

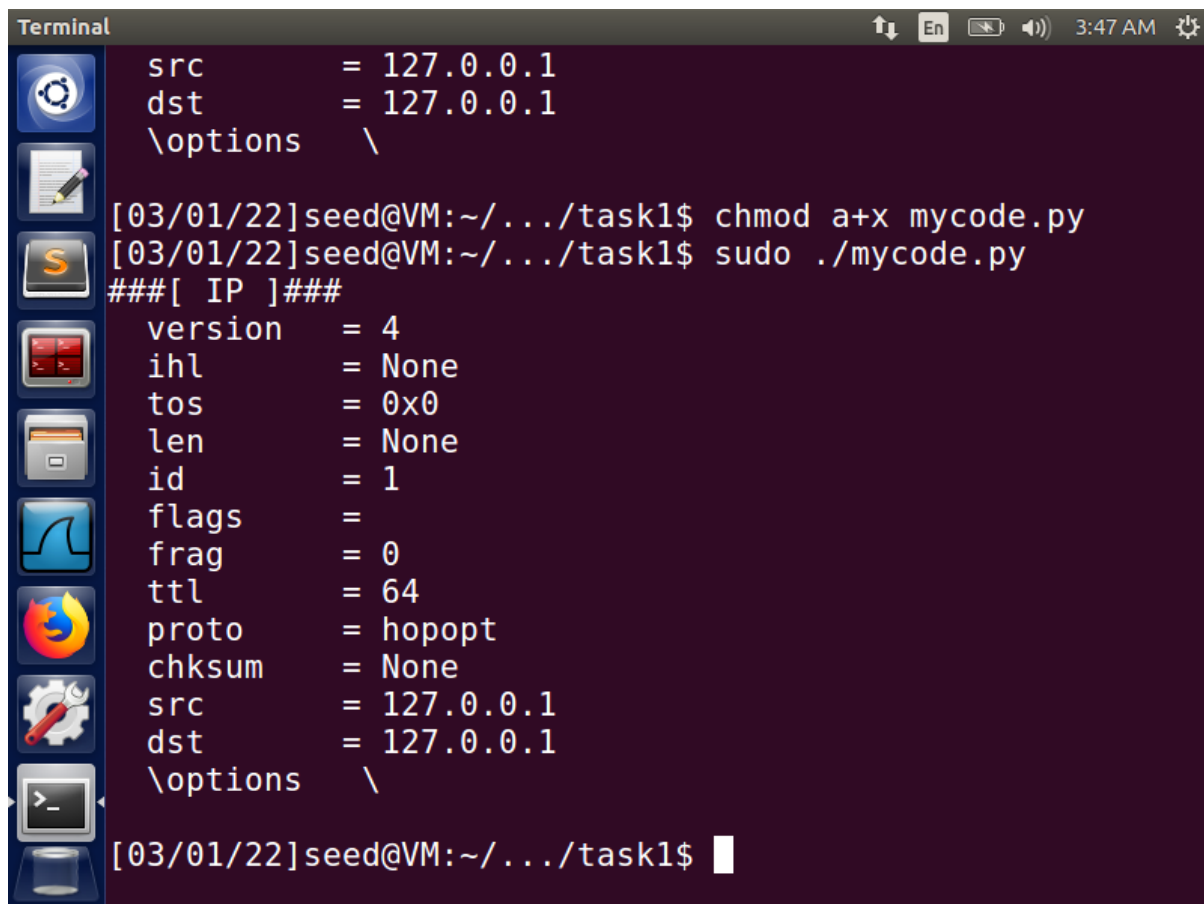


**Department of Computer Science
California State University, Channel
Islands**

**COMP-524: Security
Lab Report**

Lab Number: 5

Lab Topic: Packet Sniffing and Spoofing Lab

A terminal window titled "Terminal" with a dark purple background. The window has a top bar with system icons (up/down arrows, "En", battery, speaker, and time "3:47 AM"). On the left side, there is a vertical dock with icons for a gear, a notepad, a terminal, a file manager, a folder, a graph, a Firefox browser, a settings gear, and a terminal icon. The terminal text shows the user "seed" at a VM prompt, changing permissions and running a script. The script outputs IP-related settings.

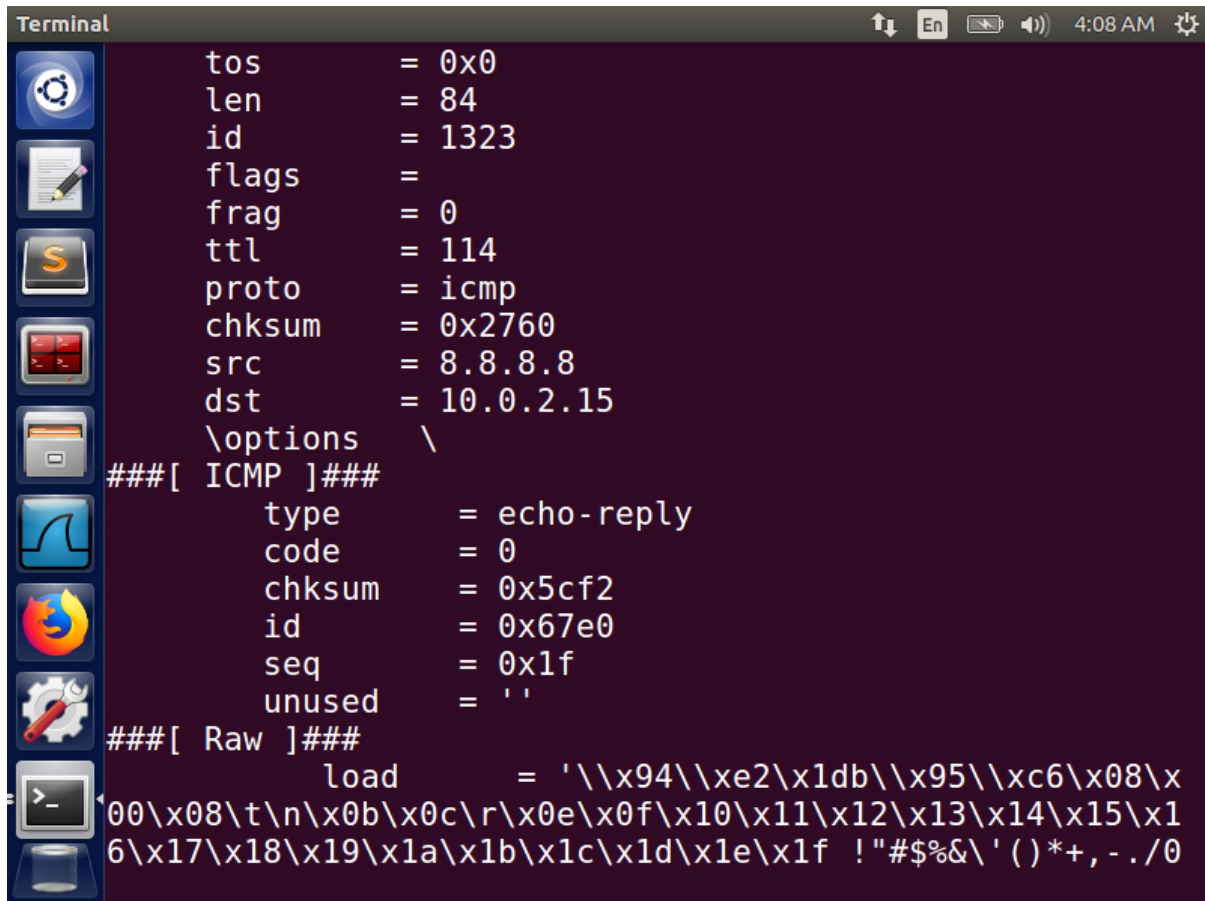
```
Terminal
src      = 127.0.0.1
dst      = 127.0.0.1
\options \

[03/01/22]seed@VM:~/.../task1$ chmod a+x mycode.py
[03/01/22]seed@VM:~/.../task1$ sudo ./mycode.py
###[ IP ]###
version  = 4
ihl      = None
tos      = 0x0
len      = None
id       = 1
flags    =
frag     = 0
ttl      = 64
proto    = hopopt
chksum   = None
src      = 127.0.0.1
dst      = 127.0.0.1
\options \

[03/01/22]seed@VM:~/.../task1$
```

When you run mycode.py, it displays some of the default settings. It includes the version number, checksum, as well as the source and destination IP addresses.

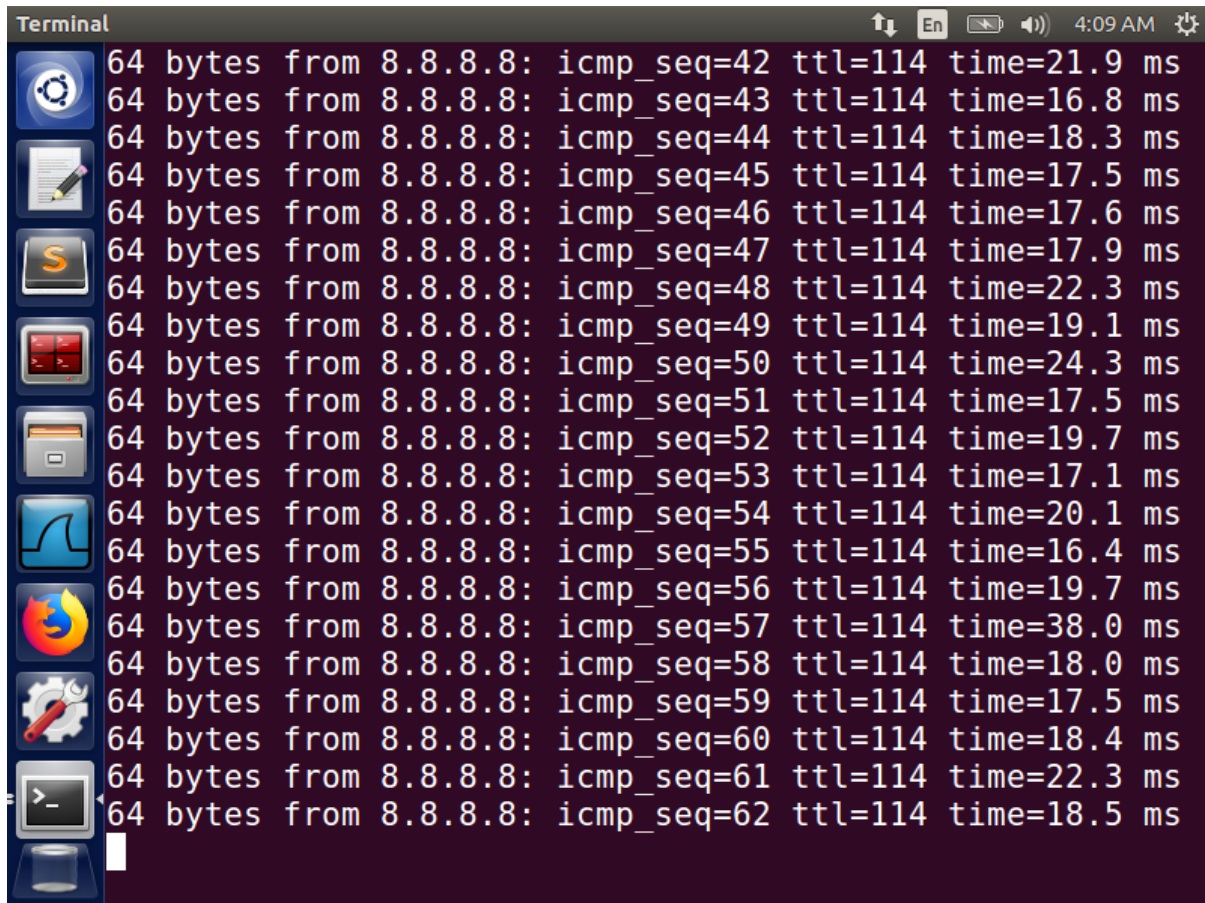
TASK1.1-A:

A terminal window titled "Terminal" with a dark purple background. On the left is a vertical dock with icons for various applications. The terminal displays the details of an ICMP echo-reply packet. The data is organized into two sections: a main header and an options section. The main header includes fields like tos, len, id, flags, frag, ttl, proto, checksum, src, and dst. The options section includes type, code, checksum, id, seq, and unused. At the bottom, there is a "Raw" section showing the packet data in hexadecimal and ASCII format.

```
tos      = 0x0
len      = 84
id       = 1323
flags    =
frag     = 0
ttl      = 114
proto    = icmp
chksum   = 0x2760
src      = 8.8.8.8
dst      = 10.0.2.15
\options \
###[ ICMP ]###
      type      = echo-reply
      code      = 0
      chksum    = 0x5cf2
      id        = 0x67e0
      seq       = 0x1f
      unused    = ''
###[ Raw ]###
      load      = '\\x94\\xe2\\x1db\\x95\\xc6\\x08\\x
00\\x08\\t\\n\\x0b\\x0c\\r\\x0e\\x0f\\x10\\x11\\x12\\x13\\x14\\x15\\x1
6\\x17\\x18\\x19\\x1a\\x1b\\x1c\\x1d\\x1e\\x1f !"#%&\\'()*+,-./0
```

The application is recording information about the source, destination, checksum, and other relevant stuff, as shown in the above snapshot. As a result, it indicates that it has received a response from Google to the destination IP address, which in my case is 10.0.2.15.

Ping 8.8.8.8 uses the ICMP protocol to ping the network.

A screenshot of a Linux terminal window titled "Terminal". The window shows the output of a series of ping commands to the IP address 8.8.8.8. The output consists of 15 lines, each showing "64 bytes from 8.8.8.8: icmp_seq=X ttl=114 time=Y ms", where X ranges from 42 to 62 and Y represents the round-trip time in milliseconds. The times vary, with the highest being 38.0 ms for sequence 57. The terminal has a dark background and a light-colored text. On the left side of the terminal window, there is a vertical dock with several application icons, including a gear, a document, a terminal, and a network icon. The top of the terminal window shows system status icons like network, battery, and the time 4:09 AM.

```
Terminal 4:09 AM
64 bytes from 8.8.8.8: icmp_seq=42 ttl=114 time=21.9 ms
64 bytes from 8.8.8.8: icmp_seq=43 ttl=114 time=16.8 ms
64 bytes from 8.8.8.8: icmp_seq=44 ttl=114 time=18.3 ms
64 bytes from 8.8.8.8: icmp_seq=45 ttl=114 time=17.5 ms
64 bytes from 8.8.8.8: icmp_seq=46 ttl=114 time=17.6 ms
64 bytes from 8.8.8.8: icmp_seq=47 ttl=114 time=17.9 ms
64 bytes from 8.8.8.8: icmp_seq=48 ttl=114 time=22.3 ms
64 bytes from 8.8.8.8: icmp_seq=49 ttl=114 time=19.1 ms
64 bytes from 8.8.8.8: icmp_seq=50 ttl=114 time=24.3 ms
64 bytes from 8.8.8.8: icmp_seq=51 ttl=114 time=17.5 ms
64 bytes from 8.8.8.8: icmp_seq=52 ttl=114 time=19.7 ms
64 bytes from 8.8.8.8: icmp_seq=53 ttl=114 time=17.1 ms
64 bytes from 8.8.8.8: icmp_seq=54 ttl=114 time=20.1 ms
64 bytes from 8.8.8.8: icmp_seq=55 ttl=114 time=16.4 ms
64 bytes from 8.8.8.8: icmp_seq=56 ttl=114 time=19.7 ms
64 bytes from 8.8.8.8: icmp_seq=57 ttl=114 time=38.0 ms
64 bytes from 8.8.8.8: icmp_seq=58 ttl=114 time=18.0 ms
64 bytes from 8.8.8.8: icmp_seq=59 ttl=114 time=17.5 ms
64 bytes from 8.8.8.8: icmp_seq=60 ttl=114 time=18.4 ms
64 bytes from 8.8.8.8: icmp_seq=61 ttl=114 time=22.3 ms
64 bytes from 8.8.8.8: icmp_seq=62 ttl=114 time=18.5 ms
```

Wireshark is a network capture and analysis program that displays the network's source, destination, and protocol.

Only the ICMP protocol source and destination analysis were captured using the function sniff and filter ICMP.

Capturing from enp0s3

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

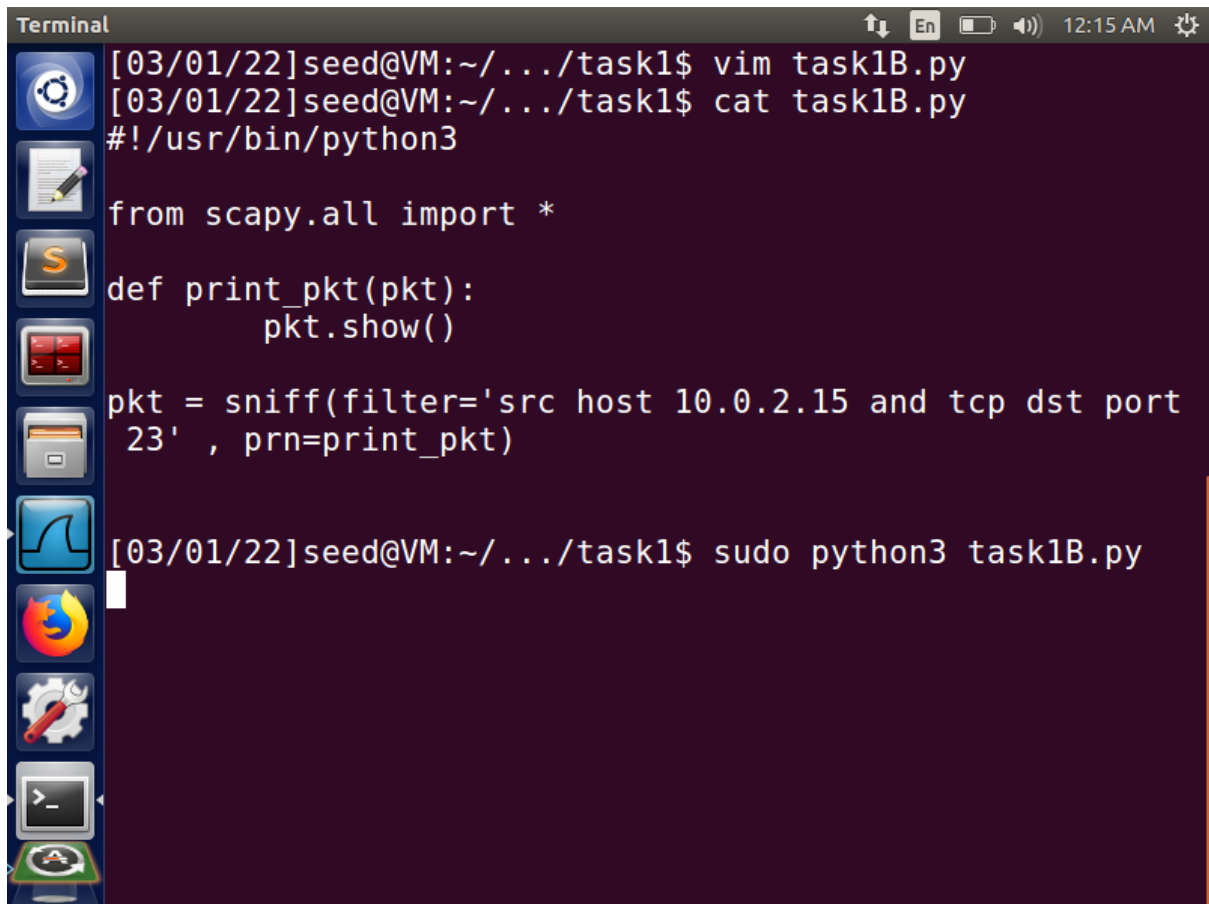
No.	Time	Source	Destination	Pr
1	2022-03-01 04:10:44.4661107...	10.0.2.15	8.8.8.8	I
2	2022-03-01 04:10:44.4877640...	8.8.8.8	10.0.2.15	I
3	2022-03-01 04:10:45.4680950...	10.0.2.15	8.8.8.8	I
4	2022-03-01 04:10:45.4852989...	8.8.8.8	10.0.2.15	I
5	2022-03-01 04:10:46.4698359...	10.0.2.15	8.8.8.8	I
6	2022-03-01 04:10:46.4874366...	8.8.8.8	10.0.2.15	I
7	2022-03-01 04:10:47.4727239...	10.0.2.15	8.8.8.8	I
8	2022-03-01 04:10:47.4924101...	8.8.8.8	10.0.2.15	I
9	2022-03-01 04:10:48.4733997...	10.0.2.15	8.8.8.8	I
10	2022-03-01 04:10:48.4903099...	8.8.8.8	10.0.2.15	I
11	2022-03-01 04:10:49.4753600...	10.0.2.15	8.8.8.8	I

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface
 Ethernet II, Src: PcsCompu_cc:f6:c6 (08:00:27:cc:f6:c6), Dst: RealtekU_12:35:00
 Internet Protocol Version 4, Src: 10.0.2.15, Dst: 8.8.8.8
 Internet Control Message Protocol

0000	52 54 00 12 35 00 08 00	27 cc f6 c6 08 00 45 00	RT..5... '.....E.
0010	00 54 38 c2 40 00 40 01	e5 c8 0a 00 02 0f 08 08	.T8.@.@.
0020	08 08 08 00 bc 3c 68 45	00 19 14 e3 1d 62 af 1c<hEb..
0030	07 00 08 09 0a 0b 0c 0d	0e 0f 10 11 12 13 14 15
0040	16 17 18 19 1a 1b 1c 1d	1e 1f 20 21 22 23 24 25 !"#\$\$%
0050	26 27 28 29 2a 2b 2c 2d	2e 2f 30 31 32 33 34 35	&'()*+,-./012345

enp0s3: <live capture in progress> Packets: 14 · Displayed: 14 (100.0%) Profile: Defau

TASK 1.1B:



A terminal window titled "Terminal" with a dark purple background. The window shows a series of commands and their outputs. On the left side, there is a vertical dock with various application icons. The terminal content is as follows:

```
[03/01/22]seed@VM:~/.../task1$ vim task1B.py
[03/01/22]seed@VM:~/.../task1$ cat task1B.py
#!/usr/bin/python3

from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter='src host 10.0.2.15 and tcp dst port
23' , prn=print_pkt)

[03/01/22]seed@VM:~/.../task1$ sudo python3 task1B.py
```

Filtering is done with a specific host and TCP destination port 23 according to the given application.

```
Terminal
ost
[03/01/22]seed@VM:~/.../task1$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Tue Mar  1 00:26:27 EST 2022 from 10.0.2.15
on pts/19
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-gener
ic i686)

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

1 package can be updated.
0 updates are security updates.

[03/01/22]seed@VM:~$
```

```
Terminal
chksum      = 0x3068
src         = 10.0.2.15
dst         = 10.0.2.5
\options    \
###[ TCP ]###
sport       = 58386
dport       = telnet
seq         = 2439094472
ack         = 1828268980
dataofs     = 8
reserved    = 0
flags       = PA
window      = 237
chksum      = 0x1843
urgptr      = 0
options     = [('NOP', None), ('NOP', None), ('Ti
mestamp', (1400939, 1400537))]
###[ Raw ]###
load        = '\\xff\\xfa\\x1f\\x007\\x00\\x15\\xf
f\\xf0'
```

The source of the telnet was 10.0.2.15, and the destination was 10.0.2.5.

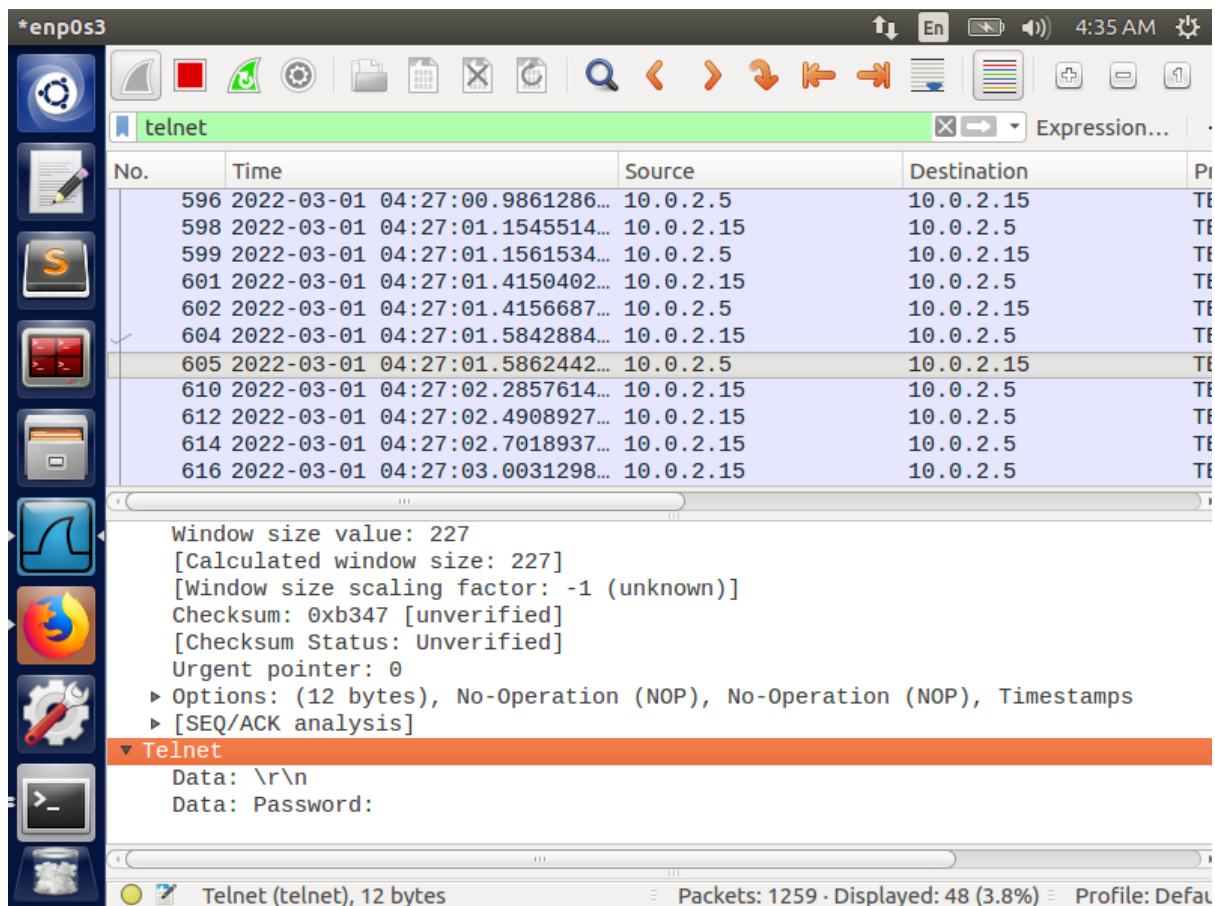
The image shows a Wireshark packet capture window titled '*enp0s3'. The filter bar is set to 'telnet'. The packet list shows several Telnet packets from source 10.0.2.15 to destination 10.0.2.5. The packet details pane shows the structure of a Telnet packet (Frame 525), including Ethernet II, Internet Protocol Version 4, and Transmission Control Protocol. The packet bytes pane shows the raw data in hexadecimal and ASCII.

Time	Source	Destination	Protocol
25 2022-03-01 04:26:54.2504274...	10.0.2.15	10.0.2.5	TELNET
26 2022-03-01 04:26:54.2513181...	10.0.2.5	10.0.2.15	TELNET
28 2022-03-01 04:26:54.9589162...	10.0.2.15	10.0.2.5	TELNET
29 2022-03-01 04:26:54.9595773...	10.0.2.5	10.0.2.15	TELNET
36 2022-03-01 04:26:56.3745561...	10.0.2.15	10.0.2.5	TELNET
37 2022-03-01 04:26:56.3752778...	10.0.2.5	10.0.2.15	TELNET
39 2022-03-01 04:26:56.5297155...	10.0.2.15	10.0.2.5	TELNET
40 2022-03-01 04:26:56.5312689...	10.0.2.5	10.0.2.15	TELNET
50 2022-03-01 04:26:57.3293235...	10.0.2.15	10.0.2.5	TELNET
51 2022-03-01 04:26:57.3300235...	10.0.2.5	10.0.2.15	TELNET
55 2022-03-01 04:26:58.1194151...	10.0.2.15	10.0.2.5	TELNET

Frame 525: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface enp0s3
Ethernet II, Src: PcsCompu_cc:f6:c6 (08:00:27:cc:f6:c6), Dst: PcsCompu_d0:fd:d6 (08:00:27:00:fd:d6)
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.5
Transmission Control Protocol, Src Port: 58386, Dst Port: 23, Seq: 2439094504, Len: 69
Telnet

0000 08 00 27 d0 fd d6 08 00 27 cc f6 c6 08 00 45 10 ..'.....'.....E.
0010 00 37 f2 5a 40 00 40 06 30 43 0a 00 02 0f 0a 00 .7.Z@. @.OC.....
0020 02 05 e4 12 00 17 91 61 a0 e8 6c f9 2d e8 80 18a ..l.-...
0030 00 ed 18 3d 00 00 01 01 08 0a 00 16 f7 cd 00 16 ...=.... ..
0040 a4 63 1b 5b 41 .c.[A

wireshark_enp0s3_20220301042617_w7lO3t Packets: 718 · Displayed: 48 (6.7%) Profile: Default

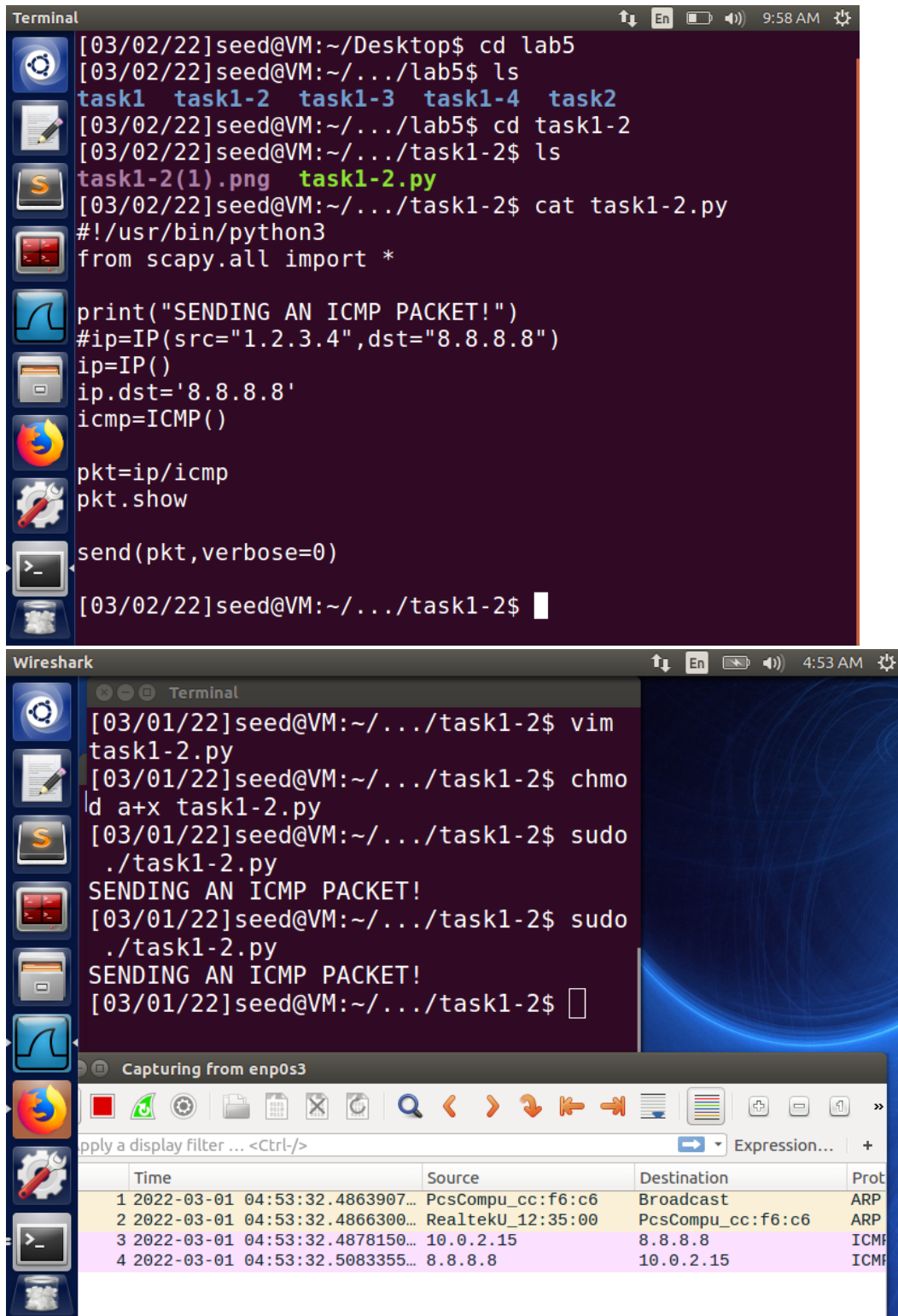


As can be seen, telnet can be more risky. The password will be whatever follows the selection in the above image.



It can also capture networks using the IP addresses 8.8.8.8 and 8.8.4.4.

TASK1.2:



The image shows two windows: a terminal and Wireshark. The terminal window shows the execution of a Python script that sends an ICMP packet from 1.2.3.4 to 8.8.8.8. The Wireshark window shows the capture of this packet on the enp0s3 interface.

```
[03/02/22]seed@VM:~/Desktop$ cd lab5
[03/02/22]seed@VM:~/.../lab5$ ls
task1 task1-2 task1-3 task1-4 task2
[03/02/22]seed@VM:~/.../lab5$ cd task1-2
[03/02/22]seed@VM:~/.../task1-2$ ls
task1-2(1).png task1-2.py
[03/02/22]seed@VM:~/.../task1-2$ cat task1-2.py
#!/usr/bin/python3
from scapy.all import *

print("SENDING AN ICMP PACKET!")
#ip=IP(src="1.2.3.4",dst="8.8.8.8")
ip=IP()
ip.dst='8.8.8.8'
icmp=ICMP()

pkt=ip/icmp
pkt.show

send(pkt,verbose=0)
[03/02/22]seed@VM:~/.../task1-2$
```

Wireshark window showing the capture of the ICMP packet:

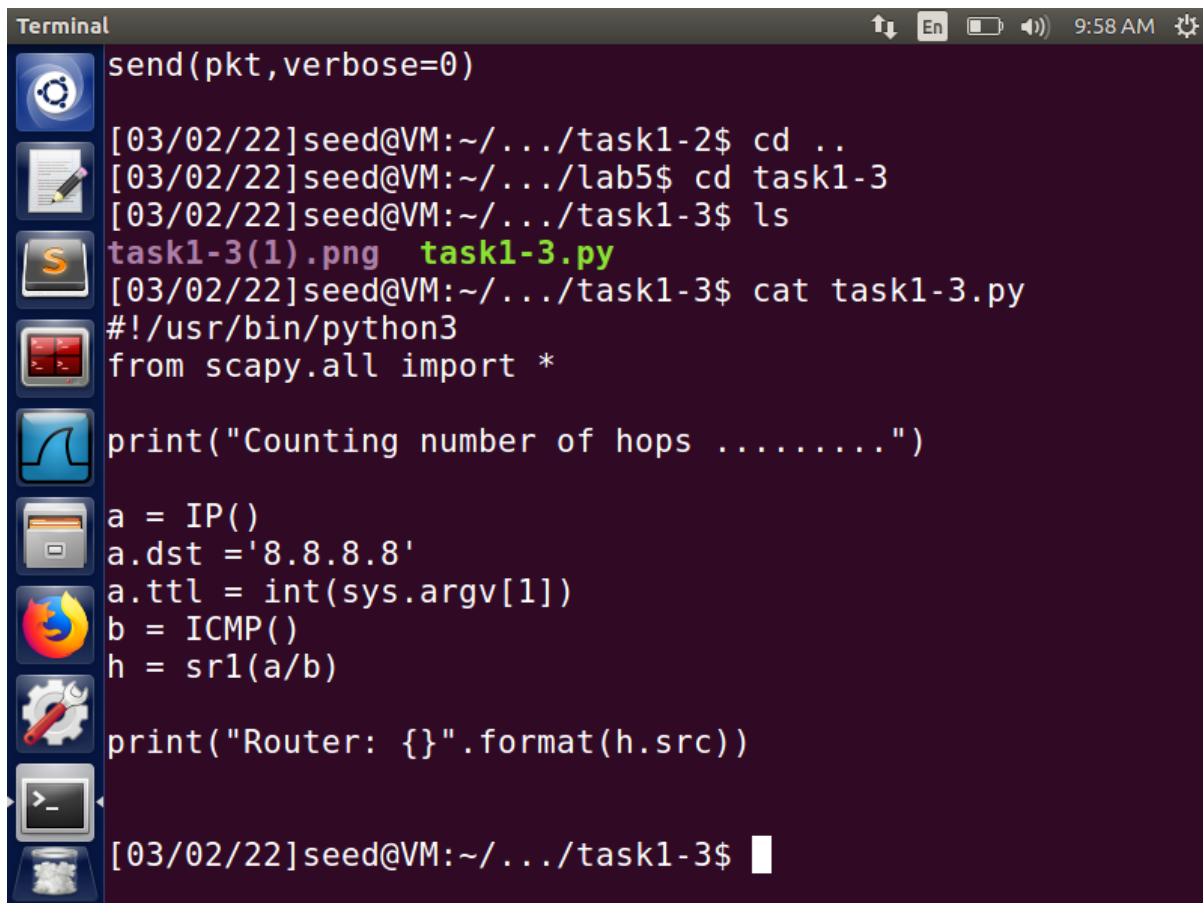
```
[03/01/22]seed@VM:~/.../task1-2$ vim task1-2.py
[03/01/22]seed@VM:~/.../task1-2$ chmod a+x task1-2.py
[03/01/22]seed@VM:~/.../task1-2$ sudo ./task1-2.py
SENDING AN ICMP PACKET!
[03/01/22]seed@VM:~/.../task1-2$ sudo ./task1-2.py
SENDING AN ICMP PACKET!
[03/01/22]seed@VM:~/.../task1-2$
```

Wireshark packet list:

No.	Time	Source	Destination	Protocol
1	2022-03-01 04:53:32.4863907...	PcsCompu_cc:f6:c6	Broadcast	ARP
2	2022-03-01 04:53:32.4866300...	RealtekU_12:35:00	PcsCompu_cc:f6:c6	ARP
3	2022-03-01 04:53:32.4878150...	10.0.2.15	8.8.8.8	ICMP
4	2022-03-01 04:53:32.5083355...	8.8.8.8	10.0.2.15	ICMP

As shown in the screenshot above, the software is sending ICMP packets. The ICMP packet is being captured by Wireshark.

TASK 1.3:

A terminal window titled "Terminal" with a dark purple background. The window shows a series of commands and their outputs. The user navigates from a previous directory to the current one, lists files, and then runs a Python script. The script uses Scapy to create an ICMP packet and send it. The output shows the source IP of the packet.

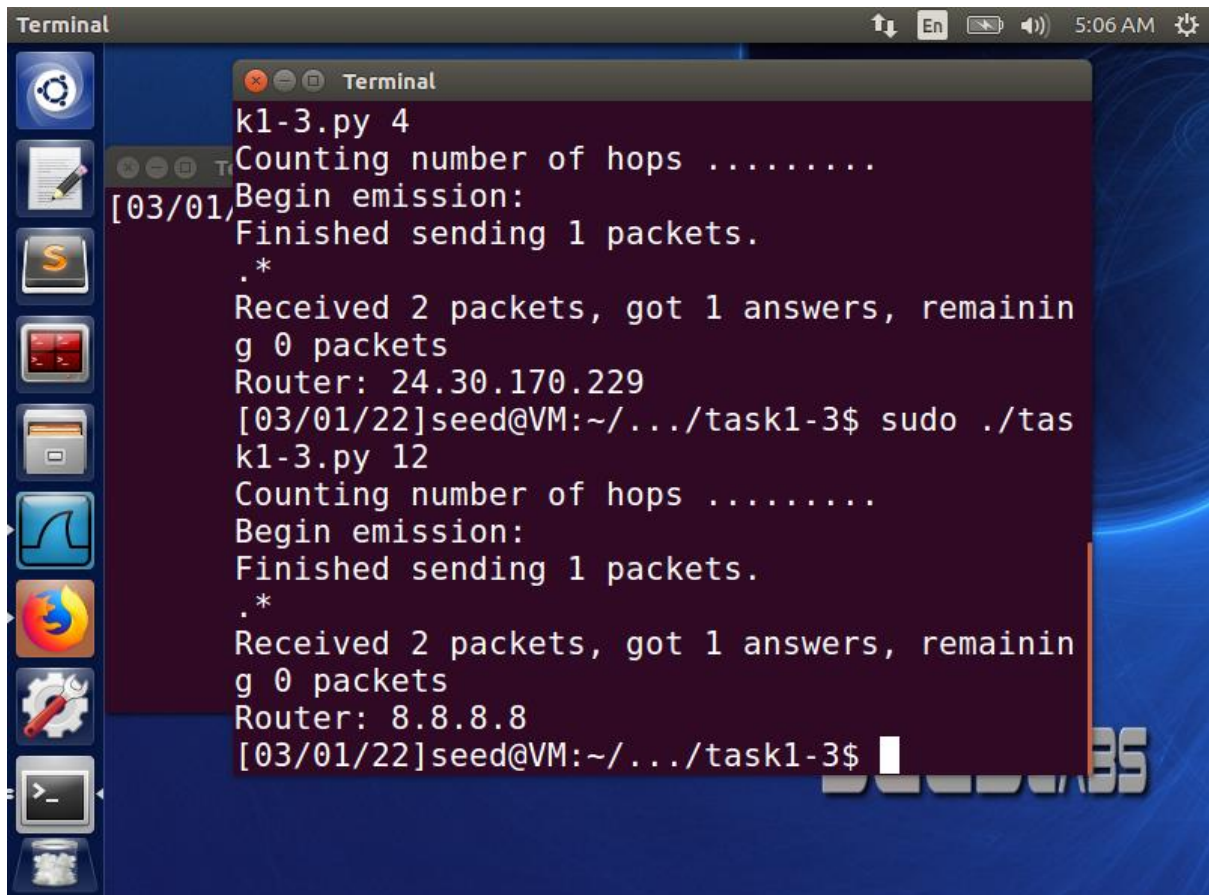
```
Terminal
send(pkt,verbose=0)
[03/02/22]seed@VM:~/.../task1-2$ cd ..
[03/02/22]seed@VM:~/.../lab5$ cd task1-3
[03/02/22]seed@VM:~/.../task1-3$ ls
task1-3(1).png  task1-3.py
[03/02/22]seed@VM:~/.../task1-3$ cat task1-3.py
#!/usr/bin/python3
from scapy.all import *

print("Counting number of hops .....")

a = IP()
a.dst = '8.8.8.8'
a.ttl = int(sys.argv[1])
b = ICMP()
h = sr1(a/b)

print("Router: {}".format(h.src))

[03/02/22]seed@VM:~/.../task1-3$
```

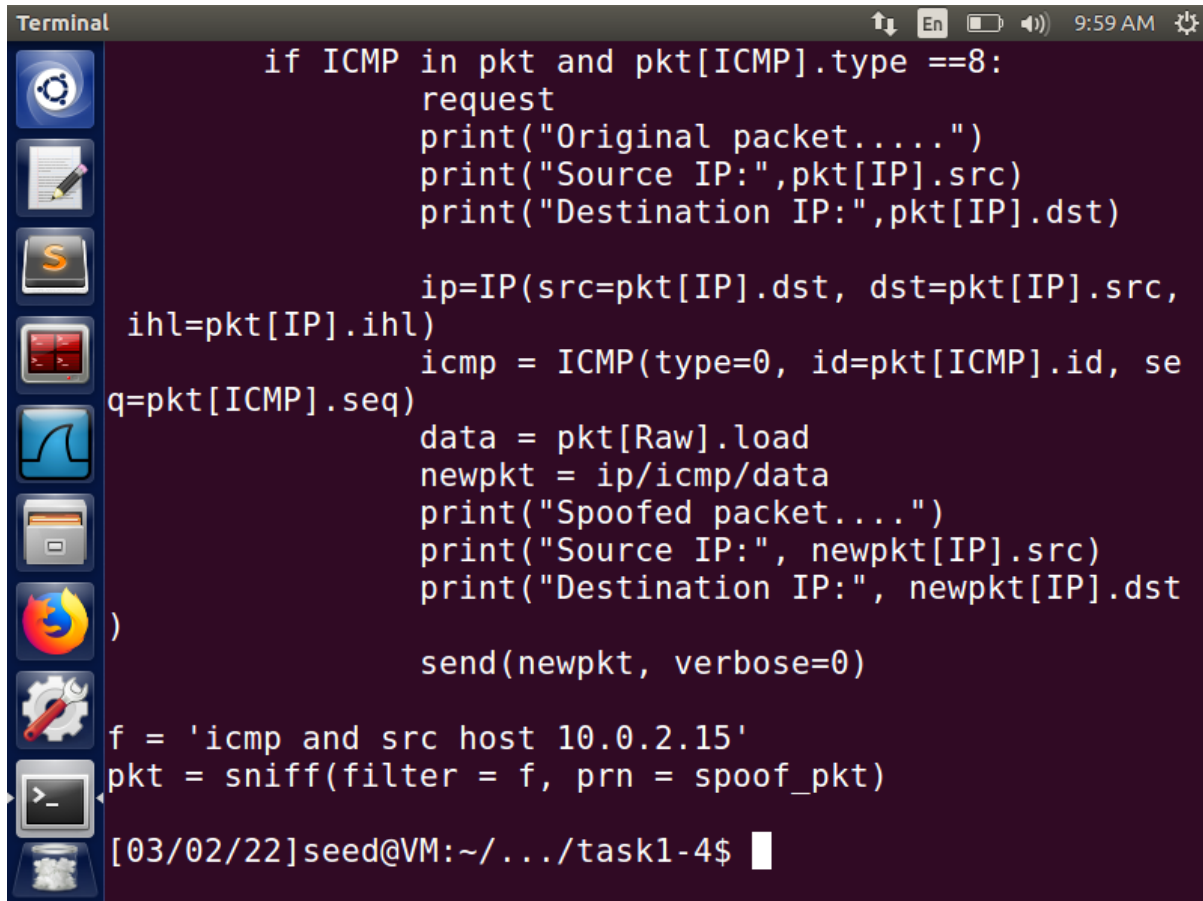


```
Terminal
k1-3.py 4
Counting number of hops .....
Begin emission:
[03/01/22] Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Router: 24.30.170.229
[03/01/22]seed@VM:~/.../task1-3$ sudo ./task1-3.py 12
Counting number of hops .....
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Router: 8.8.8.8
[03/01/22]seed@VM:~/.../task1-3$
```

The traceroute shows where the packet travels. The software counts how many hops or packets there are between my network and the specified destination.

I can see the server 24.3.170.229 when I use 4 hops, but I can only see the server 8.8.8.8 when I use 12 hops. I tried 10 and 11 hops, and it appears that 10 gives me something different, while 11 hops gives me the 8.8.8.8 destination. The minimal number of hops required to reach the server 8.8.8.8 in my case is 11.

TASK 1.4:

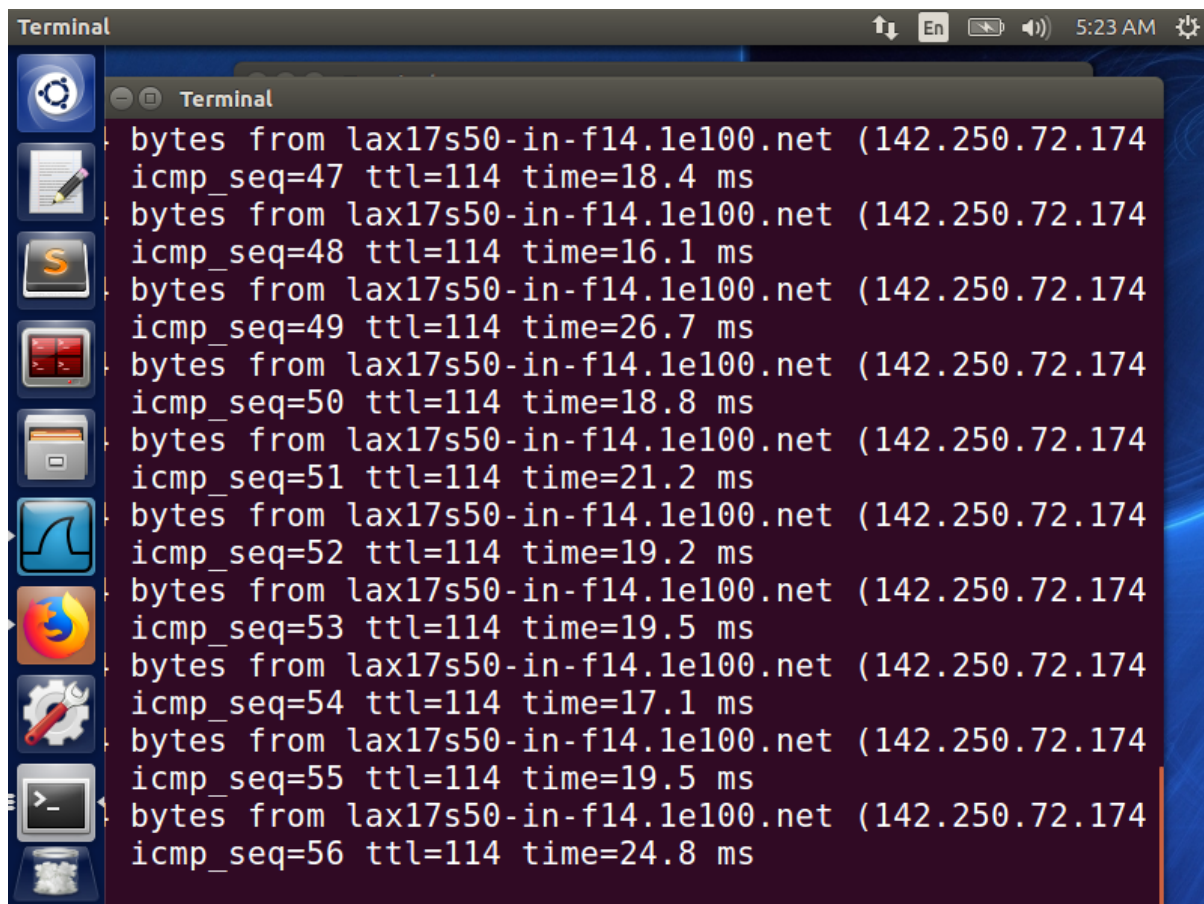
A terminal window titled "Terminal" with a dark background and light text. The window shows a Python script for ICMP spoofing. The script checks if an ICMP packet is a request, prints its details, and then creates a spoofed packet by swapping the source and destination IP addresses. The script also includes a filter to capture ICMP packets from a specific host and a function to print the spoofed packet details. The terminal shows the script being executed, with the prompt indicating the user is 'seed' on a VM.

```
Terminal 9:59 AM
if ICMP in pkt and pkt[ICMP].type ==8:
    request
    print("Original packet.....")
    print("Source IP:",pkt[IP].src)
    print("Destination IP:",pkt[IP].dst)

    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)

f = 'icmp and src host 10.0.2.15'
pkt = sniff(filter = f, prn = spoof_pkt)

[03/02/22]seed@VM:~/.../task1-4$
```

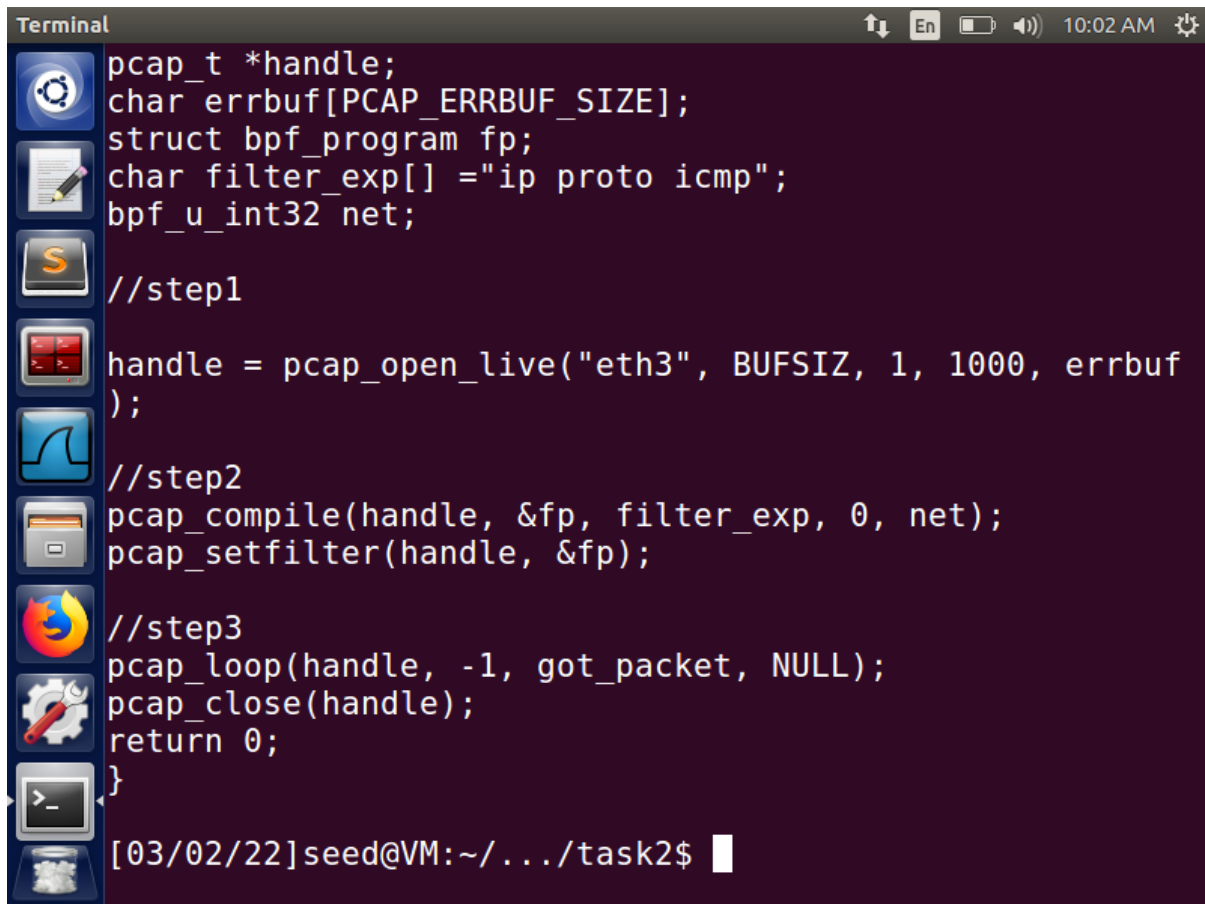

A screenshot of a Linux terminal window. The window title is "Terminal". The terminal output shows a series of ICMP echo request results. Each line contains the text "bytes from lax17s50-in-f14.1e100.net (142.250.72.174", followed by "icmp_seq=" and a sequence number from 47 to 56, "ttl=114", and "time=" followed by a time value in milliseconds. The sequence numbers increase by 1 for each line, and the time values vary between approximately 16.1 ms and 26.7 ms. The terminal has a dark background with light-colored text. On the left side of the terminal window, there is a vertical dock with several application icons, including a gear, a notepad, a terminal, and others. The top of the window shows system status icons like network, volume, and battery, along with the time "5:23 AM".

```
bytes from lax17s50-in-f14.1e100.net (142.250.72.174
icmp_seq=47 ttl=114 time=18.4 ms
bytes from lax17s50-in-f14.1e100.net (142.250.72.174
icmp_seq=48 ttl=114 time=16.1 ms
bytes from lax17s50-in-f14.1e100.net (142.250.72.174
icmp_seq=49 ttl=114 time=26.7 ms
bytes from lax17s50-in-f14.1e100.net (142.250.72.174
icmp_seq=50 ttl=114 time=18.8 ms
bytes from lax17s50-in-f14.1e100.net (142.250.72.174
icmp_seq=51 ttl=114 time=21.2 ms
bytes from lax17s50-in-f14.1e100.net (142.250.72.174
icmp_seq=52 ttl=114 time=19.2 ms
bytes from lax17s50-in-f14.1e100.net (142.250.72.174
icmp_seq=53 ttl=114 time=19.5 ms
bytes from lax17s50-in-f14.1e100.net (142.250.72.174
icmp_seq=54 ttl=114 time=17.1 ms
bytes from lax17s50-in-f14.1e100.net (142.250.72.174
icmp_seq=55 ttl=114 time=19.5 ms
bytes from lax17s50-in-f14.1e100.net (142.250.72.174
icmp_seq=56 ttl=114 time=24.8 ms
```

This program is only interested in capturing the ICMP packet(Which comes from the specific host) with echo requests. The task is to capture those and impersonate them. As the screenshot says it is impersonating the google server.

Task 2(2.1):

In c we use pcap library and in python, we use scapy module to do the tasks.

A terminal window titled "Terminal" with a dark background and light text. The window shows a C program for capturing ICMP packets on the eth3 interface. The code includes headers for pcap, bpf, and sys. It defines a filter expression "ip proto icmp" and a network interface "net". The program then opens the interface, compiles the filter, sets it, and enters a loop to capture packets. The terminal shows the code being typed, with some lines already present. The prompt is [03/02/22]seed@VM:~/.../task2\$.

```
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
struct bpf_program fp;
char filter_exp[] = "ip proto icmp";
bpf_u_int32 net;

//step1

handle = pcap_open_live("eth3", BUFSIZ, 1, 1000, errbuf);

//step2
pcap_compile(handle, &fp, filter_exp, 0, net);
pcap_setfilter(handle, &fp);

//step3
pcap_loop(handle, -1, got_packet, NULL);
pcap_close(handle);
return 0;
}

[03/02/22]seed@VM:~/.../task2$
```



```
Terminal
);
//step2
pcap_compile(handle, &fp, filter_exp, 0, net);
pcap_setfilter(handle, &fp);
//step3
pcap_loop(handle, -1, got_packet, NULL);
pcap_close(handle);
return 0;
}
[03/02/22]seed@VM:~/.../task2$ gcc -o sniff sniff.c -lpcap
[03/02/22]seed@VM:~/.../task2$ ls
filterICMP.out      last_image.png    sniffTelnet.c
filterTelnet.out    sniff              TASK2.1.png
ICMP.c              sniff.c
ICMP_image.png      sniff.out
[03/02/22]seed@VM:~/.../task2$ sudo ./sniff
```

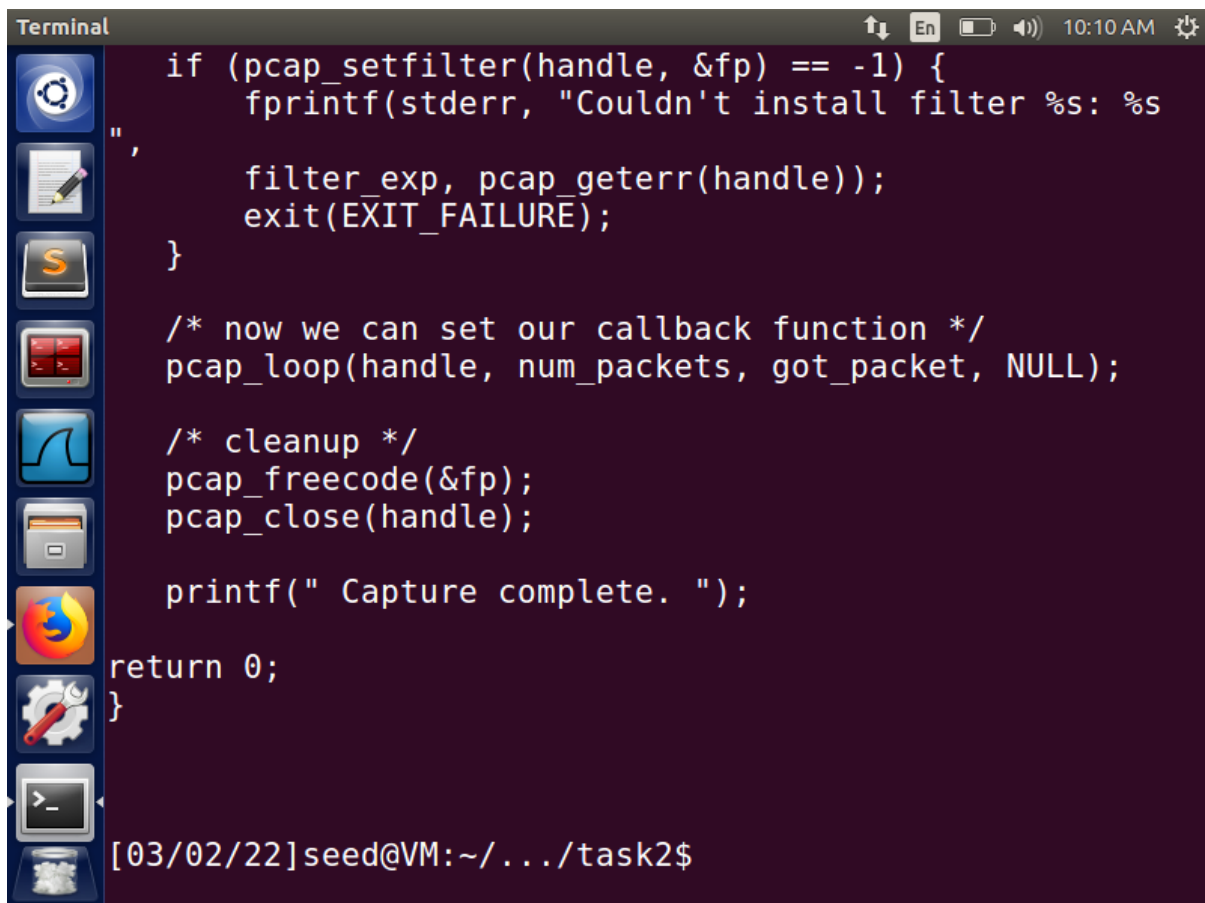
I have written a program that print out the source and destination IP addresses

pcap_loop is function that program gonna go through and capture the packet

The program print out the ip source and ip destination

TASK 2.1 A:

The task is required to write a sniffer program that prints out the source and destination IP addresses of each captured packet.

A terminal window titled "Terminal" with a dark purple background. The window shows C code for a packet sniffer. The code includes a conditional check for installing a filter, a cleanup section, and a return statement. The terminal also shows a sidebar with application icons and a status bar at the bottom with system information and the current directory path.

```
if (pcap_setfilter(handle, &fp) == -1) {
    fprintf(stderr, "Couldn't install filter %s: %s",
            filter_exp, pcap_geterr(handle));
    exit(EXIT_FAILURE);
}

/* now we can set our callback function */
pcap_loop(handle, num_packets, got_packet, NULL);

/* cleanup */
pcap_freecode(&fp);
pcap_close(handle);

printf(" Capture complete. ");

return 0;
}
```

[03/02/22]seed@VM:~/.../task2\$

Q1 Fundamental function calls that are used for sniffing programs include;

- 1. Determining and setting up the type of ethernet interface that the program will utilize.**
- 2. The initialization of the PCAP to create a session, typically there is one session per device to be sniffed.**
- 3. The call to set traffic filtering rules, this ensures that the type of traffic sniffed on an interface is the type one is going for.**
- 4. The execution of the sniff.**
- 5. Termination of the session.**

Q2

In Linux whenever network interfaces need to be accessed it is required to have root access, in this case, the program needs the ability to utilize raw sockets to send packets in the way it does, without the root user capacities the Network Interface Card would be inaccessible hence the ability to use/create raw sockets is lost.

Q3 `ifconfig [interface] promisc ip link set [interface] promisc on ip a show eth1 | grep -i promisc`

There is a difference between information sniffed in nonpromiscuous and promiscuous

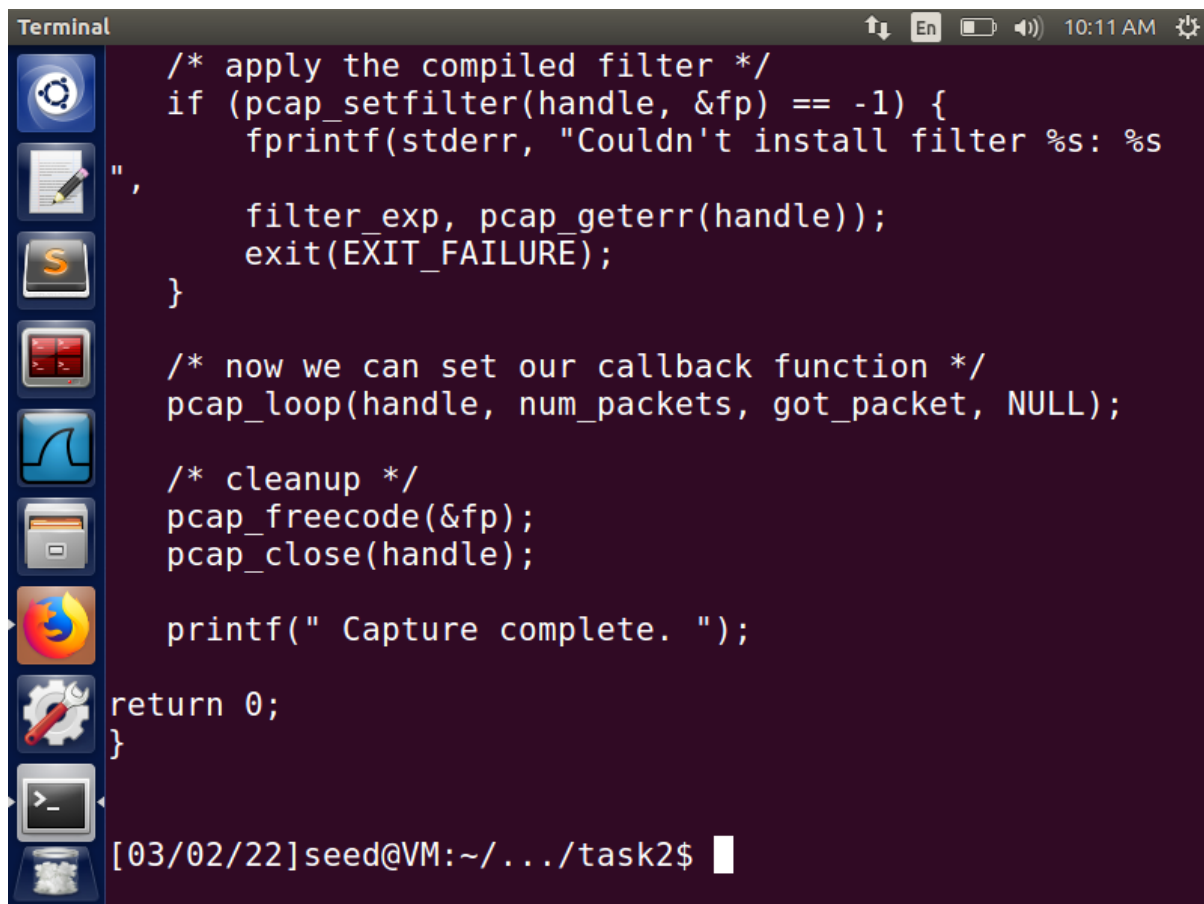
mode after switching the mode on and off using Virtual Box. It can also be turned off by modifying this line in the code

```
handle = pcap_open_live(dev, BUFSIZ, 1, 1000, errbuf);
```

Changing the red 1 to a 0.

TASK 2.1 B and 2.1 C:

Part 2.1B AND 2.1C ARE CODED IN A SINGLE CODE AS THEY ARE RELATED TO EACH OTHER AND CAN NOT BE CODED SEPARATELY

A terminal window titled "Terminal" with a dark purple background. The window shows a C program snippet for applying a pcap filter. The code includes comments and function calls like pcap_setfilter, fprintf, pcap_geterr, exit, pcap_loop, pcap_freecode, pcap_close, and printf. The terminal prompt at the bottom is [03/02/22]seed@VM:~/.../task2\$. On the left side of the terminal, there is a vertical dock with several application icons: a gear, a notepad, a terminal, a file manager, a web browser, a terminal, a terminal, a terminal, a terminal, a terminal, a terminal, and a terminal.

```
/* apply the compiled filter */
if (pcap_setfilter(handle, &fp) == -1) {
    fprintf(stderr, "Couldn't install filter %s: %s",
            filter_exp, pcap_geterr(handle));
    exit(EXIT_FAILURE);
}

/* now we can set our callback function */
pcap_loop(handle, num_packets, got_packet, NULL);

/* cleanup */
pcap_freecode(&fp);
pcap_close(handle);

printf(" Capture complete. ");

return 0;
}

[03/02/22]seed@VM:~/.../task2$
```

Hence, As you can see we have got the Password as " d e e s".

There were the filters used to accomplish the required task, they are built into pcap and syntax was used from tcpdump website. Note the password dees becomes clear as packets are received.