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

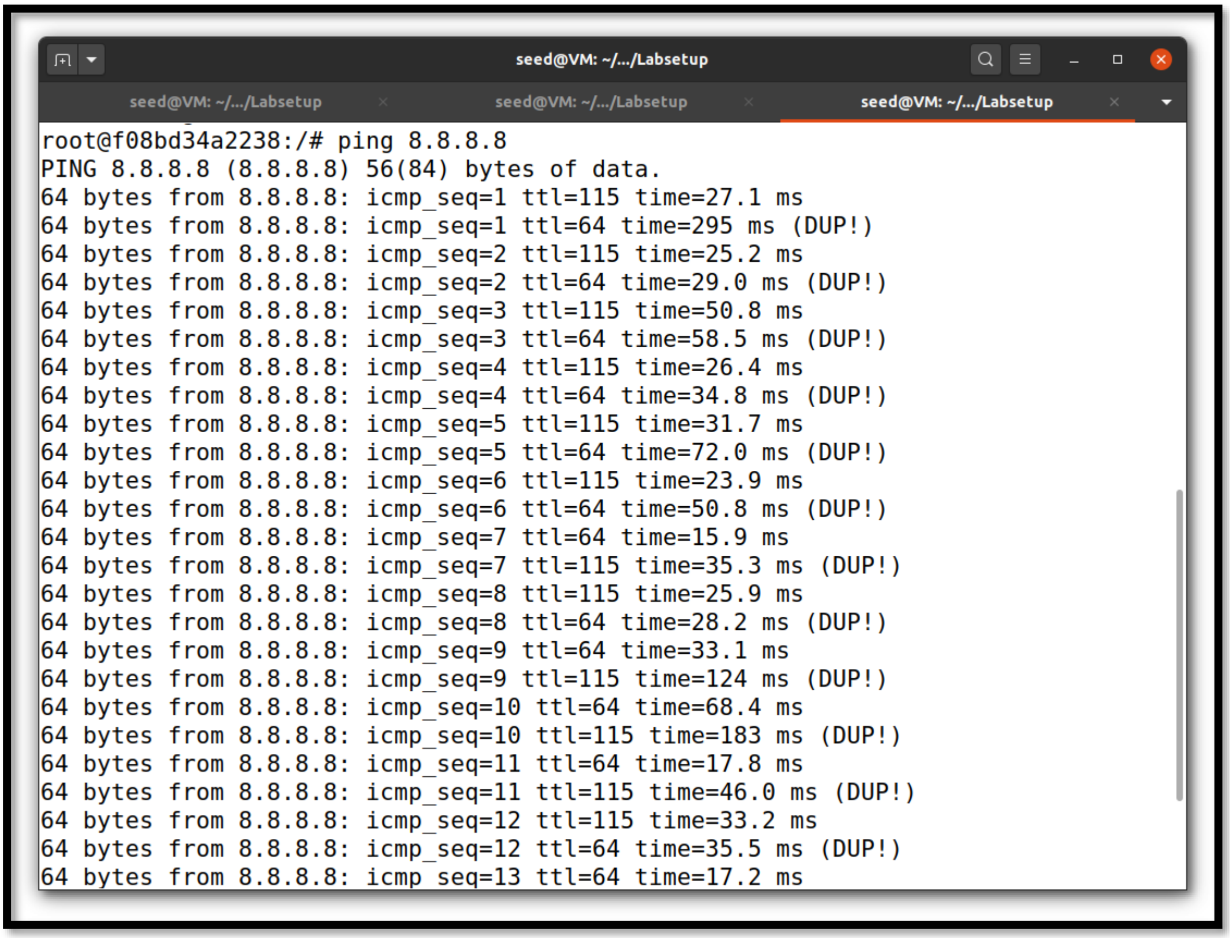
The program ,for each captured packet, the call-back function print pkt() will be invoked and this function will print out some of the information about the packet .

Attacker Terminal:

```
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:

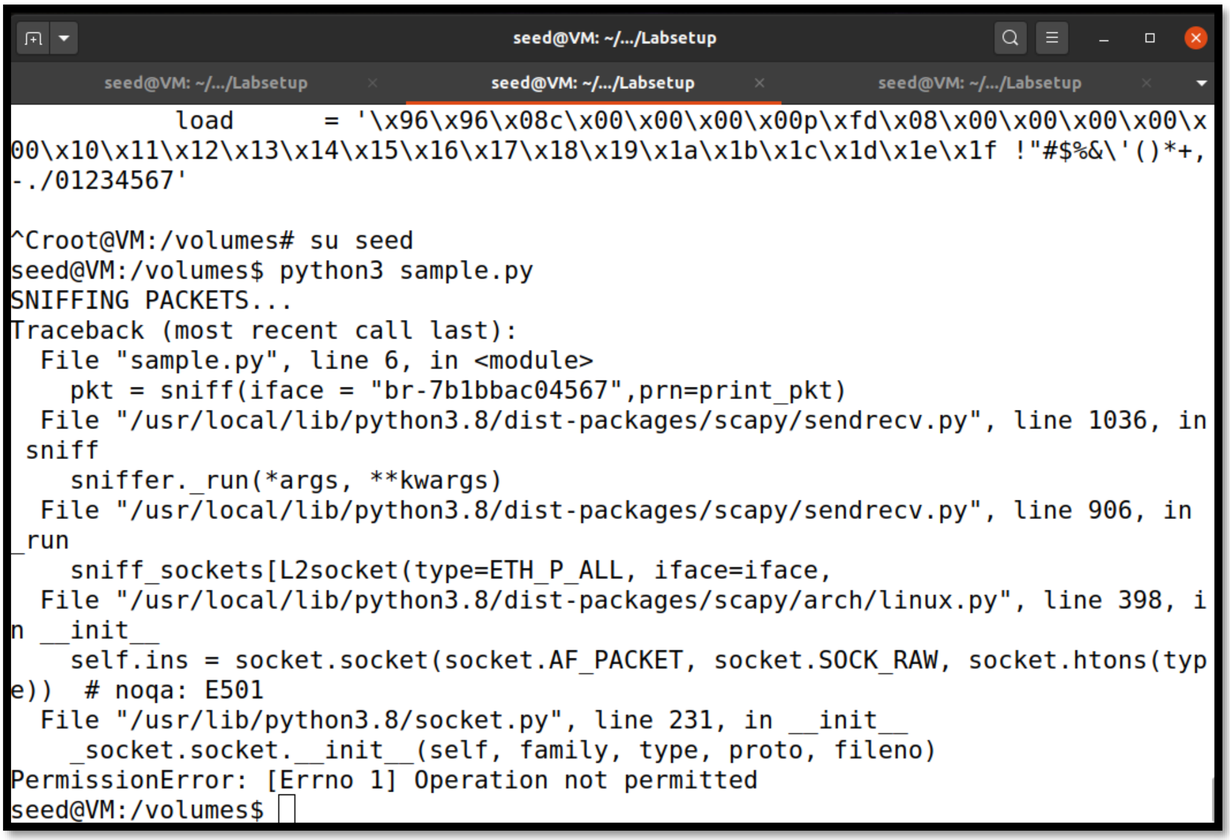



```
root@f08bd34a2238:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=27.1 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=295 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=25.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=29.0 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.



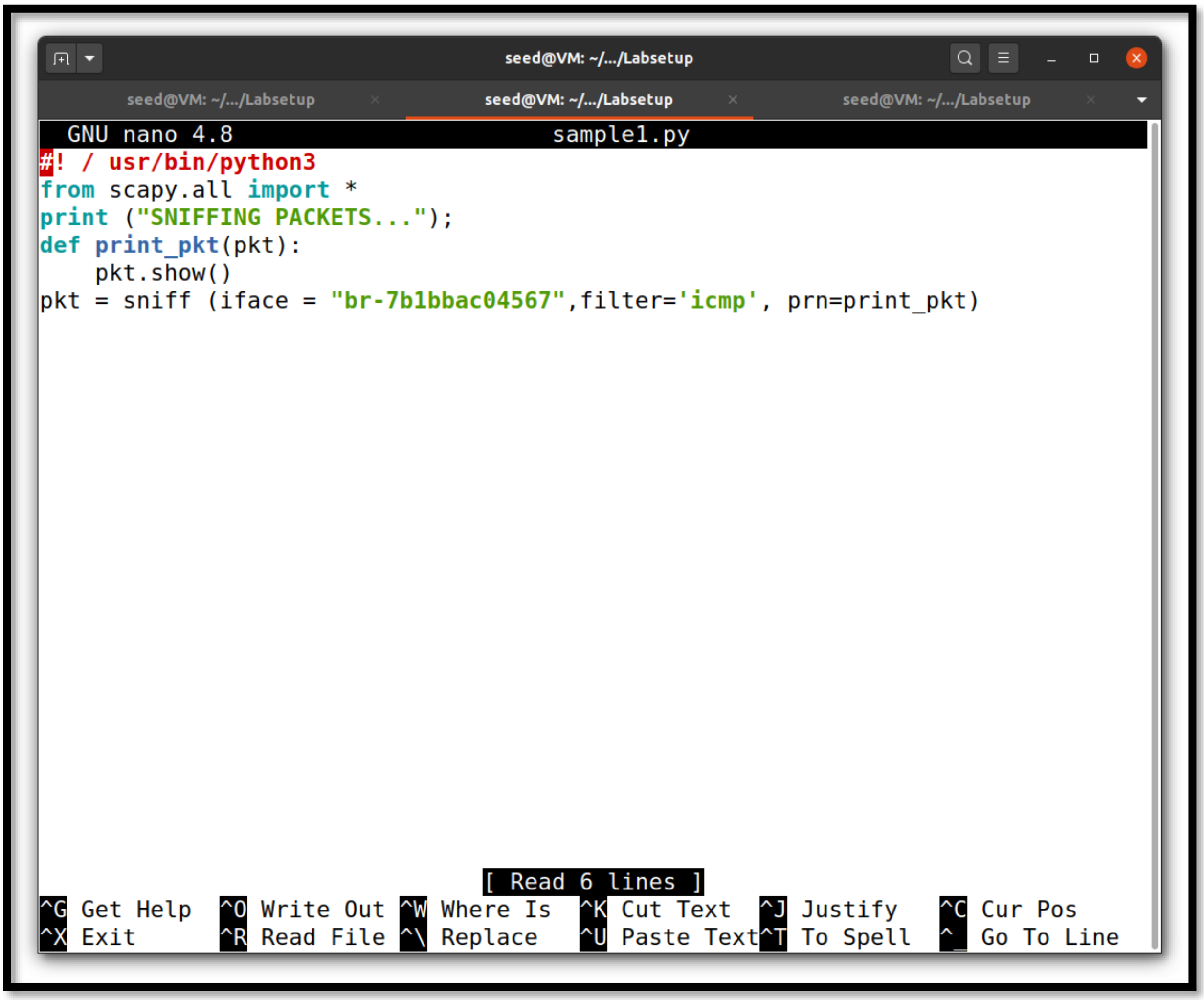
```
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup    seed@VM: ~/.../Labsetup    seed@VM: ~/.../Labsetup
           load      = '\x96\x96\x08c\x00\x00\x00\x00p\xfd\x08\x00\x00\x00\x00\x
00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,
-./01234567'

^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, i
n __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(typ
e))  # 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:



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    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:

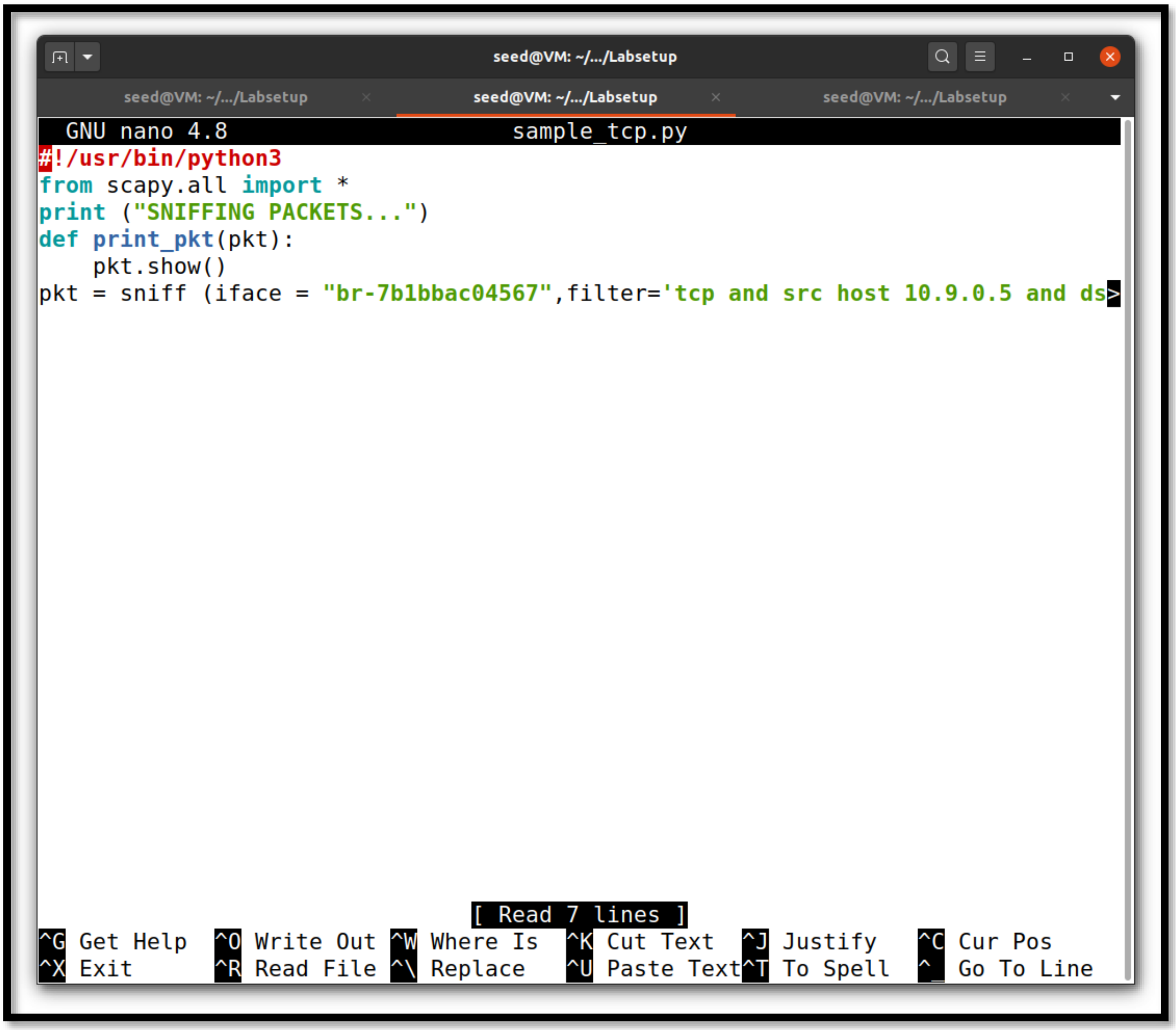
```
root@f08bd34a2238:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=55 time=501 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=55 time=208 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=55 time=140 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=55 time=258 ms
64 bytes 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:

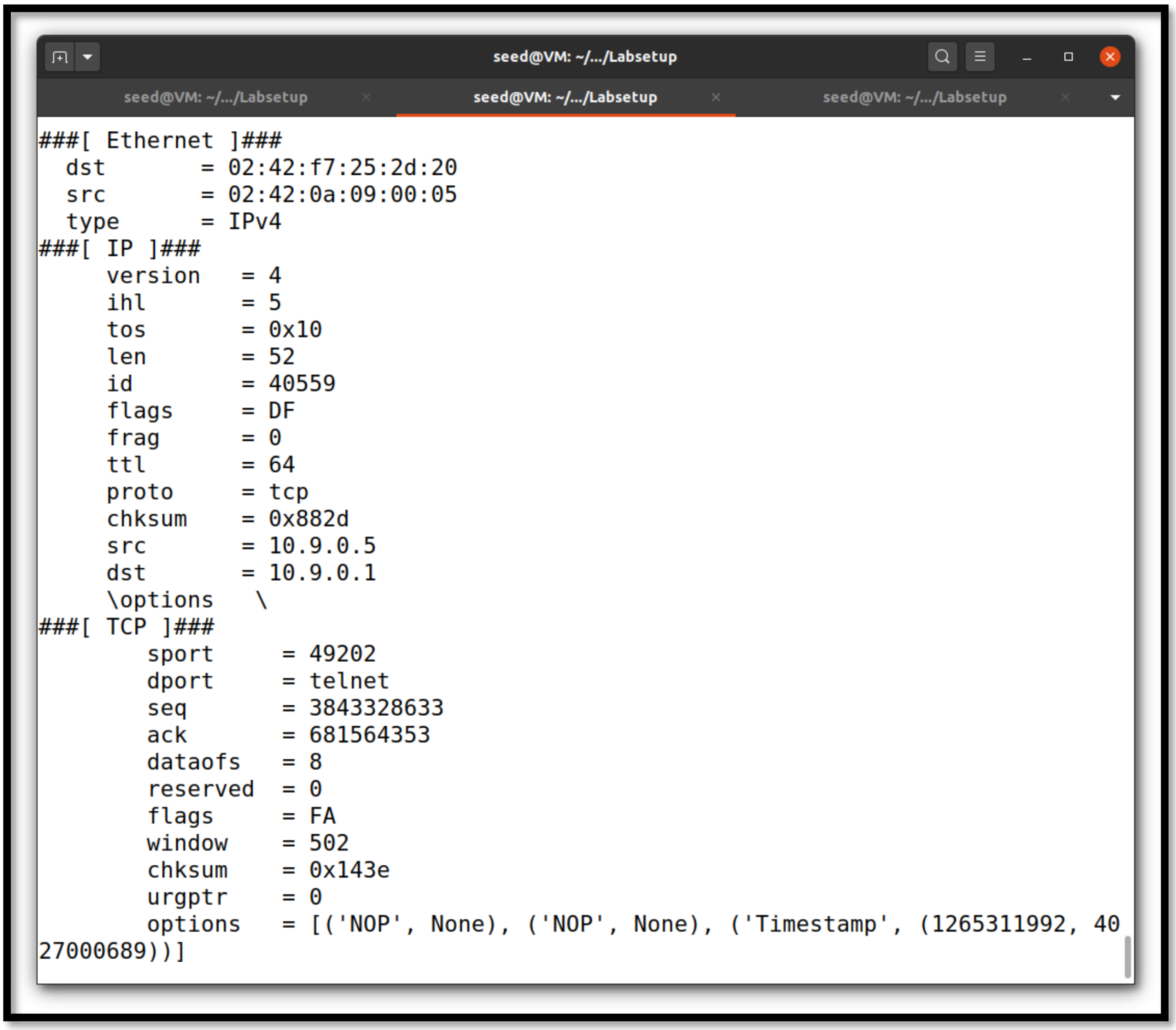
The TCP packets are captured by the Scapy sniffer program. Hence, when some machine on the same network sends ping requests, the packets get captured by the sniffer.

<u>Python code:</u>



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:

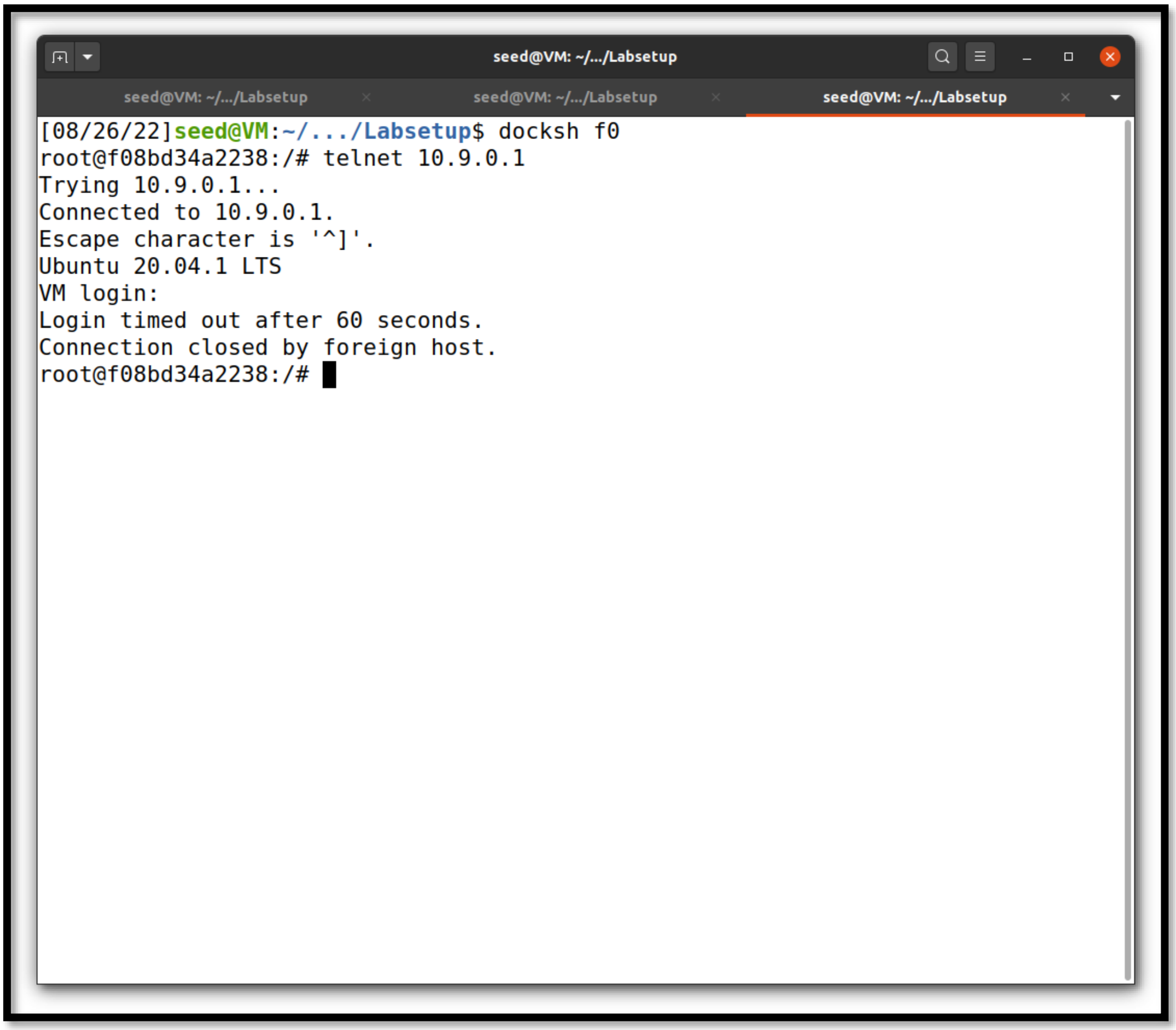


```
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup    seed@VM: ~/.../Labsetup    seed@VM: ~/.../Labsetup
###[ Ethernet ]###
  dst       = 02:42:f7:25:2d:20
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 52
     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, 40
27000689))]
```

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.



```
[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:/#
```

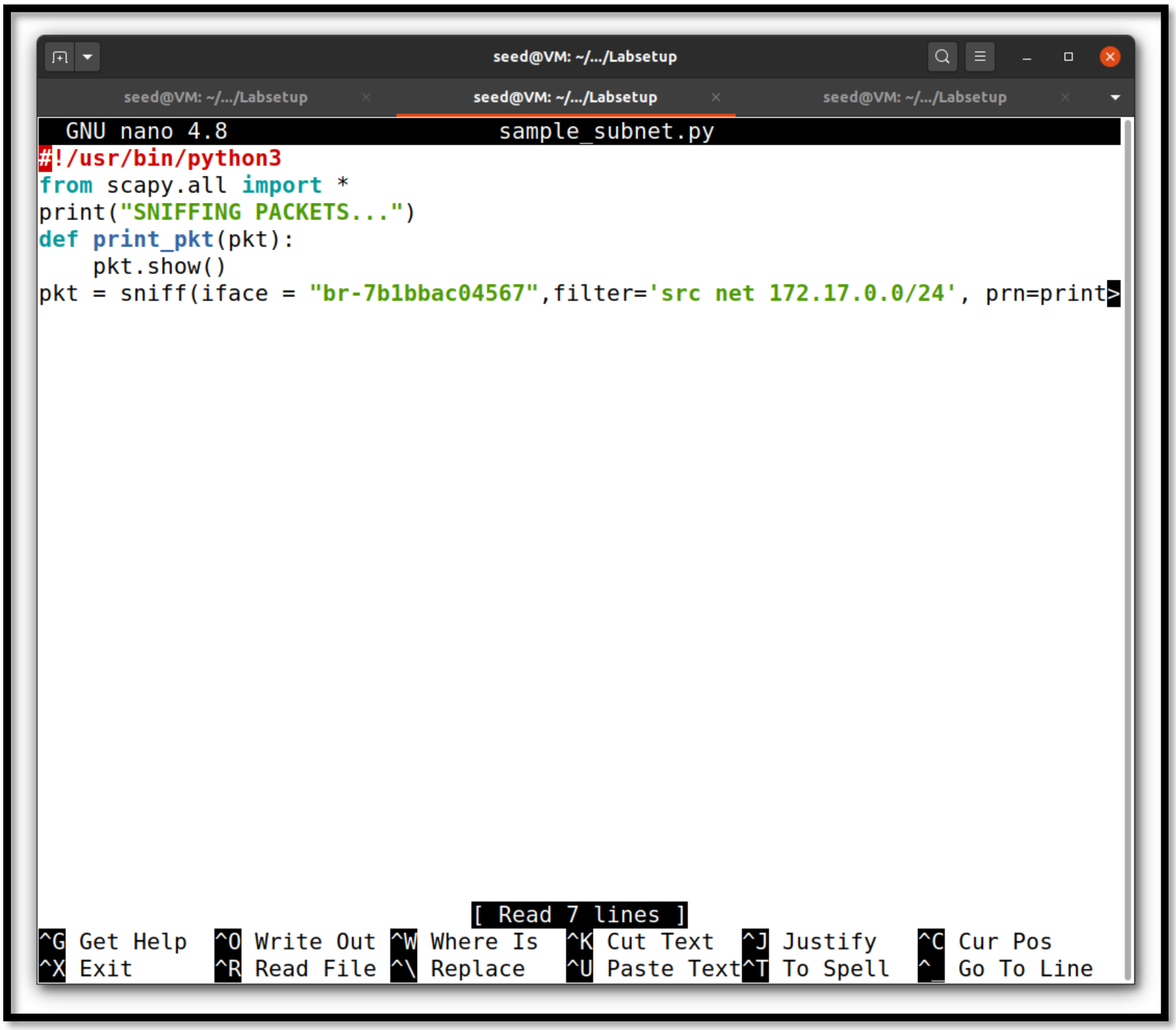
Subnet:

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:

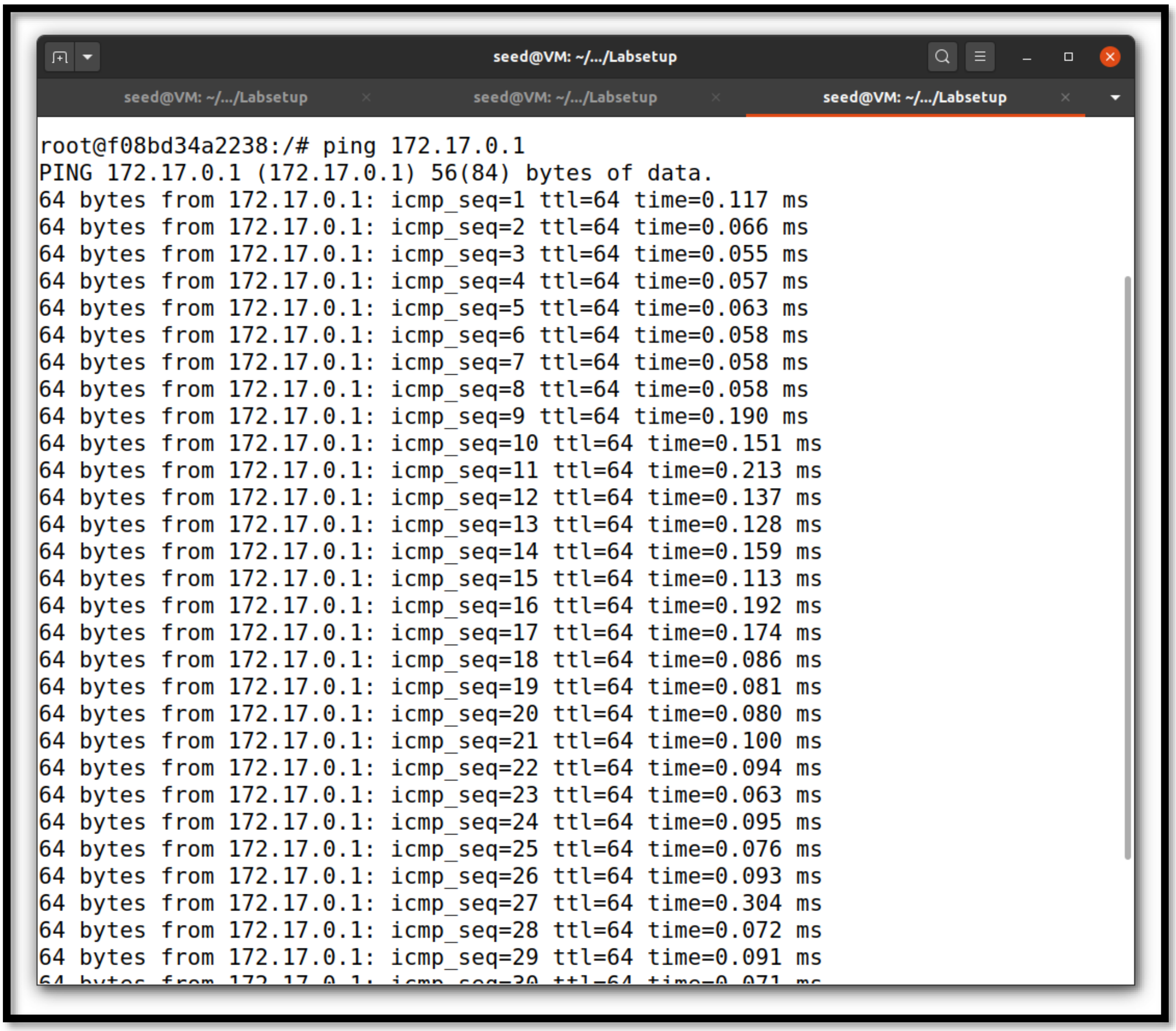
In this example we have chosen the subnet IP address as

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.



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.



Attacker's Terminal:

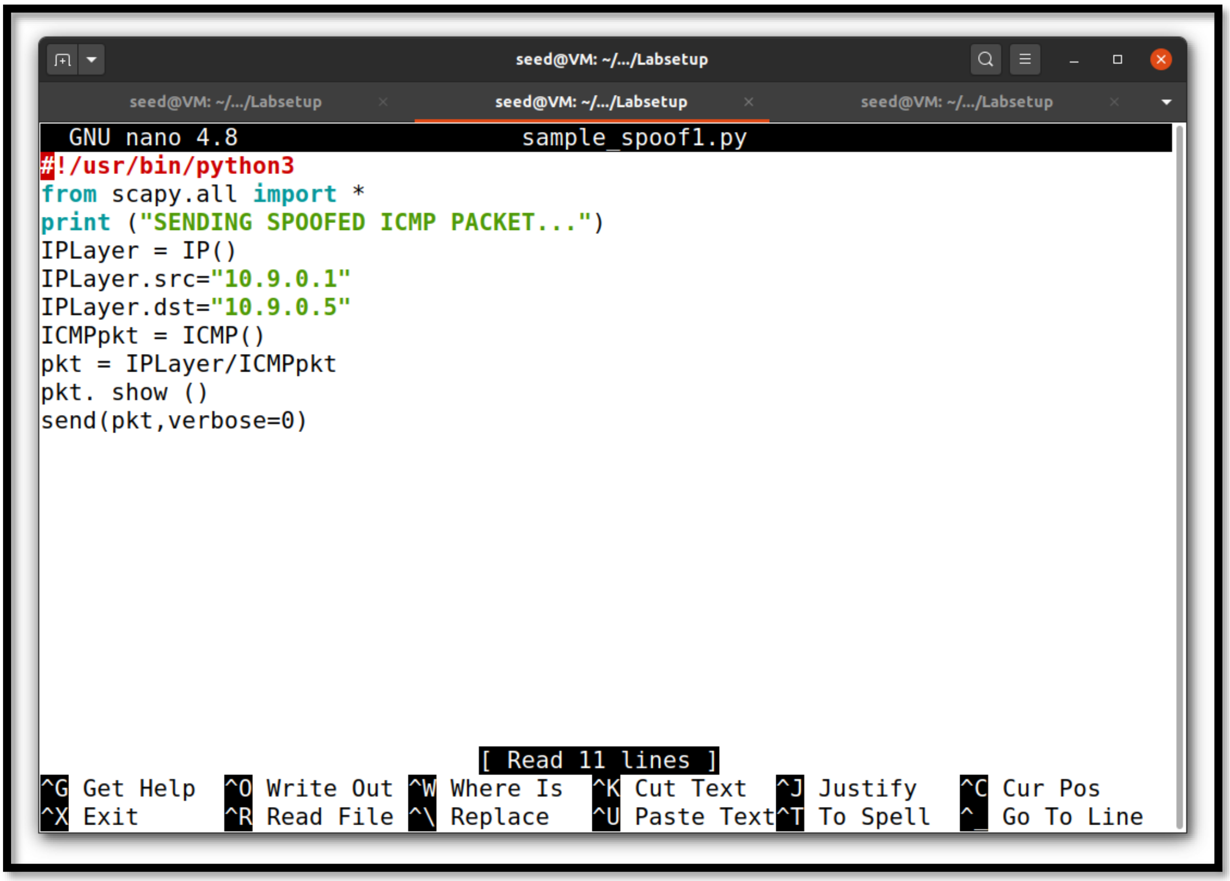
Here we can see that information of the ICMP packet has been displayed in the attacker's terminal. We can observe that all the ICMP packets are sniffed which was sent from the IP address which comes under the subnet address mentioned in the python code mentioned above.

```
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:

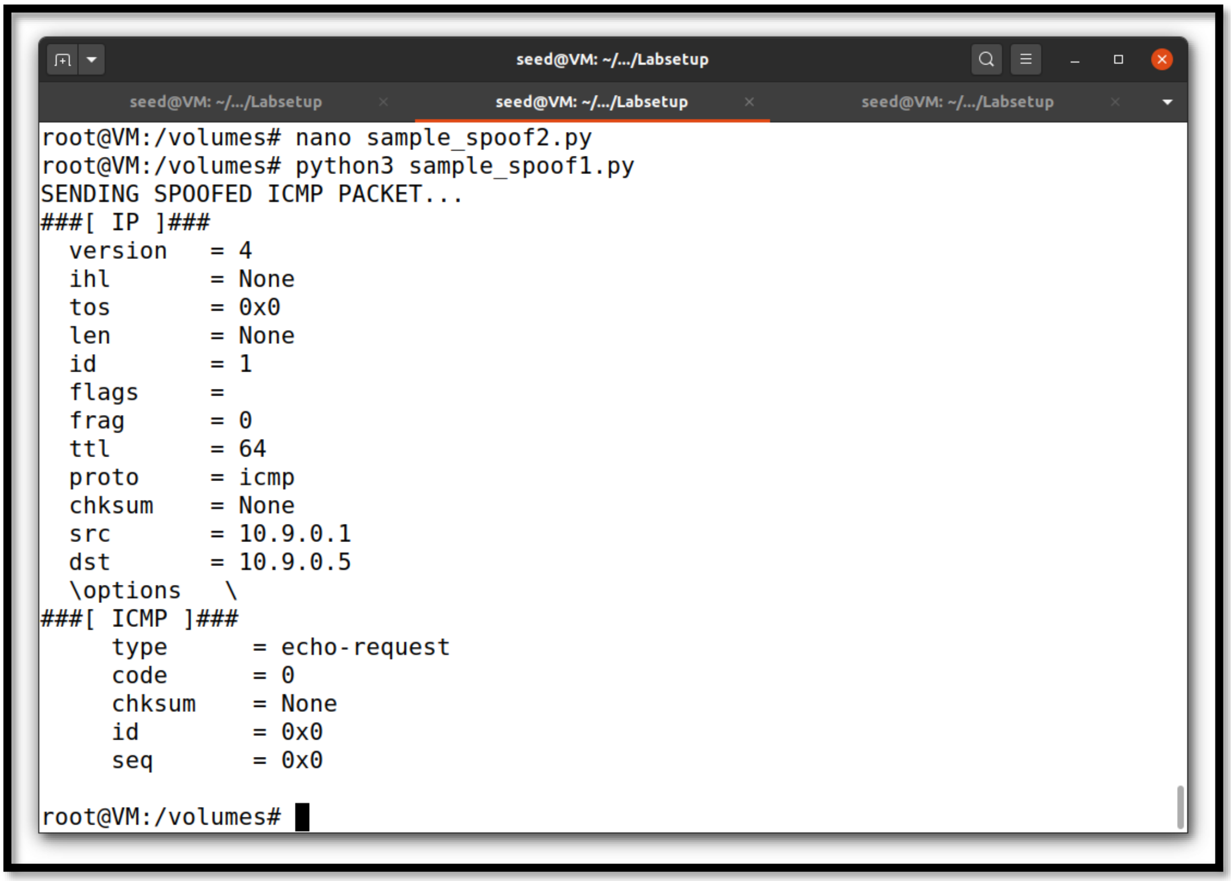


```
  GNU nano 4.8                    sample_spoof1.py
#!/usr/bin/python3
from scapy.all import *
print ("SENDING SPOOFED ICMP PACKET...")
IPLayer = IP()
IPLayer.src="10.9.0.1"
IPLayer.dst="10.9.0.5"
ICMPpkt = ICMP()
pkt = IPLayer/ICMPpkt
pkt. show ()
send(pkt,verbose=0)

                              [ Read 11 lines ]
^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
```

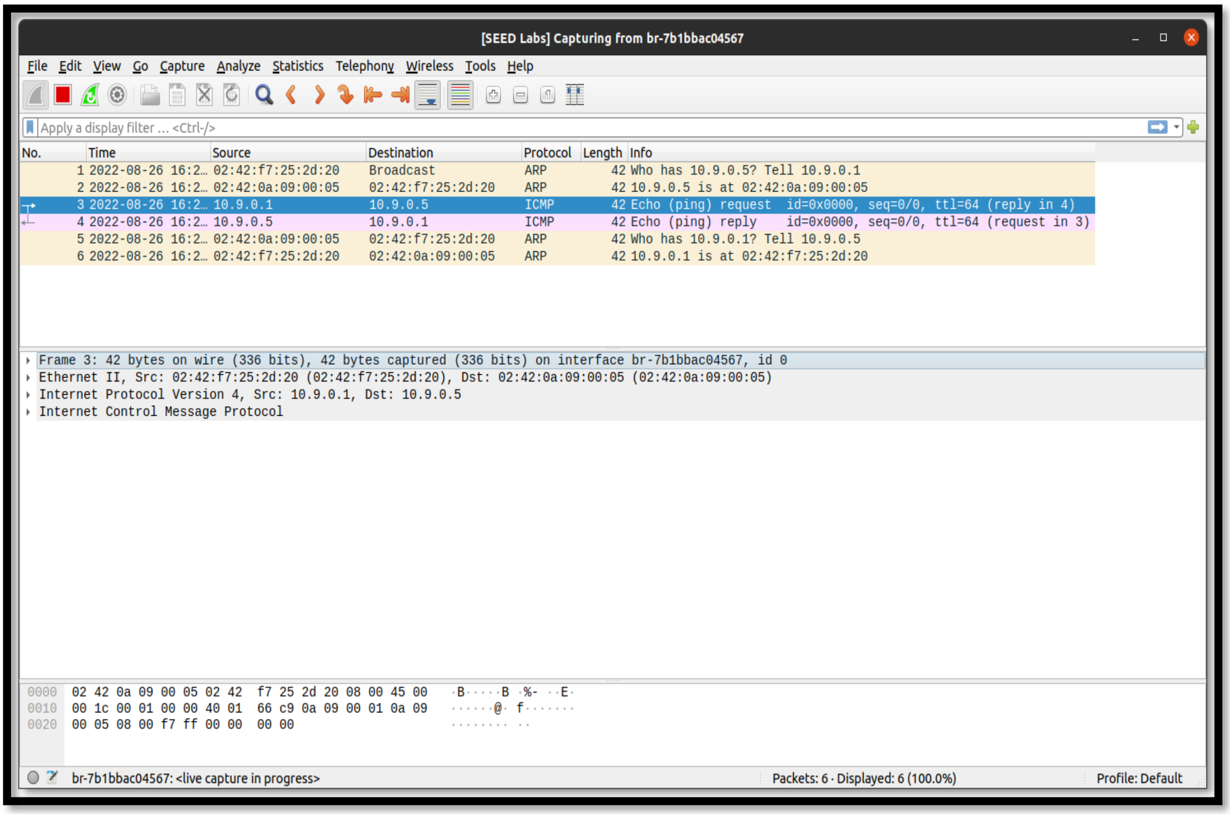
Above code is used to create the ICMP packet. The source and the Destination IP address is mentioned in the above code. The ICMP() indicates that this is an ICMP packet. The pkt() function prints some information about the sniffed packet. The spoofed request is formed by creating our own packet with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



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.

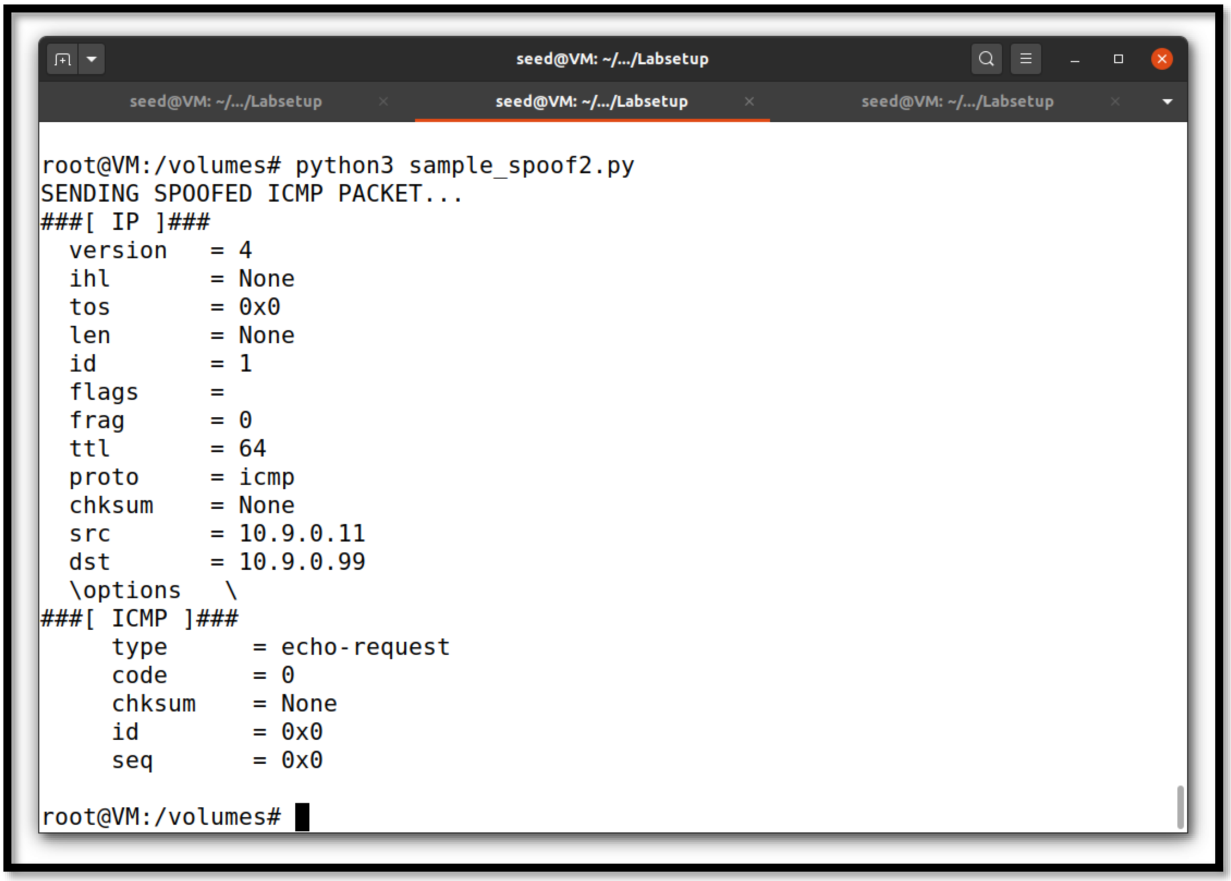
Demonstrate that you can spoof an ICMP echo request packet with an arbitrary source IP address. Open Wireshark and observe the ICMP packets as they are being captured.

Python code:

```
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)

                              [ Read 11 lines ]
^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 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:

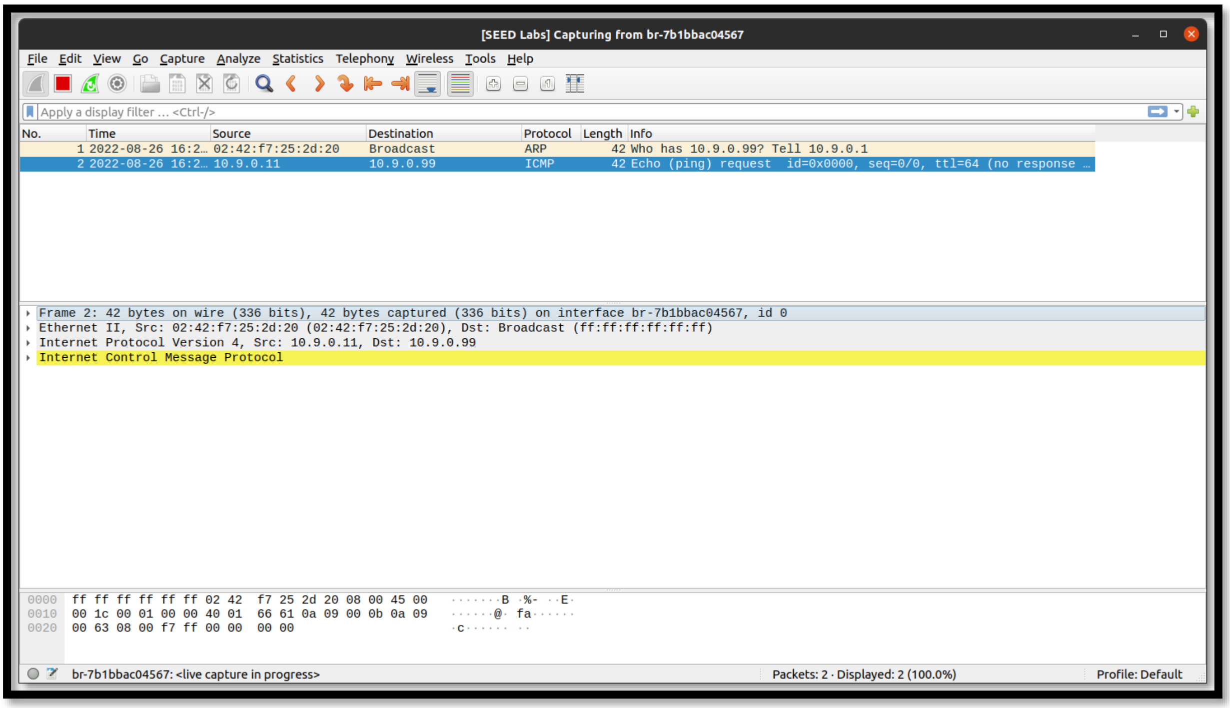



```
root@VM:/volumes# python3 sample_spoof2.py
SENDING SPOOFED ICMP PACKET...
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 10.9.0.11
  dst       = 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.

```
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup    seed@VM: ~/.../Labsetup    seed@VM: ~/.../Labsetup
  GNU nano 4.8                       sample_traceroute.py
#!/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

^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
```

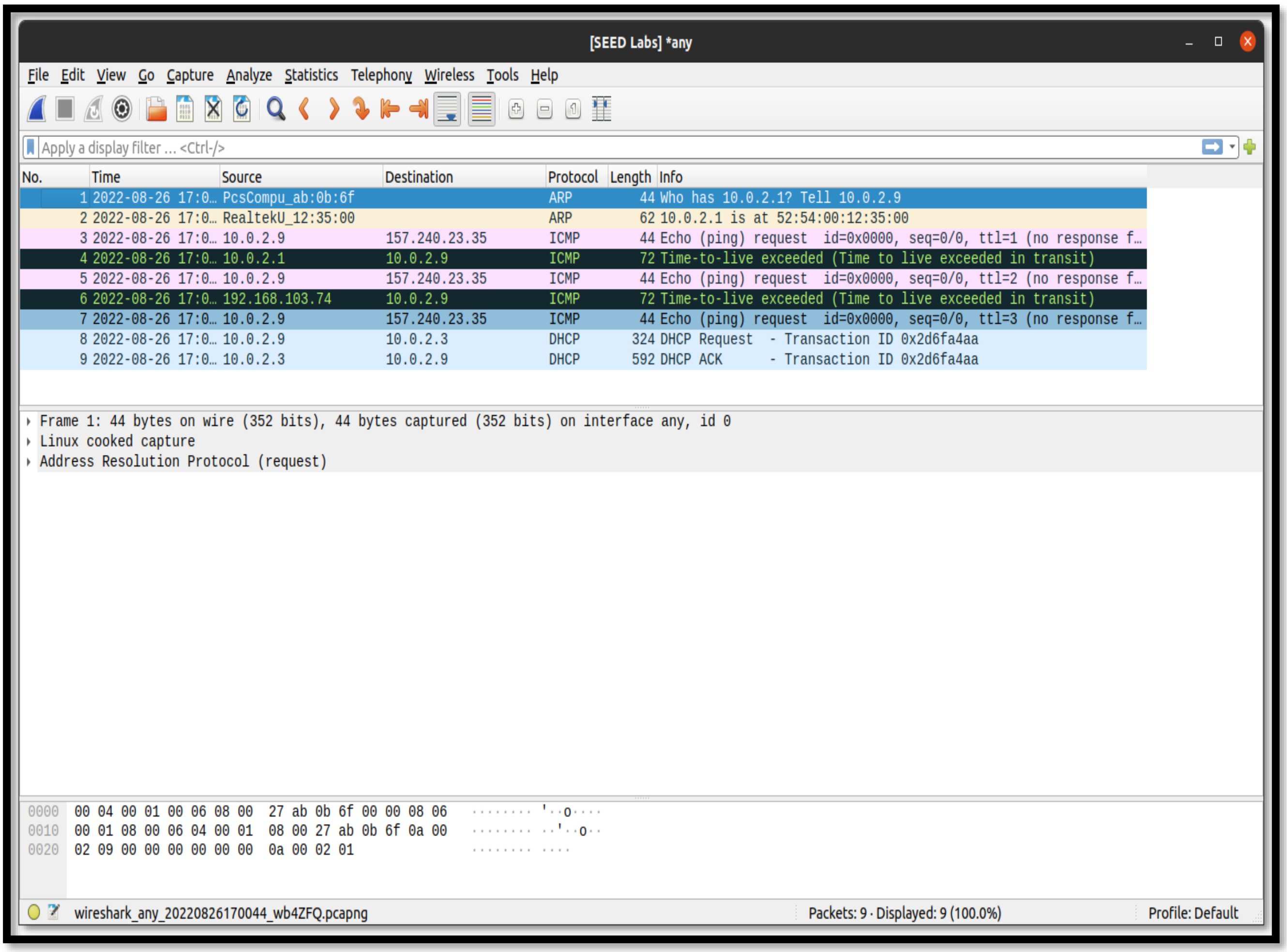
Attacker's Terminal:

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

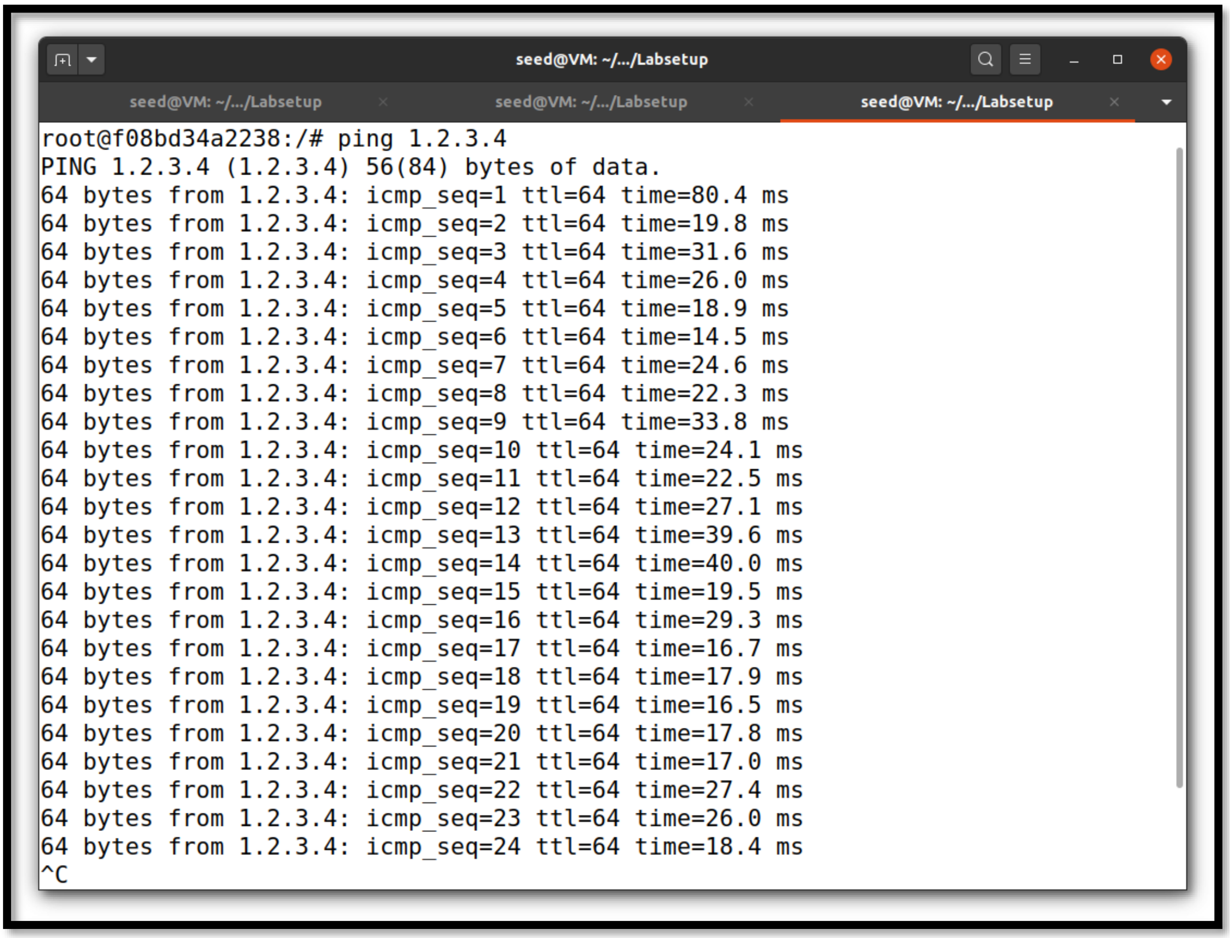
In this task, the victim machine pings a non-existing IP address "1.2.3.4". As the attacker machine is on the same network, it sniffs the request packet, creates a new echo reply packet with IP and ICMP header and sends it to the victim machine. Hence, the user will always receive an echo reply from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

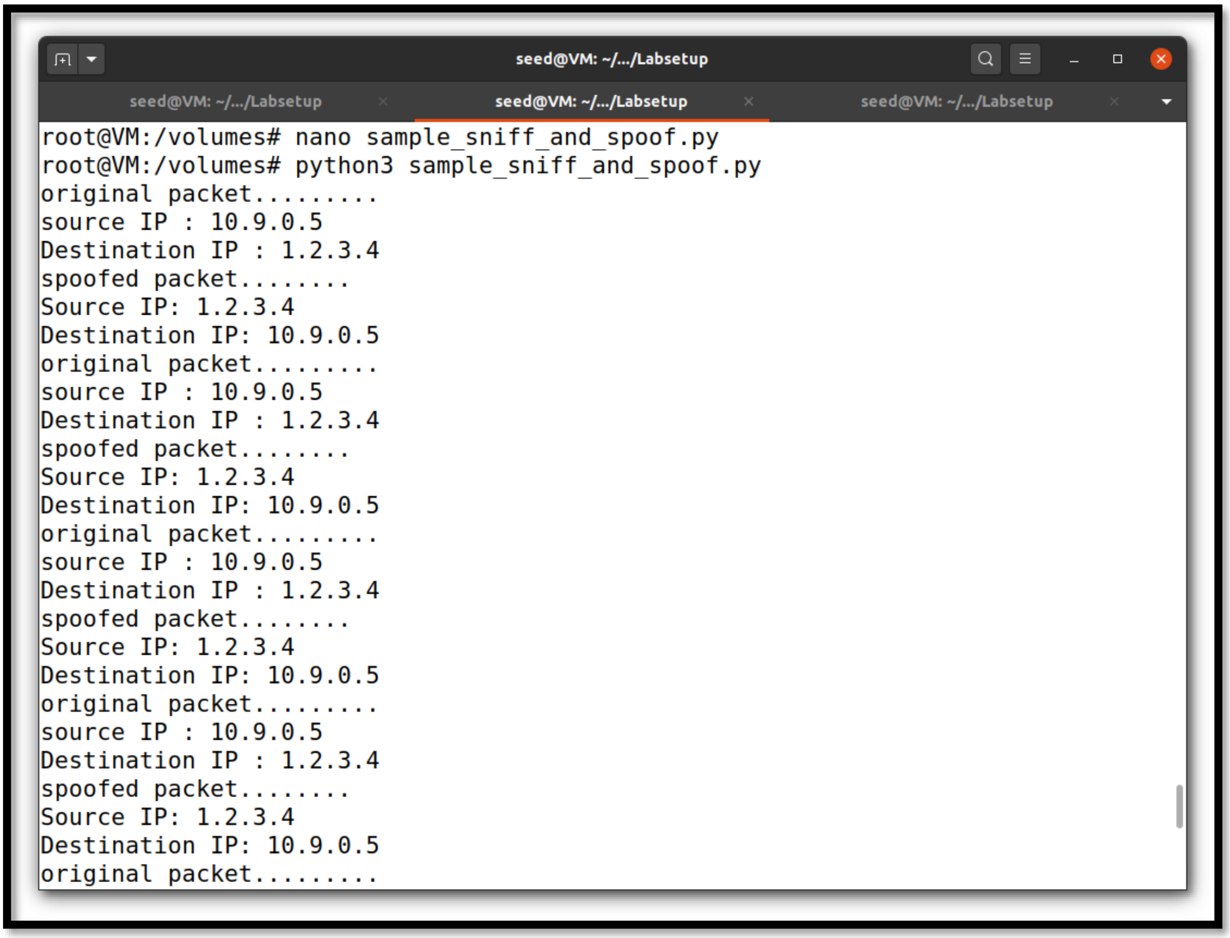
Host A's Terminal:



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

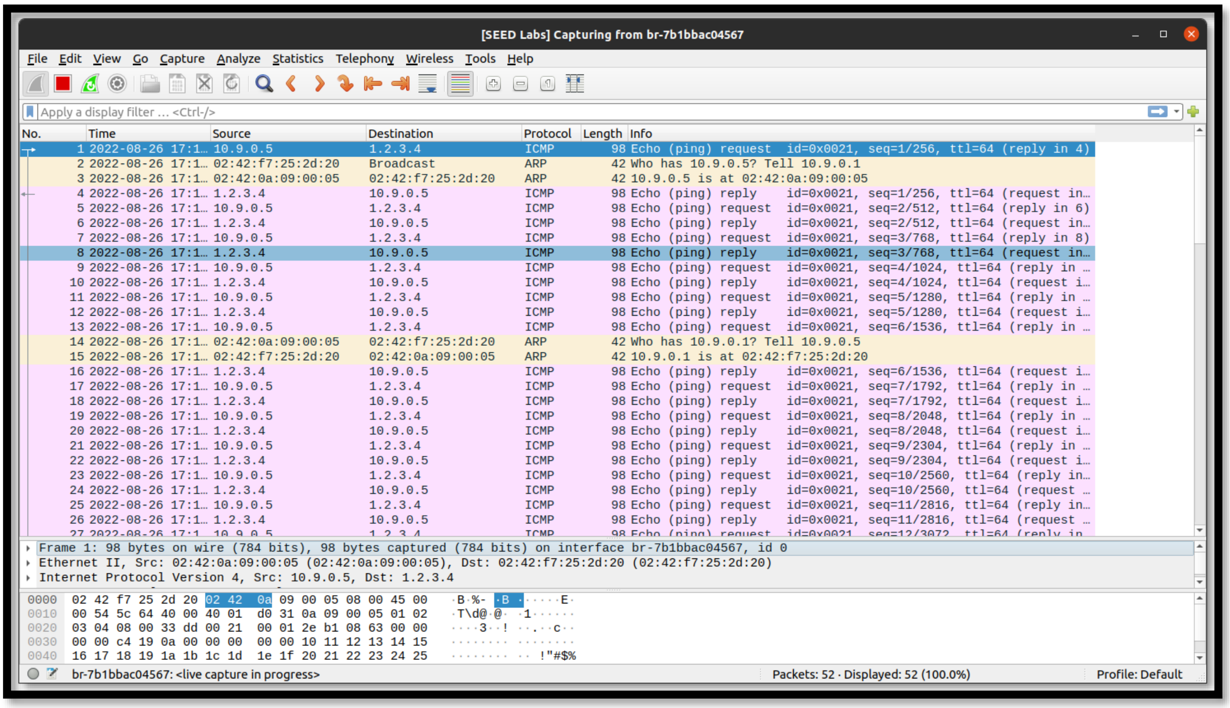
Attacker's Terminal:

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.



<u>Wireshark Screenshot:</u>

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 i…
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 i…
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 i…
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 i…
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 i…
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 i…
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 …
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 …
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
0010 00 54 5c 64 40 00 40 01 d0 31 0a 09 00 05 01 02
0020 03 04 08 00 33 dd 00 21 00 01 2e b1 08 63 00 00
0030 00 00 c4 19 0a 00 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