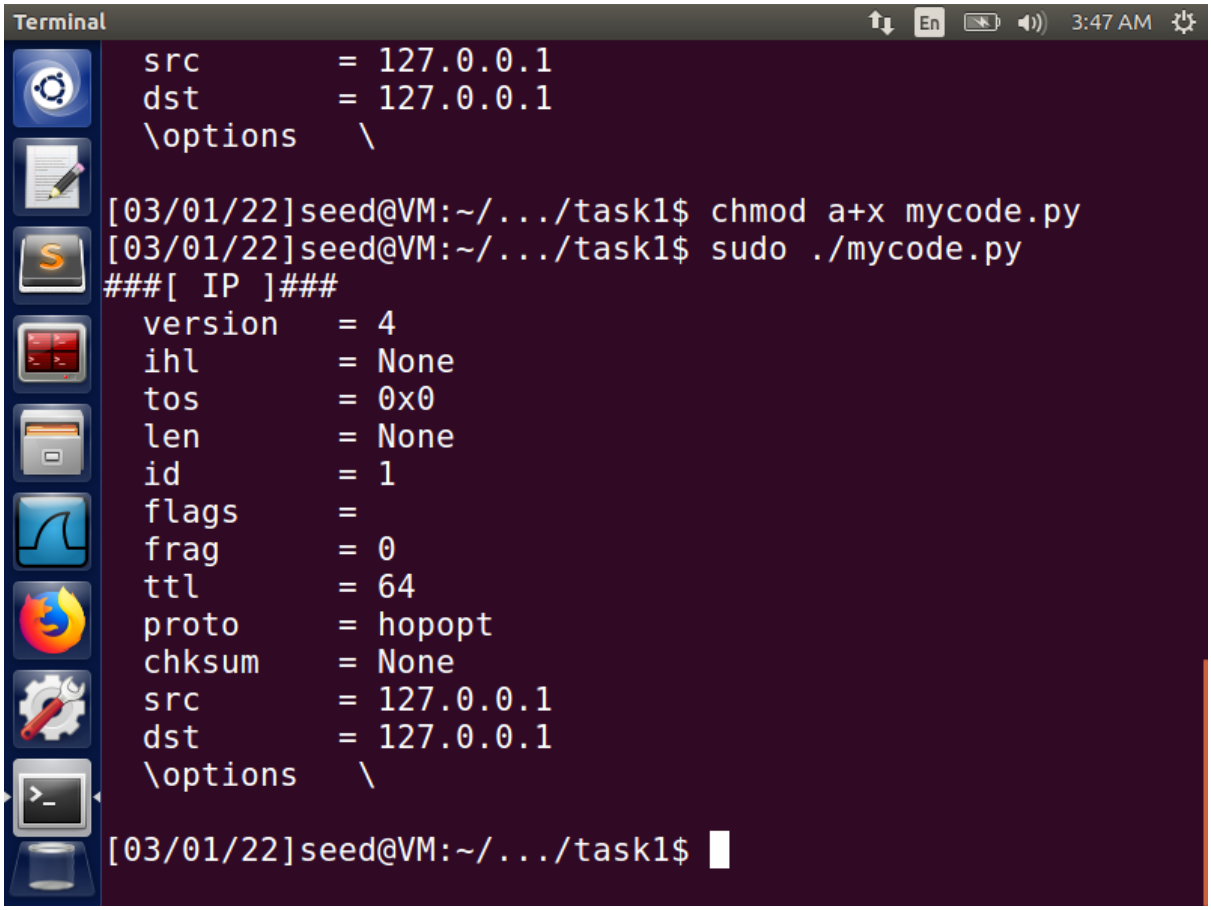

**Department of Computer Science**
**California State University, Channel Islands**

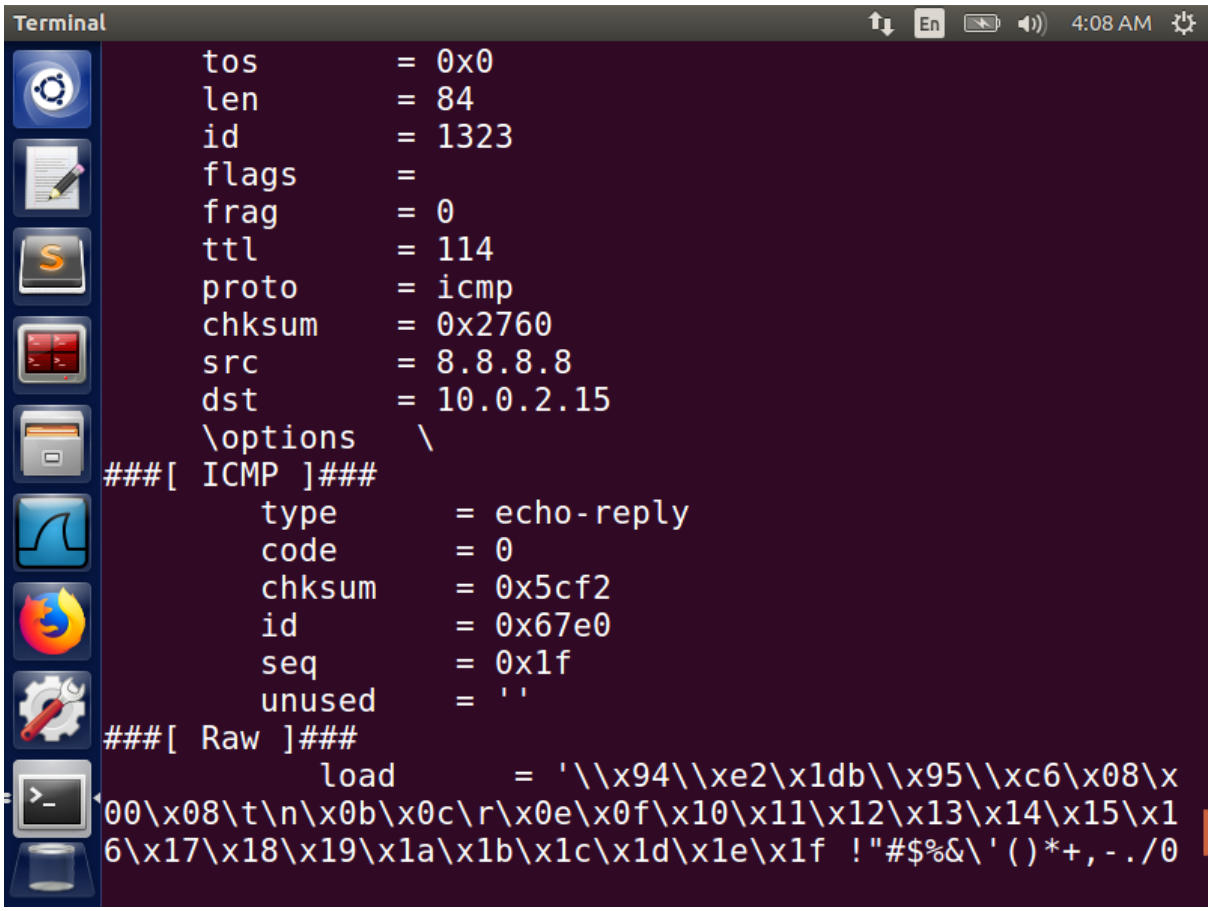
# COMP-524: Security
# Lab Report

Lab Number: 5
Lab Topic: Packet Sniffing and Spoofing Lab



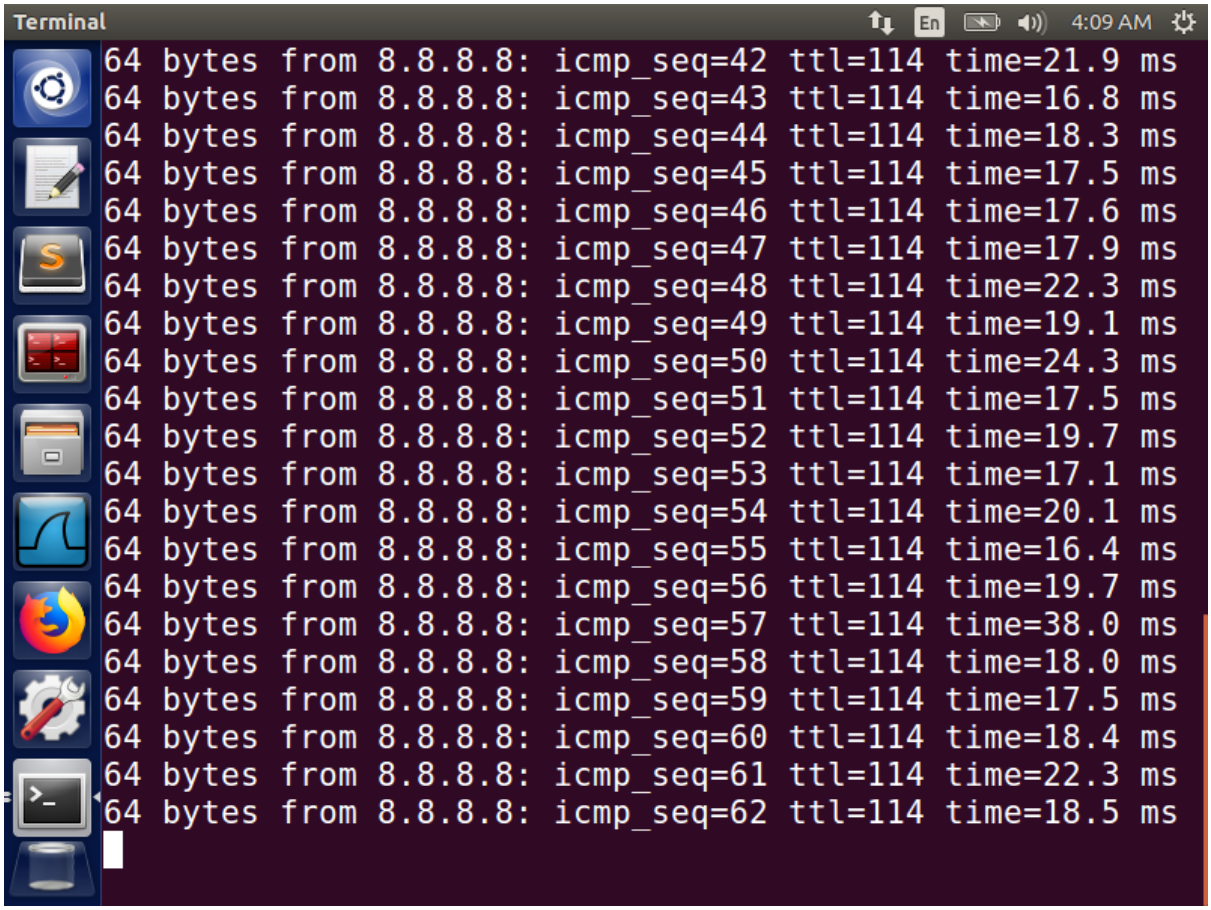

**When you run mycode.py, it displays some of the default settings. It includes the version number, checksum, as well as the source and destination IP addresses.**

## TASK1.1-A:



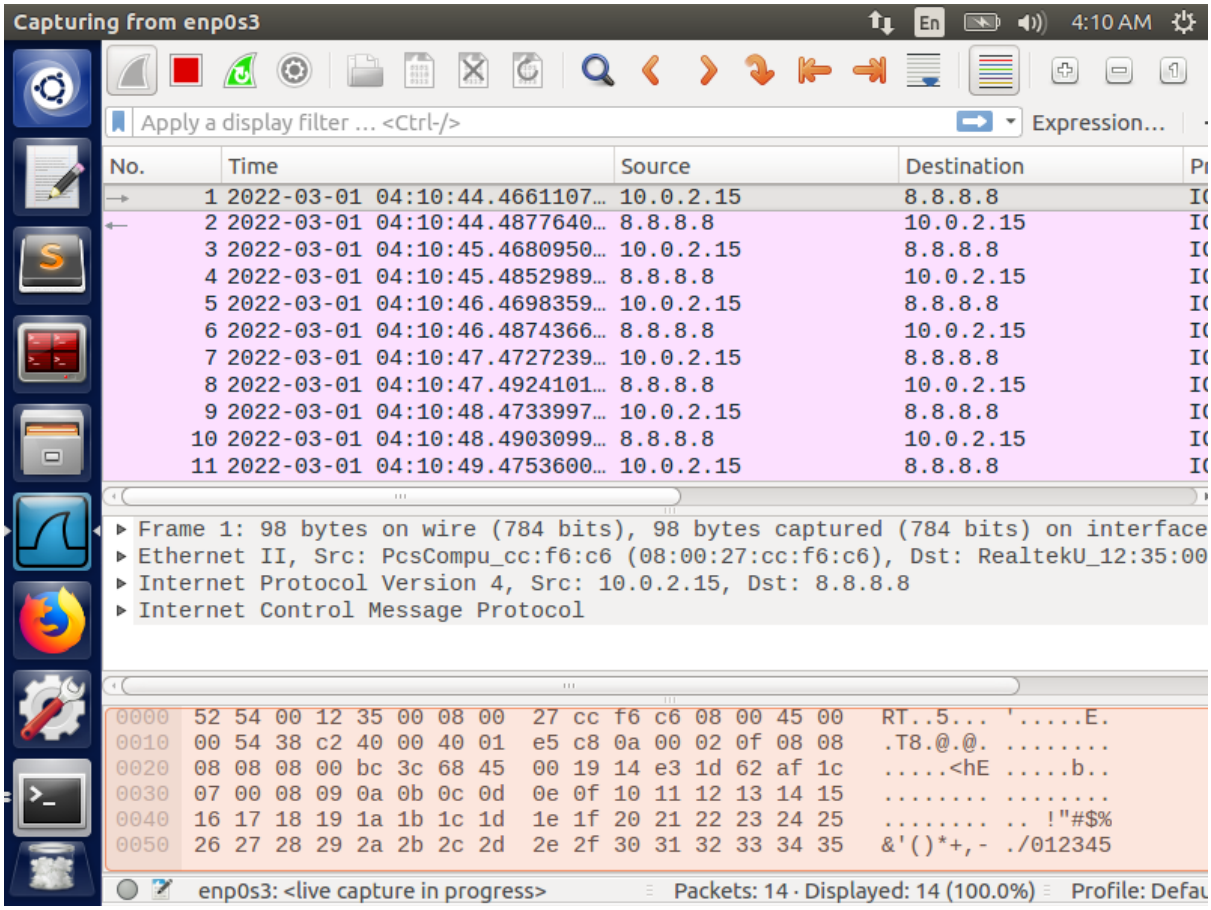

**The application is recording information about the source, destination, checksum, and other relevant stuff, as shown in the above snapshot. As a result, it indicates that it has received a response from Google to the destination IP address, which in my case is 10.0.2.15.**

**Ping 8.8.8.8 uses the ICMP protocol to ping the network.**

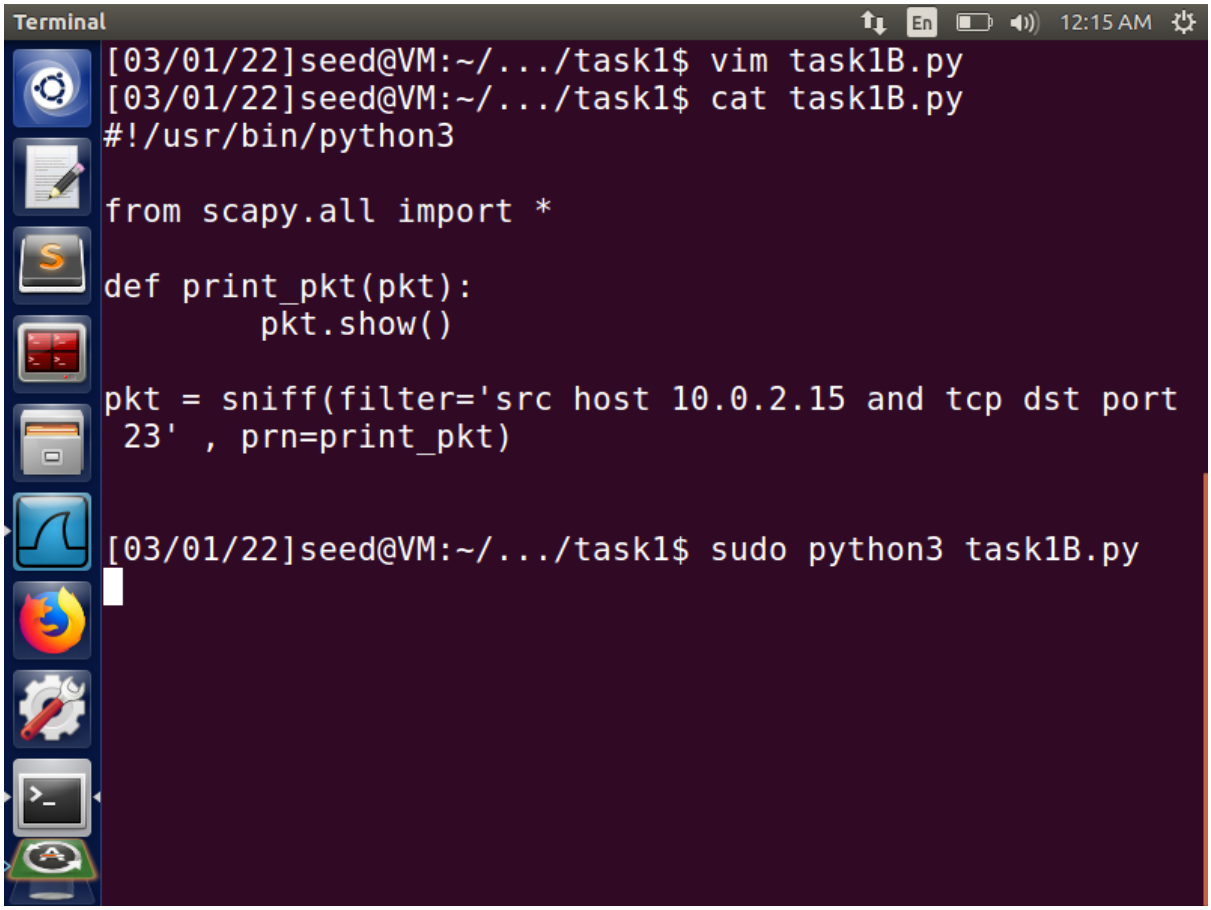



**Wireshark is a network capture and analysis program that displays the network's source, destination, and protocol.**

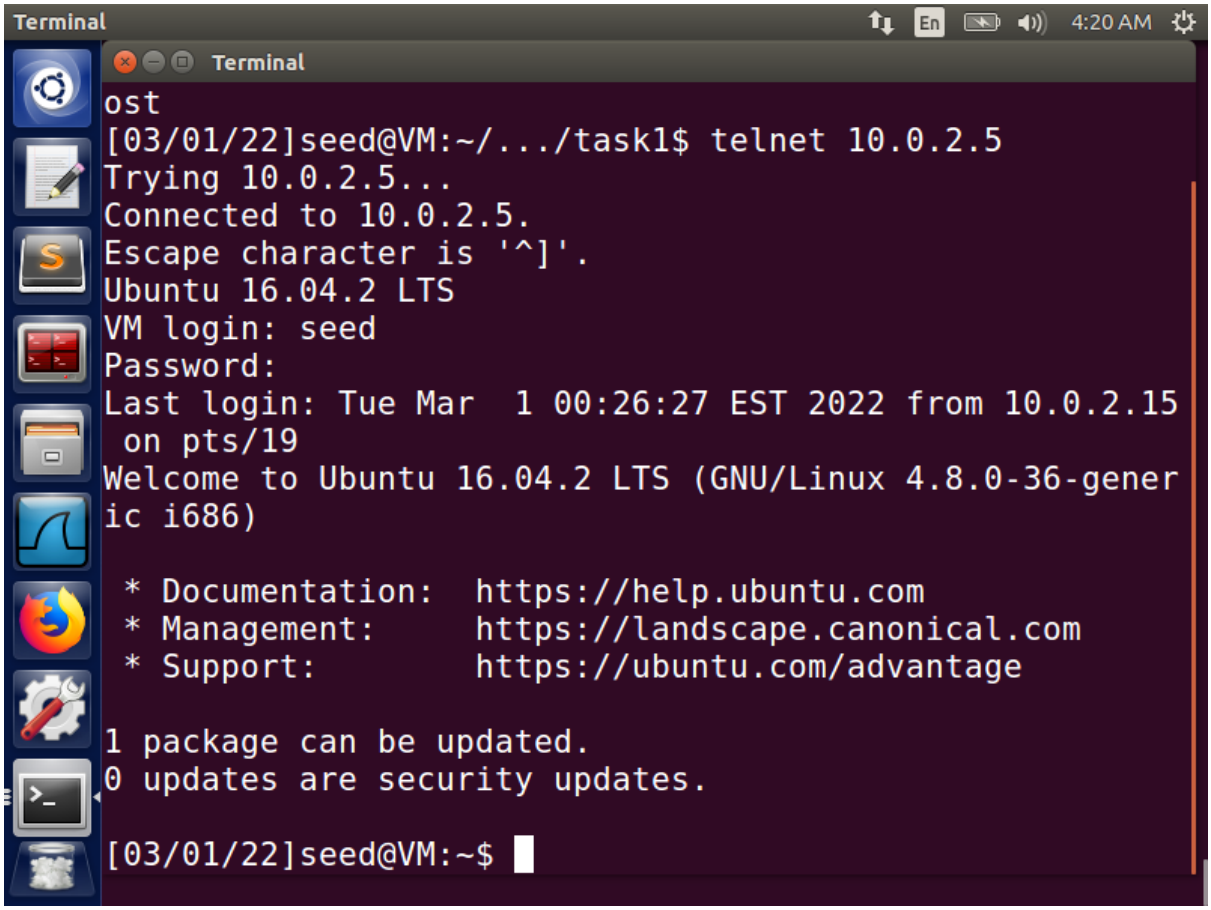
**Only the ICMP protocol source and destination analysis were captured using the function sniff and filter ICMP.**

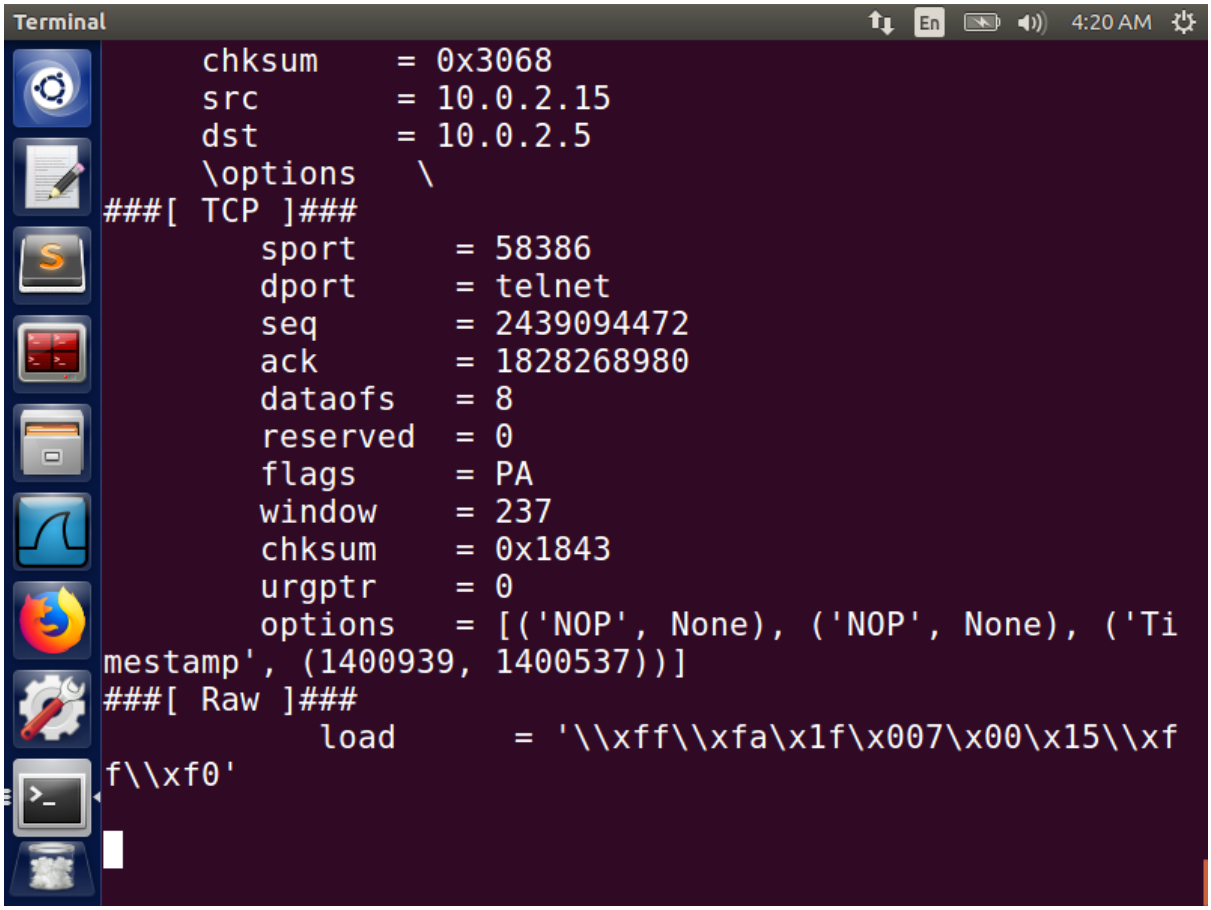

Capturing from enp0s3
En
4:10 AM
Apply a display filter ... <Ctrl-/>
Expression...
No. Time Source Destination
1 2022-03-01 04:10:44.4661107… 10.0.2.15 8.8.8.8
2 2022-03-01 04:10:44.4877640… 8.8.8.8 10.0.2.15
3 2022-03-01 04:10:45.4680950… 10.0.2.15 8.8.8.8
4 2022-03-01 04:10:45.4852989… 8.8.8.8 10.0.2.15
5 2022-03-01 04:10:46.4698359… 10.0.2.15 8.8.8.8
6 2022-03-01 04:10:46.4874366… 8.8.8.8 10.0.2.15
7 2022-03-01 04:10:47.4727239… 10.0.2.15 8.8.8.8
8 2022-03-01 04:10:47.4924101… 8.8.8.8 10.0.2.15
9 2022-03-01 04:10:48.4733997… 10.0.2.15 8.8.8.8
10 2022-03-01 04:10:48.4903099… 8.8.8.8 10.0.2.15
11 2022-03-01 04:10:49.4753600… 10.0.2.15 8.8.8.8
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 .....<hE .....b..
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:

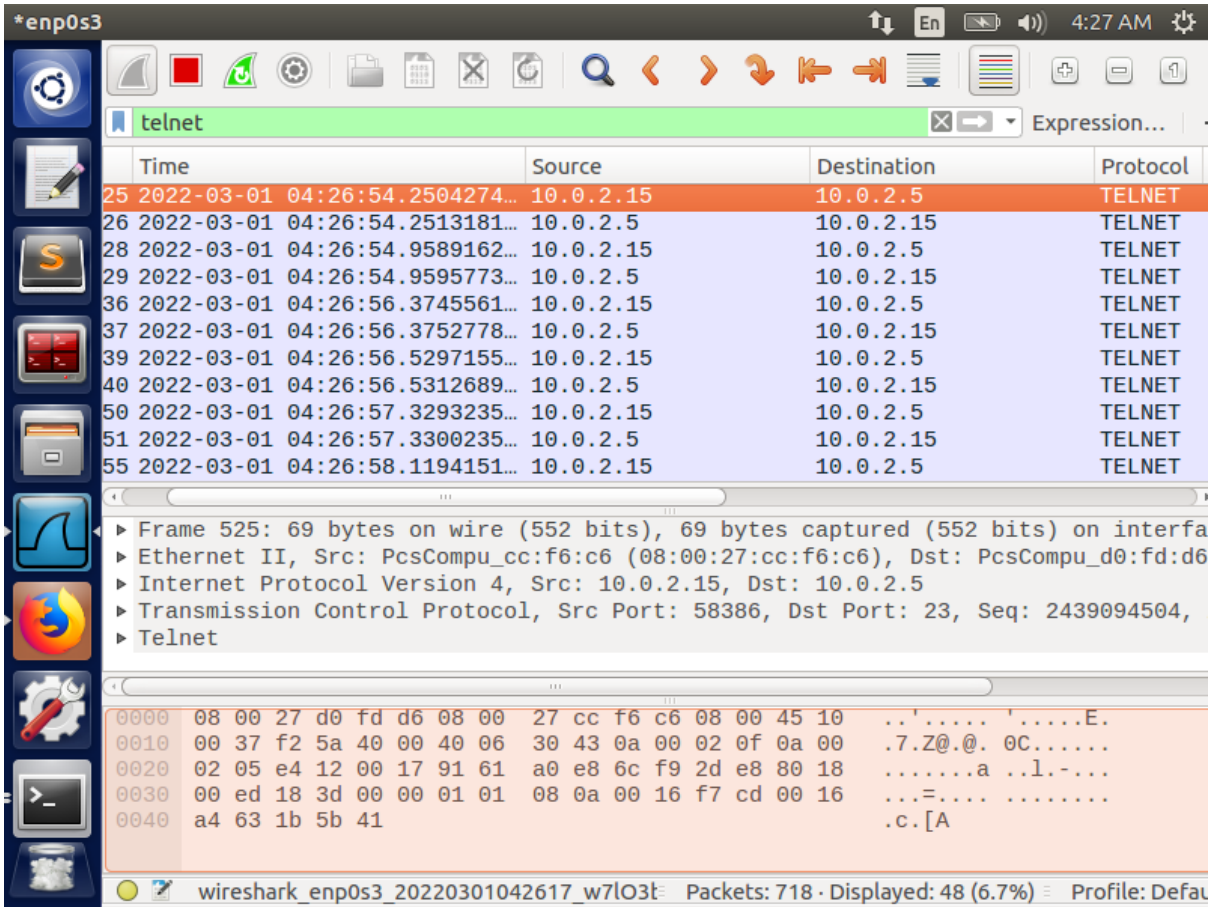


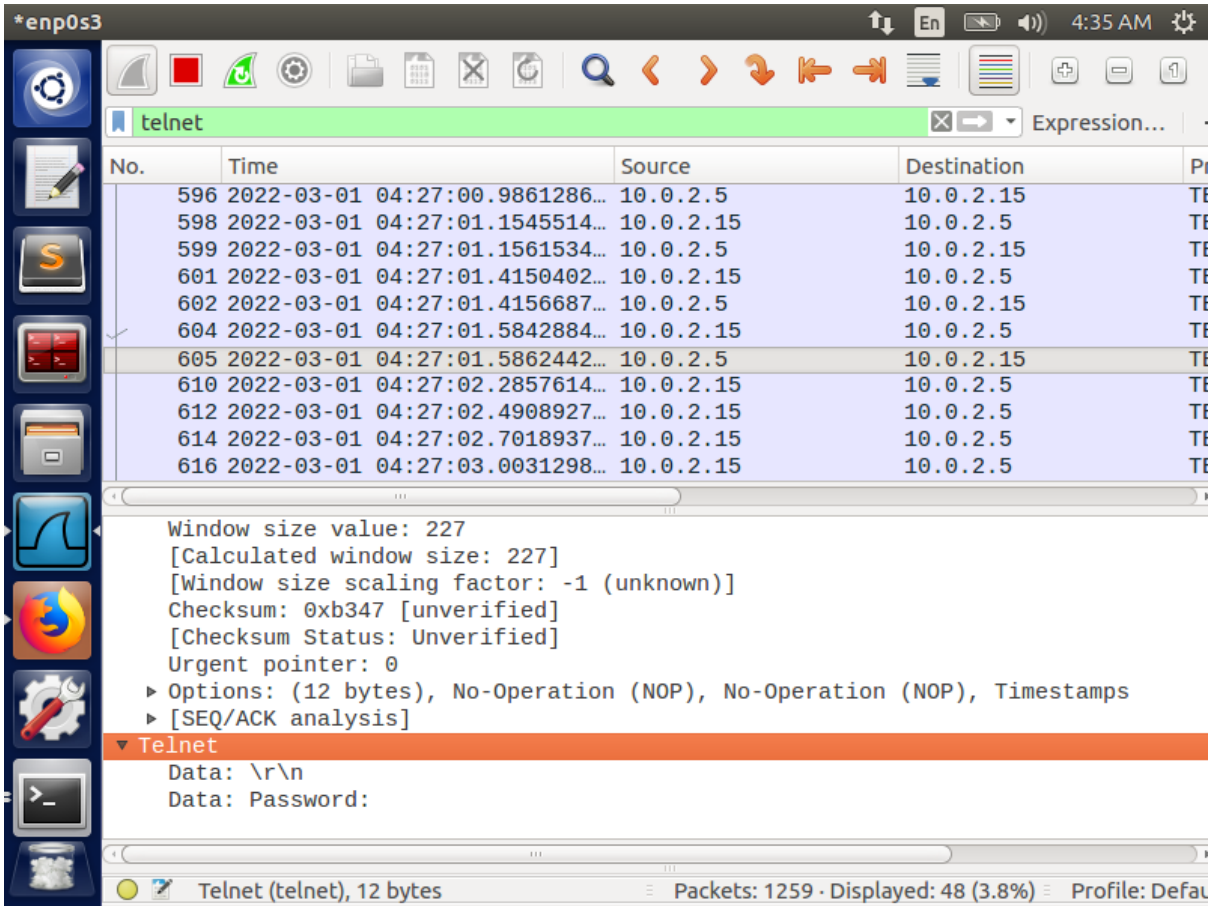

**Filtering is done with a specific host and TCP destination port 23 according to the given application.**


Terminal
En
4:20 AM
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
En
4:20 AM
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.**

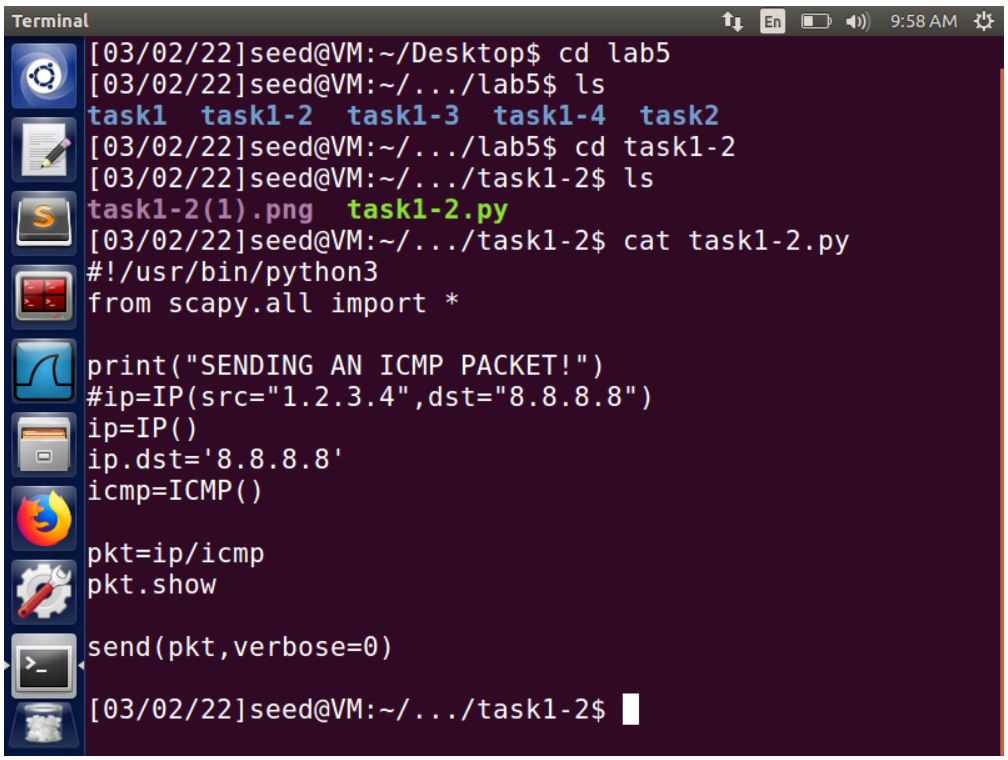


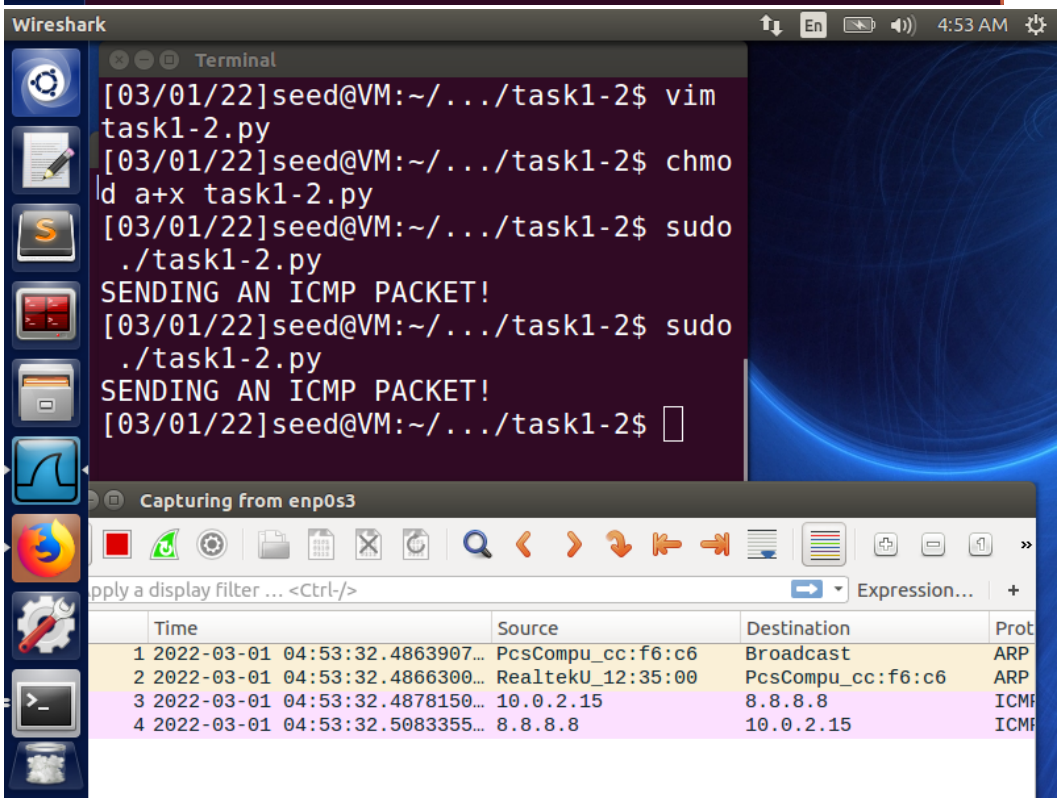



**As can be seen, telnet can be more risky. The password will be whatever follows the selection in the above image.**

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

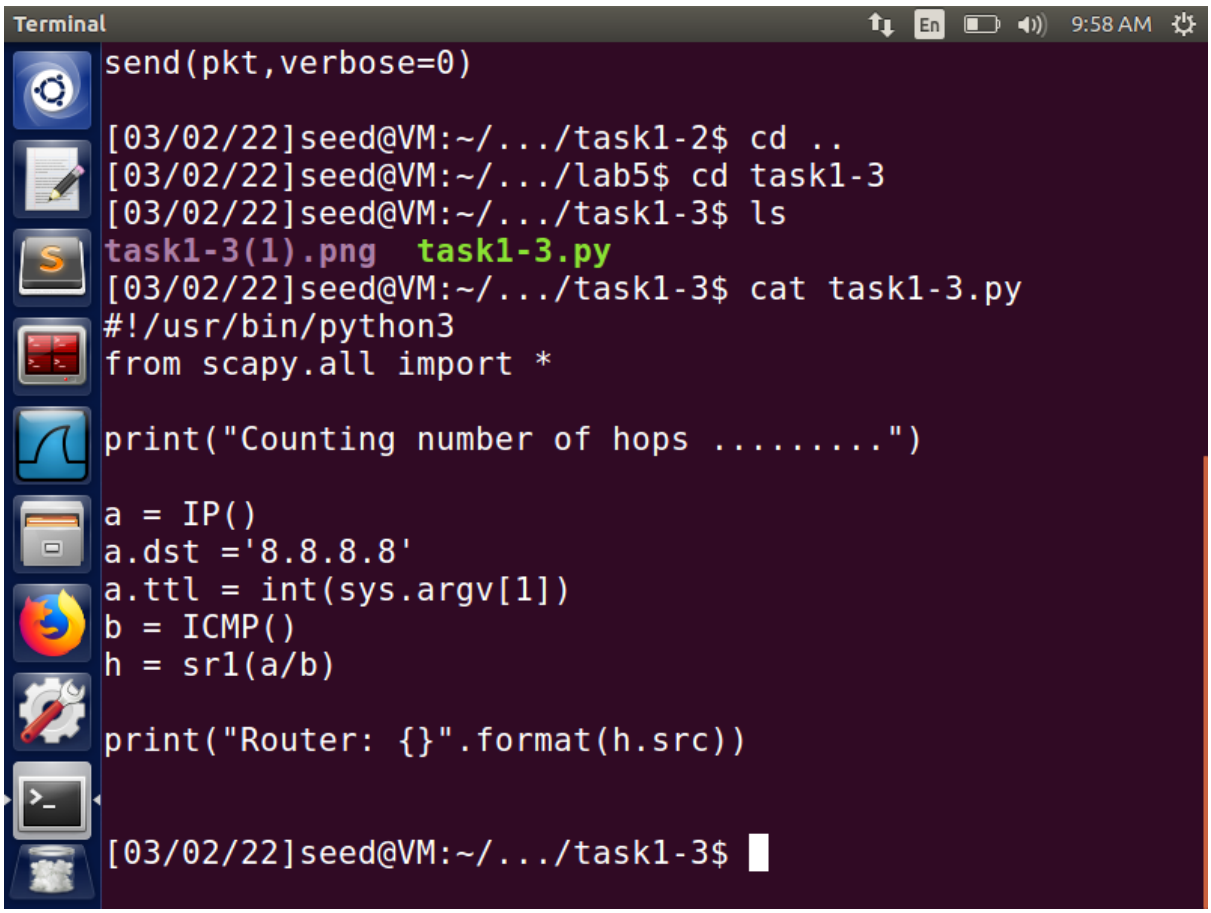
## TASK1.2:

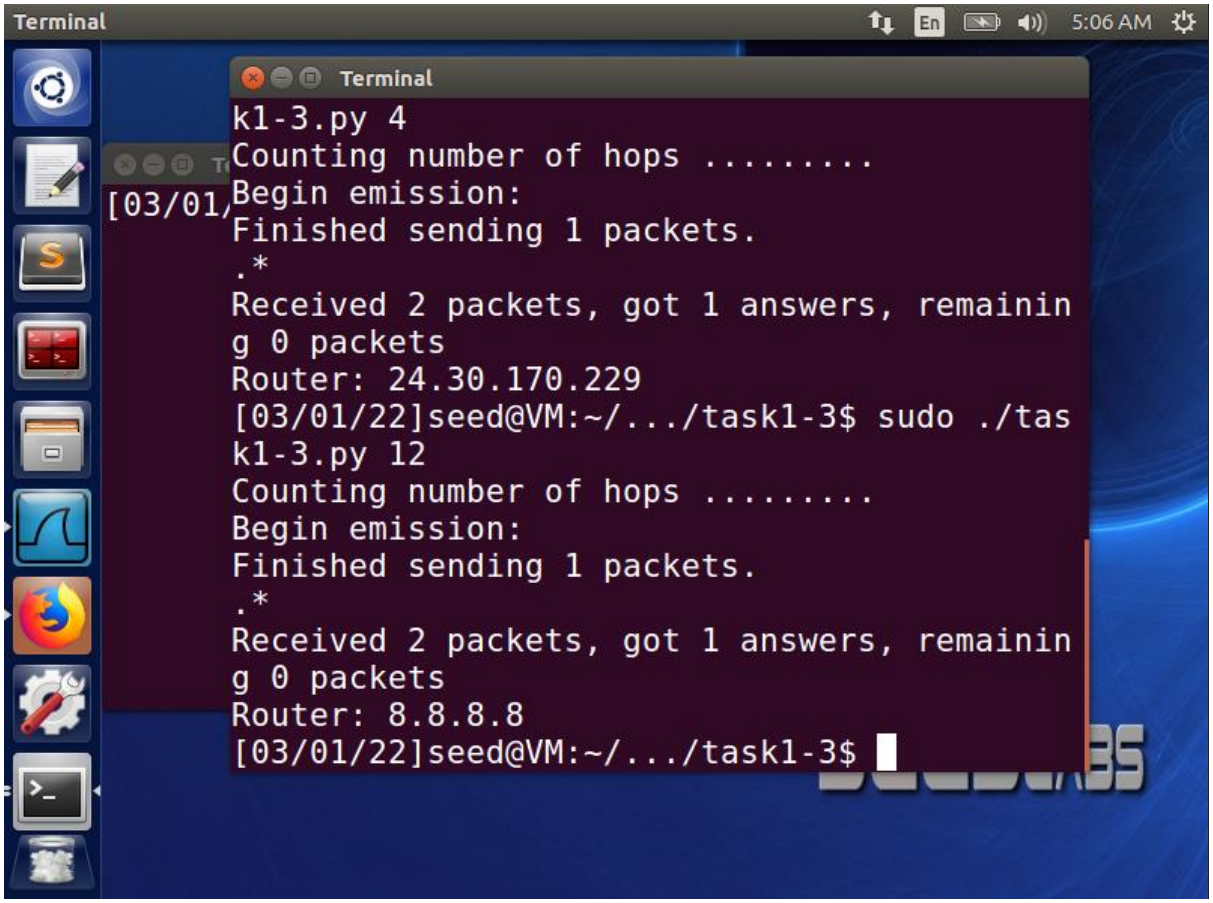


**As shown in the screenshot above, the software is sending ICMP packets. The ICMP packet is being captured by Wireshark.**

## TASK 1.3:

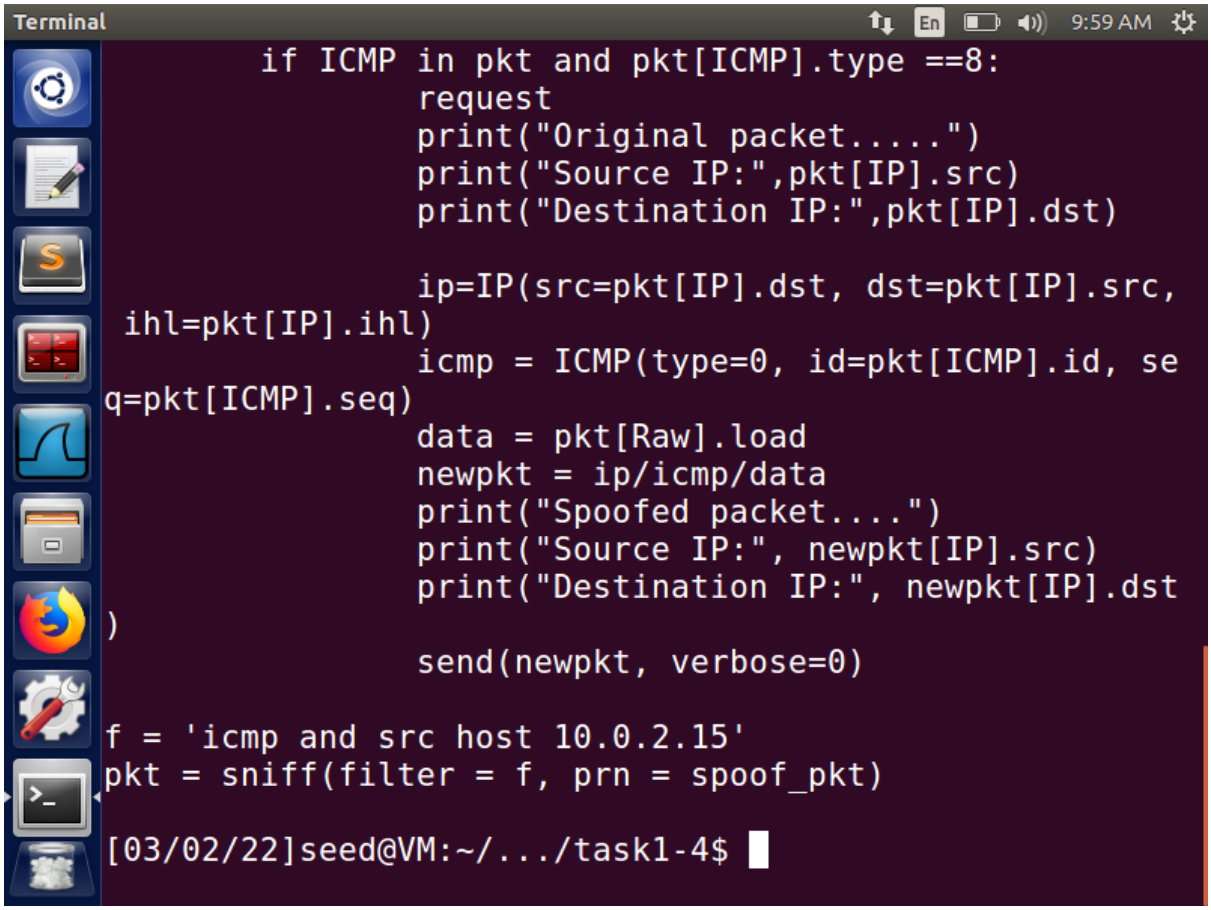


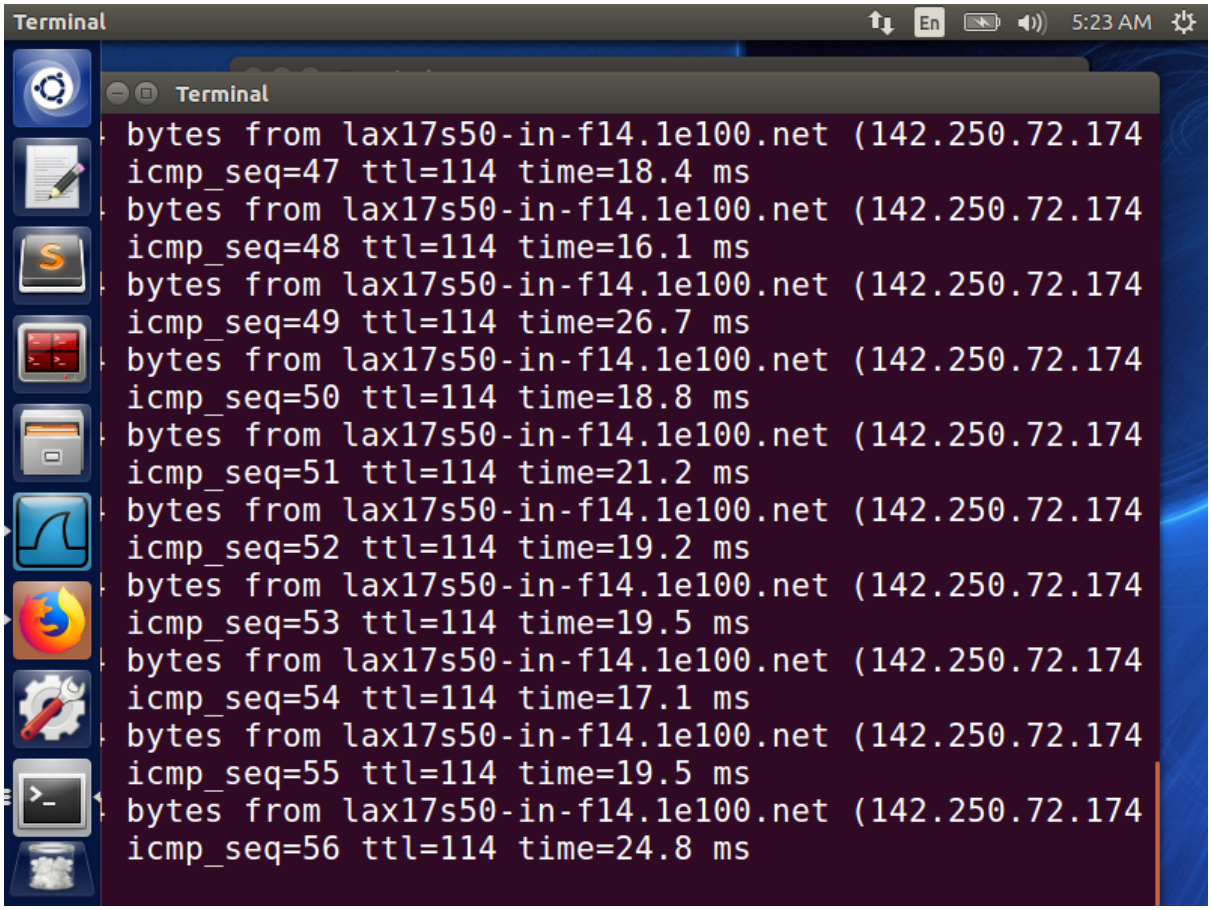



**The traceroute shows where the packet travels. The software counts how many hops or packets there are between my network and the specified destination.**

**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:

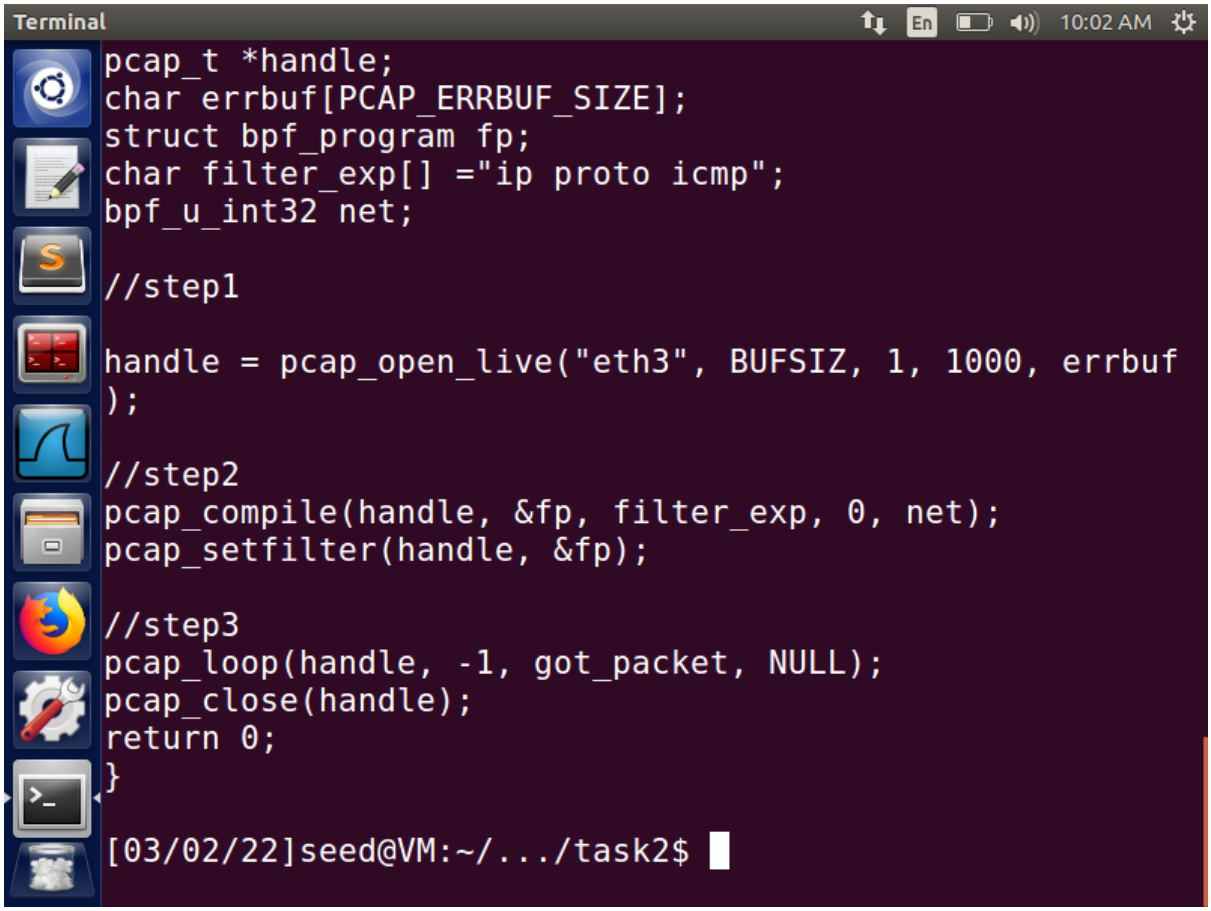

Terminal
En
9:59 AM
if ICMP in pkt and pkt[ICMP].type ==8:
request
print("Original packet.....")
print("Source IP:",pkt[IP].src)
print("Destination IP:",pkt[IP].dst)
ip=IP(src=pkt[IP].dst, dst=pkt[IP].src,
ihl=pkt[IP].ihl)
icmp = ICMP(type=0, id=pkt[ICMP].id, se
q=pkt[ICMP].seq)
data = pkt[Raw].load
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$

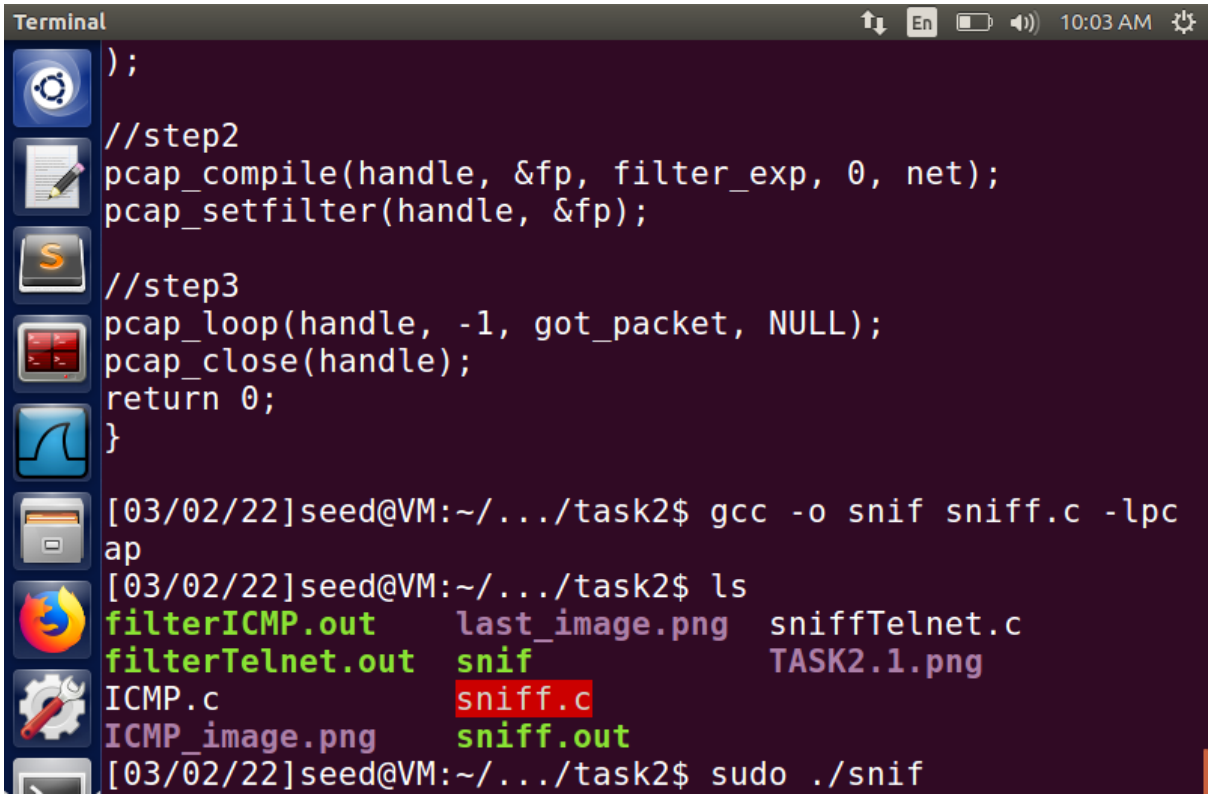



**This program is only interested in capturing the ICMP packet(Which comes from the specific host) with echo requests. The task is to capture those and impersonate them. As the screenshot says it is impersonating the google server.**

## Task 2(2.1):

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





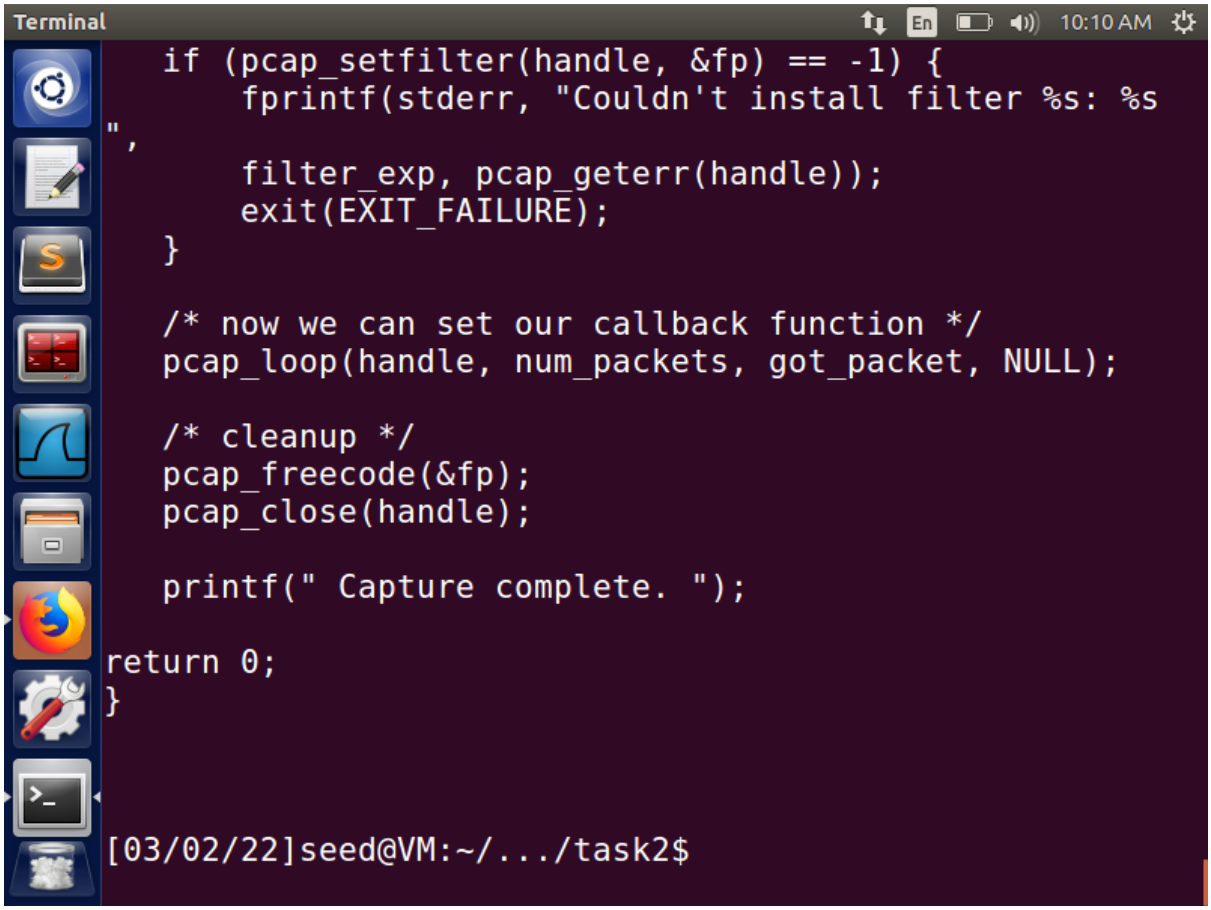

**I have written a program that print out the source and destination IP addresses**

**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.**




## 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 on 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 access 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.**

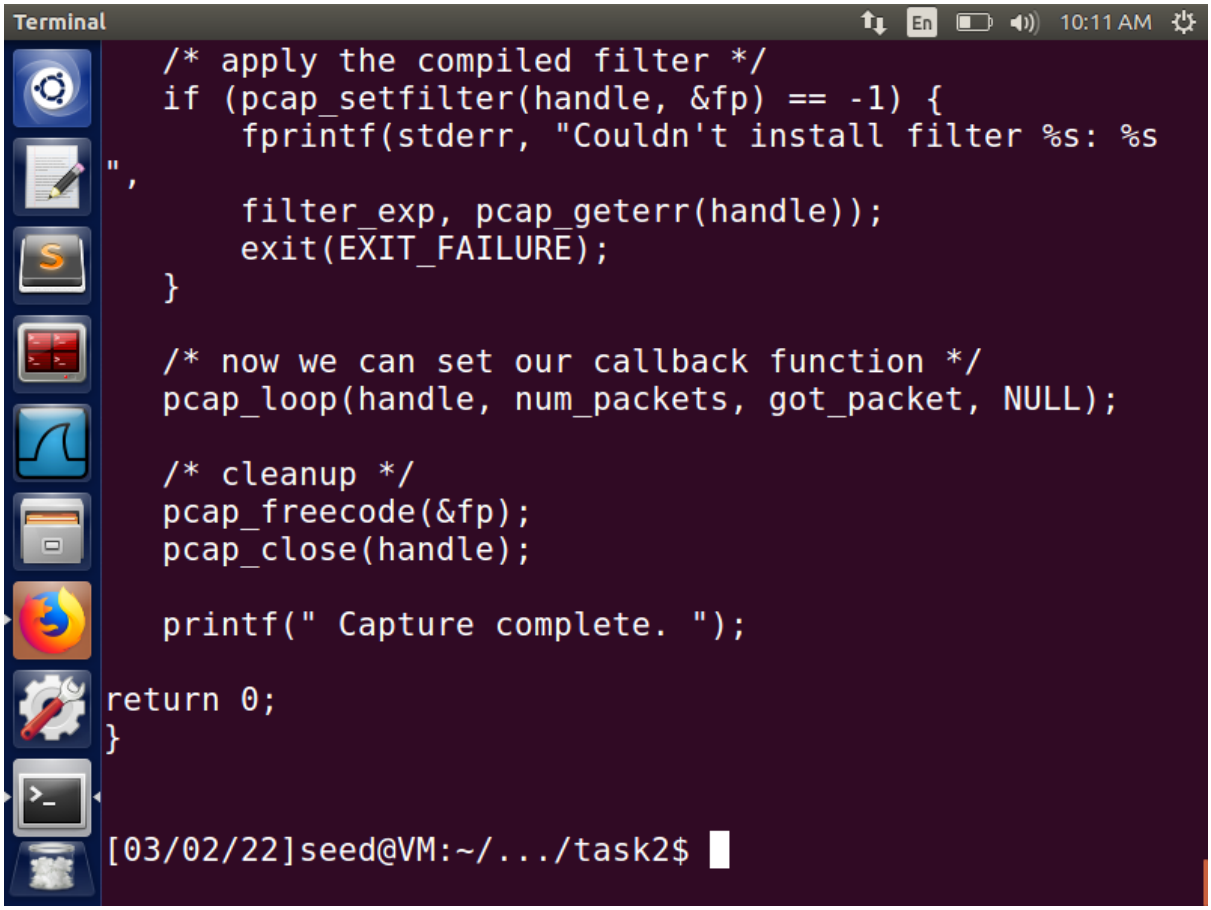
**Q3 ifconfig [interface] promisc ip link set [interface] promisc on ip a show eth1 | grep -i promisc**
**There is a difference between information sniffed in nonpromiscuous and promiscuous**

**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**




**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 intopcap and syntax was used from tcpdump website. Note the password dees becomes clear as packets are received.**