

# H M Mythreya

## PES2UG20CS130

### CNS-Lab Week-1

## Setting up the lab

First thing I did was to setup the docker file given. This sets up a simulated environment with 3 machines, i.e, Host A, Host B and the Attacker.

## Task 1.1A: Sniffing Packets

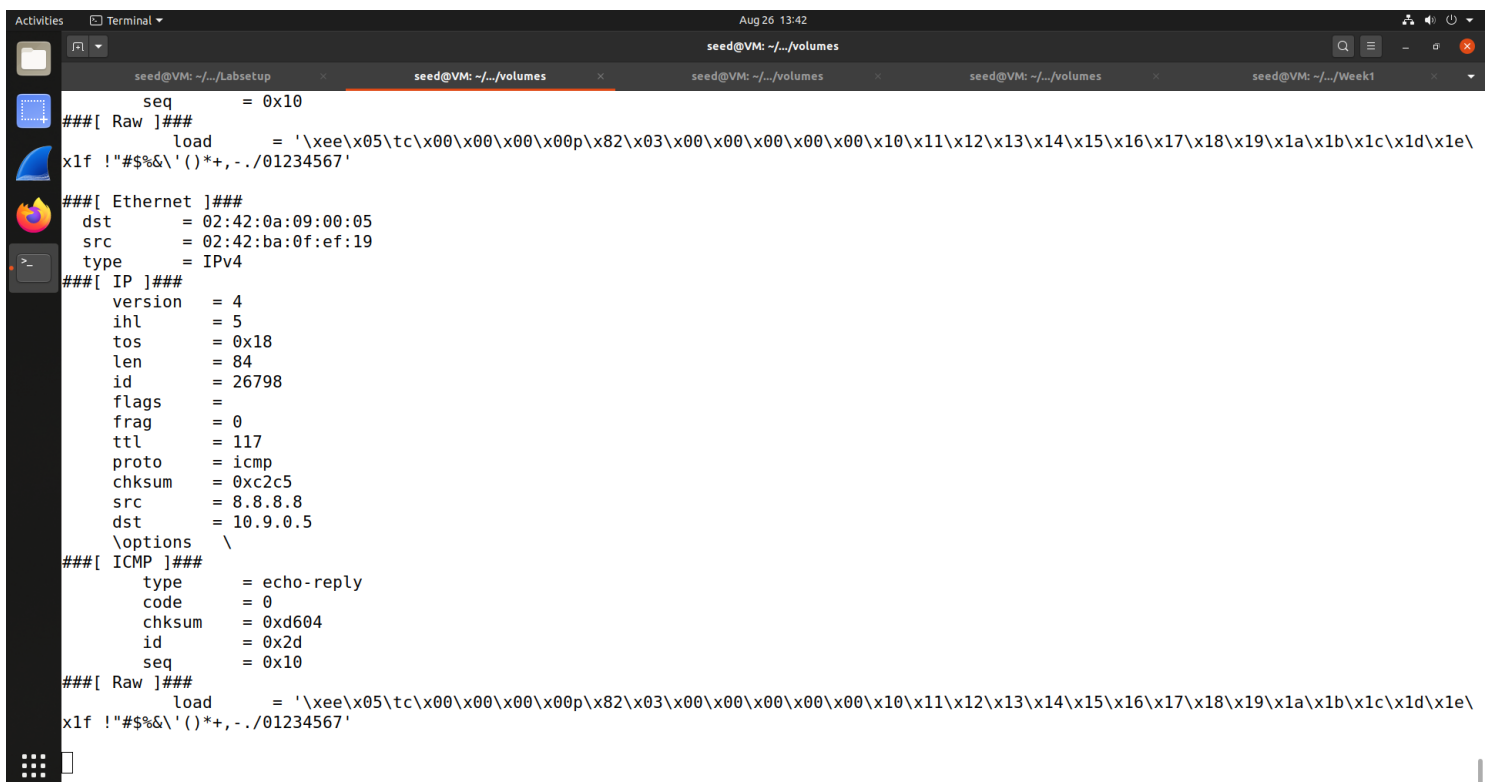
For this task, scapy was used. Initially, a ping command is run from host A, which pings host B. A ping is simply an ICMP packet used to check the presence of an I.P. At this point, on the attacker machine, the sniffing code is run.

The most important code is:

```
pkt = sniff(iface = "br-ea9efb963438",prn=print_pkt)
```

Here, a scapy function called sniff() is called. “iface” here stands for interface and the interface from which you want to sniff goes here. “prn” is the function that is going to be called after sniffing, where in we can do certain operations to the packet, in this case we just print it.

## Output



```
Aug 26 13:42
seed@VM: ~/../Labsetup
seq = 0x10
###[ Raw ]###
load = '\xee\x05\tc\x00\x00\x00p\x82\x03\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f'
xlf !"#%&'()*+,-./01234567

###[ Ethernet ]###
dst = 02:42:0a:09:00:05
src = 02:42:ba:0f:ef:19
type = IPv4
###[ IP ]###
version = 4
ihl = 5
tos = 0x18
len = 84
id = 26798
flags = 
frag = 0
ttl = 117
proto = icmp
chksum = 0xc2c5
src = 8.8.8.8
dst = 10.9.0.5
\options \
###[ ICMP ]###
type = echo-reply
code = 0
chksum = 0xd604
id = 0x2d
seq = 0x10
###[ Raw ]###
load = '\xee\x05\tc\x00\x00\x00p\x82\x03\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f'
xlf !"#%&'()*+,-./01234567
```

Observation: Running the same command without root privileges doesn't work. A `PermissionError` is returned upon calling `sniff()`

The screenshot shows a Kali Linux terminal window with the following content:

```

Aug 26 13:42
seed@VM: ~/../labsetup
seed@VM: ~/../volumes
seed@VM: ~/../volumes
seed@VM: ~/../volumes
seed@VM: ~/../Week1

flags =
frag = 0
ttl = 117
proto = icmp
checksum = 0xc2ae
src = 8.8.8.8
dst = 10.9.0.5
\options \
###[ ICMP ]###
    type = echo-reply
    code = 0
    checksum = 0x9b4e
    id = 0x2d
    seq = 0x27

###[ Raw ]###
    load = '\x05\x06\tc\x00\x00\x00\x00!\x05\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"%&'()*+,-./01234567'

^Croot@VM:/volumes/Code# su seed
seed@VM:/volumes/Code$ python3 Task1.1A.py
SNIFFING PACKETS...
Traceback (most recent call last):
  File "Task1.1A.py", line 6, in <module>
    pkt = sniff(iface = "br-ea9efb963438", prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1036, in sniff
    sniffer.run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 906, in _run
    sniff_socket(L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type)) # noqa: E501
  File "/usr/lib/python3.8/socket.py", line 231, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
seed@VM:/volumes/Code$

```

## Task 1.1B: Sniffing Packets

### Filters:

In the real world, a network will have thousands if not hundreds of thousands of packets flowing to and from. Going through every packet would be impractical, hence we use Filters. Filters only show the relevant packets depending on the filter. For example, using an ICMP filter will only sniff ICMP packets, and so on.

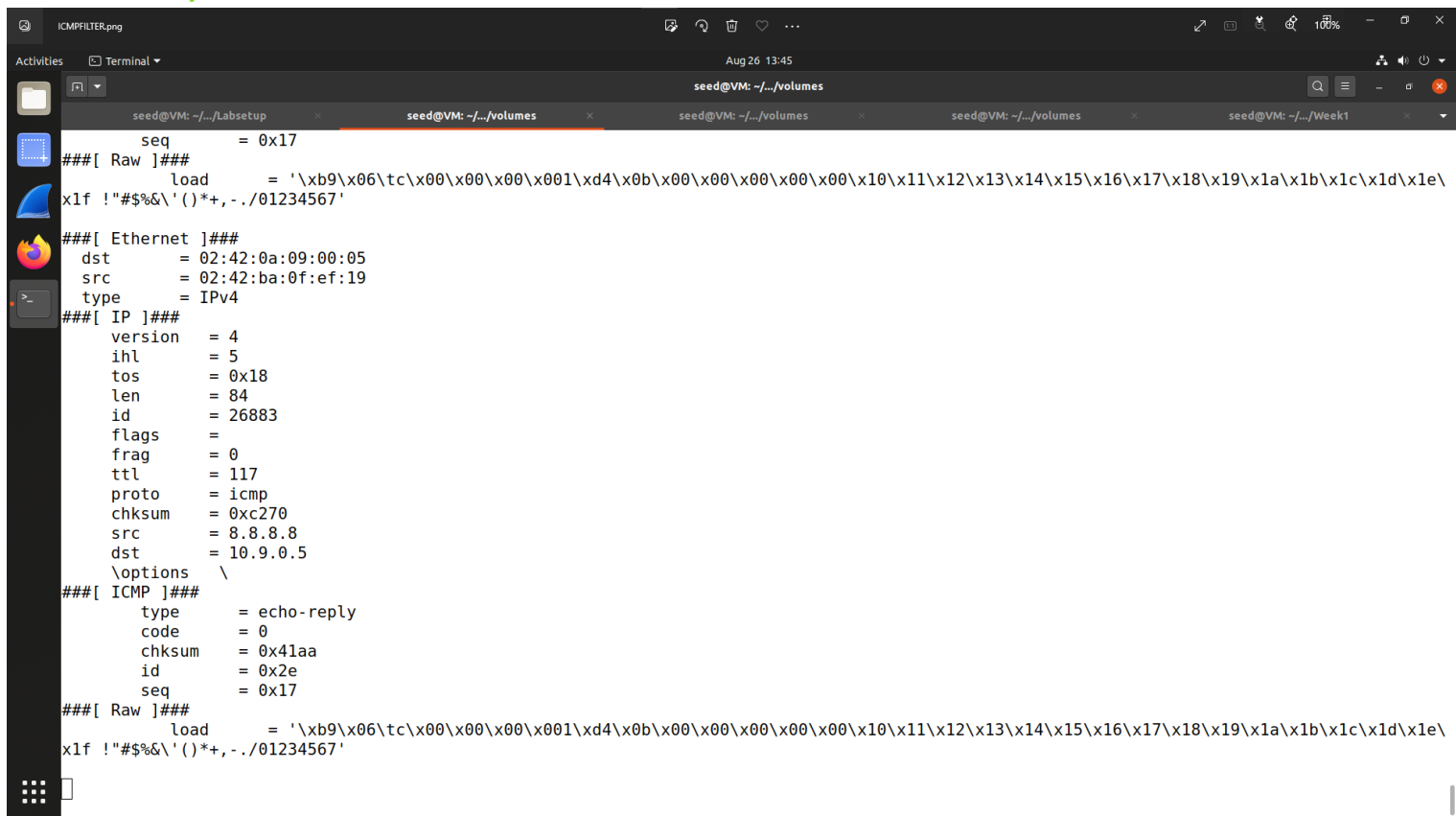
First, Host A pings Host B using the ping command. The attacker then runs the sniffing code with any filter they may desire.

### Filter 1) ICMP

```
pkt = sniff (iface = "br-ea9efb963438",filter='icmp', prn=print_pkt)
```

Here, we are using the same code as before, but with a new argument, the filter argument. For now let us pass in “icmp” as the argument and see what happens.

### Output



```
ICMPFILTER.png
Aug 26 13:45
seed@VM: ~/../volumes
seq = 0x17
###[ Raw ]###
load = '\xb9\x06\tc\x00\x00\x00\x00\x01\xd4\x0b\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./01234567'
###[ Ethernet ]###
dst = 02:42:0a:09:00:05
src = 02:42:ba:0f:ef:19
type = IPv4
###[ IP ]###
version = 4
ihl = 5
tos = 0x18
len = 84
id = 26883
flags = 
frag = 0
ttl = 117
proto = icmp
chksum = 0xc270
src = 8.8.8.8
dst = 10.9.0.5
\options \
###[ ICMP ]###
type = echo-reply
code = 0
chksum = 0x41aa
id = 0x2e
seq = 0x17
###[ Raw ]###
load = '\xb9\x06\tc\x00\x00\x00\x00\x01\xd4\x0b\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./01234567'
```

Here we can see that, the packets being sniffed are ICMP packets. We are sniffing the packets sent out by Host A, since “ping” is an ICMP protocol.

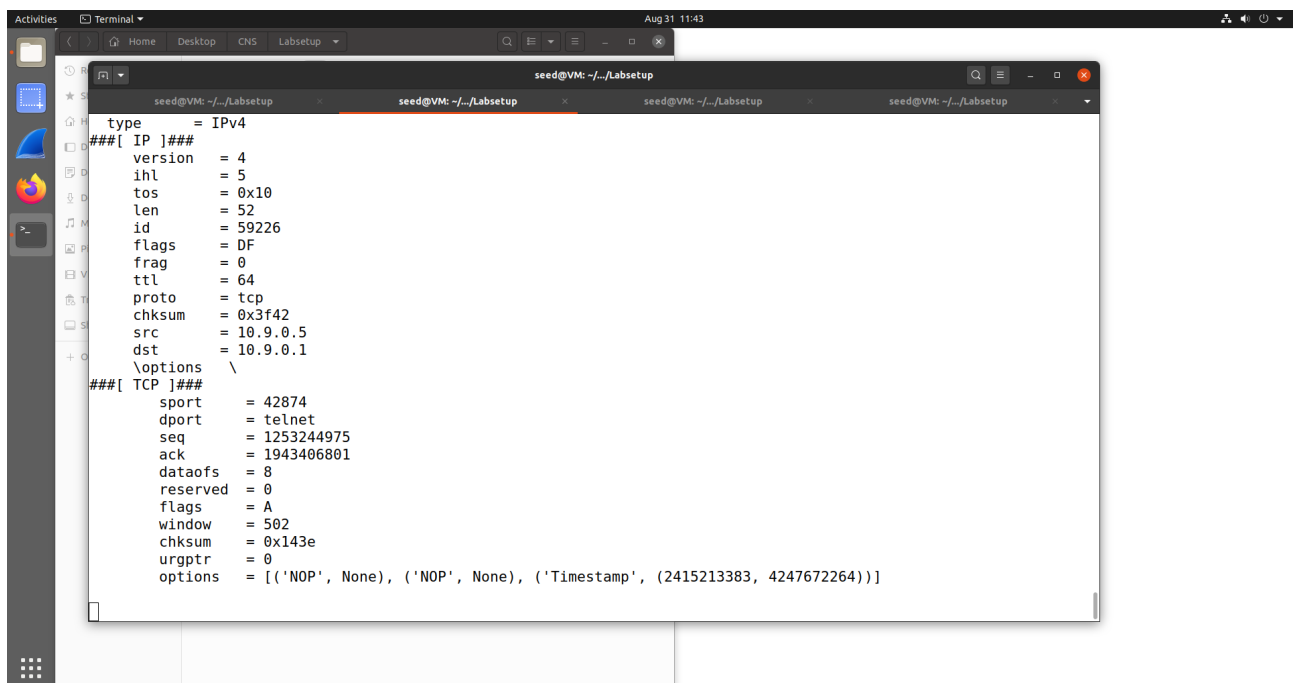
Filter 2) TCP, destination port number 23.

```
pkt = sniff (iface = "br-ea9efb963438",filter='tcp and src host 10.9.0.5 and dst port 23', prn=print_pkt)
```

Under filter, our first filter we are applying is “tcp”. It is followed by an and clause, meaning it has to be tcp AND whatever filter follows. The next filter is src host .... This means the packet must have a source IP of whatever we desire. The final filter being used is destination port number, in this case it is 23.

First we run a telnet command from host A. The telnet command uses tcp protocol and runs on port 23, hence our filter should be able to sniff these packets generated by the telnet command

## Output

A screenshot of a terminal window titled "Terminal" with a date and time of "Aug 31 11:43". The terminal shows the output of a packet sniffing operation. The output is structured as follows: It starts with "type = IPv4", followed by a section "###[ IP ]###" containing fields: version = 4, ihl = 5, tos = 0x10, len = 52, id = 59226, flags = DF, frag = 0, ttl = 64, proto = tcp, chksum = 0x3f42, src = 10.9.0.5, and dst = 10.9.0.1. This is followed by a section "\options\" and then "###[ TCP ]###" containing: sport = 42874, dport = telnet, seq = 1253244975, ack = 1943406801, dataofs = 8, reserved = 0, flags = A, window = 502, chksum = 0x143e, urgptr = 0, and options = [('NOP', None), ('NOP', None), ('Timestamp', (2415213383, 4247672264))].

```
seed@VM: ~/.../Labsetup
type = IPv4
###[ IP ]###
version = 4
ihl = 5
tos = 0x10
len = 52
id = 59226
flags = DF
frag = 0
ttl = 64
proto = tcp
chksum = 0x3f42
src = 10.9.0.5
dst = 10.9.0.1
\options\
###[ TCP ]###
sport = 42874
dport = telnet
seq = 1253244975
ack = 1943406801
dataofs = 8
reserved = 0
flags = A
window = 502
chksum = 0x143e
urgptr = 0
options = [('NOP', None), ('NOP', None), ('Timestamp', (2415213383, 4247672264))]
```

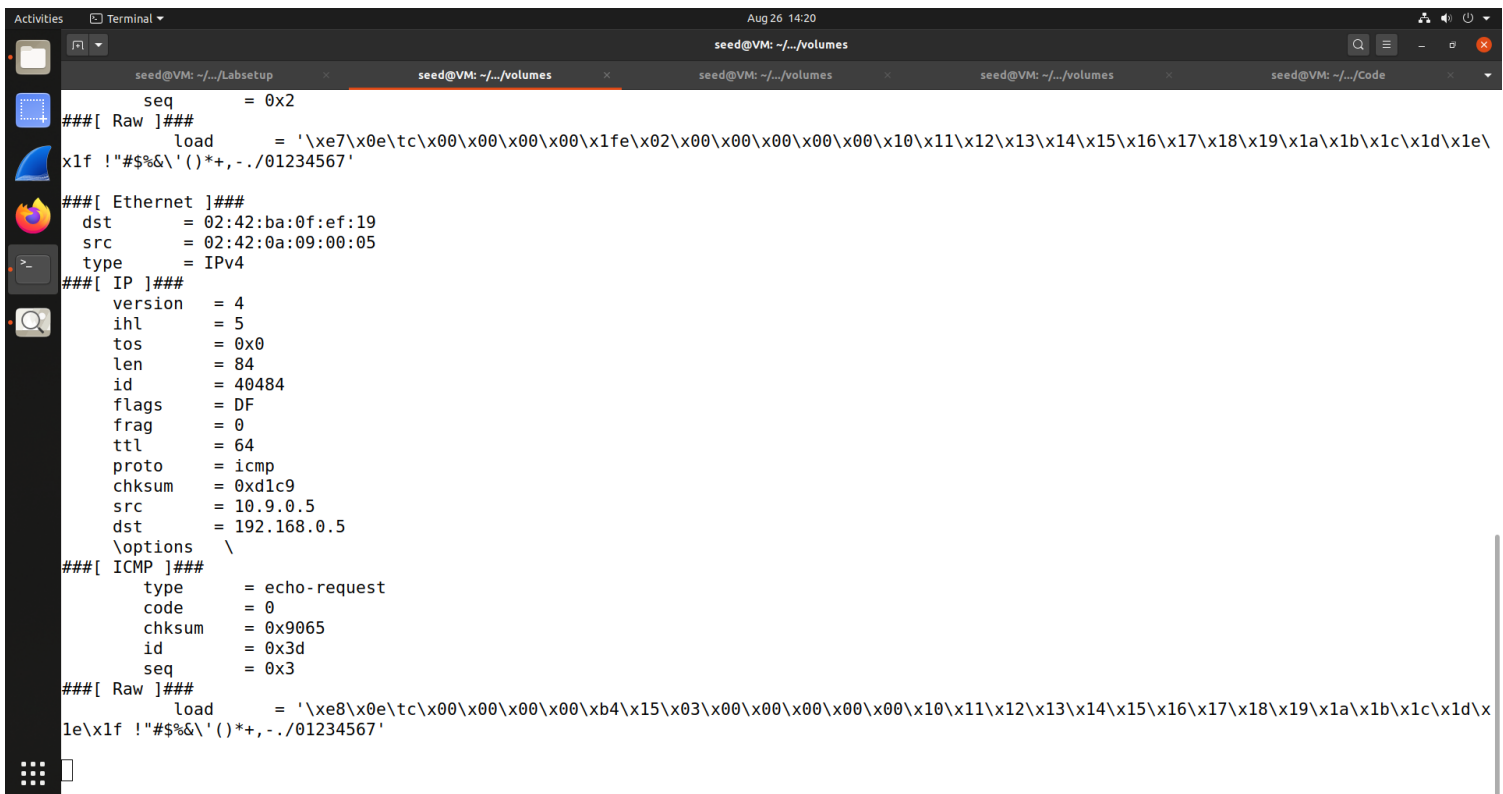
Filter 3) Packets from/to a particular subnet.

```
pkt = sniff(iface = "br-ea9efb963438",filter='src net 10.9.0.0/24', prn=print_pkt)
```

Here we are using a subnet filter. The ip format follows cidr Notation. The /24 means the first 24 bits are for host IP's, and the last 8 bits are for the subnet. Hence the subnet mask here is going to be 255.255.255.0. This filter will sniff all packets originating from this subnet.

We will ping an arbitrary IP in the subnet "10.9.0.0/24" from Host A to observe the working of this filter.

## Output



```
Aug 26 14:20
seed@VM: ~/.../volumes
seq      = 0x2
####[ Raw ]###
load     = '\xe7\xe\tc\x00\x00\x00\x00\x1fe\x02\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#&%\`()*+,-./01234567'
####[ Ethernet ]###
dst      = 02:42:ba:0f:ef:19
src      = 02:42:0a:09:00:05
type     = IPv4
####[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 40484
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xd1c9
src      = 10.9.0.5
dst      = 192.168.0.5
\options \
####[ ICMP ]###
type     = echo-request
code     = 0
chksum   = 0x9065
id       = 0x3d
seq      = 0x3
####[ Raw ]###
load     = '\xe8\xe\tc\x00\x00\x00\x00\xb4\x15\x03\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#&%\`()*+,-./01234567'
```

## Task 1.2A: Spoofing Packets

Spoofing is the process of pretending/assuming someone else's ID to trick the victim into accepting communication with the attacker. The attacker's actual ID is hidden.

```
IPLayer = IP()  
IPLayer.src="10.9.0.1"  
IPLayer.dst="10.9.0.5"  
ICMPpkt = ICMP()  
pkt = IPLayer/ICMPpkt  
pkt.show()  
send(pkt,verbose=0)
```

Here, we are creating an IPLayer using scapy's IP() function, and we are setting the source to "10.9.0.1". This IP address can be set to any arbitrary IP and hence we can pretend to be some other machine and hide our own identity.

## Output

The screenshot displays a terminal window on the right and a Wireshark packet capture window on the left. The terminal shows the execution of a Python script that sends a spoofed ICMP packet from 10.9.0.1 to 10.9.0.5. The Wireshark window shows the captured packets, with the selected packet being an ICMP Echo (ping) reply from 10.9.0.5 to 10.9.0.1.

**Wireshark Packet Capture:**

No.	Time	Source	Destination	Protocol	Length	Info
6	2022-08-26 14:2...	10.9.0.1	10.9.0.5	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response ...)
7	2022-08-26 14:2...	10.9.0.1	10.9.0.5	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 8)
8	2022-08-26 14:2...	10.9.0.5	10.9.0.1	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 7)
9	2022-08-26 14:2...	10.9.0.5	10.9.0.1	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=64

**Terminal Output:**

```
dst = 10.9.0.5  
\options \  
###[ ICMP ]###  
type = echo-request  
code = 0  
chksum = None  
id = 0x0  
seq = 0x0  
  
root@VM:/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  
  
root@VM:/volumes/Code#
```

Here we can see that, there is a reply from "10.9.0.5" (the victim), and that the victim has fallen for the spoof.

## Task 1.2A: Spoofing Packets

```
IPLayer = IP()
IPLayer.src="10.9.0.11"
IPLayer.dst="10.9.0.99"
ICMPpkt = ICMP()
pkt = IPLayer/ICMPpkt
pkt.show()
send(pkt,verbose=0)
```

Here we are changing the src and dst to an arbitrary address.

## Output

The screenshot shows a terminal window with a Wireshark capture on the left and a Python script output on the right. The Wireshark capture shows six packets: three ARP requests and three ICMP Echo (ping) requests. The Python script output shows the details of the first ICMP packet, which is a spoofed ping packet from 10.9.0.11 to 10.9.0.99.

No.	Time	Source	Destination	Protocol	Length	Info
1	2022-08-26 14:2...	02:42:ba:0f:ef:19		ARP	44	Who has 10.9.0.99? Tell 10.9.0.1
2	2022-08-26 14:2...	02:42:ba:0f:ef:19		ARP	44	Who has 10.9.0.99? Tell 10.9.0.1
3	2022-08-26 14:2...	02:42:ba:0f:ef:19		ARP	44	Who has 10.9.0.99? Tell 10.9.0.1
4	2022-08-26 14:2...	10.9.0.11	10.9.0.99	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response ...
5	2022-08-26 14:2...	10.9.0.11	10.9.0.99	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response ...
6	2022-08-26 14:2...	10.9.0.11	10.9.0.99	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response ...

```
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

root@VM:/volumes/Code#
```

We can see that an ARP packet is sent out, since the location of “10.9.0.99” is unknown.

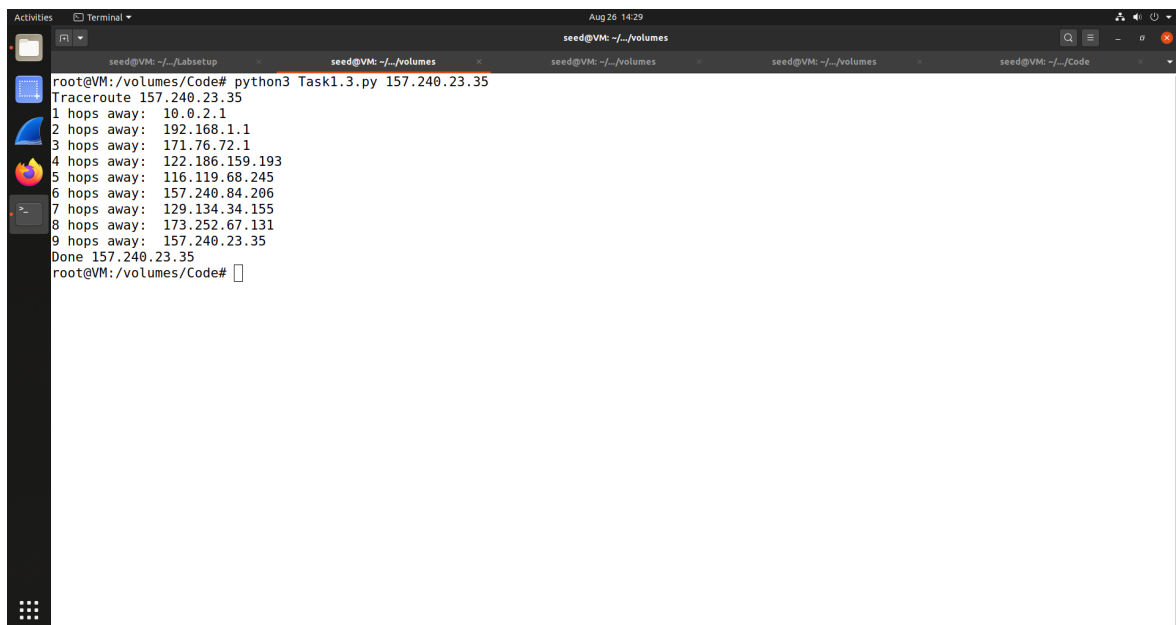
## Task 1.3: Traceroute

The `traceroute` command provides a map of how data flows from your machine to destination in the network. It gives all the hops that your packet takes to reach the destination.

We can recreate the `traceroute` command in python using `scapy`.

Let us see the route a packet takes from machine to facebook's servers.

## Output



```
root@VM:/volumes/Code# python3 Task1.3.py 157.240.23.35
Traceroute 157.240.23.35
1 hops away: 10.0.2.1
2 hops away: 192.168.1.1
3 hops away: 171.76.72.1
4 hops away: 122.186.159.193
5 hops away: 116.119.68.245
6 hops away: 157.240.84.206
7 hops away: 129.134.34.155
8 hops away: 173.252.67.131
9 hops away: 157.240.23.35
Done 157.240.23.35
root@VM:/volumes/Code#
```

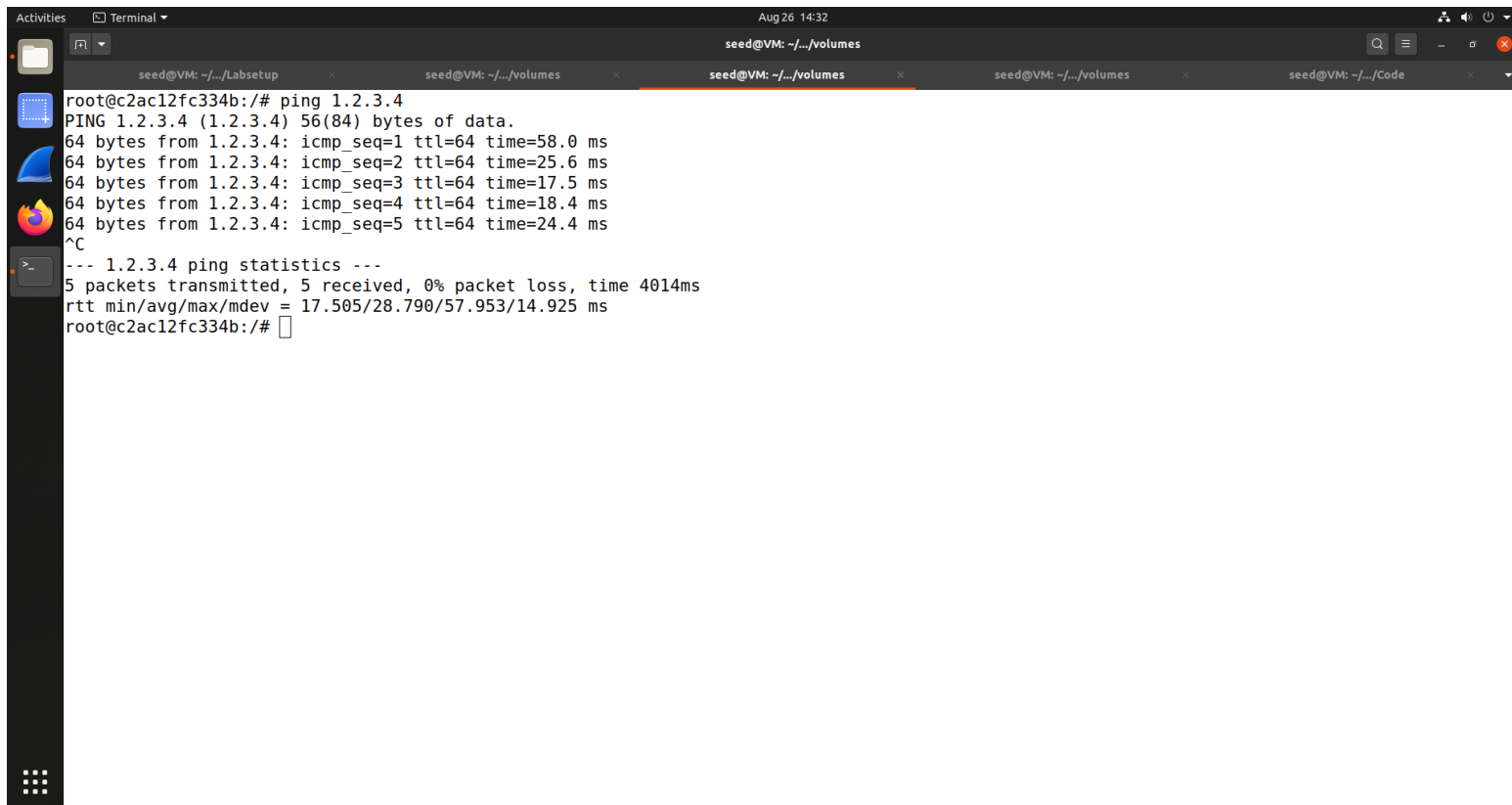
An interesting observation we can make here is that, this is not a set path. Running the command at a later time gives different results. This is because of changes in network, routers going offline/online, packets taking a shorter path, etc.

```
root@VM:/volumes/Code# python3 Task1.3.py 157.240.23.35
Traceroute 157.240.23.35
1 hops away: 10.0.2.1
2 hops away: 192.168.1.1
3 hops away: 171.76.72.1
4 hops away: 125.21.0.185
5 hops away: 116.119.57.97
6 hops away: 182.79.153.40
7 hops away: 116.119.106.152
8 hops away: 157.240.84.206
9 hops away: 129.134.34.151
10 hops away: 173.252.67.25
11 hops away: 157.240.23.35
Done 157.240.23.35
root@VM:/volumes/Code#
```



[illegible]

## Output (Victim)



The screenshot shows a terminal window titled "Terminal" with a date and time of "Aug 26 14:32". The terminal is running a command prompt on a machine with IP "c2ac12fc334b". The user has entered the command "ping 1.2.3.4". The output shows five successful ping responses from 1.2.3.4, each with a different ICMP sequence number and a time value. The statistics at the bottom show 5 packets transmitted, 5 received, 0% packet loss, and a total time of 4014ms. The terminal window has several tabs open, all titled "seed@VM: ~/.../volumes".

```
root@c2ac12fc334b:/# 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=58.0 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=25.6 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=17.5 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=18.4 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=24.4 ms
^C
--- 1.2.3.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4014ms
rtt min/avg/max/mdev = 17.505/28.790/57.953/14.925 ms
root@c2ac12fc334b:/#
```

We can see that on the victim's machine, there is a response to our ping from "1.2.3.4" even though the machine doesn't exist. This may lead to the victim believe that they are talking to "1.2.3.4", but they're actually talking to the attacker.