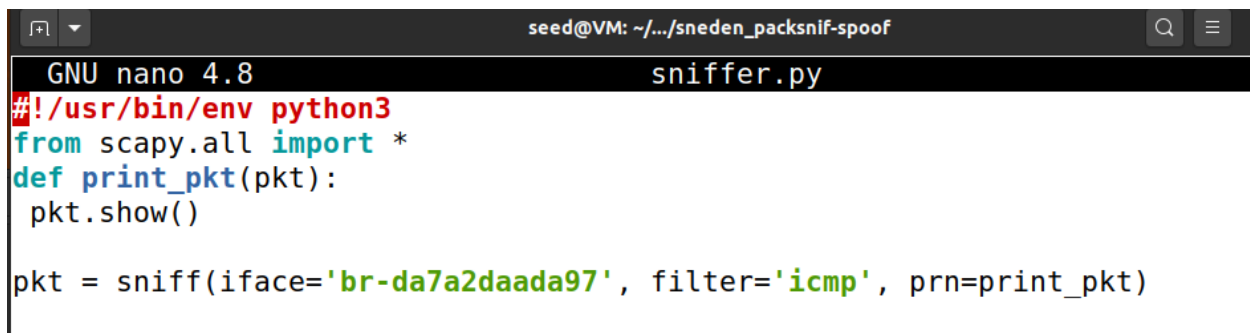


Task 1.1A - Run the Scapy program with the root privilege and demonstrate that you can indeed capture packets. After that, run the program again, but without using the root privilege; describe and explain your observations.

The sniffer.py code is as shown below. I obtained the interface to sniff on by running 'ifconfig' on the attacker container.



```
seed@VM: ~/.../sneden_packsnif-spoof
GNU nano 4.8 sniffer.py
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-da7a2daada97', filter='icmp', prn=print_pkt)
```

Using the docker host A container, login using root, run the scapy code that captures ICMP packets and displays them. Using the ping utility, we generate ICMP packets from another terminal. In this case I did a ping to IP 10.9.0.0 .

```
[02/06/22]seed@VM:~/.../sneden_packsnif-spoof$ ping 10.9.0.0
ping: Do you want to ping broadcast? Then -b. If not, check your local firewall rules
[02/07/22]seed@VM:~/.../sneden_packsnif-spoof$ ping -b 10.9.0.0
WARNING: pinging broadcast address
PING 10.9.0.0 (10.9.0.0) 56(84) bytes of data.
```

The packet's content, including Ethernet headers, IP headers, ICMP headers, and raw payload, is then shown by the running program:

```

seed@VM: ~/.../sneden_packsnif-spoof

root@VM:/# python3 sniffer.py
###[ Ethernet ]###
    dst      = ff:ff:ff:ff:ff:ff
    src      = 02:42:be:03:f6:c5
    type     = IPv4
###[ IP ]###
    version  = 4
    ihl      = 5
    tos      = 0x0
    len      = 84
    id       = 0
    flags    = DF
    frag     = 0
    ttl      = 64
    proto    = icmp
    chksum   = 0x2697
    src      = 10.9.0.1
    dst      = 10.9.0.0
    \options \
###[ ICMP ]###
    type     = echo-request
    code     = 0
    chksum   = 0xbfa6
    id       = 0x1

###[ Raw ]###
    load     = '\xd6\xa7\x00b\x00\x00\x00-\xdd\x07\x00
\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x
1c\x1d\x1e\x1f !"#%&\'()*+,-./01234567'

```

Now in this case, we execute the same code without root access by doing 'su seed' in the root container to change user. We get an error message stating that the operation is not permitted. It happens when the sniff function tries to initialize a raw socket. Promiscuous mode is enabled via raw sockets. However, the software requires root capabilities to enable promiscuous mode. As a result, we'll require root privileges to sniff the raw socket in promiscuous mode.

```

[02/07/22]seed@VM:~/.../sneden_packsnif-spoof$ ping -b 10.9.0.0
WARNING: pinging broadcast address
PING 10.9.0.0 (10.9.0.0) 56(84) bytes of data.
^Z
[1]+  Stopped                  ping -b 10.9.0.0
[02/07/22]seed@VM:~/.../sneden_packsnif-spoof$ ping -b 10.9.0.0
WARNING: pinging broadcast address
PING 10.9.0.0 (10.9.0.0) 56(84) bytes of data.
□

```

After doing a ping to 10.9.0.0, we notice the error.

```

seed@VM:/$ sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 8, in <module>
    pkt = sniff(iface='br-0d69bb96f23d', filter='icmp', 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, so
cket.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:/$ █

```

Task 1.1 b - Please set the following filters and demonstrate your sniffer program again

- Capture only the ICMP packet
- Capture any TCP packet that comes from a particular IP and with a destination port number 23.
- Capture packets comes from or to go to a particular subnet.

1.To capture ICMP packets : The sniffer python code is as below,

Reference source - <https://biot.com/capstats/bpf.html>

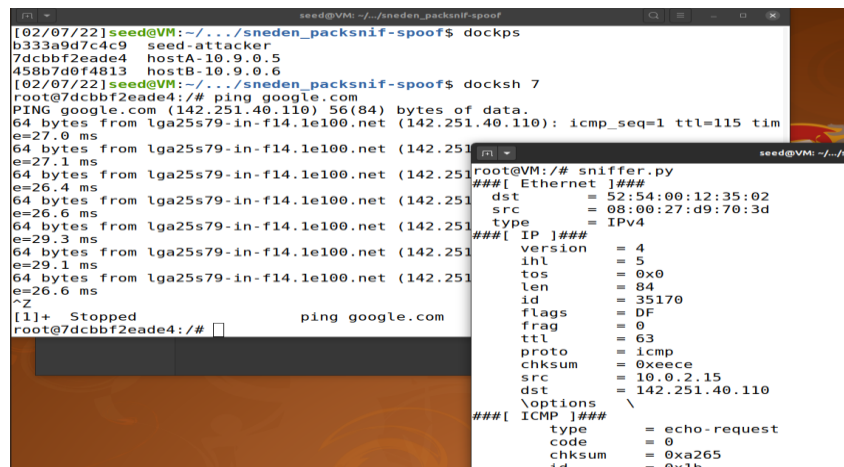
```
seed@VM: ~/.../sneden_packsnif-spoof
GNU nano 4.8 sniffer.py
#!/usr/bin/env python3

from scapy.all import *

def print_pkt(pkt):
    pkt.show()
pkt = sniff(filter='icmp', prn = print_pkt)
```

We run the above code and then ping any address, in this case 'ping google.com' We can see that the code sniffs the ICMP packets on the network and shows the information included in the packet as soon as we start the ping.

The below image shows the captured packets using the sniffing program. Only the ICMP packets are captured.



The screenshot shows a terminal window with two panes. The left pane displays the output of a network capture, showing several ICMP echo request packets from 142.251.40.110 to 142.251.40.110. The right pane shows the output of the 'sniffer.py' program, which has captured an ICMP echo request packet and displayed its details in a structured format.

```
root@7dcbbf2eade4:~# ping google.com
PING google.com (142.251.40.110): 56(84) bytes of data.
64 bytes from lga25s79-in-f14.1e100.net (142.251.40.110): icmp_seq=1 ttl=115 time=27.0 ms
64 bytes from lga25s79-in-f14.1e100.net (142.251.40.110): icmp_seq=2 ttl=115 time=27.1 ms
64 bytes from lga25s79-in-f14.1e100.net (142.251.40.110): icmp_seq=3 ttl=115 time=26.4 ms
64 bytes from lga25s79-in-f14.1e100.net (142.251.40.110): icmp_seq=4 ttl=115 time=26.6 ms
64 bytes from lga25s79-in-f14.1e100.net (142.251.40.110): icmp_seq=5 ttl=115 time=29.3 ms
64 bytes from lga25s79-in-f14.1e100.net (142.251.40.110): icmp_seq=6 ttl=115 time=26.6 ms
^Z
[1]+  Stopped                  ping google.com
root@7dcbbf2eade4:~#
```

```
root@VM:~# sniffer.py
###[ Ethernet ]###
  dst      = 52:54:00:12:35:02
  src      = 08:00:27:d9:70:3d
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 35170
  flags    = DF
  frag     = 0
  ttl      = 63
  proto    = icmp
  chksum   = 0xeece
  src      = 10.0.2.15
  dst      = 142.251.40.110
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0xa265
  id       = 0x1b
```

2. Capture TCP packets on 10.9.0.5 and port 23 :

I will be sniffing TCP packets on 10.9.0.5 (Host A) on port 23.

```
seed@VM: ~/../sneden_packsnif-spoof
GNU nano 4.8 sniffer.py
#!/usr/bin/python
from scapy.all import *
ifac=["br-da7a2daada97", "enp0s3"]
def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface=ifac,filter='tcp and src host 10.9.0.5 and dst port 23',prn=print_pkt)
```

On running the sniffer program and running the telnet service (port 23) to google server 8.8.8.8 from the host A machine. I was able to sniff the TCP packets.

```
seed@VM: ~/../sneden_packsnif-spoof
root@VM:/# nano sniffer.py
root@VM:/# python3 sniffer.py
###[ Ethernet ]###
  dst      = 02:42:42:f9:c1:b4
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x10
  len      = 60
  id       = 33761
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  checksum = 0x9cad
  src      = 10.9.0.5
  dst      = 8.8.8.8
  \options \
###[ TCP ]###
  sport    = 43524
  dport    = telnet
  seq      = 190038512
  ack      = 0
  dataofs  = 10
  reserved = 0
  flags    = S
  window   = 64240
  checksum = 0x1a4c
  urgptr   = 0
  options  = [('MSS', 1460), ('SackOK', b''), ('Timestamp', (3903115610, 0)), ('NOP',
None), ('WScale', 7)]
###[ Ethernet ]###

seed@VM: ~/../sneden_packsnif-spoof
Connection closed by foreign host.
root@7dcbbf2eade4:/# telnet 8.8.8.8
Trying 8.8.8.8...
^X^C
root@7dcbbf2eade4:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.5 netmask 255.255.255.0 broadcast 10.9.0.255
    ether 02:42:0a:09:00:05 txqueuelen 0 (Ethernet)
    RX packets 154 bytes 14778 (14.7 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 145 bytes 10232 (10.2 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 61 bytes 4385 (4.3 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 61 bytes 4385 (4.3 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@7dcbbf2eade4:/# telnet 8.8.8.8
Trying 8.8.8.8...
```

Running the following sniffer code, sniffs packets on the particular subnet included.

At first, I try to sniff on a different subnet to see if the code captures packets. I tried this on 128.130.0.6. The sniffer program could not sniff the packets after the ping. Then I tried pinging the subnet on the filter 128.230.0.0 and now the sniffer program was able to sniff the packets as shown below.

```
seed@VM: ~/.../sneden_packsnif-spoof

root@VM:/# sniffer.py
###[ Ethernet ]###
  dst      = 52:54:00:12:35:02
  src      = 08:00:27:d9:70:3d
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 34685
  flags    = DF
  frag     = 0
  ttl      = 63
  proto    = icmp
  chksum   = 0x2737
  src      = 10.0.2.15
  dst      = 128.230.0.0
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0x57e3
  id       = 0x27
  seq      = 0x1
###[ Raw ]###
  load     = '\x18\x99\x01b\x00\x00\x00\x00\xc1&\x06\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'
```

Task 1.2 - 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.

Below is the code written on python console to spoof an ICMP echo request actually from 10.9.0.1 with any arbitrary source IP address – ie. 8.8.8.8.

```
>>> from scapy.all import *
```

```
>>> a = IP(src='8.8.8.8', dst='10.9.0.5')
```

```
>>> b = ICMP()
```

```
>>> p = a/b
```

```
>>> p.show()
```

```
>>> send(p)
```

Sent 1 packets.

Now start Wireshark, to check if the spoofed packet is received. As seen here, the packet is received as we can see that the receiver accepted the request and sent back a reply.

The screenshot displays the Wireshark network protocol analyzer interface. The main window shows a capture of two ICMP packets. The first packet is an Echo (ping) request from source 8.8.8.8 to destination 10.9.0.5. The second packet is an Echo (ping) reply from source 10.9.0.5 to destination 8.8.8.8. The packet details pane on the left shows the structure of the IP and ICMP layers. The packet bytes pane shows the raw data. A terminal window in the foreground shows the Python code used to create and send the packet.

```
###[ IP ]###
version = 4
ihl = None
tos = 0x0
len = None
id = 1
flags =
frag = 0
ttl = 64
proto = icmp
chksum = None
src = 8.8.8.8
dst = 10.9.0.5
\options \
###[ ICMP ]###
type = echo-request
code = 0
chksum = None
id = 0x0
seq = 0x0

>>> send(p)
Sent 1 packets.
```

Task 1.3: The objective of this task is to use Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination. This is basically what is implemented by the traceroute tool. In this task, we will write our own tool.

The scapy code for implementing the traceroute functionality is as follows. The code will print out the distance to the destination IP, which in this case is google's server 8.8.8.8.

```
seed@VM: ~/.../sneden_packetsnif-spoof
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> from scapy.all import*
>>> i=0

>>> while(True):
...     i += 1
...     a = IP(dst = '8.8.8.8', ttl=i)
...     b = ICMP()
...     p = a/b
...     reply = sr1(p)
...     print ("source IP -", reply[IP].src)
...     if(reply[IP].src == "8.8.8.8"):
...         break
...
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
source IP - 10.9.0.1
Begin emission:
Finished sending 1 packets.
```

On running the code, and printing the distance = 'i', we are able to obtain a status of the route of the packet to reach 8.8.8.8 from the source 10.9.0.1 including all the routers in the route taken that dropped the packet with their IP address included.

Received 1 packets, got 1 answers, remaining 0 packets
source IP - 108.170.233.62

Begin emission:

Finished sending 1 packets.

*

Received 1 packets, got 1 answers, remaining 0 packets
source IP - 142.251.60.225

Begin emission:

Finished sending 1 packets.

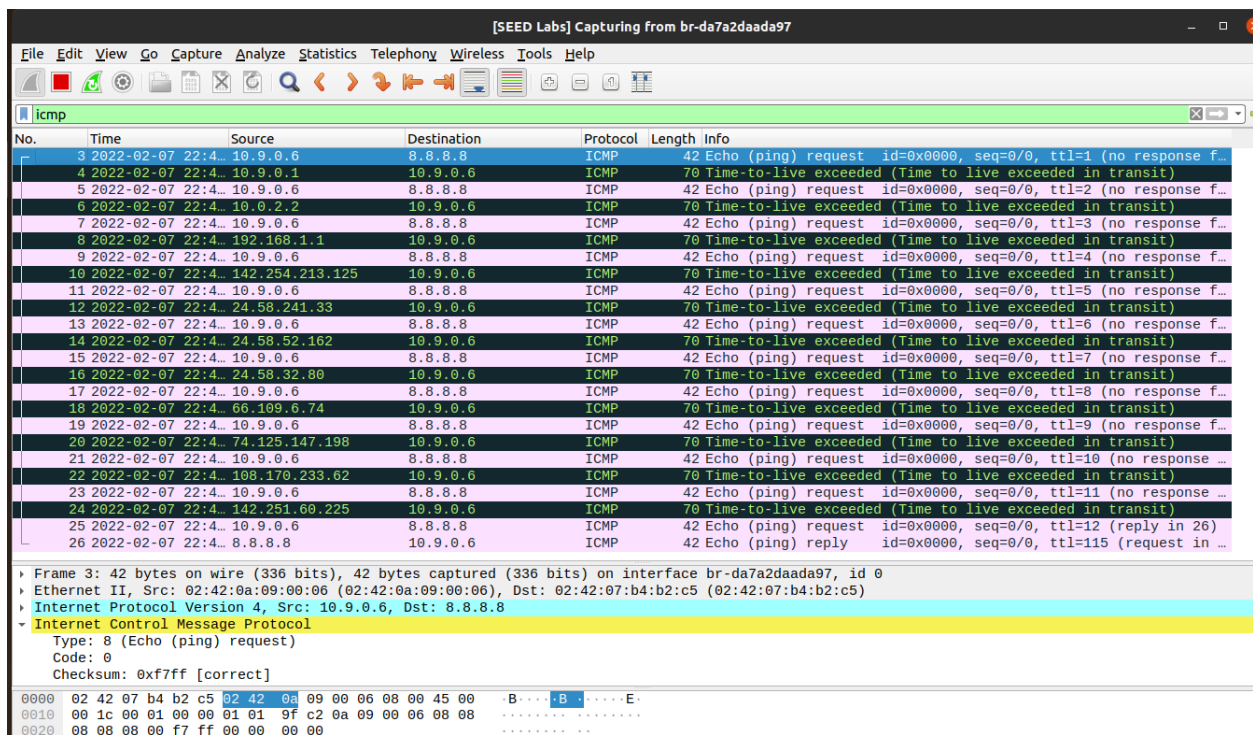
*

Received 1 packets, got 1 answers, remaining 0 packets
source IP - 8.8.8.8

Distance : 12

root@458b7d0f4813:/# nano spoof.py

Using wireshark I was able to actually see on a graphical interface a trace of the packet from source to destination after passing through different routers. In this case, the distance between the source and the destination came out to be 12 hops.



No.	Time	Source	Destination	Protocol	Length	Info
3	2022-02-07 22:4...	10.9.0.6	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=1 (no response f...
4	2022-02-07 22:4...	10.9.0.1	10.9.0.6	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
5	2022-02-07 22:4...	10.9.0.6	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=2 (no response f...
6	2022-02-07 22:4...	10.0.2.2	10.9.0.6	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
7	2022-02-07 22:4...	10.9.0.6	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=3 (no response f...
8	2022-02-07 22:4...	192.168.1.1	10.9.0.6	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
9	2022-02-07 22:4...	10.9.0.6	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=4 (no response f...
10	2022-02-07 22:4...	142.254.213.125	10.9.0.6	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
11	2022-02-07 22:4...	10.9.0.6	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=5 (no response f...
12	2022-02-07 22:4...	24.58.241.33	10.9.0.6	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
13	2022-02-07 22:4...	10.9.0.6	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=6 (no response f...
14	2022-02-07 22:4...	24.58.52.162	10.9.0.6	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
15	2022-02-07 22:4...	10.9.0.6	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=7 (no response f...
16	2022-02-07 22:4...	24.58.32.80	10.9.0.6	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
17	2022-02-07 22:4...	10.9.0.6	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=8 (no response f...
18	2022-02-07 22:4...	66.109.6.74	10.9.0.6	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
19	2022-02-07 22:4...	10.9.0.6	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=9 (no response f...
20	2022-02-07 22:4...	74.125.147.198	10.9.0.6	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
21	2022-02-07 22:4...	10.9.0.6	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=10 (no response ...
22	2022-02-07 22:4...	108.170.233.62	10.9.0.6	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
23	2022-02-07 22:4...	10.9.0.6	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=11 (no response ...
24	2022-02-07 22:4...	142.251.60.225	10.9.0.6	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
25	2022-02-07 22:4...	10.9.0.6	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=12 (reply in 26)
26	2022-02-07 22:4...	8.8.8.8	10.9.0.6	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=115 (request in ...

Frame 3: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface br-da7a2daada97, id 0

Ethernet II, Src: 02:42:0a:09:00:06 (02:42:0a:09:00:06), Dst: 02:42:07:b4:b2:c5 (02:42:07:b4:b2:c5)

Internet Protocol Version 4, Src: 10.9.0.6, Dst: 8.8.8.8

Internet Control Message Protocol

Type: 8 (Echo (ping) request)

Code: 0

Checksum: 0xf7ff [correct]

0000 02 42 07 b4 b2 c5 02 42 0a 09 00 06 08 00 45 00 .B....B.....E.

0010 00 1c 00 01 00 00 01 01 9f c2 0a 09 00 06 08 08

0020 08 08 00 00 f7 ff 00 00 00 00

Task 1.4: In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-and-then-spoof program. From the user container, you ping an IP X. This will generate an ICMP echo request packet. If X is alive, the ping program will receive an echo reply, and print out the response. Your sniff-and-then-spoof program runs on the VM, which monitors the LAN through packet sniffing. Whenever it sees an ICMP echo request, regardless of what the target IP address is, your program should immediately send out an echo reply using the packet spoofing technique. Therefore, regardless of whether machine X is alive or not, the ping program will always receive a reply, indicating that X is alive. You need to use Scapy to do this task.

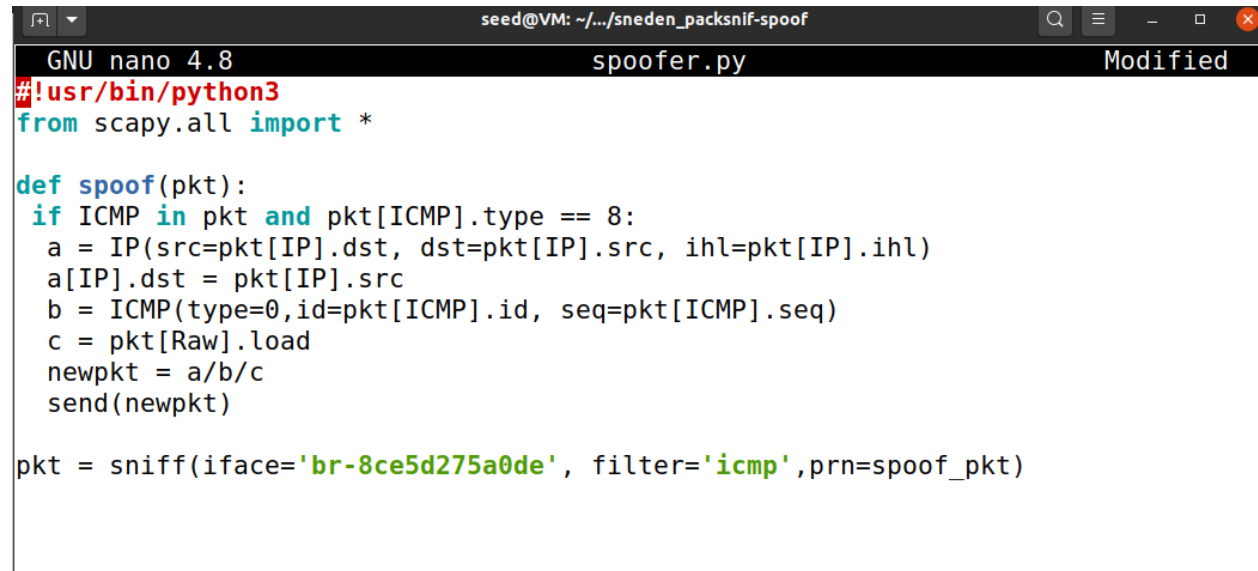
You should ping the following three IP addresses

ping 1.2.3.4 # a non-existing host on the Internet

ping 10.9.0.99 # a non-existing host on the LAN

ping 8.8.8.8 # an existing host on the Internet

The sniffing and spoofing code is implemented in the below code.



```
GNU nano 4.8                                spoofer.py                                Modified
#!/usr/bin/python3
from scapy.all import *

def spoof(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        a = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        a[IP].dst = pkt[IP].src
        b = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        c = pkt[Raw].load
        newpkt = a/b/c
        send(newpkt)

pkt = sniff(iface='br-8ce5d275a0de', filter='icmp', prn=spoof_pkt)
```

The program sniffs ICMP packets and generates and sends a faked ICMP echo reply if it is an ICMP echo request (type 8).

Ping 1.2.3.4

We run the spoofer.py program and then use the Host A (10.9.0.5) to ping a non-existing host on the Internet – 1.2.3.4 and find that the ping is successful due to the spoofed echo return, giving the impression that the host is reachable.

The screenshot shows a terminal window with two panes. The left pane shows the output of a ping command from Host A (10.9.0.5) to 1.2.3.4. The right pane shows the output of the 'spoofer.py' program, which is sending spoofed echo replies to the ping requests. Below the terminal panes, a packet capture analysis is displayed, showing the sequence of packets and the spoofed echo replies.

```
root@da85ab557762:/# 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=8.78 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=6.90 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=3.44 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=7.54 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=4.79 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=5.95 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=6.43 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=6.14 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=7.56 ms
64 bytes from 1.2.3.4: icmp_seq=10 ttl=64 time=4.89 ms
64 bytes from 1.2.3.4: icmp_seq=11 ttl=64 time=5.71 ms
64 bytes from 1.2.3.4: icmp_seq=12 ttl=64 time=5.08 ms
64 bytes from 1.2.3.4: icmp_seq=13 ttl=64 time=3.47 ms
64 bytes from 1.2.3.4: icmp_seq=14 ttl=64 time=6.06 ms
64 bytes from 1.2.3.4: icmp_seq=15 ttl=64 time=7.13 ms
64 bytes from 1.2.3.4: icmp_seq=16 ttl=64 time=5.72 ms
64 bytes from 1.2.3.4: icmp_seq=17 ttl=64 time=5.20 ms
64 bytes from 1.2.3.4: icmp_seq=18 ttl=64 time=6.88 ms
64 bytes from 1.2.3.4: icmp_seq=19 ttl=64 time=6.38 ms
64 bytes from 1.2.3.4: icmp_seq=20 ttl=64 time=7.10 ms
64 bytes from 1.2.3.4: icmp_seq=21 ttl=64 time=4.78 ms
```

```
bin dev home lib32 libx32 mnt proc run spoofer.py sys usr volumes
boot etc lib lib64 media opt root sbin srv tmp var
root@VM:/# nano spoofer.py
root@VM:/# python3 spoofer.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

No.	Time	Source	Destination	Protocol	Length	Info
1	2022-02-08 19:31:10.905	10.9.0.5	1.2.3.4	ICMP	84	Echo (ping) request id=0x0023, seq=1/256, ttl=64 (reply in 4)
2	2022-02-08 19:31:02:42:f6:6c:cc:94	Broadcast		ARP	42	who has 10.9.0.5? Tell 10.9.0.1
3	2022-02-08 19:31:02:42:f6:6c:cc:94	10.9.0.5	02:42:f6:6c:cc:94	ARP	42	10.9.0.5 is at 02:42:f6:6c:cc:94
4	2022-02-08 19:31:10.905	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0023, seq=1/256, ttl=64 (request in...)
5	2022-02-08 19:31:10.905	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) request id=0x0023, seq=2/512, ttl=64 (reply in 0)
6	2022-02-08 19:31:10.905	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0023, seq=2/512, ttl=64 (request in...)
7	2022-02-08 19:31:10.905	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) request id=0x0023, seq=3/768, ttl=64 (reply in 0)
8	2022-02-08 19:31:10.905	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0023, seq=3/768, ttl=64 (request in...)
9	2022-02-08 19:31:10.905	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) request id=0x0023, seq=4/1024, ttl=64 (reply in ...)
10	2022-02-08 19:31:10.905	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0023, seq=4/1024, ttl=64 (request in...)
11	2022-02-08 19:31:10.905	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) request id=0x0023, seq=5/1280, ttl=64 (reply in ...)
12	2022-02-08 19:31:10.905	1.2.3.4	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0023, seq=5/1280, ttl=64 (request in...)

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-8ce5d275a0de, id 0
Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:f6:6c:cc:94 (02:42:f6:6c:cc:94)
Internet Protocol Version 4, Src: 10.9.0.5, Dst: 1.2.3.4
Internet Control Message Protocol

Ping 10.9.0.99

We run the spoofer.py program and then use the Host A (10.9.0.5) to ping a non-existing host on the same LAN – 10.9.0.99 and find that the ping is not successful, and Host is unreachable. This happens because the host does not exist, and a router would not give a reply. The ARP protocol keeps broadcasting for information about 10.9.0.99 with no reply.

```
seed@VM: ~/.../sneden_packetsnif-spoof seed@VM: root@VM: /# python3 spoofer.py
^Z
[1]+  Stopped                  ping 1.2.3.4
root@7dcbbf2eade4:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
From 10.9.0.5 icmp_seq=4 Destination Host Unreachable
From 10.9.0.5 icmp_seq=5 Destination Host Unreachable
From 10.9.0.5 icmp_seq=6 Destination Host Unreachable
From 10.9.0.5 icmp_seq=7 Destination Host Unreachable
From 10.9.0.5 icmp_seq=8 Destination Host Unreachable
From 10.9.0.5 icmp_seq=9 Destination Host Unreachable
From 10.9.0.5 icmp_seq=10 Destination Host Unreachable
From 10.9.0.5 icmp_seq=11 Destination Host Unreachable
From 10.9.0.5 icmp_seq=12 Destination Host Unreachable
From 10.9.0.5 icmp_seq=13 Destination Host Unreachable
From 10.9.0.5 icmp_seq=14 Destination Host Unreachable
From 10.9.0.5 icmp_seq=15 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
17 packets transmitted, 0 received, +15 errors, 100% packet loss in time 0.000s
pipe 4
root@7dcbbf2eade4:/#
```

[SEED Labs] Capturing from br-da7a2daada97

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

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

No.	Time	Source	Destination	Protocol	Length	Info
1	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
2	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
3	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
4	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
5	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
6	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
7	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
8	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
9	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
10	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
11	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
12	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5
13	2022-02-07 23:4...	02:42:0a:09:00:05	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.5

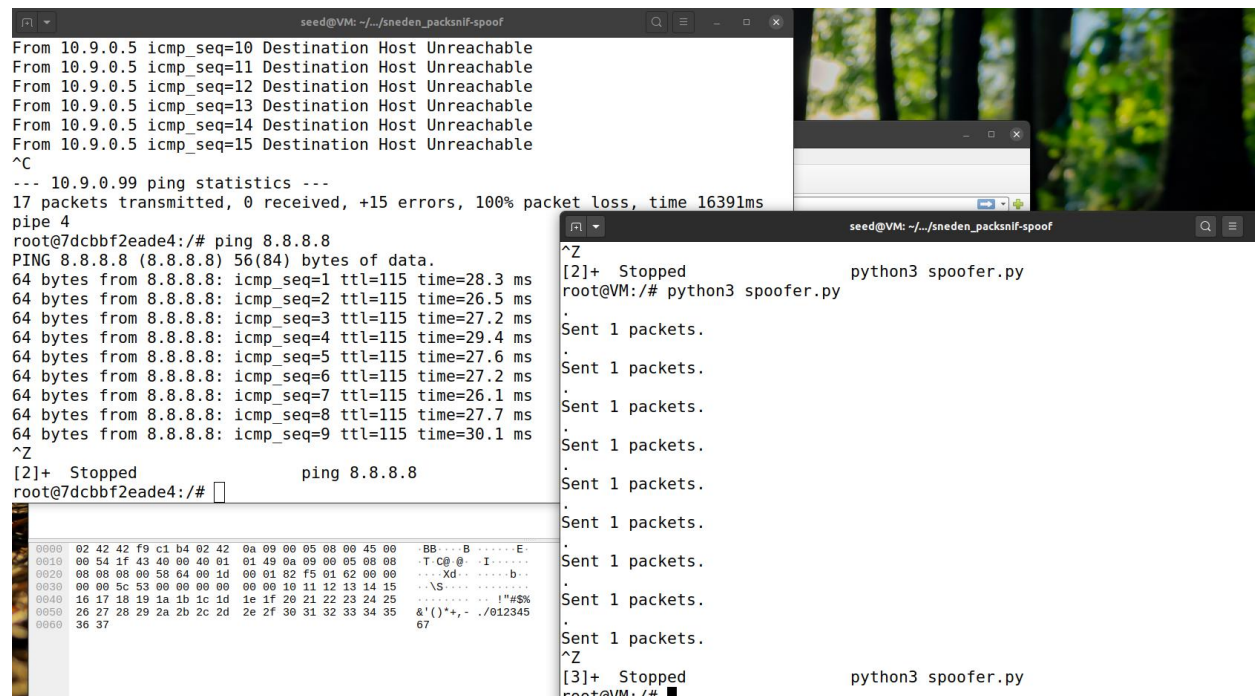
Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface br-da7a2daada97, id 0

Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: Broadcast (ff:ff:ff:ff:ff:ff)

Address Resolution Protocol (request)

```
0000  ff ff ff ff ff 02 42  0a 09 00 05 08 06 00 01  .....B .....
0010  08 00 06 04 00 01 42  0a 09 00 05 0a 09 00 05  .....B .....
```

We run the spoofer.py program and then use the Host A (10.9.0.5) to ping an existing host, google server 8.8.8.8 on the Internet and find that the ping is successful due to the host actually existing and receives a genuine response after every request from the router.



The image shows a Wireshark packet capture window. The title bar reads "[SEED Labs] Capturing from br-da7a2daada97". The menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. The toolbar contains various icons for file operations, capture control, and analysis. The packet list pane shows 16 captured packets, with the first 12 being ICMP Echo (ping) requests and replies, and the last 4 being ARP requests. The packet details pane for the selected packet (No. 12) shows the following structure:

- Ethernet II, Src: 02:42:0a:09:00:05, Dst: 02:42:42:f9:c1:b4 (02:42:42:f9:c1:b4)
- Internet Protocol Version 4, Src: 10.9.0.5, Dst: 8.8.8.8
- Internet Control Message Protocol

The packet bytes pane shows the raw data in hexadecimal and ASCII format:

```

0000  02 42 42 f9 c1 b4 02 42 0a 09 00 05 08 00 45 00  .BB....B....E.
0010  00 54 1f 43 49 00 40 01 01 49 0a 09 00 05 08 08  -T C@...I...
0020  08 08 08 00 58 64 00 1d 00 01 82 f5 01 62 00 00  ...Xd...b...
0030  00 00 5c 53 09 00 00 00 00 00 18 11 12 13 14 15  ...S...

```