Sniffing and Spoofing using PCAP Library

Name: Maryam Khan

SRN: PES2UG21CS283

Date: 31/8/23

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.

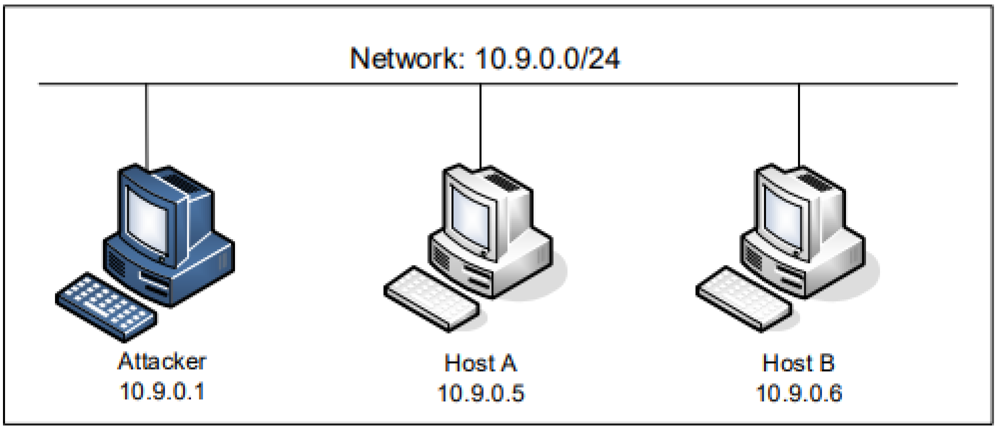


Figure 1 : Lab environment setup

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
```

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

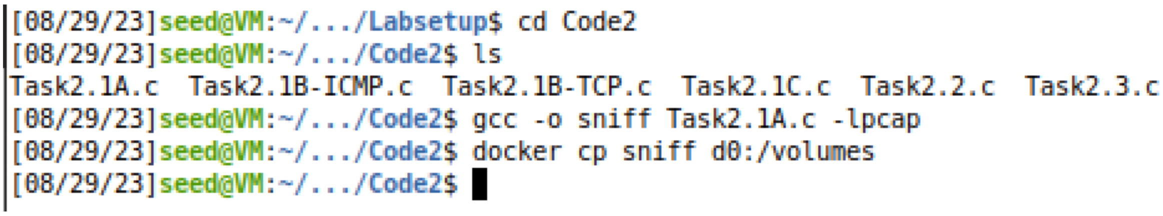
On the Attacker container run the command:

# ./sniff

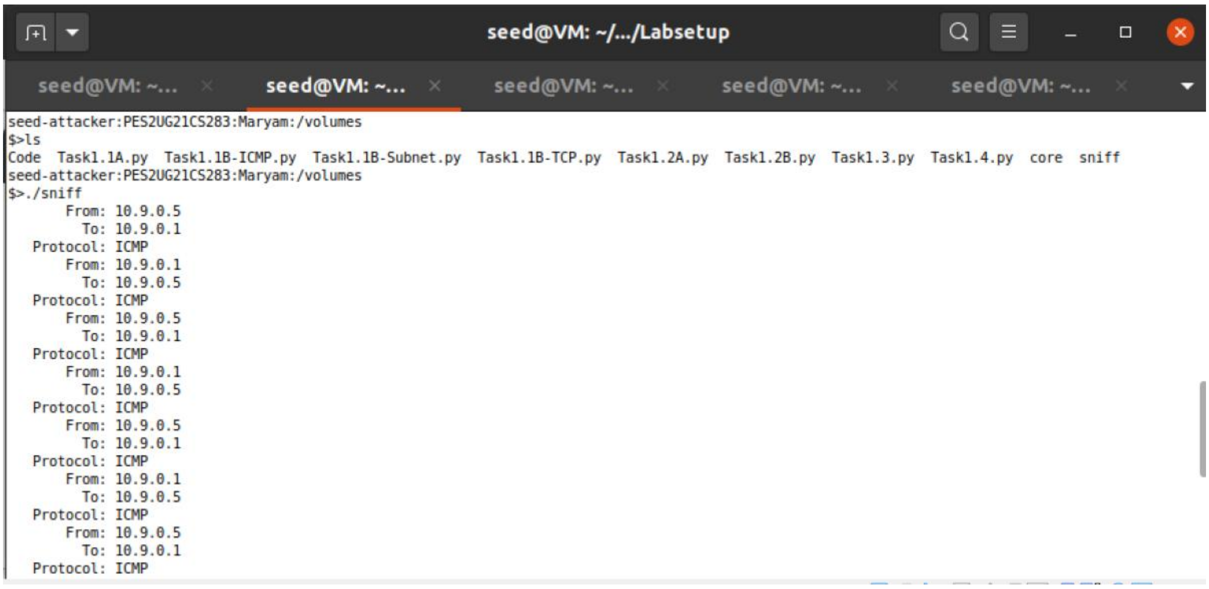
On Host A terminal :

# ping 10.9.0.1

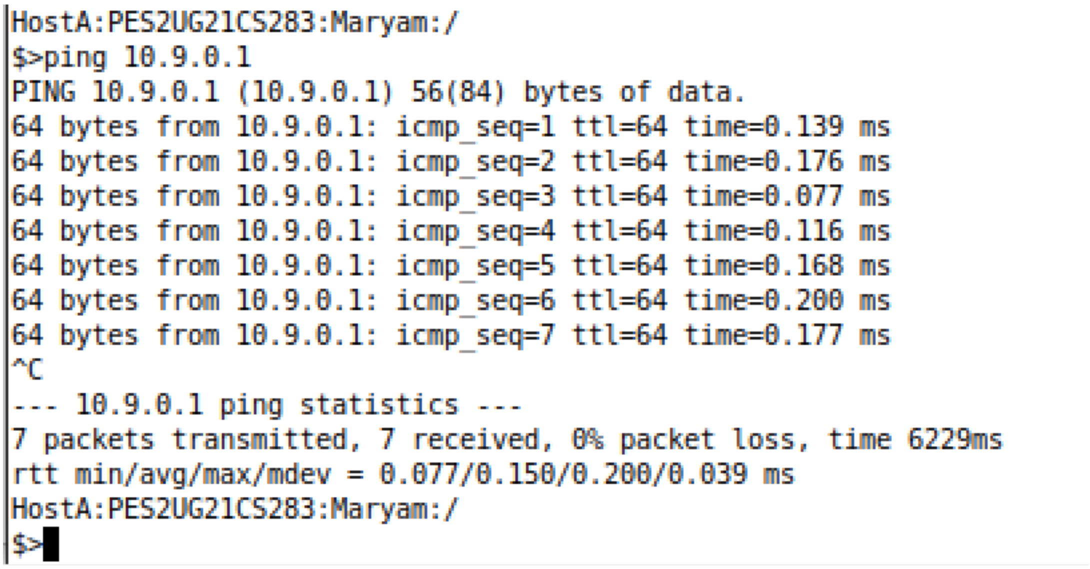
Provide screenshots of your observations.



```
[08/29/23]seed@VM:~/.../Labsetup$ cd Code2
[08/29/23]seed@VM:~/.../Code2$ ls
Task2.1A.c  Task2.1B-ICMP.c  Task2.1B-TCP.c  Task2.1C.c  Task2.2.c  Task2.3.c
[08/29/23]seed@VM:~/.../Code2$ gcc -o sniff Task2.1A.c -lpcap
[08/29/23]seed@VM:~/.../Code2$ docker cp sniff d0:/volumes
[08/29/23]seed@VM:~/.../Code2$ 
```



```
seed-attacker:PES2UG21CS283:Maryam:/volumes
$>ls
Code  Task1.1A.py  Task1.1B-ICMP.py  Task1.1B-Subnet.py  Task1.1B-TCP.py  Task1.2A.py  Task1.2B.py  Task1.3.py  Task1.4.py  core  sniff
seed-attacker:PES2UG21CS283:Maryam:/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
```



```
HostA:PES2UG21CS283:Maryam:/
$>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=0.139 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.176 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.077 ms
64 bytes from 10.9.0.1: icmp_seq=4 ttl=64 time=0.116 ms
64 bytes from 10.9.0.1: icmp_seq=5 ttl=64 time=0.168 ms
64 bytes from 10.9.0.1: icmp_seq=6 ttl=64 time=0.200 ms
64 bytes from 10.9.0.1: icmp_seq=7 ttl=64 time=0.177 ms
^C
--- 10.9.0.1 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6229ms
rtt min/avg/max/mdev = 0.077/0.150/0.200/0.039 ms
HostA:PES2UG21CS283:Maryam:/
$>
```

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 detailed explanation like the one in the tutorial.

Socket creation
Socket Configuration
Binding
Packet capture loop
Packet processing
Analysis and filtering
Display
Cleanup
Error Handling

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

The sniffer program needs access to the NIC in promiscuous mode, which can only be accessed by the root user. If we run the same executable without root permissions we get a Segmentation fault, which often happens while accessing something that the program does not have access to.

On the Attacker container run the command :

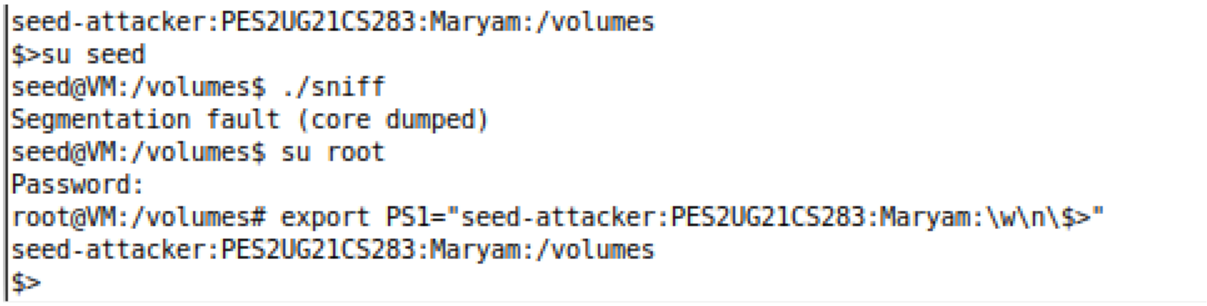
# su seed

# ./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:PES2UG21CS283:Maryam:/volumes
$>su seed
seed@VM:/volumes$ ./sniff
Segmentation fault (core dumped)
seed@VM:/volumes$ su root
Password:
root@VM:/volumes# export PS1="seed-attacker:PES2UG21CS283:Maryam:\w\n\$>"
seed-attacker:PES2UG21CS283:Maryam:/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?

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
```

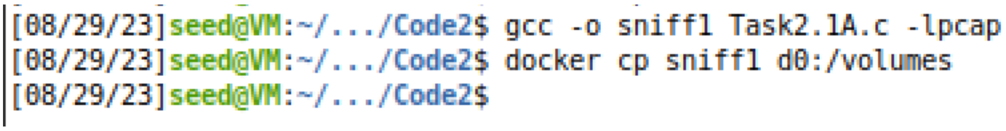
On the Attacker terminal run the command:

```
# ./sniff
```

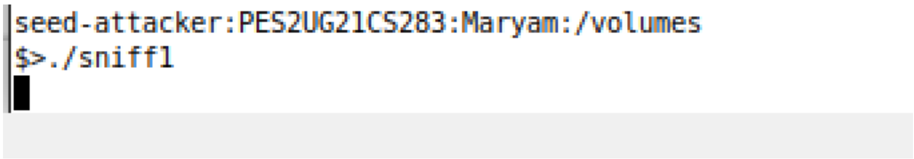
On Host A terminal :

```
# ping 10.9.0.6
```

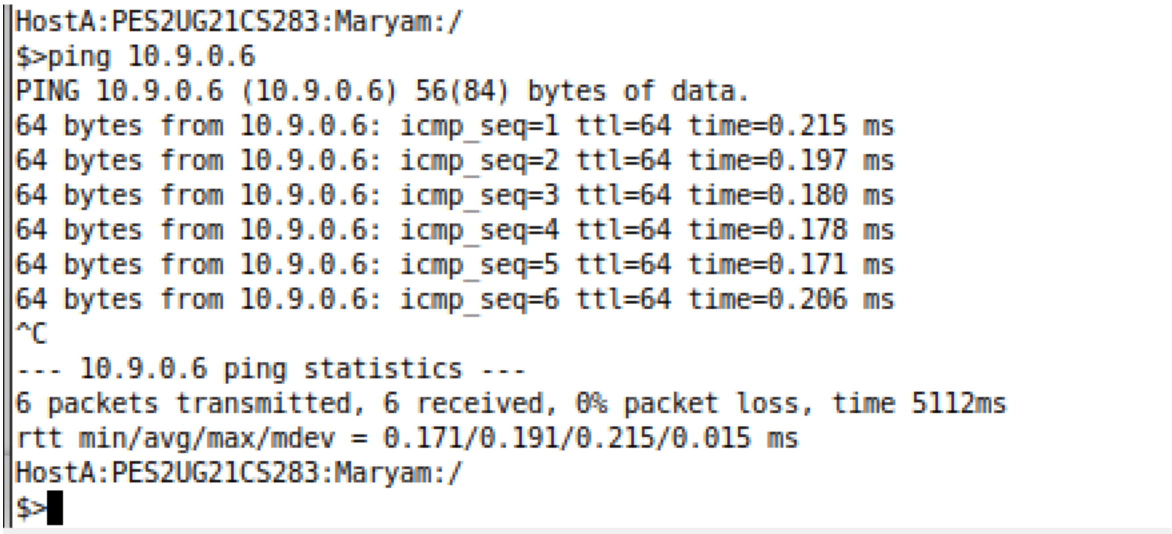
Provide screenshots of your observations.



```
[08/29/23]seed@VM:~/.../Code2$ gcc -o sniff1 Task2.1A.c -lpcap
[08/29/23]seed@VM:~/.../Code2$ docker cp sniff1 d0:/volumes
[08/29/23]seed@VM:~/.../Code2$
```



```
seed-attacker:PES2UG21CS283:Maryam:/volumes
$>./sniff1
```



```
HostA:PES2UG21CS283:Maryam:/
$>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=0.215 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.197 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.180 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.178 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.171 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.206 ms
^C
--- 10.9.0.6 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5112ms
rtt min/avg/max/mdev = 0.171/0.191/0.215/0.015 ms
HostA:PES2UG21CS283:Maryam:/
$>
```

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

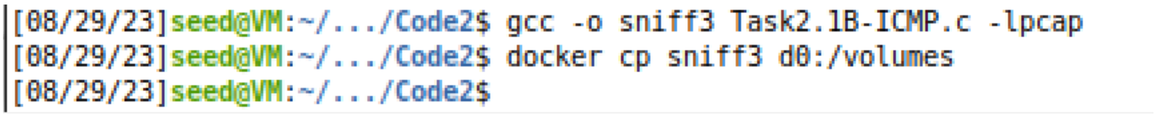
On the Attacker terminal run the command:

# ./sniff

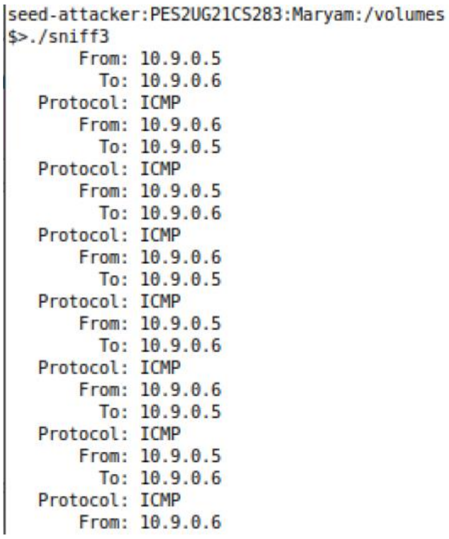
In the host A machine ping any ip address

# ping 10.9.0.6

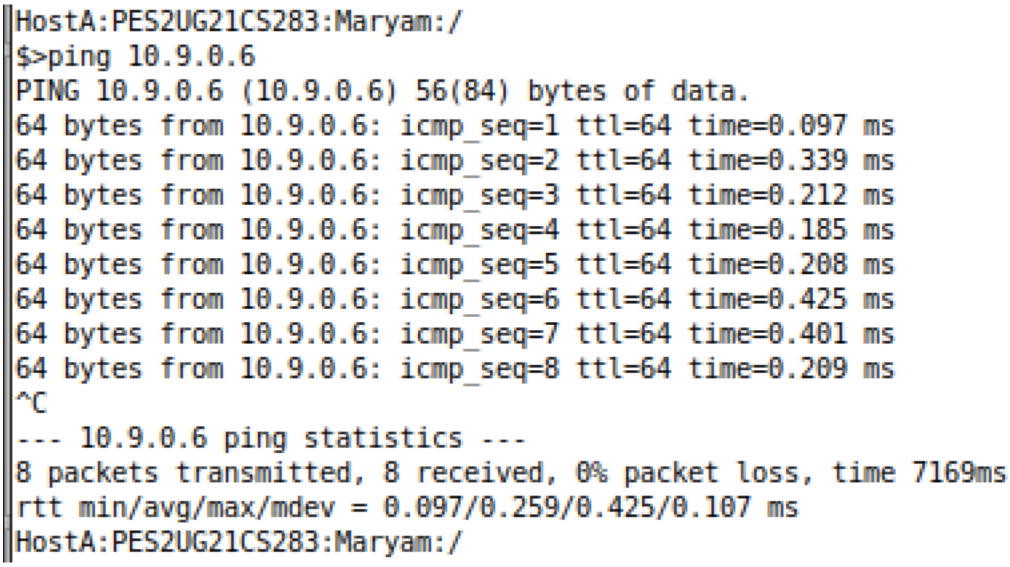
Provide screenshots of your observations.



```
[08/29/23]seed@VM:~/.../Code2$ gcc -o sniff3 Task2.1B-ICMP.c -lpcap
[08/29/23]seed@VM:~/.../Code2$ docker cp sniff3 d0:/volumes
[08/29/23]seed@VM:~/.../Code2$
```

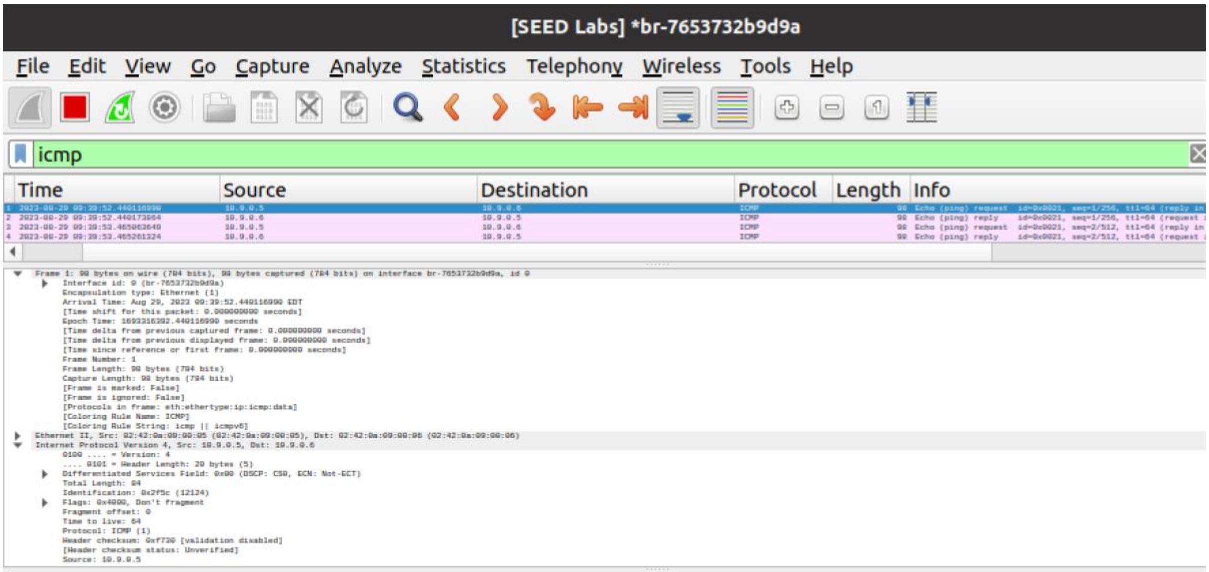


```
seed-attacker:PES2UG21CS283:Maryam:/volumes
$>./sniff3
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: ICMP
       From: 10.9.0.6
         To: 10.9.0.5
   Protocol: ICMP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: ICMP
       From: 10.9.0.6
         To: 10.9.0.5
   Protocol: 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
```



```
HostA:PES2UG21CS283:Maryam:/
$>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=0.097 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.339 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.212 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.185 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.208 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.425 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.401 ms
64 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.209 ms
^C
--- 10.9.0.6 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7169ms
rtt min/avg/max/mdev = 0.097/0.259/0.425/0.107 ms
HostA:PES2UG21CS283:Maryam:/
```

Frame 1 has request and reply message



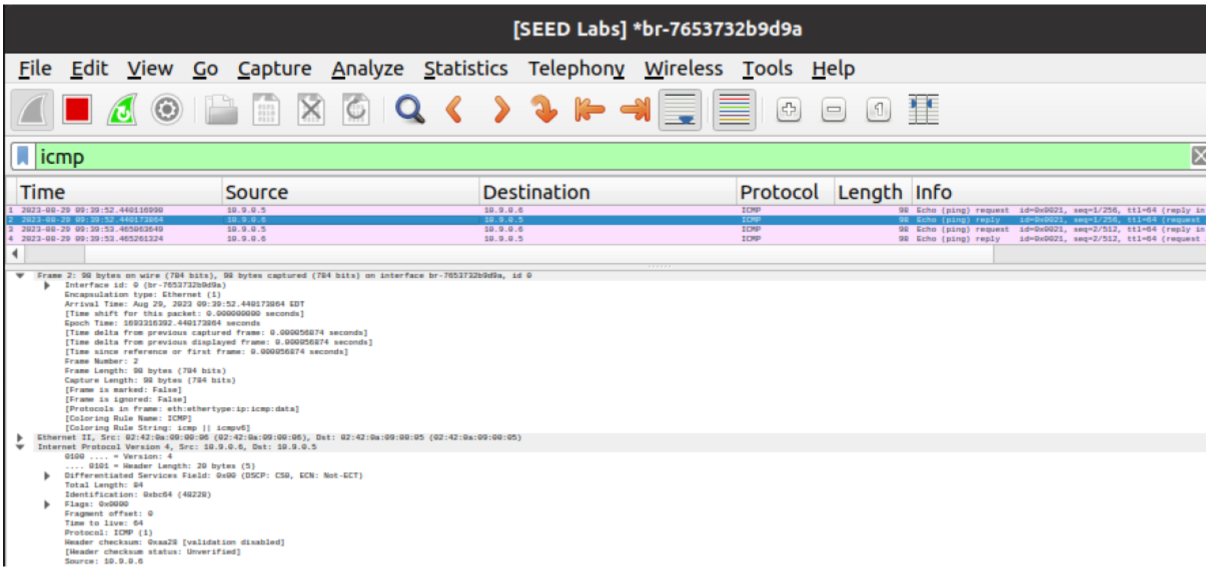
[SEED Labs] *br-7653732b9d9a
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

icmp

	Time	Source	Destination	Protocol	Length	Info
1	2023-08-29 09:39:52.440116990	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0021, seq=1/256, ttl=64 (reply in
2	2023-08-29 09:39:52.440173864	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=1/256, ttl=64 (request :
3	2023-08-29 09:39:53.465063640	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0021, seq=2/512, ttl=64 (reply in
4	2023-08-29 09:39:53.465261324	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0021, seq=2/512, ttl=64 (request :

```
▼ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-7653732b9d9a, id 0
    ▶ Interface id: 0 (br-7653732b9d9a)
      Encapsulation type: Ethernet (1)
      Arrival Time: Aug 29, 2023 09:39:52.440116990 EDT
      [Time shift for this packet: 0.000000000 seconds]
      Epoch Time: 1693316392.440116990 seconds
      [Time delta from previous captured frame: 0.000000000 seconds]
      [Time delta from previous displayed frame: 0.000000000 seconds]
      [Time since reference or first frame: 0.000000000 seconds]
      Frame Number: 1
      Frame Length: 98 bytes (784 bits)
      Capture Length: 98 bytes (784 bits)
      [Frame is marked: False]
      [Frame is ignored: False]
      [Protocols in frame: eth:ethertype:ip:icmp:data]
      [Coloring Rule Name: ICMP]
      [Coloring Rule String: icmp || icmpv6]
▶ Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:0a:09:00:06 (02:42:0a:09:00:06)
▼ Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
    ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 84
      Identification: 0x2f5c (12124)
    ▶ Flags: 0x4000, Don't fragment
      Fragment offset: 0
      Time to live: 64
      Protocol: ICMP (1)
      Header checksum: 0xf730 [validation disabled]
      [Header checksum status: Unverified]
      Source: 10.9.0.5
```

Frame 2 has request and reply message



[SEED Labs] *br-7653732b9d9a

icmp

```
  Time                          Source      Destination  Protocol  Length  Info
1 2023-08-29 09:39:52.440116990 10.9.0.5    10.9.0.6     ICMP        98  Echo (ping) request  id=0x0021, seq=1/256, ttl=64 (reply in
2 2023-08-29 09:39:52.440173864 10.9.0.6    10.9.0.5     ICMP        98  Echo (ping) reply    id=0x0021, seq=1/256, ttl=64 (request
3 2023-08-29 09:39:53.465063649 10.9.0.5    10.9.0.6     ICMP        98  Echo (ping) request  id=0x0021, seq=2/512, ttl=64 (reply in
4 2023-08-29 09:39:53.465261324 10.9.0.6    10.9.0.5     ICMP        98  Echo (ping) reply    id=0x0021, seq=2/512, ttl=64 (request
```

```
Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-7653732b9d9a, id 0
    Interface id: 0 (br-7653732b9d9a)
    Encapsulation type: Ethernet (1)
    Arrival Time: Aug 29, 2023 09:39:52.440173864 EDT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1693316392.440173864 seconds
    [Time delta from previous captured frame: 0.000056874 seconds]
    [Time delta from previous displayed frame: 0.000056874 seconds]
    [Time since reference or first frame: 0.000056874 seconds]
    Frame Number: 2
    Frame Length: 98 bytes (784 bits)
    Capture Length: 98 bytes (784 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:icmp:data]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp || icmpv6]
Ethernet II, Src: 02:42:0a:09:00:06 (02:42:0a:09:00:06), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0xbc64 (48228)
    Flags: 0x0000
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0xaa28 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.9.0.6
```

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
```

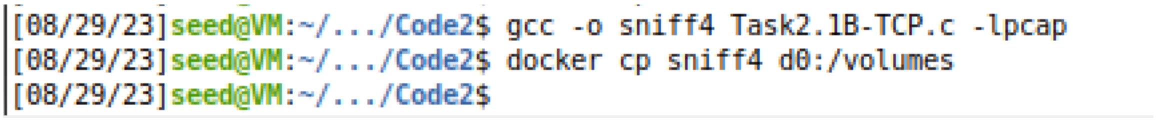
On Attacker Machine terminal :

```
# ./sniff
```

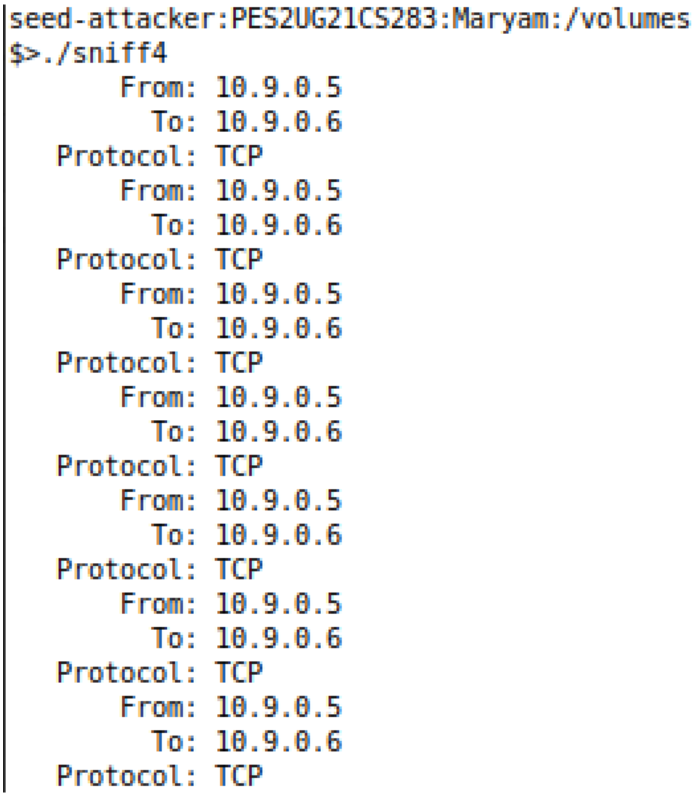
On Host A terminal :

```
# telnet 10.9.0.6
```

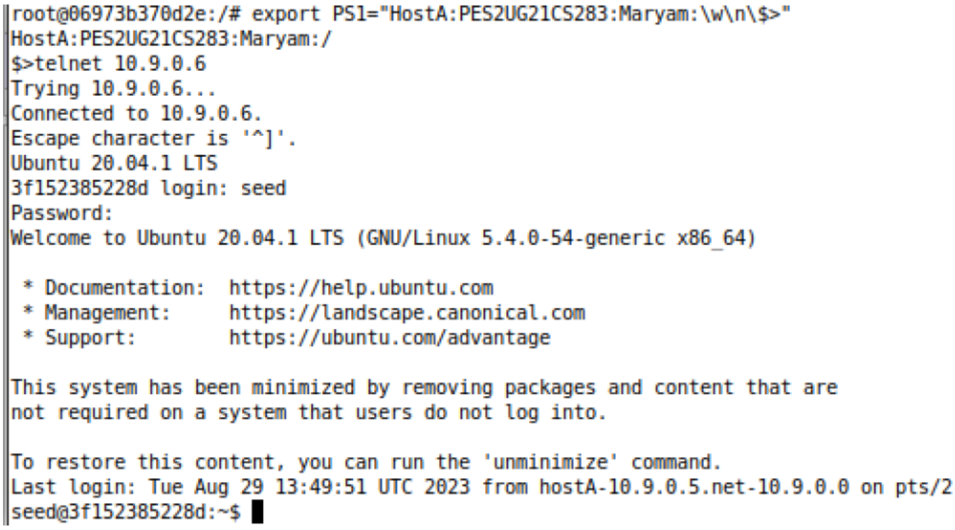
Provide screenshots of your observations.



```
[08/29/23]seed@VM:~/.../Code2$ gcc -o sniff4 Task2.1B-TCP.c -lpcap
[08/29/23]seed@VM:~/.../Code2$ docker cp sniff4 d0:/volumes
[08/29/23]seed@VM:~/.../Code2$
```



```
seed-attacker:PES2UG21CS283:Maryam:/volumes
$>./sniff4
       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
```



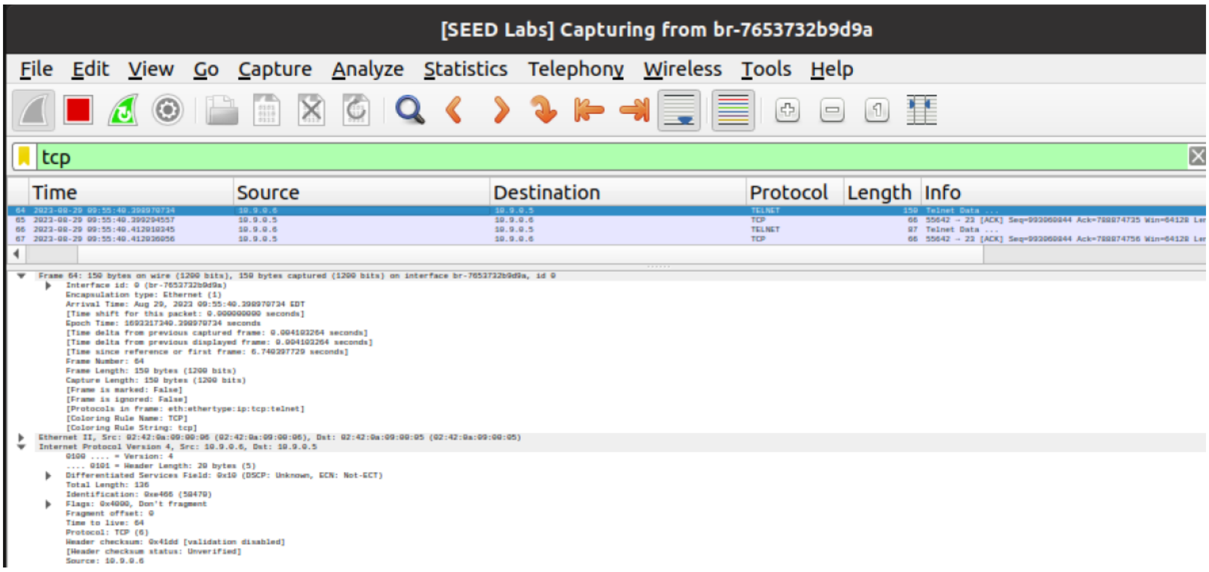
```
root@06973b370d2e:/# export PS1="HostA:PES2UG21CS283:Maryam:\w\n\$>"
HostA:PES2UG21CS283:Maryam:/
$>telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
3f152385228d login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * 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: Tue Aug 29 13:49:51 UTC 2023 from hostA-10.9.0.5.net-10.9.0.0 on pts/2
seed@3f152385228d:~$
```

Frame 64

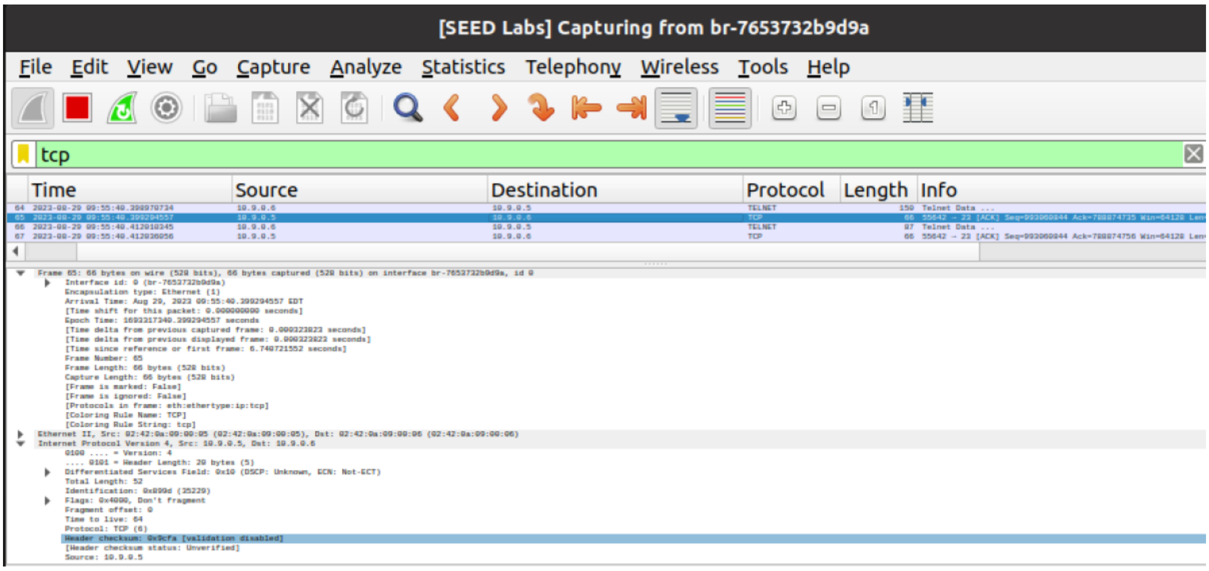

[SEED Labs] Capturing from br-7653732b9d9a
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp

```
     Time                            Source      Destination   Protocol  Length  Info
64   2023-08-29 09:55:40.398970734   10.9.0.6    10.9.0.5      TELNET       150  Telnet Data ...
65   2023-08-29 09:55:40.399294557   10.9.0.5    10.9.0.6      TCP           66  55642 → 23 [ACK] Seq=993069844 Ack=788874735 Win=64128 Len
66   2023-08-29 09:55:40.412910245   10.9.0.6    10.9.0.5      TELNET        87  Telnet Data ...
67   2023-08-29 09:55:40.412936056   10.9.0.5    10.9.0.6      TCP           66  55642 → 23 [ACK] Seq=993069844 Ack=788874756 Win=64128 Len
```

```
▼ Frame 64: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits) on interface br-7653732b9d9a, id 0
   ▶ Interface id: 0 (br-7653732b9d9a)
     Encapsulation type: Ethernet (1)
     Arrival Time: Aug 29, 2023 09:55:40.398970734 EDT
     [Time shift for this packet: 0.000000000 seconds]
     Epoch Time: 1693317340.398970734 seconds
     [Time delta from previous captured frame: 0.004103264 seconds]
     [Time delta from previous displayed frame: 0.004103264 seconds]
     [Time since reference or first frame: 6.740397729 seconds]
     Frame Number: 64
     Frame Length: 150 bytes (1200 bits)
     Capture Length: 150 bytes (1200 bits)
     [Frame is marked: False]
     [Frame is ignored: False]
     [Protocols in frame: eth:ethertype:ip:tcp:telnet]
     [Coloring Rule Name: TCP]
     [Coloring Rule String: tcp]
▶ Ethernet II, Src: 02:42:0a:09:00:06 (02:42:0a:09:00:06), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
▼ Internet Protocol Version 4, Src: 10.9.0.6, Dst: 10.9.0.5
     0100 .... = Version: 4
     .... 0101 = Header Length: 20 bytes (5)
   ▶ Differentiated Services Field: 0x10 (DSCP: Unknown, ECN: Not-ECT)
     Total Length: 136
     Identification: 0xe466 (58470)
   ▶ Flags: 0x4000, Don't fragment
     Fragment offset: 0
     Time to live: 64
     Protocol: TCP (6)
     Header checksum: 0x41dd [validation disabled]
     [Header checksum status: Unverified]
     Source: 10.9.0.6
```

Frame 65


[SEED Labs] Capturing from br-7653732b9d9a
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp

```
     Time                            Source      Destination   Protocol  Length  Info
64   2023-08-29 09:55:40.398970734   10.9.0.6    10.9.0.5      TELNET       150  Telnet Data ...
65   2023-08-29 09:55:40.399294557   10.9.0.5    10.9.0.6      TCP           66  55642 → 23 [ACK] Seq=993069844 Ack=788874735 Win=64128 Len
66   2023-08-29 09:55:40.412910245   10.9.0.6    10.9.0.5      TELNET        87  Telnet Data ...
67   2023-08-29 09:55:40.412936056   10.9.0.5    10.9.0.6      TCP           66  55642 → 23 [ACK] Seq=993069844 Ack=788874756 Win=64128 Len
```

```
▼ Frame 65: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface br-7653732b9d9a, id 0
   ▶ Interface id: 0 (br-7653732b9d9a)
     Encapsulation type: Ethernet (1)
     Arrival Time: Aug 29, 2023 09:55:40.399294557 EDT
     [Time shift for this packet: 0.000000000 seconds]
     Epoch Time: 1693317340.399294557 seconds
     [Time delta from previous captured frame: 0.000323823 seconds]
     [Time delta from previous displayed frame: 0.000323823 seconds]
     [Time since reference or first frame: 6.740721552 seconds]
     Frame Number: 65
     Frame Length: 66 bytes (528 bits)
     Capture Length: 66 bytes (528 bits)
     [Frame is marked: False]
     [Frame is ignored: False]
     [Protocols in frame: eth:ethertype:ip:tcp]
     [Coloring Rule Name: TCP]
     [Coloring Rule String: tcp]
▶ Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:0a:09:00:06 (02:42:0a:09:00:06)
▼ Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
     0100 .... = Version: 4
     .... 0101 = Header Length: 20 bytes (5)
   ▶ Differentiated Services Field: 0x10 (DSCP: Unknown, ECN: Not-ECT)
     Total Length: 52
     Identification: 0x899d (35229)
   ▶ Flags: 0x4000, Don't fragment
     Fragment offset: 0
     Time to live: 64
     Protocol: TCP (6)
     Header checksum: 0x9cfa [validation disabled]
     [Header checksum status: Unverified]
     Source: 10.9.0.5
```

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

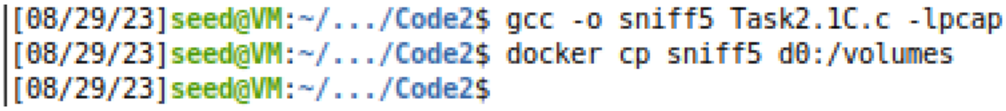
On the Attacker terminal run the command:

# ./sniff

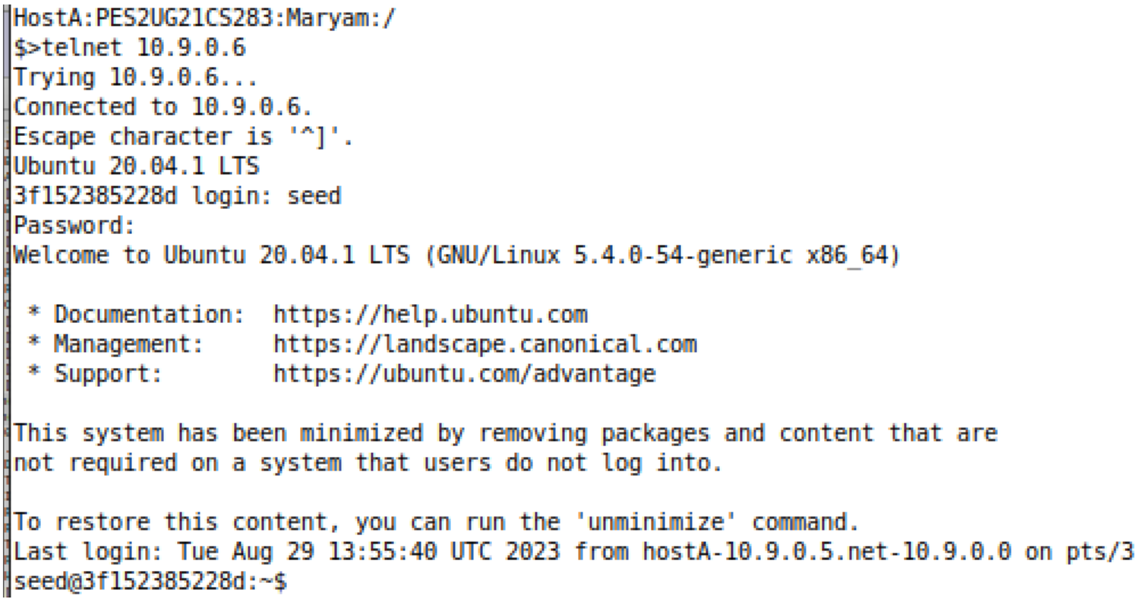
On Host A terminal :

# telnet 10.9.0.6

Provide screenshots of your observations.



```
[08/29/23]seed@VM:~/.../Code2$ gcc -o sniff5 Task2.1C.c -lpcap
[08/29/23]seed@VM:~/.../Code2$ docker cp sniff5 d0:/volumes
[08/29/23]seed@VM:~/.../Code2$
```

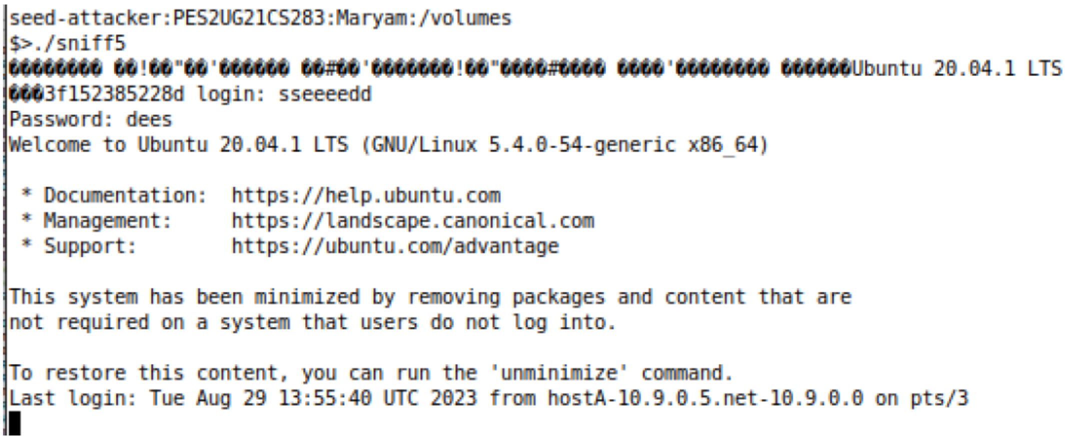


```
HostA:PES2UG21CS283:Maryam:/
$>telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
3f152385228d login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * 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: Tue Aug 29 13:55:40 UTC 2023 from hostA-10.9.0.5.net-10.9.0.0 on pts/3
seed@3f152385228d:~$
```



```
seed-attacker:PES2UG21CS283:Maryam:/volumes
$>./sniff5
�������� ��!��"��'������ ��#��'�������!��"����#���� ����'�������� ������Ubuntu 20.04.1 LTS
���3f152385228d login: sseeeedd
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: Tue Aug 29 13:55:40 UTC 2023 from hostA-10.9.0.5.net-10.9.0.0 on pts/3
█
```

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 :

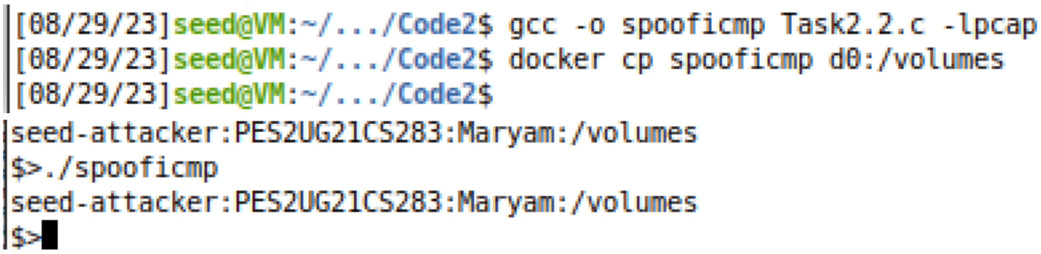
# gcc -o spooficmp Task2.2.c -lpcap

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

On Attacker Machine terminal :

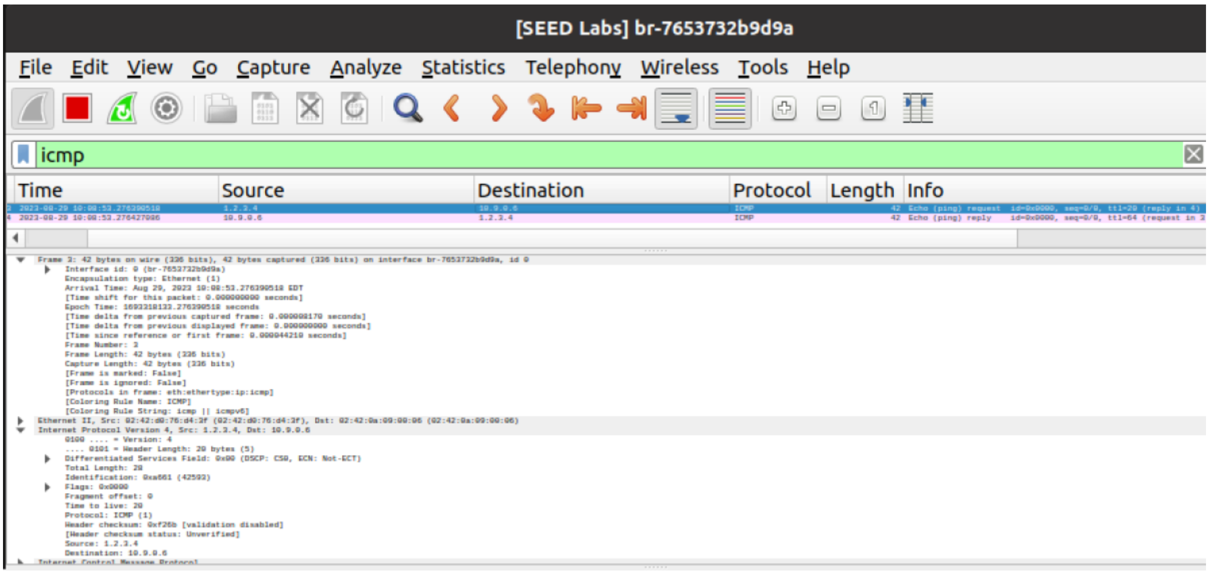
# ./spooficmp

Provide screenshots of your observations.



```
[08/29/23]seed@VM:~/.../Code2$ gcc -o spooficmp Task2.2.c -lpcap
[08/29/23]seed@VM:~/.../Code2$ docker cp spooficmp d0:/volumes
[08/29/23]seed@VM:~/.../Code2$
seed-attacker:PES2UG21CS283:Maryam:/volumes
$>./spooficmp
seed-attacker:PES2UG21CS283:Maryam:/volumes
$>
```

Frame 2



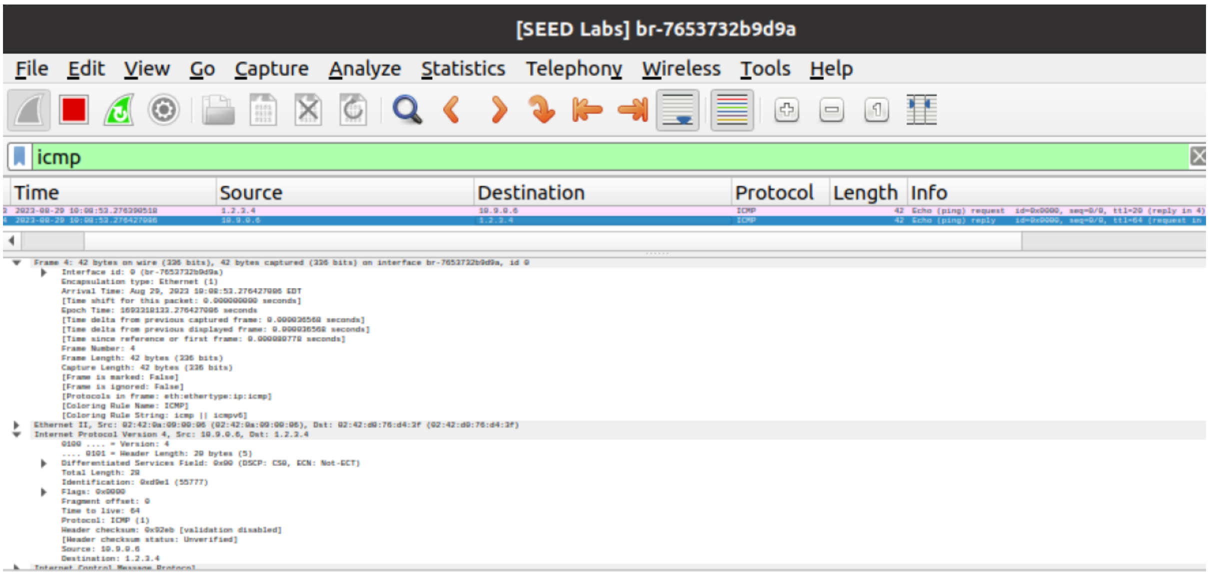
[SEED Labs] br-7653732b9d9a
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

icmp

```
  Time                           Source     Destination  Protocol  Length  Info
2 2023-08-29 10:08:53.276390518  1.2.3.4    10.9.0.6     ICMP      42      Echo (ping) request  id=0x0000, seq=0/0, ttl=20 (reply in 4)
4 2023-08-29 10:08:53.276427086  10.9.0.6   1.2.3.4      ICMP      42      Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 2

Frame 2: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface br-7653732b9d9a, id 0
    Interface id: 0 (br-7653732b9d9a)
    Encapsulation type: Ethernet (1)
    Arrival Time: Aug 29, 2023 10:08:53.276390518 EDT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1693318133.276390518 seconds
    [Time delta from previous captured frame: 0.000008170 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000044210 seconds]
    Frame Number: 2
    Frame Length: 42 bytes (336 bits)
    Capture Length: 42 bytes (336 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:icmp]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp || icmpv6]
Ethernet II, Src: 02:42:d0:76:d4:2f (02:42:d0:76:d4:2f), Dst: 02:42:0a:09:00:06 (02:42:0a:09:00:06)
Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.9.0.6
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 28
    Identification: 0xa661 (42593)
    Flags: 0x0000
    Fragment offset: 0
    Time to live: 20
    Protocol: ICMP (1)
    Header checksum: 0xf26b [validation disabled]
    [Header checksum status: Unverified]
    Source: 1.2.3.4
    Destination: 10.9.0.6
Internet Control Message Protocol
```

Frame 4



[SEED Labs] br-7653732b9d9a
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

icmp

```
  Time                           Source     Destination  Protocol  Length  Info
2 2023-08-29 10:08:53.276390518  1.2.3.4    10.9.0.6     ICMP      42      Echo (ping) request  id=0x0000, seq=0/0, ttl=20 (reply in 4)
4 2023-08-29 10:08:53.276427086  10.9.0.6   1.2.3.4      ICMP      42      Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in

Frame 4: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface br-7653732b9d9a, id 0
    Interface id: 0 (br-7653732b9d9a)
    Encapsulation type: Ethernet (1)
    Arrival Time: Aug 29, 2023 10:08:53.276427086 EDT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1693318133.276427086 seconds
    [Time delta from previous captured frame: 0.000036568 seconds]
    [Time delta from previous displayed frame: 0.000036568 seconds]
    [Time since reference or first frame: 0.000080778 seconds]
    Frame Number: 4
    Frame Length: 42 bytes (336 bits)
    Capture Length: 42 bytes (336 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:icmp]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp || icmpv6]
Ethernet II, Src: 02:42:0a:09:00:06 (02:42:0a:09:00:06), Dst: 02:42:d0:76:d4:2f (02:42:d0:76:d4:2f)
Internet Protocol Version 4, Src: 10.9.0.6, Dst: 1.2.3.4
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 28
    Identification: 0xd9e1 (55777)
    Flags: 0x0000
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0x92eb [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.9.0.6
    Destination: 1.2.3.4
Internet Control Message Protocol
```

Please answer the following questions.

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

Yes, when using raw socket programming to send or receive packets at the IP layer, you need to calculate the checksum for the IP header (setting the IP header checksum bit to 0). This lets the stack, versus the application, calculate the checksum and populate the IP header checksum value accordingly. An application must use recvfrom to read datagrams from a raw socket.

- **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?

Allowing unprivileged users to use raw sockets would break the security assumptions, since with raw sockets anything could be done in the network, like using privileged ports, spoofing IP addresses. Therefore it is not allowed.

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

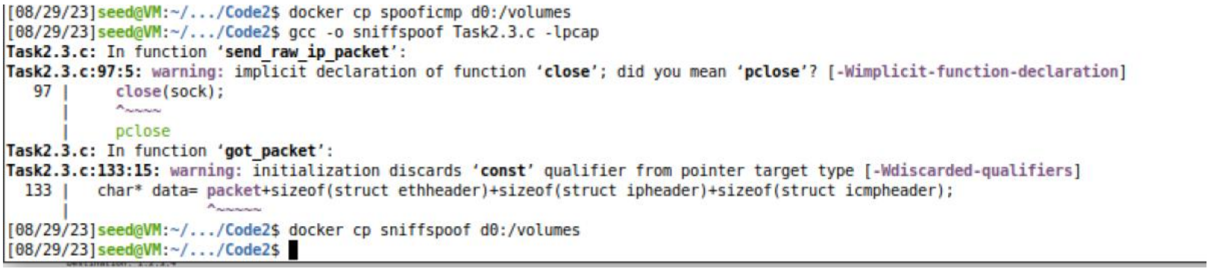
On Attacker Machine terminal :

# ./sniffspoof

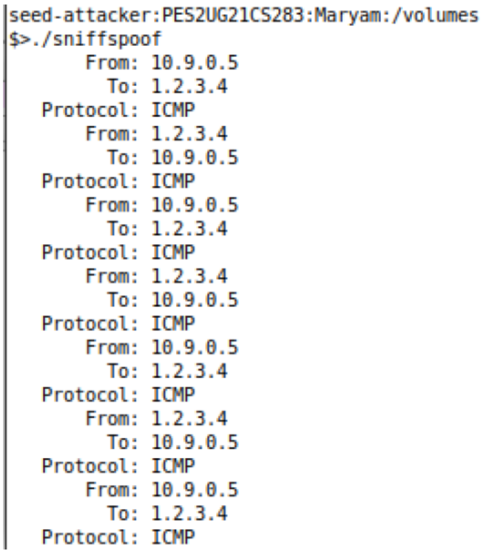
On the Host A terminal ping 1.2.3.4

# ping 1.2.3.4

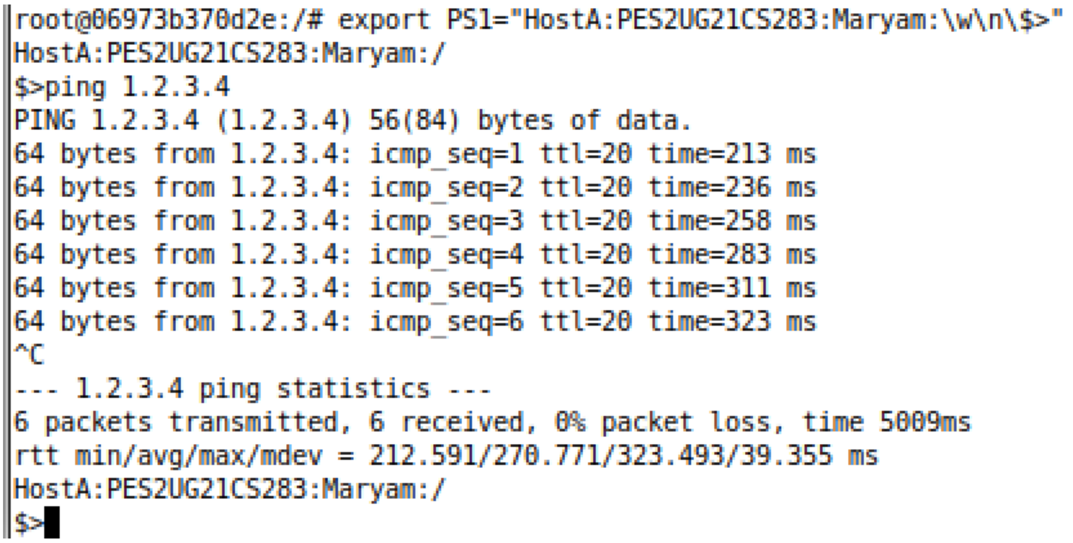
Provide screenshots of your observations.



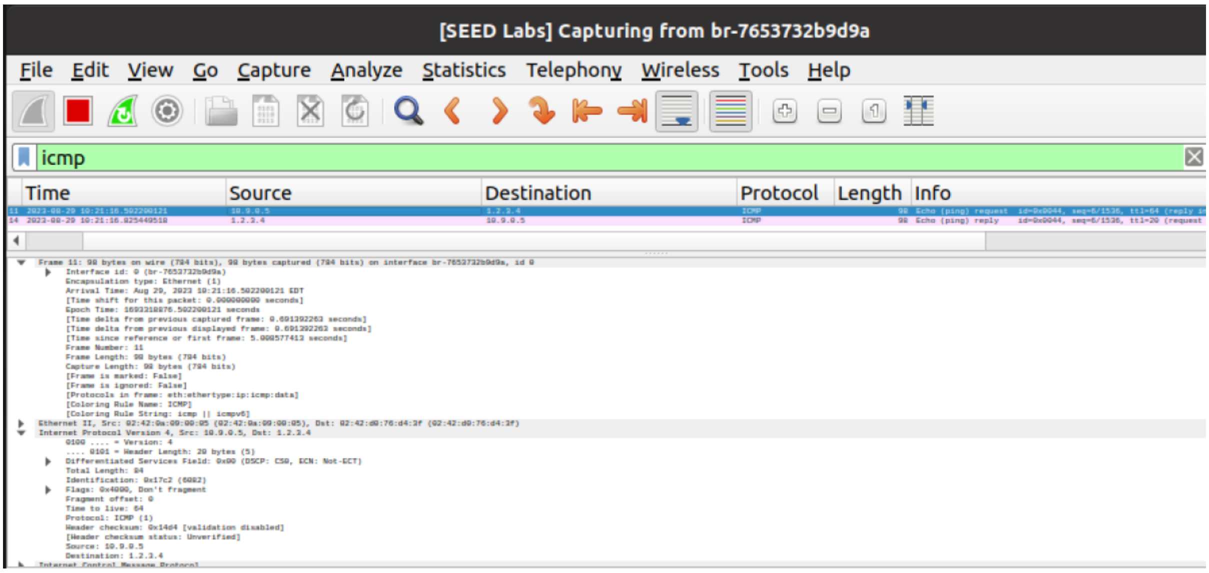
```
[08/29/23]seed@VM:~/.../Code2$ docker cp spooficmp d0:/volumes
[08/29/23]seed@VM:~/.../Code2$ gcc -o sniffspoof Task2.3.c -lpcap
Task2.3.c: In function ‘send_raw_ip_packet’:
Task2.3.c:97:5: warning: implicit declaration of function ‘close’; did you mean ‘pclose’? [-Wimplicit-function-declaration]
   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);
      |               ^~~~~~
[08/29/23]seed@VM:~/.../Code2$ docker cp sniffspoof d0:/volumes
[08/29/23]seed@VM:~/.../Code2$ 
```



```
seed-attacker:PES2UG21CS283:Maryam:/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
```



```
root@06973b370d2e:/# export PS1="HostA:PES2UG21CS283:Maryam:\w\n\$>"
HostA:PES2UG21CS283:Maryam:/
$>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=213 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=20 time=236 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=20 time=258 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=20 time=283 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=20 time=311 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=20 time=323 ms
^C
--- 1.2.3.4 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 212.591/270.771/323.493/39.355 ms
HostA:PES2UG21CS283:Maryam:/
$>
```

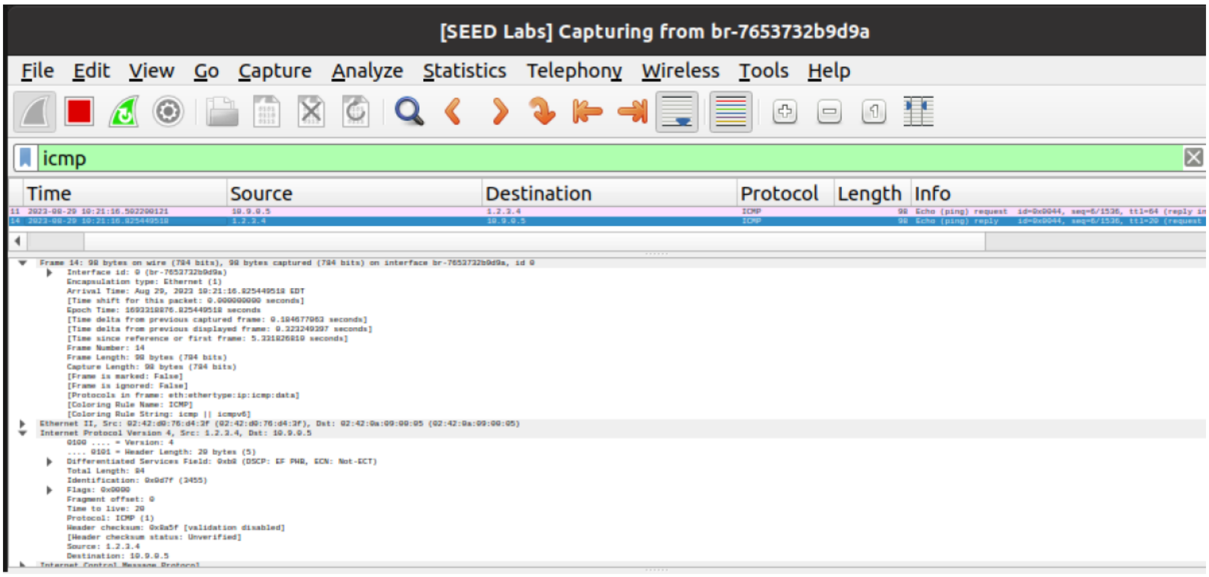


[SEED Labs] Capturing from br-7653732b9d9a
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

icmp

```
   Time                              Source       Destination  Protocol  Length Info
11 2023-08-29 10:21:16.502200121    10.9.0.5     1.2.3.4      ICMP      98 Echo (ping) request  id=0x0044, seq=6/1536, ttl=64 (reply in
14 2023-08-29 10:21:16.825449518    1.2.3.4      10.9.0.5     ICMP      98 Echo (ping) reply    id=0x0044, seq=6/1536, ttl=20 (request
```

```
▼ Frame 11: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-7653732b9d9a, id 0
    ▶ Interface id: 0 (br-7653732b9d9a)
      Encapsulation type: Ethernet (1)
      Arrival Time: Aug 29, 2023 10:21:16.502200121 EDT
      [Time shift for this packet: 0.000000000 seconds]
      Epoch Time: 1693318876.502200121 seconds
      [Time delta from previous captured frame: 0.691392263 seconds]
      [Time delta from previous displayed frame: 0.691392263 seconds]
      [Time since reference or first frame: 5.008577413 seconds]
      Frame Number: 11
      Frame Length: 98 bytes (784 bits)
      Capture Length: 98 bytes (784 bits)
      [Frame is marked: False]
      [Frame is ignored: False]
      [Protocols in frame: eth:ethertype:ip:icmp:data]
      [Coloring Rule Name: ICMP]
      [Coloring Rule String: icmp || icmpv6]
▶ Ethernet II, Src: 02:42:0a:09:00:05 (02:42:0a:09:00:05), Dst: 02:42:d0:76:d4:3f (02:42:d0:76:d4:3f)
▼ Internet Protocol Version 4, Src: 10.9.0.5, Dst: 1.2.3.4
      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
    ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 84
      Identification: 0x17c2 (6082)
    ▶ Flags: 0x4000, Don't fragment
      Fragment offset: 0
      Time to live: 64
      Protocol: ICMP (1)
      Header checksum: 0x14d4 [validation disabled]
      [Header checksum status: Unverified]
      Source: 10.9.0.5
      Destination: 1.2.3.4
▶ Internet Control Message Protocol
```



[SEED Labs] Capturing from br-7653732b9d9a
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

icmp

```
   Time                              Source       Destination  Protocol  Length Info
11 2023-08-29 10:21:16.502200121    10.9.0.5     1.2.3.4      ICMP      98 Echo (ping) request  id=0x0044, seq=6/1536, ttl=64 (reply in
14 2023-08-29 10:21:16.825449518    1.2.3.4      10.9.0.5     ICMP      98 Echo (ping) reply    id=0x0044, seq=6/1536, ttl=20 (request
```

```
▼ Frame 14: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-7653732b9d9a, id 0
    ▶ Interface id: 0 (br-7653732b9d9a)
      Encapsulation type: Ethernet (1)
      Arrival Time: Aug 29, 2023 10:21:16.825449518 EDT
      [Time shift for this packet: 0.000000000 seconds]
      Epoch Time: 1693318876.825449518 seconds
      [Time delta from previous captured frame: 0.184677063 seconds]
      [Time delta from previous displayed frame: 0.323249397 seconds]
      [Time since reference or first frame: 5.331826810 seconds]
      Frame Number: 14
      Frame Length: 98 bytes (784 bits)
      Capture Length: 98 bytes (784 bits)
      [Frame is marked: False]
      [Frame is ignored: False]
      [Protocols in frame: eth:ethertype:ip:icmp:data]
      [Coloring Rule Name: ICMP]
      [Coloring Rule String: icmp || icmpv6]
▶ Ethernet II, Src: 02:42:d0:76:d4:3f (02:42:d0:76:d4:3f), Dst: 02:42:0a:09:00:05 (02:42:0a:09:00:05)
▼ Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.9.0.5
      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
    ▶ Differentiated Services Field: 0xb8 (DSCP: EF PHB, ECN: Not-ECT)
      Total Length: 84
      Identification: 0x0d7f (3455)
    ▶ Flags: 0x0000
      Fragment offset: 0
      Time to live: 20
      Protocol: ICMP (1)
      Header checksum: 0x8a5f [validation disabled]
      [Header checksum status: Unverified]
      Source: 1.2.3.4
      Destination: 10.9.0.5
▶ Internet Control Message Protocol
```