Lab 1

Student ID: 1005288

1.1

Part A

```
[09/26/23]seed@VM:~/.../Labsetup$ dockps
e9c439522d9b  hostB-10.9.0.6
0d31473c8a87  hostA-10.9.0.5
5239d41d3812  seed-attacker
[09/26/23]seed@VM:~/.../Labsetup$
```

```
root@VM:/# ifconfig
br-160cf3e8d0c3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 15
00
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:c7ff:feac:418b  prefixlen 64  scopeid 0x20<l
ink>
```

```
root@e9c439522d9b:/# ping 10.9.0.5 -c 5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.062 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.059 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.065 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.059 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=64 time=0.083 ms

--- 10.9.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4085ms
rtt min/avg/max/mdev = 0.059/0.065/0.083/0.009 ms
```

dockps to find the container IDs and *docksh* to enter the environment.
For this part, I used B to ping A and sniffed the interface br-160cf3e8d0c3 which contains both A and B.
Code was implemented to take in the interface as input as I noticed the interface id changes each time the containers were teared-down and set-up again.

```
###[ Raw ]###
          load      = 'l\x94\x12e\x00\x00\x00\x005\xe2\x0e\x00\x00\x00\x00\x00\x10\x11\x
12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 60105
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x7bc3
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x7e2c
        id        = 0x20
        seq       = 0x5
```

As shown, I used B(10.9.0.6) to ping A (10.9.0.5), attached is the reply from A to B, and it shows that B successfully pinged A and A replied.

```
[09/26/23]seed@VM:~/.../Lab$ python3 sniffer.py
interface id:br-160cf3e8d0c3
Traceback (most recent call last):
  File "sniffer.py", line 7, in <module>
    pkt = sniff(iface=str(iface),
  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_sockets[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
```

This error was shown after switching to seed and attempting to run the sniffer. This is expected as we require root permissions to sniff packets.

Part B

The code for sniffing ICMP requests is the same as the one in A, hence no modifications were done.

Sniffing tcp connection with specific port number, destination was done by editing filter, filter='tcp and port 23 and src host 10.9.0.6'. The sniffer will sniff tcp connections on port 23, with host B (10.9.0.6)

I used netcat to make A(10.9.0.5) a server listening to tcp requests, and connected B with A on A's IP and port 23 as a client.
Finally, the sniffer was able to pick up the message sent from B to A as shown below.

.

```
root@a2977f447617:/# nc -v 10.9.0.5 23
Connection to 10.9.0.5 23 port [tcp/telnet] succeeded!
???? ??#??'
Sniffing on Port 23
```

B's message to A

```
     len       = 72
     id        = 38911
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x8e94
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
        sport     = 52752
        dport     = telnet
        seq       = 1551095924
        ack       = 590424924
        dataofs   = 8
        reserved  = 0
        flags     = PA
        window    = 502
        chksum    = 0x1457
        urgptr    = 0
        options   = [('NOP', None), ('NOP', None), ('Timestamp', (189025924, 16
74584087))]
###[ Raw ]###
           load      = 'Sniffing on Port 23\n'
```

To simulate a computer on another subnet, I added another host in docker-compose.yml to create another instance of a VM in another subnet with IP 128.230.0.17 as host C. This was done by changing the dockerfile to include a new subnet and a new container for host C.
I used the sniffer to sniff C's traffic as it pings google.com and it was able to capture the packets, which was expected since the attacker and victim had to be in the same subnet / interface. If I were to use an interface of the attacker that did not include C, the sniffer won't sniff any packets.

```
root@VM:/# ifconfig
br-cc70e3a2d9d0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:b9ff:fe3e:1900  prefixlen 64  scopeid 0x20<link>
        ether 02:42:b9:3e:19:00  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 38  bytes 4809 (4.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

br-dd490ffcc7e2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 128.230.0.1  netmask 255.255.0.0  broadcast 128.230.255.255
        inet6 fe80::42:adff:fed7:b77f  prefixlen 64  scopeid 0x20<link>
        ether 02:42:ad:d7:b7:7f  txqueuelen 0  (Ethernet)
        RX packets 10  bytes 661 (661.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 48  bytes 5852 (5.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
[09/26/23]seed@VM:~/.../A$ dockps
47ec4173acc9  hostC-128.230.0.17
98cdd0e004f8  hostA-10.9.0.5
3ce4f57fb190  seed-attacker
a2977f447617  hostB-10.9.0.6
[09/26/23]seed@VM:~/.../A$ docksh 47
root@47ec4173acc9:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 128.230.0.17  netmask 255.255.0.0  broadcast 128.230.255.255
        ether 02:42:80:e6:00:11  txqueuelen 0  (Ethernet)
        RX packets 58  bytes 7242 (7.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@47ec4173acc9:/# ping google.com
PING google.com (142.251.175.101) 56(84) bytes of data.
64 bytes from sh-in-f101.1e100.net (142.251.175.101): icmp_seq=1 ttl=101 time=7
.13 ms
64 bytes from sh-in-f101.1e100.net (142.251.175.101): icmp_seq=2 ttl=101 time=6
.69 ms
64 bytes from sh-in-f101.1e100.net (142.251.175.101): icmp_seq=3 ttl=101 time=6
.80 ms
64 bytes from sh-in-f101.1e100.net (142.251.175.101): icmp_seq=4 ttl=101 time=7
.25 ms
64 bytes from sh-in-f101.1e100.net (142.251.175.101): icmp_seq=5 ttl=101 time=1
1.2 ms
```

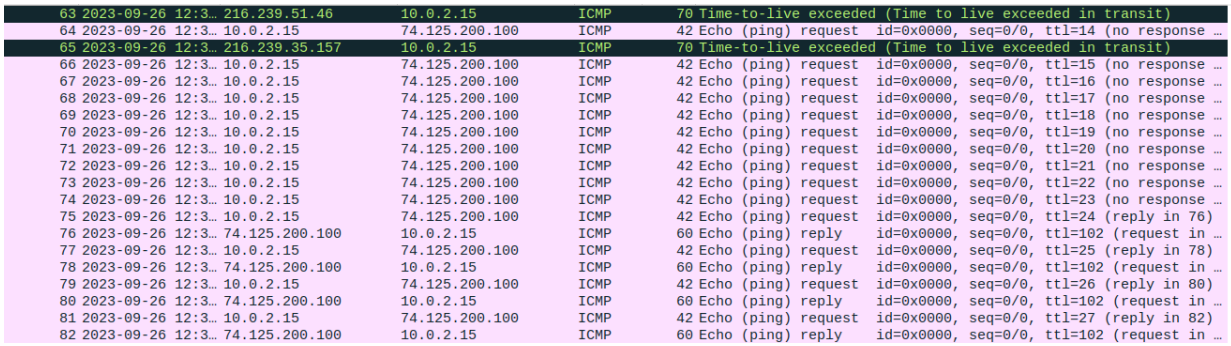
```
###[ Ethernet ]###
  dst       = 02:42:39:12:43:f2
  src       = 02:42:80:e6:00:11
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 64925
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x7db3
     src       = 128.230.0.17
     dst       = 142.251.175.101
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x5036
        id        = 0x1e
        seq       = 0x5
###[ Raw ]###
           load      = '\x92\xd2\x12e\x00\x00\x00\x006\x9c\r\x00\x00\x00\x00\x0
0\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,
-./01234567'
```

1.2

	Time	Source	Destination	Protocol	Length	Info
1	2023-09-26 11:0…	02:42:56:99:b2:fa	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
2	2023-09-26 11:0…	02:42:0a:09:00:05	02:42:56:99:b2:fa	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
3	2023-09-26 11:0…	1.2.3.4	10.9.0.5	ICMP	42	Echo (ping) request id=0x0000, seq=0/0,
4	2023-09-26 11:0…	10.9.0.5	1.2.3.4	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0,
5	2023-09-26 11:0…	02:42:0a:09:00:05	02:42:56:99:b2:fa	ARP	42	Who has 10.9.0.1? Tell 10.9.0.5
6	2023-09-26 11:0…	02:42:56:99:b2:fa	02:42:0a:09:00:05	ARP	42	10.9.0.1 is at 02:42:56:99:b2:fa

The original path of the ping was from 10.9.0.1 to 10.9.0.6, but the code spoofed it and the path was changed. The new path is from 1.2.3.4 (non-existent IP) to 10.9.0.5 (another VM on the same network). Wireshark was used to capture the change in paths and a ping request-reply was captured.

1.3

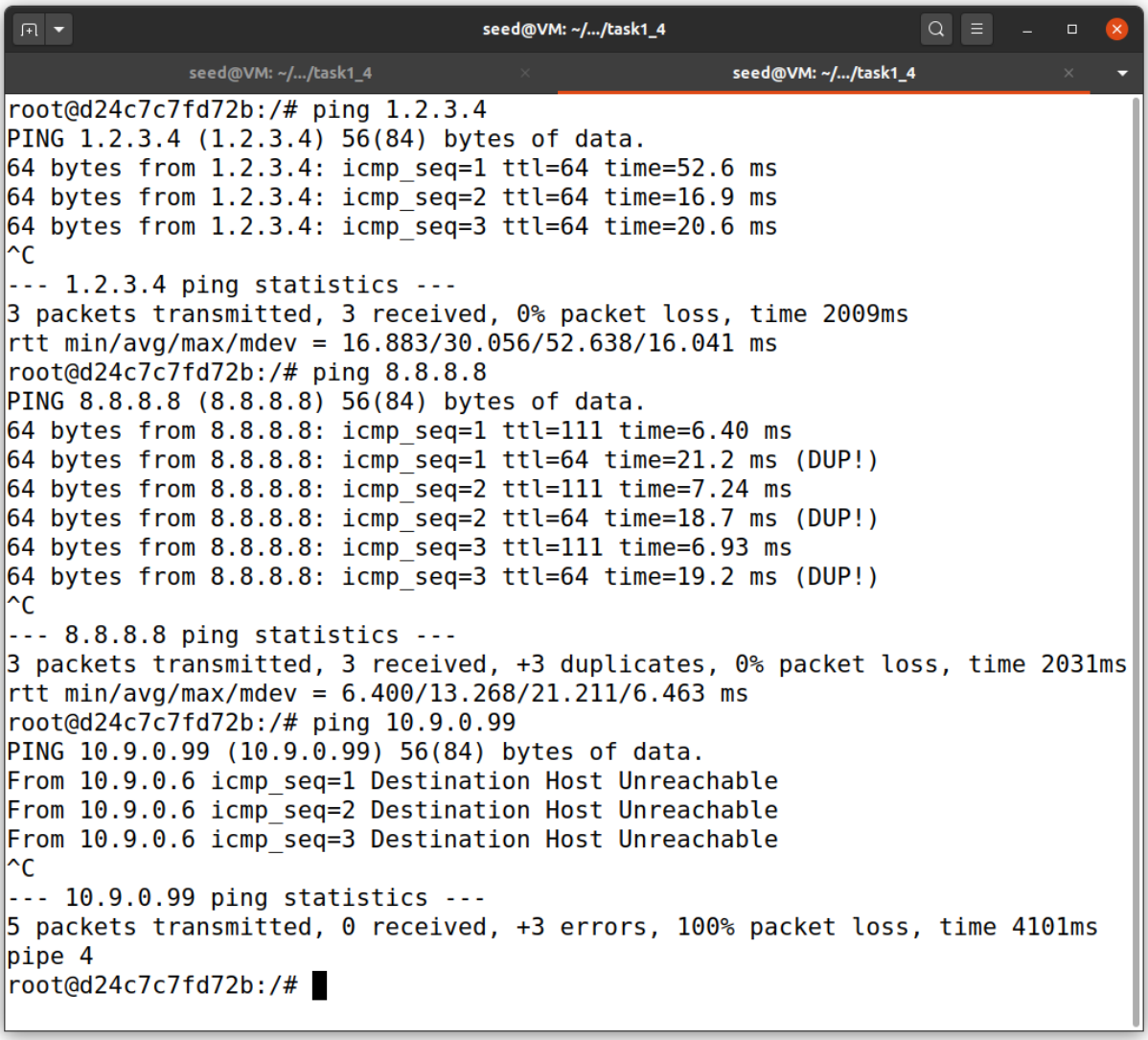


63	2023-09-26 12:3…	216.239.51.46	10.0.2.15	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
64	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=14 (no response …
65	2023-09-26 12:3…	216.239.35.157	10.0.2.15	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
66	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=15 (no response …
67	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=16 (no response …
68	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=17 (no response …
69	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=18 (no response …
70	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=19 (no response …
71	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=20 (no response …
72	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=21 (no response …
73	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=22 (no response …
74	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=23 (no response …
75	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=24 (reply in 76)
76	2023-09-26 12:3…	74.125.200.100	10.0.2.15	ICMP	60 Echo (ping) reply id=0x0000, seq=0/0, ttl=102 (request in …
77	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=25 (reply in 78)
78	2023-09-26 12:3…	74.125.200.100	10.0.2.15	ICMP	60 Echo (ping) reply id=0x0000, seq=0/0, ttl=102 (request in …
79	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=26 (reply in 80)
80	2023-09-26 12:3…	74.125.200.100	10.0.2.15	ICMP	60 Echo (ping) reply id=0x0000, seq=0/0, ttl=102 (request in …
81	2023-09-26 12:3…	10.0.2.15	74.125.200.100	ICMP	42 Echo (ping) request id=0x0000, seq=0/0, ttl=27 (reply in 82)
82	2023-09-26 12:3…	74.125.200.100	10.0.2.15	ICMP	60 Echo (ping) reply id=0x0000, seq=0/0, ttl=102 (request in …

I used a for loop to increment TTL and send the packets to destination (Google's IP address), I used wireshark to check for ICMP error messages until the TTL is not exceeded. The answer is TTL = 15.

1.4

```
root@VM:/# python3 sniff_and_spoof.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```



An inspection of the original sniffed packets shows that icmp.id, icmp.seq and load does not change in the reply and request. As such, we need to

1. Swap dst and src in the IP stack
2. Keep the original load for stack
3. Send to the machine that requested

The results are shown above, the machine got a reply from 1.2.3.4, received duplicates with alerts from 8.8.8.8 and the host is unreachable from 10.9.0.99.

References

https://dorazaria.github.io/network/packet-sniffing-and-spoofing-lab/
https://phoenixnap.com/kb/nc-command#ftoc-heading-2