Sniffing and Spoofing using PCAP Library

Submitted by: Mahika Gupta

SRN: PES1UG20CS243

Date: 02/09/2022

Table of Contents:

Lab Environment Setup

Please download the Labsetup.zip file from the link given below :
https://seedsecuritylabs.org/Labs_20.04/Networking/Sniffing_Spoofing/ Follow

the instructions in the **lab setup document** to set up the lab environment.

In this lab, we will use three machines that are connected to the same LAN. We can either use three VMs or three containers. Figure 1 depicts the lab environment setup using containers. We will do all the attacks on the attacker container, while using the other containers as the user machines.

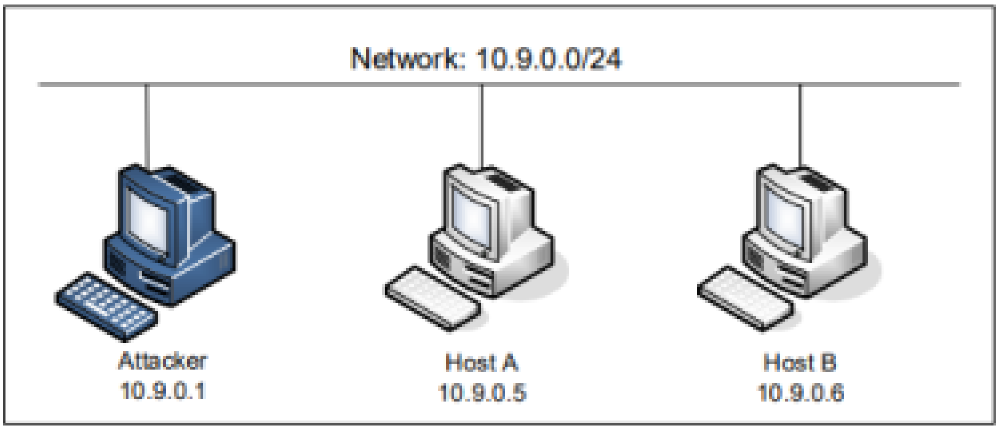


Figure 1 : Lab environment setup

```
[09/02/22]seed@VM:~/.../Labsetup$ docker-compose build
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
[09/02/22]seed@VM:~/.../Labsetup$ docker-compose up
Starting hostA-10.9.0.5 ... done
Starting seed-attacker  ... done
Starting hostB-10.9.0.6 ... done
Attaching to seed-attacker, hostB-10.9.0.6, hostA-10.9.0.5
hostA-10.9.0.5 |  * Starting internet superserver inetd                  [ OK ]
hostB-10.9.0.6 |  * Starting internet superserver inetd                  [ OK ]
```


Lab Task Set-2: Writing Programs to Sniff and

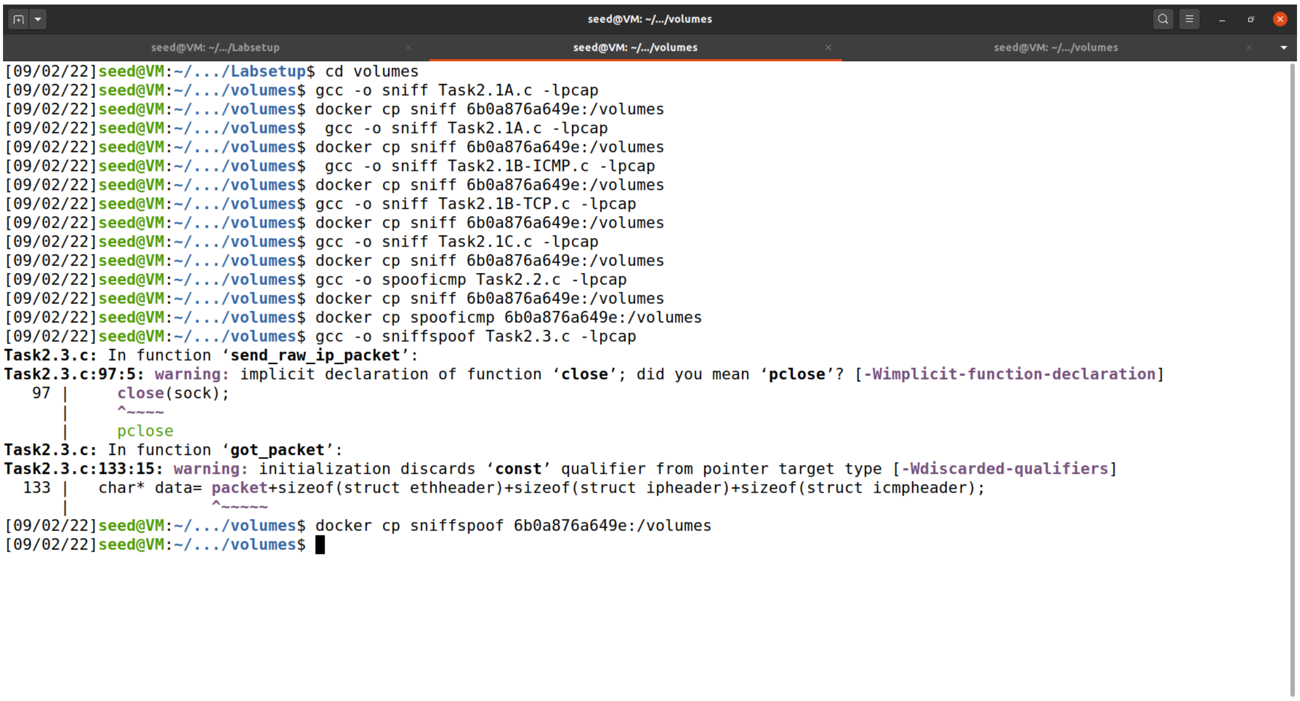
Spoof Packets using pcap (C programs)

IMPORTANT

For this set up of tasks, you should compile the C code inside the host VM, and then run the code inside the container. You can use the "docker cp" command to copy a file from the host VM to a container. See the following example (there is no need to type the docker ID in full):

Commands:

> **# docker ps**
>
> // Copy a.out to the seed-attacker container's /volumes folder
>
> **# docker cp [File Name to be copied] [Docker container ID]:/volumes**



```
[09/02/22]seed@VM:~/.../Labsetup$ cd volumes
[09/02/22]seed@VM:~/.../volumes$ gcc -o sniff Task2.1A.c -lpcap
[09/02/22]seed@VM:~/.../volumes$ docker cp sniff 6b0a876a649e:/volumes
[09/02/22]seed@VM:~/.../volumes$  gcc -o sniff Task2.1A.c -lpcap
[09/02/22]seed@VM:~/.../volumes$ docker cp sniff 6b0a876a649e:/volumes
[09/02/22]seed@VM:~/.../volumes$  gcc -o sniff Task2.1B-ICMP.c -lpcap
[09/02/22]seed@VM:~/.../volumes$ docker cp sniff 6b0a876a649e:/volumes
[09/02/22]seed@VM:~/.../volumes$ gcc -o sniff Task2.1B-TCP.c -lpcap
[09/02/22]seed@VM:~/.../volumes$ docker cp sniff 6b0a876a649e:/volumes
[09/02/22]seed@VM:~/.../volumes$ gcc -o sniff Task2.1C.c -lpcap
[09/02/22]seed@VM:~/.../volumes$ docker cp sniff 6b0a876a649e:/volumes
[09/02/22]seed@VM:~/.../volumes$ gcc -o spooficmp Task2.2.c -lpcap
[09/02/22]seed@VM:~/.../volumes$ docker cp sniff 6b0a876a649e:/volumes
[09/02/22]seed@VM:~/.../volumes$ docker cp spooficmp 6b0a876a649e:/volumes
[09/02/22]seed@VM:~/.../volumes$ gcc -o sniffspoof Task2.3.c -lpcap
Task2.3.c: In function ‘send_raw_ip_packet’:
Task2.3.c:97:5: warning: implicit declaration of function ‘close’; did you mean ‘pclose’? [-Wimplicit-function-declaration]
   97 |     close(sock);
      |     ^~~~~
      |     pclose
Task2.3.c: In function ‘got_packet’:
Task2.3.c:133:15: warning: initialization discards ‘const’ qualifier from pointer target type [-Wdiscarded-qualifiers]
  133 |   char* data= packet+sizeof(struct ethheader)+sizeof(struct ipheader)+sizeof(struct icmpheader);
      |               ^~~~~~
[09/02/22]seed@VM:~/.../volumes$ docker cp sniffspoof 6b0a876a649e:/volumes
[09/02/22]seed@VM:~/.../volumes$ 
```

Sniffer programs can be easily written using the pcap library. With pcap, the task of sniffers becomes invoking a simple sequence of procedures in the pcap library. At the end of the sequence, packets will be put in a buffer for further processing as soon as they are captured. All the details of packet capturing are handled by the pcap library.

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

In this task, students need to write a sniffer program to print out the source and destination IP addresses of each captured packet. Students can type in the above code or download the sample code from the SEED book's website (https://www.handsonsecurity.net/figurecode.html). Students should provide screenshots as evidence to show that their sniffer program can run successfully and produce expected results.

Since we can not compile the c programs within the containers, we must compile them in the host Vm and move them into the containers where we will execute them.

Check the Lab setup manual for instructions on finding the interface for the attacker machine. Change the interface value in the code to the interface of the attacker machine.

On the host VM :

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

 # docker cp sniff [Attacker machine docker container ID]:/volumes

On the Attacker container run the command:

 # ./sniff

On Host A terminal :

 # ping 10.9.0.1

Provide screenshots of your observations.

```
[09/02/22]seed@VM:~/.../volumes$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED          STATUS         PORTS
      NAMES
6b0a876a649e   handsonsecurity/seed-ubuntu:large   "/bin/sh -c /bin/bash"   10 minutes ago   Up 5 minutes
      seed-attacker
c785a4727e2c   handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init…"   22 hours ago     Up 5 minutes
      hostB-10.9.0.6
47f79633855e   handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init…"   22 hours ago     Up 5 minutes
      hostA-10.9.0.5
[09/02/22]seed@VM:~/.../volumes$ docksh 47f79633855e
@PES1UG20CS243_mahika:/# 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=0.290 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.122 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.123 ms
64 bytes from 10.9.0.1: icmp_seq=4 ttl=64 time=0.114 ms
64 bytes from 10.9.0.1: icmp_seq=5 ttl=64 time=0.115 ms
64 bytes from 10.9.0.1: icmp_seq=6 ttl=64 time=0.121 ms
^C
--- 10.9.0.1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5102ms
rtt min/avg/max/mdev = 0.114/0.147/0.290/0.063 ms
@PES1UG20CS243_mahika:/#
```

```
[09/02/22]seed@VM:~/.../Labsetup$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED         STATUS         PORTS     NAMES
6b0a876a649e   handsonsecurity/seed-ubuntu:large   "/bin/sh -c /bin/bash"   7 minutes ago   Up 3 minutes             seed-attacker
c785a4727e2c   handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init…"   22 hours ago    Up 3 minutes             hostB-10.9.0.6
47f79633855e   handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init…"   22 hours ago    Up 3 minutes             hostA-10.9.0.5
[09/02/22]seed@VM:~/.../Labsetup$ docksh 6b0a876a649e
seed-attacker:PES1UG20CS243:Mahika:/
$>./sniff
bash: ./sniff: No such file or directory
seed-attacker:PES1UG20CS243:Mahika:/
$>cd volumes/
seed-attacker:PES1UG20CS243:Mahika:/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
       From: 10.9.0.1
         To: 10.9.0.5
   Protocol: ICMP
       From: 10.9.0.5
         To: 10.9.0.1
   Protocol: ICMP
```

The attacker container sniffs all the ICMP packets received by the host A on pinging 10.9.0.1.

In addition, please answer the following questions:

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 a detailed explanation like the one in the tutorial.

ANSWER: The function calls made in sequence are:

- **pcap_open_live() - setting up the ethernet interface.**
- **pcap_compile() - to compile the filter expression into bpf pseudo code**
- **pcap_loop() - to capture packets.**
- **pcap_close(handle) - to close the handle.**

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

ANSWER: The raw socket can be created and used only in root user mode as root privileges are needed to access different network interfaces. Program fails when handle is created :

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

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

Provide a screenshot of your observations.

```
seed-attacker:PES1UG20CS243:Mahika:/volumes
$>su seed
seed@VM:/volumes$ ./sniff
Segmentation fault (core dumped)
seed@VM:/volumes$ 
```

The Program shows an error and does not run when you come out of root user mode.

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?

Change the code given in line 69 of Task2.1A.c file to the following :

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

On the host VM :

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

 # docker cp sniff [Attacker machine docker container ID]:/volumes

On the Attacker terminal run the command:

 # ./sniff

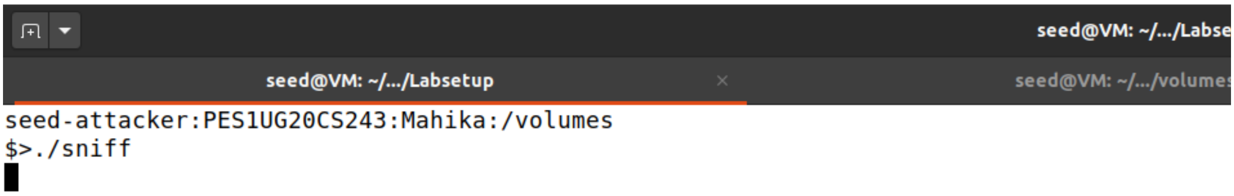
On Host A terminal :

 # ping 10.9.0.6

Provide screenshots of your observations.

```
// Step 1: Open live pcap session on NIC with name br-****
handle = pcap_open_live("br-f9a3126fae5c", BUFSIZ, 0, 1000, errbuf);
```

```
@PES1UG20CS243_mahika:/# 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=0.271 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.131 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.119 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.117 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.158 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.129 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.127 ms
64 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.134 ms
^C
--- 10.9.0.6 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7168ms
rtt min/avg/max/mdev = 0.117/0.148/0.271/0.047 ms
@PES1UG20CS243_mahika:/# 
```



There is no output in the attacker's machine, as promiscuous mode is off, which means that when the NIC receives frames that are not meant for the attacker machine, it drops those frames and only lets through the frames addressed to the attacker. Hence the attacker cannot sniff the frames sent/received by the host machine.

Task 2.1 B : Writing Filters

Capture the ICMP packets between two specific hosts

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.

Change the interface value in the code to the interface of the attacker machine as done in previous tasks.

On the host VM :

 # gcc -o sniff Task2.1B-ICMP.c -lpcap

 # docker cp sniff [Attacker machine docker container ID]:/volumes

On the Attacker terminal run the command:

 # ./sniff

In the host A machine ping any ip address

 # ping 10.9.0.6

Provide screenshots of your observations.

```
seed-attacker:PES1UG20CS243:Mahika:/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
   Protocol: ICMP
```

ICMP packets sent between hosts 10.9.0.5 and 10.9.0.6 are captured and sniffed by the attacker machine.

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.

Change the interface value in the code to the interface of the attacker machine as done in previous tasks.

On the host VM :

 # gcc -o sniff Task2.1B-TCP.c -lpcap

 # docker cp sniff [Attacker machine docker container ID]:/volumes

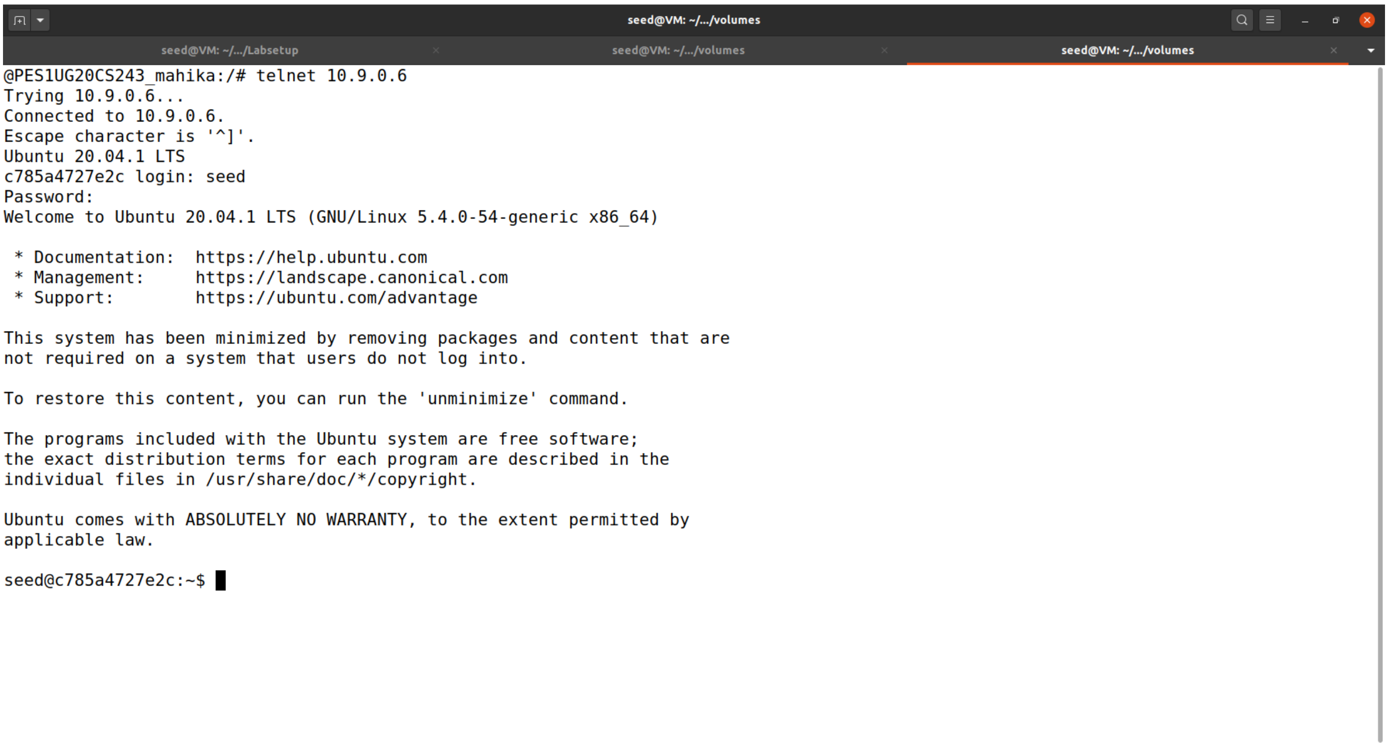
On Attacker Machine terminal :

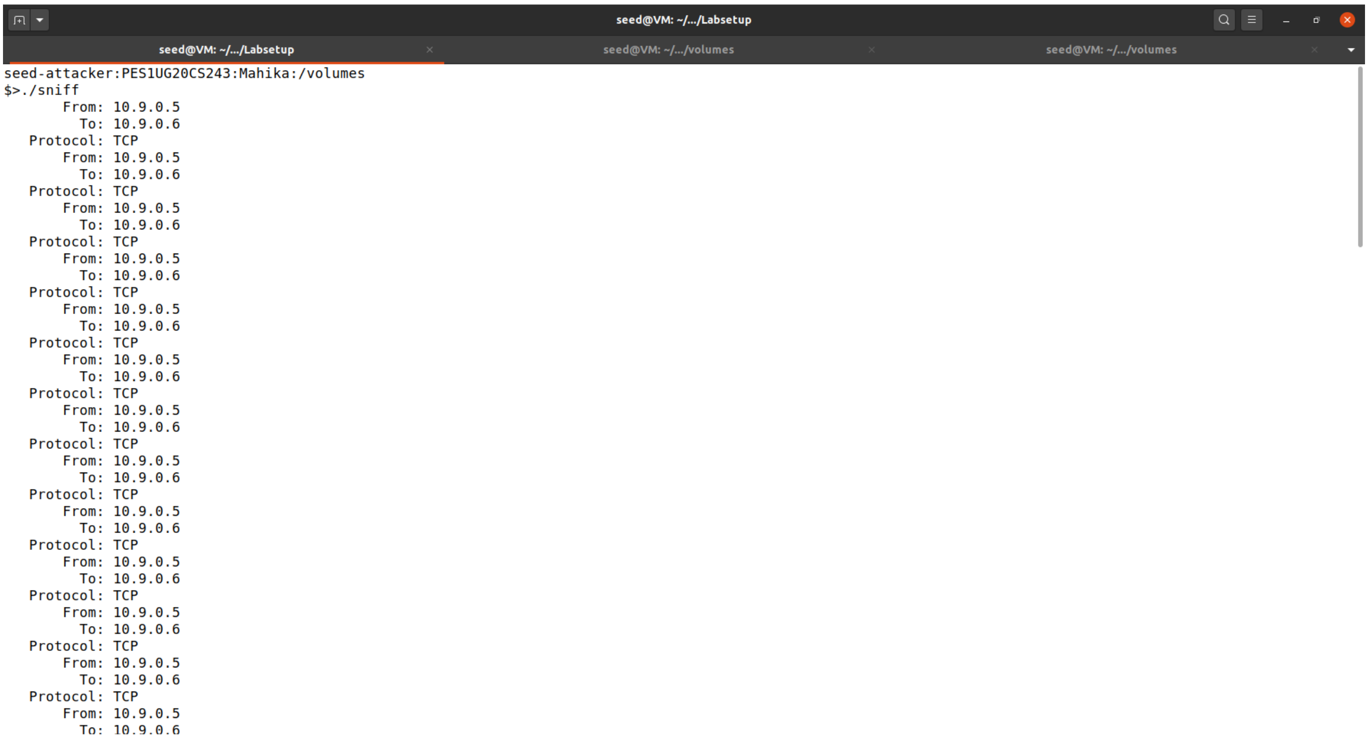
 # ./sniff

On Host A terminal :

 # telnet 10.9.0.6

Provide screenshots of your observations.






```
seed-attacker:PES1UG20CS243:Mahika:/volumes
$>./sniff
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
       From: 10.9.0.5
         To: 10.9.0.6
```

The TCP packets sent with port 21 are captured when telnet command is run on host machine A.

Task 2.1 C : Sniffing Passwords

Please show how you can use your sniffer program to capture the password when somebody is using telnet on the network that you are monitoring. It is acceptable if you print out the entire data part, and then manually mark where the password (or part of it) is.

Change the interface value in the code to the interface of the attacker machine as done in previous tasks.

On the host VM :

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

 # docker cp sniff [Attacker machine docker container ID]:/volumes

On the Attacker terminal run the command:

 # ./sniff

On Host A terminal :

 # telnet 10.9.0.6

Provide screenshots of your observations.

```
seed@c785a4727e2c:~$ telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
c785a4727e2c 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: Fri Sep  2 08:43:54 UTC 2022 from c785a4727e2c on pts/2
seed@c785a4727e2c:~$ 
```

```
seed@VM: ~/.../Labsetup                                  seed@VM: ~/...
seed-attacker:PES1UG20CS243:Mahika:/volumes
$>./sniff
tteellnneett  1100..00..00..66
Trying 10.0.0.6...
�telnet: Unable to contelnet 10.0.%�c�nd7CNo rouC99 .o0h.s6
Trying 10.9.0.6...:~$
�0.0.���!teln�Connected to 10.9.0.6.
Escape character is '^]'.
��!Bd9���!��U0�/K�Ubuntu 20.04.1 LTS
c785a4727e2c login: d@c7s[Dse[Deeedd
.assword: c$�c��f*Be�e3w<�s
0Welcome 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: Fri Sep  2 08:43:54 UTC 2022 from c785a4727e2c on pts/2
```

When telnet is used in the host A's machine, the user login and password can be captured 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

Spoof 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).

Open wireshark before executing the program and select the same interface in wireshark, as used in the code for each task ie. the attacker machines interface.

On the host VM :

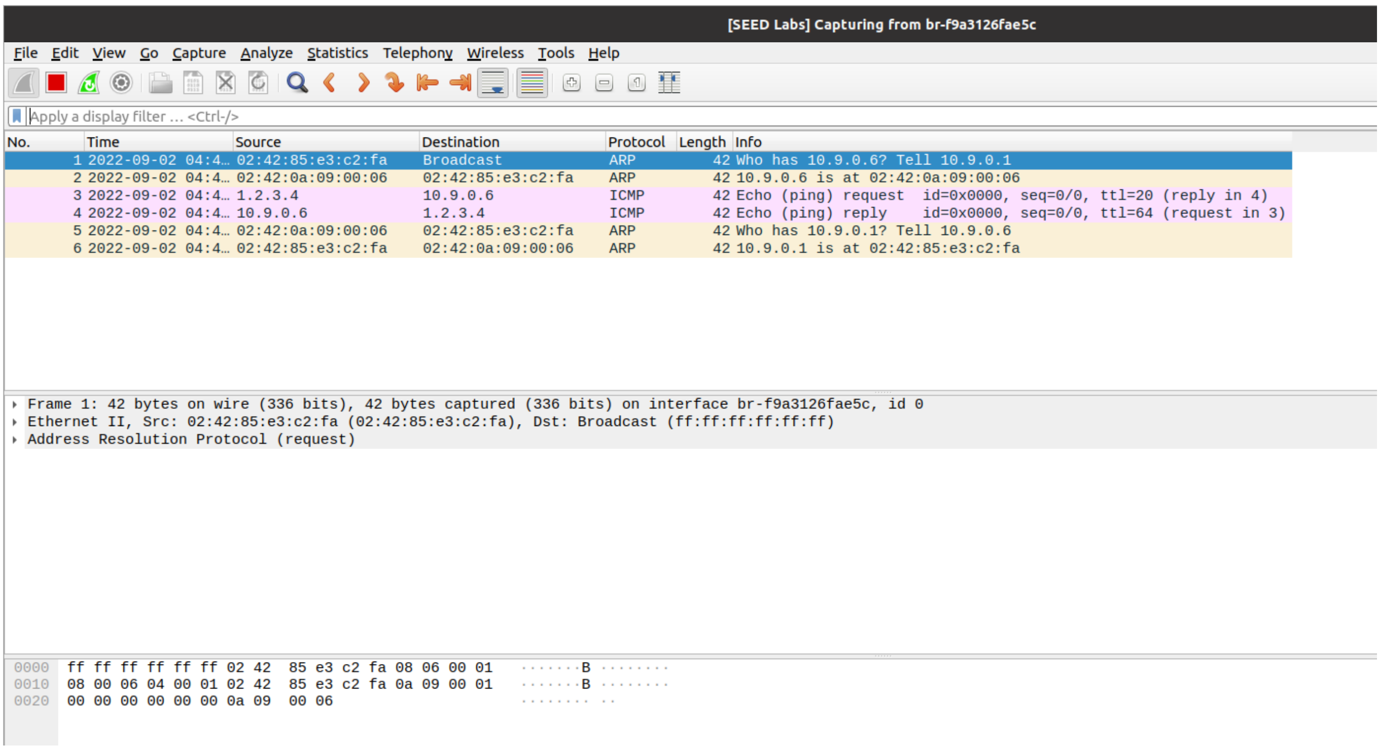
 # gcc -o spooficmp Task2.2.c -lpcap

 # docker cp spooficmp [Attacker machine docker container ID]:/volumes

On Attacker Machine terminal :

 # ./spooficmp

Provide screenshots of your observations.



Please answer the following questions.

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

ANSWER: During raw socket programming, checksum does not have to be calculated, as the operating system does it for us.

- **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?

ANSWER: Only root users are allowed to open raw sockets. It's because you can spoof custom packets, which may interfere with inbound traffic. So, during the execution of the program, you have to be the root user. The program fails at the line where the raw socket is created, i.e,
sd = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);

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.

Open wireshark before executing the program and select the same interface in wireshark, as used in the code for each task ie. the attacker machines interface.

Change the interface value in the code to the interface of the attacker machine as done in previous tasks.

On the host VM :

 # gcc -o sniffspoof Task2.3.c -lpcap

 # docker cp sniffspoof [Attacker machine docker container ID]:/volumes

On Attacker Machine terminal :

 # ./sniffspoof

On the Host A terminal ping 1.2.3.4

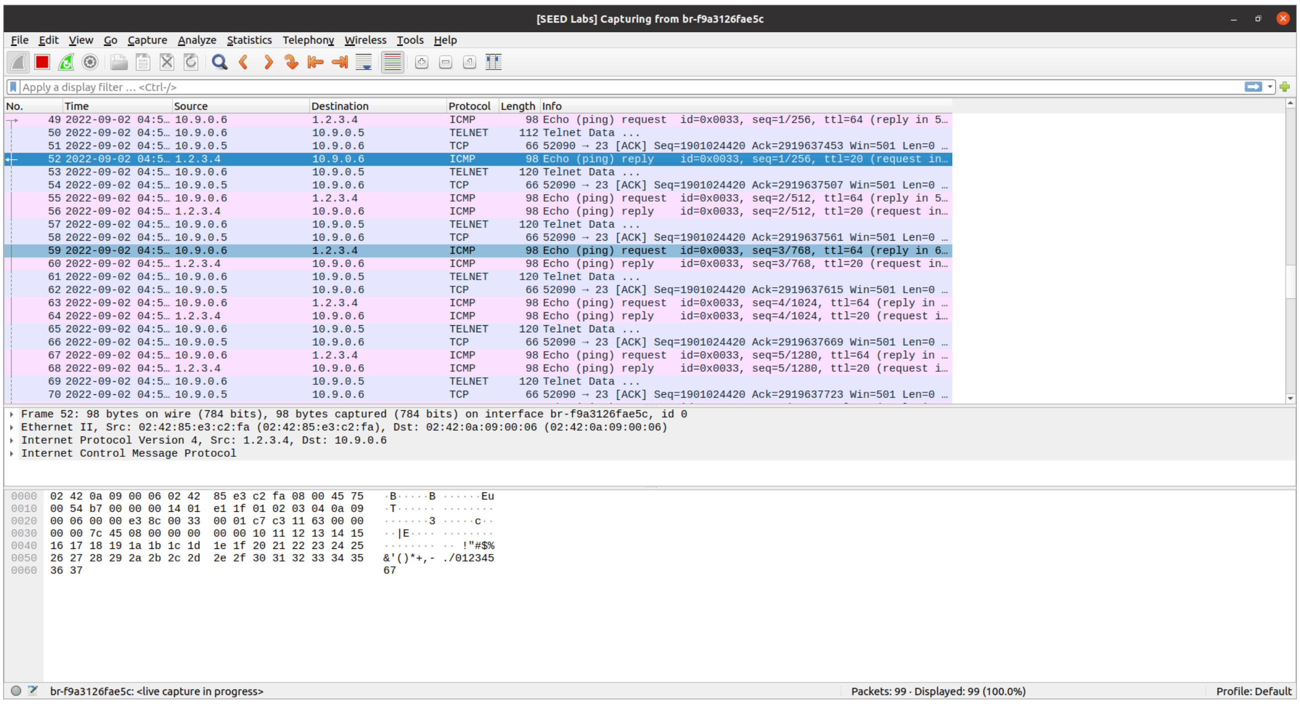
 # ping 1.2.3.4

Provide screenshots of your observations.

```
seed@c785a4727e2c:~$ 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=602 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=20 time=626 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=20 time=649 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=20 time=672 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=20 time=697 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=20 time=731 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=20 time=744 ms
^C
--- 1.2.3.4 ping statistics ---
8 packets transmitted, 7 received, 12.5% packet loss, time 7023ms
rtt min/avg/max/mdev = 601.512/674.463/743.997/49.077 ms
seed@c785a4727e2c:~$
```

```
seed-attacker:PES1UG20CS243:Mahika:/volumes
$>./sniffspoof
       From: 10.9.0.6
         To: 1.2.3.4
   Protocol: ICMP
       From: 1.2.3.4
         To: 10.9.0.6
   Protocol: ICMP
       From: 10.9.0.6
         To: 1.2.3.4
   Protocol: ICMP
       From: 1.2.3.4
         To: 10.9.0.6
   Protocol: ICMP
       From: 10.9.0.6
         To: 1.2.3.4
   Protocol: ICMP
       From: 1.2.3.4
         To: 10.9.0.6
   Protocol: ICMP
       From: 10.9.0.6
         To: 1.2.3.4
   Protocol: ICMP
       From: 1.2.3.4
         To: 10.9.0.6
   Protocol: ICMP
       From: 10.9.0.6
         To: 1.2.3.4
   Protocol: ICMP
       From: 1.2.3.4
         To: 10.9.0.6
   Protocol: ICMP
```

All the packets sent/received by 10.9.0.6 are sniffed.



As we can see here, a spoofed ICMP reply is sent from 1.2.3.4 to the host A machine.