

Communications and Computing – Final Project:

Alon Firestein – 314984402

Hananel Lazar – 312495161

Due Date: 08/03/21

Introduction:

In this document, we have attached numerous explanations for each task in the project, including screenshots and also for particular tasks also highlighted certain details in the screenshots to make it easier to see the successful results.

In addition, we have not screenshotted the Wireshark result for *every* task, but we did include all the relevant pcap files located in the “pcap files” folder in the project.

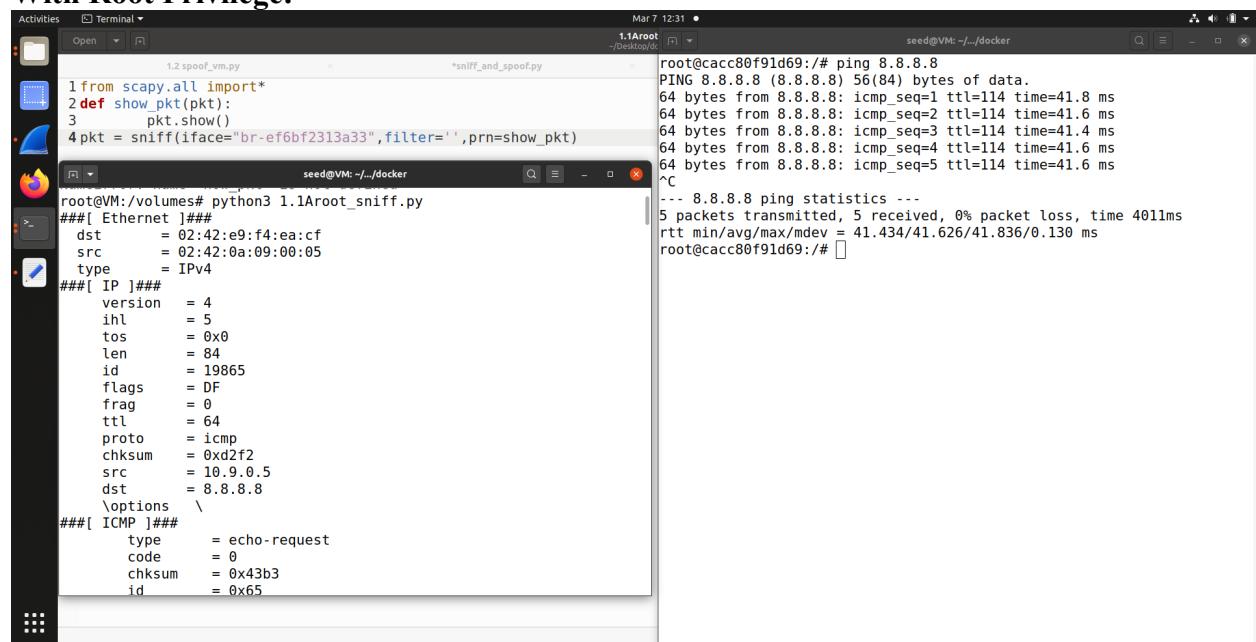
Same thing regarding the code for all the tasks, including the Python and C files.

Lab Task Set 1: Using Scapy to Sniff and Spoof Packets:

Task 1.1A:

Using Scapy, we build a short python program that sniffs any packet and prints to the terminal all the corresponding information (including Ethernet, IP and RAW information of the packet). We ran the code in two methods, with root privilege and without. Below are screenshots of both methods and the outcome:

With Root Privilege:



The screenshot shows a Linux desktop environment with a terminal window titled "1.1Root" running as root. The terminal displays the output of a ping command to 8.8.8.8 and the source code of a Python script named "1.1Aroot_sniff.py".

```
root@acc80f91d69:/# 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=114 time=41.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=41.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=114 time=41.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=114 time=41.6 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=114 time=41.6 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4011ms
rtt min/avg/max/mdev = 41.434/41.626/41.836/0.130 ms
root@acc80f91d69:/#
```

```
1.2 spoof_vm.py          *sniff_and_spoof.py
1 from scapy.all import*
2 def show_pkt(pkt):
3     pkt.show()
4 pkt = sniff(iface="br-ef6bf2313a33", filter='', prn=show_pkt)

root@VM:/volumes# python3 1.1Aroot_sniff.py
###[ Ethernet ]###
dst      = 02:42:e9:f4:ea:cf
src      = 02:42:0a:09:00:05
type     = IPv4
###[ IP ]###
version   = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 19865
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xd2f2
src      = 10.9.0.5
dst      = 8.8.8.8
'options  \
###[ ICMP ]###
    type    = echo-request
    code    = 0
    chksum = 0x43b3
    id     = 0x65
```

Without Root Privilege:



The screenshot shows a terminal window titled "Terminal" with the command "su seed" run as root. The user then attempts to execute "python3 1.1Aroot_sniff.py". The terminal displays a detailed stack trace of the error, which occurs at line 906 of the "sniff" module. The error message is "PermissionError: [Errno 1] Operation not permitted". The terminal window is part of a desktop environment with a dock containing icons for various applications like a file manager, terminal, browser, and file editor.

```
root@VM:/volumes# su seed
seed@VM:/volumes$ python3 1.1Aroot_sniff.py
Traceback (most recent call last):
  File "1.1Aroot_sniff.py", line 4, in <module>
    pkt = sniff(iface="br-ef6bf2313a33",filter='',prn=show_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$
```

As we can see, when running the program without root privilege we get an error, and the code is not able to sniff for packets. This is because the Scapy library and its functions opens a raw socket which needs root privilege. Opening raw socket has to get root privilege because it allows the changing the modification of packet info/data.

Task 1.1B:

In this task, we used our current python program and modified its sniff function and used its BPF filter parameter. The objectives of this task were to filter out three different things:

For each one, provided is a screenshot.

ICMP Packets only:

The screenshot shows a terminal window with two panes. The left pane displays the source code of a Python script named `sniff_icmp_1.1B.py`. The right pane shows the output of the command `ping www.ariel.ac.il`, which is capturing ICMP echo-request and echo-reply packets. Arrows point from the highlighted `proto = 'icmp'` lines in the code to the corresponding ICMP fields in the captured packets.

```
root@VM:/volumes# python3 sniff_icmp_1.1B.py
###[ Ethernet ]###
dst      = 02:42:e9:f4:ea:cf
src      = 02:42:0a:09:00:05
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len     = 52510
id       = DF
flags    = 0
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xcae2
src      = 10.9.0.5
dst      = 34.96.118.58
options   =
###[ ICMP ]###
type     = echo-request
code    = 0
chksum  = 0x9701
id       = 0x67
seq     = 0x1
###[ Raw ]###
load    = '\xcb\x0e\x00\x00\x00\x00\x00\x88t\x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !#$%`()*)+-./01234567'
###[ Ethernet ]###
dst      = 02:42:0a:09:00:05
src      = 02:42:e9:f4:ea:cf
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x20
len     = 84
id       = 1941
flags    = 
frag     = 0
ttl      = 114
proto    = icmp
chksum   = 0x9e4c
src      = 34.96.118.58
dst      = 10.9.0.5
options   =
###[ ICMP ]###
type     = echo-reply
code    = 0

```

```
root@VM:/volumes# ping www.ariel.ac.il
PING www.ariel.ac.il (34.96.118.58) 56(84) bytes of data.
64 bytes from 58.118.96.34.bc.googleusercontent.com (34.96.118.58): icmp_seq=1 ttl=114 time=41.6 ms
64 bytes from 58.118.96.34.bc.googleusercontent.com (34.96.118.58): icmp_seq=2 ttl=114 time=41.5 ms
64 bytes from 58.118.96.34.bc.googleusercontent.com (34.96.118.58): icmp_seq=3 ttl=114 time=41.6 ms
^C
--- www.ariel.ac.il ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 41.526/41.552/41.575/0.020 ms
```

File tabs at the bottom show: `sniff_and_spoof.py`, `sniff_and_spoof_arp.py`, `1.1Root_sniff.py`, and `sniff_icmp_1.1B.py`.

Capturing any TCP packets that come from a particular IP and with a destination port 23:
(2 photos provided, one showing the outcome of the execution, the second showing the code)

```
Feb 25 08:22 • seed@VM: ~/codes
[02/25/21]seed@VM:~$ telnet 10.0.2.15
Trying 10.0.2.15...
Connected to 10.0.2.15.
Escape character is '^'.
Ubuntu 20.04.1 LTS
VM 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

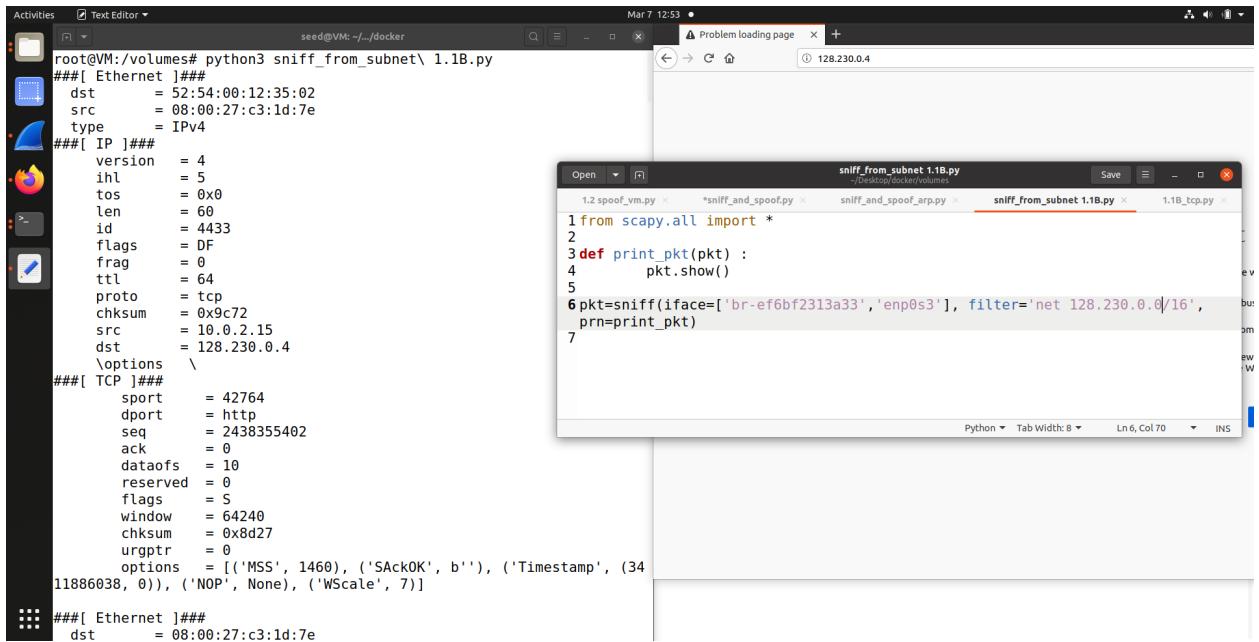
64 updates can be installed immediately.
64 of these updates are security updates.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Thu Feb 25 08:19:43 EST 2021 from VM on pts/6
[02/25/21]seed@VM:~$ 
```

```
###[ Ethernet ]##
dst      = 00:00:00:00:00:00
src      = 00:00:00:00:00:00
type     = IPv4
###[ IP ]##
version  = 4
ihl      = 5
tos      = 0x10
len      = 52
id       = 38448
flags    = DF
frag     = 0
ttl      = 64
proto    = tcp
chksum   = 0x8c66
src      = 10.0.2.15
dst      = 10.0.2.15
options   \
###[ TCP ]##
sport    = 35094
dport    = telnet
seq      = 2060273852
ack      = 2091461994
dataofs  = 8
reserved = 0
flags    = A
window   = 508
chksum   = 0x1844
urgptr   = 0
options   = [ ('NOP', None), ('NOP', None), ('Timestamp', (3325091016, 3325091016)) ]
###[ Ethernet ]##
dst      = 00:00:00:00:00:00
```

```
Feb 25 08:24 • sniff.py
~/Desktop/project/codes
from scapy.all import *
def print_pkt(pkt) :
    pkt.show()
pkt=sniff(iface='lo', filter='tcp and dst port 23 and src host 10.0.2.15', prn=print_pkt)
```

Capturing packets from/to a particular subnet:



The screenshot shows a terminal window with the command `root@VM:/volumes# python3 sniff_from_subnet\ 1.1B.py` running. The output displays detailed information about captured network packets, including fields like dst, src, type, IP version, ihl, tos, len, id, flags, frag, ttl, proto, checksum, src, dst, and options for both Ethernet and TCP layers.

```
root@VM:/volumes# python3 sniff_from_subnet\ 1.1B.py
###[ Ethernet ]###
    dst      = 52:54:00:12:35:02
    src      = 08:00:27:c3:1d:7e
    type     = IPv4
###[ IP ]###
    version   = 4
    ihl       = 5
    tos       = 0x0
    len       = 60
    id        = 4433
    flags     = DF
    frag      = 0
    ttl       = 64
    proto     = tcp
    checksum  = 0x9c72
    src       = 10.0.2.15
    dst       = 128.230.0.4
    \options  \
###[ TCP ]###
    sport     = 42764
    dport     = http
    seq       = 2438355402
    ack       = 0
    dataofs   = 10
    reserved  = 0
    flags     = S
    window    = 64240
    checksum  = 0x8d27
    urgptr   = 0
    options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (3411886038, 0)), ('NOP', None), ('WScale', 7)]
###[ Ethernet ]###
    dst      = 08:00:27:c3:1d:7e
```

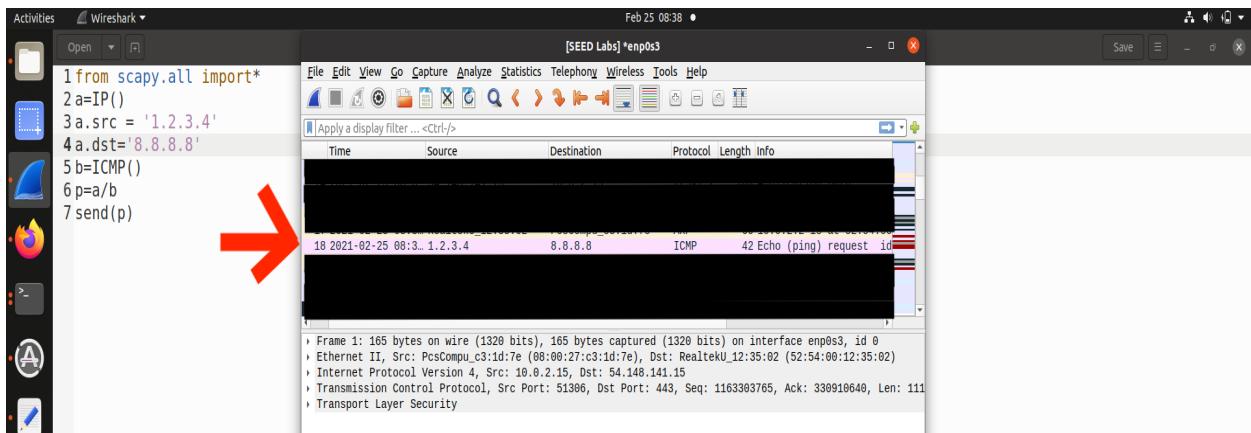
Below the terminal, a code editor window titled "sniff_from_subnet 1.1B.py" is open, showing the corresponding Python script. The script uses the scapy library to sniff on specific interfaces (br-ef6bf2313a33 and enp0s3) and filter for traffic on the 128.230.0.0/16 subnet. It defines a function `print_pkt` to handle each captured packet.

```
from scapy.all import *
def print_pkt(pkt) :
    pkt.show()
pkt=sniff(iface=['br-ef6bf2313a33','enp0s3'], filter='net 128.230.0.0/16', prn=print_pkt)
```

Task 1.2:

Using Scapy, we are able to spoof packets and set the fields of the IP packets to arbitrary values. For this task, using Wireshark we were able to see that we successfully spoofed the “IP Source” of the packet and change it so that the Wireshark tool would see that the packet was sent from a different IP address (as requested).

In the screenshot below we can see that Wireshark was able to sniff the ICMP packet sent to Google’s Public DNS (8.8.8.8), but instead of the source IP address being our IP, it read as if the packet was sent from this IP address: 1.2.3.4



Task 1.3:

In this task, to find the distance between our VM and the selected destination (in terms of routers) similar to the traceroute tool, we used Scapy and its TTL field to find how many routers were passed in order to reach the final IP destination. To do this automatically, and not have to change the TTL field each time and increment it by 1, we used a for loop in a particular range and used Wireshark to see which TTL value was exceeded each time and where the ping was successfully transferred to the desired destination.

In the screenshot below, we can see the code and the Wireshark program when ran executed our program, and we can see where each ICMP packet failed to receive a reply, and finally where it did receive a reply and there, we can see which TTL value it belongs to.

```
1 from scapy.all import*
2
3 a = IP()
4 a.dst = '34.96.118.58'
5 for num in range (1,15):
6     a.ttl=num
7     b=ICMP()
8     send(a/b)
9     a.show()
10
11
```

Wireshark Screenshot Description: The screenshot shows the Wireshark interface with a Python script in the left pane and a packet list in the right pane. The script performs a ping sweep from TTL 1 to 15. The packet list shows 469 total packets, with 27 displayed. The first 15 packets (TTL 1-15) are highlighted with a red box. The last packet (TTL 14) is highlighted with a green box. A blue arrow points from the green box back to the corresponding request packet in the list. The status bar at the bottom indicates 'Packets: 469 - Displayed: 27 (5.8%)'.

No.	Time	Source	Destination	Protocol	Length	Info
49	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=1 (no response found!)
50	2021-02-25 09:0: 10.0.2.2	10.0.2.15	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=2 (no response found!)
51	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=3 (no response found!)
52	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=4 (no response found!)
53	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=5 (no response found!)
54	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=6 (no response found!)
55	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=7 (no response found!)
56	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=8 (no response found!)
57	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=9 (no response found!)
58	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=10 (no response found!)
59	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=11 (no response found!)
60	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=12 (no response found!)
61	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=13 (no response found!)
62	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=14 (no response found!)
63	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=15 (no response found!)
64	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=16 (no response found!)
65	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=17 (no response found!)
66	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=18 (no response found!)
67	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=19 (no response found!)
68	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=20 (no response found!)
69	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=21 (no response found!)
70	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=22 (no response found!)
71	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=23 (no response found!)
72	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=24 (no response found!)
73	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=25 (no response found!)
74	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=26 (no response found!)
75	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=27 (no response found!)
76	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=28 (no response found!)
77	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=29 (no response found!)
78	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=30 (no response found!)
79	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=31 (no response found!)
80	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=32 (no response found!)
81	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=33 (no response found!)
82	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=34 (no response found!)
83	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=35 (no response found!)
84	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=36 (no response found!)
85	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=37 (no response found!)
86	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=38 (no response found!)
87	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=39 (no response found!)
88	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=40 (no response found!)
89	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=41 (no response found!)
90	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=42 (no response found!)
91	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=43 (no response found!)
92	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=44 (no response found!)
93	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=45 (no response found!)
94	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=46 (no response found!)
95	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=47 (no response found!)
96	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=48 (no response found!)
97	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=49 (no response found!)
98	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=50 (no response found!)
99	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=51 (no response found!)
100	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=52 (no response found!)
101	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=53 (no response found!)
102	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=54 (no response found!)
103	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=55 (no response found!)
104	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=56 (no response found!)
105	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=57 (no response found!)
106	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=58 (no response found!)
107	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=59 (no response found!)
108	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=60 (no response found!)
109	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=61 (no response found!)
110	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=62 (no response found!)
111	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=63 (no response found!)
112	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=64 (no response found!)
113	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=65 (no response found!)
114	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=66 (no response found!)
115	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=67 (no response found!)
116	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=68 (no response found!)
117	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=69 (no response found!)
118	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=70 (no response found!)
119	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=71 (no response found!)
120	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=72 (no response found!)
121	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=73 (no response found!)
122	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=74 (no response found!)
123	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=75 (no response found!)
124	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=76 (no response found!)
125	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=77 (no response found!)
126	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=78 (no response found!)
127	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=79 (no response found!)
128	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=80 (no response found!)
129	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=81 (no response found!)
130	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=82 (no response found!)
131	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=83 (no response found!)
132	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=84 (no response found!)
133	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=85 (no response found!)
134	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=86 (no response found!)
135	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=87 (no response found!)
136	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=88 (no response found!)
137	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=89 (no response found!)
138	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=90 (no response found!)
139	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=91 (no response found!)
140	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=92 (no response found!)
141	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=93 (no response found!)
142	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=94 (no response found!)
143	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=95 (no response found!)
144	2021-02-25 09:0: 10.0.2.15	34.96.118.58	ICMP	42	Echo (ping) request id=0x0000, seq=0/0	ttl=96 (no response found!)
145	2021-02-25 09:0: 10.0.2.15	34.96.118.				

Task 1.4:

In this task, we were able to sniff out pings that generated ICMP echo requests to a certain IP (whether that IP address to a certain machine was alive or not) and then returned a spoof reply to the same machine that sent out the ping.

Therefore, no matter if machine A sent a ping to a non-existing host or even to a machine that is not alive, regardless, machine A received a reply.

Below are 3 screenshots representing the three different IP's we pinged on our host machine, and running the program on our attacker container we were able to sniff and spoof accordingly.

Above each screenshot is written the IP that the host machine pinged.

1.2.3.4: A non-existing host on the Internet

10.9.0.99: A non-existing host on the LAN

8.8.8.8: An existing host on the Internet

Lab Task Set 2: Writing Programs to Sniff and Spoof Packets:

Task 2.1A:

We built a sniffer program in C that sniffs packets and prints out information of the captured packet, including the source and destination IP addresses, protocol type, source port... etc. We wrote this program using the pcap library, and through that the sniffer became a simple sequence of procedures. In addition, all details of the packet capturing are handled by the pcap library.

Below is a screenshot as requested of the execution of the sniffer program:

Question 1:

There are a sequence of library calls that are essential for sniffer programs. The fundamental function calls that are used for such sniffer programs include a couple steps:

- Setting up the device and determining which ethernet interface that the program will utilize and to start capturing with.
- After setting up the device, occurs the initialization of the pcap to create a session and tells it to sniff on a particular device. Typically, there is only one session per device to be sniffed.
- The traffic filtering rules occurs, and this ensures that the type of traffic sniffed on an interface is the type that we were looking for (filtering). It is also possible to use a blank filter but, in that case, it will be sniffing all packets and will be analyzing all the fields.
- Now the actual execution of the sniff occurs.
- The session ends/is terminated, and the program stops sniffing.

Question 2:

Similar to what I explained in task 1.1A, when running the scappy sniffer program without root privilege. Running the sniffer program that uses pcap needs root access because it wants to access network interfaces and it is impossible to do so without having root access in Linux.

Otherwise the program will fail at the “pcap_open_live” stage. Sniffer programs need raw sockets that allow the direct sending of packets by the applications bypassing all applications in network software of the operating system. And in order to create raw sockets, we need to be a root user because we can't discover the NIC (network interface card) until we are root.

Question 3:

As requested, to demonstrate this for example, let's consider we have two VM's, with the sniffer program running on machine the first machine (1), and the second machine tries to ping a random IP address. Now in non-promiscuous mode I found that no packet is sniffed at machine 1 other than those with the destinations address of the first machine.

In other words, we can simply say that promiscuous mode is one in which all the packets are sent to a computer or sniffed by a sniffer and not only those which are addressed to it.

Whereas when we are in a non-promiscuous mode, only those packets are sent to the computer or sniffed by a sniffer which are specifically addressed to it.

We do this using the “pcap_open_live” function and enter “1” as the third parameter of the function to turn on promiscuous mode, and enter “0”, to turn it off. Therefore, we will listen to the traffic that passes only in the specific virtual machine on which we ran the program, and not on all the network traffic.

Task 2.1B:

Sniffer programs using the pcap library, we can easily filter out specific data we want from the packets using pcap filters. The filter expression consists of one or more primitives. Primitives usually consist of an ID (name or number) preceded by one or more qualifiers. There are three different kinds of qualifiers (type, dir, and proto).

Below are two screenshots of the execution of the sniffer program consisting of two different filters. The first is to capture ICMP packets between two specific hosts, and the second is filtered to capture the TCP packets with a destination port number in the range from 10 to 100.

Capturing ICMP packets between two specific hosts:

The screenshot shows a terminal window with the following content:

```
root@VM:/volumes# gcc -o test 2.1B_sniff_icmp.c -lpcap
root@VM:/volumes# ./test
From: 10.9.0.5
To: 8.8.8.8
Protocol: ICMP
  From: 8.8.8.8
  To: 10.9.0.5
Protocol: ICMP
  From: 10.9.0.5
  To: 8.8.8.8
Protocol: ICMP
  From: 8.8.8.8
  To: 10.9.0.5
Protocol: ICMP
  From: 10.9.0.5
  To: 8.8.8.8
Protocol: ICMP
  From: 8.8.8.8
  To: 10.9.0.5
Protocol: ICMP
  From: 10.9.0.5
  To: 8.8.8.8
Protocol: ICMP
  From: 8.8.8.8
  To: 10.9.0.5
Protocol: ICMP
```

After the command is run, the terminal shows the output of the sniffer program. It captures ICMP packets between the two specified hosts (10.9.0.5 and 8.8.8.8). The captured packets are shown in the terminal window, with arrows pointing to the 'Protocol: ICMP' line and the 'From' and 'To' fields of the captured packets.

```
root@acc80f91d69:# 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=114 time=41.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=42.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=114 time=44.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=114 time=42.9 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3025ms
rtt min/avg/max/mdev = 41.775/42.735/43.959/0.816 ms
root@acc80f91d69:#
```

```
root@acc80f91d69:# ping www.ariel.ac.il
PING www.ariel.ac.il (34.96.118.58) 56(84) bytes of data.
64 bytes from 58.118.96.34.bc.googleusercontent.com (34.96.118.58): icmp_seq=1 ttl=114 time=42.3 ms
64 bytes from 58.118.96.34.bc.googleusercontent.com (34.96.118.58): icmp_seq=2 ttl=114 time=43.3 ms
64 bytes from 58.118.96.34.bc.googleusercontent.com (34.96.118.58): icmp_seq=3 ttl=114 time=43.4 ms
64 bytes from 58.118.96.34.bc.googleusercontent.com (34.96.118.58): icmp_seq=4 ttl=114 time=43.1 ms
^C
--- www.ariel.ac.il ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 42.332/43.029/43.350/0.411 ms
root@acc80f91d69:#
```

Capturing the TCP packets with a destination port number in the range from 10 to 100:

The screenshot shows a Wireshark interface on the left displaying several TCP connections between 10.0.2.15 and 34.107.221.82. The connections are highlighted with red circles around the 'Protocol' field and the destination port '80'. To the right, a terminal window displays a C program named 'sniff.c'. The code uses the pcap library to open a live session on 'enp0s3', compile a BPF filter for TCP port range 10-100, and then capture packets until EOF. The line containing the BPF filter is circled in red.

```
From: 10.0.2.15  
To: 34.107.221.82  
Source Port: 44478  
Destination Port: 80  
Protocol: TCP  
Payload (20 bytes):  
.....  
From: 10.0.2.15  
To: 34.107.221.82  
Source Port: 44478  
Destination Port: 80  
Protocol: TCP  
From: 10.0.2.15  
To: 34.107.221.82  
Source Port: 44478  
Destination Port: 80  
Protocol: TCP  
Payload (296 bytes):  
.....  
From: 10.0.2.15  
To: 34.107.221.82  
Source Port: 44478  
Destination Port: 80  
Protocol: TCP  
From: 10.0.2.15  
To: 34.107.221.82  
Source Port: 44486  
Destination Port: 80  
Protocol: TCP  
Payload (20 bytes):  
.....  
From: 10.0.2.15  
To: 34.107.221.82  
Source Port: 44486
```

```
84         data+=12;  
85         printf(".....%c\n", *data);  
86         //for(int i = 0; i < size_data; i++) {  
87         //    if (isprint(*data)) printf("%c", *data);  
88         //    else printf(".");  
89         //    data++;  
90     }  
91 }  
92  
93 }  
94 }  
95  
96 int main() {  
97     pcap_t *handle;  
98     char errbuf[PCAP_ERRBUF_SIZE];  
99     struct bpf_program fp;  
100    char filter_exp[] = "proto TCP and dst portrange 10-100";  
101    bpf_u_int32 net;  
102  
103    // step 1: open live pcap session on NIC with interface name  
104    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);  
105  
106    // step 2: compile filter_exp into BPF pseudo-code  
107    pcap_compile(handle, &fp, filter_exp, 0, net);  
108    pcap_setfilter(handle, &fp);  
109  
110    // step 3: capture packets  
111    pcap_loop(handle, -1, got_packet, NULL);  
112  
113  
114    pcap_close(handle); // close the handle
```

Task 2.1C:

In this program we used our sniffer program to capture the password when using telnet on the network we are monitoring. Given that we ran this program on the SEED VM, when using telnet we logged in to the seed account and entered the password in the terminal and at the same time ran our sniffer program to sniff out the password.

In the screenshot below we can see that we successfully were able to sniff out the password of the host machine. The password was: “dees”.

The screenshot shows a Linux desktop environment with a terminal window and a packet sniffer application. The terminal window on the right displays a telnet session where the user has entered their password. The packet sniffer application on the left shows several captured TCP packets. Four specific bytes from the second packet are highlighted with red circles and labeled 'd', 'e', 'e', and 's' respectively, corresponding to the password 'dees'. The terminal output includes the telnet session, system updates information, and a login timestamp.

```
[02/25/21]seed@VM:~$ telnet 10.0.2.15
Trying 10.0.2.15...
Connected to 10.0.2.15.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
VM login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

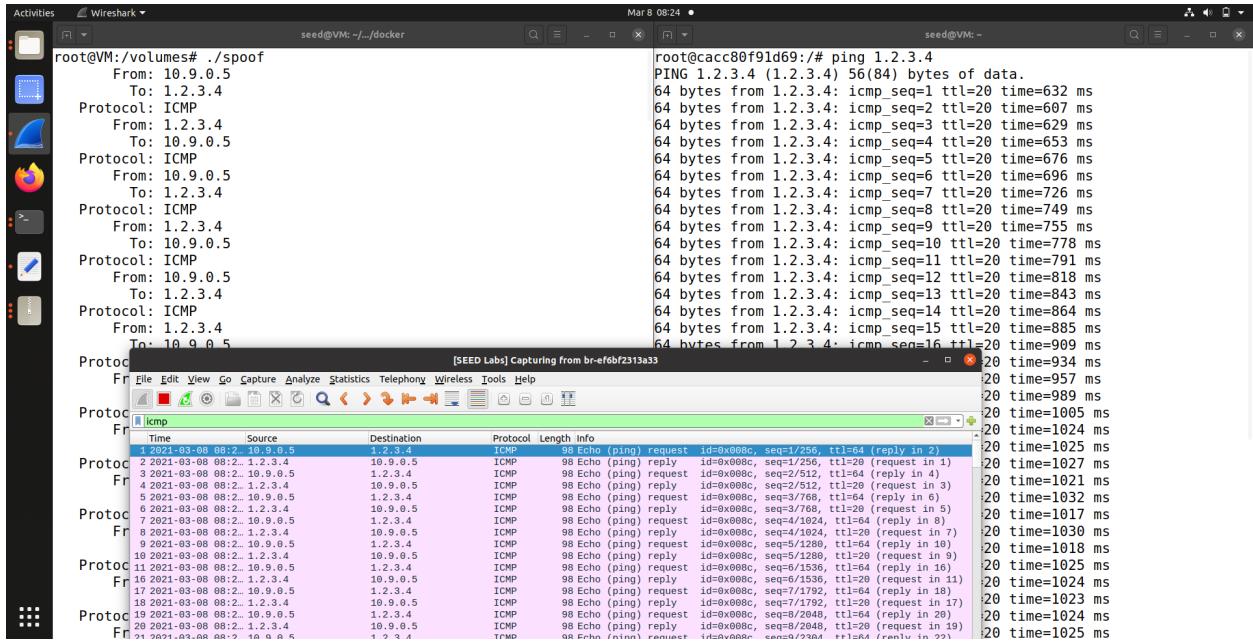
 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

64 updates can be installed immediately.
64 of these updates are security updates.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Thu Feb 25 09:46:53 EST 2021 from VM on pts/4
[02/25/21]seed@VM:~$
```

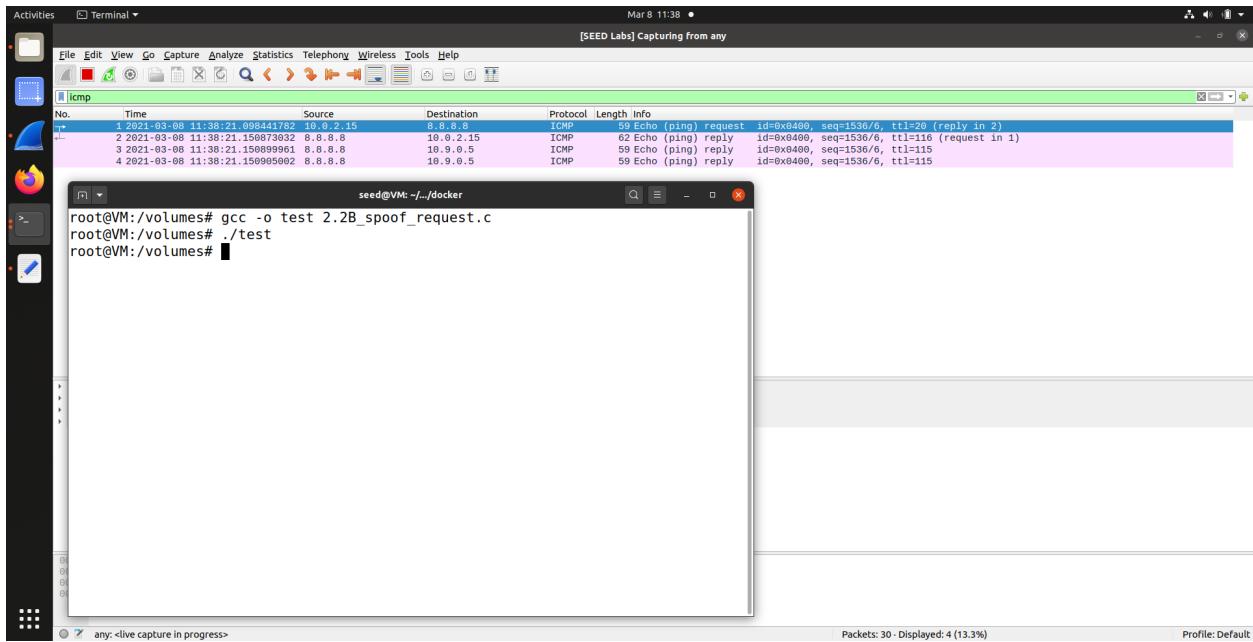
Task 2.2A:

Below is a screenshot of our packet spoofing program in action. In addition, added to the project located in the Wireshark pcap files folder, is the Wireshark packet trace that proves that the packet was successfully spoofed.



Task 2.2B:

Below is a screenshot of our packet spoofing program in action that spoofed an ICMP echo request on behalf of another machine. In addition, added to the project located in the Wireshark pcap files folder, is the Wireshark packet trace that proves that the packet was successfully spoofed, and we can see the echo reply coming back from the remote machine.



Question 4:

Yes, the length of the IP header can be set to arbitrary values, but it won't change, and it will return to its original size. The length of the IP header can be set to arbitrary values regardless of how big the actual packet is. This is because when we send a packet larger than its actual size, the additional data in the payload is a chunk of zeros. And that is in fact what we see on the captured packets when we inspect them using a program such as Wireshark. The payload will manually increase until completing the total length of the packet.

Question 5:

No, for IP headers the computer/operating system automatically does this, or rather fills it in by default.

Question 6:

In order to run programs that use raw sockets, similarly to what we explained in question 2 (for the sniffer program), we need root privilege because we can spoof packets, which may interfere with inbound traffic. In other words, opening a raw socket allows to read anything that is received in a given interface, so, basically, we can read any packet that is directed to any application - even if that application is owned by another user. That basically means that the user with this capability is able to read any and all communications of all users on the machine.

Task 2.3:

Below is a screenshot of our sniffing and spoofing program in action that sniffs ICMP echoes of another machine and spoofs out a reply regardless of the target IP address, and regardless of whether the machine is alive or not.

Therefore, the ping program will always receive a reply.

Here we pinged 1.2.3.4 – a non-existing IP on the Internet and received a reply.

