

COMPUTER NETWORK

SECURITY

LAB REPORT

Sniffing and Spoofing using PCAP Library

NAME: VISHWAS M

SRN : PES2UG20CS390

SEC : F

DATE : 03/09/2022

LAB NO : 2

Task 2.1 : Sniffing - Writing Packet Sniffing Program

The objective of this lab is to understand the sniffing program which uses the pcap library. With pcap, the task of sniffers becomes invoking a simple sequence of procedures in the pcap library. You should provide screenshots to show that your program runs successfully and produces expected results.

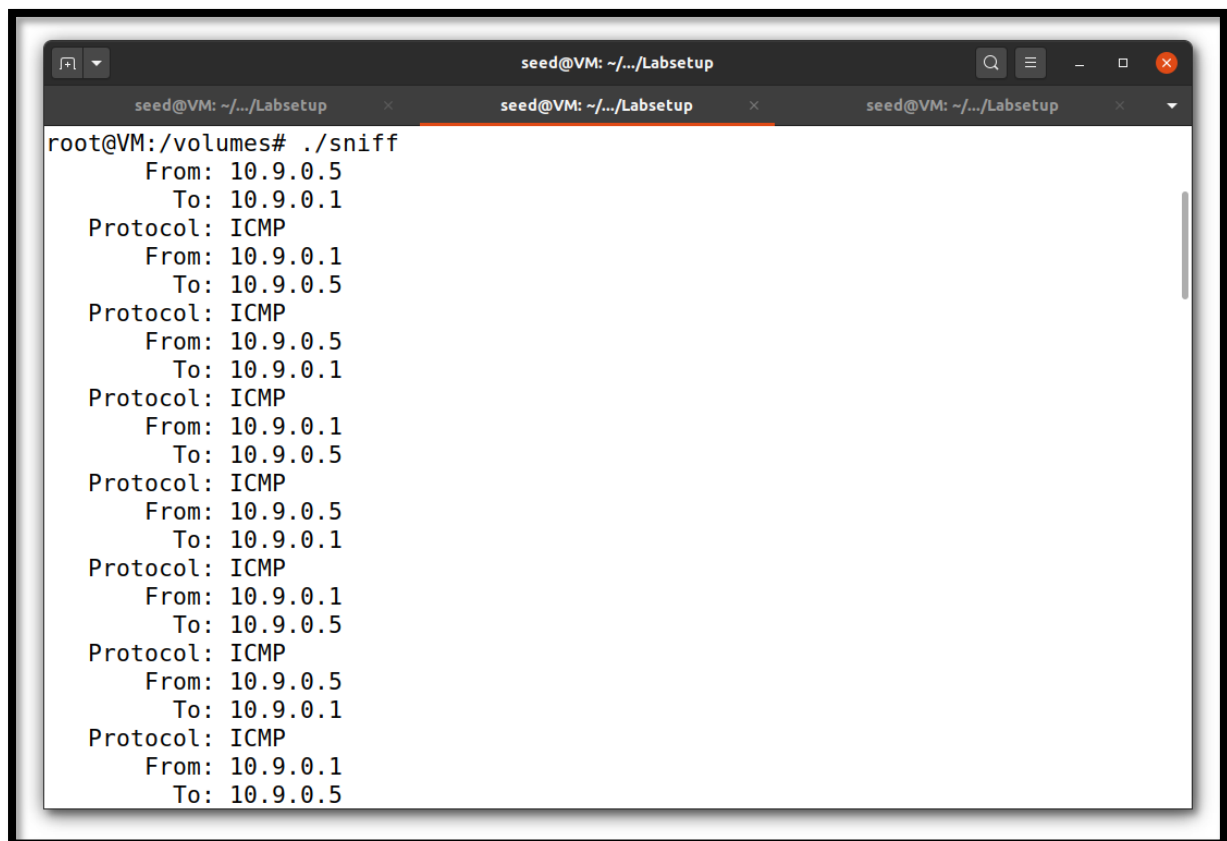
Task 2.1 A : Understanding how a Sniffer Works

Host VM:

```
[09/03/22]seed@VM:~/.../Lab2$ gcc -o sniff Task2.1A.c -lpcap  
[09/03/22]seed@VM:~/.../Lab2$ docker cp sniff 9753f0e34f43:/volumes
```

We are compiling the code in the host machine and then we are copying the compiled code inside the container.

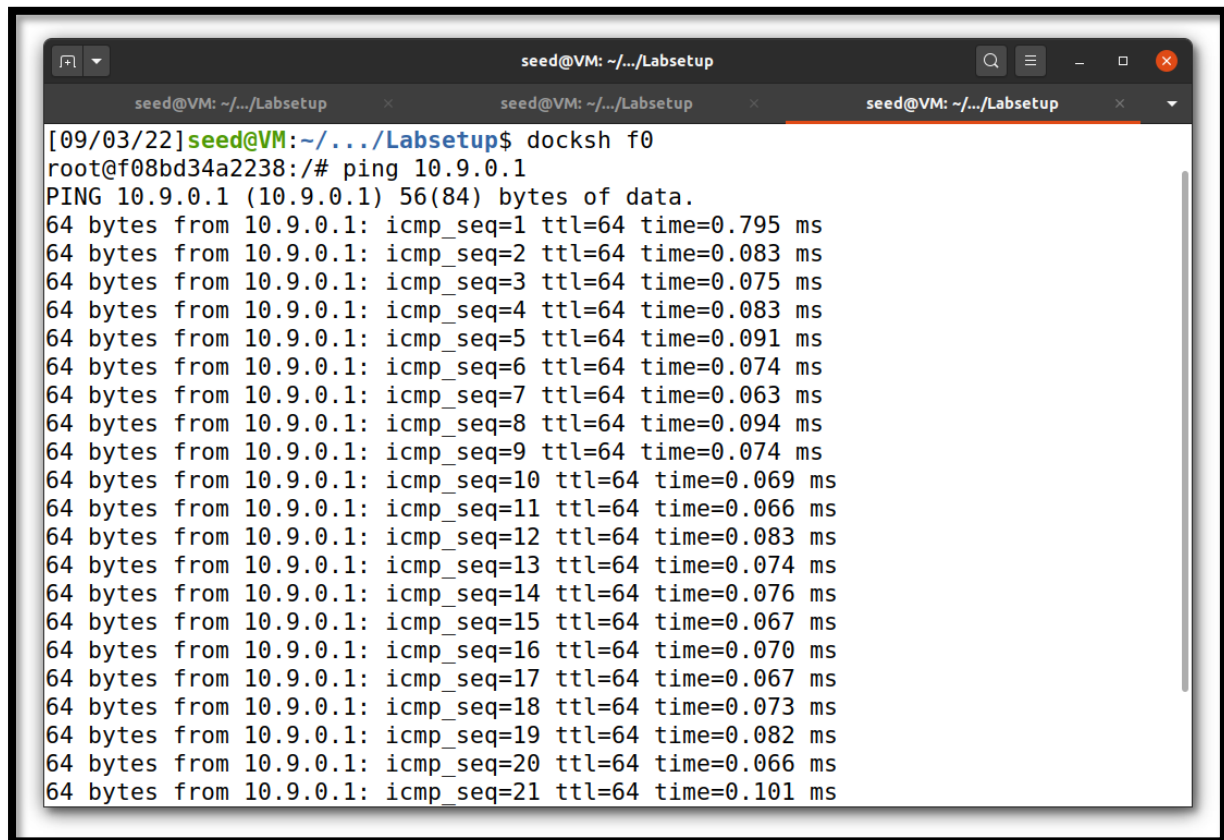
Attacker's Terminal:

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows the output of a command './sniff' run as root. The output displays a series of captured ICMP packets between two IP addresses: 10.9.0.5 and 10.9.0.1. The packets alternate in direction, with some from 10.9.0.5 to 10.9.0.1 and others from 10.9.0.1 to 10.9.0.5. The terminal window has a dark theme and standard window controls at the top.

```
seed@VM: ~/.../Labsetup
root@VM:/volumes# ./sniff
  From: 10.9.0.5
  To: 10.9.0.1
  Protocol: ICMP
  From: 10.9.0.1
  To: 10.9.0.5
  Protocol: ICMP
  From: 10.9.0.5
  To: 10.9.0.1
  Protocol: ICMP
  From: 10.9.0.1
  To: 10.9.0.5
  Protocol: ICMP
  From: 10.9.0.5
  To: 10.9.0.1
  Protocol: ICMP
  From: 10.9.0.1
  To: 10.9.0.5
  Protocol: ICMP
  From: 10.9.0.5
  To: 10.9.0.1
  Protocol: ICMP
```

We are printing out the source and destination IP addresses of each captured packet in the attacker's terminal.

Host A terminal:

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows a user running 'docksh f0' to switch to a container named 'f0'. Inside the container, the user is root and runs 'ping 10.9.0.1'. The output shows 21 successful ping requests, each with 64 bytes of data, a TTL of 64, and various response times ranging from approximately 0.063 ms to 0.101 ms.

```
[09/03/22]seed@VM:~/.../Labsetup$ docksh f0
root@f08bd34a2238:/# 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=0.795 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.083 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.075 ms
64 bytes from 10.9.0.1: icmp_seq=4 ttl=64 time=0.083 ms
64 bytes from 10.9.0.1: icmp_seq=5 ttl=64 time=0.091 ms
64 bytes from 10.9.0.1: icmp_seq=6 ttl=64 time=0.074 ms
64 bytes from 10.9.0.1: icmp_seq=7 ttl=64 time=0.063 ms
64 bytes from 10.9.0.1: icmp_seq=8 ttl=64 time=0.094 ms
64 bytes from 10.9.0.1: icmp_seq=9 ttl=64 time=0.074 ms
64 bytes from 10.9.0.1: icmp_seq=10 ttl=64 time=0.069 ms
64 bytes from 10.9.0.1: icmp_seq=11 ttl=64 time=0.066 ms
64 bytes from 10.9.0.1: icmp_seq=12 ttl=64 time=0.083 ms
64 bytes from 10.9.0.1: icmp_seq=13 ttl=64 time=0.074 ms
64 bytes from 10.9.0.1: icmp_seq=14 ttl=64 time=0.076 ms
64 bytes from 10.9.0.1: icmp_seq=15 ttl=64 time=0.067 ms
64 bytes from 10.9.0.1: icmp_seq=16 ttl=64 time=0.070 ms
64 bytes from 10.9.0.1: icmp_seq=17 ttl=64 time=0.067 ms
64 bytes from 10.9.0.1: icmp_seq=18 ttl=64 time=0.073 ms
64 bytes from 10.9.0.1: icmp_seq=19 ttl=64 time=0.082 ms
64 bytes from 10.9.0.1: icmp_seq=20 ttl=64 time=0.066 ms
64 bytes from 10.9.0.1: icmp_seq=21 ttl=64 time=0.101 ms
```

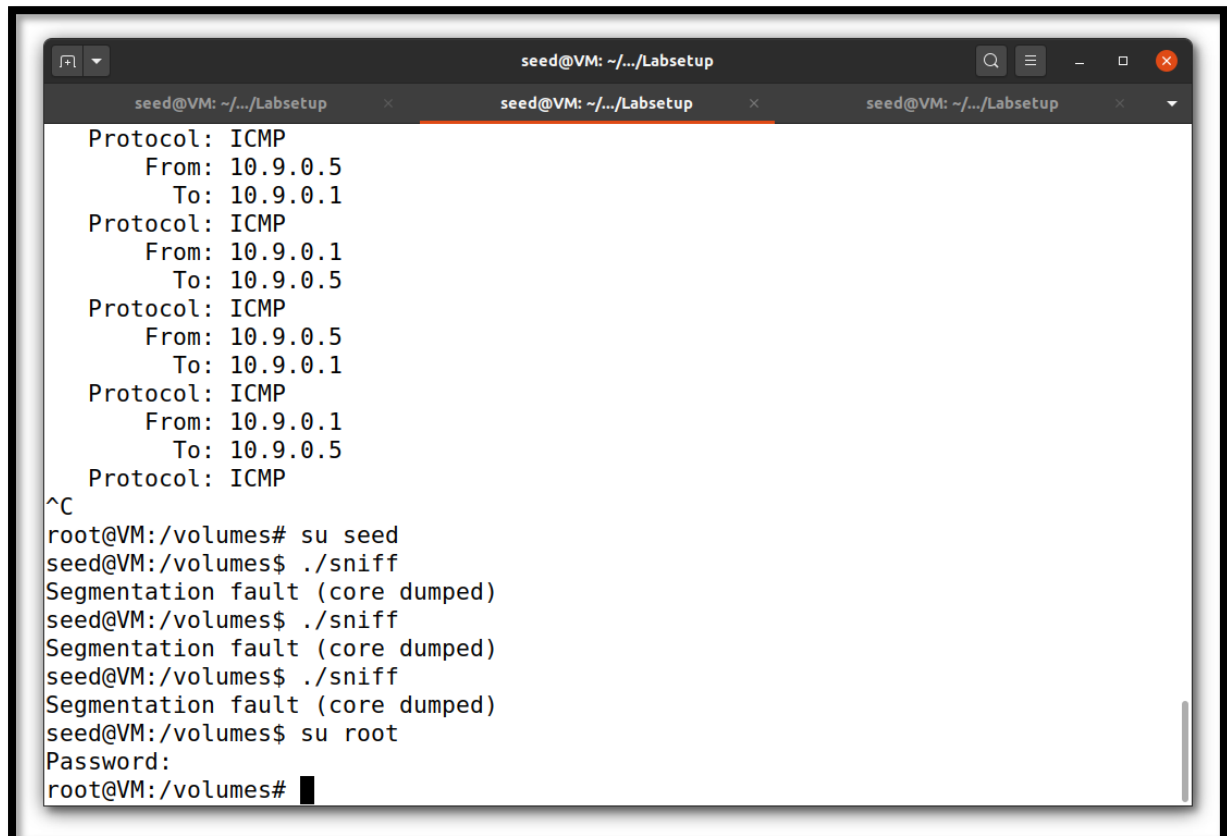
We are pinging 10.9.0.1 from the host A terminal.

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.

Ans: We are using PCAP library to sniff and spoof the packet across different hosts. It is also used to create packets and modify the packets.

Question 2: Why do you need the root privilege to run sniffer? Where does the program fail if executed without the root privilege?

Ans: Packets will not be sniffed and Segmentation fault arises.

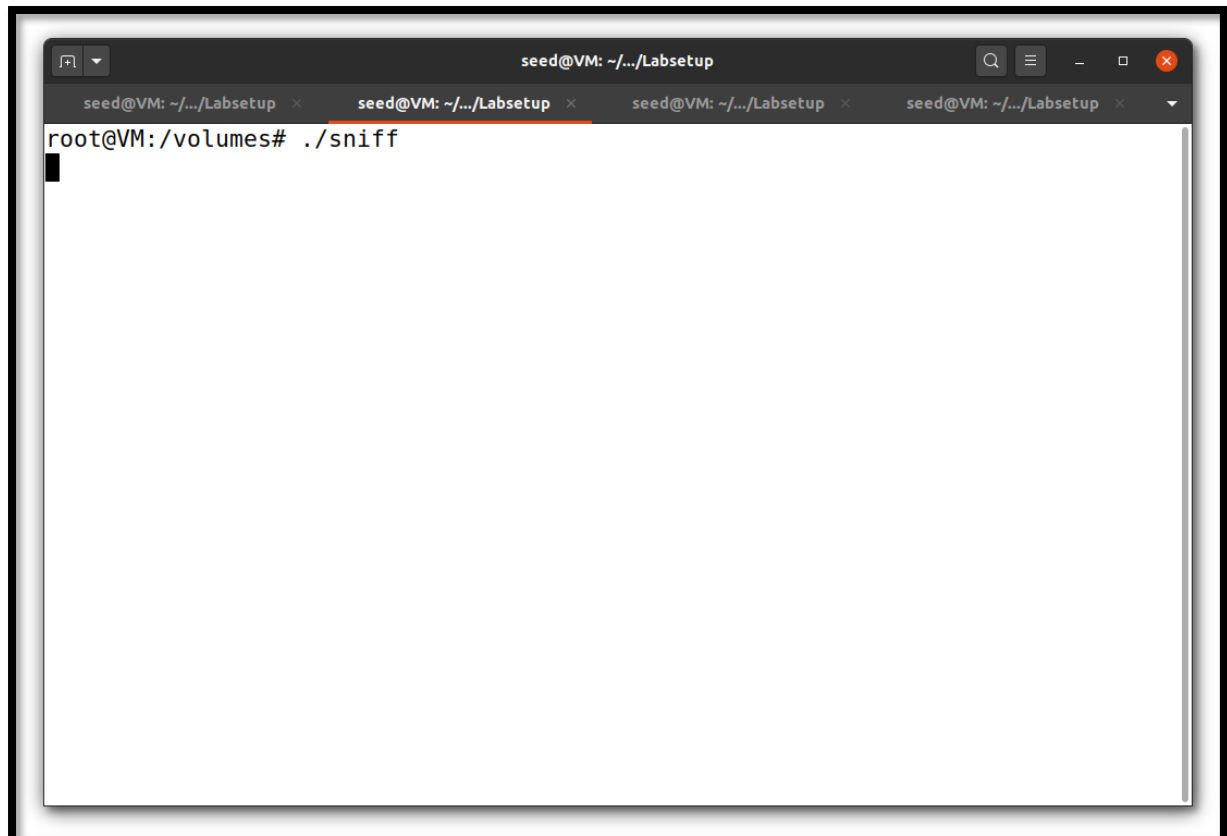


```
seed@VM: ~/.../Labsetup
Protocol: ICMP
  From: 10.9.0.5
  To: 10.9.0.1
Protocol: ICMP
  From: 10.9.0.1
  To: 10.9.0.5
Protocol: ICMP
  From: 10.9.0.5
  To: 10.9.0.1
Protocol: ICMP
  From: 10.9.0.1
  To: 10.9.0.5
Protocol: ICMP
^C
root@VM:/volumes# su seed
seed@VM:/volumes$ ./sniff
Segmentation fault (core dumped)
seed@VM:/volumes$ ./sniff
Segmentation fault (core dumped)
seed@VM:/volumes$ ./sniff
Segmentation fault (core dumped)
seed@VM:/volumes$ su root
Password:
root@VM:/volumes#
```

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?

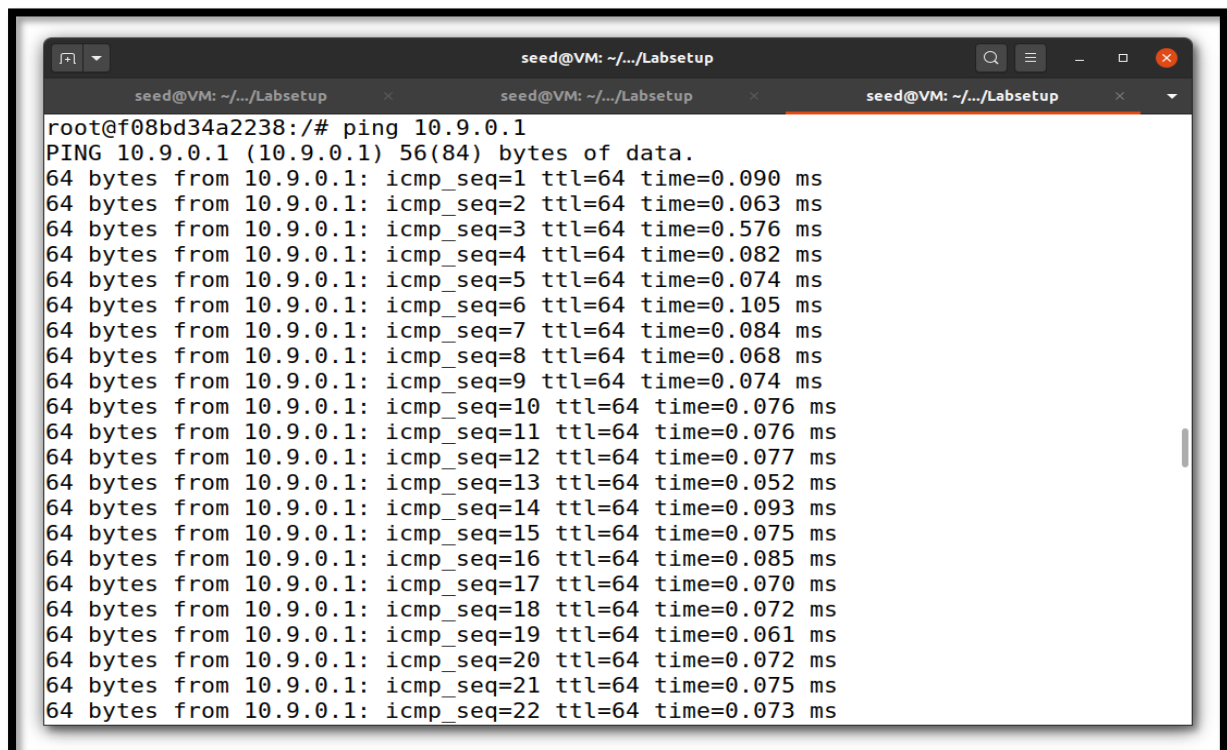
Ans: If we turn off the Promiscuous mode then we packets will not be sniffed.

Attacker's Terminal:

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows the command 'root@VM:/volumes# ./sniff' being entered, with a cursor on the line below it. The window has multiple tabs, all with the same title.

```
seed@VM: ~/.../Labsetup
root@VM:/volumes# ./sniff
```

Host A terminal:

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows the output of a 'ping 10.9.0.1' command, displaying 22 successful ping results with various TTL and time values. The window has multiple tabs, all with the same title.

```
seed@VM: ~/.../Labsetup
root@f08bd34a2238:/# 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=0.090 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.063 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.576 ms
64 bytes from 10.9.0.1: icmp_seq=4 ttl=64 time=0.082 ms
64 bytes from 10.9.0.1: icmp_seq=5 ttl=64 time=0.074 ms
64 bytes from 10.9.0.1: icmp_seq=6 ttl=64 time=0.105 ms
64 bytes from 10.9.0.1: icmp_seq=7 ttl=64 time=0.084 ms
64 bytes from 10.9.0.1: icmp_seq=8 ttl=64 time=0.068 ms
64 bytes from 10.9.0.1: icmp_seq=9 ttl=64 time=0.074 ms
64 bytes from 10.9.0.1: icmp_seq=10 ttl=64 time=0.076 ms
64 bytes from 10.9.0.1: icmp_seq=11 ttl=64 time=0.076 ms
64 bytes from 10.9.0.1: icmp_seq=12 ttl=64 time=0.077 ms
64 bytes from 10.9.0.1: icmp_seq=13 ttl=64 time=0.052 ms
64 bytes from 10.9.0.1: icmp_seq=14 ttl=64 time=0.093 ms
64 bytes from 10.9.0.1: icmp_seq=15 ttl=64 time=0.075 ms
64 bytes from 10.9.0.1: icmp_seq=16 ttl=64 time=0.085 ms
64 bytes from 10.9.0.1: icmp_seq=17 ttl=64 time=0.070 ms
64 bytes from 10.9.0.1: icmp_seq=18 ttl=64 time=0.072 ms
64 bytes from 10.9.0.1: icmp_seq=19 ttl=64 time=0.061 ms
64 bytes from 10.9.0.1: icmp_seq=20 ttl=64 time=0.072 ms
64 bytes from 10.9.0.1: icmp_seq=21 ttl=64 time=0.075 ms
64 bytes from 10.9.0.1: icmp_seq=22 ttl=64 time=0.073 ms
```

Task 2.1 B : Writing Filters

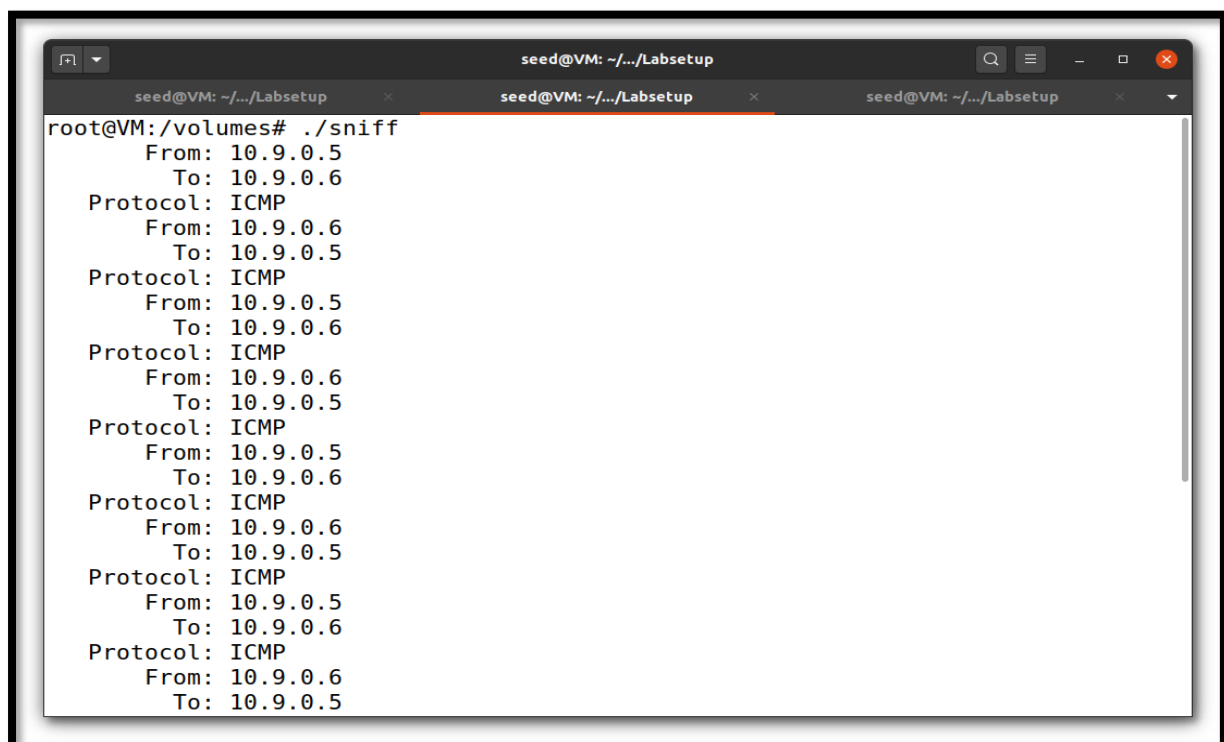
In this task we capture all ICMP packets between two hosts. In this task, we need to modify the pcap filter of the sniffer code. The filter will allow us to capture ICMP packets between two hosts . Complete the filter expression in the code and show that when we send ICMP packets to IP address 1 from IP address 2 using the ping command, the sniffer program captures the packets based on the filter. Observe the packets being sent using Wireshark.

Host VM:

```
[09/03/22] seed@VM: ~/.../Lab2$ gcc -o sniff Task2.1B-ICMP.c -lpcap
[09/03/22] seed@VM: ~/.../Lab2$ docker cp sniff 9753f0e34f43:/volumes
[09/03/22] seed@VM: ~/.../Lab2$
```

We are compiling the code in the host machine and then we are copying the compiled code inside the container.

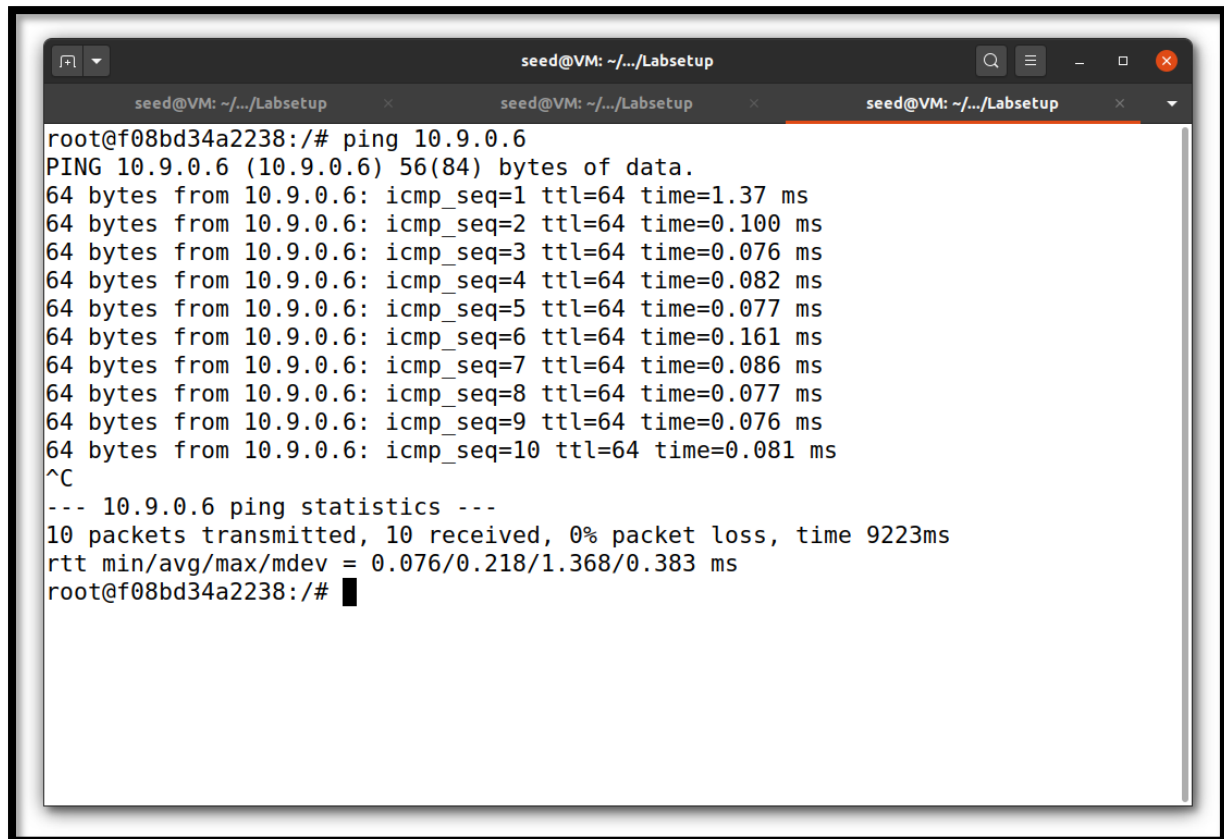
Attacker's Terminal:

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows the output of a command that has been executed, displaying a series of ICMP packet details. The output is as follows:

```
root@VM:/volumes# ./sniff
From: 10.9.0.5
To: 10.9.0.6
Protocol: ICMP
From: 10.9.0.6
To: 10.9.0.5
Protocol: ICMP
From: 10.9.0.5
To: 10.9.0.6
Protocol: ICMP
From: 10.9.0.6
To: 10.9.0.5
Protocol: ICMP
From: 10.9.0.5
To: 10.9.0.6
Protocol: ICMP
From: 10.9.0.6
To: 10.9.0.5
Protocol: ICMP
From: 10.9.0.5
To: 10.9.0.6
Protocol: ICMP
From: 10.9.0.6
To: 10.9.0.5
Protocol: ICMP
From: 10.9.0.5
To: 10.9.0.6
Protocol: ICMP
From: 10.9.0.6
To: 10.9.0.5
```

We are capturing all the ICMP packets moving across the two hosts and displaying the source and destination address

Host A terminal:

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows a root user at IP f08bd34a2238 running a ping command to 10.9.0.6. The output displays 10 successful ping responses with varying times and TTL values. After pressing Ctrl-C, it shows ping statistics: 10 packets transmitted, 10 received, 0% packet loss, and a total time of 9223ms. The terminal prompt returns to root@f08bd34a2238:/#.

```
seed@VM: ~/.../Labsetup
root@f08bd34a2238:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=1.37 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.100 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.076 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.082 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.077 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.161 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.086 ms
64 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.077 ms
64 bytes from 10.9.0.6: icmp_seq=9 ttl=64 time=0.076 ms
64 bytes from 10.9.0.6: icmp_seq=10 ttl=64 time=0.081 ms
^C
--- 10.9.0.6 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9223ms
rtt min/avg/max/mdev = 0.076/0.218/1.368/0.383 ms
root@f08bd34a2238:/#
```

We are pinging the IP address 10.9.0.6 from the host A terminal.

Capture the TCP packets that have a destination port range from to sort 10 –100

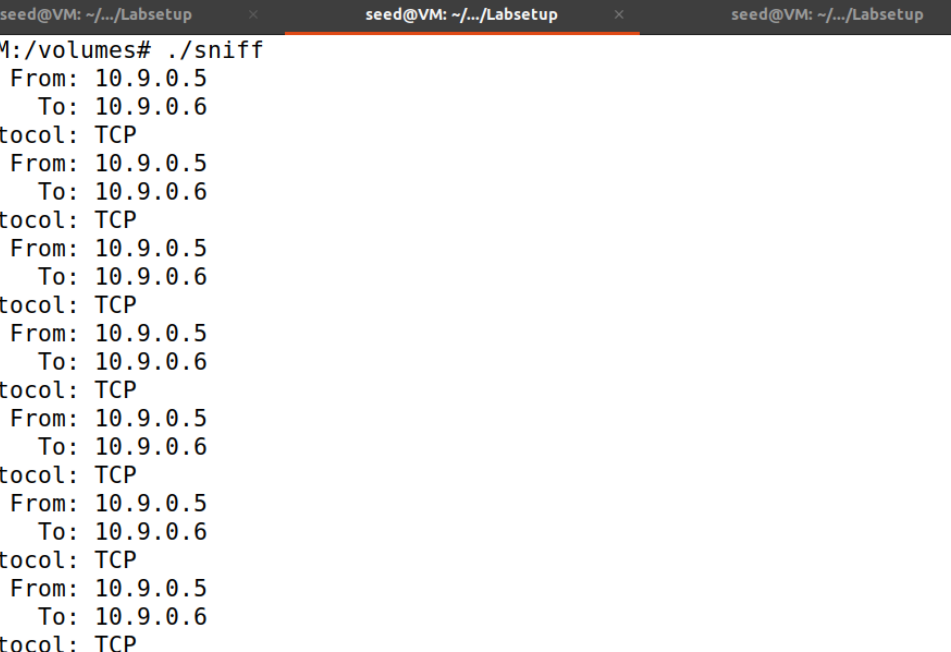
In this task we capture all TCP packets with a destination port range 10-100. Below we have the filter expression required to filter for TCP packets in a given port range.

We send FTP (runs over TCP) packets to the destination machine. As telnet runs over port 21, we should be able to capture all the packets sent with destination port 21.

Host VM:

```
[09/03/22] seed@VM:~/.../Lab2$ gcc -o sniff Task2.1B-TCP.c -lpcap
[09/03/22] seed@VM:~/.../Lab2$ docker cp sniff 9753f0e34f43:/volumes
```

Attacker's VM terminal:



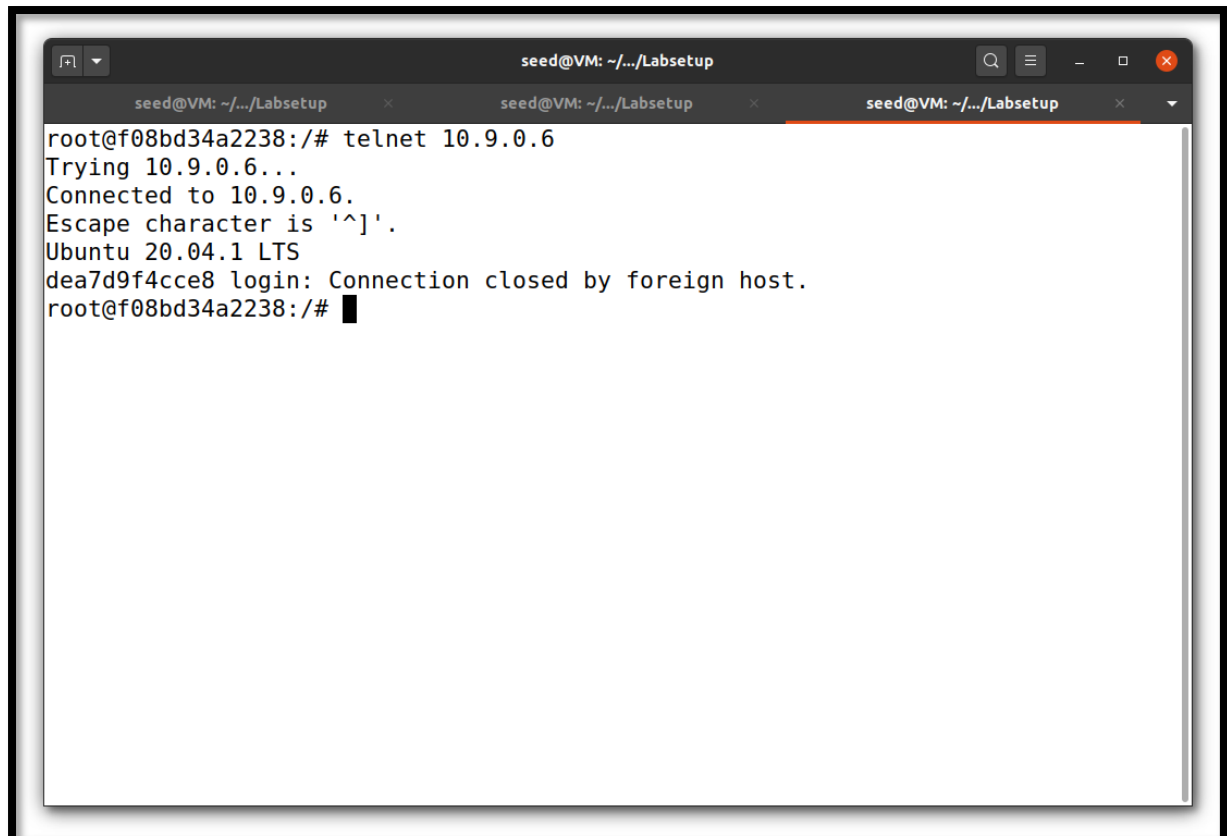
The screenshot shows a terminal window titled "seed@VM: ~/.../Labsetup". The prompt is "root@VM:/volumes#". The command executed is "./sniff". The output displays a series of network packets, all identified as TCP, with the following details:

- From: 10.9.0.5
- To: 10.9.0.6
- Protocol: TCP

This sequence of information is repeated for eight different packets, indicating a continuous stream of traffic from 10.9.0.5 to 10.9.0.6 over a TCP connection.

We are capturing all the TCP packets moving across the two hosts whose destination port ranges from 10-100 and displaying the source and destination address.

Host A Terminal:

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows a user at the prompt 'root@f08bd34a2238:/' typing 'telnet 10.9.0.6'. The output shows the connection attempt: 'Trying 10.9.0.6...', 'Connected to 10.9.0.6.', 'Escape character is '^_'.', 'Ubuntu 20.04.1 LTS', and 'dea7d9f4cce8 login: Connection closed by foreign host.' The prompt returns to 'root@f08bd34a2238:/' with a cursor. The terminal window has three tabs, all with the same title, and standard window controls at the top right.

```
seed@VM: ~/.../Labsetup
root@f08bd34a2238:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^_'.
Ubuntu 20.04.1 LTS
dea7d9f4cce8 login: Connection closed by foreign host.
root@f08bd34a2238:/#
```

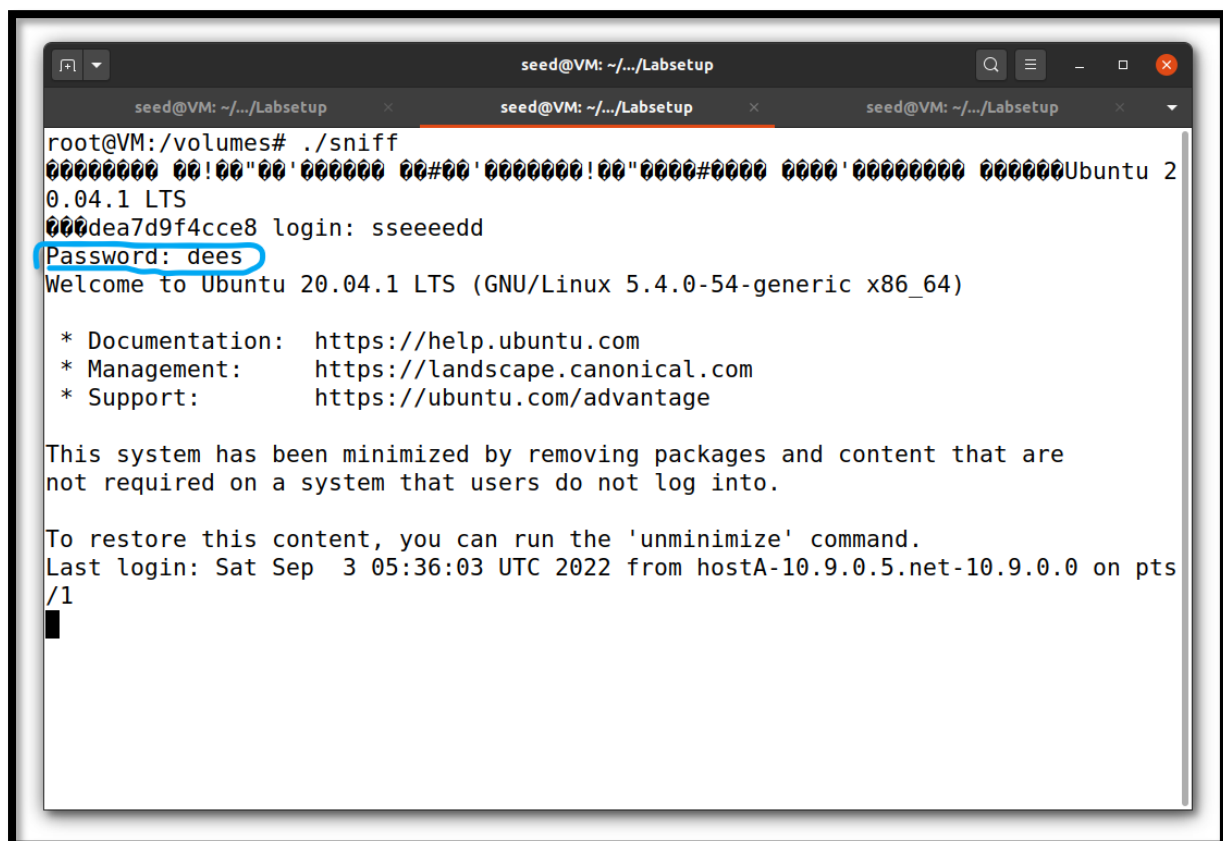
We send FTP (runs over TCP) packets to the destination machine. As telnet runs over port 21, we should be able to capture all the packets sent with destination port 21.

Task 2.1 C : Sniffing Passwords

Host VM:

```
[09/03/22] seed@VM: ~/.../Lab2$ gcc -o sniff Task2.1C.c -lpcap
[09/03/22] seed@VM: ~/.../Lab2$ docker cp sniff 9753f0e34f43:/volumes
```

Attacker's Host terminal:



```
seed@VM: ~/.../Labsetup
root@VM:/volumes# ./sniff
00000000 00!00"00'000000 00#00'000000!00"0000#0000 0000'00000000 000000Ubuntu 2
0.04.1 LTS
000dea7d9f4cce8 login: sseeedd
Password: dees
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

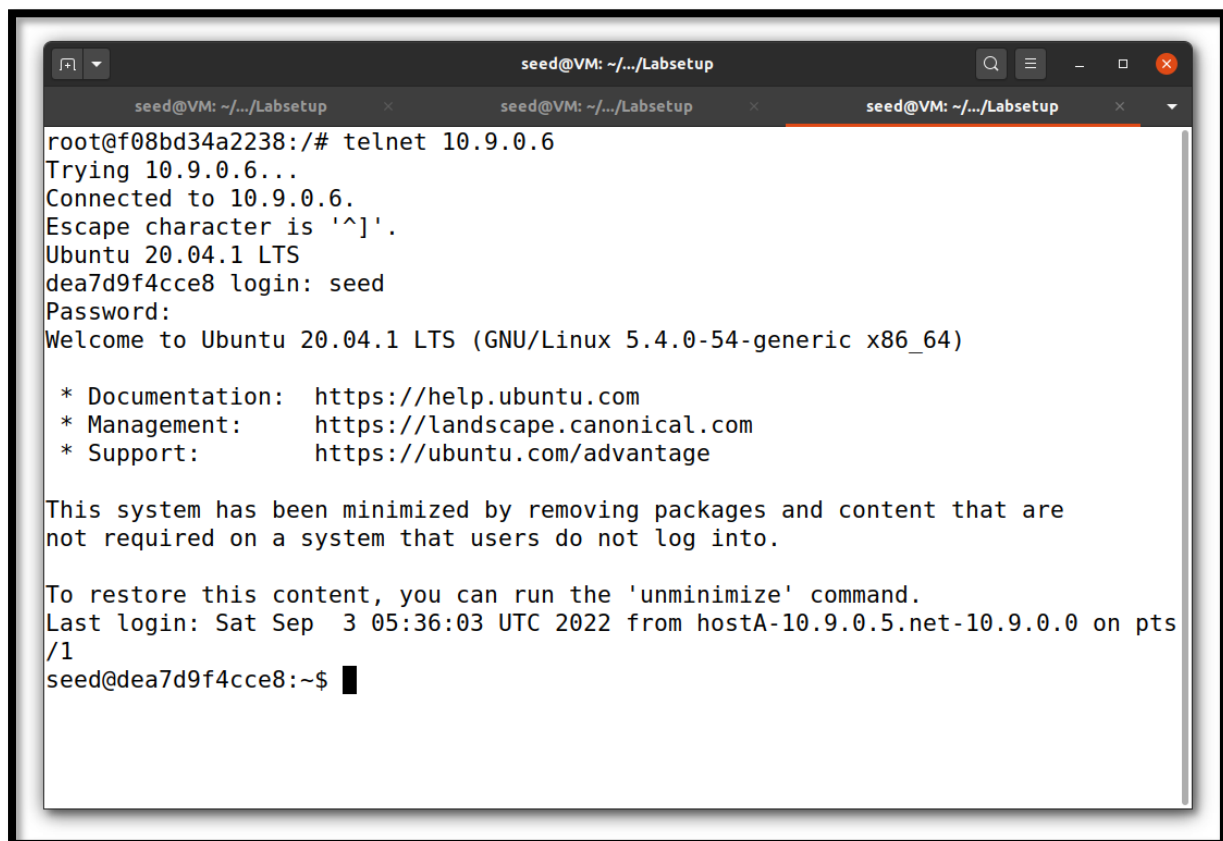
* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

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: Sat Sep  3 05:36:03 UTC 2022 from hostA-10.9.0.5-net-10.9.0.0 on pts
/1
```

We are capturing the password when somebody is using telnet on the network that we are monitoring. We are printing out the entire data part, and then manually mark where the password is (marked in blue colour).

Host A Terminal:

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows a telnet session initiated from a root user on IP f08bd34a2238 to 10.9.0.6. The connection is successful, and the user 'seed' logs in to an Ubuntu 20.04.1 LTS system. The terminal displays the system's welcome message, including documentation, management, and support links. It also notes that the system is minimized and provides instructions on how to restore content using the 'unminimize' command. The last login is recorded as Saturday, September 3, 2022, at 05:36:03 UTC from hostA-10.9.0.5.net-10.9.0.0 on pts/1. The prompt is 'seed@dea7d9f4cce8:~\$' with a cursor.

```
seed@f08bd34a2238:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
dea7d9f4cce8 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * 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: Sat Sep  3 05:36:03 UTC 2022 from hostA-10.9.0.5.net-10.9.0.0 on pts
/1
seed@dea7d9f4cce8:~$
```

In host we are using telnet in the network that the attacker is keeping an eye on. Apparently all these data will be sniffed by the attacker.

Task 2.2 Spoofing

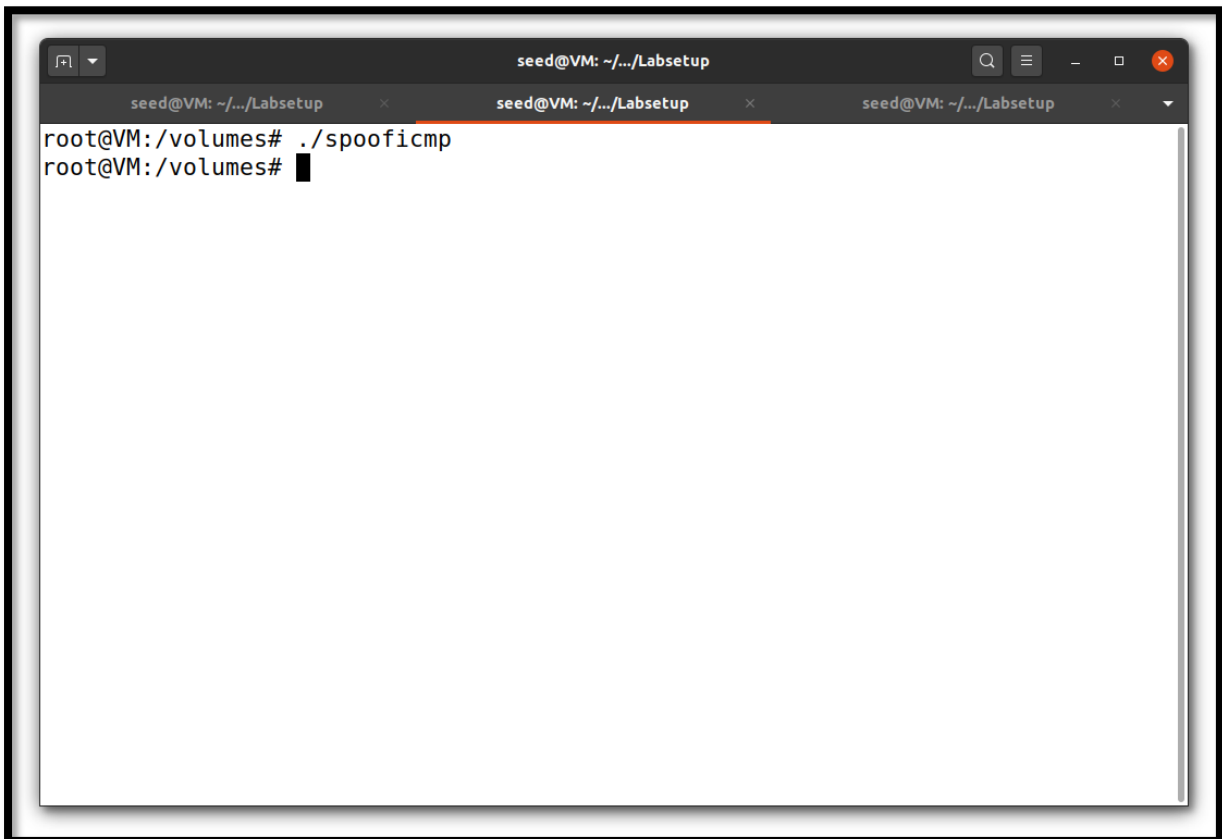
The objective of this task is to create raw sockets and send spoof packets to the user/victim machine raw sockets give programmers the absolute control over the packet construction.

Task 2.2 B: Spoof an ICMP Echo Request

Host VM:

```
[09/03/22] seed@VM: ~/.../Task 2.2 and Task 2.3$ gcc -o spooficmp checksum.c  
spoof_icmp.c send_raw_ip_packet.c -lpcap  
[09/03/22] seed@VM: ~/.../Task 2.2 and Task 2.3$ docker cp spooficmp 9753f0e3  
4f43:/volumes  
[09/03/22] seed@VM: ~/.../Task 2.2 and Task 2.3$
```

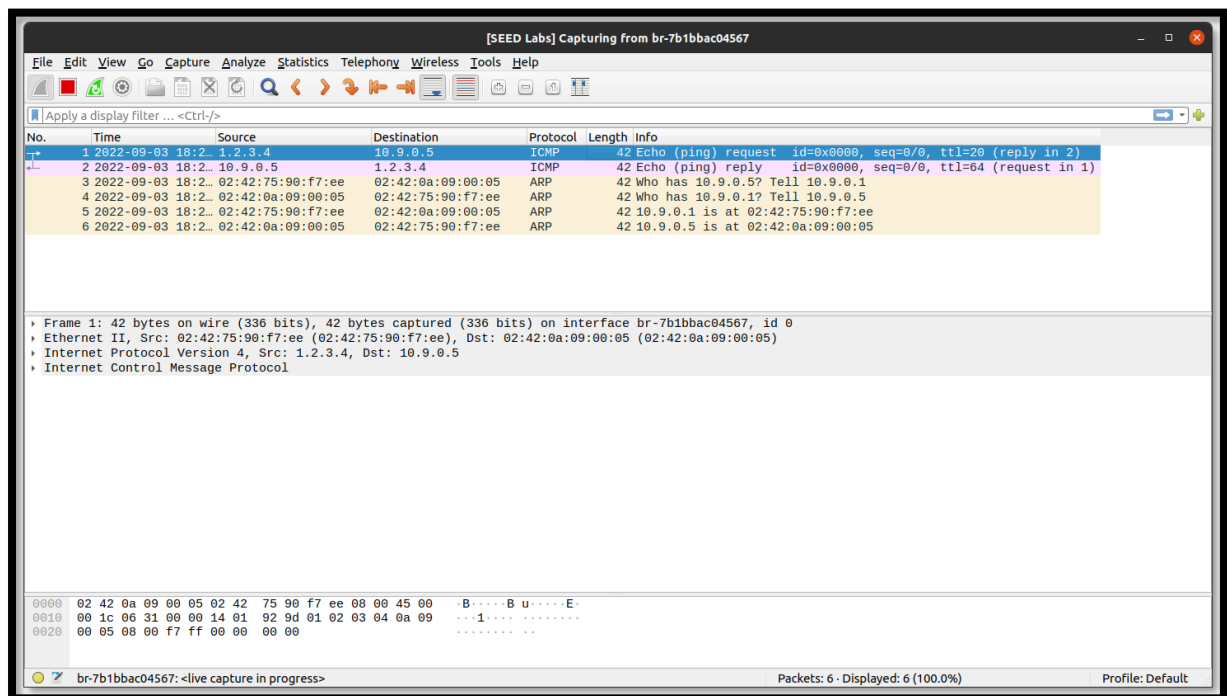
Attacker's Terminal:



We are Spoofing an ICMP echo request packet on behalf of another machine (i.e., using another

machine's IP address as its source IP address). This packet should be sent to a remote machine on the Internet (the machine must be alive).

Wireshark Screenshot:



Question 4: Using the raw socket programming, do you have to calculate the checksum for the IP header?

Ans: Yes we are calculating the checksum while using the raw socket programming.

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?

Ans: If we did not run using root privileges, then we will not be able to sniff any packet because segmentation error occurs when we run the command in the attacker's terminal.

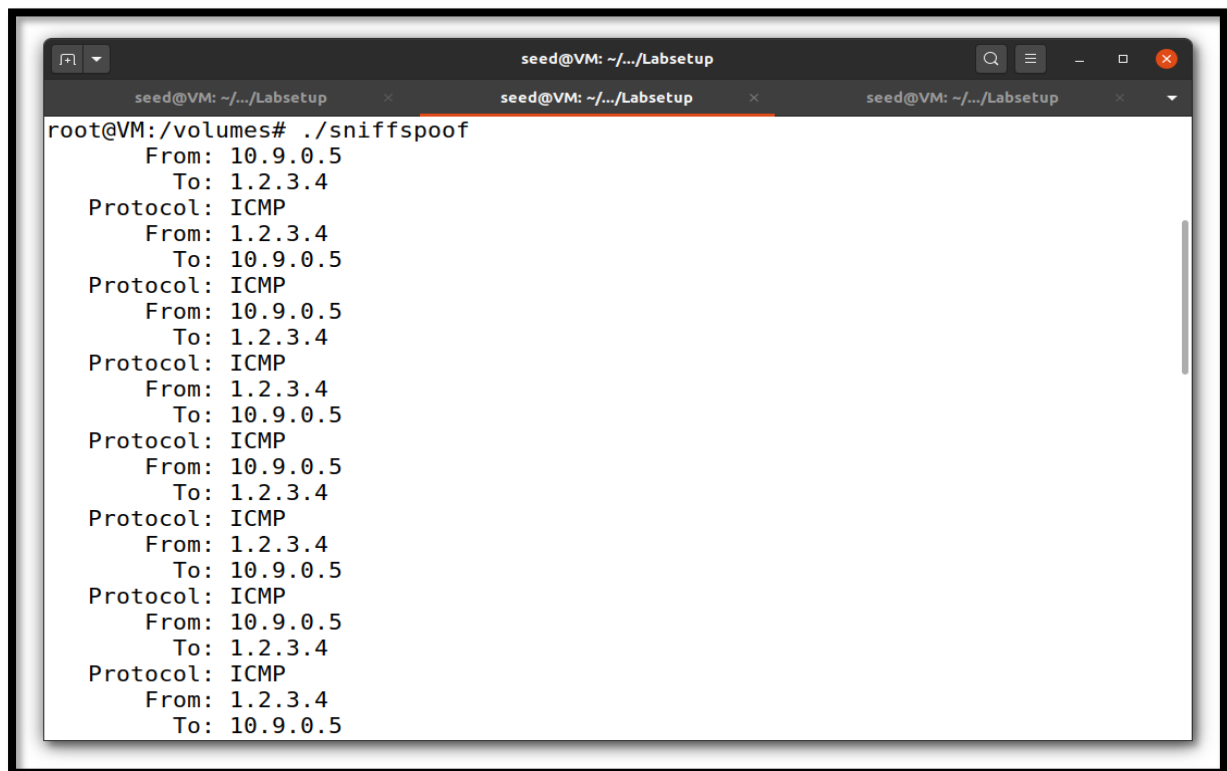
Task 2.3 Sniff and then Spoof

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.

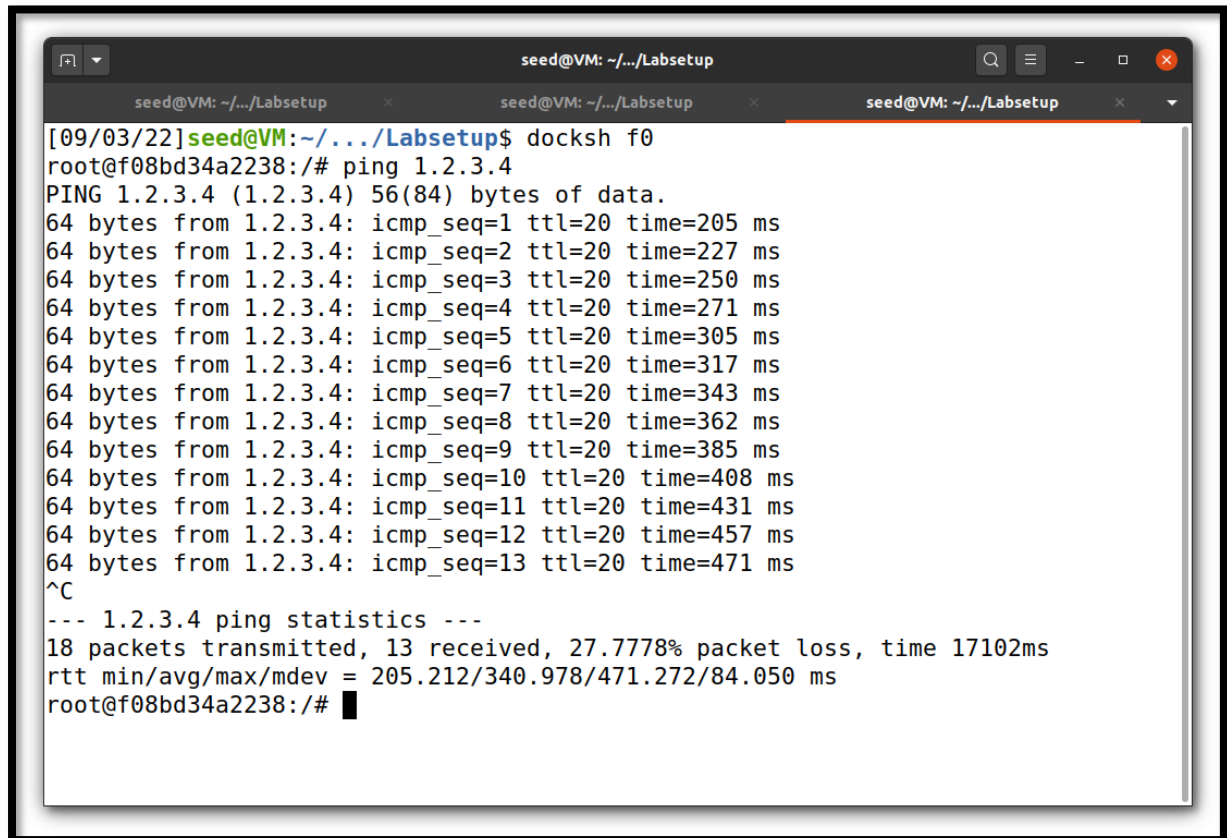
Attacker's Terminal:

A screenshot of a terminal window titled 'seed@VM: ~/.../Labsetup'. The terminal shows the output of the command './sniffspoof' run as root. The output displays a series of ICMP packets between two hosts. The first packet is from 10.9.0.5 to 1.2.3.4, and the second is from 1.2.3.4 to 10.9.0.5. This pattern repeats for a total of six packets, showing a continuous exchange of traffic.

```
root@VM:/volumes# ./sniffspoof
  From: 10.9.0.5
  To: 1.2.3.4
  Protocol: ICMP
  From: 1.2.3.4
  To: 10.9.0.5
  Protocol: ICMP
  From: 10.9.0.5
  To: 1.2.3.4
  Protocol: ICMP
  From: 1.2.3.4
  To: 10.9.0.5
  Protocol: ICMP
  From: 10.9.0.5
  To: 1.2.3.4
  Protocol: ICMP
  From: 1.2.3.4
  To: 10.9.0.5
  Protocol: ICMP
  From: 10.9.0.5
  To: 1.2.3.4
  Protocol: ICMP
  From: 1.2.3.4
  To: 10.9.0.5
```

Here we can see the source and destination IP address. We can observe clearly that both request and reply are getting transferred between each host.

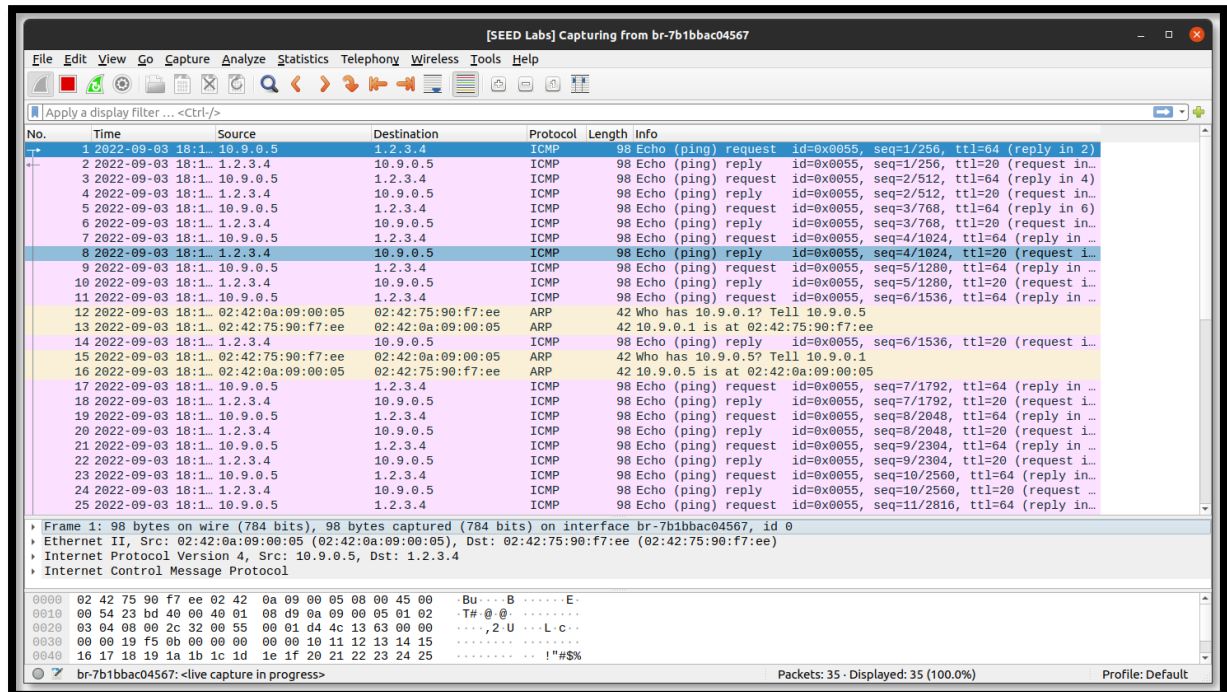
Host A Terminal:



```
seed@VM: ~/.../Labsetup
[09/03/22]seed@VM:~/.../Labsetup$ docksh f0
root@f08bd34a2238:/# 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=20 time=205 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=20 time=227 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=20 time=250 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=20 time=271 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=20 time=305 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=20 time=317 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=20 time=343 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=20 time=362 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=20 time=385 ms
64 bytes from 1.2.3.4: icmp_seq=10 ttl=20 time=408 ms
64 bytes from 1.2.3.4: icmp_seq=11 ttl=20 time=431 ms
64 bytes from 1.2.3.4: icmp_seq=12 ttl=20 time=457 ms
64 bytes from 1.2.3.4: icmp_seq=13 ttl=20 time=471 ms
^C
--- 1.2.3.4 ping statistics ---
18 packets transmitted, 13 received, 27.7778% packet loss, time 17102ms
rtt min/avg/max/mdev = 205.212/340.978/471.272/84.050 ms
root@f08bd34a2238:/#
```

Here the host is pinging the IP address 1.2.3.4 which is an unknown IP address.

Wireshark Terminal:



The attacker is spoofing the packet from the host and gets those packets from raw socket. When this happens the host machine thinks that the IP address really exists. The attacker who sniffs the packet will change the data and again sends the echo reply back to the host. This repeats every time the host tries to contact this unknown IP address.