PES UNIVERSITY

Department of Computer Science & Engineering

Computer Network Security

UE23CS343AB6

Assignment 2 Submission

Name of the Student	Pranav Hemanth
SRN	PES1UG23CS433
Section	G
Department	CSE
Campus	RR

Computer Network Security

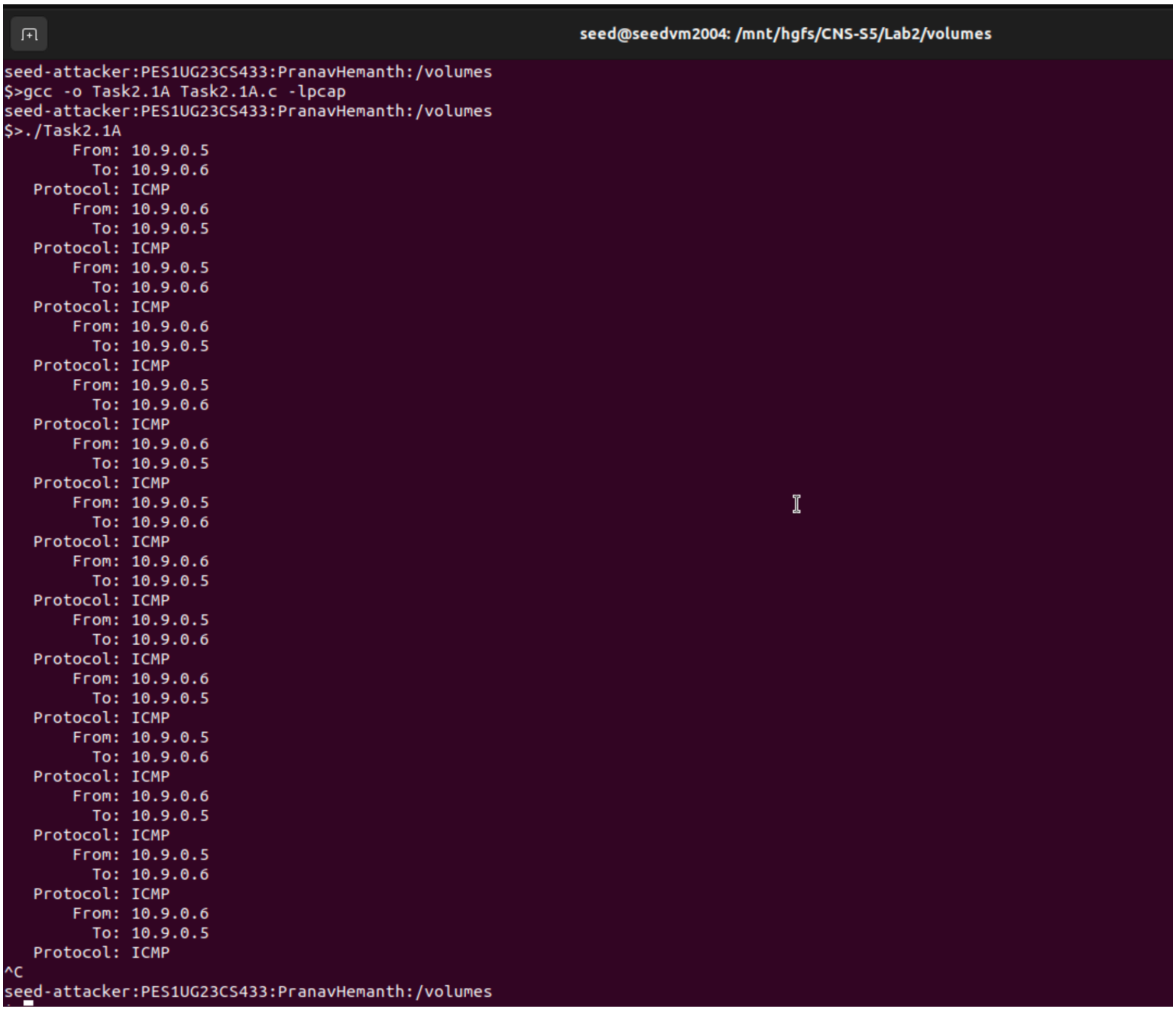
UE23CS343AB6

Task 2.1 : Sniffing - Writing Packet Sniffing Program

Task 2.1 A : Understanding how a Sniffer Works

Step 1:

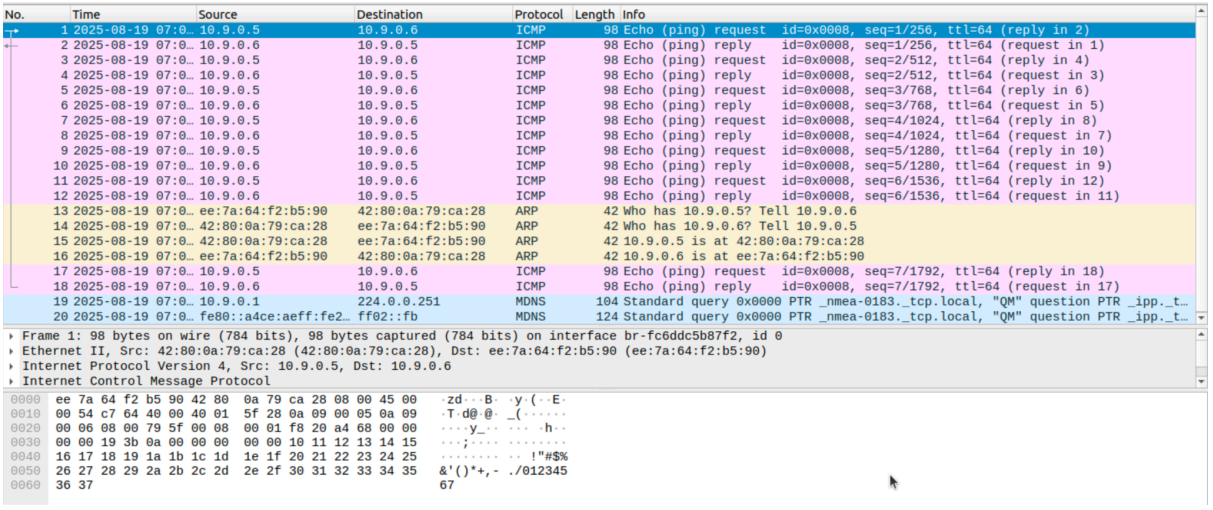
Attacker:



Host A:

```
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>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=0.246 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.155 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.166 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.135 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.149 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.427 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.162 ms
^C
--- 10.9.0.6 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6150ms
rtt min/avg/max/mdev = 0.135/0.205/0.427/0.096 ms
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:



On the attacker we see that the sniffer successfully captures ICMP packets generated by the ping, showing the source (10.9.0.5) and destination (10.9.0.6) IPs along with protocol. In Wireshark we observe the same ICMP echo request and reply packets confirming that both capture samenetwork traffic.

Question 1: Describe the sequence of the library calls that are essential for sniffer programs using pcap.

Answer:

For a sniffer program using libpcap, the sequence of library calls is:

1. pcap_open_live() - opens a live capture session on a given network interface.
2. pcap_compile() - compiles a human readable filter expression (e.g., "icmp") into BPF bytecode.
3. pcap_setfilter() - applies the compiled filter to the capture session.
4. pcap_loop() - starts capturing packets and invokes the defined callback (got_packet) for each captured packet.
5. pcap_close() - closes the capture session and releases resources.

Question 2: Why do you need the root privilege to run sniffer? Where does the program fail if executed without the root privilege? Provide a screenshot of the output on the terminal and explain your observation.

Answer:
Sniffer programs need root privileges because capturing raw packets and enabling promiscuous mode require sudo permission. Without root pcap_open_live() fails due to lack of access, returning a NULL handle. Since the program doesn't check this properly the later pcap calls use the invalid handle, causing a segmentation fault on the terminal.

Non root user:

```
seed@seedvm2004:/volumes$ PS1="seedvm2004:PES1UG23CS433:PranavHemanth:\w\n\$>"
seedvm2004:PES1UG23CS433:PranavHemanth:/volumes
$>./Task2.1A
Segmentation fault (core dumped)
seedvm2004:PES1UG23CS433:PranavHemanth:/volumes
```

Question 3: 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?

Answer:
With promiscuous mode on, the NIC captures all packets on the network so the sniffer can see traffic between other hosts. With promiscuous mode off, it only captures packets meant for the sniffer's own machine, so unrelated host-to-host traffic is not visible.

With promiscuous mode we will get the same output as question 1

Attacker:

```
root@seedvm2004:/volumes# PS1="seed-attacker:PES1UG23CS433:PranavHemanth:\w\n\$>
"
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>cc -o Task2.1A_Promiscous_off Task2.1A.c -lpcap
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>./Task2.1A_Promiscous_off
^C
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>
```

Host A:

```
$>PS1="seed-HostA:PES1UG23CS433:PranavHemanth:\w\n\$>"
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>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=0.166 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.140 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.136 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.115 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.140 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.141 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.142 ms
^C
--- 10.9.0.6 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6153ms
rtt min/avg/max/mdev = 0.115/0.140/0.166/0.013 ms
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>
```

As explained in Question 3 when promiscuous mode is turned off, the attacker's machine cannot capture the ping packets generated by Host A, since those packets are not addressed to its MAC address. As a result no output is produced by the sniffer.

Task 2.1 B : Writing Filters

Step 1 : Capture the ICMP packets between two specific hosts:
Attacker:

```
seed@seedvm2004: /mnt/hgfs/CNS-S5/Lab2/volumes
$>gcc -o Task2.1B-ICMP Task2.1B-ICMP.c -lpcap
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>./Task2.1B-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
       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
^C
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>
```

Host A:

```
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>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=0.454 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.117 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.152 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.154 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.154 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.140 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.151 ms
^C
--- 10.9.0.6 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6130ms
rtt min/avg/max/mdev = 0.117/0.188/0.454/0.108 ms
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	2025-08-19 07:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0009, seq=1/256, ttl=64 (reply in 2)
2	2025-08-19 07:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0009, seq=1/256, ttl=64 (request in 1)
3	2025-08-19 07:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0009, seq=2/512, ttl=64 (reply in 4)
4	2025-08-19 07:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0009, seq=2/512, ttl=64 (request in 3)
5	2025-08-19 07:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0009, seq=3/768, ttl=64 (reply in 6)
6	2025-08-19 07:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0009, seq=3/768, ttl=64 (request in 5)
7	2025-08-19 07:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0009, seq=4/1024, ttl=64 (reply in 8)
8	2025-08-19 07:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0009, seq=4/1024, ttl=64 (request in 7)
9	2025-08-19 07:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0009, seq=5/1280, ttl=64 (reply in 10)
10	2025-08-19 07:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0009, seq=5/1280, ttl=64 (request in 9)
11	2025-08-19 07:0…	ee:7a:64:f2:b5:90	42:80:0a:79:ca:28	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6
12	2025-08-19 07:0…	42:80:0a:79:ca:28	ee:7a:64:f2:b5:90	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
13	2025-08-19 07:0…	42:80:0a:79:ca:28	ee:7a:64:f2:b5:90	ARP	42	10.9.0.5 is at 42:80:0a:79:ca:28
14	2025-08-19 07:0…	ee:7a:64:f2:b5:90	42:80:0a:79:ca:28	ARP	42	10.9.0.6 is at ee:7a:64:f2:b5:90
15	2025-08-19 07:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0009, seq=6/1536, ttl=64 (reply in 16)
16	2025-08-19 07:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0009, seq=6/1536, ttl=64 (request in 15)
17	2025-08-19 07:0…	10.9.0.5	10.9.0.6	ICMP	98	Echo (ping) request id=0x0009, seq=7/1792, ttl=64 (reply in 18)
18	2025-08-19 07:0…	10.9.0.6	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0009, seq=7/1792, ttl=64 (request in 17)
19	2025-08-19 07:0…	10.9.0.1	224.0.0.251	MDNS	104	Standard query 0x0000 PTR _nmea-0183._tcp.local, "QM" question PTR _ipp._tcp…
20	2025-08-19 07:0…	fe80::a4ce:aeff:fe2…	ff02::fb	MDNS	124	Standard query 0x0000 PTR _nmea-0183._tcp.local, "QM" question PTR _ipp._tcp…

```
▸ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-fc6ddc5b87f2, id 0
▸ Ethernet II, Src: 42:80:0a:79:ca:28 (42:80:0a:79:ca:28), Dst: ee:7a:64:f2:b5:90 (ee:7a:64:f2:b5:90)
▸ Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
```

```
0000  ee 7a 64 f2 b5 90 42 80  0a 79 ca 28 08 00 45 00   ·zd···B· ·y·(··E·
0010  00 54 71 9e 40 00 40 01  b4 ee 0a 09 00 05 0a 09   ·Tq·@·@· ········
0020  00 06 08 00 b2 4c 00 09  00 01 f2 22 a4 68 00 00   ·····L·· ···"·h··
0030  00 00 e6 4a 0a 00 00 00  00 00 10 11 12 13 14 15   ···J···· ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········ ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37                                             67
```

The sniffer is configured with a filter that restricts capture to ICMP packets exchanged only between Host A (10.9.0.5) and Host B (10.9.0.6). When Host A pings Host B, the sniffer displays the source IP (10.9.0.5) and destination IP (10.9.0.6) for ICMP echo requests, and the reverse for ICMP replies. Any other ICMP traffic is ignored, ensuring that only communication between these two hosts is captured.

Step 2 : Capture the TCP packets that have a destination port range from to sort 10 - 100:

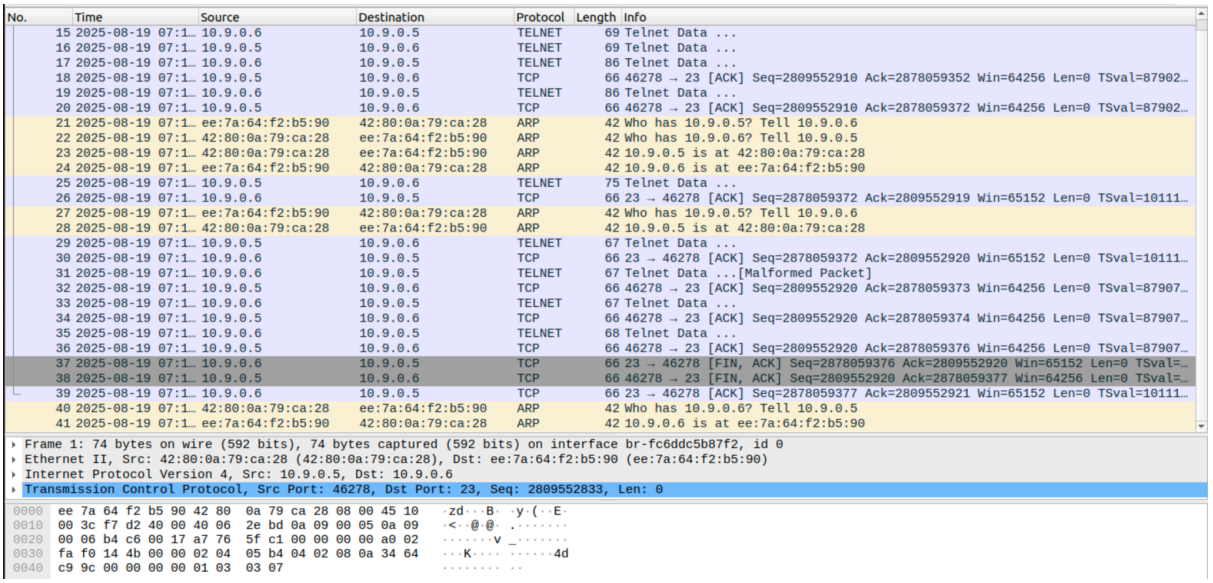
Attacker:

```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>nano Task2.1B-TCP.c
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>gcc -o Task2.1B-TCP Task2.1B-TCP.c -lpcap
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>./Task2.1B-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
^C
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>
```

Host A:

```
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.6 LTS
87c65ba2465b login: ^CConnection closed by foreign host.
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>
```

Wireshark:



The sniffer is set to capture only TCP packets with destination ports between 10 and 100. When Host A connects to Host B using Telnet on port 21, the packets fall within this range and are captured. The sniffer output shows the source and destination IPs with protocol TCP, while ignoring traffic on ports outside the specified range.

Task 2.1 C : Sniffing Passwords:

Attacker:

```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>nano Task2.1C.c
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>gcc -o Task2.1C Task2.1C.c -lpcap
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>./Task2.1C
tteellnneett  1100..99..00..66
Trying 10.9.0.6...
♦♦♦!Connected to 10.9.0.6.
Escape character is '^]'.
♦♦!♦♦Ubuntu 20.04.6 LTS
♦87c65ba2465b login: sseeeedd
lassword: dteees
nWelcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-151-generic aarch64)

 * 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 19 07:18:16 UTC 2025 from hostA-10.9.0.5.net-10.9.0.0 on pts/1
^C
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>
```

Host A:

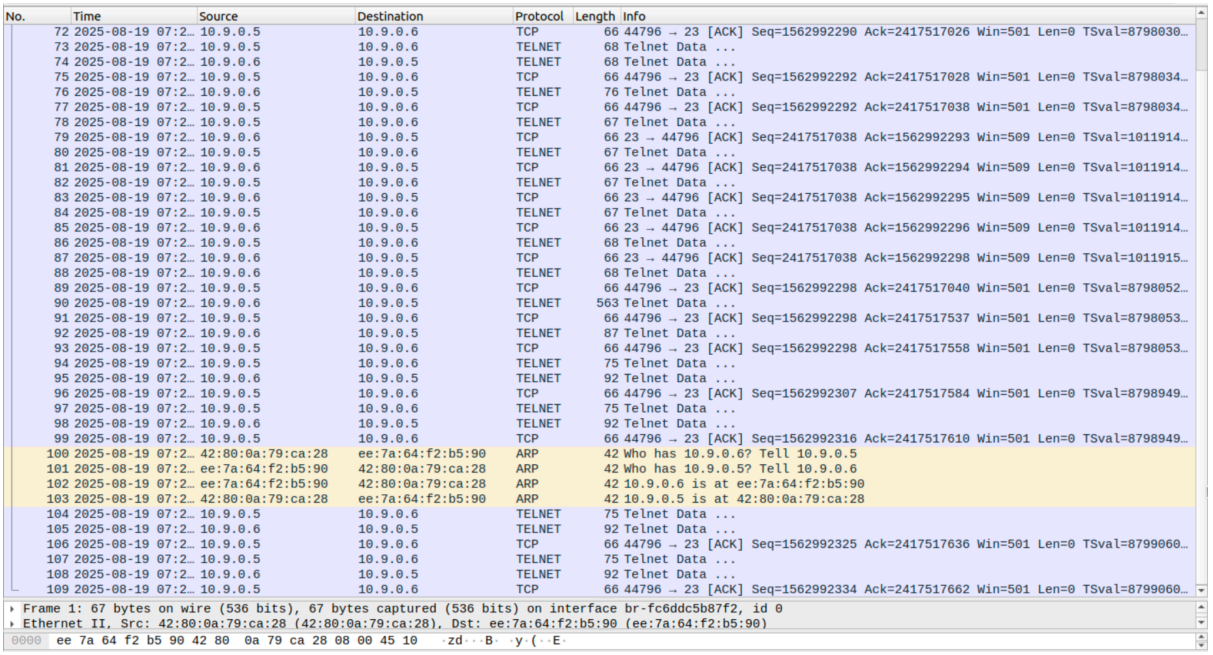
```
seed@87c65ba2465b:~$ PS1="seed-HostA:PES1UG23CS433:PranavHemanth:\w\n\$>"
seed-HostA:PES1UG23CS433:PranavHemanth:~
$>telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.6 LTS
87c65ba2465b login: seed
Password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-151-generic aarch64)

 * 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 19 07:18:16 UTC 2025 from hostA-10.9.0.5.net-10.9.0.0 on pts/1
seed@87c65ba2465b:~$
```

Wireshark:



No.	Time	Source	Destination	Protocol	Length	Info
72	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992290 Ack=2417517026 Win=501 Len=0 TSval=8798030…
73	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TELNET	68	Telnet Data ...
74	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TELNET	68	Telnet Data ...
75	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992292 Ack=2417517028 Win=501 Len=0 TSval=8798034…
76	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TELNET	76	Telnet Data ...
77	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992292 Ack=2417517038 Win=501 Len=0 TSval=8798034…
78	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...
79	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TCP	66	23 → 44796 [ACK] Seq=2417517038 Ack=1562992293 Win=509 Len=0 TSval=1011914…
80	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...
81	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TCP	66	23 → 44796 [ACK] Seq=2417517038 Ack=1562992294 Win=509 Len=0 TSval=1011914…
82	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...
83	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TCP	66	23 → 44796 [ACK] Seq=2417517038 Ack=1562992295 Win=509 Len=0 TSval=1011914…
84	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...
85	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TCP	66	23 → 44796 [ACK] Seq=2417517038 Ack=1562992296 Win=509 Len=0 TSval=1011914…
86	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TELNET	68	Telnet Data ...
87	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TCP	66	23 → 44796 [ACK] Seq=2417517038 Ack=1562992298 Win=509 Len=0 TSval=1011915…
88	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TELNET	68	Telnet Data ...
89	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992298 Ack=2417517040 Win=501 Len=0 TSval=8798052…
90	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TELNET	563	Telnet Data ...
91	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992298 Ack=2417517537 Win=501 Len=0 TSval=8798053…
92	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TELNET	87	Telnet Data ...
93	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992298 Ack=2417517558 Win=501 Len=0 TSval=8798053…
94	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TELNET	75	Telnet Data ...
95	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TELNET	92	Telnet Data ...
96	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992307 Ack=2417517584 Win=501 Len=0 TSval=8798949…
97	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TELNET	75	Telnet Data ...
98	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TELNET	92	Telnet Data ...
99	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992316 Ack=2417517610 Win=501 Len=0 TSval=8798949…
100	2025-08-19 07:2…	42:80:0a:79:ca:28	ee:7a:64:f2:b5:90	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
101	2025-08-19 07:2…	ee:7a:64:f2:b5:90	42:80:0a:79:ca:28	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6
102	2025-08-19 07:2…	ee:7a:64:f2:b5:90	42:80:0a:79:ca:28	ARP	42	10.9.0.6 is at ee:7a:64:f2:b5:90
103	2025-08-19 07:2…	42:80:0a:79:ca:28	ee:7a:64:f2:b5:90	ARP	42	10.9.0.5 is at 42:80:0a:79:ca:28
104	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TELNET	75	Telnet Data ...
105	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TELNET	92	Telnet Data ...
106	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992325 Ack=2417517636 Win=501 Len=0 TSval=8799060…
107	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TELNET	75	Telnet Data ...
108	2025-08-19 07:2…	10.9.0.6	10.9.0.5	TELNET	92	Telnet Data ...
109	2025-08-19 07:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992334 Ack=2417517662 Win=501 Len=0 TSval=8799060…

```
▸ Frame 1: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface br-fc6ddc5b87f2, id 0
▸ Ethernet II, Src: 42:80:0a:79:ca:28 (42:80:0a:79:ca:28), Dst: ee:7a:64:f2:b5:90 (ee:7a:64:f2:b5:90)
0000  ee 7a 64 f2 b5 90 42 80  0a 79 ca 28 08 00 45 10   ·zd···B· ·y·(··E·
```

The sniffer is set to capture traffic on port 23 (Telnet) and prints the data portion of the packets. Since Telnet sends credentials in plaintext, the username and password entered by Host A are visible in the captured payload. Thus, the attacker can observe and extract the password directly from the sniffed Telnet session.

Task 2.2 Spoofing

Task 2.2 B : Spoof an ICMP Echo Request:

Attacker:

```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>nano Task2.2.c
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>gcc -o Task2.2 Task2.2.c -lpcap
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>./Task2.2
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>./Task2.2
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes
$>
```

Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	2025-08-19 08:1…	1.2.3.4	10.9.0.6	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=20 (reply in 2)
2	2025-08-19 08:1…	10.9.0.6	1.2.3.4	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 1)

The program crafts and sends a spoofed ICMP echo request with a fake source IP (1.2.3.4) to the victim machine (10.9.0.6). In Wireshark, we observe the ICMP echo request appearing to originate from 1.2.3.4 instead of the attacker's IP. This demonstrates how raw sockets allow full packet manipulation and enable IP spoofing.

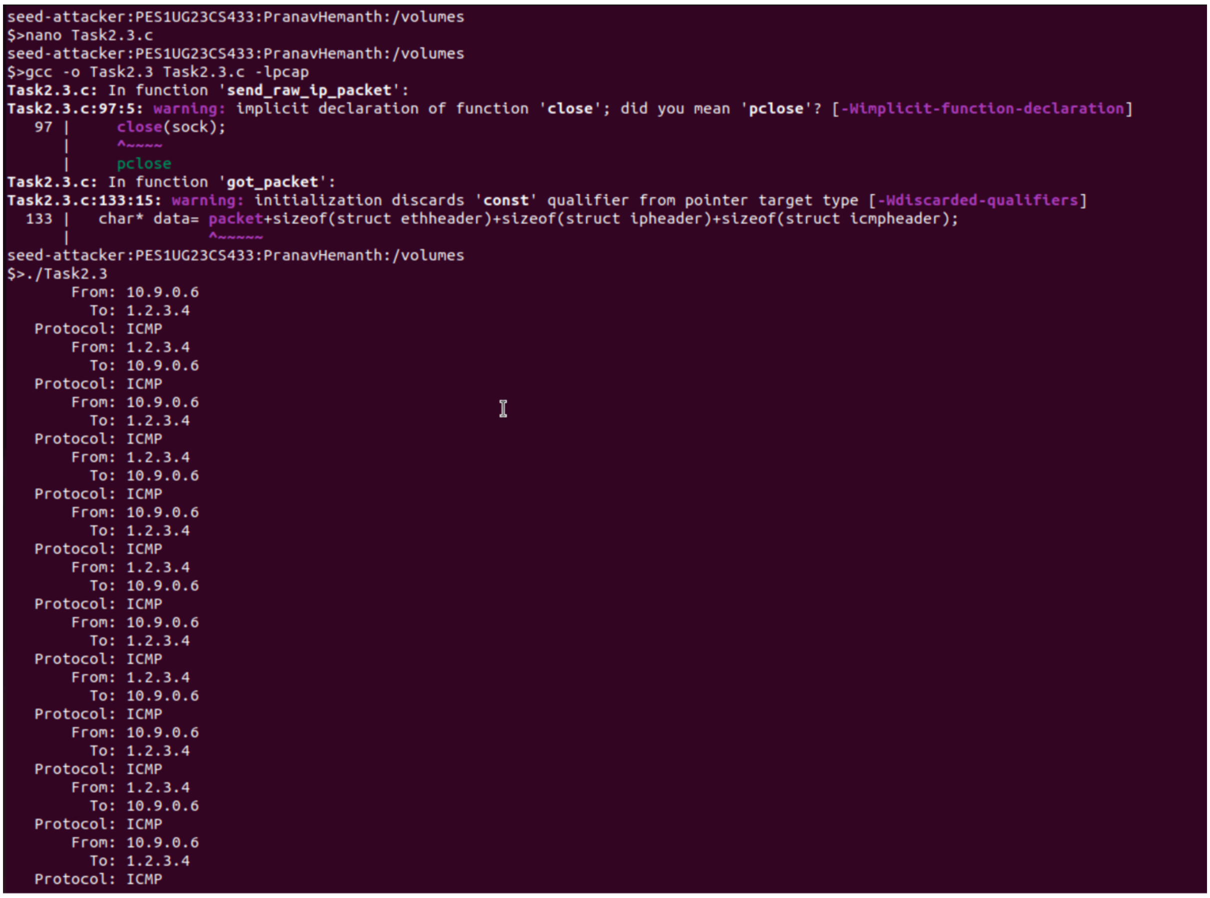
Question 4: Using the raw socket programming, do you have to calculate the checksum for the IP header?

Answer:
No, we don't have to calculate the checksum for the IP header. The operating system automatically recalculates and fills in the correct IP header checksum before sending the packet, even if we set it to zero or an incorrect value. However, checksums for transport-layer protocols (ICMP, TCP, UDP) must still be computed manually.

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:
Programs using raw sockets need root privilege because they allow direct packet crafting and injection, which could be exploited for attacks. Without root, the socket() system call fails with “Operation not permitted,” preventing the raw socket from being created. Thus, the program stops right at socket creation.

Task 2.3 Sniff and then Spoof

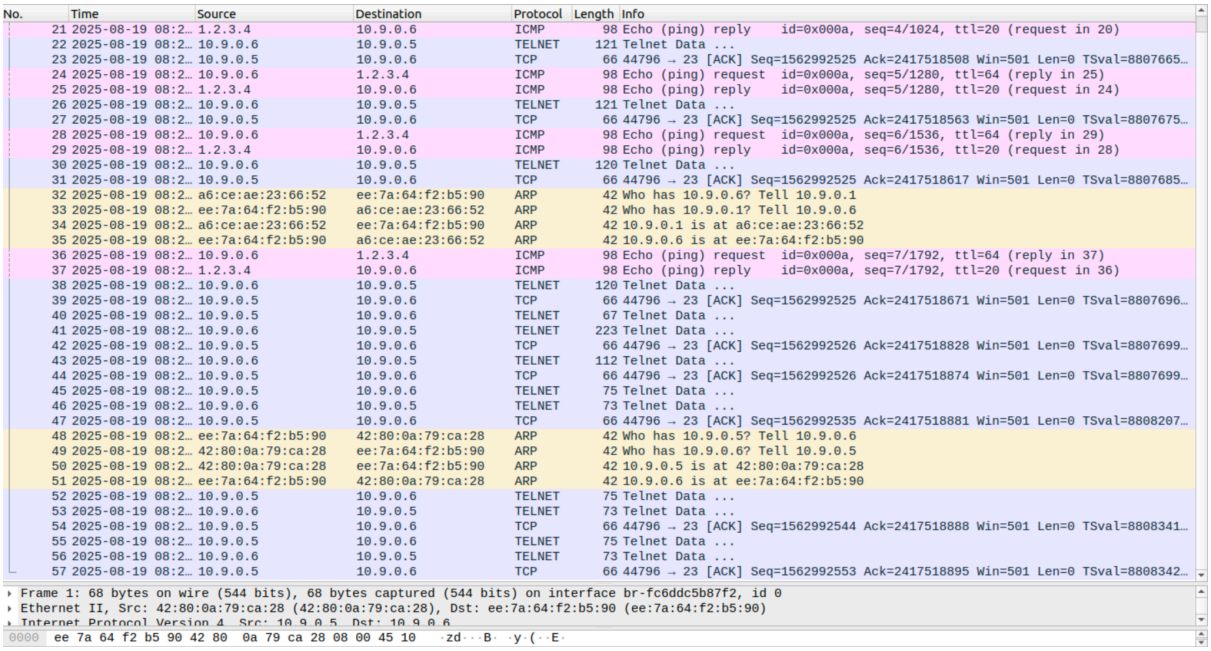
Attacker:



Host A:

```
seed-HostA:PES1UG23CS433:PranavHemanth:~
$>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=10.9 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=20 time=33.0 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=20 time=54.3 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=20 time=77.0 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=20 time=99.8 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=20 time=122 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=20 time=144 ms
^C
--- 1.2.3.4 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6011ms
rtt min/avg/max/mdev = 10.920/77.263/144.236/44.491 ms
seed-HostA:PES1UG23CS433:PranavHemanth:~
$>
```

Wireshark:



No.	Time	Source	Destination	Protocol	Length	Info
21	2025-08-19 08:2…	1.2.3.4	10.9.0.6	ICMP	98	Echo (ping) reply id=0x000a, seq=4/1024, ttl=20 (request in 20)
22	2025-08-19 08:2…	10.9.0.6	10.9.0.5	TELNET	121	Telnet Data ...
23	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992525 Ack=2417518508 Win=501 Len=0 TSval=8807665…
24	2025-08-19 08:2…	10.9.0.6	1.2.3.4	ICMP	98	Echo (ping) request id=0x000a, seq=5/1280, ttl=64 (reply in 25)
25	2025-08-19 08:2…	1.2.3.4	10.9.0.6	ICMP	98	Echo (ping) reply id=0x000a, seq=5/1280, ttl=20 (request in 24)
26	2025-08-19 08:2…	10.9.0.6	10.9.0.5	TELNET	121	Telnet Data ...
27	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992525 Ack=2417518563 Win=501 Len=0 TSval=8807675…
28	2025-08-19 08:2…	10.9.0.6	1.2.3.4	ICMP	98	Echo (ping) request id=0x000a, seq=6/1536, ttl=64 (reply in 29)
29	2025-08-19 08:2…	1.2.3.4	10.9.0.6	ICMP	98	Echo (ping) reply id=0x000a, seq=6/1536, ttl=20 (request in 28)
30	2025-08-19 08:2…	10.9.0.6	10.9.0.5	TELNET	120	Telnet Data ...
31	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992525 Ack=2417518617 Win=501 Len=0 TSval=8807685…
32	2025-08-19 08:2…	a6:ce:ae:23:66:52	ee:7a:64:f2:b5:90	ARP	42	Who has 10.9.0.6? Tell 10.9.0.1
33	2025-08-19 08:2…	ee:7a:64:f2:b5:90	a6:ce:ae:23:66:52	ARP	42	Who has 10.9.0.1? Tell 10.9.0.6
34	2025-08-19 08:2…	a6:ce:ae:23:66:52	ee:7a:64:f2:b5:90	ARP	42	10.9.0.1 is at a6:ce:ae:23:66:52
35	2025-08-19 08:2…	ee:7a:64:f2:b5:90	a6:ce:ae:23:66:52	ARP	42	10.9.0.6 is at ee:7a:64:f2:b5:90
36	2025-08-19 08:2…	10.9.0.6	1.2.3.4	ICMP	98	Echo (ping) request id=0x000a, seq=7/1792, ttl=64 (reply in 37)
37	2025-08-19 08:2…	1.2.3.4	10.9.0.6	ICMP	98	Echo (ping) reply id=0x000a, seq=7/1792, ttl=20 (request in 36)
38	2025-08-19 08:2…	10.9.0.6	10.9.0.5	TELNET	120	Telnet Data ...
39	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992525 Ack=2417518671 Win=501 Len=0 TSval=8807696…
40	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TELNET	67	Telnet Data ...
41	2025-08-19 08:2…	10.9.0.6	10.9.0.5	TELNET	223	Telnet Data ...
42	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992526 Ack=2417518828 Win=501 Len=0 TSval=8807699…
43	2025-08-19 08:2…	10.9.0.6	10.9.0.5	TELNET	112	Telnet Data ...
44	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992526 Ack=2417518874 Win=501 Len=0 TSval=8807699…
45	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TELNET	75	Telnet Data ...
46	2025-08-19 08:2…	10.9.0.6	10.9.0.5	TELNET	73	Telnet Data ...
47	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992535 Ack=2417518881 Win=501 Len=0 TSval=8808207…
48	2025-08-19 08:2…	ee:7a:64:f2:b5:90	42:80:0a:79:ca:28	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6
49	2025-08-19 08:2…	42:80:0a:79:ca:28	ee:7a:64:f2:b5:90	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
50	2025-08-19 08:2…	42:80:0a:79:ca:28	ee:7a:64:f2:b5:90	ARP	42	10.9.0.5 is at 42:80:0a:79:ca:28
51	2025-08-19 08:2…	ee:7a:64:f2:b5:90	42:80:0a:79:ca:28	ARP	42	10.9.0.6 is at ee:7a:64:f2:b5:90
52	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TELNET	75	Telnet Data ...
53	2025-08-19 08:2…	10.9.0.6	10.9.0.5	TELNET	73	Telnet Data ...
54	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992544 Ack=2417518888 Win=501 Len=0 TSval=8808341…
55	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TELNET	75	Telnet Data ...
56	2025-08-19 08:2…	10.9.0.6	10.9.0.5	TELNET	73	Telnet Data ...
57	2025-08-19 08:2…	10.9.0.5	10.9.0.6	TCP	66	44796 → 23 [ACK] Seq=1562992553 Ack=2417518895 Win=501 Len=0 TSval=8808342…

```
▸ Frame 1: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface br-fc6ddc5b87f2, id 0
▸ Ethernet II, Src: 42:80:0a:79:ca:28 (42:80:0a:79:ca:28), Dst: ee:7a:64:f2:b5:90 (ee:7a:64:f2:b5:90)
▸ Internet Protocol Version 4, Src: 10.9.0.5, Dst: 10.9.0.6
0000  ee 7a 64 f2 b5 90 42 80  0a 79 ca 28 08 00 45 10   ·zd···B· ·y·(··E·
```

This program sniffs ICMP echo requests (ping) using pcap, extracts their IP and ICMP headers, and then crafts a spoofed ICMP echo reply using raw sockets. It swaps the source and destination IPs, modifies the ICMP type to “reply,” recalculates the checksums, and sends it back to the victim. As a result, when the victim pings the fake IP 1.2.3.4, they still receive replies, making it appear alive.