Q1 Lab 2 - Packet Sniffing and Spoofing Lab

15 Points

Lab PDF: Packet Sniffing and Spoofing Lab

Note: For this lab, please only complete Task 1.1 - Task 1.4 (i.e. the scapy portions) from "3 Lab Task Set 1: Using Scapy to Sniff and Spoof Packets" in the SEED Labs PDF. Stop on page 7.

- Lab setup files: Packet Sniffing and Spoofing Lab - **Please use this Labsetup for this lab. Make sure to take down the docker containers from Lab 1, and install the ones for Lab 2.** You should use `Labsetup.zip` unless you're on Apple Silicon in which you should use `Labsetup-arm.zip`
- Docker Manual (if more help is needed)

Note: Only images or TXT files are acceptable upload formats

Normal due date: **5 Mar**
Earliest acceptance date: 29 Feb (+1% per day, up to +5% bonus for five days early)
Latest acceptance date: 13 Mar (-8% points)

Please follow the instructions from the PDF document, but submit your work based on the instructions below. For any code snippets, use the ones from Gradescope as they would have any corrections.

Q1.1 Task 1.1: Sniffing Packets - Task 1.1A
0 Points

```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
   pkt.show()
pkt = sniff(iface='br-c93733e9f913', filter='icmp', prn=print_pkt)
```

```
// Make the program executable
# chmod a+x sniffer.py
// Run the program with the root privilege
# sniffer.py
// Switch to the "seed" account, and
// run the program without the root privilege
# su seed
$ sniffer.py
```

Nothing need to be submitted for Task 1.1.

Q1.2 Task 1.1B(a) - Only ICMP
2.5 Points

Using `sniff(filter='___')`, capture only ICMP packets.

Select the option that goes in the blank

```
def print_pkt(pkt):
      pkt.show()
pkt = sniff(filter=      (1)      ,prn=print_pkt)
```

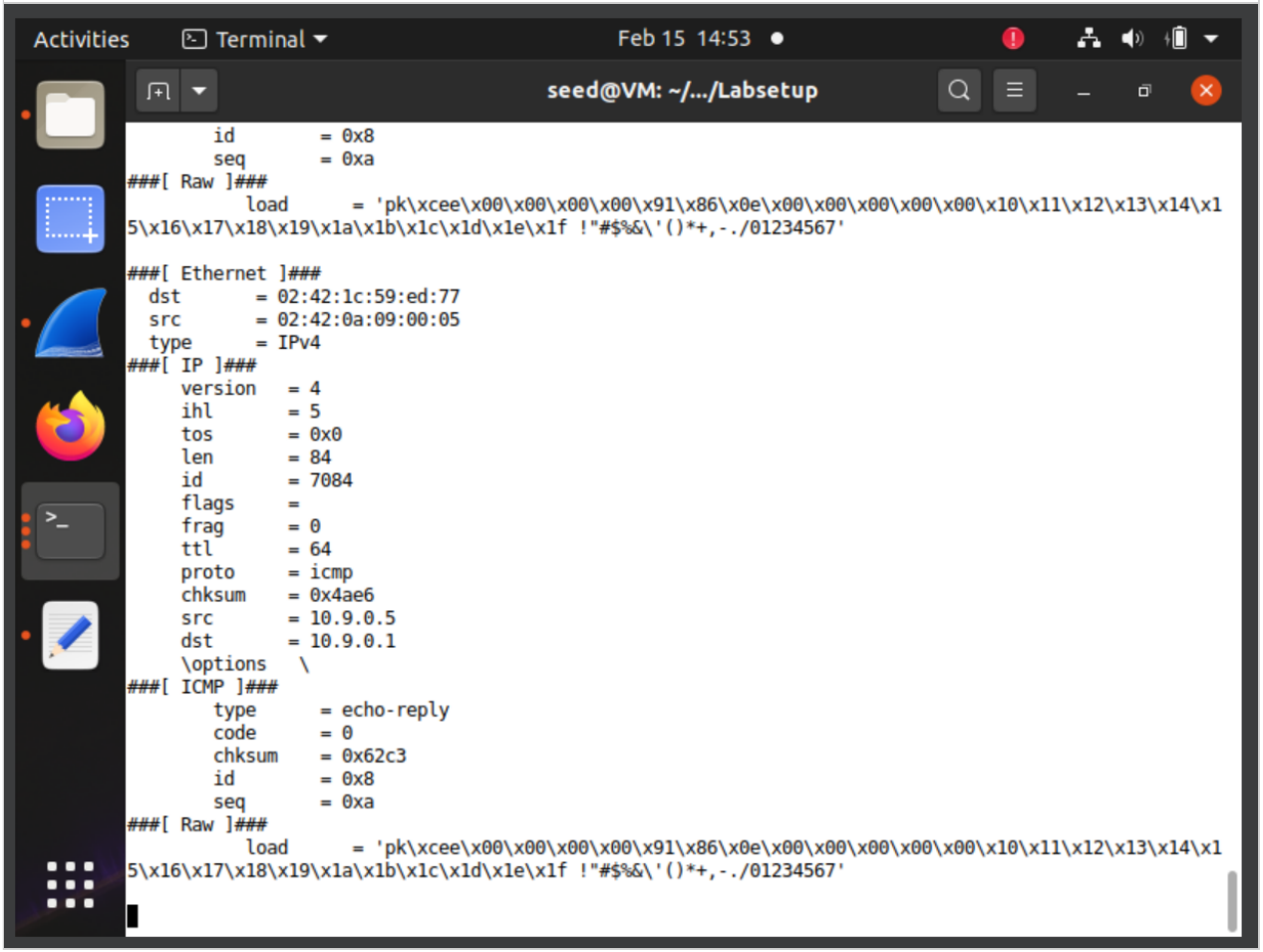
- ○ ping
- ◉ icmp
- ○ tcp

Q1.3 Task 1.1B(a) - Only ICMP (Screenshot)
2.5 Points

Take a screenshot demonstrating that ICMP packets are captured (must screenshot the entire VM along with date and time).

Hint: Be sure to adjust the `iface` *correctly when working locally*

▾ **ICMP packets capture.png** Download

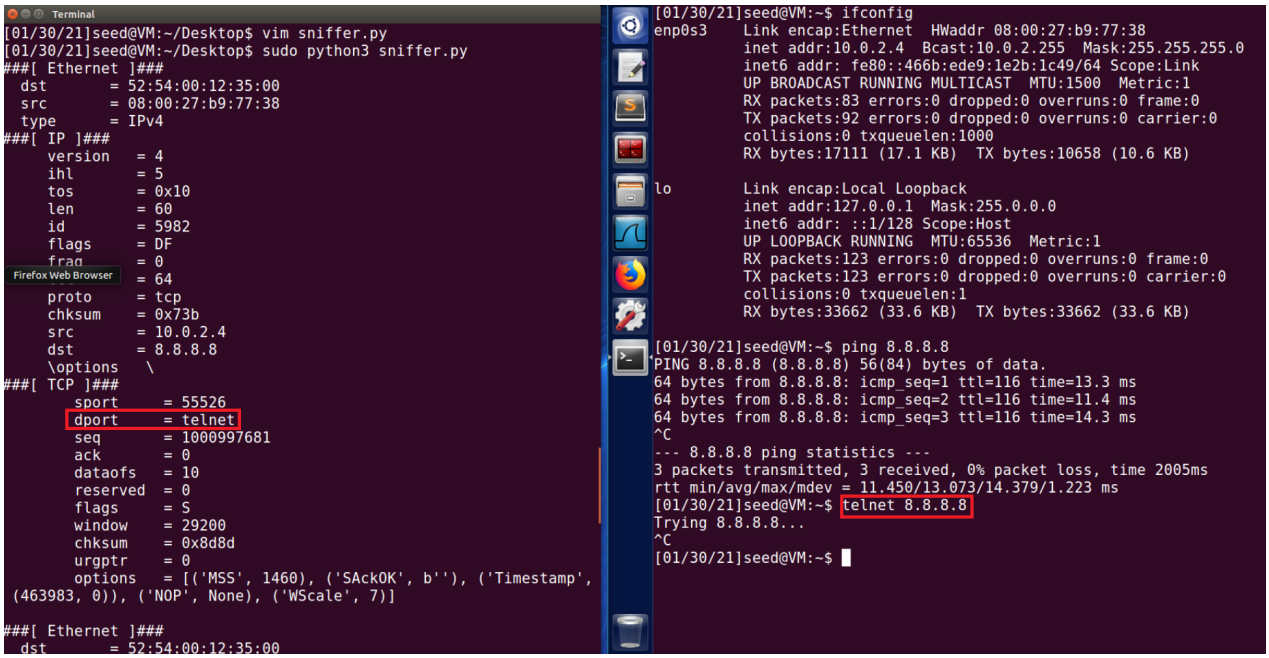


Q1.4 Task 1.1B(b) - Capture specific TCP/port 23
2.5 Points

Capture any TCP packet that comes from a specific IP and with a destination port number 23. Your code must include the following: (1) capture TCP; (2) capture from any specific IP address; and (3) capture to destination port 23.

Hint: Use the attacker machine and write your code for sniffing. Then generate telnet traffic from the attacker machine to test that your program works.

Expected output



Select the option that goes in the blank

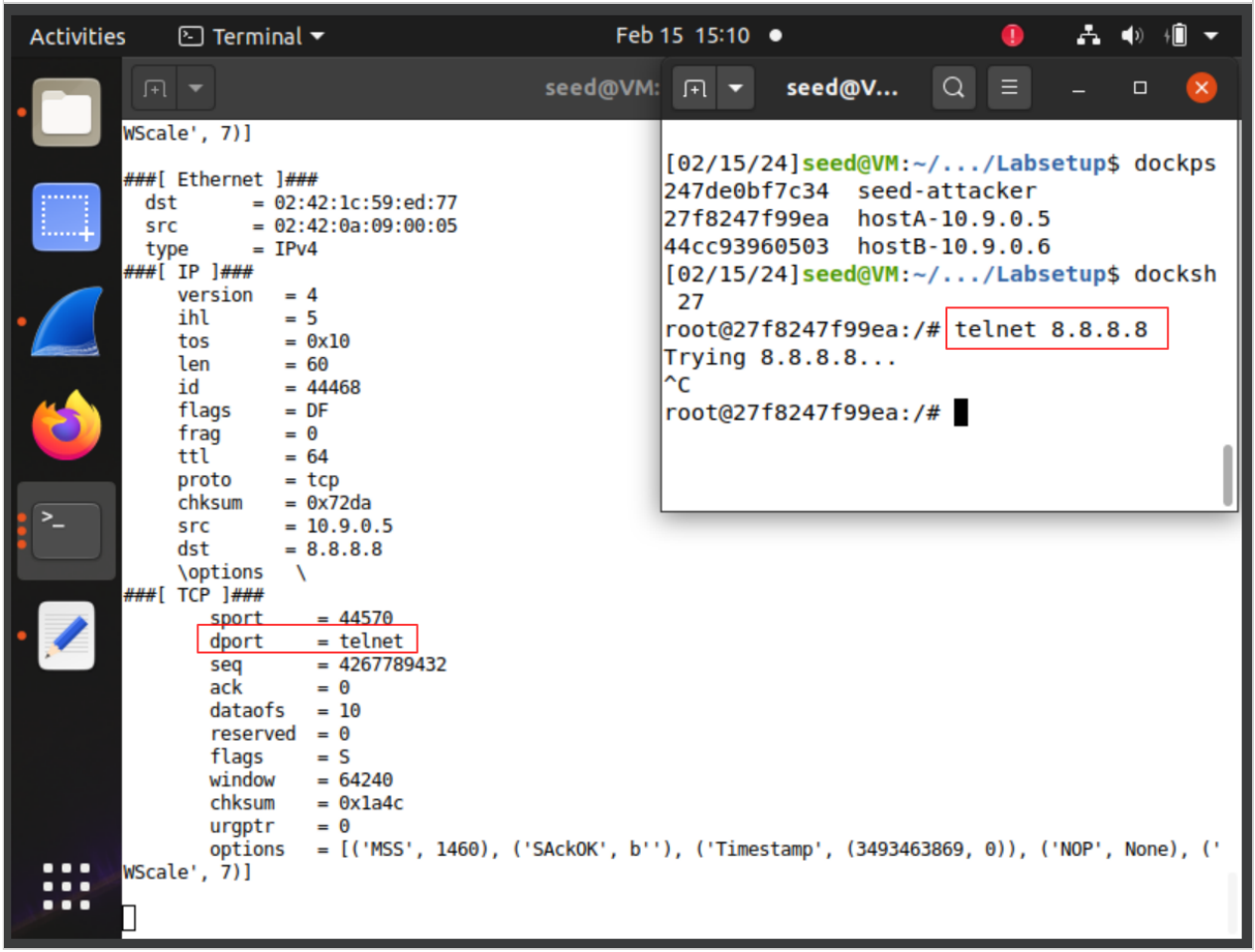
```
def print_pkt(pkt):
      pkt.show()
pkt = sniff(filter=      (1)      ,prn=print_pkt)
```

- ○ tcp port 23
- ○ host 10.9.0.3 and port 23
- ◉ tcp and host 10.9.0.1 and port 23
- ○ icmp and host 10.9.0.2 and port 23

Q1.5 Task 1.1B(b) - Capture specific TCP/port 23 (Screenshot)
2.5 Points

Take a screenshot demonstrating that that the TCP/port 23 packet was captured (must screenshot the entire VM along with date and time).

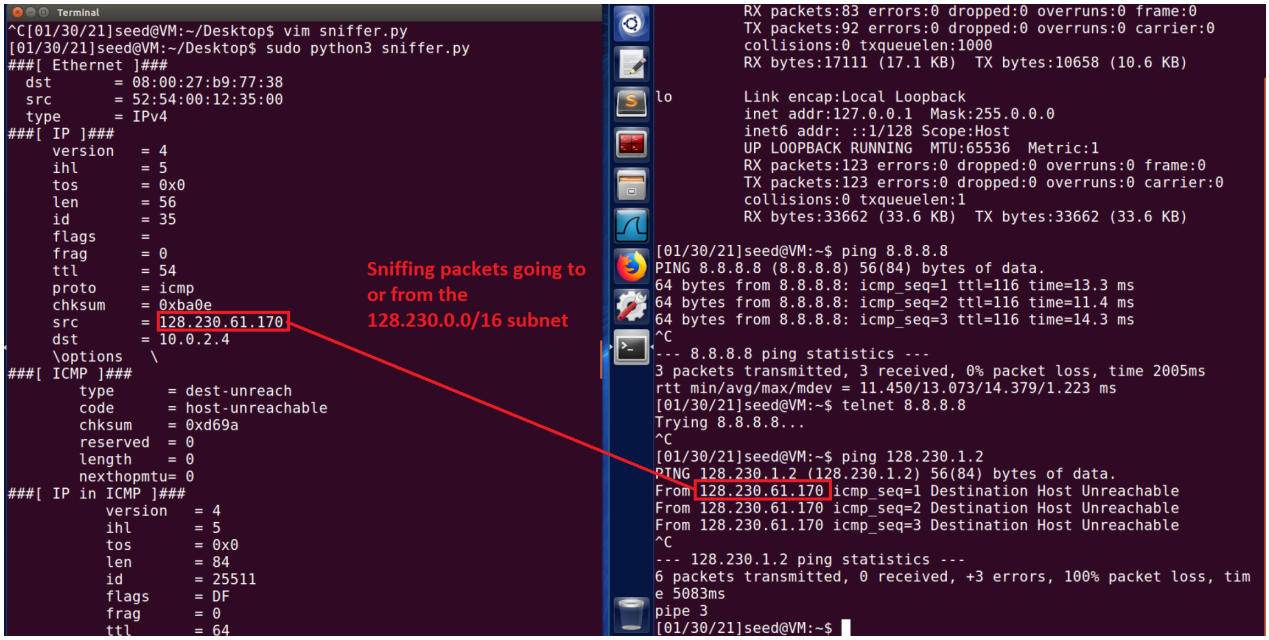
▾ Capture of specific TCP-subnet-port labelled.png Download



Q1.6 Task 1.1B(c) - Particular Subnet
2.5 Points

Capture packets coming from or going to a subnet 10.9.0.0/24. Fill in the blank and take a screenshot demonstrating the traffic generated.

Expected output



Select the option that goes in the blank

Note: Use subnet 10.9.0.0/24

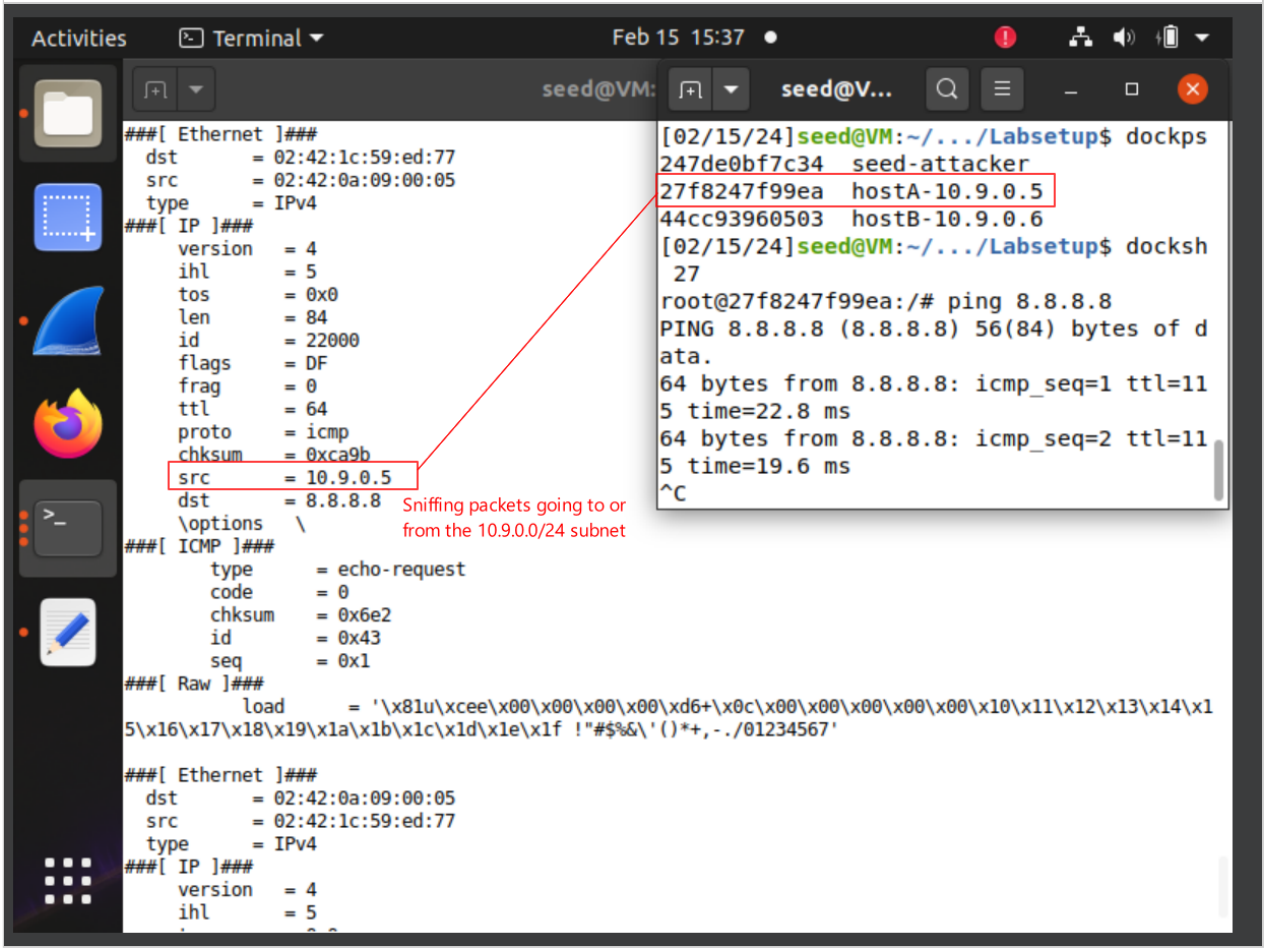
```
def print_pkt(pkt):
      pkt.show()
pkt = sniff(filter=      (1)      ,prn=print_pkt)
```

- (●) net 10.9.0.0/24
- ( ) net 10.9.0.0
- ( ) net 10.9.0.0/26
- ( ) net 128.238.0.0/24

Q1.7 Task 1.1B(c) - Particular Subnet (Screenshot)
2.5 Points

Take a screenshot demonstrating that that packets from the 10.9.0.0/24 subnet were captured (must screenshot the entire VM along with date and time).

▾ Capture of particular subnet labelled.png Download



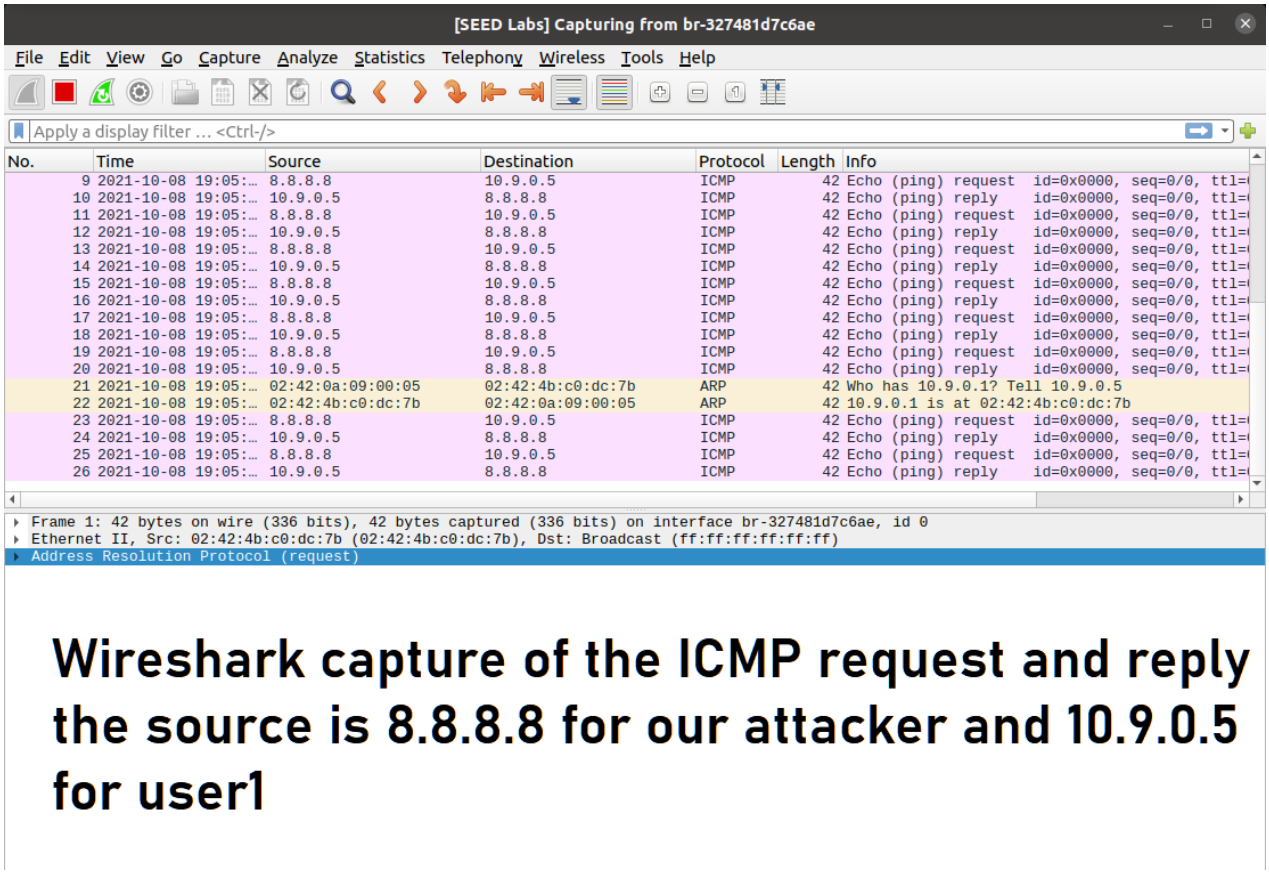
Q2 Task 1.2: Spoofing ICMP Packets

15 Points

```
>>> from scapy.all import *
>>> a = IP()
>>> a.dst = '10.0.2.3'
>>> b = ICMP()
>>> p = a/b
>>> send(p)
.
Sent 1 packets.
```

Spoof an ICMP echo request packet with source IP address 8.8.8.8 from the first VM and send to the second VM. Use Wireshark on the second VM to show that it replies back with echo replies.

Expected wireshark output



Wireshark capture of the ICMP request and reply the source is 8.8.8.8 for our attacker and 10.9.0.5 for user1

Q2.1 Task 1.2 - Scapy code
10 Points

```
from scapy.all import *
ip = IP(src=    (1)    , dst=    (2)    )
icmp = ICMP()
packet = ip/icmp
packet.show()
send(packet)
```

a. Based on the code above, what is the IP address for (1)?

8.8.8.8

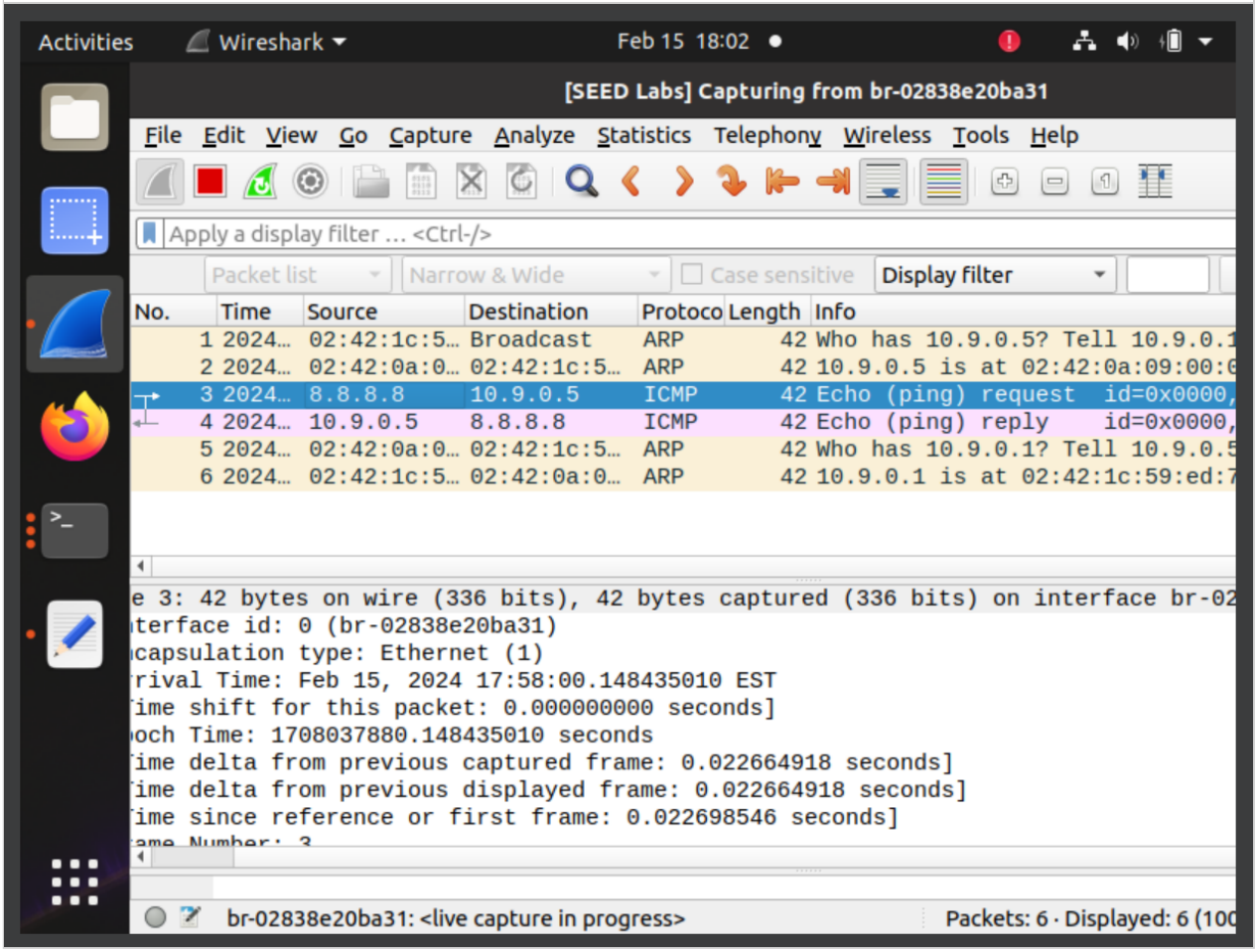
b. Based on the code above, what is the IP address for (2)?

10.9.0.5

Q2.2 Task 1.2 - Submit a screenshot of Wireshark showing the spoof echo request from 8.8.8.8 and that the VM replied back to it with an echo reply.
5 Points

(must screenshot the entire VM along with date and time)

▾ Spoofing ICMP packets.png Download



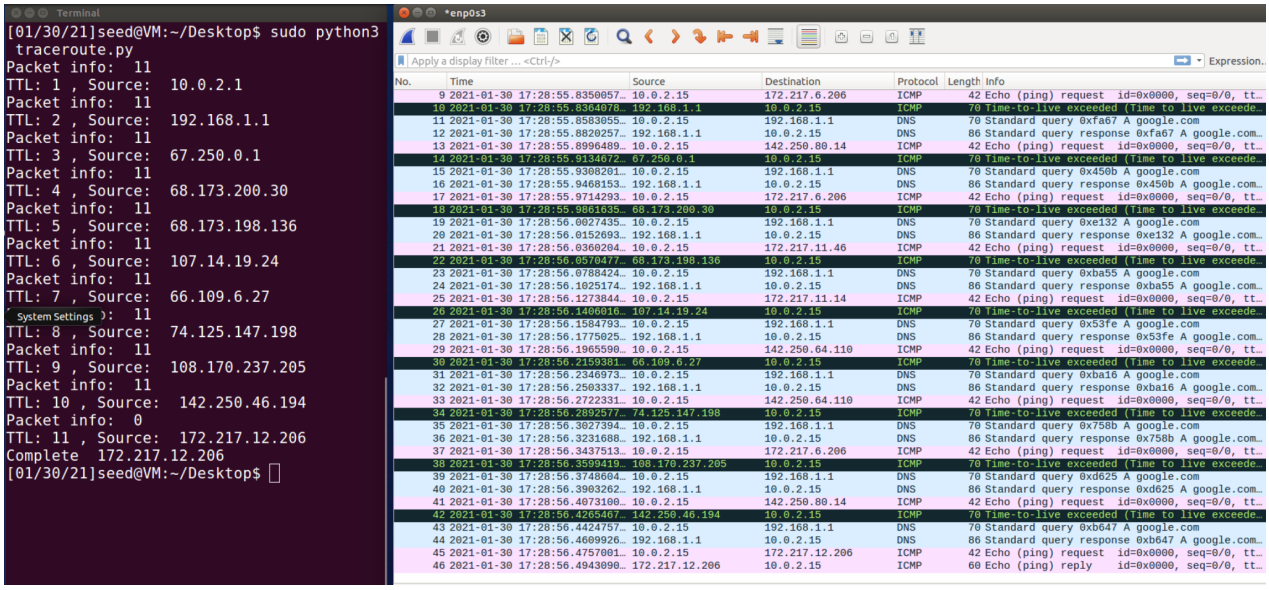
Q3 Task 1.3: Fully-automated Traceroute

20 Points

Using the skeleton code below, implement ICMP traceroute using scapy. Do NOT use the built-in scapy traceroute function. Perform a traceroute to 8.8.8.8. Show proof using a Wireshark capture and take a screenshot of your program's output.

Expected output

Note: you can see we finally got a reply in the end in wireshark



Q3.1 Task 1.3 - Scapy code
15 Points

```
from scapy.all import *

ttl = (1)
while True:
      a = IP(dst=(2), ttl=ttl)
      b = ICMP()
      p = a/b
      pkt = sr1(p, verbose = 0)

      if (3) == 0:
            print("Complete ", pkt[IP].src)
            break
      else:
            print("TTL: %d, Source: " %ttl , pkt[IP].src)
            ttl +=1
```

a. Fill in the blank for 1.

1

b. Fill in the blank for 2.

"8.8.8.8"

c. Fill in the blank for 3.

pkt[ICMP].type

Q3.2 Task 1.3 - Write your own trace route program
5 Points

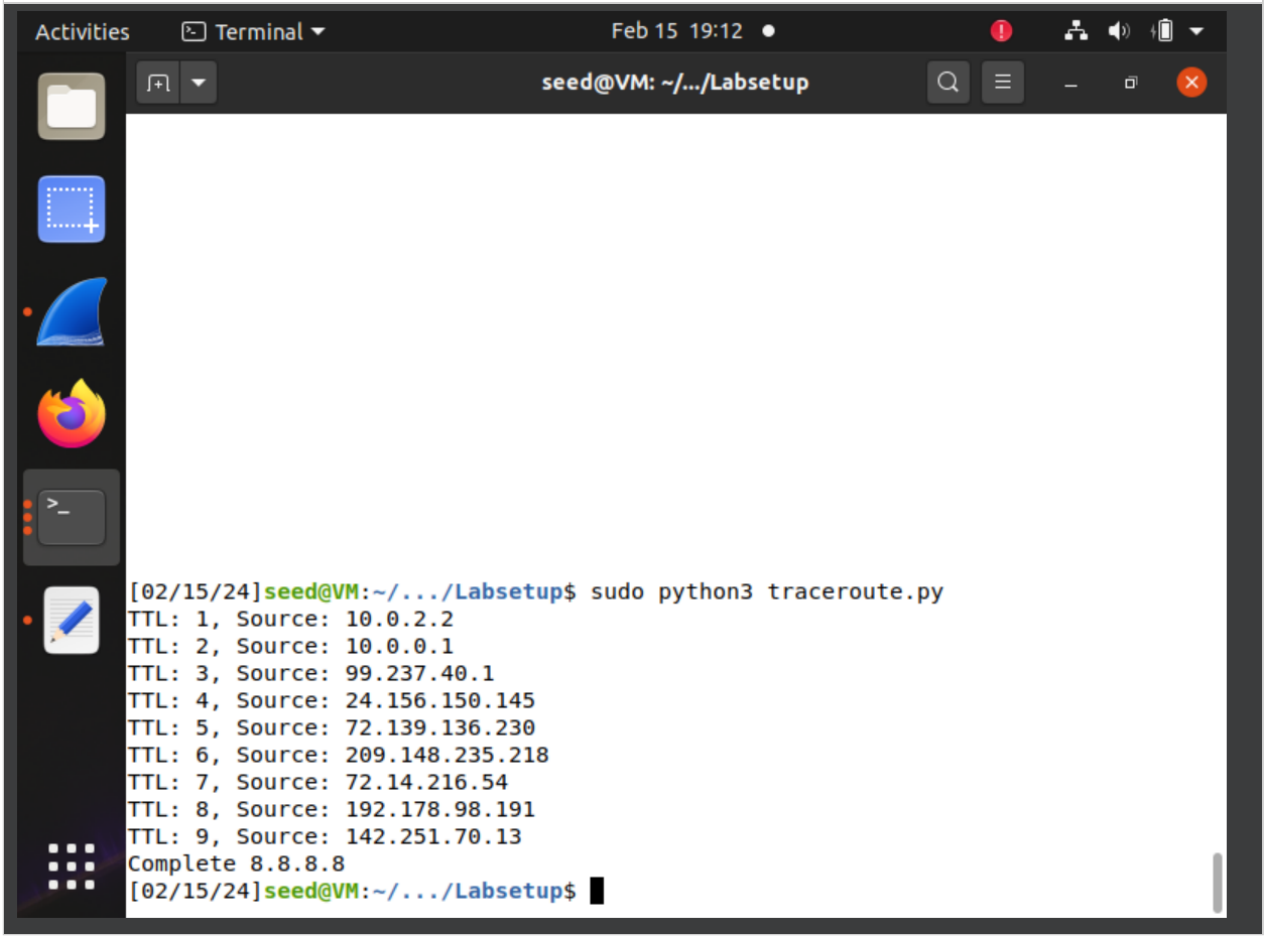
Write your own traceroute program using the skeleton code above. Test your traceroute program by tracerouting 8.8.8.8. Note: You may need to add additional code to handle conditions where the router does not respond. **Your code must fully automate automate the trace route for credit.**

For Q3.2 Task 1.3, submit the following:

- Your traceroute program
- Screenshot of Wireshark showing proof of traceroute to 8.8.8.8
- Your program output
- Screenshot of built-in Ubuntu traceroute to 8.8.8.8 (`traceroute -I 8.8.8.8`)

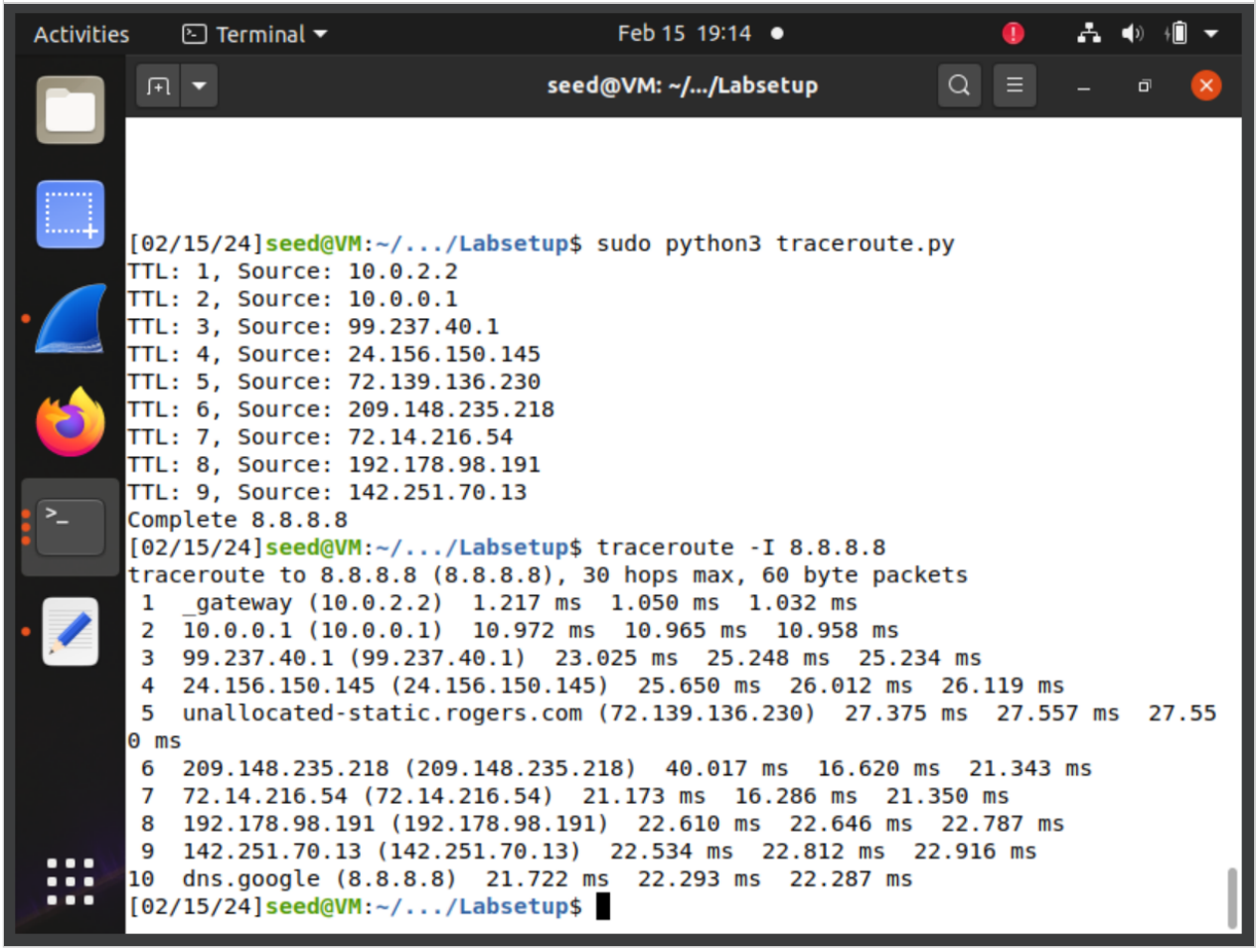
Submit a screenshot of Wireshark **(must screenshot the entire VM along with date and time)**

Traceroute program output.png Download

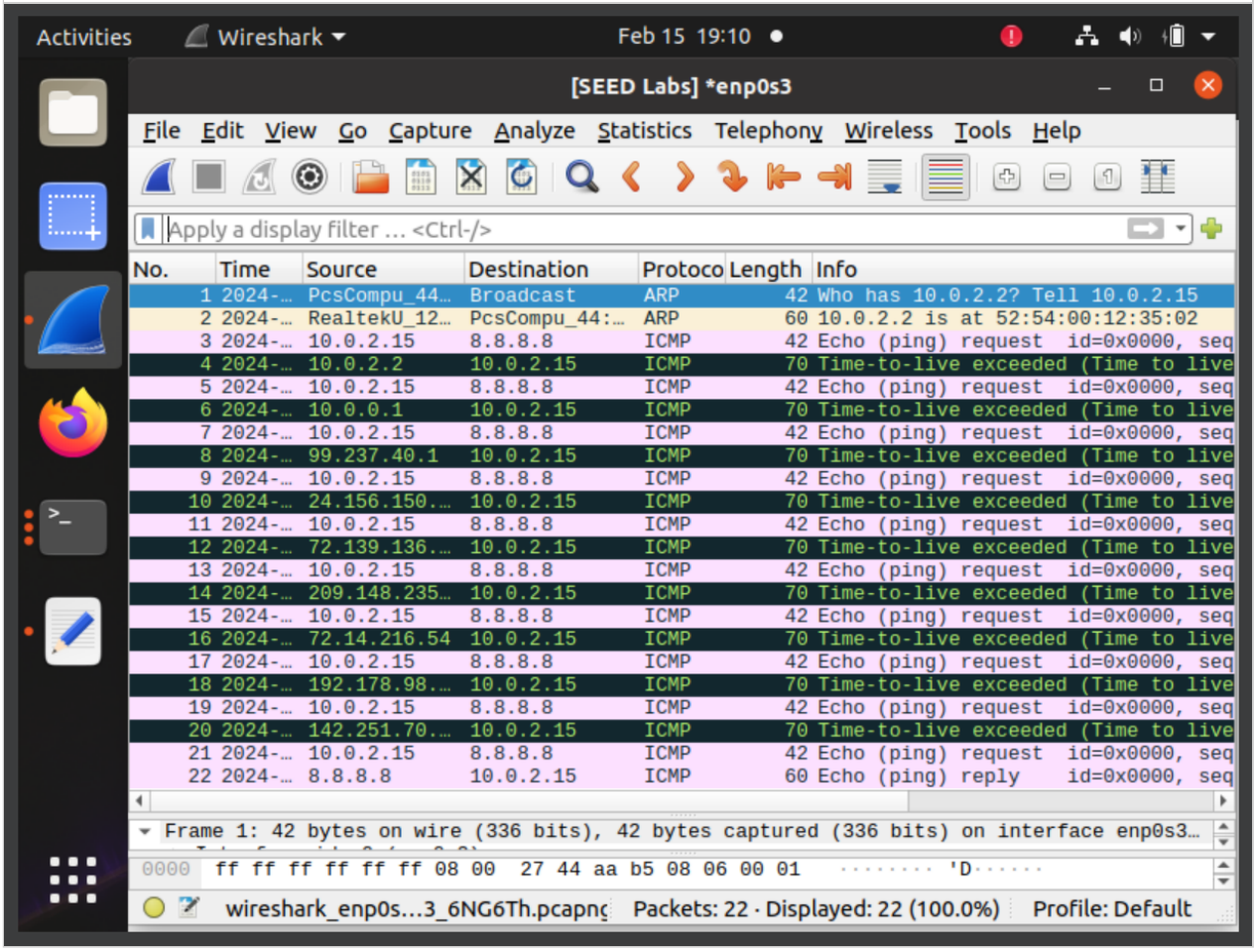


Traceroute Ubuntu output.png

Download



Traceroute wireshark.png Download



traceroute.py

Download

```
1  from scapy.all import *
2
3  ttl=1
4  while True:
5    ip = IP(dst='8.8.8.8', ttl=ttl)
6    icmp = ICMP()
7    p = ip/icmp
8
9    resp = sr1(p, verbose=0, timeout=5)
10
11   if (resp==None):
12     print("TTL: {0}, Source: ???".format(ttl))
13     ttl+=1
14   elif (resp[ICMP].type==0):
15     print("Complete", resp[IP].src)
16     break
17   else:
18     print("TTL: {0}, Source: {1}".format(ttl, resp[IP].src))
19     ttl+=1
20
```

Q4 Task 1.4: Sniffing and-then Spoofing
50 Points

Sniffing and-then Spoofing. You need two machines on the same LAN: the VM and the user container.

You will find that when you ping from the terminal, 10.9.0.99 will have a destination unreachable response. That is the expected result. Your program does not need to work for this IP. (You do not need to force it to work by performing ARP spoofing.) You will, however, need to explain why your program does not work for IP 10.9.0.99 (while it's suppose to work for 1.2.3.4 and 8.8.8.8).

Q4.1 Task 1.4 - Sniffing and then spoofing program
20 Points

Using the skeleton code below, implement the sniffing and the spoofing program.

```
from scapy.all import *
def sniff_and_spoof(packet):
      if ICMP in packet:

            ip = IP(src=      (1)      , dst=      (2)      )
            icmp = ICMP(type=      (3)      , id=      (4)      , seq=      (5)      )

            raw_data=      (6)
            newpacket=      (7)

            send(newpacket,verbose=0)

pkt = sniff(filter=      (8)      ,prn=sniff_and_spoof)
```

a. Fill in the blank for 1.

packet[IP].dst

b. Fill in the blank for 2.

packet[IP].src

c. Fill in the blank for 3.

0

d. Fill in the blank for 4.

packet[ICMP].id

e. Fill in the blank for 5.

packet[ICMP].seq

f. Fill in the blank for 6.

```
packet[Raw].load
```

g. Fill in the blank for 7.

```
ip/icmp/raw_data
```

h. [10 pts] Fill in the blank for 8. **You must filter out the packets that are sent out by your program. This question is worth ten points.**

```
"not ether src 02:42:1c:59:ed:77"
```

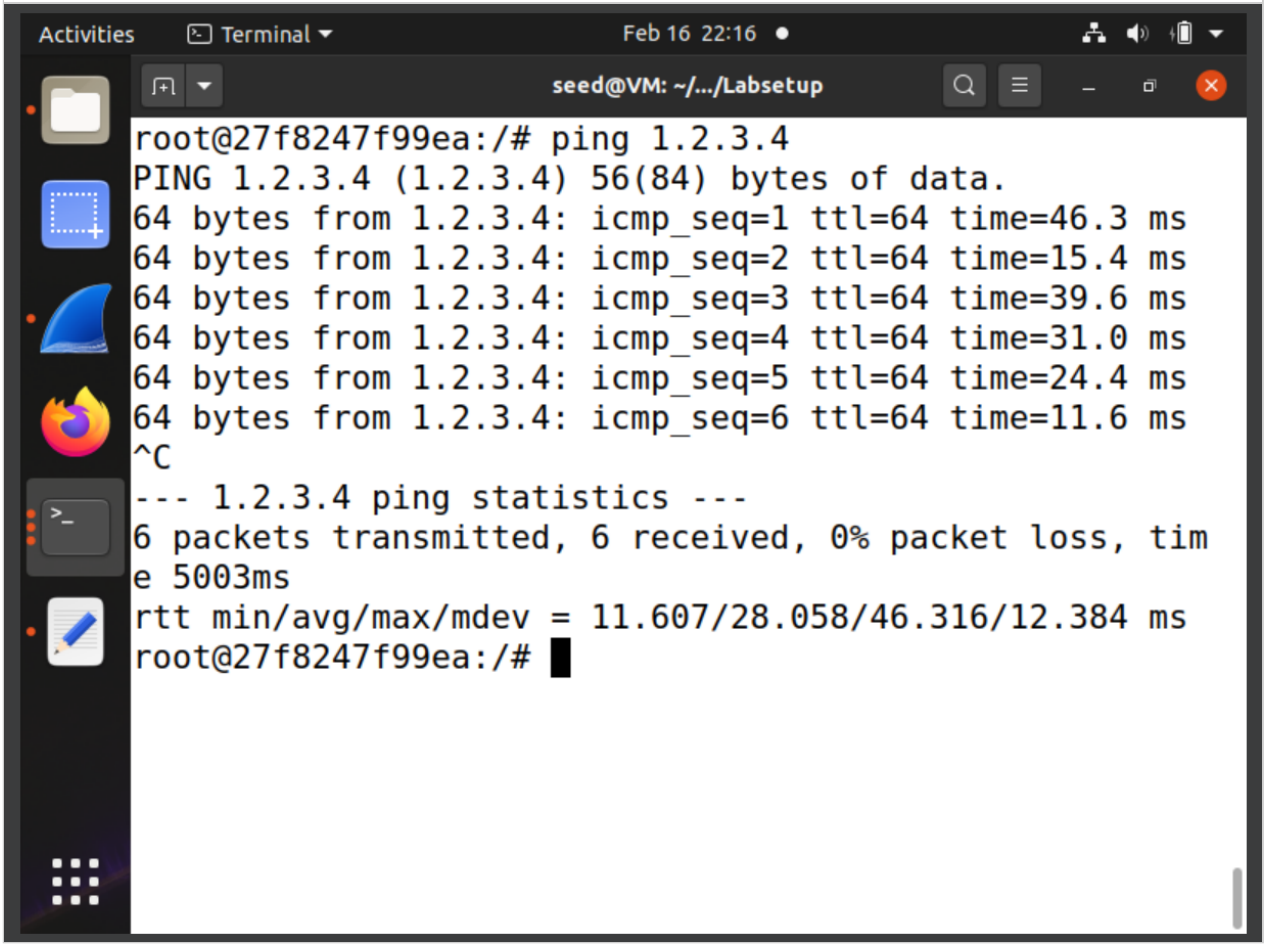
Q4.2 Task 1.4 - Ping 1.2.3.4 and show screenshots of the output from your program and with the ping from the terminal
10 Points

Expected output for 1.2.3.4

```
64 bytes from 1.2.3.4: icmp_seq=29 ttl=64 time=35.4 ms
64 bytes from 1.2.3.4: icmp_seq=30 ttl=64 time=17.0 ms
64 bytes from 1.2.3.4: icmp_seq=31 ttl=64 time=15.9 ms
64 bytes from 1.2.3.4: icmp_seq=32 ttl=64 time=28.0 ms
64 bytes from 1.2.3.4: icmp_seq=33 ttl=64 time=27.0 ms
64 bytes from 1.2.3.4: icmp_seq=34 ttl=64 time=27.5 ms
64 bytes from 1.2.3.4: icmp_seq=35 ttl=64 time=15.9 ms
64 bytes from 1.2.3.4: icmp_seq=36 ttl=64 time=19.2 ms
64 bytes from 1.2.3.4: icmp_seq=37 ttl=64 time=25.0 ms
```

Must screenshot the entire VM along with date and time

▾ Sniff and spoof ping 1.2.3.4.png Download



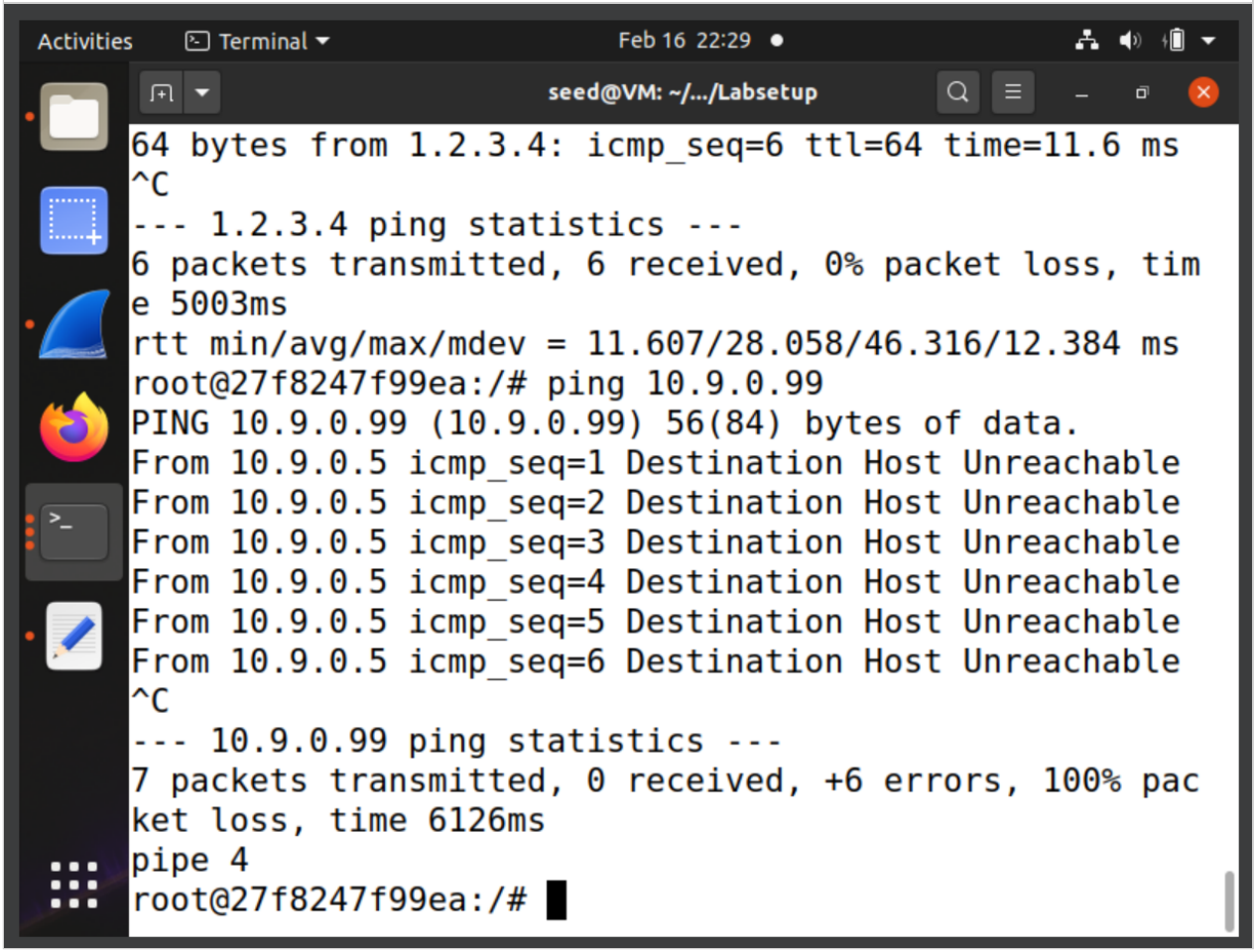
Q4.3 Task 1.4 - Ping 10.9.0.99 and show screenshots of the output from your program and with the ping from the terminal. If this does not work, please explain why.
10 Points

Expected output for 10.9.0.99

```
root@d6d2c918703b:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.7 icmp_seq=1 Destination Host Unreachable
From 10.9.0.7 icmp_seq=2 Destination Host Unreachable
From 10.9.0.7 icmp_seq=3 Destination Host Unreachable
From 10.9.0.7 icmp_seq=4 Destination Host Unreachable
From 10.9.0.7 icmp_seq=5 Destination Host Unreachable
From 10.9.0.7 icmp_seq=6 Destination Host Unreachable
```

Must screenshot the entire VM along with date and time

▾ Sniff and spoof ping 10.9.0.99.png Download



Explain why it does not work. Explain what the host is doing and what protocol is failing to cause this error.

When we type in this command, the attacker VM looks at the IP and realizes that it is on the local subnet. So, the attacker VM will do an ARP request to see the MAC address associated with the IP being pinged. Since the IP does not exist on the subnet, the ARP request fails, and the attacker VM does not even send the ICMP ping request. Since an ICMP ping request is not sent, our program fails, since that is what it replies to.

Q4.4 Task 1.4 - Ping 8.8.8.8 and show screenshots of the output from your program and with the ping from the terminal
10 Points

Expected output for 8.8.8.8

```
64 bytes from 8.8.8.8: icmp_seq=5 ttl=56 time=11.2 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=27.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=6 ttl=56 time=13.4 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=64 time=28.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=7 ttl=56 time=14.8 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=64 time=28.1 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=8 ttl=56 time=15.6 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=64 time=24.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=9 ttl=56 time=11.3 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=64 time=22.1 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=10 ttl=56 time=14.6 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=64 time=24.7 ms (DUP!)
```

Must screenshot the entire VM along with date and time

▾ Sniff and spoof ping 8.8.8.8.png Download