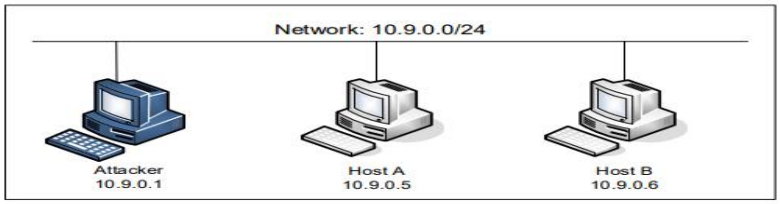
# COMPUTER NETWORK SECURITY LABORATORY

| NAME : PREM SAGAR J S | SRN : PES1UG20CS825 | SEC : H |
|---|---|---|

# Sniffing and Spoofing using PCAP Library

**Lab Environment Setup**




## Lab Task Set-2:

## Writing Programs to Sniff and Spoof Packets using pcap (C programs)

## Task 2.1 : Sniffing - Writing Packet Sniffing Program

### Task 2.1 A : Understanding how a Sniffer Works

On the host VM :

# gcc -o sniff Task2.1A.c -lpcap

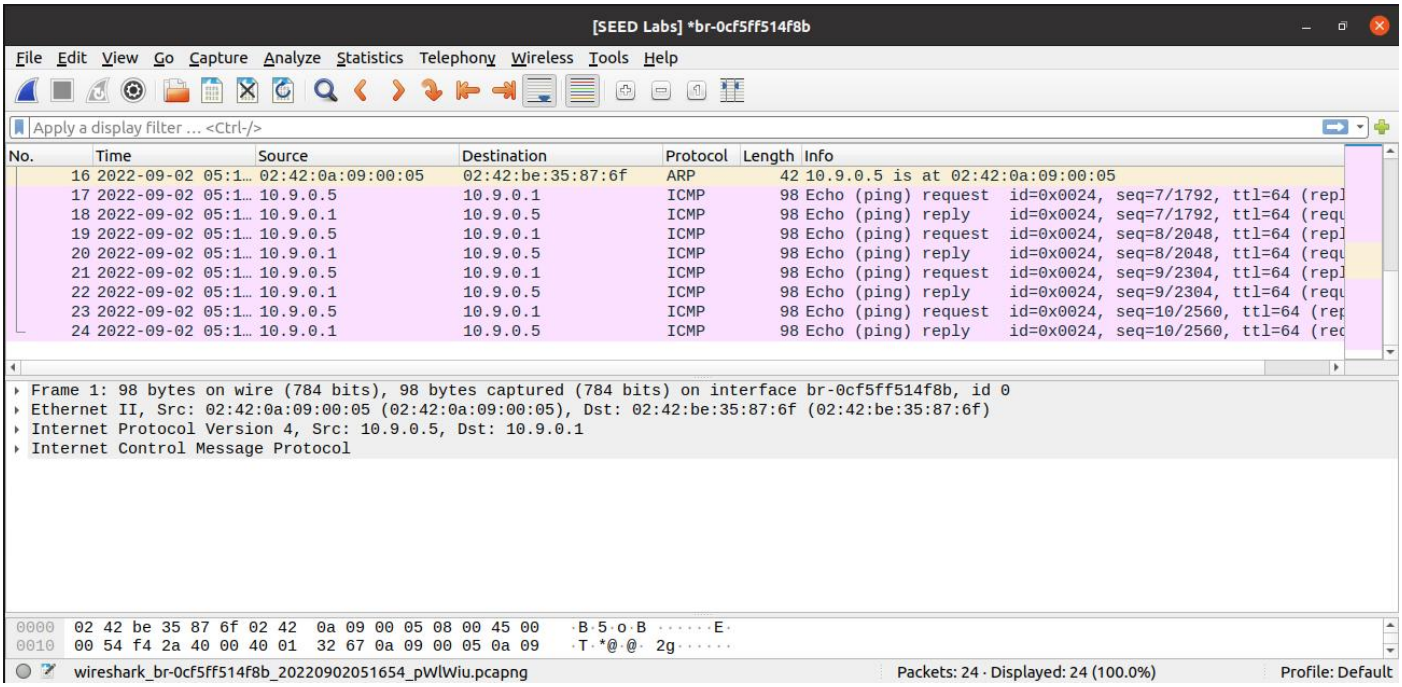
Code has been compiled on the Host System and Copied to the Volumes Directory

On Host A terminal :
# ping 10.9.0.1

```
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup  seed@VM: ~/.../Labsetup  seed@VM: ~/.../Labsetup  seed@VM: ~/.../Labsetup  seed@VM: ~/.../Code
Host A:PES1UG20CS825:Prem Sagar J S:/
>ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
64 bytes from 10.9.0.1: icmp_seq=1 ttl=64 time=0.361 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.130 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.149 ms
64 bytes from 10.9.0.1: icmp_seq=4 ttl=64 time=0.140 ms
64 bytes from 10.9.0.1: icmp_seq=5 ttl=64 time=0.226 ms
64 bytes from 10.9.0.1: icmp_seq=6 ttl=64 time=0.140 ms
64 bytes from 10.9.0.1: icmp_seq=7 ttl=64 time=6.85 ms
64 bytes from 10.9.0.1: icmp_seq=8 ttl=64 time=0.198 ms
64 bytes from 10.9.0.1: icmp_seq=9 ttl=64 time=0.190 ms
64 bytes from 10.9.0.1: icmp_seq=10 ttl=64 time=0.121 ms
^C
--- 10.9.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9222ms
rtt min/avg/max/mdev = 0.121/0.850/6.852/2.001 ms
```

On the Attacker container running the command:
# ./sniff


Wireshark Screenshot of capturing the packets :




**Question 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.

1. pcap_lookupdev: Finds a capture device to sniff on
2. pcap_lookupnet: Returns the network number and mask for the capture device
3. pcap_open_live: Starts sniffing on the capture device

4. pcap_datalink: Returns the kind of device we're capturing on
5. pcap_compile: Compiles the filter expression stored in a regular stringin order to set the filter
6. pcap_setfilter: Sets the compiled filter
7. At this point, we can either sniff one packet at a time (pcap_next) or continuously sniff (pcap_loop). Since sniff.c uses we'll continue with pcap_loop: Sets callback function for new (filtered!) packets
8. pcap_freecode: Frees up allocated memory generated by pcap_compile
9. pcap_close: Closes the sniffing session

**Question 2**:

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

You need root in order for sniffex to run because sniffex will need to access a network device which a non-root user cannot do.

The code that causes this to fail is:

```
/* 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);

exit(EXIT_FAILURE);

}
```

Running commands without root privilege

On the Attacker container run the command :
# su seed
# ./sniff


After running the sniff program run the command to return to root user on the attacker container:
# su root

**Question 3**:

Please turn on and turn off the promiscuous mode in your sniffer program. The value 1 of the third parameter in the **pcap_open_live() function** turns on the promiscuous mode (use 0 to turn it off). Can you demonstrate the difference when this mode is on and off?

Promiscuous mode allows for a network sniffer to pass all traffic from a network controller and not just the traffic that the network controller was intended to receive. Whether or not the capture device is in promiscuous mode determines on the third parameter (a 'boolean' int) in pcap_open_live on line 69 . The code below highlights the difference:

/* promisc mode on */

**handle = pcap_open_live("br-****", BUFSIZ, 1, 1000, errbuf);**

/* promisc mode off */

**handle = pcap_open_live("br-****", BUFSIZ, 0, 1000, errbuf);**

Change the code given in line 69 of Task2.1A.c file to the following :
**handle = pcap_open_live("br-****", BUFSIZ, 0, 1000, errbuf);**

-> I have turned off the promisc mode off as shown in the below screenshot.

Turning off promiscuous mode is as easy as fliping a 1 to a 0 in pcap_open_live.

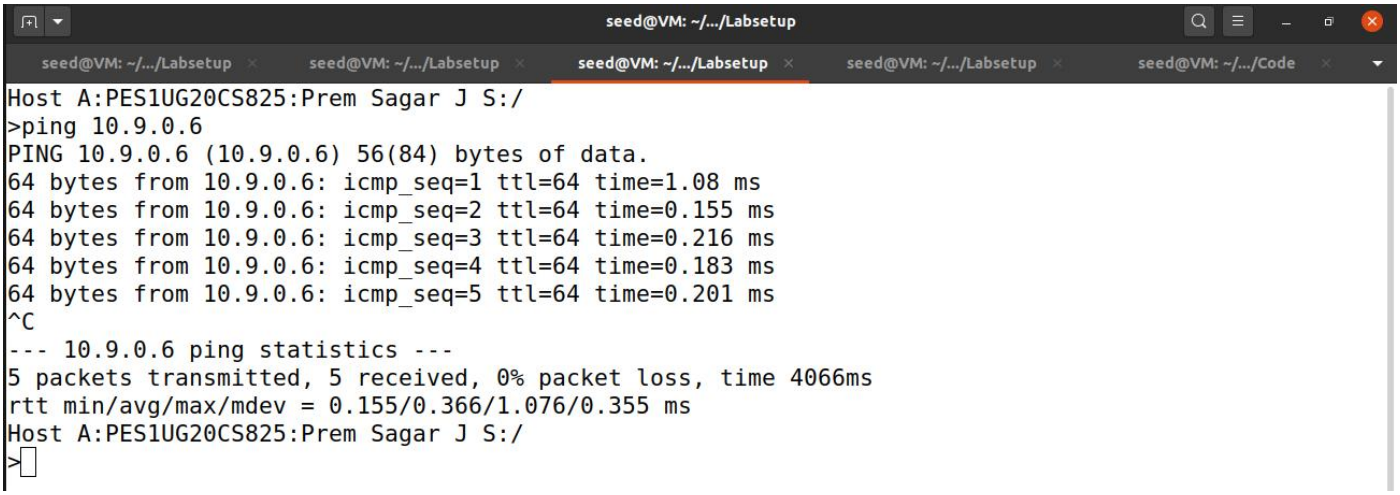
```
67
68  // Step 1: Open live pcap session on NIC with name br-****
69  handle = pcap_open_live("br-0cf5ff514f8b", BUFSIZ, 0, 1000, errbuf);
70
```

On the host VM :

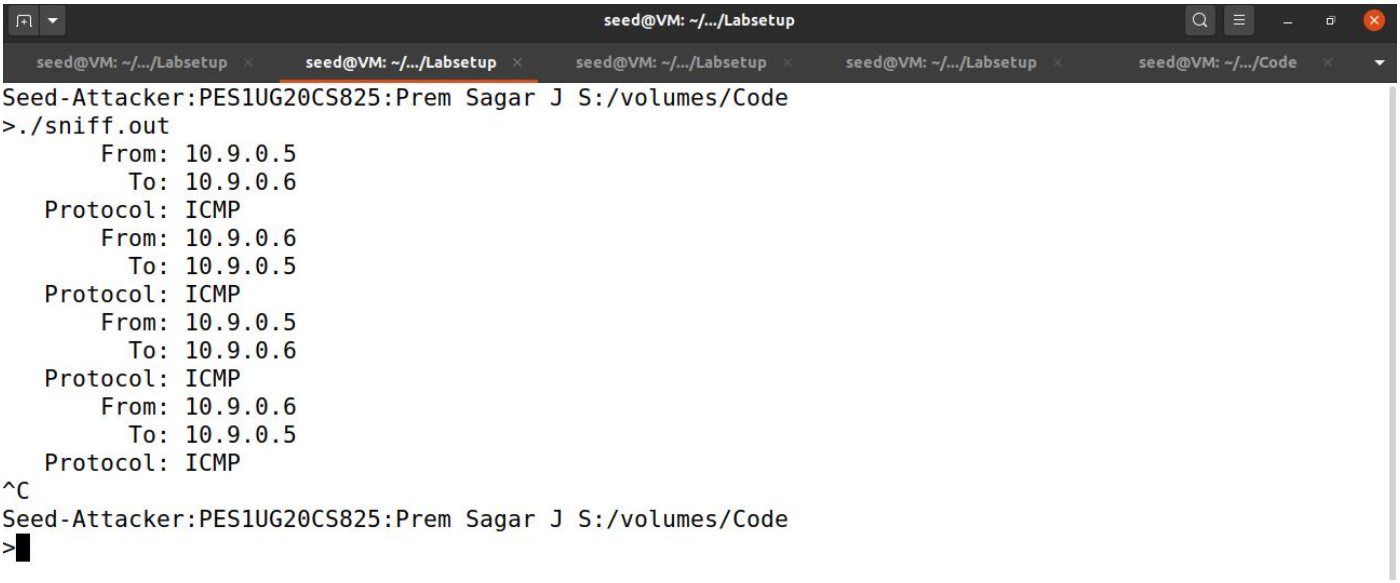
Code has been compiled on the Host System and Copied to the Volumes Directory

# gcc -o sniff Task2.1A.c -lpcap

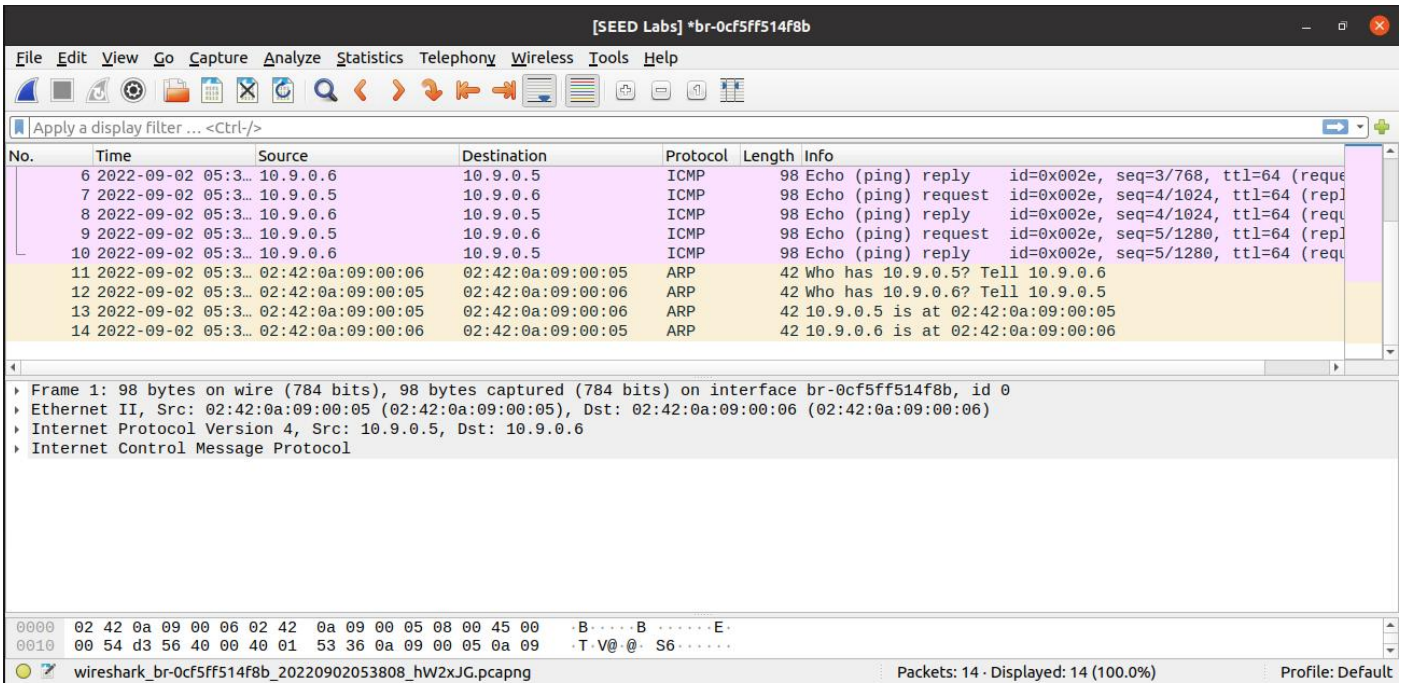
On Host A terminal :
# ping 10.9.0.6




On the Attacker terminal running the command:
# ./sniff




Wireshark screenshot of capturing packets:




## Observation :

- I have executed the program successfully and sniffed the packets.
- I got to know that if the promisc mode is off then we can only sniff the packets that are destined to my system.
- And I can't turn on promisc mode without the root privilege.
- I noticed that there several pcap lib functions that needs to be executed in sequence in order to achieve the sniffing process.
- Turning off promiscuous mode is as easy as fliping a 1 to a 0 in pcap_open_live.

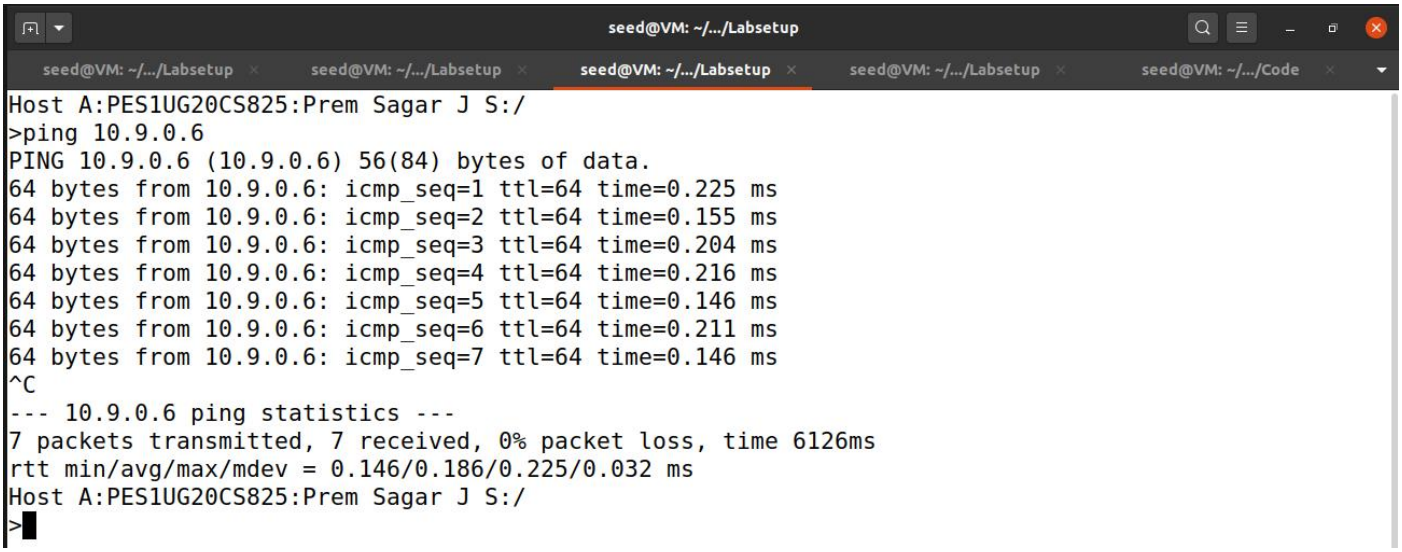
## Task 2.1 B : Writing Filters

### Capture the ICMP packets between two specific hosts

On the host VM :
# gcc -o sniff Task2.1B-ICMP.c -lpcap

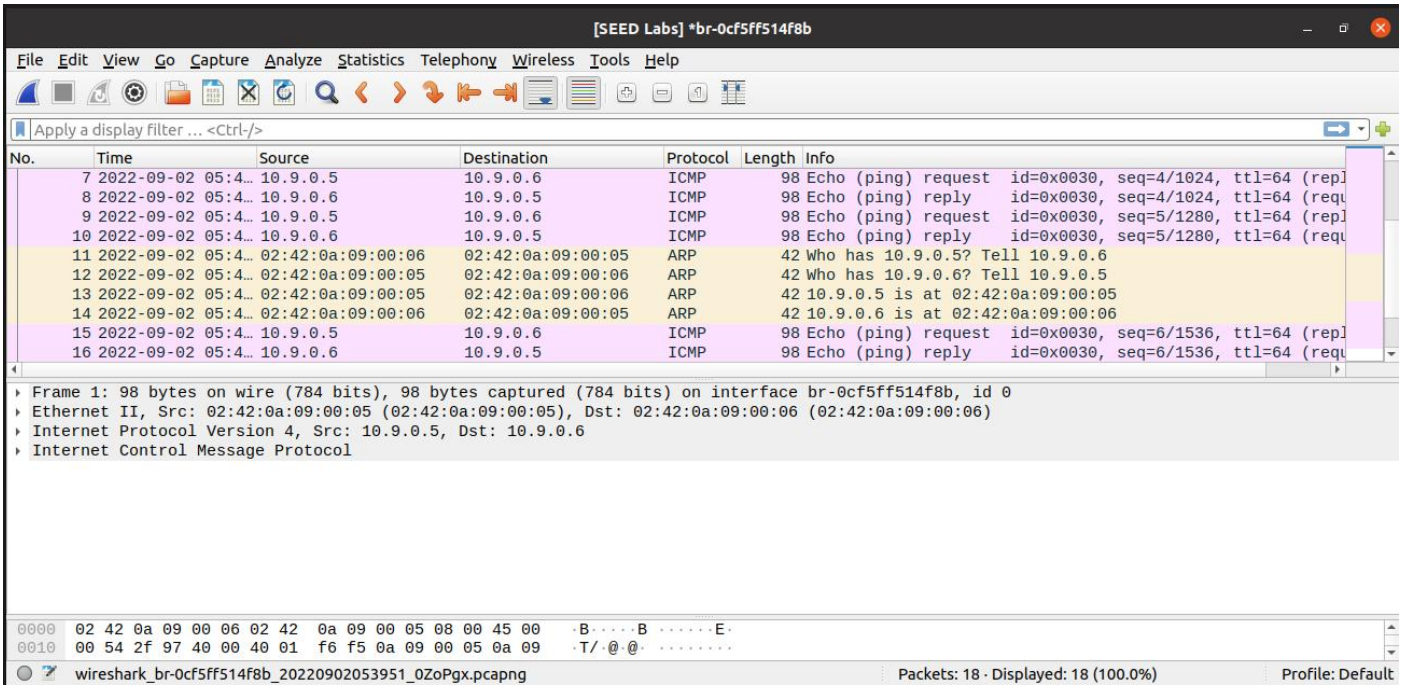
Code has been compiled on the Host System and Copied to the Volumes Directory

In the host A machine ping any ip address
# ping 10.9.0.6




On the Attacker terminal run the command:
# ./sniff


Wireshark Screenshot of Capturing packets :




## Observation :

- In this task we going to Capture the ICMP packets between two specific hosts that is Host A - 10.9.0.6 and Host B - 10.9.0.5.
- I'm able perform the sniffing ICMP packets between the Host A and Host B
- we can filter only the ICMP packets by modifying the filter_exp[] string.

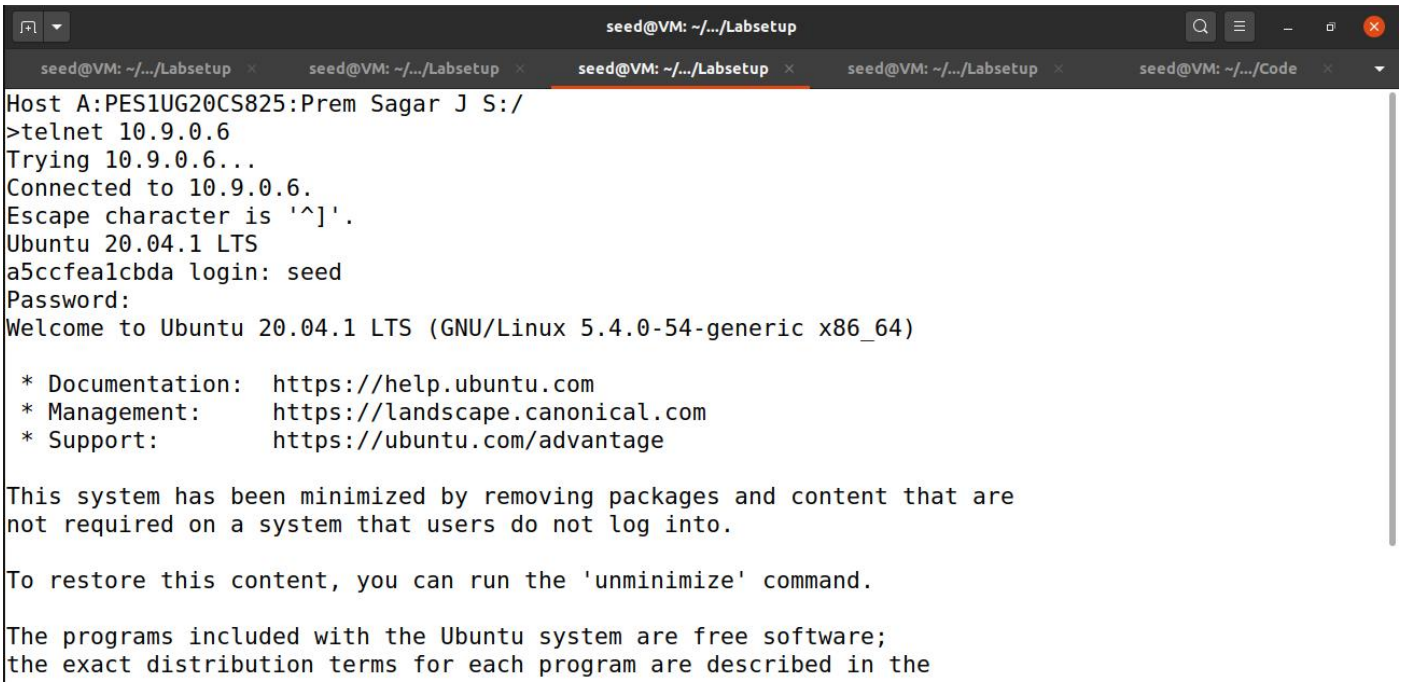
```
// ICMP packets between the host A and host b

char filter_exp[] = "icmp and (src host 10.9.0.6 and dst host 10.9.0.5) or

(src host 10.6.9.5 and dst host 10.9.0.6)";
```

**Capture the TCP packets that have a destination port range from to sort 10 - 100.**

On the host VM :
# gcc -o sniff Task2.1B-TCP.c -lpcap

Code has been compiled on the Host System and Copied to the Volumes Directory
On Attacker Machine terminal :
# ./sniff

On Host A terminal :
# telnet 10.9.0.6




## Observation :

- In this Task Capture the TCP packets that have a destination port range from to sort 10 - 100.
- For capturing TCP packets we are using telnet protocol which uses TCP and runs on the port number 23.
- We are logging remotely into the host B from host A while its happening we are sniffing the TCP packets between these hosts.
- Changing the filter_exp[] variable again we can sniff only TCP packets with a destination port from 10 to 100.

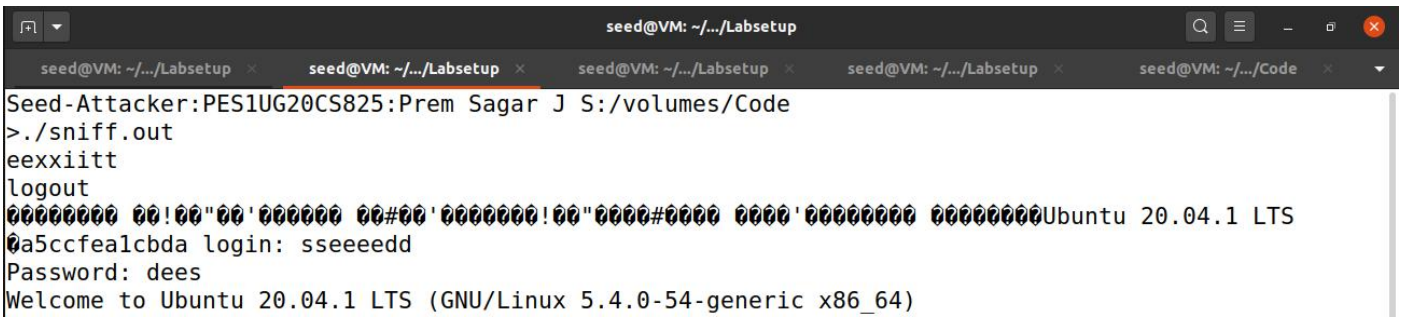
```
// TCP packets with dest port 10-100char

filter_exp[] = "tcp dst portrange 10-100";
```

## Task 2.1 C : Sniffing Passwords

**On the host VM** :

**# gcc -o sniff Task2.1C.c -lpcap**
Code has been compiled on the Host System and Copied to the Volumes Directory

**On the Attacker terminal run the command:**
**# ./sniff**



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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Fri Sep  2 09:53:50 UTC 2022 from hostA-10.9.0.5.net-10.9.0.0 on pts/2
```

**On Host A terminal** :
**# telnet 10.9.0.6**


## Observation :

- Here in this task we are sniffing the passwords.
- Since we're sniffing telnet passwords, we can just look for tcp packets on port 23.

  char filter_exp[] = "tcp port 23";

- I'm able to successfully run the sniffer code and sniffed the telnet password while the host A was using telnet login into the host B.

# Task 2.2 Spoofing

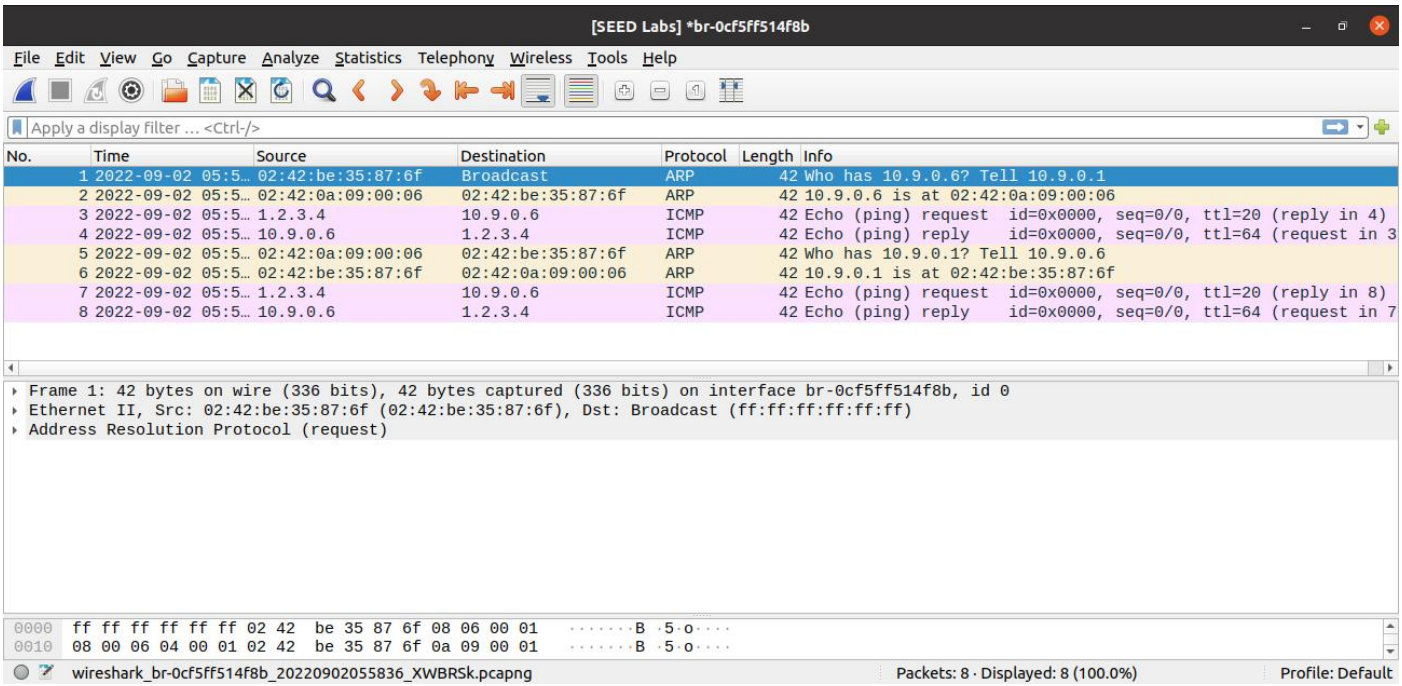
### Task 2.2 B : Spoof an ICMP Echo Request

On the host VM :

# gcc -o spooficmp Task2.2.c -lpcap
Code has been compiled on the Host System and Copied to the Volumes Directory

On Attacker Machine terminal :
# ./spooficmp


Wireshark Screenshot of spoofing ICMP packets :




- **Question 4**:

  Using the raw socket programming, do you have to calculate the checksum for the IP header?

  No the computer generally the system automatically does this, or rather it fills it in.

  When you create a socket and bind it to a process/port, you don't care about IP or TCP header fields as long as you are able to communicate with the server. The kernel or the underlying operating system builds the packet including the checksum for your data.

- **Question 5**:

  Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?

In short this is how it is defined by the authorities who set networking rules. Due to the fact one can create custom packets that could prove detrimental to a network configuration.

### Observation :

- Here in this task we are spoofing the ICMP echo request packet
- I have changed interface Id with my interface Id in the code.
- I'm able to execute this code and perform spoofing process and get echo reply from the Host, That means we successfully spoofed the ICMP echo request packet.

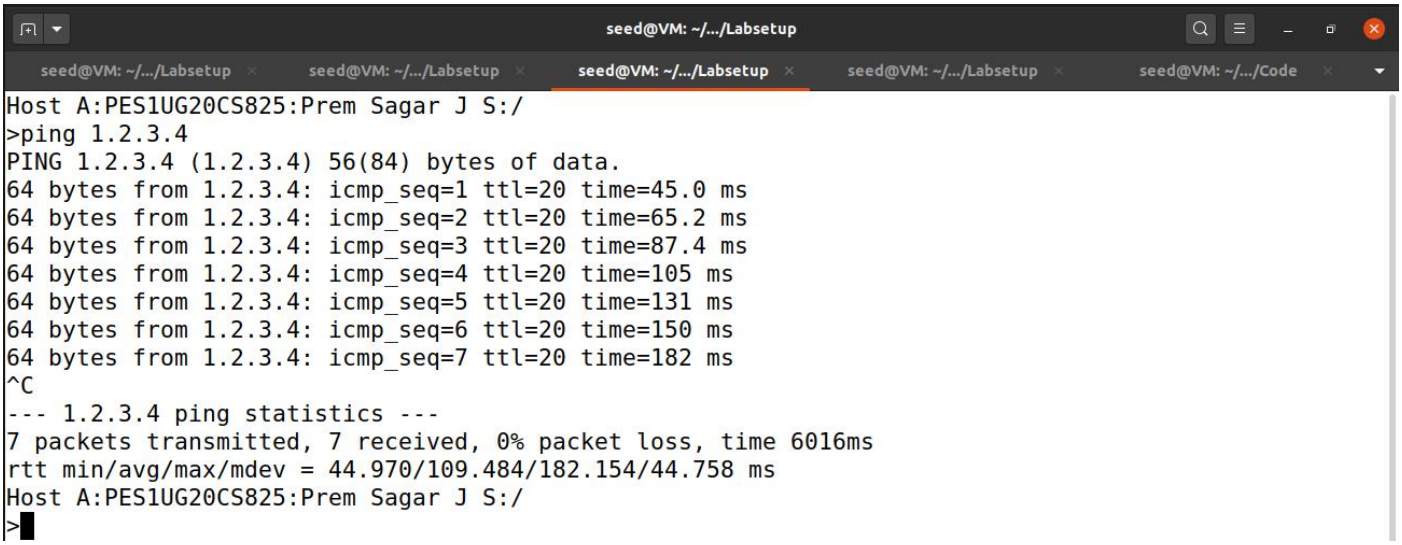
# Task 2.3 Sniff and then Spoof

On the host VM :
# gcc -o sniffspoof Task2.3.c -lpcap

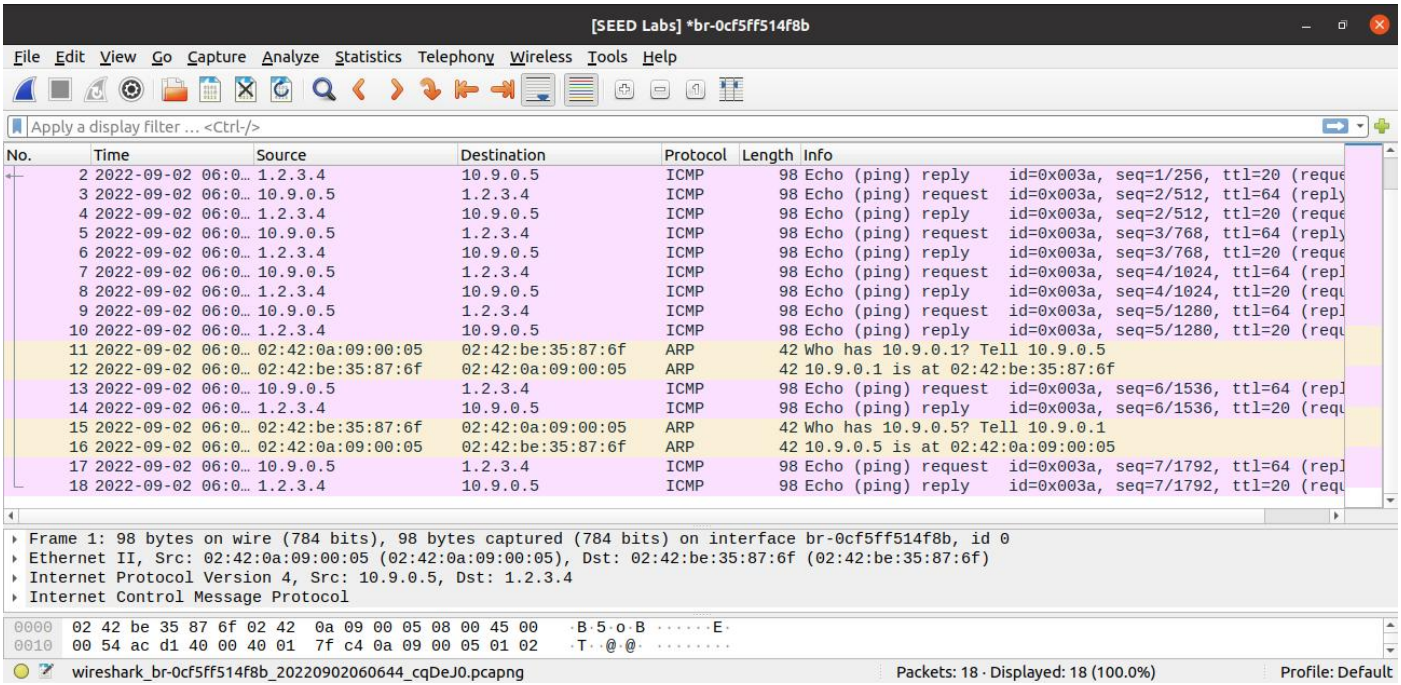
Code has been compiled on the Host System and Copied to the Volumes Directory.

On Attacker Machine terminal :
# ./sniffspoof


On the Host A terminal ping 1.2.3.4
# ping 1.2.3.4



Wireshark screenshot of sniffed and spoofed packets :




## Observation :

- Here in this task the victim machine pings a non-existing IP address “1.2.3.4”. As the attacker machine is in the same network, it sniffs the request packet, creates a new echo reply packet with IP and ICMP header and sends it to the victim machine. Hence the user will always receive an echo reply from a non-existing IP address indicating that the machine is alive.

- We create a buffer of maximum length and fill it with an IP request header. We modify the IP header and ICMP header with our response data. In the new IP header, we interchange the source IP address and destination IP address and send the new IP packet using the raw sockets.

- I’m able to execute this task successfully as shown in the above screenshot.

- In this experiment I understood that how we can sniff the packets and then spoof packets.

- Sniff the packets on the same network as the victim and then create new echo reply with the info gathered while sniffing and Interchanging the header values in the ICMP packets.

- This kind of Tasks can be used while doing the Passive attacks such as Man In The Middle Attack.