

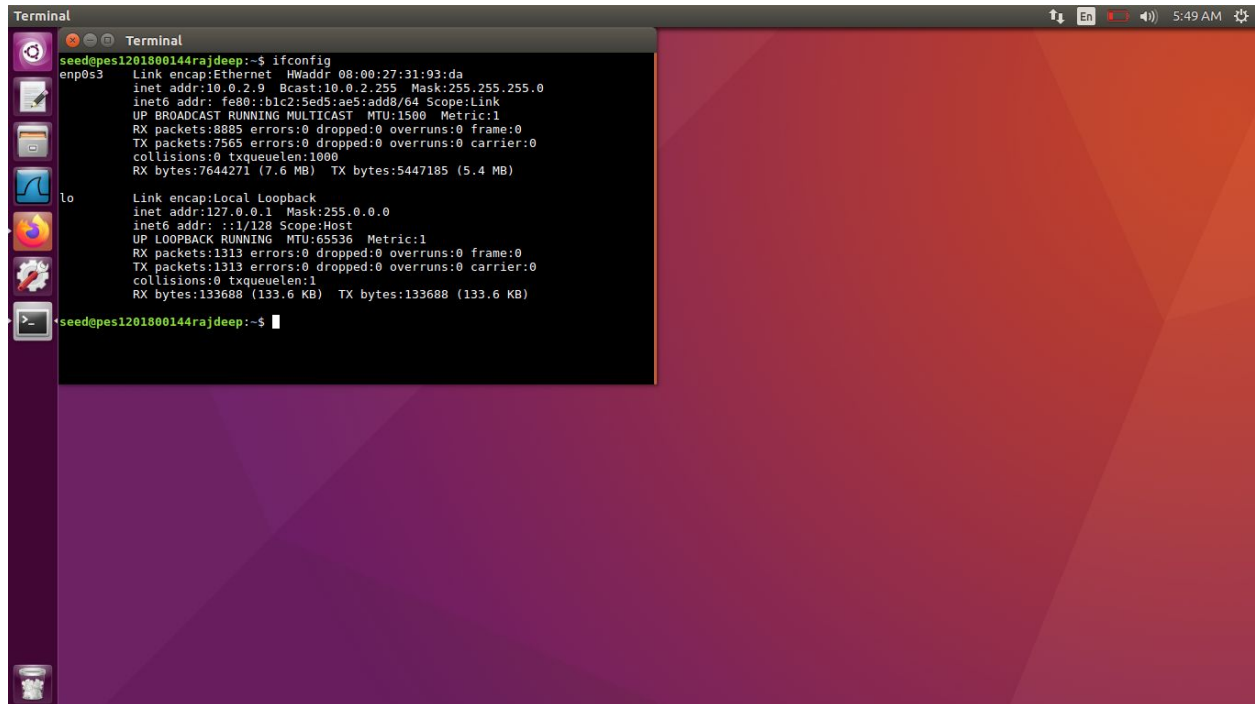
COMPUTER NETWORKS SECURITY
LABORATORY
ASSIGNMENT 2
BY: RAJDEEP SENGUPTA
SRN: PES1201800144
SECTION: C

NOTE: Please find my SRN 'PES1201800144rajdeep' as the terminal username.

MY CONFIGURATIONS:

VM1 (Attacker)

IP Address: 10.0.2.9



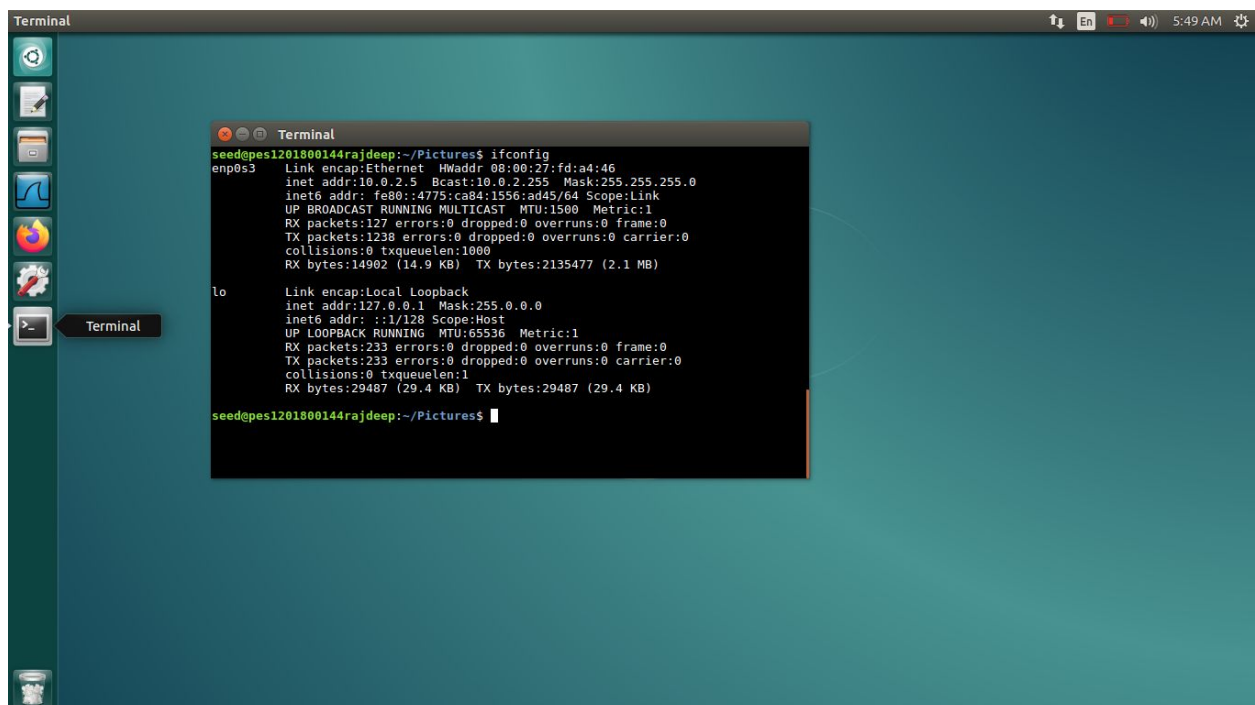
```
seed@pes1201800144rajdeep:~$ ifconfig
enp0s3  Link encap:Ethernet  HWaddr 08:00:27:31:93:da
        inet addr:10.0.2.9  Bcast:10.0.2.255  Mask:255.255.255.0
        inet6 addr: fe80::b1c2:5ed5:ae5:ad8/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:8885 errors:0 dropped:0 overruns:0 frame:0
        TX packets:7565 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:7644271 (7.6 MB)  TX bytes:5447185 (5.4 MB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:1313 errors:0 dropped:0 overruns:0 frame:0
        TX packets:1313 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:133688 (133.6 KB)  TX bytes:133688 (133.6 KB)

seed@pes1201800144rajdeep:~$
```

VM2 (Victim)

IP Address: 10.0.2.5

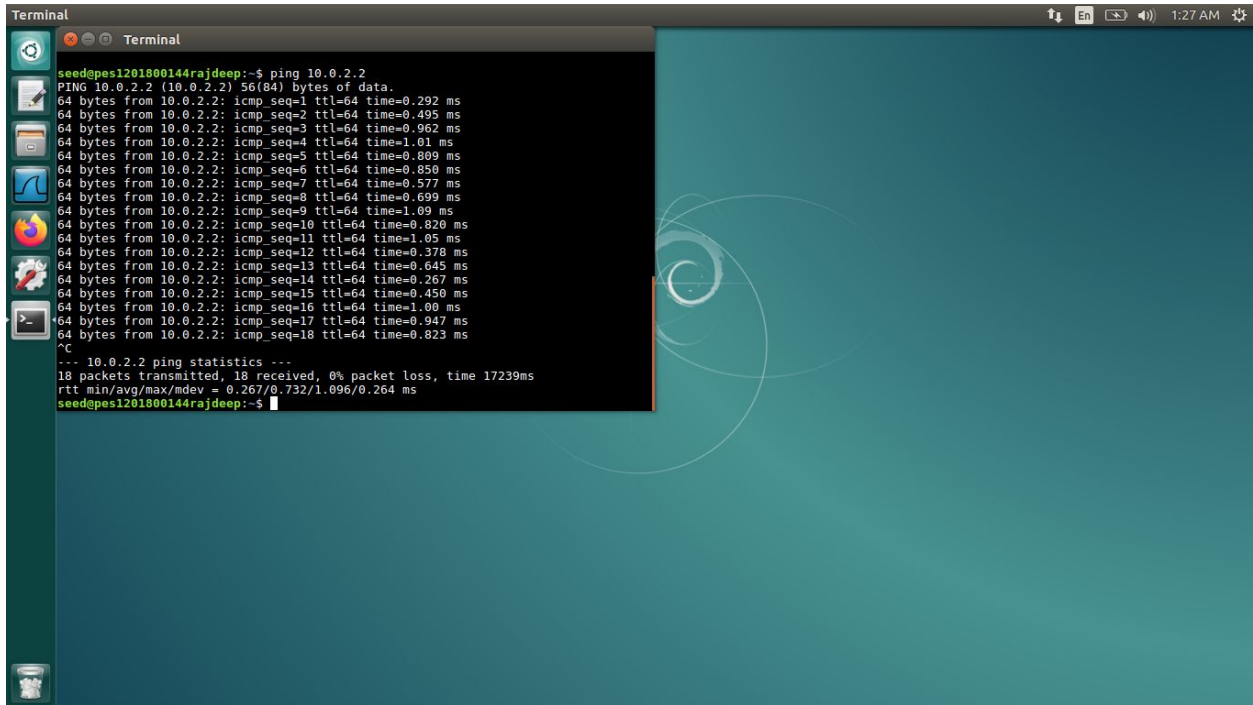


```
seed@pes1201800144rajdeep:~/Pictures$ ifconfig
enp0s3  Link encap:Ethernet  HWaddr 08:00:27:fd:a4:46
        inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.0
        inet6 addr: fe80::4775:ca84:1856:ad45/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:127 errors:0 dropped:0 overruns:0 frame:0
        TX packets:1238 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:14902 (14.9 KB)  TX bytes:2135477 (2.1 MB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:233 errors:0 dropped:0 overruns:0 frame:0
        TX packets:233 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:29487 (29.4 KB)  TX bytes:29487 (29.4 KB)

seed@pes1201800144rajdeep:~/Pictures$
```

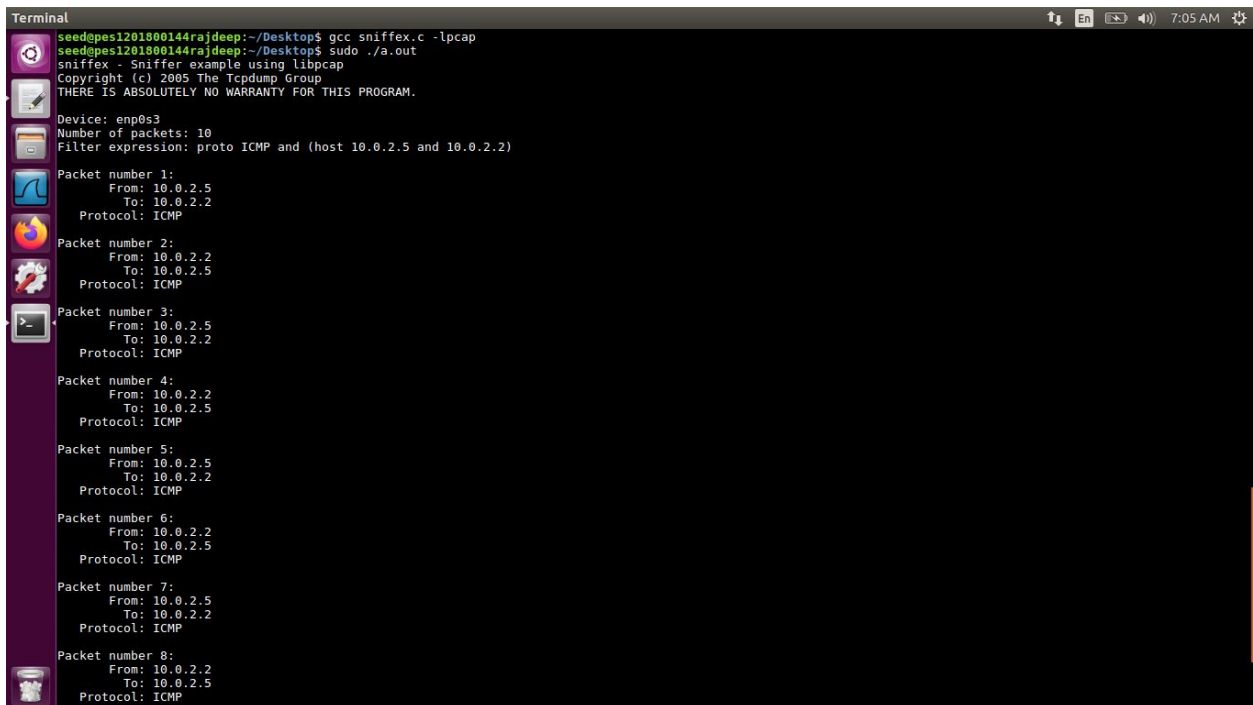
TASK 1.1A



A terminal window on a Linux desktop with a teal background. The terminal shows a user running a ping command to 10.0.2.2. The output displays 18 successful ping requests with varying response times. At the bottom, it shows statistics: 18 packets transmitted, 18 received, 0% packet loss, and a total time of 17239ms. The user's prompt is 'seed@pes1201800144rajdeep:~\$'.

```
Terminal
seed@pes1201800144rajdeep:~$ ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data:
64 bytes from 10.0.2.2: icmp_seq=1 ttl=64 time=0.292 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=64 time=0.495 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=64 time=0.962 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=64 time=1.01 ms
64 bytes from 10.0.2.2: icmp_seq=5 ttl=64 time=0.809 ms
64 bytes from 10.0.2.2: icmp_seq=6 ttl=64 time=0.850 ms
64 bytes from 10.0.2.2: icmp_seq=7 ttl=64 time=0.577 ms
64 bytes from 10.0.2.2: icmp_seq=8 ttl=64 time=0.699 ms
64 bytes from 10.0.2.2: icmp_seq=9 ttl=64 time=1.09 ms
64 bytes from 10.0.2.2: icmp_seq=10 ttl=64 time=0.820 ms
64 bytes from 10.0.2.2: icmp_seq=11 ttl=64 time=1.05 ms
64 bytes from 10.0.2.2: icmp_seq=12 ttl=64 time=0.378 ms
64 bytes from 10.0.2.2: icmp_seq=13 ttl=64 time=0.645 ms
64 bytes from 10.0.2.2: icmp_seq=14 ttl=64 time=0.267 ms
64 bytes from 10.0.2.2: icmp_seq=15 ttl=64 time=0.450 ms
64 bytes from 10.0.2.2: icmp_seq=16 ttl=64 time=1.00 ms
64 bytes from 10.0.2.2: icmp_seq=17 ttl=64 time=0.947 ms
64 bytes from 10.0.2.2: icmp_seq=18 ttl=64 time=0.823 ms
^C
--- 10.0.2.2 ping statistics ---
18 packets transmitted, 18 received, 0% packet loss, time 17239ms
rtt min/avg/max/mdev = 0.267/0.732/1.096/0.264 ms
seed@pes1201800144rajdeep:~$
```

Victim machine(10.0.2.5) pinging to 10.0.2.2



A terminal window showing the execution of a sniffing tool named 'sniffex'. The user runs 'gcc sniffex.c -lpcap' and 'sudo ./a.out'. The tool displays its configuration: device 'enp0s3', 10 packets, and a filter expression 'proto ICMP and (host 10.0.2.5 and 10.0.2.2)'. It then lists 8 captured packets, all of which are ICMP requests from 10.0.2.5 to 10.0.2.2. The user's prompt is 'seed@pes1201800144rajdeep:~/Desktop\$'.

```
Terminal
seed@pes1201800144rajdeep:~/Desktop$ gcc sniffex.c -lpcap
seed@pes1201800144rajdeep:~/Desktop$ sudo ./a.out
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Topdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: enp0s3
Number of packets: 10
Filter expression: proto ICMP and (host 10.0.2.5 and 10.0.2.2)

Packet number 1:
  From: 10.0.2.5
  To: 10.0.2.2
  Protocol: ICMP

Packet number 2:
  From: 10.0.2.2
  To: 10.0.2.5
  Protocol: ICMP

Packet number 3:
  From: 10.0.2.5
  To: 10.0.2.2
  Protocol: ICMP

Packet number 4:
  From: 10.0.2.2
  To: 10.0.2.5
  Protocol: ICMP

Packet number 5:
  From: 10.0.2.5
  To: 10.0.2.2
  Protocol: ICMP

Packet number 6:
  From: 10.0.2.2
  To: 10.0.2.5
  Protocol: ICMP

Packet number 7:
  From: 10.0.2.5
  To: 10.0.2.2
  Protocol: ICMP

Packet number 8:
  From: 10.0.2.2
  To: 10.0.2.5
  Protocol: ICMP

seed@pes1201800144rajdeep:~/Desktop$
```

Attacker machine(10.0.2.9) sniffing the packets from victim machine(10.0.2.5) to a host(10.0.2.2)

Ques 1. Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not detailed explanation like the one in the tutorial.

Sol 1. The library calls made and their uses are:

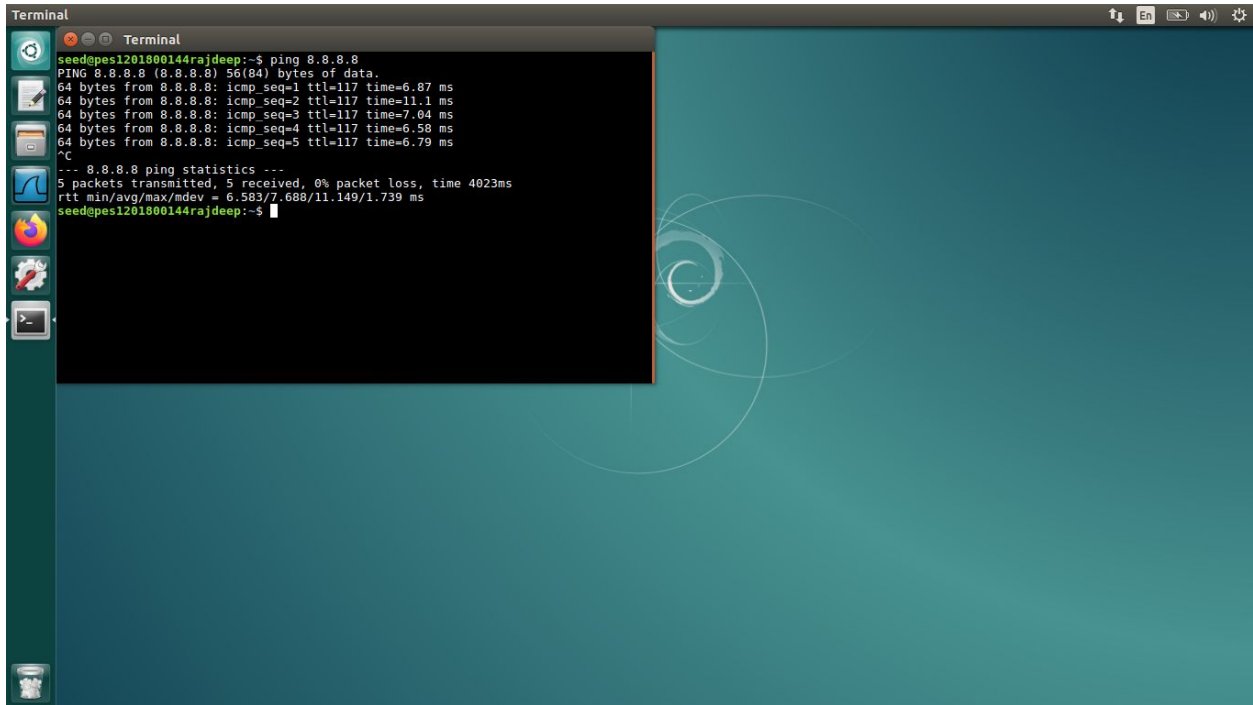
- **pcap - provides packet capture and filtering engines**
- **pcap_open_live() - start session on a device in the network to initiate sniffing**
- **pcap_compile() - compile filter expression stored in a string**
- **pcap_setfilter() - set the compiled filter to filter the packets to be sniffed**
- **pcap_loop() - to make the program wait for multiple packets for being sniffed**
- **pcap_close() - close the sniffing session**

Ques 2. Why do you need the root privilege to run sniffex? Where does the program fail if executed without the root privilege?

Sol 2. When sniffex code is executed without root privilege, the terminal responds with "Permission denied". In this code, pcap library is used which tries to access the NIC(Network Interface Card) to sniff packets within the network even if the packets are not meant for the attacker machine(promiscuous mode).

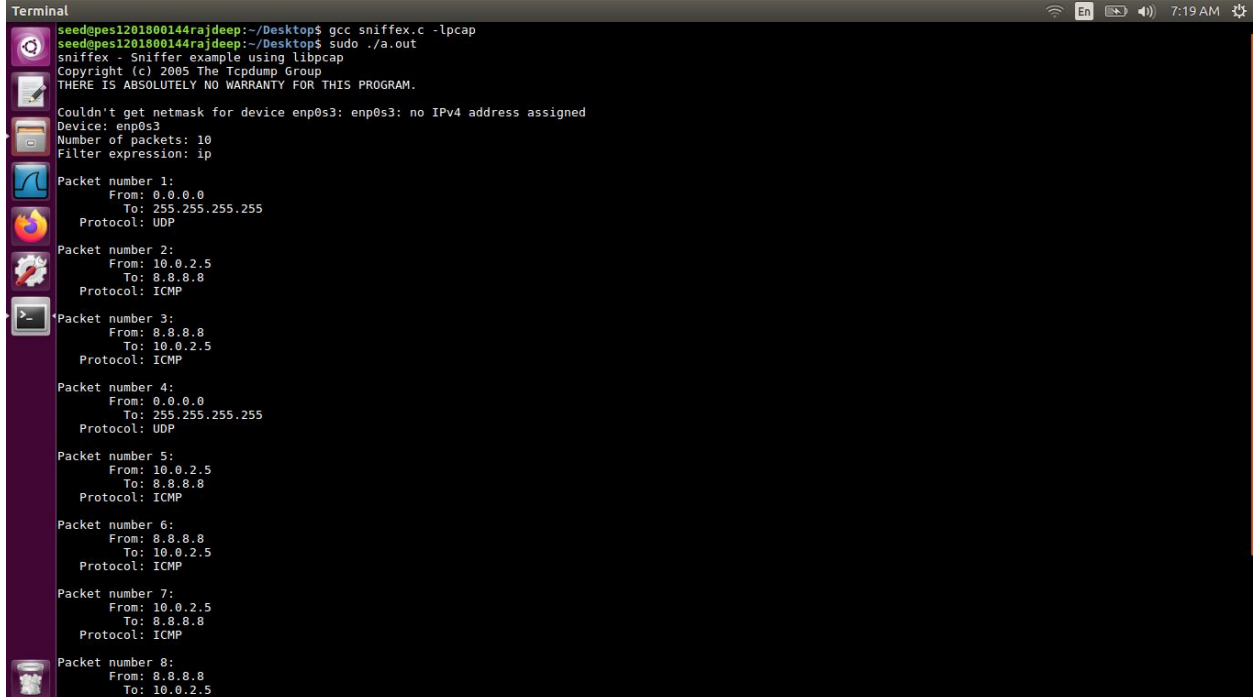
Ques 3> Promiscuous Mode On/Off Demonstration:

Promiscuous Mode On:



```
Terminal
seed@pes1201800144rajdeep:~$ 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=117 time=6.87 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=117 time=11.1 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=117 time=7.04 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=117 time=6.58 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=117 time=6.79 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4023ms
rtt min/avg/max/mdev = 6.583/7.688/11.149/1.739 ms
seed@pes1201800144rajdeep:~$
```

Victim machine(10.0.2.5) ping to 8.8.8.8



```
Terminal
seed@pes1201800144rajdeep:~/Desktop$ gcc sniffex.c -lpcap
seed@pes1201800144rajdeep:~/Desktop$ sudo ./a.out
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Couldn't get netmask for device enp0s3: enp0s3: no IPv4 address assigned
Device: enp0s3
Number of packets: 10
Filter expression: ip

Packet number 1:
  From: 0.0.0.0
  To: 255.255.255.255
  Protocol: UDP

Packet number 2:
  From: 10.0.2.5
  To: 8.8.8.8
  Protocol: ICMP

Packet number 3:
  From: 8.8.8.8
  To: 10.0.2.5
  Protocol: ICMP

Packet number 4:
  From: 0.0.0.0
  To: 255.255.255.255
  Protocol: UDP

Packet number 5:
  From: 10.0.2.5
  To: 8.8.8.8
  Protocol: ICMP

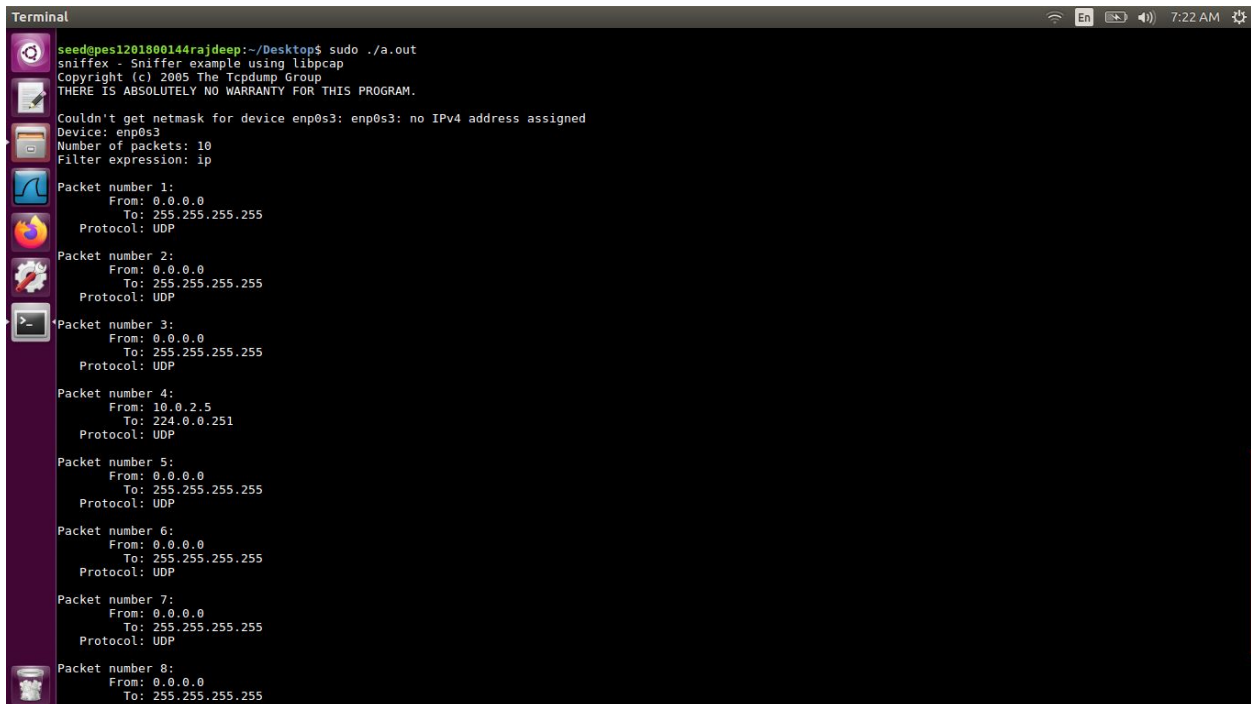
Packet number 6:
  From: 8.8.8.8
  To: 10.0.2.5
  Protocol: ICMP

Packet number 7:
  From: 10.0.2.5
  To: 8.8.8.8
  Protocol: ICMP

Packet number 8:
  From: 8.8.8.8
  To: 10.0.2.5
```

Attacker Machine(10.0.2.9) sniffing ICMP ping packets from victim machine(10.0.2.5) to Google server(8.8.8.8)

Promiscuous Mode Off:



```
Terminal
seedpes1201800144rajdeep:~/Desktop$ sudo ./a.out
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Couldn't get netmask for device enp0s3: enp0s3: no IPv4 address assigned
Device: enp0s3
Number of packets: 10
Filter expression: ip

Packet number 1:
  From: 0.0.0.0
  To: 255.255.255.255
  Protocol: UDP

Packet number 2:
  From: 0.0.0.0
  To: 255.255.255.255
  Protocol: UDP

Packet number 3:
  From: 0.0.0.0
  To: 255.255.255.255
  Protocol: UDP

Packet number 4:
  From: 10.0.2.5
  To: 224.0.0.251
  Protocol: UDP

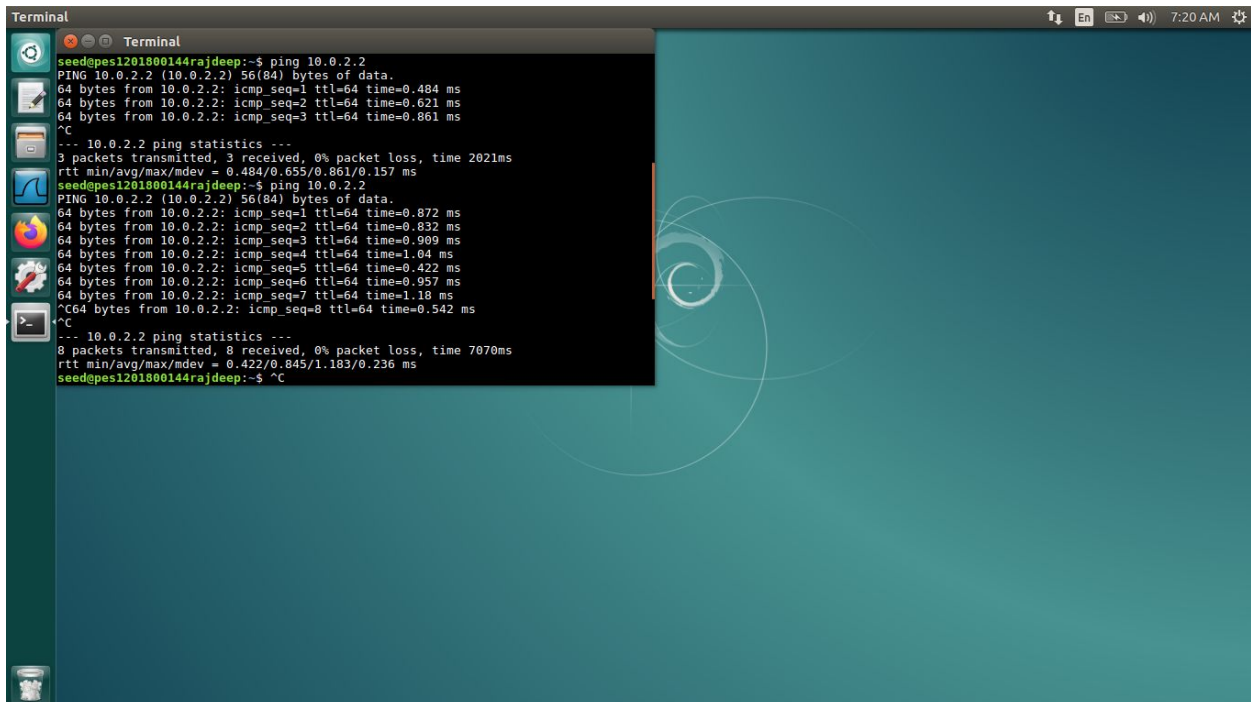
Packet number 5:
  From: 0.0.0.0
  To: 255.255.255.255
  Protocol: UDP

Packet number 6:
  From: 0.0.0.0
  To: 255.255.255.255
  Protocol: UDP

Packet number 7:
  From: 0.0.0.0
  To: 255.255.255.255
  Protocol: UDP

Packet number 8:
  From: 0.0.0.0
  To: 255.255.255.255
  Protocol: UDP
```

Attacker machine (10.0.2.9) cannot sniff packets that don't involve its IP address in source or destination.



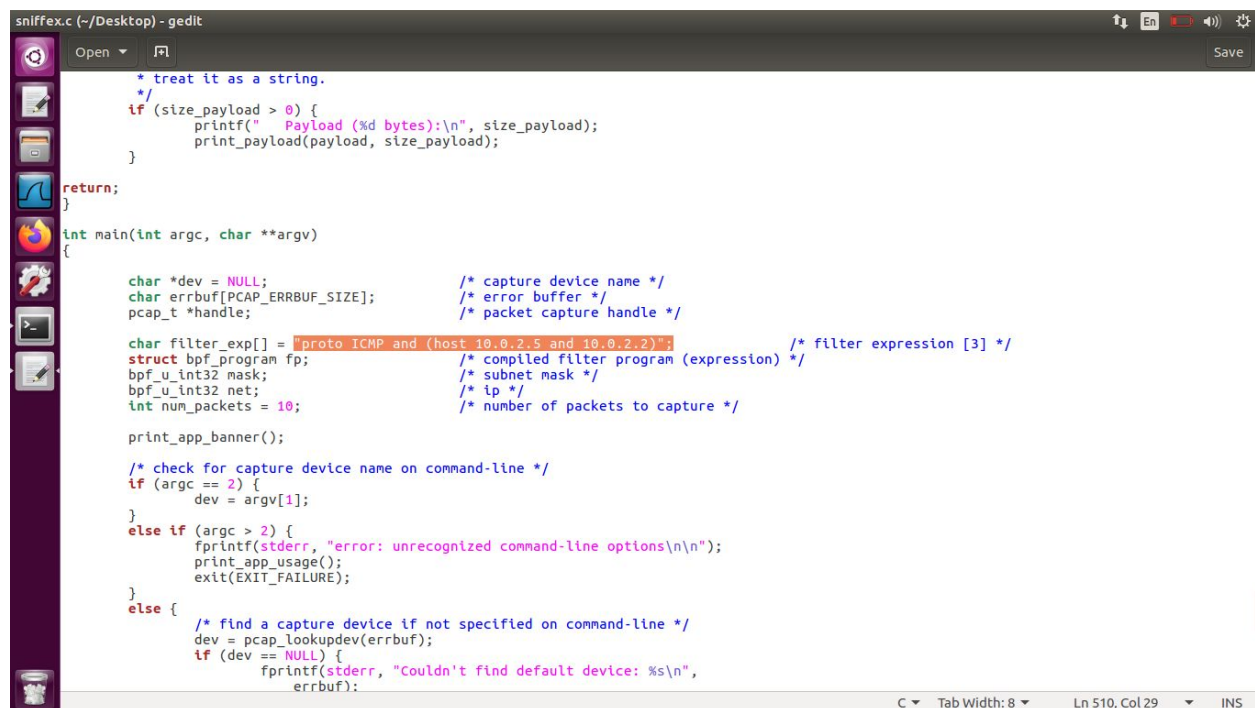
```
Terminal
seedpes1201800144rajdeep:~$ ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data:
64 bytes from 10.0.2.2: icmp_seq=1 ttl=64 time=0.484 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=64 time=0.621 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=64 time=0.861 ms
^C
--- 10.0.2.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2021ms
rtt min/avg/max/mdev = 0.484/0.655/0.861/0.157 ms
seedpes1201800144rajdeep:~$ ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data:
64 bytes from 10.0.2.2: icmp_seq=1 ttl=64 time=0.872 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=64 time=0.832 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=64 time=0.909 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=64 time=1.04 ms
64 bytes from 10.0.2.2: icmp_seq=5 ttl=64 time=0.422 ms
64 bytes from 10.0.2.2: icmp_seq=6 ttl=64 time=0.957 ms
64 bytes from 10.0.2.2: icmp_seq=7 ttl=64 time=1.18 ms
^C64 bytes from 10.0.2.2: icmp_seq=8 ttl=64 time=0.542 ms
^C
--- 10.0.2.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7070ms
rtt min/avg/max/mdev = 0.422/0.845/1.183/0.236 ms
seedpes1201800144rajdeep:~$ ^C
```

The victim pings to a host other than the attacker machine and the attacker cannot sniff this connection(in the screenshot above) since the promiscuous mode is off in the attacker machine.

When promiscuous mode is turned on, the attacker machine can sniff all the packets in the network may those be for the attacker machine or not. On the other hand, when the promiscuous mode is turned off, the attacker machine only captures the packets in which it itself is the source or destination.

TASK 1.2B

ICMP Connection:



```
sniffex.c (~/Desktop) - gedit
/* treat it as a string.
*/
if (size_payload > 0) {
    printf(" Payload (%d bytes):\n", size_payload);
    print_payload(payload, size_payload);
}
return;
}

int main(int argc, char **argv)
{
    char *dev = NULL; /* capture device name */
    char errbuf[PCAP_ERRBUF_SIZE]; /* error buffer */
    pcap_t *handle; /* packet capture handle */

    char filter_exp[] = "proto ICMP and (host 10.0.2.5 and 10.0.2.2)"; /* filter expression [3] */
    struct bpf_program fp; /* compiled filter program (expression) */
    bpf_u_int32 mask; /* subnet mask */
    bpf_u_int32 net; /* ip */
    int num_packets = 10; /* number of packets to capture */

    print_app_banner();

    /* check for capture device name on command-line */
    if (argc == 2) {
        dev = argv[1];
    }
    else if (argc > 2) {
        fprintf(stderr, "error: unrecognized command-line options\n\n");
        print_app_usage();
        exit(EXIT_FAILURE);
    }
    else {
        /* find a capture device if not specified on command-line */
        dev = pcap_lookupdev(errbuf);
        if (dev == NULL) {
            fprintf(stderr, "Couldn't find default device: %s\n",
                errbuf);
        }
    }
}
```

Filter set to → “proto ICMP and (host 10.0.2.5 and 10.0.2.2)”


```
Terminal
seed@pes1201800144rajdeep:~$ ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data:
64 bytes from 10.0.2.2: icmp_seq=1 ttl=64 time=0.292 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=64 time=0.495 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=64 time=0.962 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=64 time=1.01 ms
64 bytes from 10.0.2.2: icmp_seq=5 ttl=64 time=0.809 ms
64 bytes from 10.0.2.2: icmp_seq=6 ttl=64 time=0.850 ms
64 bytes from 10.0.2.2: icmp_seq=7 ttl=64 time=0.577 ms
64 bytes from 10.0.2.2: icmp_seq=8 ttl=64 time=0.699 ms
64 bytes from 10.0.2.2: icmp_seq=9 ttl=64 time=1.09 ms
64 bytes from 10.0.2.2: icmp_seq=10 ttl=64 time=0.820 ms
64 bytes from 10.0.2.2: icmp_seq=11 ttl=64 time=1.05 ms
64 bytes from 10.0.2.2: icmp_seq=12 ttl=64 time=0.378 ms
64 bytes from 10.0.2.2: icmp_seq=13 ttl=64 time=0.645 ms
64 bytes from 10.0.2.2: icmp_seq=14 ttl=64 time=0.267 ms
64 bytes from 10.0.2.2: icmp_seq=15 ttl=64 time=0.450 ms
64 bytes from 10.0.2.2: icmp_seq=16 ttl=64 time=1.00 ms
64 bytes from 10.0.2.2: icmp_seq=17 ttl=64 time=0.947 ms
64 bytes from 10.0.2.2: icmp_seq=18 ttl=64 time=0.823 ms
^C
--- 10.0.2.2 ping statistics ---
18 packets transmitted, 18 received, 0% packet loss, time 17239ms
rtt min/avg/max/mdev = 0.267/0.732/1.096/0.264 ms
seed@pes1201800144rajdeep:~$
```

Victim machine(10.0.2.5) pings to 10.0.2.2

```
Terminal
seed@pes1201800144rajdeep:~/Desktop$ gcc sniffex.c -lpcap
seed@pes1201800144rajdeep:~/Desktop$ sudo ./a.out
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: enp0s3
Number of packets: 10
Filter expression: proto ICMP and (host 10.0.2.5 and 10.0.2.2)

Packet number 1:
  From: 10.0.2.5
  To: 10.0.2.2
  Protocol: ICMP

Packet number 2:
  From: 10.0.2.2
  To: 10.0.2.5
  Protocol: ICMP

Packet number 3:
  From: 10.0.2.5
  To: 10.0.2.2
  Protocol: ICMP

Packet number 4:
  From: 10.0.2.2
  To: 10.0.2.5
  Protocol: ICMP

Packet number 5:
  From: 10.0.2.5
  To: 10.0.2.2
  Protocol: ICMP

Packet number 6:
  From: 10.0.2.2
  To: 10.0.2.5
  Protocol: ICMP

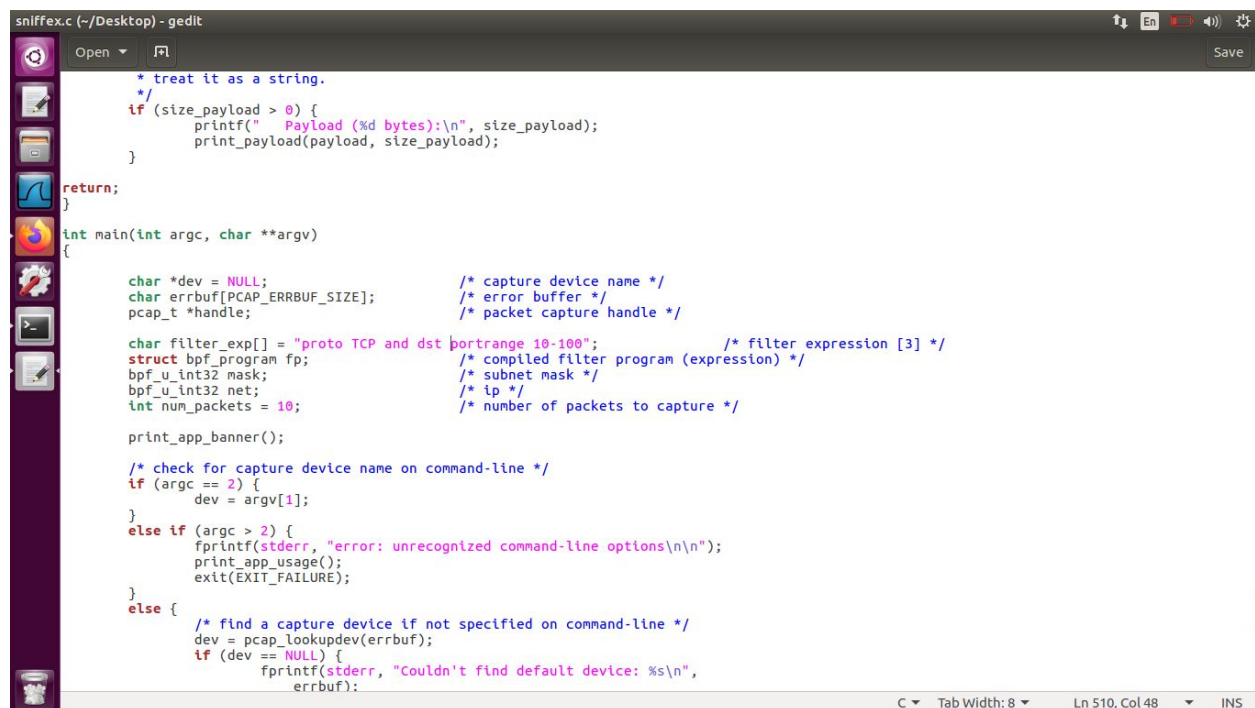
Packet number 7:
  From: 10.0.2.5
  To: 10.0.2.2
  Protocol: ICMP

Packet number 8:
  From: 10.0.2.2
  To: 10.0.2.5
  Protocol: ICMP
```

Attacker machine(10.0.2.9) captures ICMP packets from victim machine(10.0.2.5)

When the filter is set to “proto ICMP and (host 10.0.2.5 and 10.0.2.2)” then the attacker machine filters the sniffed packets and only receives the ICMP packets.

FTP Connection:



```
sniffex.c (~/Desktop) - gedit
Open [icon] Save

/* treat it as a string.
*/
if (size_payload > 0) {
    printf("    Payload (%d bytes):\n", size_payload);
    print_payload(payload, size_payload);
}
}
return;
}

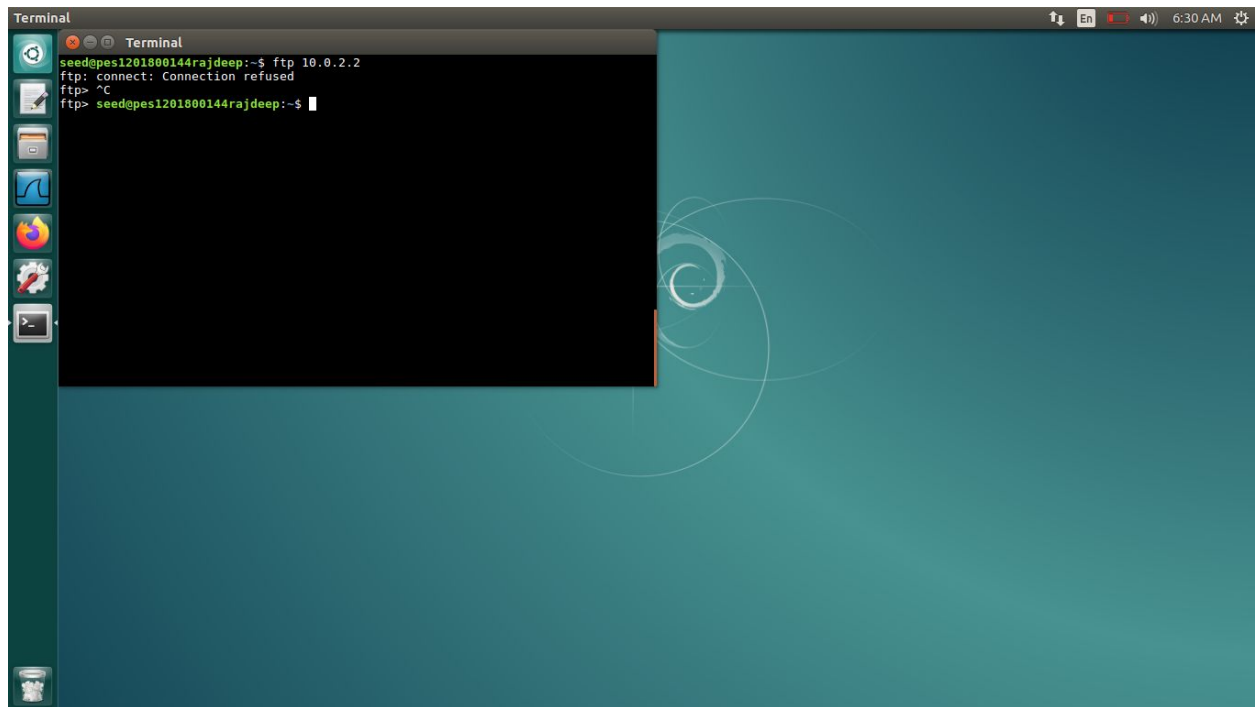
int main(int argc, char **argv)
{
    char *dev = NULL; /* capture device name */
    char errbuf[PCAP_ERRBUF_SIZE]; /* error buffer */
    pcap_t *handle; /* packet capture handle */

    char filter_exp[] = "proto TCP and dst portrange 10-100"; /* filter expression [3] */
    struct bpf_program fp; /* compiled filter program (expression) */
    bpf_u_int32 mask; /* subnet mask */
    bpf_u_int32 net; /* ip */
    int num_packets = 10; /* number of packets to capture */

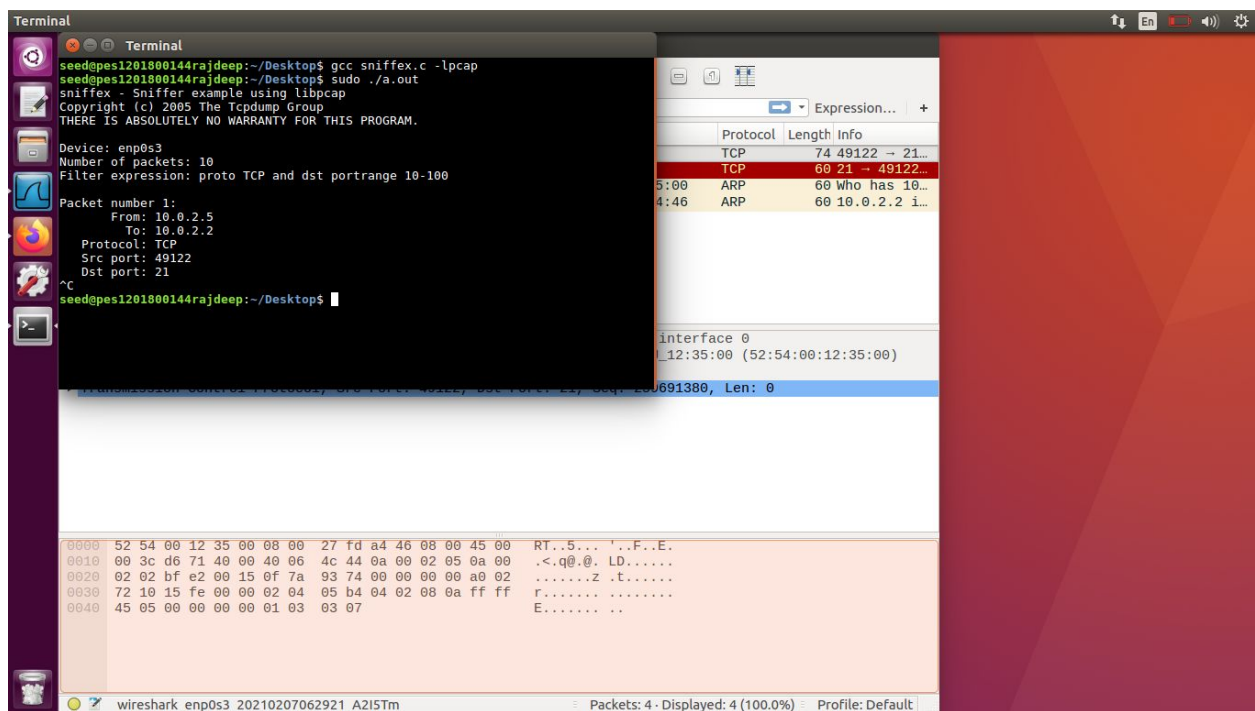
    print_app_banner();

    /* check for capture device name on command-line */
    if (argc == 2) {
        dev = argv[1];
    }
    else if (argc > 2) {
        fprintf(stderr, "error: unrecognized command-line options\n\n");
        print_app_usage();
        exit(EXIT_FAILURE);
    }
    else {
        /* find a capture device if not specified on command-line */
        dev = pcap_lookupdev(errbuf);
        if (dev == NULL) {
            fprintf(stderr, "Couldn't find default device: %s\n",
                errbuf);
        }
    }
}
```

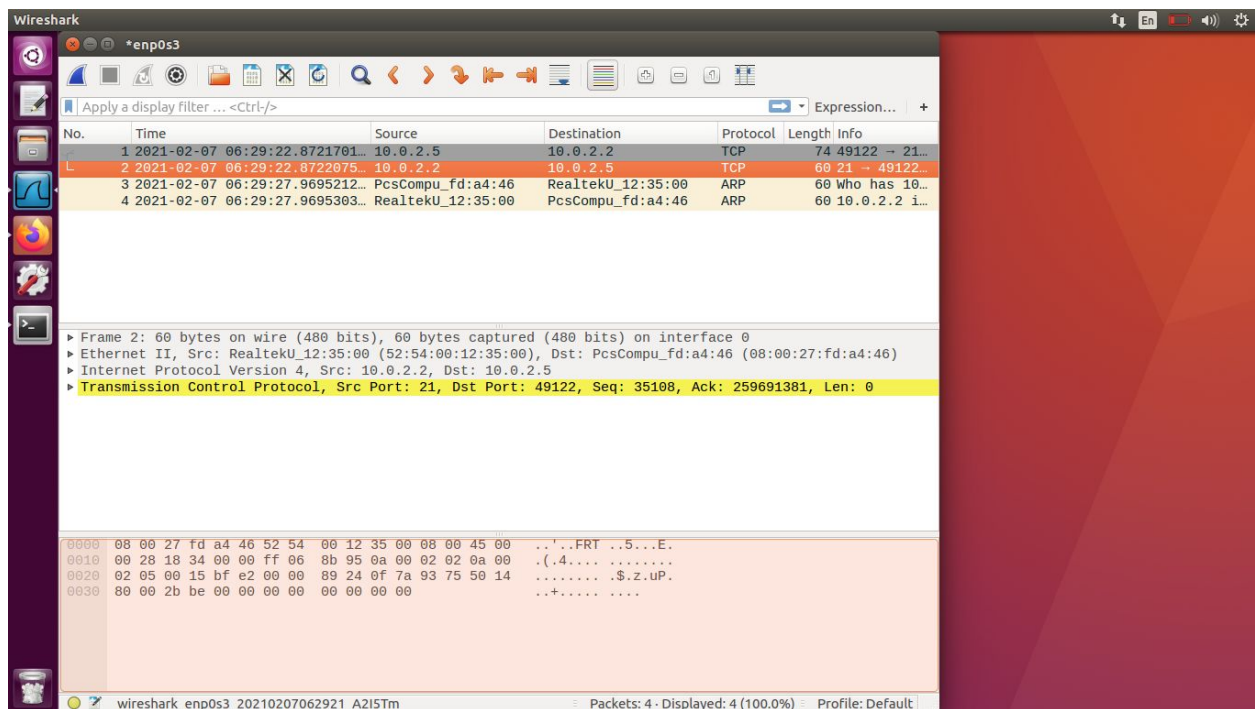
Filter set to → “proto TCP and dst portrange 10-100”



Victim machine(10.0.2.5) connecting through FTP protocol which uses TCP through port 21



Attacker machine(10.0.2.9) sniffs the FTP connection and captures username and password

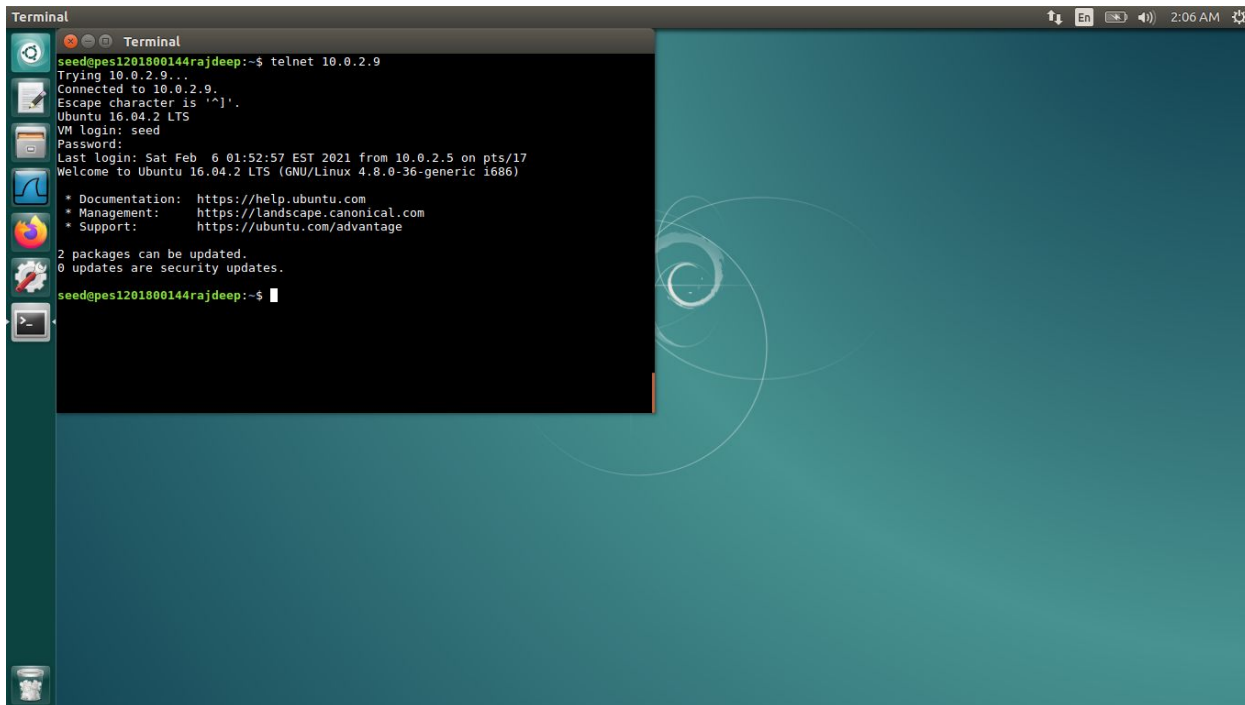


On wireshark in the attacker machine, the port number can be found as 21.

When the filter is set to “proto TCP and dst portrange 10-100” then the attacker machine filters the sniffed packets and only receives the TCP packets with port numbers in the range 10 to 100. These include FTP, telnet, SMTP, HTTP connections.



TASK 1.2C

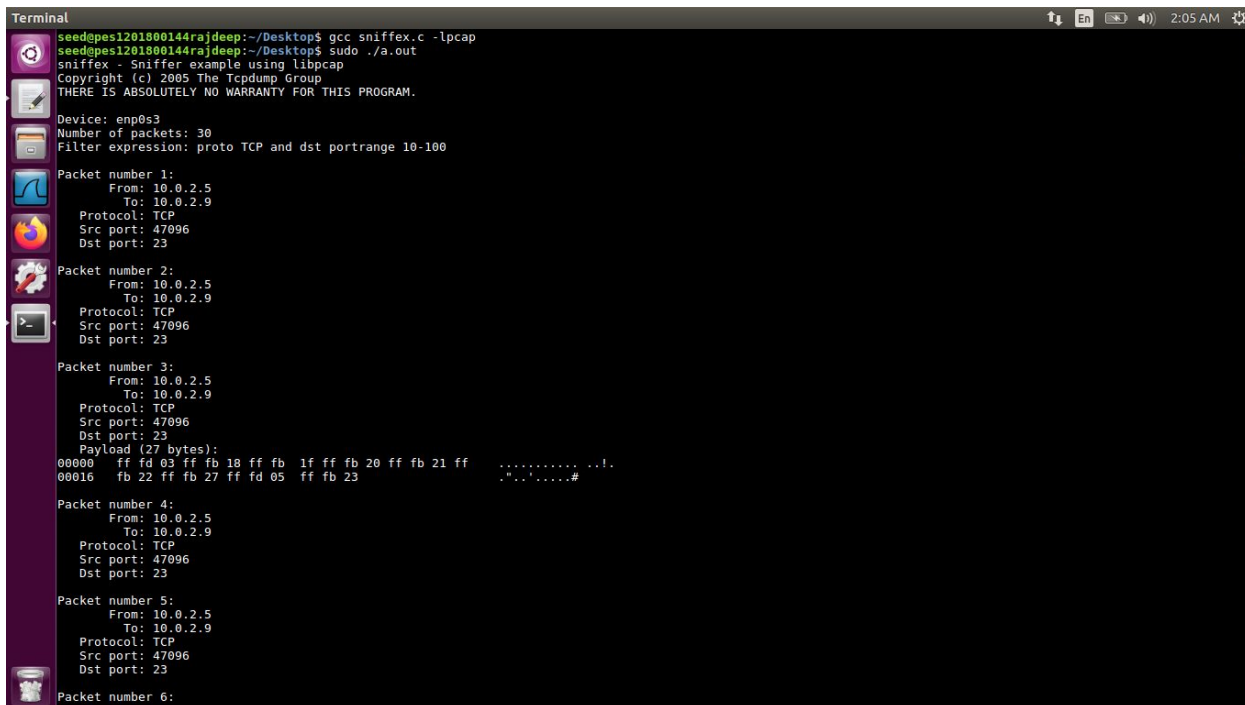


```
Terminal
seed@pes1201800144rajdeep:~$ telnet 10.0.2.9
Trying 10.0.2.9...
Connected to 10.0.2.9.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sat Feb  6 01:52:57 EST 2021 from 10.0.2.5 on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

2 packages can be updated.
0 updates are security updates.
seed@pes1201800144rajdeep:~$
```

Victim machine(10.0.2.5) connecting to the telnet server



```
Terminal
seed@pes1201800144rajdeep:~/Desktop$ gcc sniffex.c -lpcap
seed@pes1201800144rajdeep:~/Desktop$ sudo ./a.out
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Topdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: enp0s3
Number of packets: 30
Filter expression: proto TCP and dst portrange 10-100

Packet number 1:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23

Packet number 2:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23

Packet number 3:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23
  Payload (27 bytes):
00000  ff fd 03 ff fb 18 ff fb 1f ff fb 20 ff fb 21 ff  .....!..
00016  fb 22 ff fb 27 ff fd 05 ff fb 23  .....#

Packet number 4:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23

Packet number 5:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23

Packet number 6:
```

Attacker machine(10.0.2.9) sniffs the connection

```
Terminal
00000 73 s
Packet number 12:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23
Packet number 13:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23
  Payload (1 bytes):
00000 65 e
Packet number 14:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23
Packet number 15:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23
  Payload (1 bytes):
00000 65 e
Packet number 16:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23
Packet number 17:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23
  Payload (1 bytes):
00000 64 d
```

Further, on the terminal, we can find the username as 'seed' on the right side

```
Terminal
Packet number 22:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23
  Payload (1 bytes):
00000 64 d
Packet number 23:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23
  Payload (1 bytes):
00000 65 e
Packet number 24:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23
  Payload (1 bytes):
00000 65 e
Packet number 25:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23
  Payload (1 bytes):
00000 73 s
Packet number 26:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
  Src port: 47096
  Dst port: 23
  Payload (2 bytes):
00000 0d 00 ..
Packet number 27:
  From: 10.0.2.5
  To: 10.0.2.9
  Protocol: TCP
```

Then we can find the password as 'dees' on scrolling down the terminal

Since telnet connection runs on TCP port 23, all the traffic is sniffed by the code and the username and password is displayed in the terminal.

TASK 2

```
spoof.c (~/Desktop) - gedit
#include<unistd.h>
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<netinet/ip.h>
#include<arpa/inet.h>

struct udpheader
{
    u_int16_t udp_sport ;
    u_int16_t udp_dport ;
    u_int16_t udp_ulen ;
    u_int16_t udp_sum;
};

struct ipheader
{
    u_char iph_ihl:4,
    lph_ver:4;
    u_char iph_tos;
    u_short iph_len;
    u_short iph_id;
    u_short iph_off;
    u_char iph_ttl;
    u_char iph_protocol;
    u_short iph_sum;
    struct in_addr iph_src,iph_dst;
};

void send_raw_ip_packet (struct ipheader* ip )
{
    struct sockaddr_in dest_info ;
    int enable = 1 ;
    int sock= socket (AF_INET , SOCK_RAW, IPPROTO_RAW) ;
    setsockopt (sock , IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));
    dest_info.sin_family = AF_INET ;
    dest_info.sin_addr = ip->iph_dst ;
    sendto (sock, ip, ntohs(ip->iph_len), 0, (struct sockaddr*)&dest_info, sizeof(dest_info )) ;
    close (sock);
}
```

```
spoof.c (~/Desktop) - gedit
};

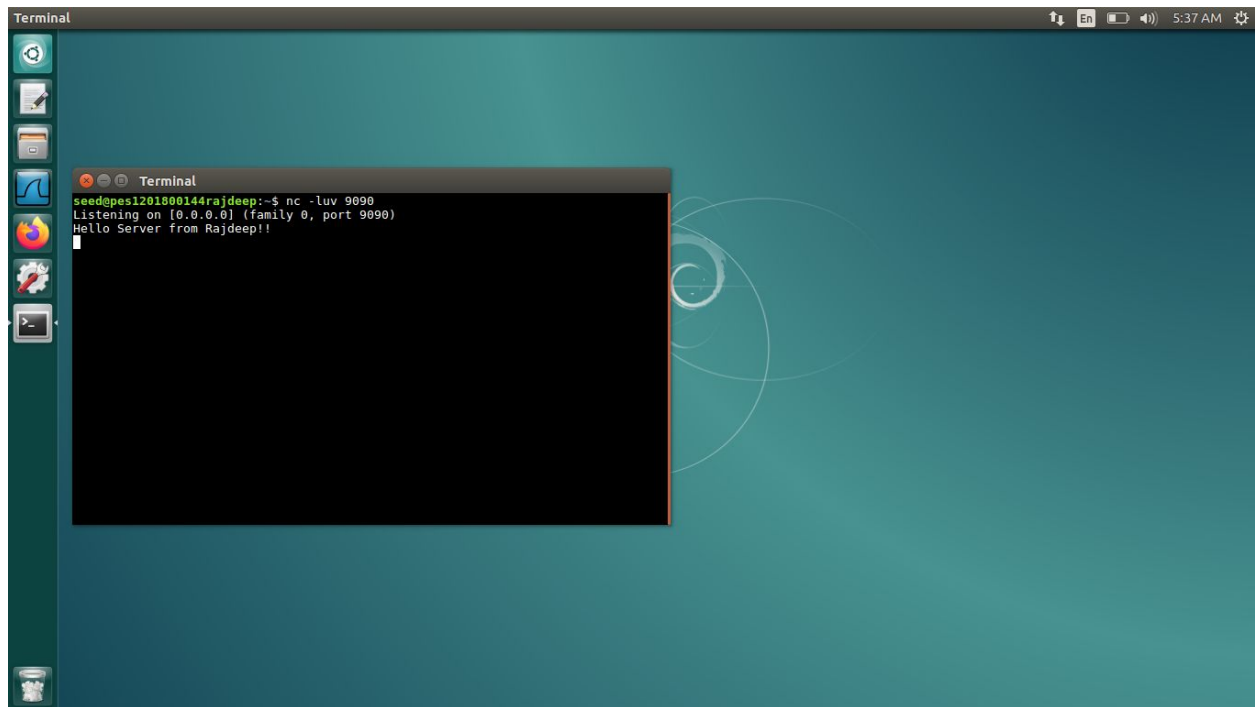
void send_raw_ip_packet (struct ipheader* ip )
{
    struct sockaddr_in dest_info ;
    int enable = 1 ;
    int sock= socket (AF_INET , SOCK_RAW, IPPROTO_RAW) ;
    setsockopt (sock , IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));
    dest_info.sin_family = AF_INET ;
    dest_info.sin_addr = ip->iph_dst ;
    sendto (sock, ip, ntohs(ip->iph_len), 0, (struct sockaddr*)&dest_info, sizeof(dest_info )) ;
    close (sock);
}

int main()
{
    char buffer[1500];
    memset(buffer, 0 , 1500) ;
    struct ipheader *ip = (struct ipheader *) buffer;
    struct udpheader *udp = (struct udpheader *) (buffer +sizeof (struct ipheader)) ;
    char *data= buffer+ sizeof(struct ipheader) +sizeof(struct udpheader);
    const char *msg = "Hello Server from Rajdeep!!\n";
    int data_len = strlen (msg) ;
    strncpy (data , msg , data_len);
    udp->udp_sport = htons (12345);
    udp->udp_dport = htons(9090) ;
    udp->udp_ulen = htons(sizeof(struct udpheader) + data_len) ;
    udp->udp_sum = 0;

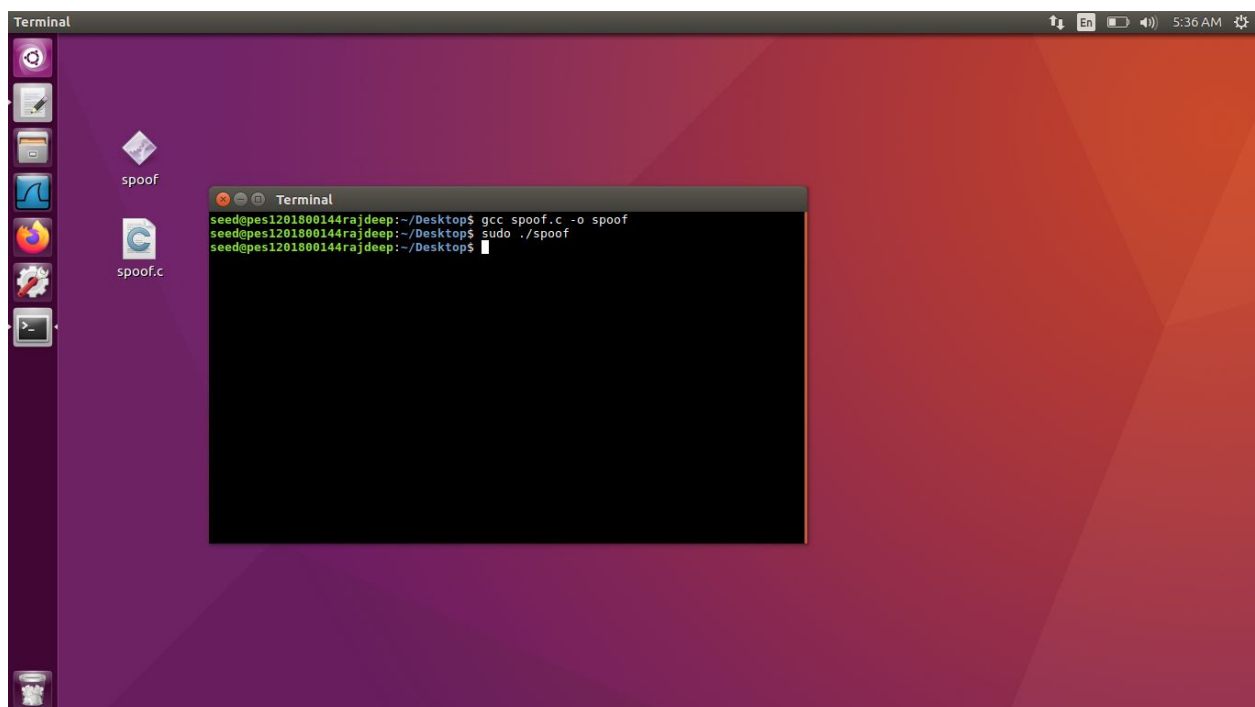
    ip->iph_ver = 4 ;
    ip->iph_ihl = 5 ;
    ip->iph_ttl = 64 ;
    ip->iph_src.s_addr = inet_addr("1.2.3.4");
    ip->iph_dst.s_addr = inet_addr("10.0.2.5");
    ip->iph_protocol = IPPROTO_UDP ;
    ip->iph_len = htons(sizeof(struct ipheader) +sizeof(struct udpheader) + data_len);

    send_raw_ip_packet(ip);
    return 0;
}
```

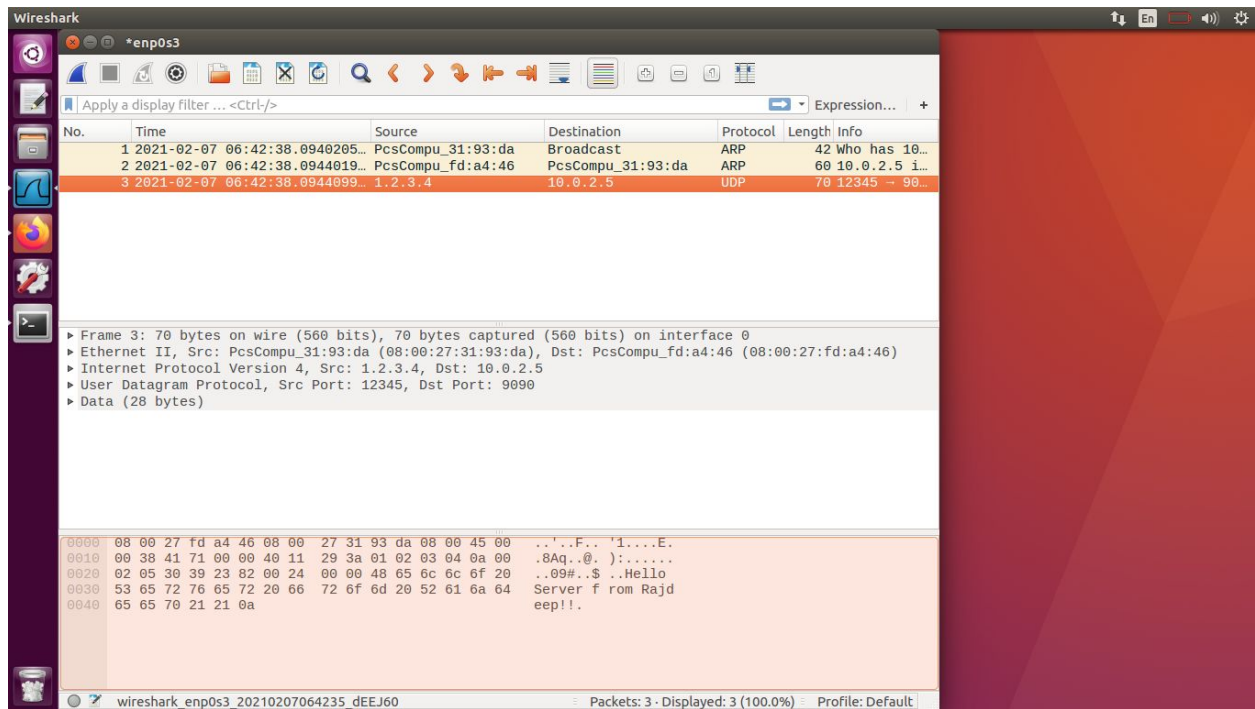
The spoof.c full code



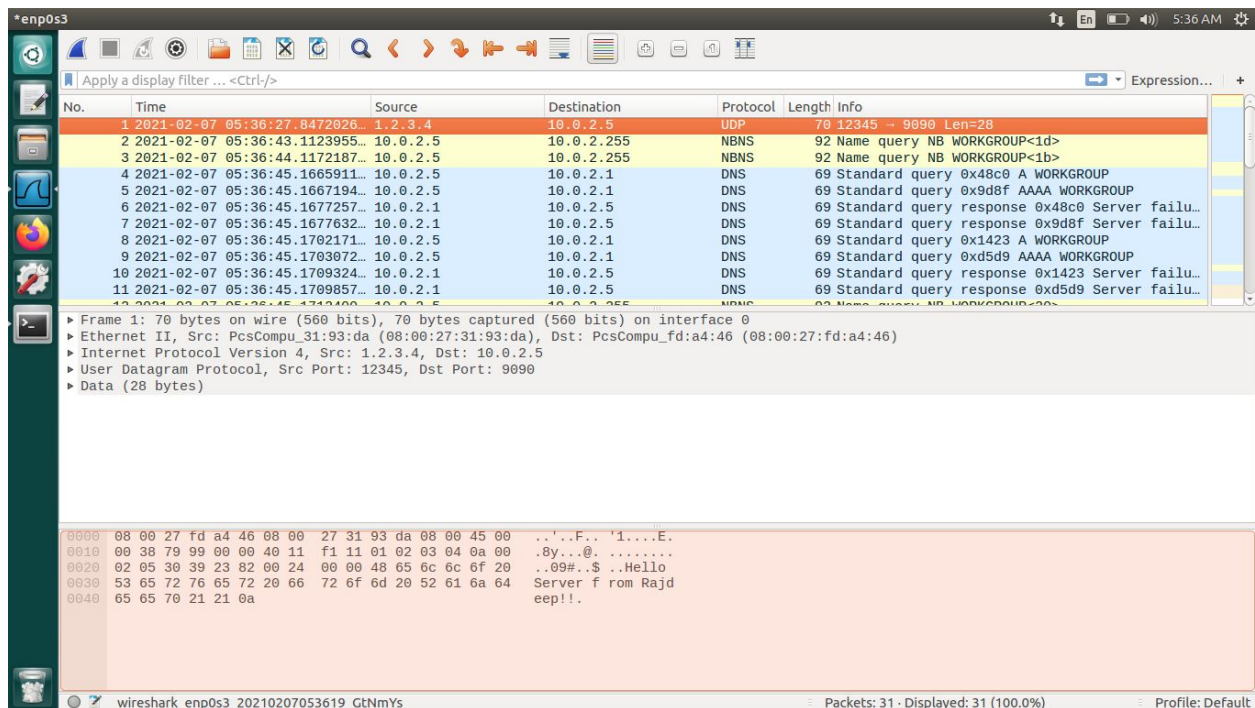
Victim machine listens on port 9090



Attacker machine executing the spoof.c code



Wireshark on attacker machine



Wireshark on victim machine

The server machine(10.0.2.5) listens using netcat command on port 9090. The attacker machine(10.0.2.9) sends a spoofed UDP packet to the victim machine(10.0.2.5) from the IP address 1.2.3.4. This is also shown using wireshark screenshots.

