PES UNIVERSITY

Department of Computer Science & Engineering

Computer Network Security

UE23CS343AB6

Assignment 1 Submission

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

Aug -Dec 2025 Assignment SUBMISSION_UE23CS343AB6

Computer Network Security

UE23CS343AB6

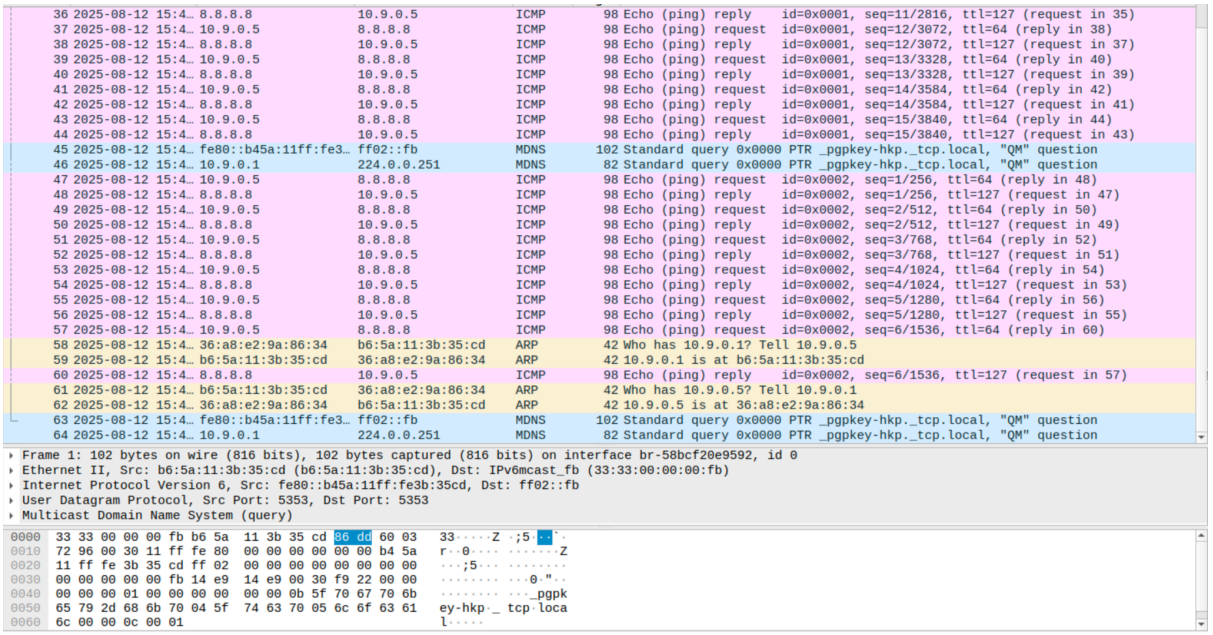
Task 1.1A: Sniff IP packets using Scapy

Step 1:

```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes/Code
$>python3 Task1.1A.py
SNIFFING PACKETS...
###[ Ethernet ]###
  dst       = 33:33:00:00:00:fb
  src       = b6:5a:11:3b:35:cd
  type      = IPv6
###[ IPv6 ]###
     version   = 6
     tc        = 0
     fl        = 225942
     plen      = 48
     nh        = UDP
     hlim      = 255
     src       = fe80::b45a:11ff:fe3b:35cd
     dst       = ff02::fb
###[ UDP ]###
        sport     = mdns
        dport     = mdns
        len       = 48
        chksum    = 0xf922
###[ DNS ]###
           id        = 0
           qr        = 0
           opcode    = QUERY
           aa        = 0
           tc        = 0
           rd        = 0
           ra        = 0
           z         = 0
           ad        = 0
           cd        = 0
           rcode     = ok
           qdcount   = 1
           ancount   = 0
           nscount   = 0
           arcount   = 0
           \qd        \
            |###[ DNS Question Record ]###
            |  qname     = '_pgpkey-hkp._tcp.local.'
            |  qtype     = PTR
            |  qclass    = IN
           an        = None
           ns        = None
           ar        = None

###[ Ethernet ]###
  dst       = 01:00:5e:00:00:fb
```

```
root@de8ce9aabcf8:/# PS1="seed-HostA:PES1UG23CS433:PranavHemanth:\w\n\$>"
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=127 time=39.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=127 time=34.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=127 time=49.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=127 time=38.8 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=127 time=36.4 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=127 time=63.5 ms
^C
--- 8.8.8.8 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 34.792/43.715/63.534/9.943 ms
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>
```



In this setup, a Python script running on the attacker machine uses Scapy's sniff function to capture network packets. Since the Docker container runs with root privileges by default, packet sniffing is possible. When Host A sends a ping request to 8.8.8.8, the attacker intercepts the ICMP echo request and its reply. This activity is visible both in Wireshark and in the terminal output from the Python script. The captured data includes the complete details of the ICMP packets, allowing the attacker to see the communication between Host A and 8.8.8.8.

Step 2:

```
$>export PS1="seed-attacker:PES1UG23CS433:PranavHemanth:\w\n\$>"
seed-attacker:PES1UG23CS433:PranavHemanth:~/Downloads/Labsetup-arm/volumes/Code
$>python3 Task1.1A.py
SNIFFING PACKETS...
Traceback (most recent call last):
  File "/home/seed/Downloads/Labsetup-arm/volumes/Code/Task1.1A.py", line 6, in <module>
    pkt = sniff(iface = "br-58bcf20e9592",prn=print_pkt)
  File "/usr/local/lib/python3.10/dist-packages/scapy/sendrecv.py", line 1424, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/scapy/sendrecv.py", line 1273, in _run
    sniff_sockets[_RL2(iface)(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.10/dist-packages/scapy/arch/linux/__init__.py", line 218, in __init__
    self.ins = socket.socket(
  File "/usr/lib/python3.10/socket.py", line 232, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
seed-attacker:PES1UG23CS433:PranavHemanth:~/Downloads/Labsetup-arm/volumes/Code
$>
```

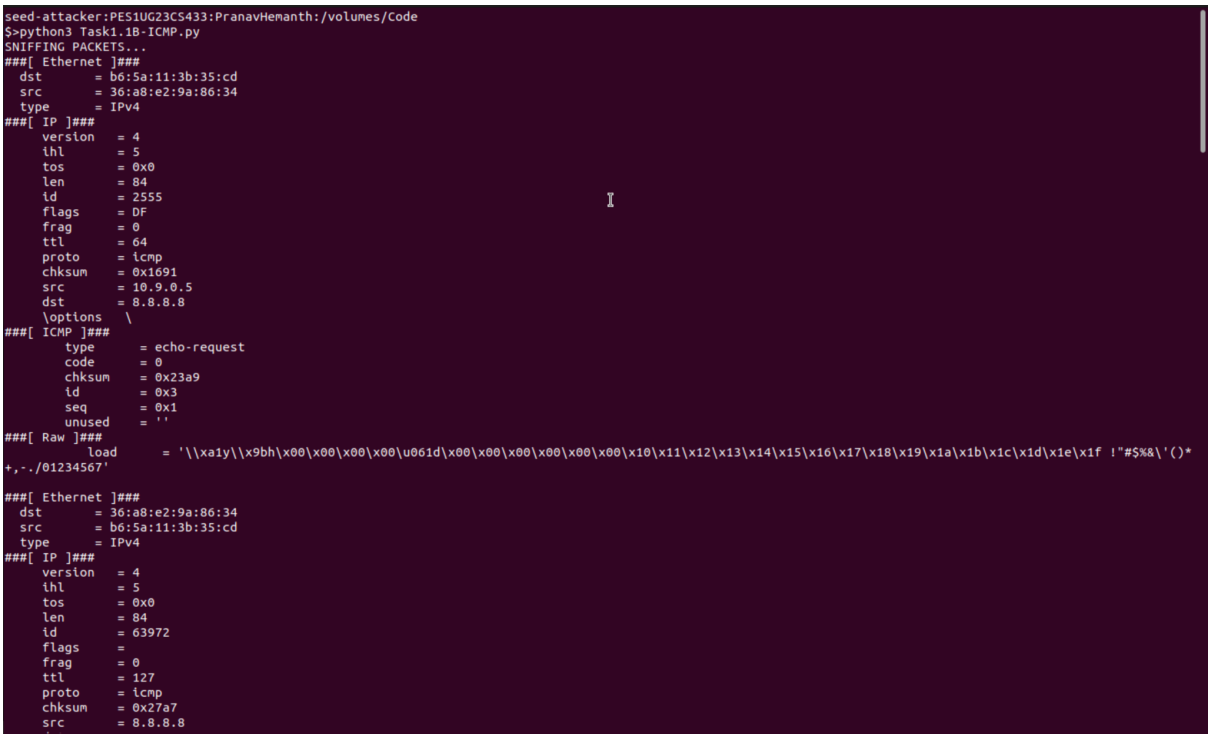
Before running the script, root privileges are removed. Executing it results in a PermissionError, showing that packet sniffing is only possible when the attacker has root access. This is because sending spoofed ICMP packets requires creating raw sockets. Only the root user or privileged processes can create raw sockets, as they allow creating arbitrary packets and bypassing the normal network stack.

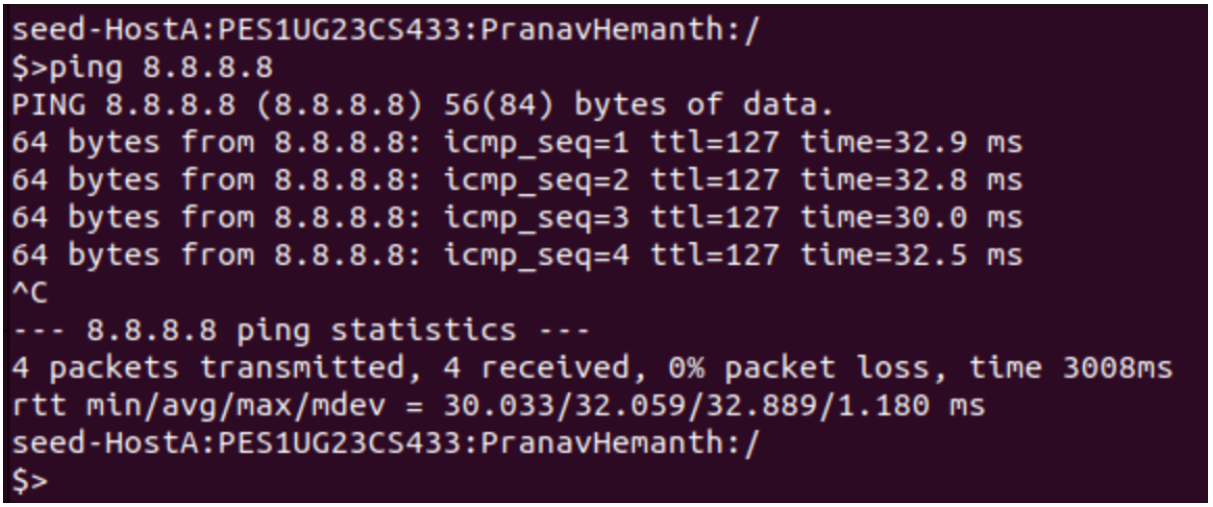
Step 3:

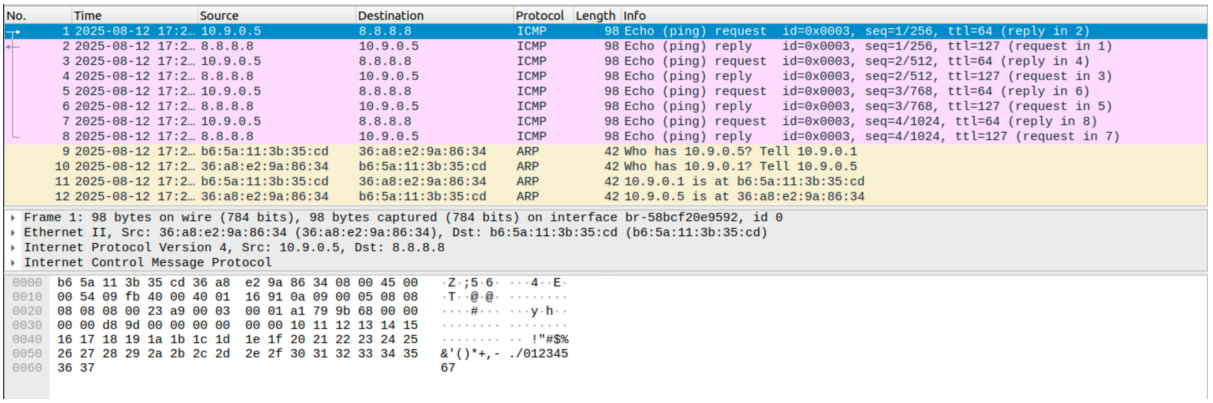
```
$>export PS1="seed-attacker:PES1UG23CS433:PranavHemanth:\w\n\$>"
seed-attacker:PES1UG23CS433:PranavHemanth:~/Downloads/Labsetup-arm/volumes/Code
$>python3 Task1.1A.py
SNIFFING PACKETS...
```

Task 1.1B : Applying Packet Filters to Capture certain types of packets for example : ICMP, TCP packet and Subnet

Step 1:







No.	Time	Source	Destination	Protocol	Length	Info
1	2025-08-12 17:2…	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x0003, seq=1/256, ttl=64 (reply in 2)
2	2025-08-12 17:2…	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0003, seq=1/256, ttl=127 (request in 1)
3	2025-08-12 17:2…	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x0003, seq=2/512, ttl=64 (reply in 4)
4	2025-08-12 17:2…	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0003, seq=2/512, ttl=127 (request in 3)
5	2025-08-12 17:2…	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x0003, seq=3/768, ttl=64 (reply in 6)
6	2025-08-12 17:2…	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0003, seq=3/768, ttl=127 (request in 5)
7	2025-08-12 17:2…	10.9.0.5	8.8.8.8	ICMP	98	Echo (ping) request id=0x0003, seq=4/1024, ttl=64 (reply in 8)
8	2025-08-12 17:2…	8.8.8.8	10.9.0.5	ICMP	98	Echo (ping) reply id=0x0003, seq=4/1024, ttl=127 (request in 7)
9	2025-08-12 17:2…	b6:5a:11:3b:35:cd	36:a8:e2:9a:86:34	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
10	2025-08-12 17:2…	36:a8:e2:9a:86:34	b6:5a:11:3b:35:cd	ARP	42	Who has 10.9.0.1? Tell 10.9.0.5
11	2025-08-12 17:2…	b6:5a:11:3b:35:cd	36:a8:e2:9a:86:34	ARP	42	10.9.0.1 is at b6:5a:11:3b:35:cd
12	2025-08-12 17:2…	36:a8:e2:9a:86:34	b6:5a:11:3b:35:cd	ARP	42	10.9.0.5 is at 36:a8:e2:9a:86:34

```
▸ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-58bcf20e9592, id 0
▸ Ethernet II, Src: 36:a8:e2:9a:86:34 (36:a8:e2:9a:86:34), Dst: b6:5a:11:3b:35:cd (b6:5a:11:3b:35:cd)
▸ Internet Protocol Version 4, Src: 10.9.0.5, Dst: 8.8.8.8
▸ Internet Control Message Protocol
```

```
0000  b6 5a 11 3b 35 cd 36 a8  e2 9a 86 34 08 00 45 00   ·Z·;5·6· ···4··E·
0010  00 54 09 fb 40 00 40 01  16 91 0a 09 00 05 08 08   ·T··@·@· ········
0020  08 08 08 00 23 a9 00 03  00 01 a1 79 9b 68 00 00   ····#··· ···y·h··
0030  00 00 d8 9d 00 00 00 00  00 00 10 11 12 13 14 15   ········ ········
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
```

In this task, the sniffer script was set up and run while Wireshark simultaneously captured traffic on the same interface. The filter feature of sniff to capture only ICMP packets is used. When Host A sent ping requests to 8.8.8.8, both script and wireshark recorded the ICMP echo requests from Host A as well as echo replies from the destination. This confirmed that the sniffer was working correctly, capturing only ICMP traffic.

Step 2:

Aug -Dec 2025 Assignment SUBMISSION_UE23CS343AB6

```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes/Code
$>python3 Task1.1B-TCP.py
SNIFFING PACKETS...
###[ Ethernet ]###
  dst       = b6:5a:11:3b:35:cd
  src       = 36:a8:e2:9a:86:34
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 60
     id        = 52598
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x5318
     src       = 10.9.0.5
     dst       = 8.8.8.8
     \options   \
###[ TCP ]###
        sport     = 40530
        dport     = telnet
        seq       = 3299790136
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x1a4c
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (46484658, 0)), ('NOP', None), ('WScale', 7)]

###[ Ethernet ]###
  dst       = b6:5a:11:3b:35:cd
  src       = 36:a8:e2:9a:86:34
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 60
     id        = 52599
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
```

```
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>telnet 8.8.8.8
Trying 8.8.8.8...
telnet: Unable to connect to remote host: Connection refused
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>
```

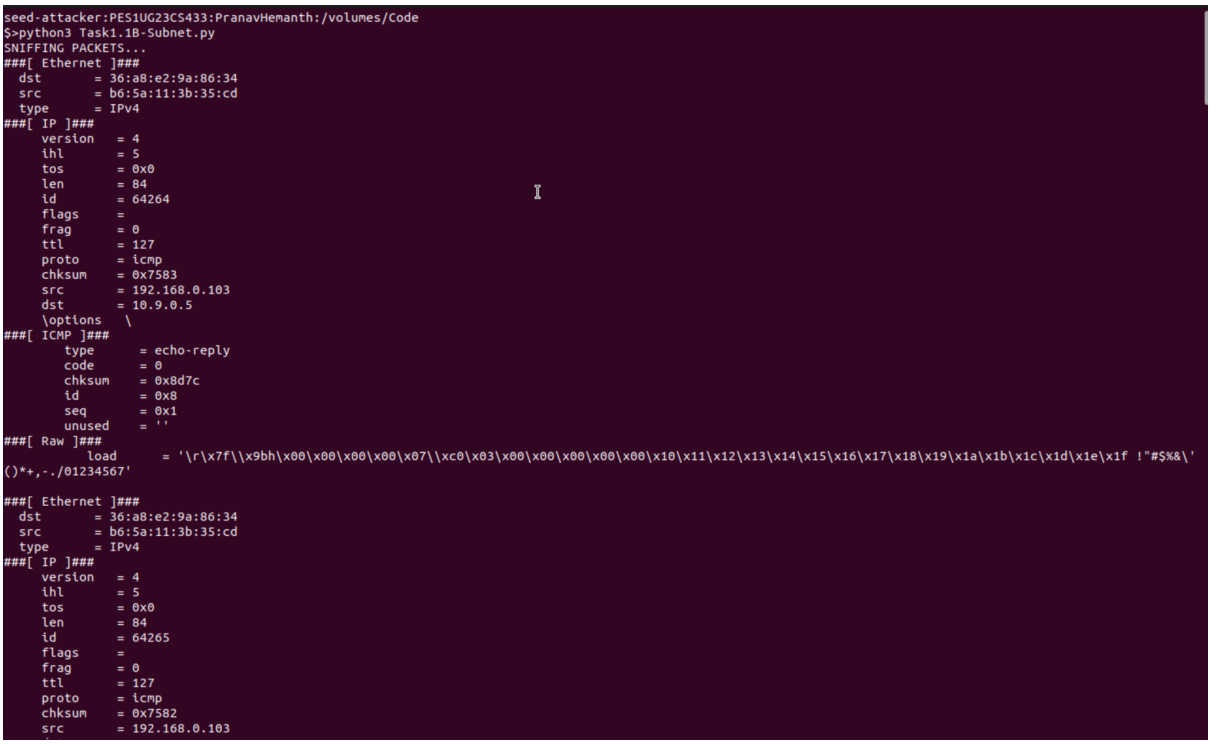
No.	Time	Source	Destination	Protocol	Length	Info
1	2025-08-12 17:3…	fe80::b45a:11ff:fe3…	ff02::fb	MDNS	107	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.loca…
2	2025-08-12 17:3…	10.9.0.5	8.8.8.8	TCP	74	40530 → 23 [SYN] Seq=3299790136 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=4…
3	2025-08-12 17:3…	10.9.0.5	8.8.8.8	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 40530 → 23 [SYN] Seq=32997901…
4	2025-08-12 17:3…	10.9.0.5	8.8.8.8	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 40530 → 23 [SYN] Seq=32997901…
5	2025-08-12 17:3…	36:a8:e2:9a:86:34	b6:5a:11:3b:35:cd	ARP	42	Who has 10.9.0.1? Tell 10.9.0.5
6	2025-08-12 17:3…	b6:5a:11:3b:35:cd	36:a8:e2:9a:86:34	ARP	42	10.9.0.1 is at b6:5a:11:3b:35:cd
7	2025-08-12 17:3…	10.9.0.5	8.8.8.8	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 40530 → 23 [SYN] Seq=32997901…
8	2025-08-12 17:3…	10.9.0.5	8.8.8.8	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 40530 → 23 [SYN] Seq=32997901…
9	2025-08-12 17:3…	10.9.0.5	8.8.8.8	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 40530 → 23 [SYN] Seq=32997901…
10	2025-08-12 17:3…	36:a8:e2:9a:86:34	b6:5a:11:3b:35:cd	ARP	42	Who has 10.9.0.1? Tell 10.9.0.5
11	2025-08-12 17:3…	b6:5a:11:3b:35:cd	36:a8:e2:9a:86:34	ARP	42	10.9.0.1 is at b6:5a:11:3b:35:cd
12	2025-08-12 17:3…	10.9.0.5	8.8.8.8	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 40530 → 23 [SYN] Seq=32997901…
13	2025-08-12 17:3…	36:a8:e2:9a:86:34	b6:5a:11:3b:35:cd	ARP	42	Who has 10.9.0.1? Tell 10.9.0.5
14	2025-08-12 17:3…	b6:5a:11:3b:35:cd	36:a8:e2:9a:86:34	ARP	42	10.9.0.1 is at b6:5a:11:3b:35:cd
15	2025-08-12 17:3…	8.8.8.8	10.9.0.5	TCP	54	23 → 40530 [RST, ACK] Seq=229014201 Ack=3299790137 Win=64240 Len=0
16	2025-08-12 17:3…	b6:5a:11:3b:35:cd	36:a8:e2:9a:86:34	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
17	2025-08-12 17:3…	36:a8:e2:9a:86:34	b6:5a:11:3b:35:cd	ARP	42	10.9.0.5 is at 36:a8:e2:9a:86:34

```
▸ Frame 1: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface br-58bcf20e9592, id 0
▸ Ethernet II, Src: b6:5a:11:3b:35:cd (b6:5a:11:3b:35:cd), Dst: IPv6mcast_fb (33:33:00:00:00:fb)
▸ Internet Protocol Version 6, Src: fe80::b45a:11ff:fe3b:35cd, Dst: ff02::fb
▸ User Datagram Protocol, Src Port: 5353, Dst Port: 5353
▸ Multicast Domain Name System (query)
```

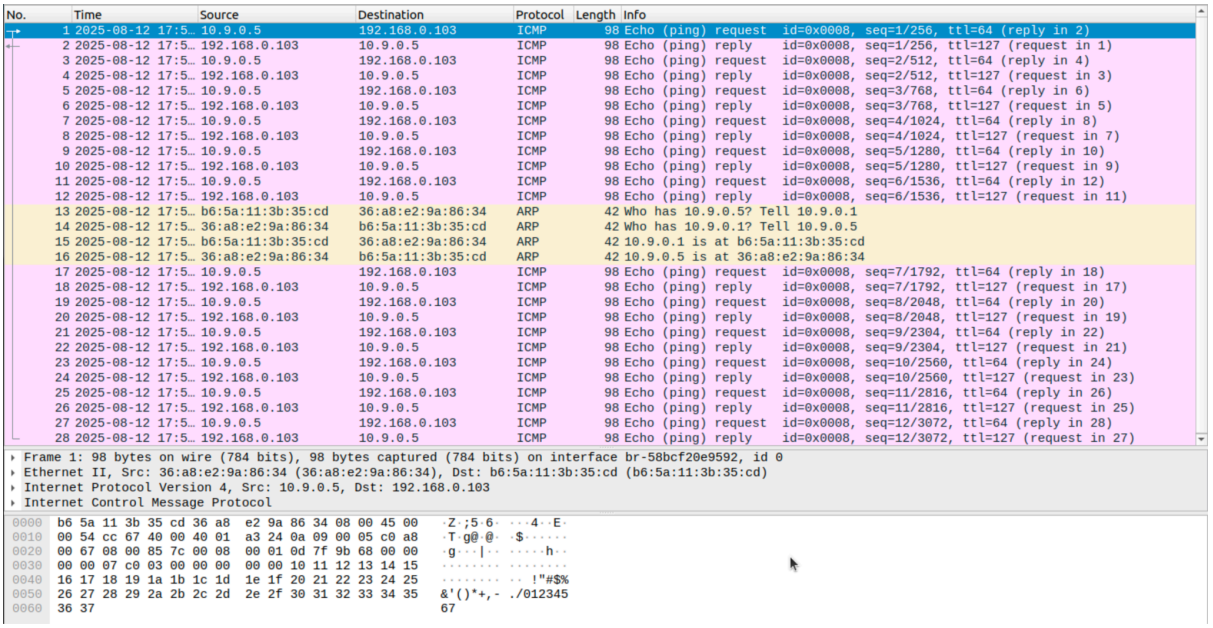
```
0000  33 33 00 00 00 fb b6 5a  11 3b 35 cd 86 dd 60 03   33·····Z ·;5···`·
0010  72 96 00 35 11 ff fe 80  00 00 00 00 00 00 b4 5a   r··5···· ·······Z
0020  11 ff fe 3b 35 cd ff 02  00 00 00 00 00 00 00 00   ···;5··· ········
0030  00 00 00 00 00 fb 14 e9  14 e9 00 35 f9 27 00 00   ········ ···5·'··
0040  00 00 00 02 00 00 00 00  00 00 05 5f 69 70 70 73   ········ ···_ipps
0050  04 5f 74 63 70 05 6c 6f  63 61 6c 00 00 0c 00 01   ·_tcp·lo cal·····
0060  04 5f 69 70 70 c0 12 00  0c 00 01                  ·_ipp··· ···
```

In this task, the script was configured to capture only TCP traffic from a specific IP on port 23 (Telnet) using the correct network interface. With Wireshark capturing on the same interface, Host A started a Telnet connection to 8.8.8.8. Both Scapy and Wireshark recorded the TCP packets, including the three-way handshake and Telnet data, confirming that the filter accurately isolated only TCP port 23 traffic and that both captures matched.

Step 3:



```
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>ping 192.168.0.103
PING 192.168.0.103 (192.168.0.103) 56(84) bytes of data.
64 bytes from 192.168.0.103: icmp_seq=1 ttl=127 time=1.36 ms
64 bytes from 192.168.0.103: icmp_seq=2 ttl=127 time=1.14 ms
64 bytes from 192.168.0.103: icmp_seq=3 ttl=127 time=1.96 ms
64 bytes from 192.168.0.103: icmp_seq=4 ttl=127 time=0.602 ms
64 bytes from 192.168.0.103: icmp_seq=5 ttl=127 time=0.801 ms
64 bytes from 192.168.0.103: icmp_seq=6 ttl=127 time=1.23 ms
64 bytes from 192.168.0.103: icmp_seq=7 ttl=127 time=0.733 ms
64 bytes from 192.168.0.103: icmp_seq=8 ttl=127 time=1.29 ms
64 bytes from 192.168.0.103: icmp_seq=9 ttl=127 time=1.05 ms
64 bytes from 192.168.0.103: icmp_seq=10 ttl=127 time=0.652 ms
64 bytes from 192.168.0.103: icmp_seq=11 ttl=127 time=1.07 ms
64 bytes from 192.168.0.103: icmp_seq=12 ttl=127 time=1.14 ms
^C
--- 192.168.0.103 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11071ms
rtt min/avg/max/mdev = 0.602/1.086/1.964/0.358 ms
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>
```

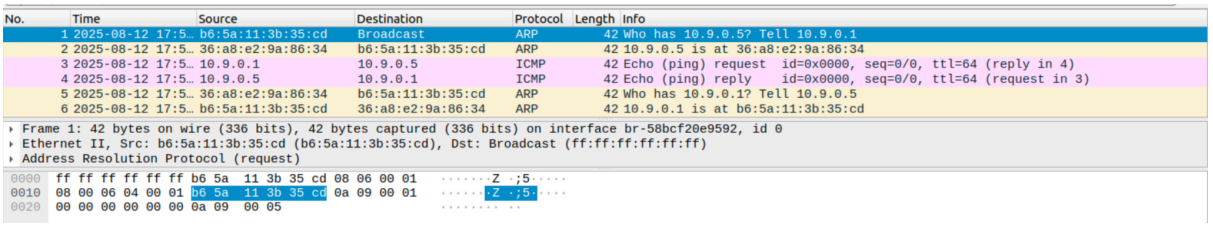


In this task, the attacker's script was set to capture traffic from the subnet 192.168.0.0/24, different from the VM's active subnet. When Host A pinged 192.168.0.103, the sniffer captured the ICMP packets matching the filter, despite the destination being outside the attacker's active network. Wireshark displayed the same packets, confirming that the Python sniffer accurately filtered and captured only the targeted subnet's traffic.

Task 1.2: Packet Spoofing

Step 1:

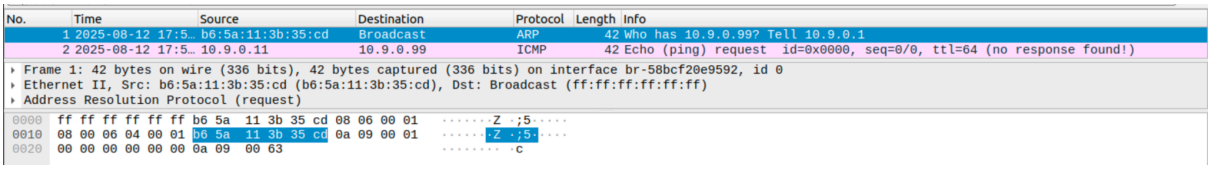
```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes/Code
$>python3 Task1.2A.py
SENDING SPOOFED ICMP PACKET...
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 10.9.0.1
  dst       = 10.9.0.5
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = None
     id        = 0x0
     seq       = 0x0
     unused    = ''
```



In this task, the script used Scapy to send a spoofed ICMP Echo Request, setting the source IP to another local machine and the destination to an active internet host. As a result, any Echo Reply was sent to the spoofed machine, not the attacker. Wireshark confirmed the forged packet details, demonstrating the attacker's ability to inject spoofed traffic into the network without receiving replies.

Step 2:

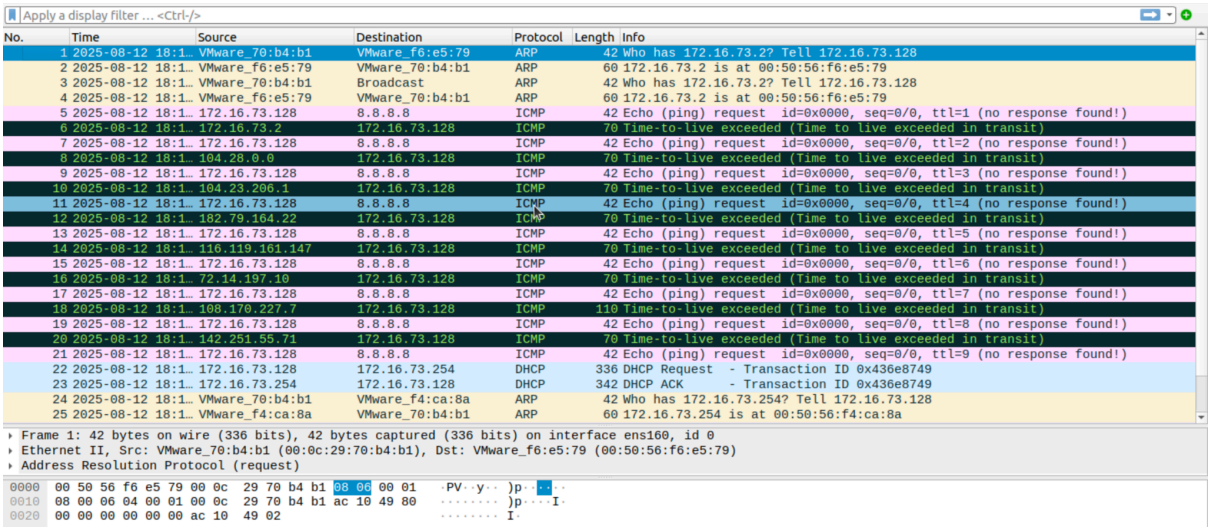
```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes/Code
$>python3 Task1.2B.py
SENDING SPOOFED ICMP PACKET...
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 10.9.0.11
  dst       = 10.9.0.99
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = None
     id        = 0x0
     seq       = 0x0
     unused    = ''
```



In this task, the script sent a spoofed ICMP Echo Request with a non-existent source IP and a chosen destination. Wireshark showed the fake source and real destination, confirming successful spoofing. Since the source was unreachable, no Echo Reply was received, illustrating how spoofing can conceal the true origin of network traffic.

Task 1.3: Traceroute

```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes/Code
$>python3 Task1.3.py 8.8.8.8
Traceroute 8.8.8.8
1 hops away:  172.16.73.2
2 hops away:  104.28.0.0
3 hops away:  104.23.206.1
4 hops away:  182.79.164.22
5 hops away:  116.119.161.147
6 hops away:  72.14.197.10
7 hops away:  108.170.227.7
8 hops away:  142.251.55.71
^Cseed-attacker:PES1UG23CS433:PranavHemanth:/volumes/Code
```

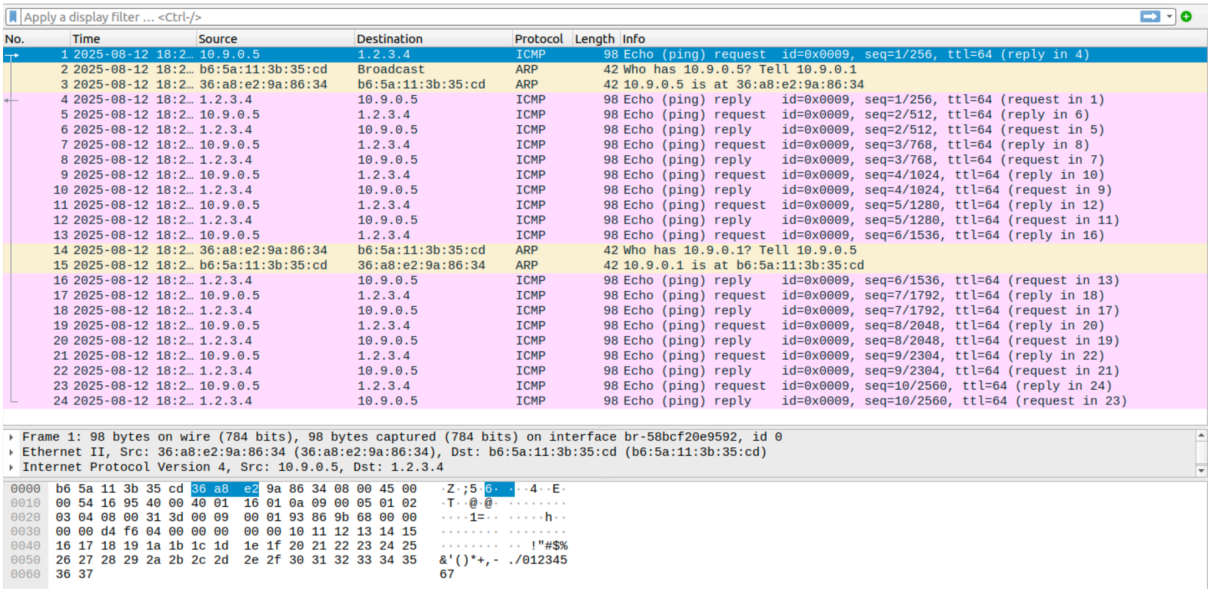


In this task, the attacker's script implemented a simple traceroute by sending ICMP Echo Requests with increasing TTL values to 8.8.8.8. Routers returning ICMP Time Exceeded messages revealed each hop, and the destination replied with an Echo Reply. Wireshark confirmed the sequence, showing that the route was only one hop before reaching the target.

Task 1.4: Sniffing and-then Spoofing

```
seed-attacker:PES1UG23CS433:PranavHemanth:/volumes/Code
$>python3 Task1.4.py
original packet.........
source IP : 10.9.0.5
Destination IP : 1.2.3.4
spoofed packet........
Source IP: 1.2.3.4
Destination IP: 10.9.0.5
original packet.........
source IP : 10.9.0.5
Destination IP : 1.2.3.4
spoofed packet........
Source IP: 1.2.3.4
Destination IP: 10.9.0.5
original packet.........
source IP : 10.9.0.5
Destination IP : 1.2.3.4
spoofed packet........
Source IP: 1.2.3.4
Destination IP: 10.9.0.5
original packet.........
source IP : 10.9.0.5
Destination IP : 1.2.3.4
spoofed packet........
Source IP: 1.2.3.4
Destination IP: 10.9.0.5
original packet.........
source IP : 10.9.0.5
Destination IP : 1.2.3.4
spoofed packet........
Source IP: 1.2.3.4
Destination IP: 10.9.0.5
original packet.........
source IP : 10.9.0.5
Destination IP : 1.2.3.4
spoofed packet........
Source IP: 1.2.3.4
Destination IP: 10.9.0.5
original packet.........
source IP : 10.9.0.5
Destination IP : 1.2.3.4
spoofed packet........
Source IP: 1.2.3.4
Destination IP: 10.9.0.5
original packet.........
source IP : 10.9.0.5
```

```
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=64 time=67.2 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=27.0 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=21.2 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=15.3 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=29.2 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=27.6 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=21.4 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=26.8 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=26.0 ms
64 bytes from 1.2.3.4: icmp_seq=10 ttl=64 time=25.3 ms
^C
--- 1.2.3.4 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9018ms
rtt min/avg/max/mdev = 15.260/28.685/67.223/13.426 ms
seed-HostA:PES1UG23CS433:PranavHemanth:/
$>
```



In this task, Host A pinged the non-existent IP 1.2.3.4, which the attacker's script sniffed. The attacker then sent a spoofed ICMP Echo Reply to Host A, copying the original packet's details but using 1.2.3.4 as the source. Wireshark showed the forged reply following the request, and Host A's terminal reported a successful ping, demonstrating how spoofing can create false network reachability.