

Final project - network communication

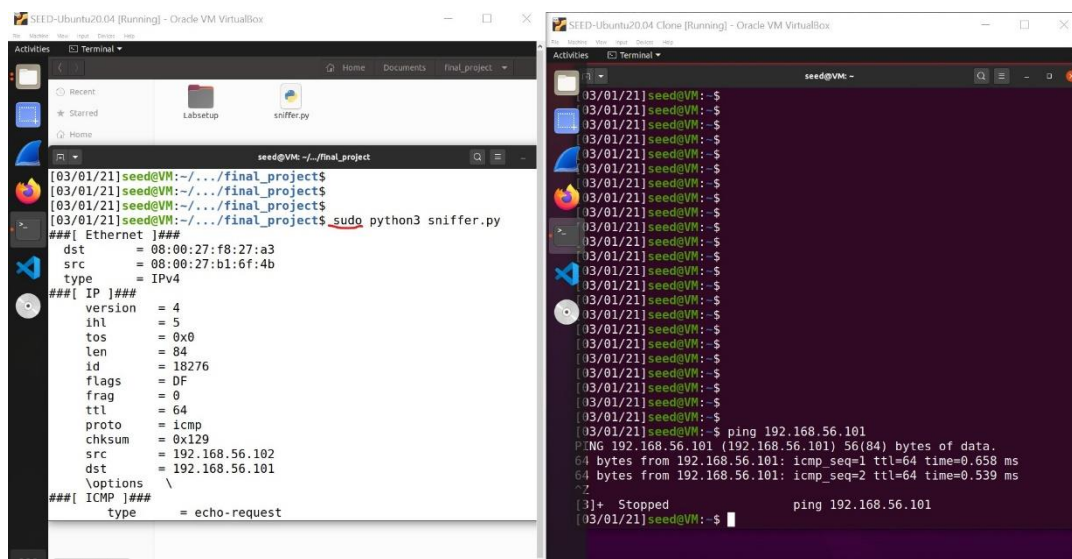
Moshe crespin

Id- 315344515

1.1A-

We can run the sniffer.py program only with root privilege because of that the sniffing action uses Raw socket, and we can use this raw socket only with root privilege O.W we will get a permission error.

screenshot that demonstrate sniffing with root privilege:



screenshot that demonstrate sniffing without root privilege – we can see that we got a permission error:

```
[03/01/21]seed@VM:~/../final_project$ python3 sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 5, in <module>
    pkt=sniff(iface='enp0s8' , 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", lin
e 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.h
tons(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
```

1.1B-

Capture only the ICMP packet:

I have set The filter to get only icmp packet (filter="icmp"):

In this pic we can infer that my prog got only the icmp packets, because in wireshrak there are also other packets(with other protocols).

The screenshot shows a terminal window on the left and a Wireshark packet capture window on the right. The terminal window displays the output of a ping command to 8.8.8.8, showing a successful response with 56 bytes of data and a time of 70.6 ms. The Wireshark window shows a list of captured packets, with the ICMP packet (ping) highlighted in red. The packet details pane on the right shows the structure of the ICMP Echo (ping) request, including the type (icmp), code (0), and sequence number (1).

```
[03/01/21]seed@VM:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data:
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=70.6 ms
^C
[10]+ Stopped ping 8.8.8.8
[03/01/21]seed@VM:~$
```

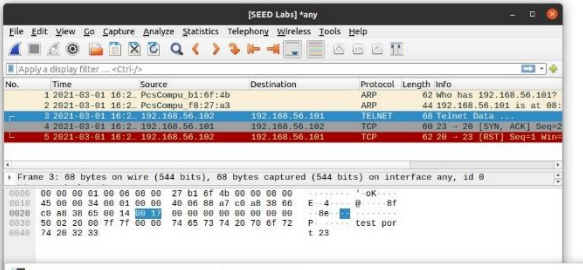
No.	Time	Source	Destination	Protocol	Length	Info
539	2021-03-01 12:11:10.0.2.15	23.221.156.151	10.0.2.15	TCP	56	37806 → 443 [ACK]
540	2021-03-01 12:11:10.0.2.15	13.226.0.223	10.0.2.15	TCP	56	[TCP Dup ACK 77#1]
541	2021-03-01 12:11:13.226.0.223	10.0.2.15	10.0.2.15	TCP	62	[TCP Dup ACK 198#1]
542	2021-03-01 12:11:10.0.2.15	56.7.82.60	10.0.2.15	TCP	76	[TCP Retransmission]
543	2021-03-01 12:11:10.0.2.15	8.8.8.8	10.0.2.15	ICMP	100	Echo (ping) request
544	2021-03-01 12:11:10.0.2.15	10.0.2.15	8.8.8.8	ICMP	100	Echo (ping) reply
545	2021-03-01 12:11:13.197.21.290	10.0.2.15	10.0.2.15	TCP	62	443 → 4424 [RST, Seq=19721290]
546	2021-03-01 12:11:10.0.2.15	45.192.152.90	10.0.2.15	TCP	56	[TCP Dup ACK 198#1]
547	2021-03-01 12:11:02.102.152.90	10.0.2.15	10.0.2.15	TCP	62	[TCP Dup ACK 198#1]
548	2021-03-01 12:11:10.0.2.15	23.221.156.151	10.0.2.15	TLSv1.2	262	Application Data
549	2021-03-01 12:11:10.0.2.15	10.0.2.15	23.221.156.151	TCP	62	443 → 37806 [ACK]
550	2021-03-01 12:11:23.221.156.151	10.0.2.15	10.0.2.15	TLSv1.2	406	Application Data
551	2021-03-01 12:11:23.221.156.151	10.0.2.15	10.0.2.15	TLSv1.2	406	Application Data

```
##[ Ethernet ]##
dst      = 52:54:00:12:35:02
src      = 08:00:27:7a:77:e7
type     = IPv4
##[ IP ]##
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 13542
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xe9a4
src      = 10.0.2.15
dst      = 8.8.8.8
\options \
##[ ICMP ]##
type     = echo-request
code     = 0
chksum   = 0x407d
id       = 0xb
seq      = 0x1
```

Capture any TCP packet that comes from a particular IP and with a destination port number 23:

I have two VM so with the first VM("the particular ip") I sent packet using python prog with dest port 23 to my second VM-we can see that my sniffer sniffed only the right packet.

```
[03/01/21]seed@VM:~/../final_projects
[03/01/21]seed@VM:~/../final_projects$ sudo python3 q1.1b_tcp.py
[+]sniffing TCP packets from 192.168.56.102 who sent to port 23...
dsfs
###[ Ethernet ]###
dst      = 08:00:27:f8:27:a3
src      = 08:00:27:b1:6f:4b
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 52
id       = 1
flags    =
frag     = 0
ttl      = 64
proto    = tcp
chksum   = 0x88a7
src      = 192.168.56.102
dst      = 192.168.56.101
\options
###[ TCP ]###
sport    = ftp_data
dport    = telnet
seq      = 0
ack      = 0
dataoffs = 5
reserved = 0
flags    = S
window   = 8192
chksum   = 0x7f7f
urgptr   = 0
options  = []
###[ Raw ]###
load     = 'test port 23'
```

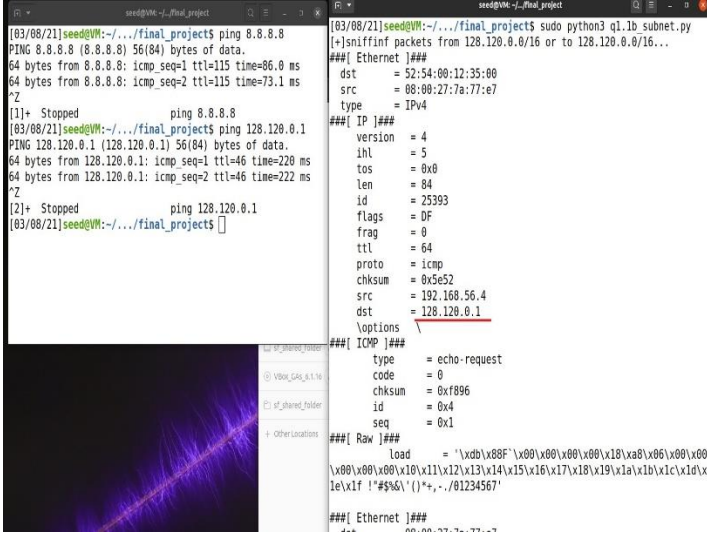


```
[03/01/21]seed@VM:~$
[03/01/21]seed@VM:~$
[03/01/21]seed@VM:~$ sudo python3
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> ip=IP(src="192.168.56.102", dst="192.168.56.101")
>>> _tcp=TCP(dport=23)
>>> data="test port 23"
>>> packet=ip/_tcp/data
>>> send(packet)

Sent 1 packets.
>>>
```

Capture packets comes from or to 128.120.0.0/16:

First I sent ping to google, we can see that my sniffer didn't got this pkt, then I sent ping to 128.120.0.1 who belongs to our subnet, we can see that we got the "to" message (echo-request) and the "from" message(echo-reply).



```

[03/08/21]seed@VM:~/final_project$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data:
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=86.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=73.1 ms
^C
[1]- Stopped                  ping 8.8.8.8
[03/08/21]seed@VM:~/final_project$ ping 128.120.0.1
PING 128.120.0.1 (128.120.0.1) 56(84) bytes of data:
64 bytes from 128.120.0.1: icmp_seq=1 ttl=46 time=220 ms
64 bytes from 128.120.0.1: icmp_seq=2 ttl=46 time=222 ms
^C
[2]- Stopped                  ping 128.120.0.1
[03/08/21]seed@VM:~/final_project$

```

```

###[ Ethernet ]###
dst      = 52:54:00:12:35:00
src      = 08:00:27:7a:77:e7
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 25393
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0x5e52
src      = 192.168.56.4
dst      = 128.120.0.1
\options \
###[ ICMP ]###
type     = echo-request
code     = 0
chksum   = 0xf896
id       = 0x4
seq      = 0x1
###[ Raw ]###
load     = '\xdb\x8f'\x00\x00\x00\x00\x18\xa8\x06\x00\x00
\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x
1e\x1f !"#%&'()*+,-./01234567'
###[ Ethernet ]###
dst      = 08:00:27:7a:77:e7
src      = 52:54:00:12:35:00
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x20
len      = 84
id       = 30163
flags    =
frag     = 0
ttl      = 46
proto    = icmp
chksum   = 0x9d90
src      = 128.120.0.1
dst      = 192.168.56.4
\options \
###[ ICMP ]###
type     = echo-reply
code     = 0
chksum   = 0x97
id       = 0x4
seq      = 0x1
###[ Raw ]###
load     = '\xdb\x8f'\x00\x00\x00\x00\x18\xa8\x06\x00\x00
\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x
1e\x1f !"#%&'()*+,-./01234567'
###[ Ethernet ]###
dst      = 52:54:00:12:35:00
src      = 08:00:27:7a:77:e7

```

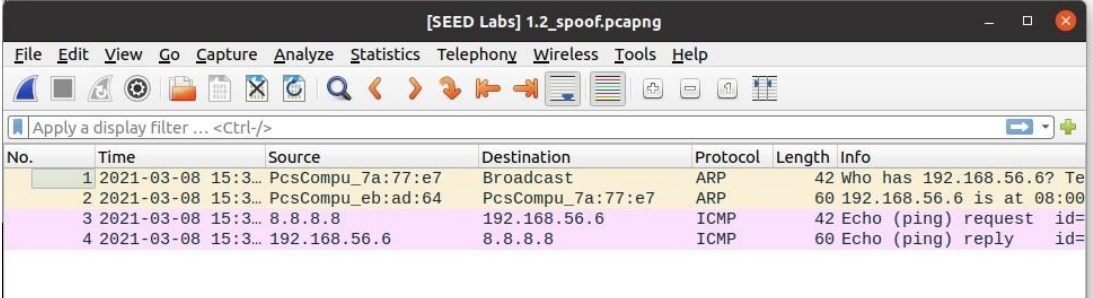
Task 1.2: Spoofing ICMP Packet:

Here I spoofed an ICMP echo request from google and sent this pkt from my first VM to my second VM, we can see in Wireshark report that the second VM actually sent an echo-reply to google.

```

seed@VM: ~/.../final_project
[03/08/21]seed@VM:~/.../final_project$ sudo python3 1.2.py
.
Sent 1 packets.
[03/08/21]seed@VM:~/.../final_project$

```



The Wireshark window shows a packet capture titled "[SEED Labs] 1.2_spoof.pcapng". The packet list contains four entries:

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-03-08 15:3...	PcsCompu_7a:77:e7	Broadcast	ARP	42	Who has 192.168.56.6? Te
2	2021-03-08 15:3...	PcsCompu_eb:ad:64	PcsCompu_7a:77:e7	ARP	60	192.168.56.6 is at 08:00
3	2021-03-08 15:3...	8.8.8.8	192.168.56.6	ICMP	42	Echo (ping) request id=
4	2021-03-08 15:3...	192.168.56.6	8.8.8.8	ICMP	60	Echo (ping) reply id=

Task 1.3: Traceroute:

I wrote a python program that find out the length of the traceroute, I sent an icmp echo request with ttl who set at the start to 1, any time that we get response that not echo reply (time to live exceeded), we will had 1 to the old ttl and send the pkt again, when we will get an echo reply; then we got to our destination; and now we know the length of the route – how many time we got time to live exceeded. I have checked the traceroute to google.com, the length I got was 13:

```

seed@VM: ~/.../python
[03/08/21]seed@VM:~/.../python$ sudo python3 q1.3.py
1 - curr addr: 192.168.56.1
2 - curr addr: 192.168.1.1
3 - curr addr: 82.102.128.240
4 - Request timed out.
5 - Request timed out.
6 - curr addr: 82.102.132.150
7 - curr addr: 80.179.165.221
8 - curr addr: 195.66.236.125
9 - curr addr: 108.170.246.143
10 - curr addr: 216.239.58.3
11 - curr addr: 209.85.142.166
12 - curr addr: 209.85.142.97
13 - curr addr: 209.85.242.78
14 - curr addr: 108.170.251.129
15 - curr addr: 172.253.71.89
dest addr: 142.250.186.46 is in length of: 13
[03/08/21]seed@VM:~/.../python$

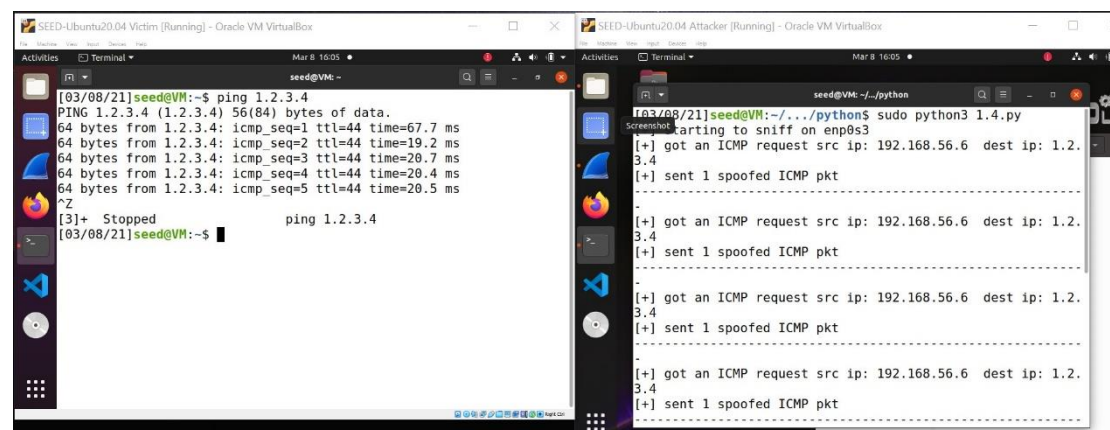
```


Task 1.4: Sniffing and-then Spoofing:

For this task I wrote a sniffer that has filter of ARP and ICMP protocol with a victim IP,. when my sniffer get an ICMP or ARP request then the sniffer sends these pkt to a function that decide whether this pkt is kind of ARP or kind of ICMP. if it is an ARP then this func sends it to an ARP spoofer func (arp reply) that sends a spoofed ARP reply, O.W if it is an ICMP request pkt then it send it to my ICMP spoofer that sends a spoofed ICMP reply.

ping 1.2.3.4 - a non-existing host on the Internet

We can see that the victim got reply, this reply came from the Attacker computer via my ICMP spoofer



ping 192.168.56.99 a non-existing host on the LAN:

first, the communication inside the local network is doing via the mac addresses. To understanding how it works, let's have an example- Let's say that my local network called moshe, so if a computer A who belongs to moshe wants to send a msg to device B that is also belongs to moshe.

A has the IP address of B but at the first time A does not have the MAC address of B. so if A wants the MAC address of B then A sends an ARP msg to all the devices that belongs to moshe, only B should answer to this ,sg by sending an ARP reply with his own mac address. So if I wants to tell A that B is alive although B is not exist, I should reply to the msg that A sent with an ARP reply that contains my mac address, now A think that B has my mac address so A sends me ICMP msg' and I response with a spoofed icmp reply.

The image shows two terminal windows from an Oracle VM VirtualBox. The left window, titled 'SEED-Ubuntu20.04 Victim [Running]', shows a user running a ping command to 192.168.56.99. The output shows three successful pings with TTL=44. The right window, titled 'SEED-Ubuntu20.04 Attacker [Running]', shows a user running a Python script '1.4.py'. The script's output shows it successfully sniffed an ARP request and sent a spoofed ARP packet to the victim.

```

[03/08/21]seed@VM:~$ ping 192.168.56.99
PING 192.168.56.99 (192.168.56.99) 56(84) bytes of data.
64 bytes from 192.168.56.99: icmp_seq=1 ttl=44 time=113 ms
64 bytes from 192.168.56.99: icmp_seq=2 ttl=44 time=15.9 ms
64 bytes from 192.168.56.99: icmp_seq=3 ttl=44 time=20.5 ms
^Z
[4]+  Stopped                  ping 192.168.56.99
[03/08/21]seed@VM:~$

[03/08/21]seed@VM:~/python$ sudo python3 1.4.py
[+] starting to sniff on enp0s3
[+] got an ARP request src ip: 192.168.56.6 dest ip: 192.168.56.99
[+] sent 1 spoofed ARP pkt
-----
[+] got an ICMP request src ip: 192.168.56.6 dest ip: 192.168.56.99
[+] sent 1 spoofed ICMP pkt
-----
[+] got an ICMP request src ip: 192.168.56.6 dest ip: 192.168.56.99
[+] sent 1 spoofed ICMP pkt
-----
[+] got an ICMP request src ip: 192.168.56.6 dest ip: 192.168.56.99
[+] sent 1 spoofed ICMP pkt
-----
^Z
[3]+  Stopped                  sudo python3 1.4.py
[03/08/21]seed@VM:~/python$

```

ping 8.8.8.8 - an existing host on the Internet:

we can see that the victim got a reply from the Attacker(my spoofed icmp msg has ttl that set to 44) and from the real server. We can see that the victim got my spoofed icmp reply before he got the icmp reply from the real server(now occurs all the time). DUP! means duplicate packet.

The image shows two terminal windows from an Oracle VM VirtualBox. The left window, titled 'SEED-Ubuntu20.04 Victim [Running]', shows a user running a ping command to 1.2.3.4. The output shows five successful pings with TTL=44. The right window, titled 'SEED-Ubuntu20.04 Attacker [Running]', shows a user running a Python script '1.4.py'. The script's output shows it successfully sniffed an ICMP request and sent a spoofed ICMP packet to the victim.

```

[03/08/21]seed@VM:~$ 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=44 time=67.7 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=44 time=19.2 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=44 time=20.7 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=44 time=20.4 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=44 time=20.5 ms
^Z
[3]+  Stopped                  ping 1.2.3.4
[03/08/21]seed@VM:~$

[03/08/21]seed@VM:~/python$ sudo python3 1.4.py
[+] starting to sniff on enp0s3
[+] got an ICMP request src ip: 192.168.56.6 dest ip: 1.2.3.4
[+] sent 1 spoofed ICMP pkt
-----
[+] got an ICMP request src ip: 192.168.56.6 dest ip: 1.2.3.4
[+] sent 1 spoofed ICMP pkt
-----
[+] got an ICMP request src ip: 192.168.56.6 dest ip: 1.2.3.4
[+] sent 1 spoofed ICMP pkt
-----
[+] got an ICMP request src ip: 192.168.56.6 dest ip: 1.2.3.4
[+] sent 1 spoofed ICMP pkt
-----
[+] got an ICMP request src ip: 192.168.56.6 dest ip: 1.2.3.4
[+] sent 1 spoofed ICMP pkt
-----
[+] got an ICMP request src ip: 192.168.56.6 dest ip: 1.2.3.4
[+] sent 1 spoofed ICMP pkt
-----
^Z
[3]+  Stopped                  sudo python3 1.4.py
[03/08/21]seed@VM:~/python$

```

Task 2.1A: Understanding How a Sniffer Works:

In this task I wrote an icmp sniffer, when it sniffs an icmp pkt, it prints src and dest.


```

[03/07/21]seed@VM:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=72.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=71.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=92.1 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=72.3 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=72.4 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=115 time=76.2 ms
^Z
[2]+  Stopped                  ping 8.8.8.8
[03/07/21]seed@VM:~$

[03/07/21]seed@VM:~/..C$ sudo ./2.1A
got a new ICMP meassege:
Source IP: 192.168.56.4 Destination IP: 8.8.8.8
got a new ICMP meassege:
Source IP: 8.8.8.8 Destination IP: 192.168.56.4
got a new ICMP meassege:
Source IP: 192.168.56.4 Destination IP: 8.8.8.8
got a new ICMP meassege:
Source IP: 8.8.8.8 Destination IP: 192.168.56.4
got a new ICMP meassege:
Source IP: 192.168.56.4 Destination IP: 8.8.8.8
got a new ICMP meassege:
Source IP: 8.8.8.8 Destination IP: 192.168.56.4
got a new ICMP meassege:
Source IP: 192.168.56.4 Destination IP: 8.8.8.8
got a new ICMP meassege:
Source IP: 8.8.8.8 Destination IP: 192.168.56.4
got a new ICMP meassege:
Source IP: 192.168.56.4 Destination IP: 8.8.8.8
^Z

```

Q1-

First we should understand that there are 3 steps that essential for sniffing,: initial configuration, filter and and start sniffing

Now let's look at the essential functions for these steps:

pcap_open_live: this function is responsible for the initial configuration step, meaning: responsible to which interface we want to sniff, and if we want to turn on/off the promiscuous mode, we also should specify the error buffer size, time out and snap length- how many bytes we want from each packet.

For the filter part there are two essential function:

pcap_compile: this function is responsible for doing the translation from the "human" language into the machine language.

pcap_set_filter: set the filter after the translation in the above function.

for the actual capturing step there is only 1 function:

pcap_loop: this function is responsible for the actual sniffing, also we can tell this function what we want to do with the captured packets, and how many packets we want to get from the sniffing (-1 for infinity)

Q2-

The sniffer uses raw_socket, we can open raw socket only with root privilege. So if we will not open the sniffer with root privilege then the software will fail

```
seed@VM: ~/.../C
[03/07/21] seed@VM:~/.../C$ ./2.1A
Segmentation fault
[03/07/21] seed@VM:~/.../C$ sudo ./2.1A
```

Q3-

I sent ping to google through my second VM, with promiscuous mode and without the results was the same, I also tried to send ping to 1.2.3.4 and to my router' with all these 3 I did not noticed at ant changing in the behavior. I think it's because of the settings of the virtual machines, because promiscuous mode should cause my sniffer to see only packets that should pass through or to him, but now there is no differences between the situations!

Task 2.1B: Writing Filters:

ICMP between 192.168.56.6 – 142.250.186.142

At the first time I sent to 8.8.8.8 and my icmp sniffer did not captured these icmp packets (we can see that in wireshark), at the second time I sent ping from the another VM to 142.250.186.142 and this time my sniffer got these icmp packets

The screenshot displays two windows. The top window is Wireshark, titled '[SEED Labs] 2.1B_icmp.pcapng', showing a list of captured packets. The bottom window is a terminal titled 'seed@VM: ~', showing the execution of a ping command and the resulting output.

Wireshark Packet List:

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-03-07 13:3	192.168.56.4	8.8.8.8	ICMP	96	Echo (ping) request id=...
2	2021-03-07 13:3	8.8.8.8	192.168.56.4	ICMP	96	Echo (ping) reply id=...
3	2021-03-07 13:3	192.168.56.4	8.8.8.8	ICMP	96	Echo (ping) request id=...
4	2021-03-07 13:3	8.8.8.8	192.168.56.4	ICMP	96	Echo (ping) reply id=...
5	2021-03-07 13:3	192.168.56.4	8.8.8.8	ICMP	96	Echo (ping) request id=...
6	2021-03-07 13:3	8.8.8.8	192.168.56.4	ICMP	96	Echo (ping) reply id=...
7	2021-03-07 13:3	PcsCompu_7a:77:e7	RealtekU_12:35:09	ARP	42	Who has 192.168.56.1? Te...
8	2021-03-07 13:3	RealtekU_12:35:09	PcsCompu_7a:77:e7	ARP	60	192.168.56.1 is at 52:54...
9	2021-03-07 13:3	192.168.56.6	142.250.186.142	ICMP	96	Echo (ping) request id=...
10	2021-03-07 13:3	142.250.186.142	192.168.56.6	ICMP	96	Echo (ping) reply id=...
11	2021-03-07 13:3	192.168.56.6	142.250.186.142	ICMP	96	Echo (ping) request id=...
12	2021-03-07 13:3	142.250.186.142	192.168.56.6	ICMP	96	Echo (ping) reply id=...
13	2021-03-07 13:3	192.168.56.6	142.250.186.142	ICMP	96	Echo (ping) request id=...
14	2021-03-07 13:3	142.250.186.142	192.168.56.6	ICMP	96	Echo (ping) reply id=...

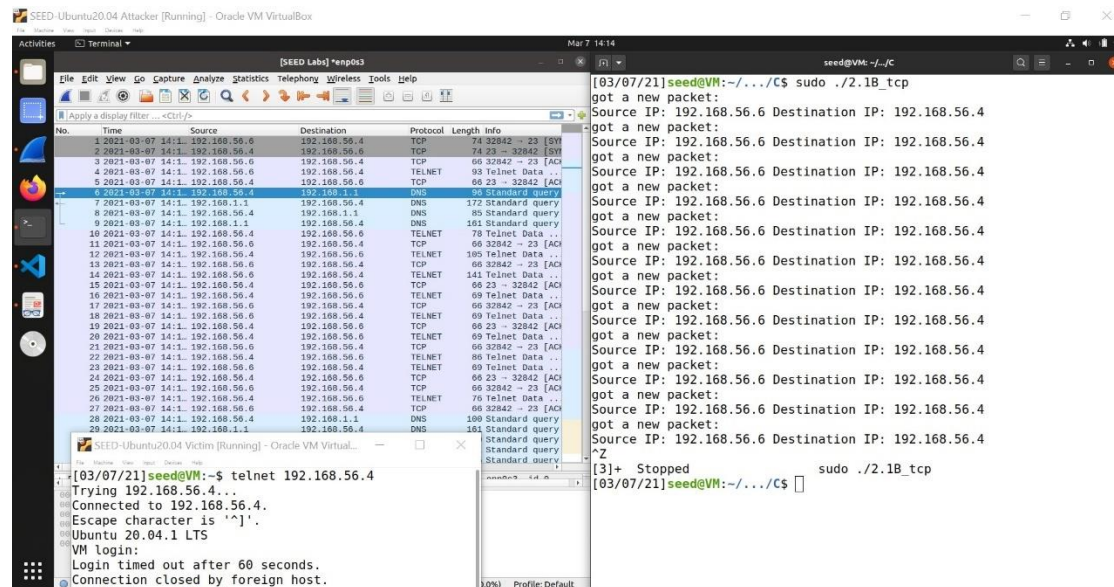
Terminal Output:

```
[03/07/21] seed@VM:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=89.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=85.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=70.2 ms
^Z
[2]+  Stopped                  ping 8.8.8.8
[03/07/21] seed@VM:~$

[03/07/21] seed@VM:~/.../C$ sudo ./2.1B
got a new packet:
Source IP: 192.168.56.6 Destination IP: 142.250.186.142
got a new packet:
Source IP: 142.250.186.142 Destination IP: 192.168.56.6
got a new packet:
Source IP: 192.168.56.6 Destination IP: 142.250.186.142
got a new packet:
Source IP: 142.250.186.142 Destination IP: 192.168.56.6
got a new packet:
Source IP: 192.168.56.6 Destination IP: 142.250.186.142
got a new packet:
Source IP: 142.250.186.142 Destination IP: 192.168.56.6
got a new packet:
[16]+  Stopped                  ping 142.250.186.142
[03/07/21] seed@VM:~$
```

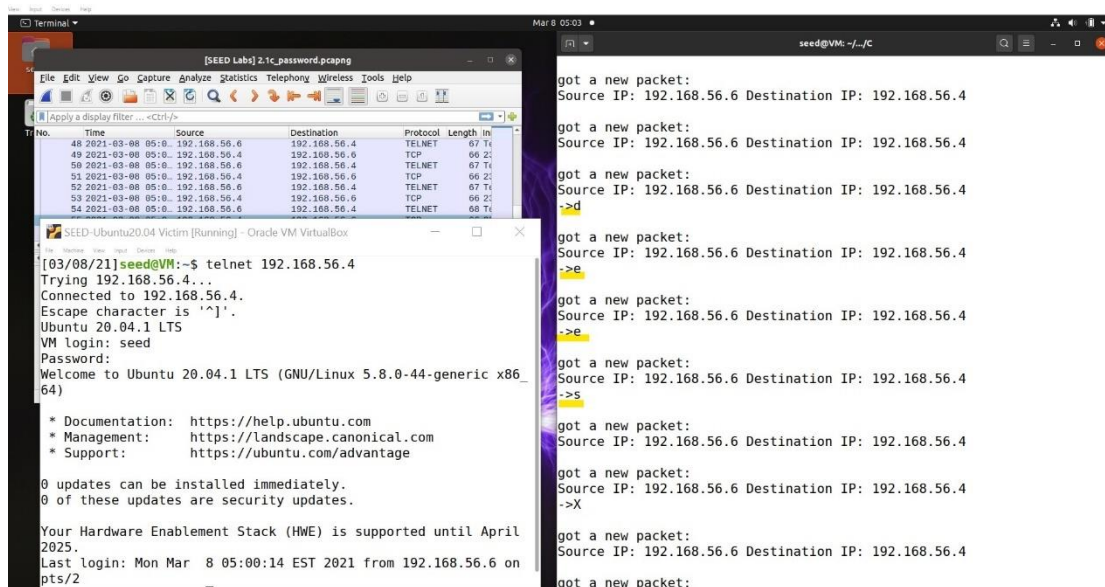
TCP packets with a destination port number in the range from 10 to 100:

For testing this case I sent telnet from my second VM to my main VM. telnet is a TCP packet with port port number 23. So we can see that my tcp_sniffer captured only the packets that has dest port num of 23, dest port number 23 will always be in the reply from my main VM to my second VM.



Task 2.1C: Sniffing Passwords:

I sent a telnet from my Victim VM to the Attacker VM. In telnet msg, any character belongs to the password sends by its own msg. the character will be at the and of the data part in the packet. So my password sniffer prints only this character with out the all unnecessary data part. We can see in the screenshot the password('dees') packet by packet marked in yellow .



Task 2.2A: Write a spoofing program:

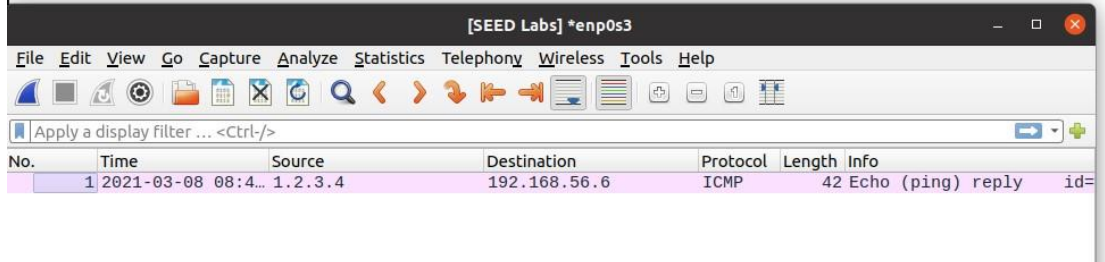
in this task I wrote a reply icmp spoofer, I sent an icmp echo reply msg from my Attacker VM to my Victim VM such that the victim think that the reply came from an non existence host in the internet 1.2.3.4:

we can see in the Wireshark report that my victim successfully got the pkt

```

seed@VM: ~/.../C
[03/08/21] seed@VM:~/.../C$ gcc 2.2A.c -o 2.2A
[03/08/21] seed@VM:~/.../C$ sudo ./2.2A
[+] spoofed ICMP echo reply sent
[03/08/21] seed@VM:~/.../C$

```



Wireshark capture showing a single ICMP Echo (ping) reply. The packet list table is as follows:

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-03-08 08:4...	1.2.3.4	192.168.56.6	ICMP	42	Echo (ping) reply id=

Task 2.2B: Spoof an ICMP Echo Request:

in this task I wrote a request icmp spoofer, I sent an icmp echo request msg from my Attacker VM to my Victim VM such that the victim think that the request came from an non existence host in the internet 1.2.3.4. when the Victim see the request he automatically sent an icmp echo reply to the spoofed ip.

```

[SEED Labs] *enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-03-08 08:5...	1.2.3.4	192.168.56.6	ICMP	42	Echo (ping) request id=
2	2021-03-08 08:5...	192.168.56.6	1.2.3.4	ICMP	60	Echo (ping) reply id=

```

seed@VM: ~/.../C
[03/08/21] seed@VM:~/.../C$ gcc 2.2B.c -o 2.2B
[03/08/21] seed@VM:~/.../C$ sudo ./2.2B
[+] spoofed ICMP echo request sent
[03/08/21] seed@VM:~/.../C$

```


Q4-

It depends on whether the length is bigger than the original length or smaller:

If smaller then we will get an error msg belongs to `sendto()` func, but if bigger, there will not be a problem to send the pkt, but all the extra space will be filled with zeros.

The image shows two windows. The top window is a terminal with the following commands and output:

```
seed@VM: ~/.../C
[03/08/21] seed@VM:~/.../C$ gcc Q4.c -o Q4
[03/08/21] seed@VM:~/.../C$ sudo ./Q4
sendto: Invalid argument
[03/08/21] seed@VM:~/.../C$ gcc Q4.c -o Q4
[03/08/21] seed@VM:~/.../C$ sudo ./Q4
[+] spoofed ICMP echo reply sent
[03/08/21] seed@VM:~/.../C$
```

The bottom window is a packet capture tool titled "[SEED Labs] Q4_extra_len.pcapng". It shows a single packet with the following details:

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-03-08 09:1...	1.2.3.4	192.168.56.6	ICMP	102	Echo (ping) reply id=...

The packet data is displayed in hexadecimal and ASCII. The ASCII part shows the string "zw...E" and "9...".

Q5-

no, that part will be filled by the OS

Q6-

When the OS will get a pkt, it will make a copy of the pkt and give the copied pkt to the opening Raw socket without any processing on like a regular socket. To open a raw socket we need the root privilege to access the access to the system API. If we

would not open our prog with root privilege then there will be an error with opening the raw socket.

Task 2.3: Sniff and then Spoof:

in this task we needed to send a spoofed icmp echo reply if we get an icmp echo request:

I have tested my prog by doing this process-

My victim VM first tried to send ping to a non existence host in the internet(1.2.3.4), he got response a spoofed response from my Attacker VM. In the second test my Victim VM sent ping to google(8.8.8.8), he got for any echo request two echo reply, one from the real server and one from the attacker.

```

[03/08/21]seed@VM:~$ 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=44 time=71.6 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=44 time=89.1 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=44 time=111 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=44 time=133 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=44 time=155 ms
^Z
[16]+  Stopped                  ping 1.2.3.4
[03/08/21]seed@VM:~$

[03/08/21]seed@VM:~/.../C$ gcc 2.3.c -lpcap -o 2.3
[03/08/21]seed@VM:~/.../C$ sudo ./2.3
got a new ICMP packet:
[+] spoofed ICMP echo reply sent
got a new ICMP packet:
got a new ICMP packet:
[+] spoofed ICMP echo reply sent
got a new ICMP packet:
got a new ICMP packet:
[+] spoofed ICMP echo reply sent
got a new ICMP packet:
got a new ICMP packet:
[+] spoofed ICMP echo reply sent
got a new ICMP packet:
got a new ICMP packet:
[+] spoofed ICMP echo reply sent
got a new ICMP packet:
[+] spoofed ICMP echo reply sent
got a new ICMP packet:
^Z
[2]+  Stopped                  sudo ./2.3
[03/08/21]seed@VM:~/.../C$

```

```
SEED-Ubuntu20.04 Victim [Running] - Oracle VM VirtualBox
Activities Terminal Mar 8 13:10 seed@VM: ~
64 bytes from 1.2.3.4: icmp_seq=1 ttl=44 time=71.6 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=44 time=89.1 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=44 time=111 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=44 time=133 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=44 time=155 ms
^Z
[16]+ Stopped ping 1.2.3.4
[03/08/21]seed@VM:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=73.1 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=44 time=173 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=71.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=44 time=195 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=71.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=44 time=224 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=71.3 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=44 time=244 ms (DUP!)

[03/08/21]seed@VM:~/.../C$ sudo ./2.3
got a new ICMP packet:
[+] spoofed ICMP echo reply sent
got a new ICMP packet:
got a new ICMP packet:
got a new ICMP packet:
[+] spoofed ICMP echo reply sent
got a new ICMP packet:
got a new ICMP packet:
got a new ICMP packet:
[+] spoofed ICMP echo reply sent
got a new ICMP packet:
got a new ICMP packet:
got a new ICMP packet:
[+] spoofed ICMP echo reply sent
got a new ICMP packet:
got a new ICMP packet:
[+] spoofed ICMP echo reply sent
got a new ICMP packet:
[+] spoofed ICMP echo reply sent
got a new ICMP packet:
```