sniff command which indicates that only ICMP packets will get sniffed in the attacker system. The pkt() function prints some information about the sniffed ICMP packet.

## Attacker's terminal:

Here we can see that information of the ICMP packet has been displayed in the attacker's terminal. We can observe that under protocol it is mentioned as ICMP which tells that we have captured the right packets or sniffed the right packets. The source and destination address is also mentioned in the given information of a particular ICMP packet.

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

###[ Ethernet ]###
dst      = 02:42:50:a8:f5:2e
src      = 02:42:0a:09:00:05
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 26
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0x2072
src      = 10.9.0.5
dst      = 8.8.8.8
\options \
###[ ICMP ]###
type     = echo-request
code     = 0
chksum   = 0xfa88
id       = 0x1e
seq      = 0xf
###[ Raw ]###
load     = 'M\x98\x08c\x00\x00\x00\x00\xe0{\x08\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$$%&\'()*+,-./01234567'
```

## Host A terminal:

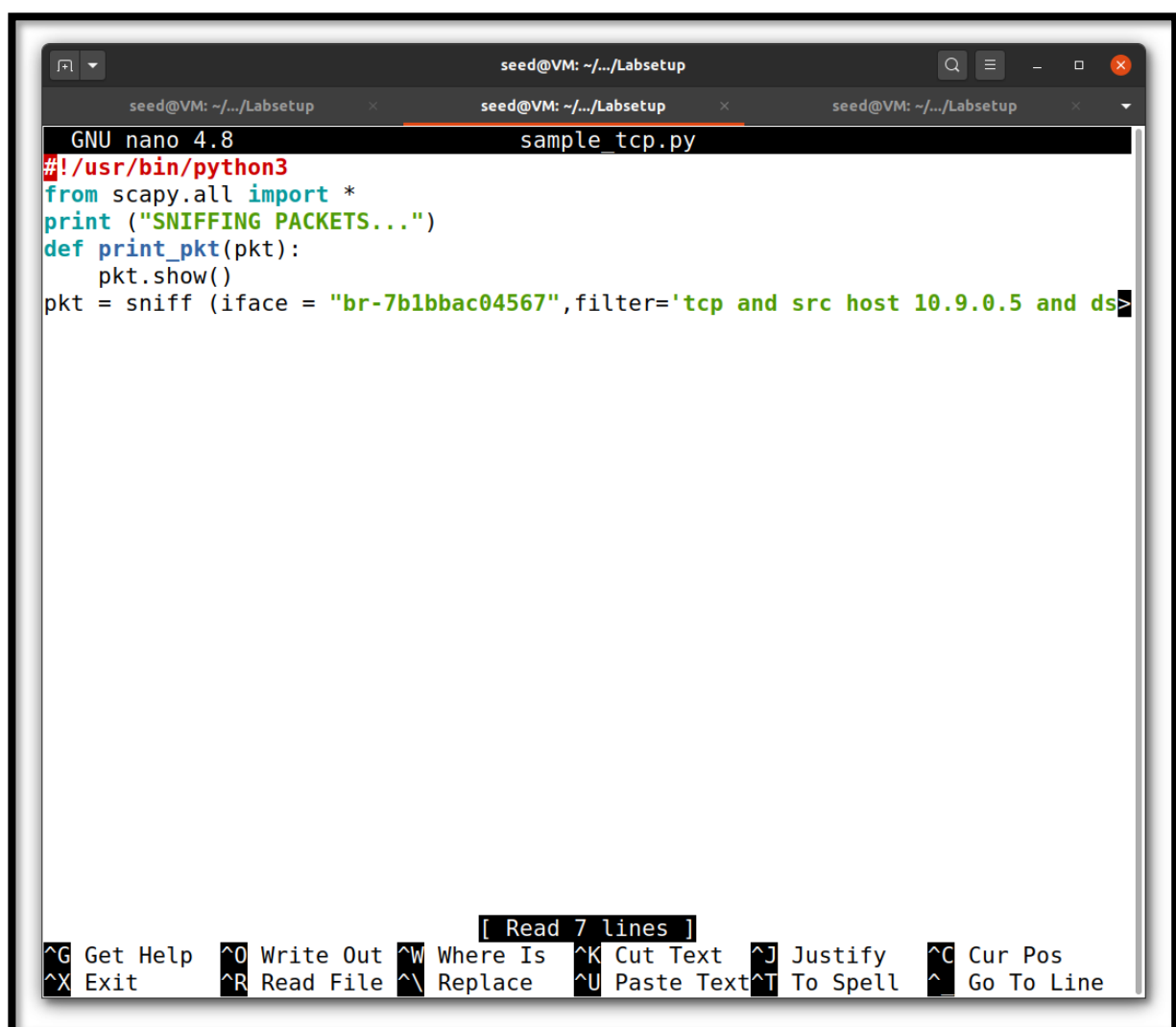
```
root@f08bd34a2238:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=55 time=501 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=55 time=208 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=55 time=140 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=55 time=258 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=55 time=281 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=55 time=95.5 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=55 time=219 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=55 time=242 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=55 time=162 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=55 time=82.7 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=55 time=831 ms
```

As we can see the host has pinged a random IP address and we have considered 8.8.8.8.

## TCP:

The TCP packets are captured by the Scapy sniffer program. Hence, when some machine on the same network sends ping requests, the packets get captured by the sniffer.

### Python code:

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The window shows the GNU nano 4.8 text editor editing a file named 'sample\_tcp.py'. The code in the editor is a Python script using Scapy to sniff network packets. It imports from scapy.all, prints a message, defines a function 'print\_pkt' to show packet details, and then uses the 'sniff' function to capture packets on interface 'br-7b1bbac04567' with a filter for TCP packets from host 10.9.0.5. The terminal window has a standard Linux-style window manager with search, menu, and window control buttons in the title bar. At the bottom, there is a status bar with various keyboard shortcuts for nano editor operations like 'Get Help', 'Write Out', 'Where Is', 'Cut Text', 'Justify', 'Cur Pos', 'Exit', 'Read File', 'Replace', 'Paste Text', 'To Spell', and 'Go To Line'.

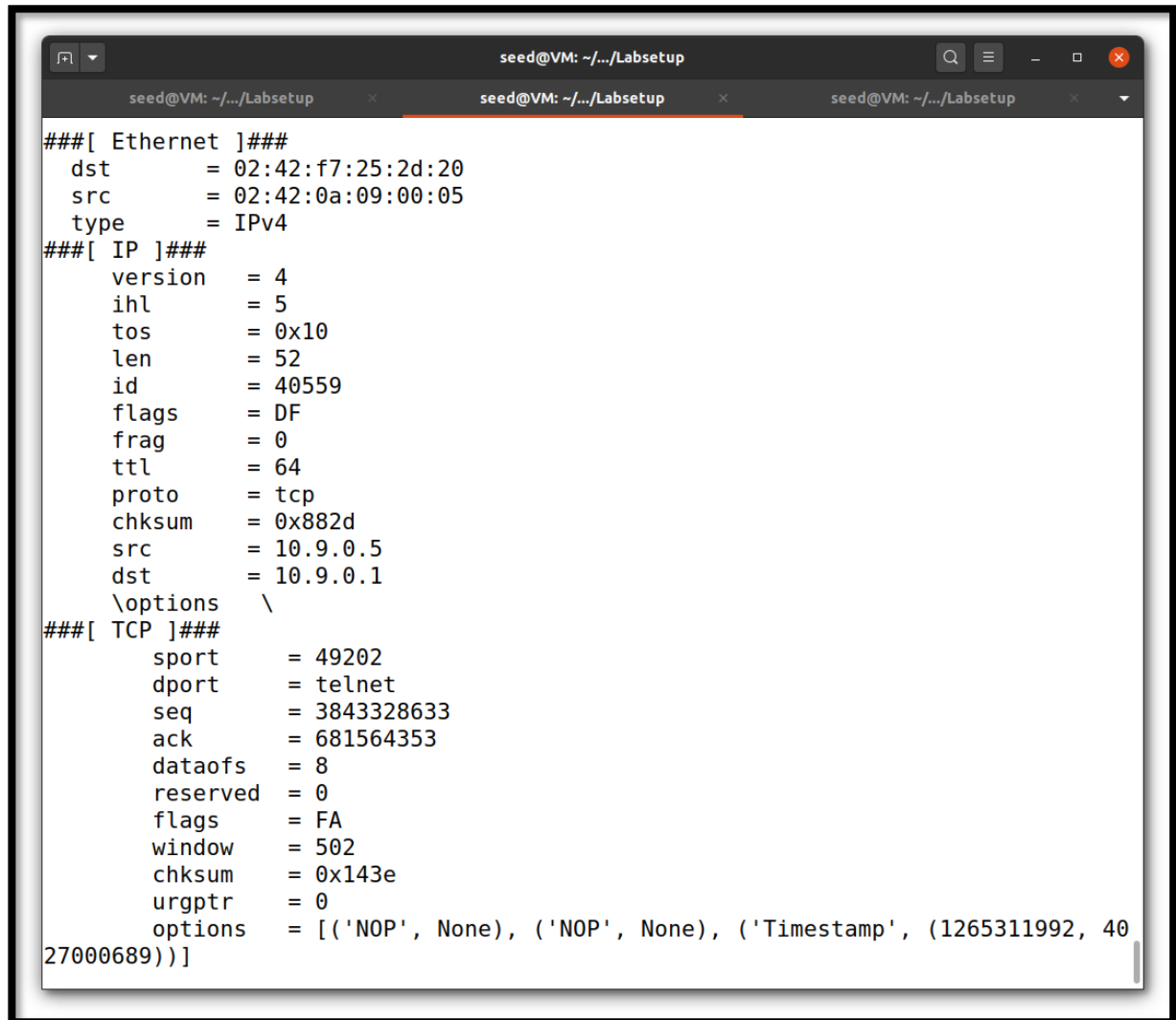
```
seed@VM: ~/.../Labsetup
GNU nano 4.8 sample_tcp.py
#!/usr/bin/python3
from scapy.all import *
print ("SNIFFING PACKETS...")
def print_pkt(pkt):
    pkt.show()
pkt = sniff (iface = "br-7b1bbac04567",filter='tcp and src host 10.9.0.5 and ds>
```

We provided the filter as TCP and also mentioned that the packet should be sent only by the host 10.9.0.5 inside the sniff command which indicates that only ICMP packets will get sniffed in the attacker system. The pkt() function prints some information about



the sniffed TCP packet.

### Attacker's terminal:

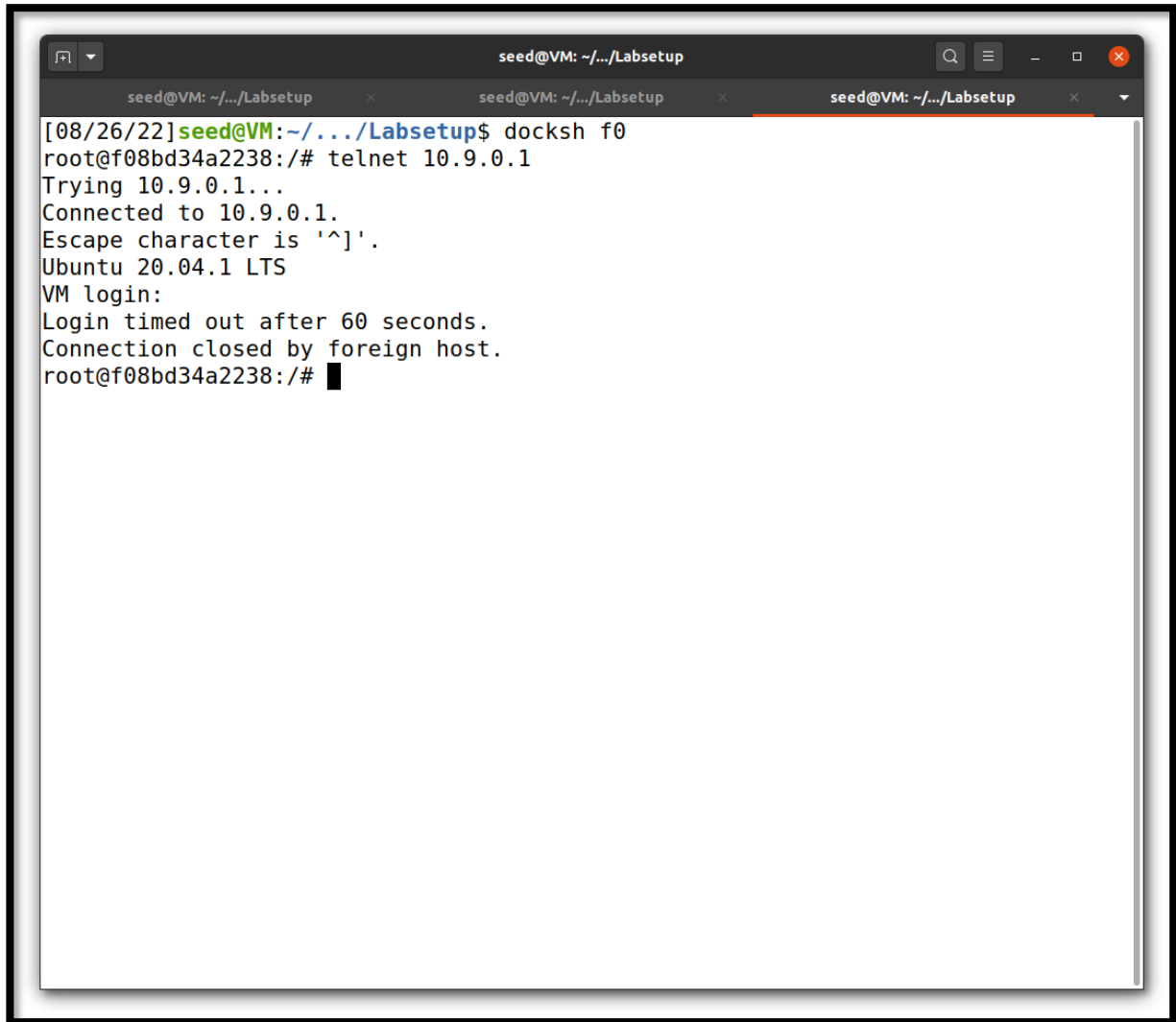
A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal displays the details of a captured network packet, organized into three sections: Ethernet, IP, and TCP. The Ethernet section shows the destination (dst) as 02:42:f7:25:2d:20, the source (src) as 02:42:0a:09:00:05, and the type as IPv4. The IP section shows version 4, header length (ihl) 5, total length (len) 52, identification (id) 40559, flags DF, fragmentation offset (frag) 0, time-to-live (ttl) 64, protocol (proto) tcp, checksum 0x882d, source address (src) 10.9.0.5, and destination address (dst) 10.9.0.1. The TCP section shows source port (sport) 49202, destination port (dport) telnet, sequence number (seq) 3843328633, acknowledgment number (ack) 681564353, data offset (dataofs) 8, reserved bits 0, flags FA, window size 502, checksum 0x143e, urgent pointer (urgptr) 0, and options including NOP, NOP, and a Timestamp with values (1265311992, 4027000689).

```
seed@VM: ~/.../Labsetup
###[ Ethernet ]###
  dst      = 02:42:f7:25:2d:20
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x10
  len      = 52
  id       = 40559
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  chksum   = 0x882d
  src      = 10.9.0.5
  dst      = 10.9.0.1
  \options \
###[ TCP ]###
  sport    = 49202
  dport    = telnet
  seq      = 3843328633
  ack      = 681564353
  dataofs  = 8
  reserved = 0
  flags    = FA
  window   = 502
  chksum   = 0x143e
  urgptr   = 0
  options  = [('NOP', None), ('NOP', None), ('Timestamp', (1265311992, 4027000689))]
```

Here we can see that information of the TCP packet has been displayed in the attacker's terminal. We can observe that under protocol it is mentioned as TCP which tells that we have captured the right packets or sniffed the right packets. We can also see that all the TCP packets that are sniffed have the source IP address as 10.9.0.5 as indicated in the code.

### Host A terminal:

Here we are capturing all the TCP packet that comes from a particular IP and with a destination port number 23. So we have used the telnet command with IP address 10.9.0.1.

A terminal window titled 'seed@VM: ~/.../Labsetup' with three tabs. The active tab shows the following text:

```
[08/26/22]seed@VM:~/.../Labsetup$ docksh f0
root@f08bd34a2238:/# telnet 10.9.0.1
Trying 10.9.0.1...
Connected to 10.9.0.1.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
VM login:
Login timed out after 60 seconds.
Connection closed by foreign host.
root@f08bd34a2238:/#
```

## Subnet:

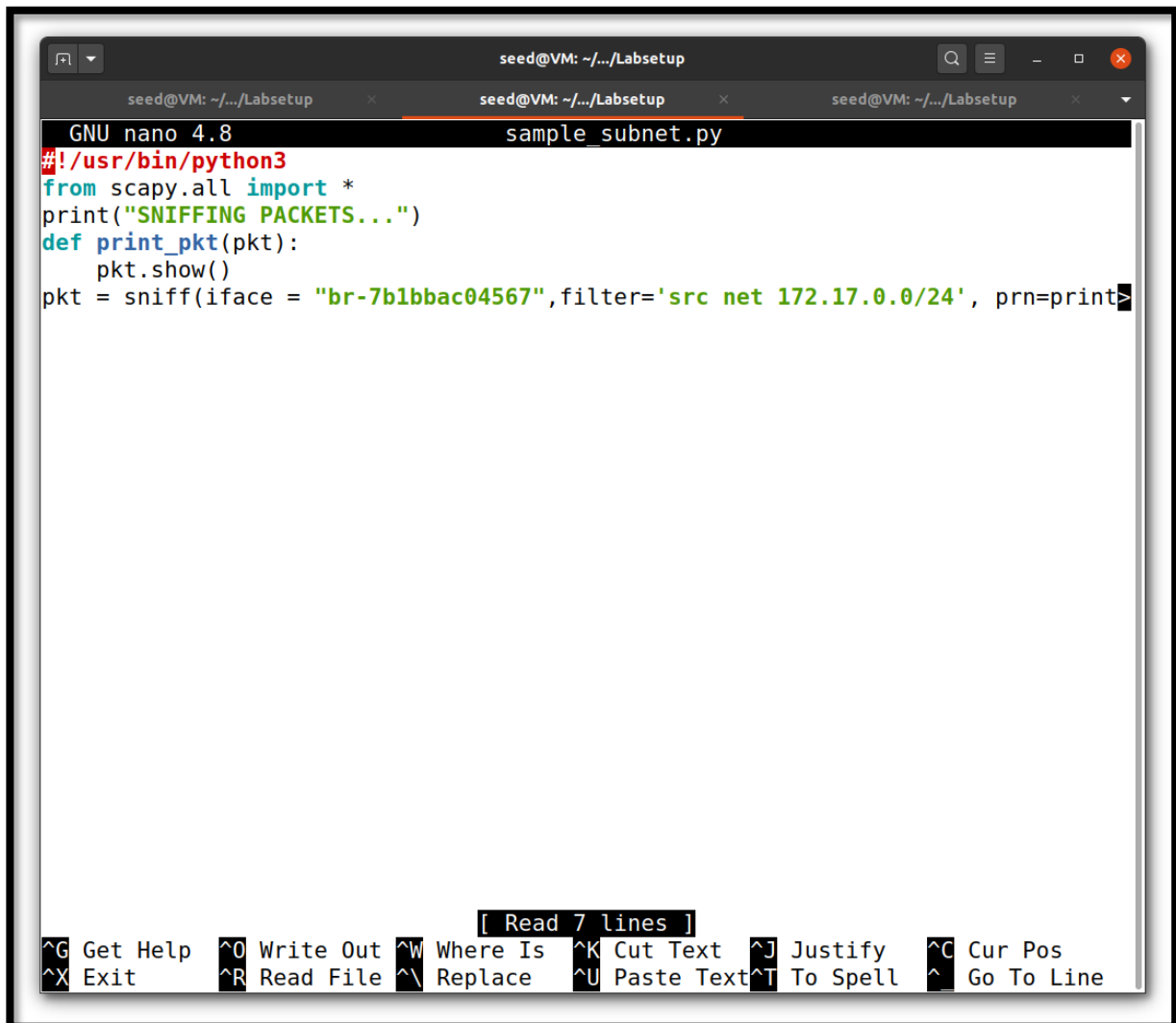
You can pick any subnet, such as 192.168.254.0/24; you should not pick the subnet that your VM is attached to.

Show that on sending ICMP packets to 192.168.254.1, the sniffer program captures the packets sent out from 192.168.254.1 .

## Python code:

In this example we have chosen the subnet IP address as

172.17.0.0/24 as mentioned in the filter. The pkt() function prints all the ICMP packets which are sent out from any random IP address chosen from the above subnet address.



The screenshot shows a terminal window titled 'seed@VM: ~/.../Labsetup'. Inside, the GNU nano 4.8 editor is open, editing a file named 'sample\_subnet.py'. The script content is as follows:

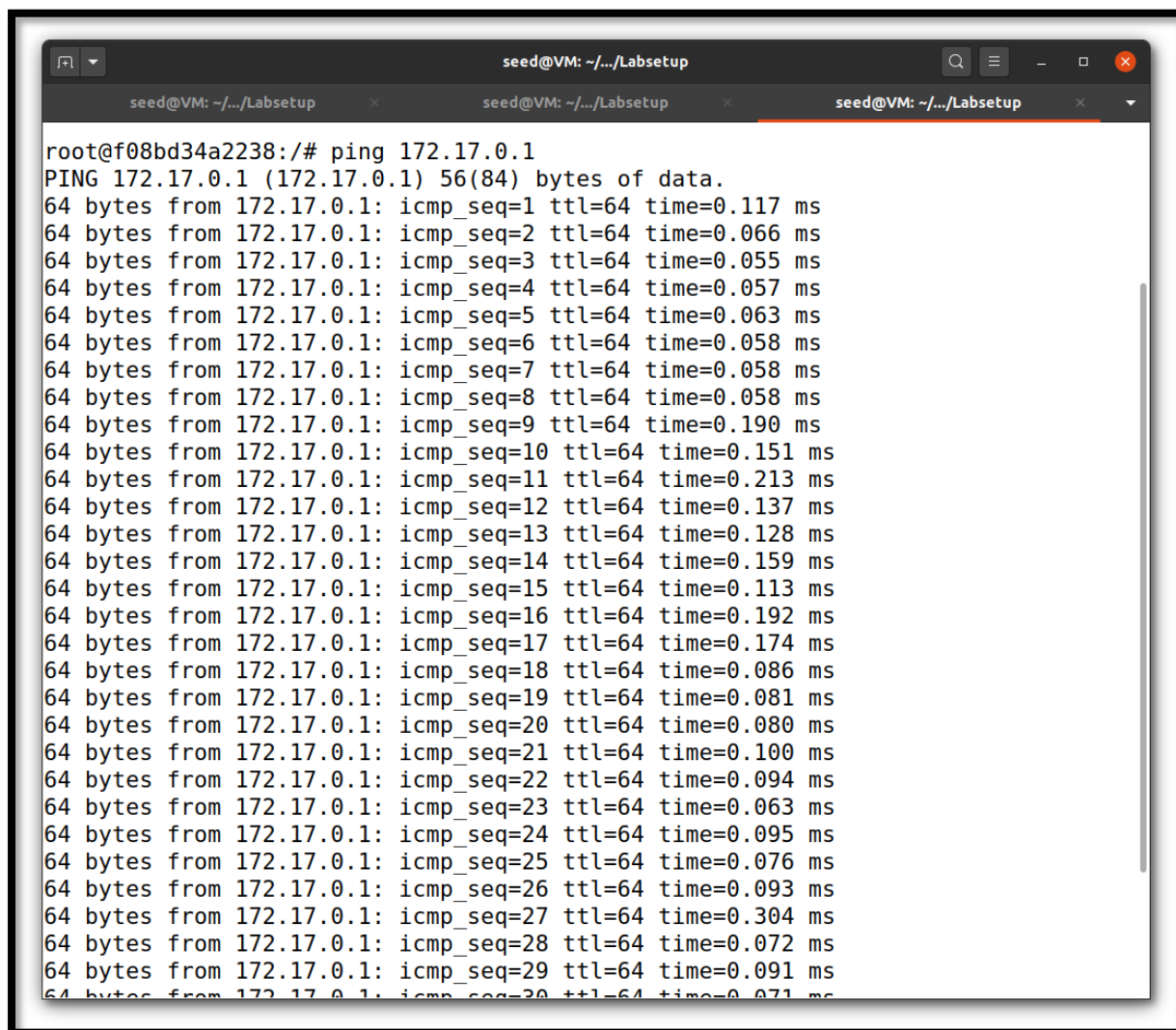
```
#!/usr/bin/python3
from scapy.all import *
print("SNIFFING PACKETS...")
def print_pkt(pkt):
    pkt.show()
pkt = sniff(iface = "br-7b1bbac04567", filter='src net 172.17.0.0/24', prn=print)
```

The bottom of the terminal displays the nano editor's help menu:

<b>^G</b> Get Help	<b>^O</b> Write Out	<b>^W</b> Where Is	<b>^K</b> Cut Text	<b>^J</b> Justify	<b>^C</b> Cur Pos
<b>^X</b> Exit	<b>^R</b> Read File	<b>^_\</b> Replace	<b>^U</b> Paste Text	<b>^T</b> To Spell	<b>^_</b> Go To Line

### Host A's Terminal:

In this example we have chosen a random IP address from the above-mentioned subnet (172.17.0.0/24) as 172.17.0.0.

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows a user running a ping command to 172.17.0.1. The output displays 30 ICMP echo requests, each with a sequence number, TTL of 64, and a response time. The response times vary, with most between 0.055 ms and 0.113 ms, and a few outliers at 0.117 ms, 0.151 ms, 0.190 ms, 0.192 ms, and 0.304 ms. The terminal window has three tabs, all with the same title, and standard window controls at the top.

```
root@f08bd34a2238:/# ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1) 56(84) bytes of data.
64 bytes from 172.17.0.1: icmp_seq=1 ttl=64 time=0.117 ms
64 bytes from 172.17.0.1: icmp_seq=2 ttl=64 time=0.066 ms
64 bytes from 172.17.0.1: icmp_seq=3 ttl=64 time=0.055 ms
64 bytes from 172.17.0.1: icmp_seq=4 ttl=64 time=0.057 ms
64 bytes from 172.17.0.1: icmp_seq=5 ttl=64 time=0.063 ms
64 bytes from 172.17.0.1: icmp_seq=6 ttl=64 time=0.058 ms
64 bytes from 172.17.0.1: icmp_seq=7 ttl=64 time=0.058 ms
64 bytes from 172.17.0.1: icmp_seq=8 ttl=64 time=0.058 ms
64 bytes from 172.17.0.1: icmp_seq=9 ttl=64 time=0.190 ms
64 bytes from 172.17.0.1: icmp_seq=10 ttl=64 time=0.151 ms
64 bytes from 172.17.0.1: icmp_seq=11 ttl=64 time=0.213 ms
64 bytes from 172.17.0.1: icmp_seq=12 ttl=64 time=0.137 ms
64 bytes from 172.17.0.1: icmp_seq=13 ttl=64 time=0.128 ms
64 bytes from 172.17.0.1: icmp_seq=14 ttl=64 time=0.159 ms
64 bytes from 172.17.0.1: icmp_seq=15 ttl=64 time=0.113 ms
64 bytes from 172.17.0.1: icmp_seq=16 ttl=64 time=0.192 ms
64 bytes from 172.17.0.1: icmp_seq=17 ttl=64 time=0.174 ms
64 bytes from 172.17.0.1: icmp_seq=18 ttl=64 time=0.086 ms
64 bytes from 172.17.0.1: icmp_seq=19 ttl=64 time=0.081 ms
64 bytes from 172.17.0.1: icmp_seq=20 ttl=64 time=0.080 ms
64 bytes from 172.17.0.1: icmp_seq=21 ttl=64 time=0.100 ms
64 bytes from 172.17.0.1: icmp_seq=22 ttl=64 time=0.094 ms
64 bytes from 172.17.0.1: icmp_seq=23 ttl=64 time=0.063 ms
64 bytes from 172.17.0.1: icmp_seq=24 ttl=64 time=0.095 ms
64 bytes from 172.17.0.1: icmp_seq=25 ttl=64 time=0.076 ms
64 bytes from 172.17.0.1: icmp_seq=26 ttl=64 time=0.093 ms
64 bytes from 172.17.0.1: icmp_seq=27 ttl=64 time=0.304 ms
64 bytes from 172.17.0.1: icmp_seq=28 ttl=64 time=0.072 ms
64 bytes from 172.17.0.1: icmp_seq=29 ttl=64 time=0.091 ms
64 bytes from 172.17.0.1: icmp_seq=30 ttl=64 time=0.071 ms
```

### **Attacker's Terminal:**

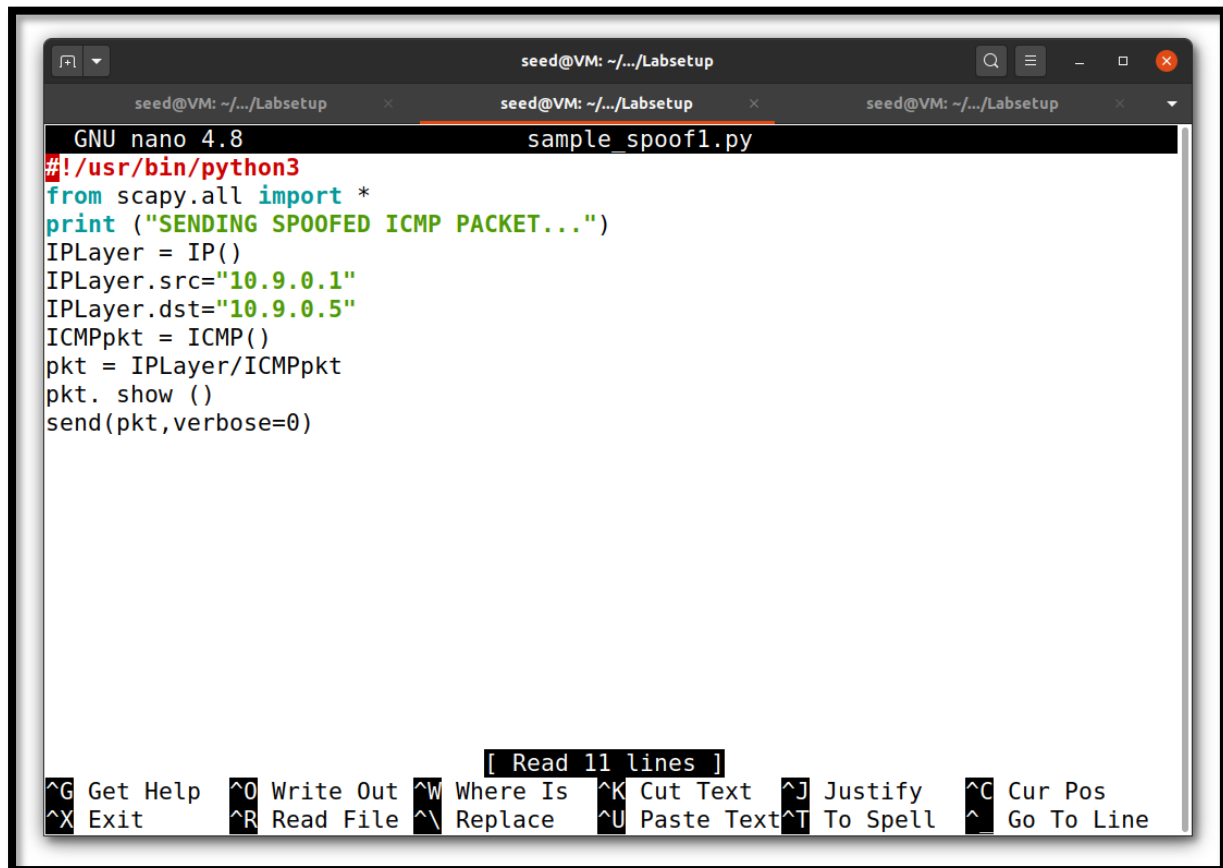
Here we can see that information of the ICMP packet has been displayed in the attacker's terminal. We can observe that all the ICMP packets are sniffed which was sent from the IP address which comes under the subnet address mentioned in the python code mentioned above.

```
seed@VM: ~/.../Labsetup
root@VM:/volumes# python3 sample_subnet.py
SNIFFING PACKETS...
###[ Ethernet ]###
  dst      = 02:42:0a:09:00:05
  src      = 02:42:f7:25:2d:20
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 1765
  flags    =
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0xbda4
  src      = 172.17.0.1
  dst      = 10.9.0.5
  \options \
###[ ICMP ]###
  type     = echo-reply
  code     = 0
  chksum   = 0x166b
  id       = 0x1e
  seq      = 0x9
###[ Raw ]###
  load     = '\x08\xa3\x08c\x00\x00\x00\x00\x0b\x95\x0e\x00\x00\x00\x0
0\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&'()*
*+,-./01234567'
###[ Ethernet ]###
```

## Task 1.2: SPOOFING

The objective of this task is to spoof IP packets with an arbitrary source IP address. We will spoof ICMP echo request packets and send them to another VM on the same network. We will use Wireshark to observe whether our request will be accepted by the receiver. If it is accepted, an echo reply packet will be sent to the spoofed IP address. Below shows the code to create the ICMP packet. The spoofed request is formed by creating our own packet with the header specifications. Similarly, we fill the IP header with source IP address of any machine within the local network and destination IP address of any remote machine on the internet which is alive.

## Python code:



The screenshot shows a terminal window with a nano editor. The title bar indicates the user is 'seed@VM' in the directory '~/.../Labsetup'. The editor is editing a file named 'sample\_spoof1.py'. The code in the editor is as follows:

```
GNU nano 4.8 sample_spoof1.py
#!/usr/bin/python3
from scapy.all import *
print ("SENDING SPOOFED ICMP PACKET...")
IPLayer = IP()
IPLayer.src="10.9.0.1"
IPLayer.dst="10.9.0.5"
ICMPpkt = ICMP()
pkt = IPLayer/ICMPpkt
pkt.show()
send(pkt,verbose=0)
```

At the bottom of the terminal, there is a status bar with the text '[ Read 11 lines ]' and a list of keyboard shortcuts: ^G Get Help, ^O Write Out, ^W Where Is, ^K Cut Text, ^J Justify, ^C Cur Pos, ^X Exit, ^R Read File, ^\ Replace, ^U Paste Text, ^T To Spell, ^\_ Go To Line.

Above code is used to create the ICMP packet. The source and the Destination IP address is mentioned in the above code. The ICMP() indicates that this is an ICMP packet. The pkt() function prints some information about the sniffed packet. The spoofed request is formed by creating our own packet with the header specifications.

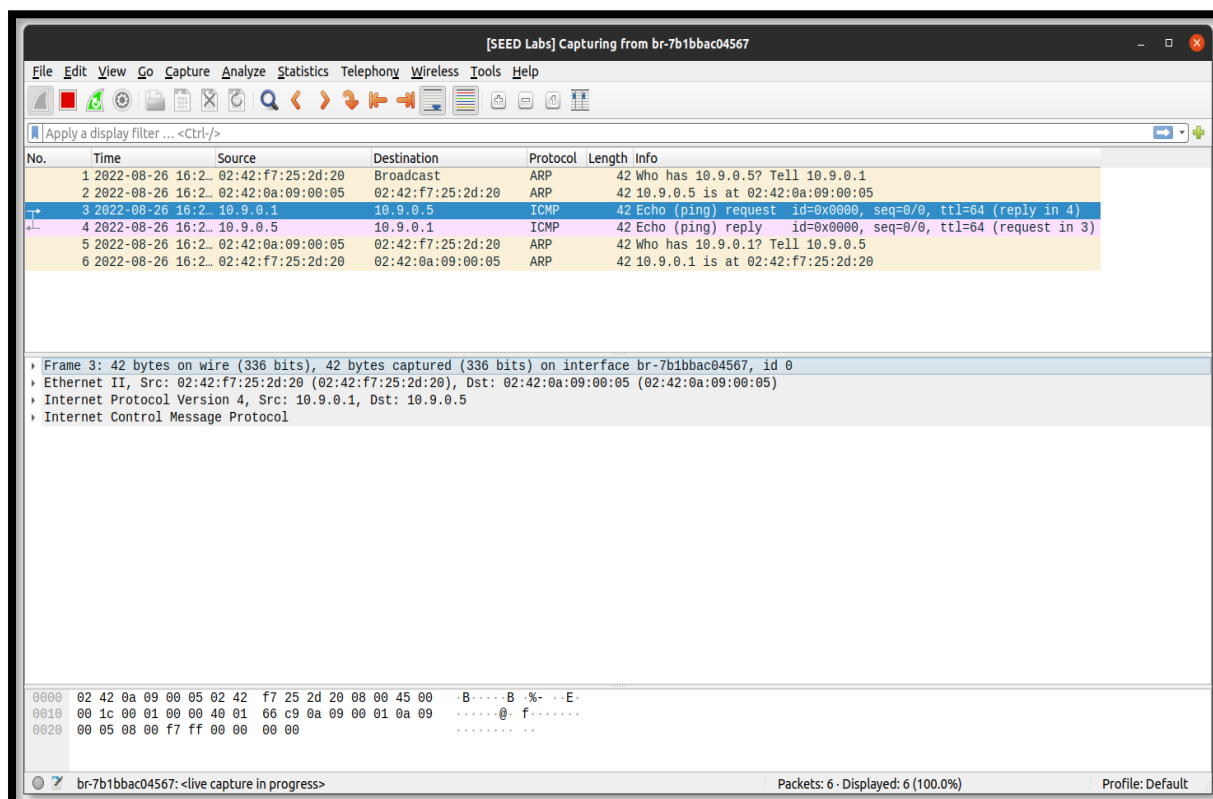
## Attacker's Terminal:

Here we have created an ICMP packet and spoofed it. Then we are sending this spoofed ICMP packet back to the destination IP address. As we can see that this is a request which is being sniffed and spoofed and then it is being sent to the destination address.

```
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
root@VM:/volumes# nano sample_spoof2.py
root@VM:/volumes# python3 sample_spoof1.py
SENDING SPOOFED ICMP PACKET...
###[ IP ]###
version      = 4
ihl          = None
tos          = 0x0
len          = None
id           = 1
flags        =
frag         = 0
ttl          = 64
proto        = icmp
chksum       = None
src          = 10.9.0.1
dst          = 10.9.0.5
\options     \
###[ ICMP ]###
type         = echo-request
code         = 0
chksum       = None
id           = 0x0
seq          = 0x0

root@VM:/volumes#
```

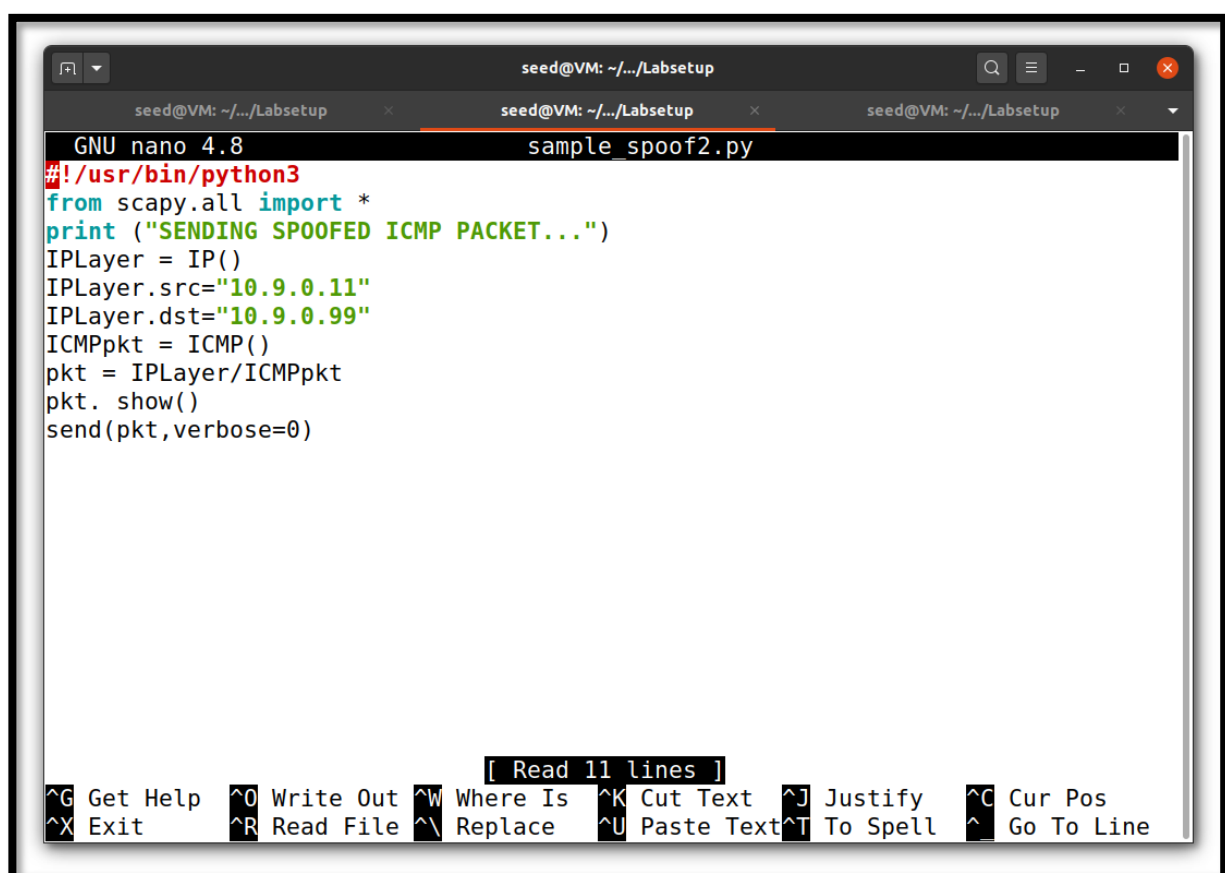
## Wireshark Screenshot:



We can observe that the spoofed ICMP packet has reached the destination IP address and the reply is sent back to the source IP address by indicating that it has received the ICMP packet.

**Demonstrate that you can spoof an ICMP echo request packet with an arbitrary source IP address. Open Wireshark and observe the ICMP packets as they are being captured.**

### Python code:

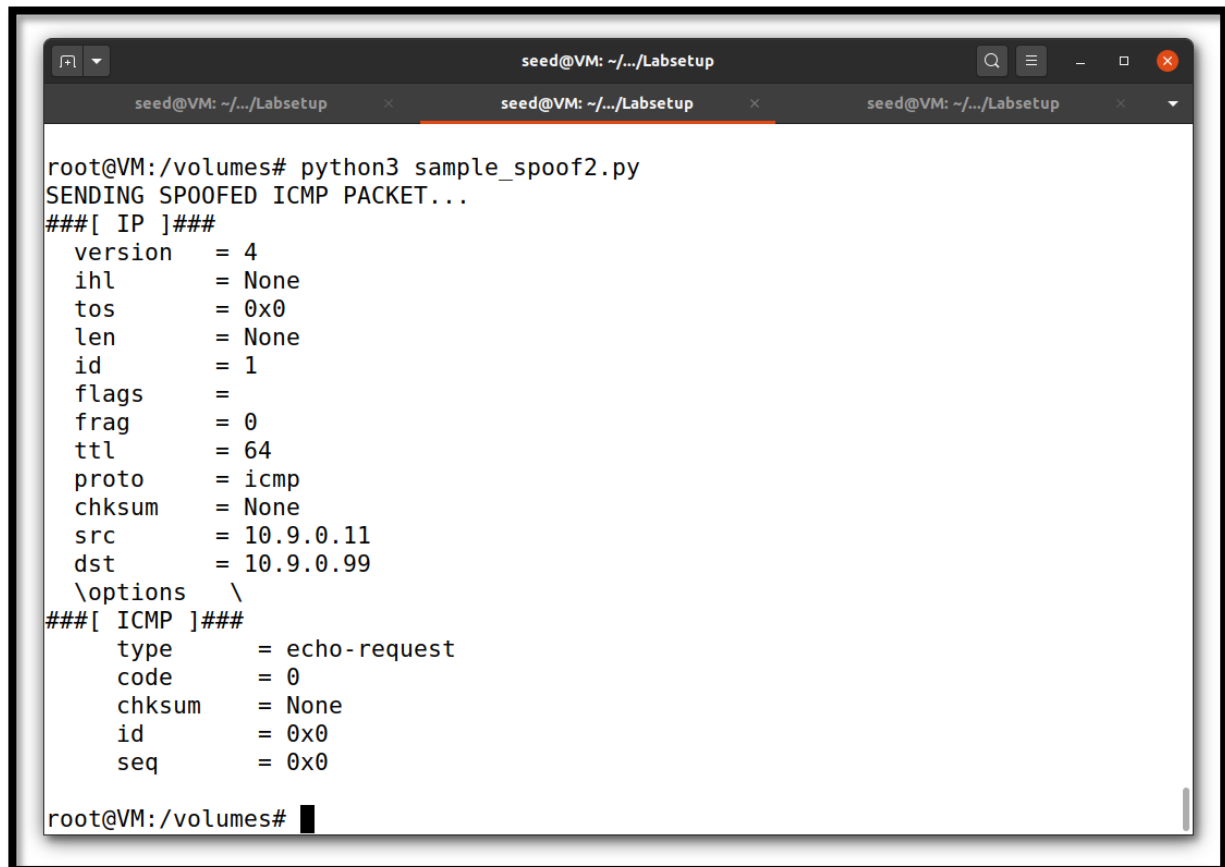
A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows the GNU nano 4.8 editor with a file named 'sample\_spoof2.py'. The code is a Python script that uses the Scapy library to create and send a spoofed ICMP packet. The source IP is set to '10.9.0.11' and the destination IP is '10.9.0.99'. The script prints 'SENDING SPOOFED ICMP PACKET...' before sending the packet. The bottom of the terminal shows the nano editor's command palette with various shortcuts like ^G for Get Help, ^O for Write Out, etc.

```
seed@VM: ~/.../Labsetup
GNU nano 4.8 sample_spoof2.py
#!/usr/bin/python3
from scapy.all import *
print ("SENDING SPOOFED ICMP PACKET...")
IPLayer = IP()
IPLayer.src="10.9.0.11"
IPLayer.dst="10.9.0.99"
ICMPpkt = ICMP()
pkt = IPLayer/ICMPpkt
pkt.show()
send(pkt,verbose=0)
```

Above code is used to create the ICMP packet. The source and the Destination IP address is mentioned in the above **code are random.** The ICMP() indicates that this is an ICMP packet. The pkt() function prints some information about the sniffed packet. The spoofed request is formed by creating our own packet with the header specifications.



## Attacker's Terminal:

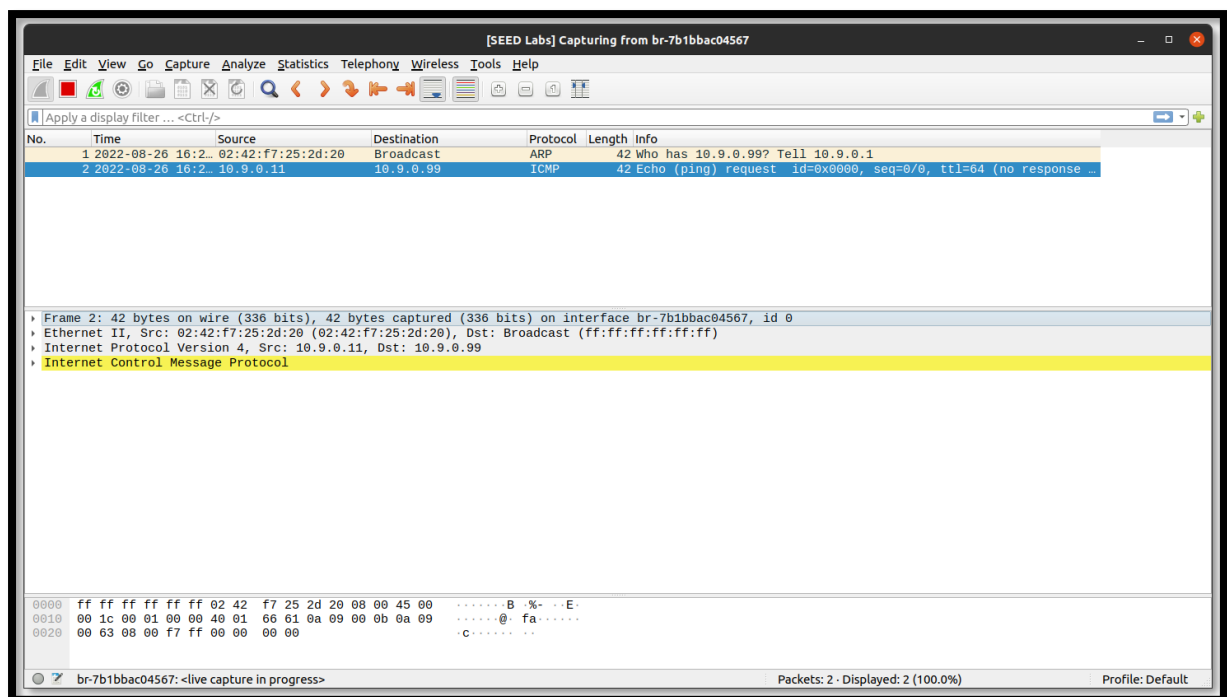
A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows the execution of a Python script 'sample\_spoof2.py' which sends a spoofed ICMP packet. The output displays the IP header and ICMP details. The IP header shows version 4, TTL 64, protocol ICMP, source IP 10.9.0.11, and destination IP 10.9.0.99. The ICMP header shows type 'echo-request', code 0, and sequence number 0x0. The terminal prompt is 'root@VM:/volumes#'.

```
seed@VM: ~/.../Labsetup
root@VM:/volumes# python3 sample_spoof2.py
SENDING SPOOFED ICMP PACKET...
###[ IP ]###
  version  = 4
  ihl      = None
  tos      = 0x0
  len      = None
  id       = 1
  flags    =
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = None
  src      = 10.9.0.11
  dst      = 10.9.0.99
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = None
  id       = 0x0
  seq      = 0x0
root@VM:/volumes#
```

Here we have created an ICMP packet and spoofed it. Then we are sending this spoofed ICMP packet back to the destination IP address. As we can see that this is a request which is being sniffed and spoofed and then it is being sent to the destination address.

## Wireshark Screenshot:

Here we are not getting any reply from the destination IP address.

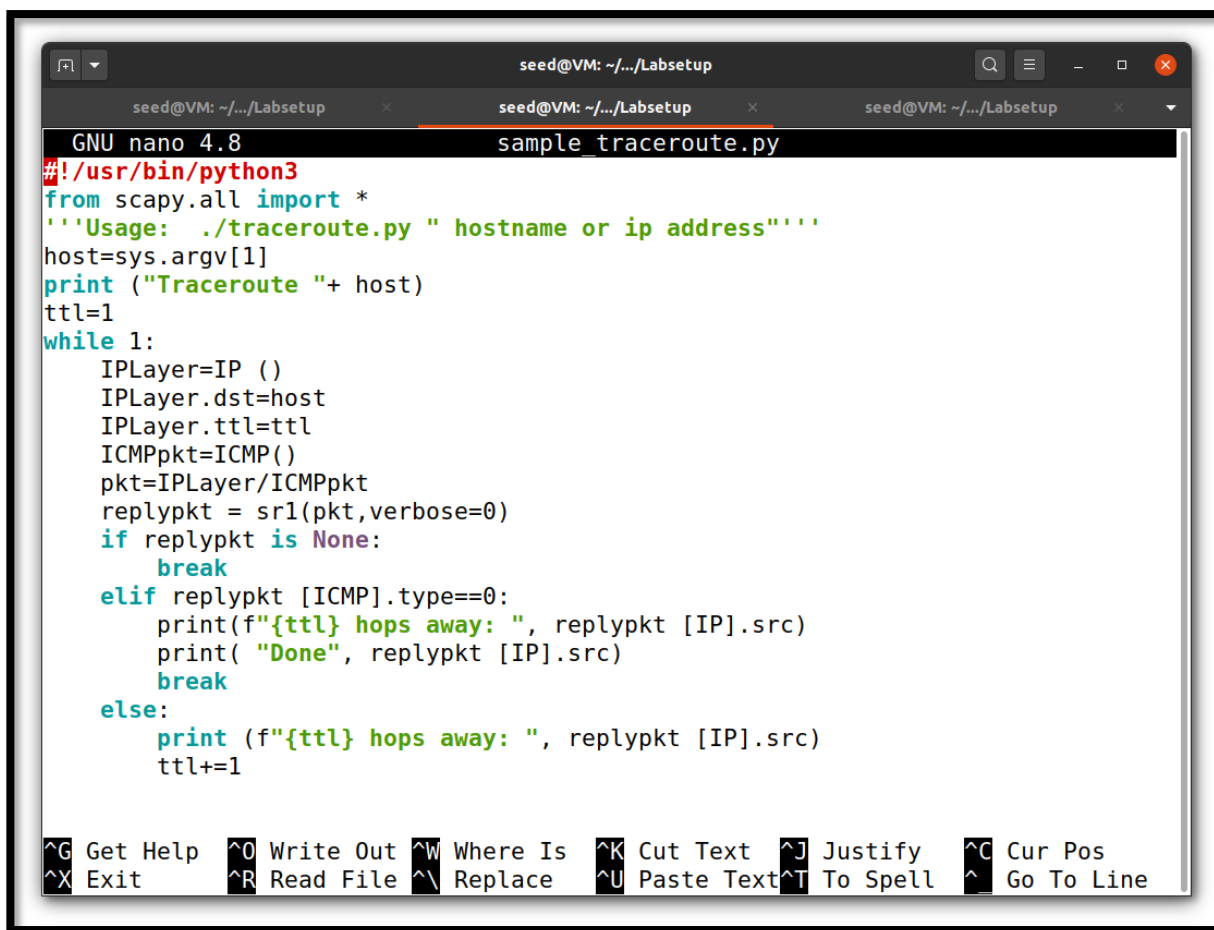


## Task 1.3: TRACEROUTE

The objective of this task is to implement a simple traceroute tool using Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination.

### Python code:

The below code is a simple traceroute implementation using Scapy. It takes hostname or IP address as the input. We create an IP packet with destination address and TTL value and ICMP packet. We send the packet using function sr1(). This function waits for the reply from the destination. If the ICMP reply type is 0, we receive an echo response from the destination, else we increase the TTL value and resend the packet.

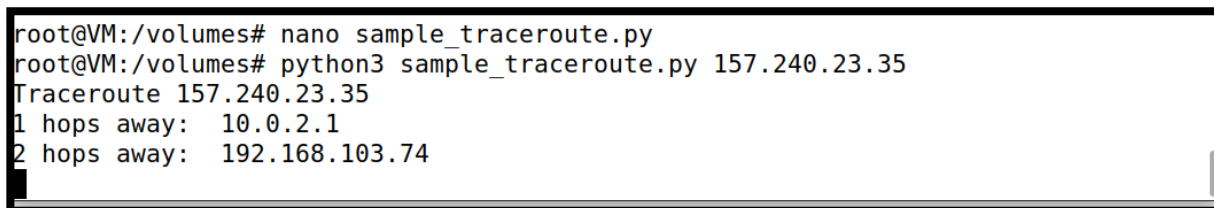


The screenshot shows a terminal window titled 'seed@VM: ~/.../Labsetup'. The user is editing a file named 'sample\_traceroute.py' using 'GNU nano 4.8'. The script is a Python program that uses the 'scapy' library to perform a traceroute. It takes a host as an argument and prints the hops taken to reach it. The script is as follows:

```
#!/usr/bin/python3
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(f"{ttl} hops away: ", replypkt [IP].src)
        print( "Done", replypkt [IP].src)
        break
    else:
        print (f"{ttl} hops away: ", replypkt [IP].src)
        ttl+=1
```

At the bottom of the terminal, there is a status bar with various keyboard shortcuts: ^G Get Help, ^O Write Out, ^W Where Is, ^K Cut Text, ^J Justify, ^C Cur Pos, ^X Exit, ^R Read File, ^\ Replace, ^U Paste Text, ^T To Spell, and ^\_ Go To Line.

### Attacker's Terminal:



The screenshot shows a terminal window titled 'root@VM:/volumes#'. The user has executed the following commands:

```
root@VM:/volumes# nano sample_traceroute.py
root@VM:/volumes# python3 sample_traceroute.py 157.240.23.35
```

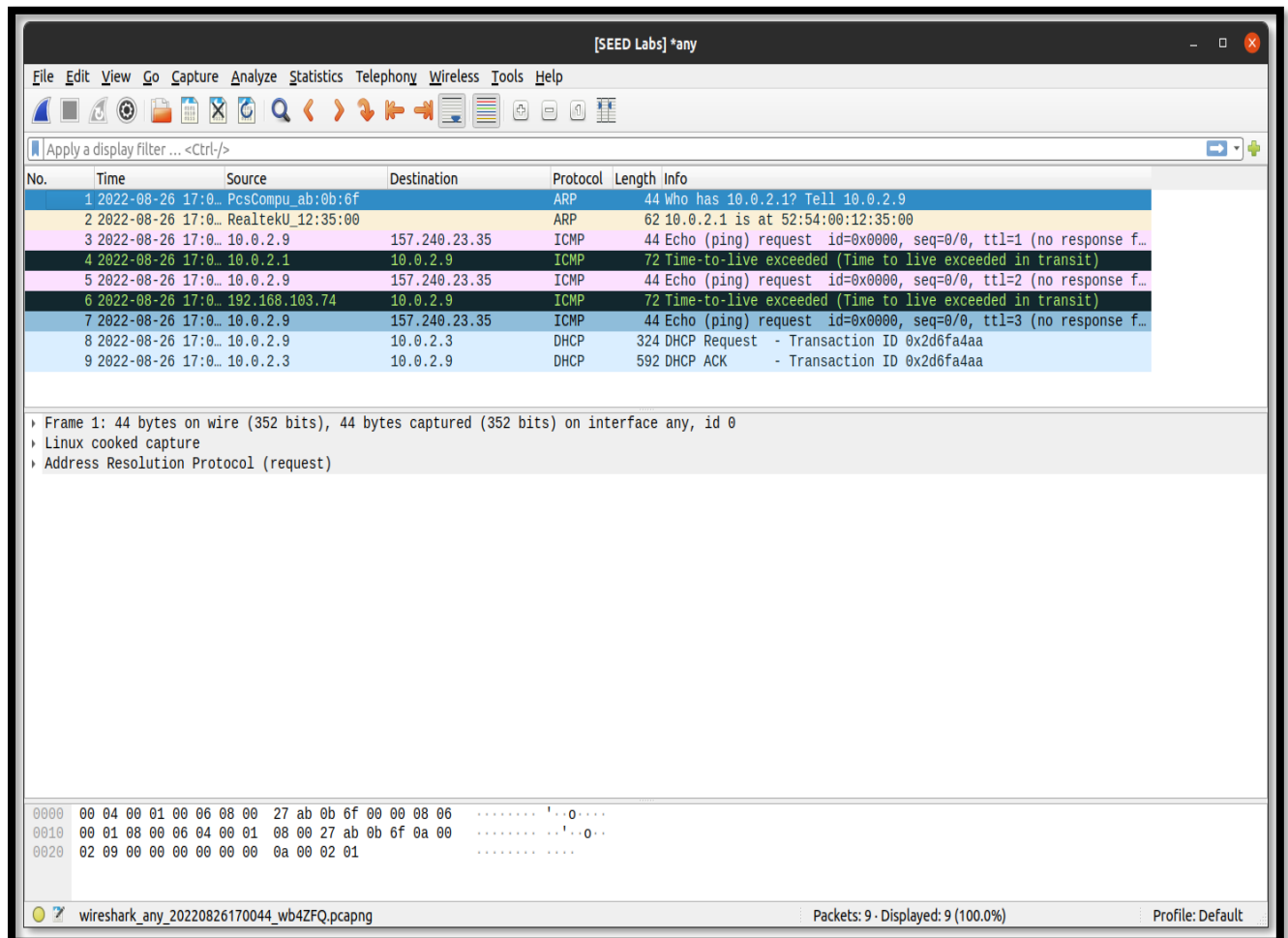
The output of the script is as follows:

```
Traceroute 157.240.23.35
1 hops away: 10.0.2.1
2 hops away: 192.168.103.74
```

Here we are executing the above python code and I have considered IP address of Facebook.com.

We can observe that it takes 2 hops.

## Wireshark Screenshot:

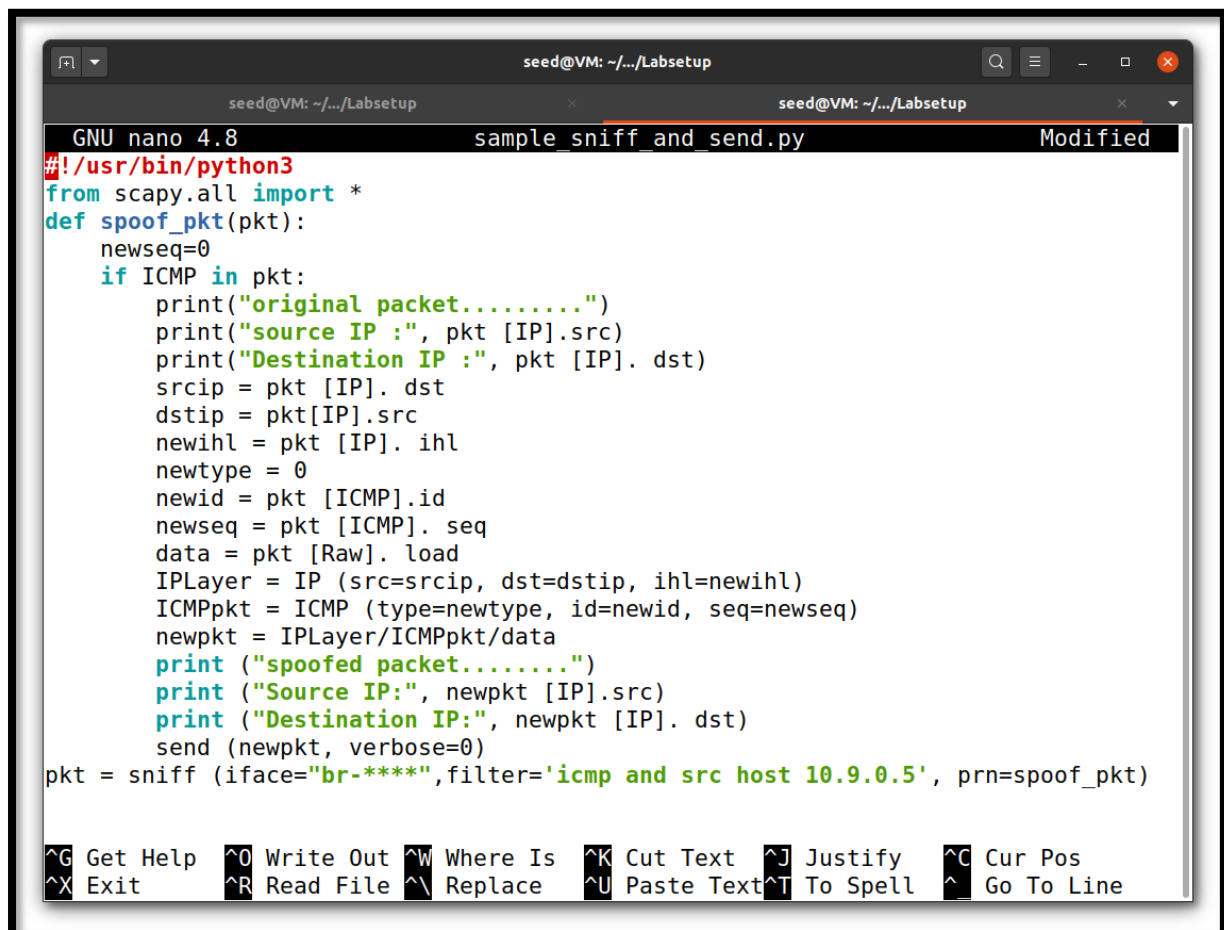


As we can see here there are 2 ICMP packets which gives an error saying that Time-to-live exceeded because there are 2 hops required to reach the IP address of Facebook.com. The TTL value increases and then resends the packet.

## TASK 1.4: SNIFFING AND-THEN SPOOFING

In this task, the victim machine pings a non-existing IP address "1.2.3.4". As the attacker machine is on the same network, it sniffs the request packet, creates a new echo reply packet with IP and ICMP header and sends it to the victim machine. Hence, the user will always receive an echo reply from a non-existing IP address indicating that the machine is alive.

### Python Code:



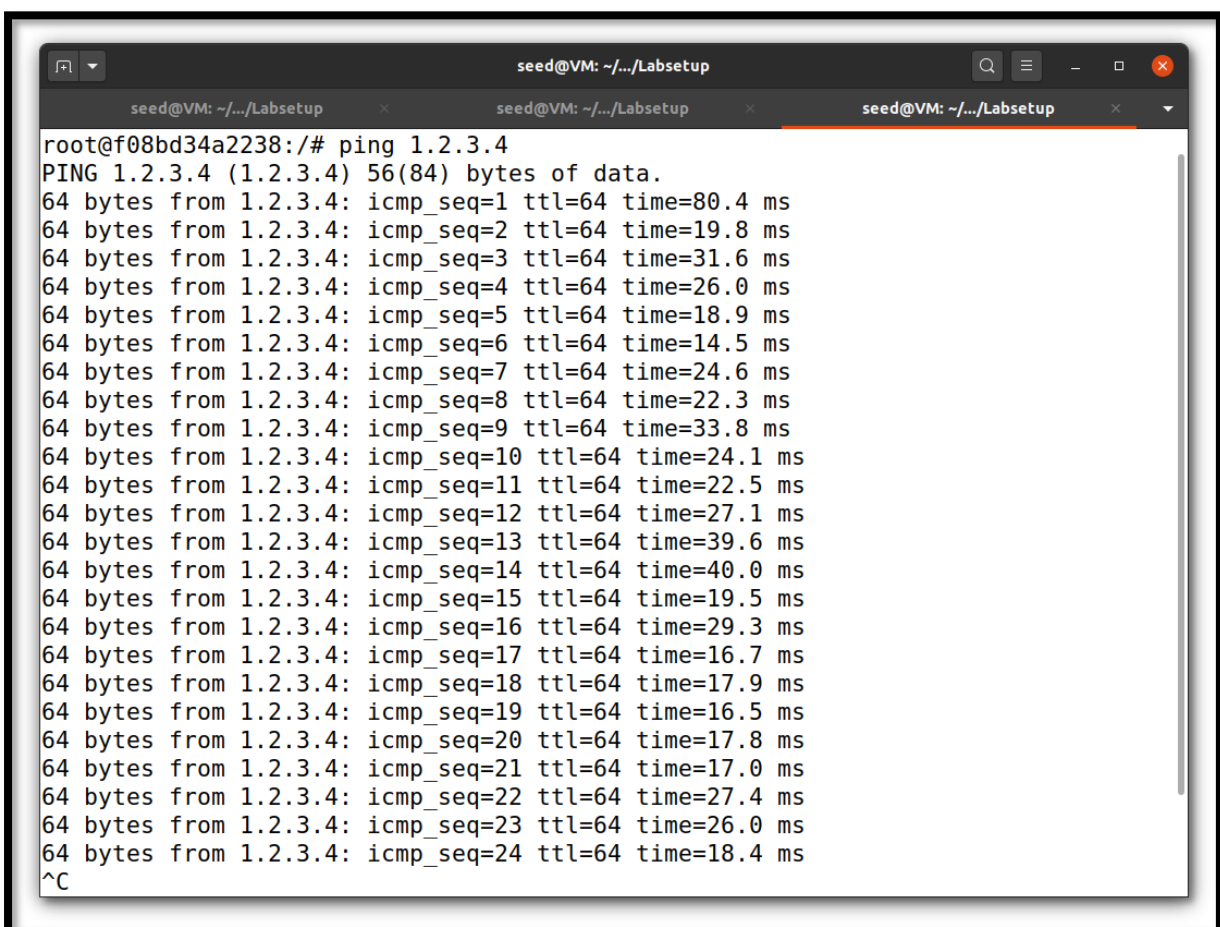
```
seed@VM: ~/.../Labsetup
GNU nano 4.8 sample_sniff_and_send.py Modified
#!/usr/bin/python3
from scapy.all import *
def spoof_pkt(pkt):
    newseq=0
    if ICMP in pkt:
        print("original packet.....")
        print("source IP :", pkt [IP].src)
        print("Destination IP :", pkt [IP]. dst)
        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 (iface="br-****",filter='icmp and src host 10.9.0.5', prn=spoof_pkt)

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell ^_ Go To Line
```

The above code sniffs ICMP packets sent out by the victim machine. Using the callback function, we can use the packets to send the spoofed packets. We retrieve source IP and destination IP from the

sniffed packet and create a new IP packet. The new source IP of the spoofed packet is the sniffed packet's destination IP address and vice versa. We also generate ICMP packets with id and sequence number. In the new packet, ICMP type should be 0 (ICMP reply). To avoid truncated packets, we also add the data to the new packet.

### **Host A's Terminal:**

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows a root user at an IP address of f08bd34a2238 running a 'ping 1.2.3.4' command. The output shows 24 ICMP echo requests, each receiving a reply from 1.2.3.4 with varying TTL and time values. The window has three tabs, all with the same title. The terminal text is as follows:

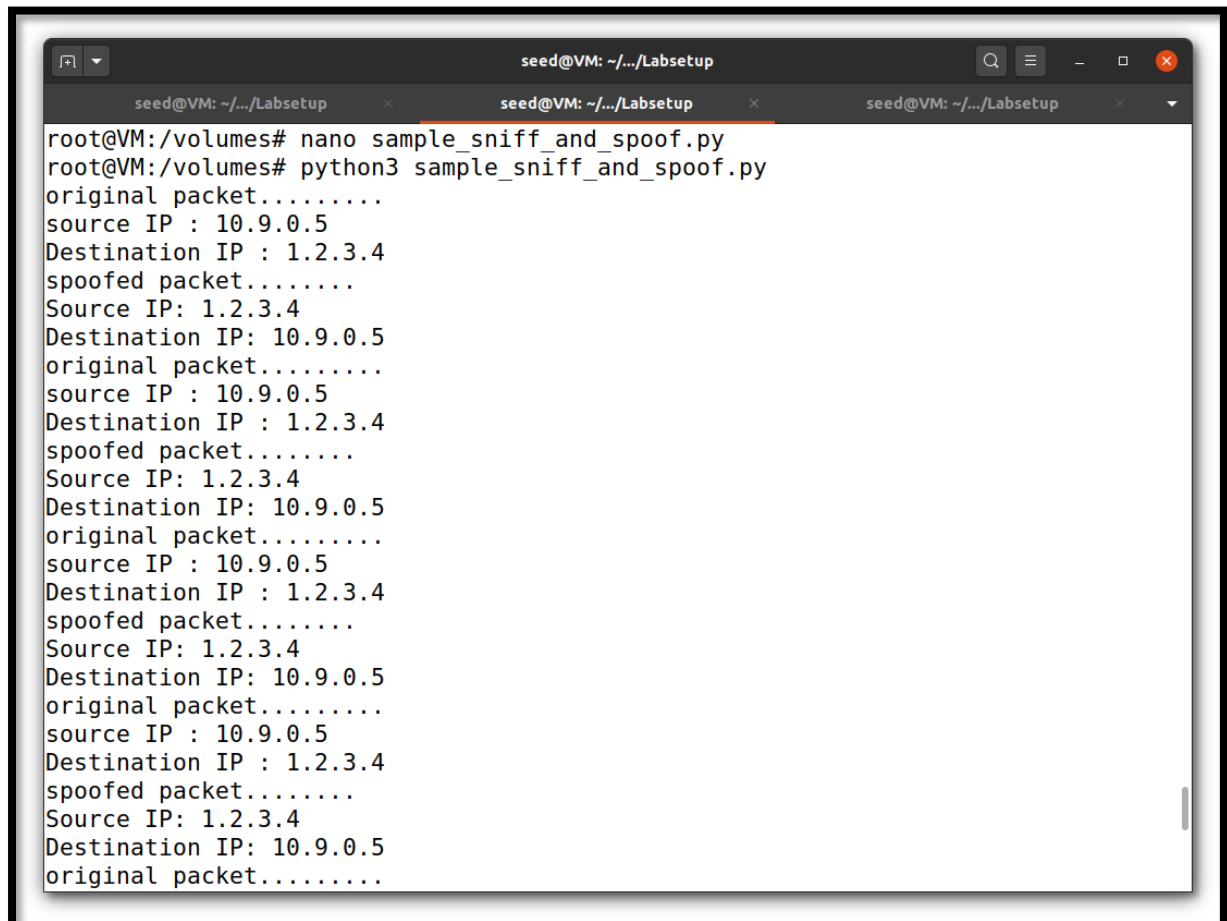
```
root@f08bd34a2238:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=80.4 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=19.8 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=31.6 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=26.0 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=18.9 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=14.5 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=24.6 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=22.3 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=33.8 ms
64 bytes from 1.2.3.4: icmp_seq=10 ttl=64 time=24.1 ms
64 bytes from 1.2.3.4: icmp_seq=11 ttl=64 time=22.5 ms
64 bytes from 1.2.3.4: icmp_seq=12 ttl=64 time=27.1 ms
64 bytes from 1.2.3.4: icmp_seq=13 ttl=64 time=39.6 ms
64 bytes from 1.2.3.4: icmp_seq=14 ttl=64 time=40.0 ms
64 bytes from 1.2.3.4: icmp_seq=15 ttl=64 time=19.5 ms
64 bytes from 1.2.3.4: icmp_seq=16 ttl=64 time=29.3 ms
64 bytes from 1.2.3.4: icmp_seq=17 ttl=64 time=16.7 ms
64 bytes from 1.2.3.4: icmp_seq=18 ttl=64 time=17.9 ms
64 bytes from 1.2.3.4: icmp_seq=19 ttl=64 time=16.5 ms
64 bytes from 1.2.3.4: icmp_seq=20 ttl=64 time=17.8 ms
64 bytes from 1.2.3.4: icmp_seq=21 ttl=64 time=17.0 ms
64 bytes from 1.2.3.4: icmp_seq=22 ttl=64 time=27.4 ms
64 bytes from 1.2.3.4: icmp_seq=23 ttl=64 time=26.0 ms
64 bytes from 1.2.3.4: icmp_seq=24 ttl=64 time=18.4 ms
^C
```

Host pings a non-existing IP address and sends the ICMP packets to that particular IP address.

### **Attacker's Terminal:**

The attacker sniffs the packet which was sent to a non-existing IP address by the host having the IP address 10.9.0.5. After sniffing the

attacker will spoof the packet and then sends that spoofed packet back to the host. Here the attacker will changes his IP address to that unknown IP address. Host will think that the non-existing IP address has replied to the response sent by him. But it's the attacker who has sent the reply by sending spoofed packet.

A terminal window titled 'seed@VM: ~/.../Labsetup' with three tabs. The terminal output shows a script named 'sample\_sniff\_and\_spoof.py' being executed. The script displays a series of 'original packet' and 'spoofed packet' messages. Each 'original packet' message shows a source IP of 10.9.0.5 and a destination IP of 1.2.3.4. Each 'spoofed packet' message shows a source IP of 1.2.3.4 and a destination IP of 10.9.0.5. This sequence is repeated five times.

```
seed@VM: ~/.../Labsetup
root@VM:/volumes# nano sample_sniff_and_spoof.py
root@VM:/volumes# python3 sample_sniff_and_spoof.py
original packet.....
source IP : 10.9.0.5
Destination IP : 1.2.3.4
spoofed packet.....
Source IP: 1.2.3.4
Destination IP: 10.9.0.5
original packet.....
source IP : 10.9.0.5
Destination IP : 1.2.3.4
spoofed packet.....
Source IP: 1.2.3.4
Destination IP: 10.9.0.5
original packet.....
source IP : 10.9.0.5
Destination IP : 1.2.3.4
spoofed packet.....
Source IP: 1.2.3.4
Destination IP: 10.9.0.5
original packet.....
source IP : 10.9.0.5
Destination IP : 1.2.3.4
spoofed packet.....
Source IP: 1.2.3.4
Destination IP: 10.9.0.5
original packet.....
```

### **Wireshark Screenshot:**

We can see series of response and replies between host and non-existing IP address. But the host will not get to know that it was the attacker who was replying by sending spoofed packet in reply to his response. Host will think that it's the non-existing IP address that is sending the reply.

[SEED Labs] Capturing from br-7b1bbac04567

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	2022-08-26 17:1...	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x0021, seq=1/256, ttl=64 (reply in 4)
2	2022-08-26 17:1...	02:42:f7:25:2d:20	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
3	2022-08-26 17:1...	02:42:0a:09:00:05	02:42:f7:25:2d:20	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
4	2022-08-26 17:1...	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=1/256, ttl=64 (request in ...)
5	2022-08-26 17:1...	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x0021, seq=2/512, ttl=64 (reply in 6)
6	2022-08-26 17:1...	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=2/512, ttl=64 (request in ...)
7	2022-08-26 17:1...	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x0021, seq=3/768, ttl=64 (reply in 8)
8	2022-08-26 17:1...	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=3/768, ttl=64 (request in ...)
9	2022-08-26 17:1...	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x0021, seq=4/1024, ttl=64 (reply in ...)
10	2022-08-26 17:1...	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=4/1024, ttl=64 (request in ...)
11	2022-08-26 17:1...	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x0021, seq=5/1280, ttl=64 (reply in ...)
12	2022-08-26 17:1...	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=5/1280, ttl=64 (request in ...)
13	2022-08-26 17:1...	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x0021, seq=6/1536, ttl=64 (reply in ...)
14	2022-08-26 17:1...	02:42:0a:09:00:05	02:42:f7:25:2d:20	ARP	42	Who has 10.9.0.1? Tell 10.9.0.5
15	2022-08-26 17:1...	02:42:f7:25:2d:20	02:42:0a:09:00:05	ARP	42	10.9.0.1 is at 02:42:f7:25:2d:20
16	2022-08-26 17:1...	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=6/1536, ttl=64 (request in ...)
17	2022-08-26 17:1...	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x0021, seq=7/1792, ttl=64 (reply in ...)
18	2022-08-26 17:1...	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=7/1792, ttl=64 (request in ...)
19	2022-08-26 17:1...	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x0021, seq=8/2048, ttl=64 (reply in ...)
20	2022-08-26 17:1...	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=8/2048, ttl=64 (request in ...)
21	2022-08-26 17:1...	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x0021, seq=9/2304, ttl=64 (reply in ...)
22	2022-08-26 17:1...	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=9/2304, ttl=64 (request in ...)
23	2022-08-26 17:1...	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x0021, seq=10/2560, ttl=64 (reply in ...)
24	2022-08-26 17:1...	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=10/2560, ttl=64 (request in ...)
25	2022-08-26 17:1...	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x0021, seq=11/2816, ttl=64 (reply in ...)
26	2022-08-26 17:1...	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=11/2816, ttl=64 (request in ...)
27	2022-08-26 17:1...	10.9.0.5	1.2.3.4	ICMP	98	Echo (ping) request id=0x0021, seq=12/3072, ttl=64 (reply in ...)

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-7b1bbac04567, id 0

Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:f7:25:2d:20 (02:42:f7:25:2d:20)

Internet Protocol Version 4, Src: 10.9.0.5, Dst: 1.2.3.4

0000 02 42 f7 25 2d 20 02 42 0a 09 00 05 08 00 45 00 .B-%.B.....E.

0010 00 54 5c 64 40 00 40 01 d0 31 0a 09 00 05 01 02 .T\de@.1.....

0020 03 04 08 00 33 dd 00 21 00 01 2e b1 08 63 00 00 ....3-!.....c...

0030 00 00 c4 19 0a 00 00 00 00 10 11 12 13 14 15 ..... !\*#\$%

0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25

br-7b1bbac04567: <live capture in progress>

Packets: 52 - Displayed: 52 (100.0%) Profile: Default