

Lab3 - SEED Labs – Packet Sniffing and Spoofing Lab

- Lohith Kumar Bhamore

Setup – Using Containers

- Used the SEED-Ubuntu 20.04 (64-bit) image provided on the website in VM on a windows 10 machine.
- Followed the instructions as demonstrated in the video shared by the TA
- Next, I used the docker-compose.yml file to set up the environment using the alias container commands as mentioned below –
 - o \$dcbuild: Alias for docker-compose build
 - o \$dcup: Alias for docker-compose up
 - o \$dcdown: Alias for docker-compose down
 - o \$dockps: Alias for docker ps –format “{{.ID}} {{.Names}}”
 - o \$docksh <id>: Alias for docker exec -it <id> /bin/bash



```
Activities Terminal Nov 26 16:44
seed@VM: ~
seed@VM: ~/Labsetup
[11/26/23]seed@VM:~$ dockps
9c833b2da571 seed-attacker
b164b6cf0ea8 hostB-10.9.0.6
a7c843dd9f0e hostA-10.9.0.5
[11/26/23]seed@VM:~$ docksh b1
root@b164b6cf0ea8:/#
root@b164b6cf0ea8:/# exit
exit
[11/26/23]seed@VM:~$
```

- The following is the setup of the containers –

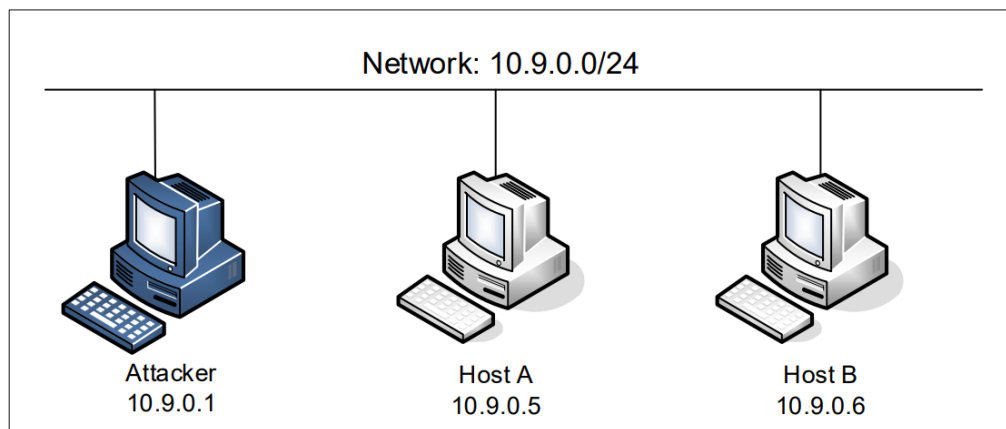


Figure 1: Lab environment setup

- We run these using docker containers using the commands mentioned above

```

seed@VM: ~/.../Labsetup
hostB uses an image, skipping
[11/26/23]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Pulling attacker (handsonsecurity/seed-ubuntu:large)...
large: Pulling from handsonsecurity/seed-ubuntu
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
b5e99359ad22: Pull complete
3d2251ac1552: Pull complete
1059cf087055: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab02008f016a7936d9cadf8e8238146d07c1c12b39cd63c3e73a0297c07a
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:large
Creating hostA-10.9.0.5 ... done
Creating seed-attacker ... done
Creating hostB-10.9.0.6 ... done
Attaching to seed-attacker, hostB-10.9.0.6, hostA-10.9.0.5
hostA-10.9.0.5 | * Starting internet superserver inetd      [ OK ]
hostB-10.9.0.6 | * Starting internet superserver inetd      [ OK ]

```

- Points to note about the attacker container –
 - o **Shared Folder** – Cide editing inside a VM is easier than in containers as it is possible to use the editors we want. To allow the VM and the containers to share files, we must create a shared folder b/w them using Docker volumes. Using the docker compose file there is an entry which mounts ./volumes folder on the VM to the /volumes folder inside the container. Our code is written in ./volumes folder on the VM
 - o **Host mode** – To run sniffer programs in a container, which because of being attached to a virtual switch can only check its own traffic and not of other containers, we make the attacker container a host. This allows this container to see all traffic using the entry – network_mode: host
- To get the network interface name – We need to find the name of our network interface on VM to use it in the programs using the ifconfig command as shown below –

```

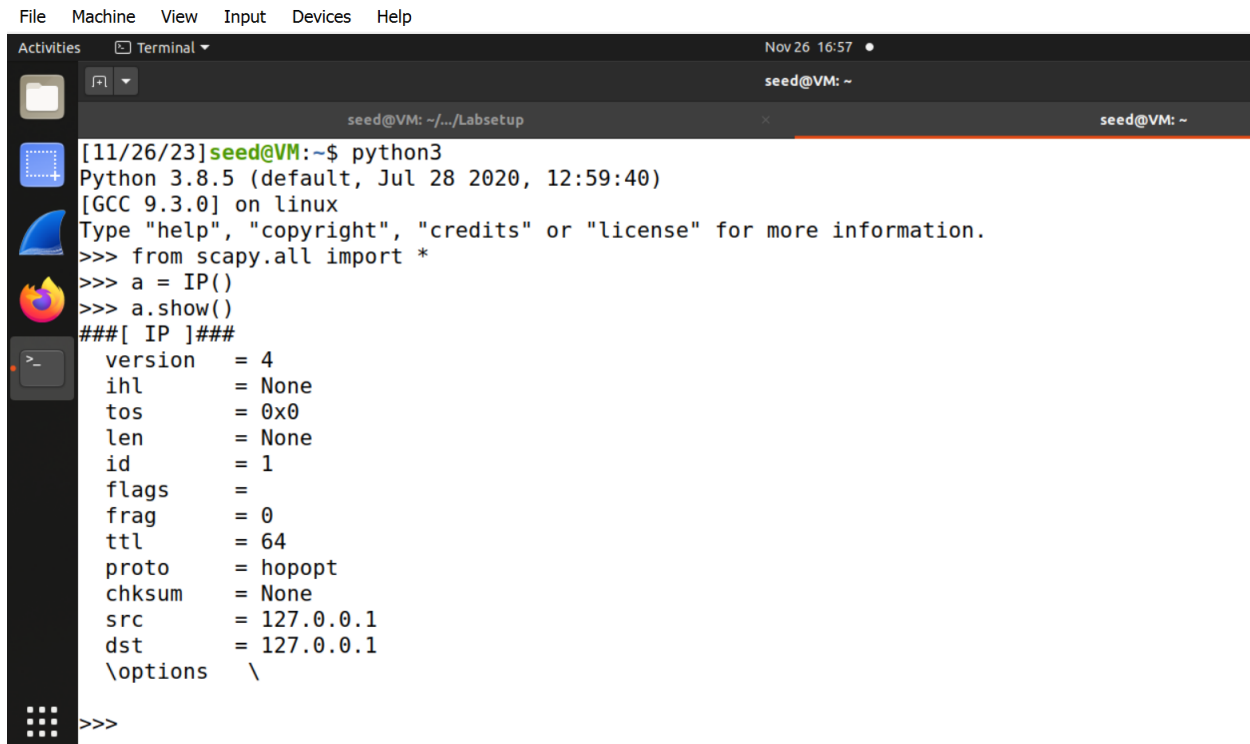
[11/26/23]seed@VM:~$ ifconfig
br-2af6eaa1bbb1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
    inet6 fe80::42:72ff:fe2c:ef18 prefixlen 64 scopeid 0x20<link>
    ether 02:42:72:2c:ef:18 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 47 bytes 7147 (7.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Task 1 – Using Scapy to Sniff and Spoof Packets

Scapy is used as it can be used not only as a tool but also as a building block to create other sniffing and spoofing tools.

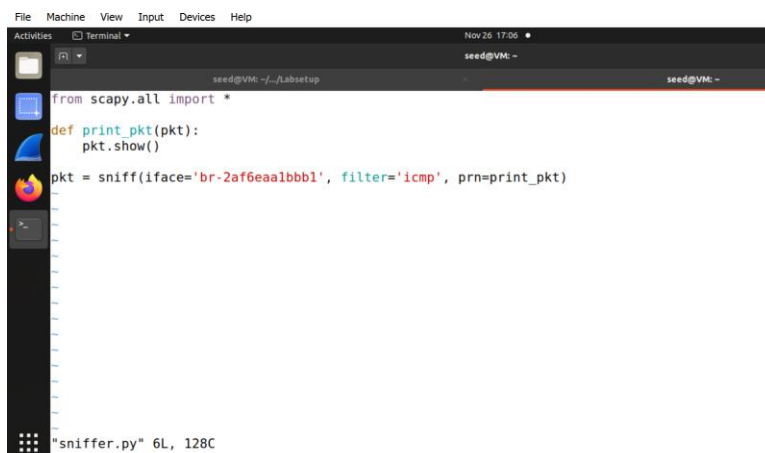
We use scapy using a python program running on root privilege as it is required to spoof packets



```
File Machine View Input Devices Help
Activities Terminal Nov 26 16:57
seed@VM: ~
seed@VM: ~/Labsetup
[11/26/23]seed@VM:~$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> a = IP()
>>> a.show()
####[ IP ]####
version      = 4
ihl          = None
tos          = 0x0
len          = None
id           = 1
flags        =
frag         = 0
ttl          = 64
proto        = hopopt
chksum       = None
src          = 127.0.0.1
dst          = 127.0.0.1
\options     \
>>>
```

Task 1.1 – Sniffing Packets

Though Wireshark is a very popular sniffing tool, it can't be used as building block to construct other tools, thus we use scapy. This task is about learning how to use Scapy to do packet sniffing in python program sniffer.py in volumes folder in VM, which is also accessible in container –



```
File Machine View Input Devices Help
Activities Terminal Nov 26 17:06
seed@VM: ~
seed@VM: ~/Labsetup
from scapy.all import *
def print_pkt(pkt):
    pkt.show()
pkt = sniff(iface='br-2af6eaa1bbb1', filter='icmp', prn=print_pkt)

"sniffer.py" 6L, 128C
```

Task 1.1A

As can be seen in the code above, we call the function `print_pkt()` for each packet sniffed, which will then print out some information about the packet.

Let's run the program with **root** privileges in attacker container –

```
a7c843dd9f0e hostA-10.9.0.5
[11/26/23]seed@VM:~/../volumes$ docksh 9c
root@VM:/# cd volumes/
root@VM:/volumes# ls
sniffer.py
root@VM:/volumes# chmod a+x sniffer.py
root@VM:/volumes# sniffer.py
./sniffer.py: line 1: from: command not found
./sniffer.py: line 3: syntax error near unexpected token `('
./sniffer.py: line 3: `def print_pkt(pkt):'
root@VM:/volumes# python3 sniffer.py
```

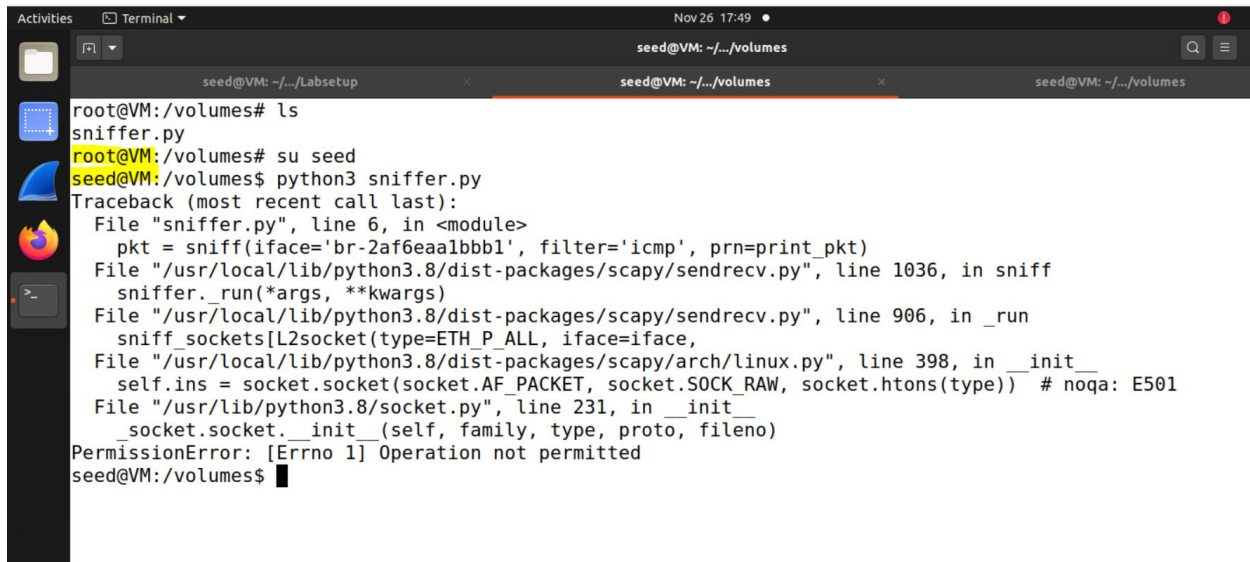
Let's ping google.com from hostA –

```
[11/26/23]seed@VM:~/../volumes$ dockkps
9c833b2da571 seed-attacker
b164b6cf0ea8 hostB-10.9.0.6
a7c843dd9f0e hostA-10.9.0.5
[11/26/23]seed@VM:~/../volumes$ docksh a7
root@a7c843dd9f0e:/# ping google.com
PING google.com (142.250.68.78) 56(84) bytes of data.
64 bytes from lax31s11-in-f14.1e100.net (142.250.68.78): icmp_seq=1 ttl=54 time=15.9 ms
64 bytes from lax31s11-in-f14.1e100.net (142.250.68.78): icmp_seq=2 ttl=54 time=15.7 ms
64 bytes from lax31s11-in-f14.1e100.net (142.250.68.78): icmp_seq=3 ttl=54 time=16.4 ms
64 bytes from lax31s11-in-f14.1e100.net (142.250.68.78): icmp_seq=4 ttl=54 time=58.7 ms
^C
--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 15.701/26.663/58.707/18.502 ms
root@a7c843dd9f0e:/#
```

And we can see the packet information in the attacker container and since we used the ICMP filter in our python file, we see ICMP information of the packet. We are now able to sniff every packet being sent from hosts A and B. We can also see that echo-request is of type ICMP and the echo-reply type when we receive a reply from the ping –

```
./sniffer.py: line 3: `def print_pkt(pkt):'
root@VM:/volumes# python3 sniffer.py
###[ Ethernet ]###
  dst      = 02:42:72:2c:ef:18
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 47059
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  checksum = 0xa57f
  src      = 10.9.0.5
  dst      = 142.250.68.78
  \options
###[ ICMP ]###
  type     = echo-request
  code     = 0
  checksum = 0xa749
```

When we try to run the sniffer.py file without root privileges (in seed account as follows) –



```
root@VM:/volumes# ls
sniffer.py
root@VM:/volumes# su seed
seed@VM:/volumes$ python3 sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 6, in <module>
    pkt = sniff(iface='br-2af6eaalbbb1', filter='icmp', 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_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
seed@VM:/volumes$
```

As expected, we get a permission error as sniffing packets can only be done when you have root privileges.

Task 1.1B

Using different filters

- Capture only ICMP packet –
Ping google.com from Host A



```
root@a7c843dd9f0e:/# ping google.com
PING google.com (142.250.189.14) 56(84) bytes of data.
64 bytes from lax31s16-in-f14.1e100.net (142.250.189.14): icmp_seq=1 ttl=54 time=19.3 ms
64 bytes from lax31s16-in-f14.1e100.net (142.250.189.14): icmp_seq=2 ttl=54 time=18.2 ms
64 bytes from lax31s16-in-f14.1e100.net (142.250.189.14): icmp_seq=3 ttl=54 time=16.4 ms
64 bytes from lax31s16-in-f14.1e100.net (142.250.189.14): icmp_seq=4 ttl=54 time=25.7 ms
64 bytes from lax31s16-in-f14.1e100.net (142.250.189.14): icmp_seq=5 ttl=54 time=15.5 ms
```

Run sniffer.py with ICMP filter in attacker container and capture packet information –


```

File Machine View Input Devices Help
Nov 26 17:58
seed@VM: ~/../volumes
seed@VM: ~/../volumes
seed@VM: ~/../volumes
###[ IP ]###
version    = 4
ihl        = 5
tos        = 0x0
len        = 84
id         = 14415
flags      = DF
frag       = 0
ttl        = 64
proto      = icmp
chksum     = 0x2064
src        = 10.9.0.5
dst        = 142.250.72.238
\options   \
###[ ICMP ]###
type       = echo-request
code       = 0
chksum     = 0x9746
id         = 0x25
seq        = 0x6
###[ Raw ]###
load       = 'v\xcdce\x00\x00\x00\x00\xc2\x88\x05\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'

```

- Capture any TCP packet coming from a particular IP and with destination port 23 – For this we need to modify sniffer.py to change the filter to TCP with port 23 as shown below

```

File Machine View Input Devices Help
Nov 26 18:07
seed@VM: ~/../volumes
seed@VM: ~/../volumes
seed@VM: ~/../volumes
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-2af6eaa1bbb1', filter='tcp and src host 10.9.0.5 and dst port 23', prn=print_pkt)

```


Run a telnet command from 10.9.0.5 (Host A) to 10.9.0.6 (Host B)

```

File Machine View Input Devices Help
Nov 26 18:08
seed@VM: ~/../volumes
seed@VM: ~/../volumes
seed@VM: ~/../volumes
root@a7c843dd9f0e:~# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b164b6cf0ea8 login:

```

Run this new sniffer.py in attacker container and observe the TCP packets –



```
id = 34535
flags = DF
frag = 0
ttl = 64
proto = tcp
chksum = 0x9f8e
src = 10.9.0.5
dst = 10.9.0.6
\options \
###[ TCP ]###
  sport = 40832
  dport = telnet
  seq = 2009972561
  ack = 3436721764
  dataofs = 8
  reserved = 0
  flags = PA
  window = 502
  chksum = 0x1465
  urgptr = 0
  options = [('NOP', None), ('NOP', None), ('Timestamp', (1432412441, 4196691951))]
###[ Raw ]###
  load = "\xff\xfa \x0038400,38400\xff\xfa\xff\xfa\x00\xff\xfa\xff\xfa\x18\x00xterm\xff\xfa"
```

- Capture packets from or to a particular subnet

Modify the sniffer.py to capture packets from the subnet 10.0.2.0/24 as shown below –

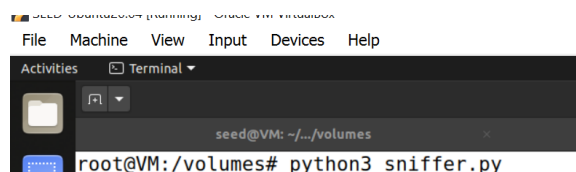


```
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-2af6eaa1bbb1', filter='net 10.0.2.0/24', prn=print_pkt)
```

Now run the sniffer.py in attacker container with root privileges –



```
root@VM:/volumes# python3 sniffer.py
```

Next, let's ping 10.0.2.11 on the subnet from Host A –

```
File Machine View Input Devices Help
Activities Terminal Nov 26 18:20
seed@VM: ~/.../volumes
seed@VM: ~/.../volumes
seed@VM: ~/.../volumes
root@a7c843dd9f0e:/# ping 10.0.2.11
PING 10.0.2.11 (10.0.2.11) 56(84) bytes of data.
From 10.0.2.15 icmp_seq=1 Destination Host Unreachable
From 10.0.2.15 icmp_seq=2 Destination Host Unreachable
From 10.0.2.15 icmp_seq=3 Destination Host Unreachable
From 10.0.2.15 icmp_seq=4 Destination Host Unreachable
From 10.0.2.15 icmp_seq=5 Destination Host Unreachable
From 10.0.2.15 icmp_seq=6 Destination Host Unreachable
From 10.0.2.15 icmp_seq=7 Destination Host Unreachable
From 10.0.2.15 icmp_seq=8 Destination Host Unreachable
From 10.0.2.15 icmp_seq=9 Destination Host Unreachable
From 10.0.2.15 icmp_seq=10 Destination Host Unreachable
From 10.0.2.15 icmp_seq=11 Destination Host Unreachable
From 10.0.2.15 icmp_seq=12 Destination Host Unreachable
From 10.0.2.15 icmp_seq=13 Destination Host Unreachable
From 10.0.2.15 icmp_seq=14 Destination Host Unreachable
From 10.0.2.15 icmp_seq=15 Destination Host Unreachable
From 10.0.2.15 icmp_seq=16 Destination Host Unreachable
From 10.0.2.15 icmp_seq=17 Destination Host Unreachable
From 10.0.2.15 icmp_seq=18 Destination Host Unreachable
From 10.0.2.15 icmp_seq=19 Destination Host Unreachable
From 10.0.2.15 icmp_seq=20 Destination Host Unreachable
From 10.0.2.15 icmp_seq=21 Destination Host Unreachable
From 10.0.2.15 icmp_seq=22 Destination Host Unreachable
```

We can observe these packets by running sniffer on attacker container as shown below –

```
File Machine View Input Devices Help
Activities Terminal Nov 26 18:21
seed@VM: ~/.../volumes
seed@VM: ~/.../volumes
seed@VM: ~/.../volumes
dst      = 02:42:72:2c:ef:18
src      = 02:42:0a:09:00:05
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 11349
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xf83b
src      = 10.9.0.5
dst      = 10.0.2.11
\options
###[ ICMP ]###
type     = echo-request
code     = 0
chksum   = 0x571a
id       = 0x2e
seq      = 0x2
###[ Raw ]###
```


Task 1.2 – Spoofing ICMP Packets

Scapy has a feature to let us set the fields of IP packets to arbitrary values. Now we will spoof IP packets with irrelevant source IP addresses.

In particular, we will be spoofing ICMP echo-request packets and then send them to a different VM, though on the same network.

In this task, we use Wireshark to check if our request is accepted by the receiver. In case it is accepted, we get an echo-reply packet from the spoofed IP address.

- First, create a new python file spoofICMP.py by creating an IP object a, and then specifying arbitrary src and destination IP address.
- Next, create an ICMP object, b with default request type being echo-request. Then stack a and b together to form a new object. It is basically adding b as the payload field of a and modifying the fields of a correspondingly to get a new ICMP packet
- We then send the packet.



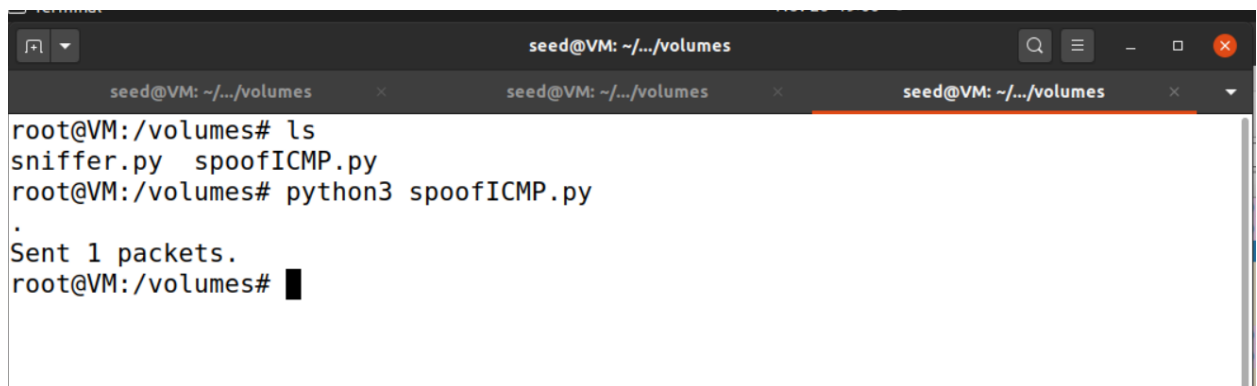
```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Nov 26 18
seed@VM: ~/.../volumes
seed@VM: ~/.../volumes
from scapy.all import *

a = IP()

a.src = '8.8.8.8' # Arbitrary source - Google
a.dst = '10.9.0.6' #Host B

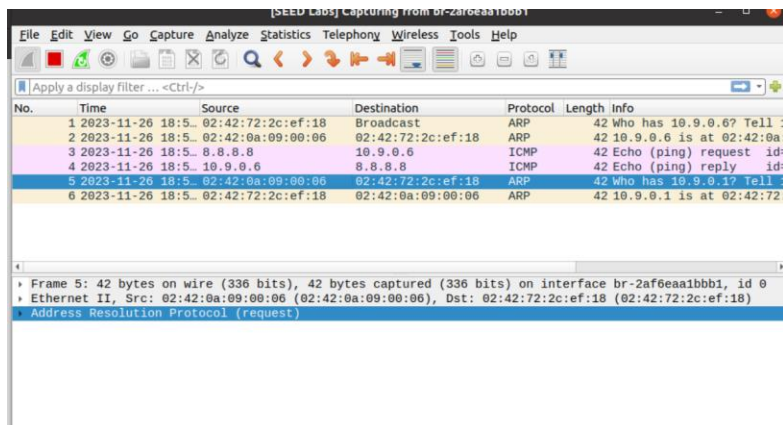
b = ICMP()
p = a/b
send(p)
```

- We run this code in attacker container –



```
seed@VM: ~/.../volumes
root@VM:/volumes# ls
sniffer.py  spoofICMP.py
root@VM:/volumes# python3 spoofICMP.py
.
Sent 1 packets.
root@VM:/volumes#
```

- As can be seen below in Wireshark, the request was accepted by the receiver –



Task 1.3 – Traceroute

In this task, we want to use Scapy to estimate the distance in terms of number of nodes (routers) in between the VM and specified destination. This is done using traceroute.py (our own traceroute tool).

The basic principle –

- Send a packet to the destination with its TTL set to 1 at the start
- This will be sent to the first router and is subsequently dropped and send us an ICMP error message, due to the fact that it's time-to-live has exceeded.
- We can then increase the TTL field to 2 and send another packet which can go till second router
- We need to repeat this till we reach our destination.
- The code is written in file traceroute.py in volumes folder in VM as shown –

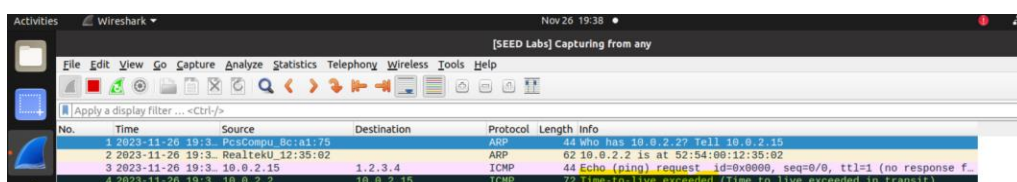
```
seed@VM: ~/.../volumes
seed@VM: ~/.../volumes
from scapy.all import *

a = IP()
a.dst = '1.2.3.4'
a.ttl = 1

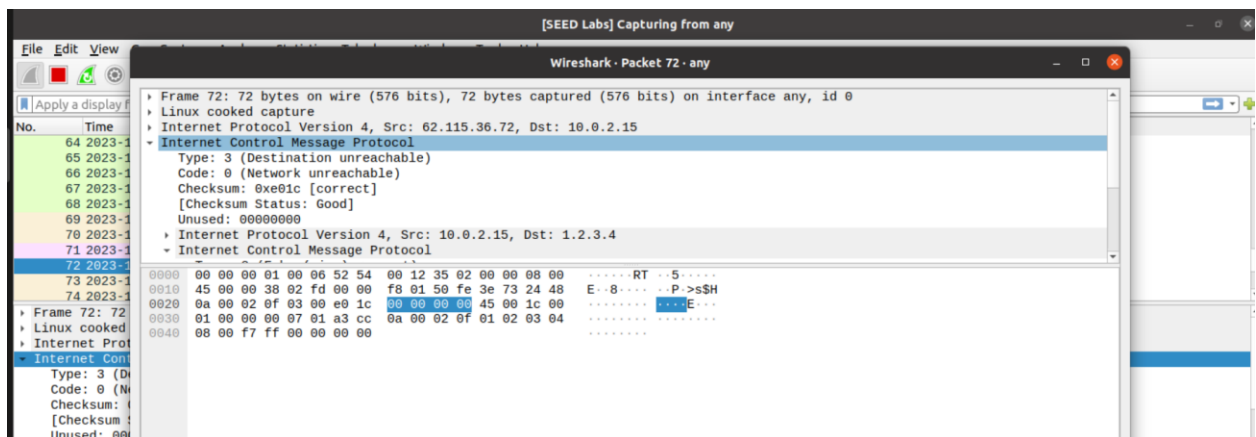
b = ICMP()
p = a/b

send(p)
```

- For TTL = 1, we can see the packet is dropped due to the time-to-live exceeded as shown –



- For the destination 1.2.3.4, When TTL was 8, I got a message that the destination is not reachable. Which makes sense as it is a random made-up address as shown below



- Let's try a reachable address like 8.8.8.8 which is for google
- For this destination, I got TTL exceeded until 10 and finally succeeded at TTL = 11 as shown below –

[SEED Labs] Capturing from any						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
Apply a display filter ... <Ctrl-/>						
No.	Time	Source	Destination	Protocol	Length	Info
1	2023-11-26 19:5...	PcsCompu_8c:a1:75		ARP	44	Who has 10.0.2.2? Tell 10.0.2.15
2	2023-11-26 19:5...	RealtekU_12:35:02		ARP	62	10.0.2.2 is at 52:54:00:12:35:02
3	2023-11-26 19:5...	10.0.2.15	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=10 (no response ...)
4	2023-11-26 19:5...	142.251.233.237	10.0.2.15	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
5	2023-11-26 19:5...	PcsCompu_8c:a1:75		ARP	44	Who has 10.0.2.2? Tell 10.0.2.15
6	2023-11-26 19:5...	RealtekU_12:35:02		ARP	62	10.0.2.2 is at 52:54:00:12:35:02
7	2023-11-26 19:5...	10.0.2.15	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=11 (reply in 8)
8	2023-11-26 19:5...	8.8.8.8	10.0.2.15	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=55 (request in 7)

Task 1.4 – Sniffing and then Spoofing

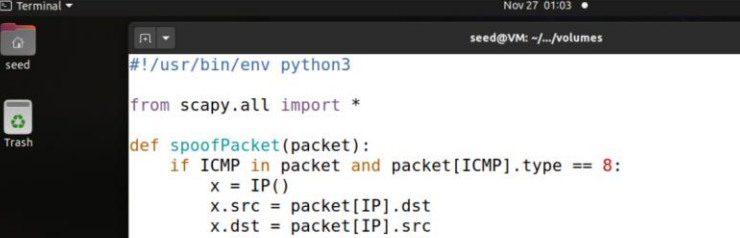
In this one we want to implement a sniff-and-then-spoof program.

Requirement – 2 machines on one LAN (VM and User container)

Steps –

- Ping an IP X from user container. This creates an ICMP echo-request packet
- If X is alive, we get a ICMP echo-reply and print response
- The sniff-and-then-spoof program is run on the VM and which can monitors LAN through packet sniffing
- Whenever it comes across a an ICMP echo-request, regardless of the destination address, the program should immediately send an echo-reply using the spoofing technique

- The code which is run on the VM with root privileges –



The screenshot shows a terminal window titled "SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox". The terminal is running a Python script named "SniffThenSnoop.py". The script uses the Scapy library to craft and send a spoofed ICMP Echo (ping) packet. The target IP is 10.10.10.10, and the spoofed source IP is 10.10.10.1. The terminal output shows the script being executed and the resulting packet capture data.

```
#!/usr/bin/env python3

from scapy.all import *

def spoofPacket(packet):
    if ICMP in packet and packet[ICMP].type == 8:
        x = IP()
        x.src = packet[IP].dst
        x.dst = packet[IP].src
        x.ihl = packet[IP].ihl
        x[IP].dst = packet[IP].src

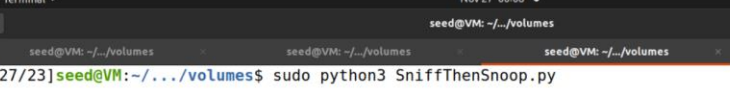
        y = ICMP()
        y.type = 0
        y.id = packet[ICMP].id
        y.seq = packet[ICMP].seq

        unloaded = packet[Raw].load
        newPacket = x / y / unloaded
        send(newPacket)

packet = sniff(filter = 'icmp', prn = spoofPacket)

"SniffThenSnoop.py" 23L, 531C
```

- For ping 1.2.3.4 (A host which doesn't exist on the internet)
 - Run the code on the VM



```
File Machine View Input Devices Help
Activities Terminal Nov 27 00:08
seed@VM: ~/../volumes seed@VM: ~/../volumes seed@VM: ~/../volumes seed@
[11/27/23]seed@VM:~/../volumes$ sudo python3 SniffThenSnoop.py
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

- Send a ping on a user container (Host A) to 1.2.3.4. We can see that the Packet is sent.

```

seed@VM: ~/../volumes
root@a7c843dd9f0e:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
^C

```

- We can see that we receive a echo-reply to the echo-request on wire shark

No.	Time	Source	Destination	Protocol	Length	Info
3569	2023-11-27 00:00:00.000000	10.9.0.5	1.2.3.4	ICMP	100	Echo (ping) request id=0x0023, seq=20/5120, ttl=64 (no response yet)
3570	2023-11-27 00:00:00.000000	10.9.0.5	1.2.3.4	ICMP	100	Echo (ping) request id=0x0023, seq=20/5120, ttl=64 (reply in progress)
3571	2023-11-27 00:00:00.000000	1.2.3.4	10.9.0.5	ICMP	100	Echo (ping) reply id=0x0023, seq=20/5120, ttl=64 (request in progress)
3572	2023-11-27 00:00:00.000000	10.9.0.5	1.2.3.4	ICMP	100	Echo (ping) request id=0x0023, seq=21/5376, ttl=64 (no response yet)
3573	2023-11-27 00:00:00.000000	10.9.0.5	1.2.3.4	ICMP	100	Echo (ping) request id=0x0023, seq=21/5376, ttl=64 (reply in progress)
3574	2023-11-27 00:00:00.000000	10.9.0.5	1.2.3.4	ICMP	100	Echo (ping) request id=0x0023, seq=21/5376, ttl=64 (reply in progress)
3575	2023-11-27 00:00:00.000000	1.2.3.4	10.9.0.5	ICMP	100	Echo (ping) reply id=0x0023, seq=21/5376, ttl=64 (request in progress)
3576	2023-11-27 00:00:00.000000	10.9.0.5	1.2.3.4	ICMP	100	Echo (ping) request id=0x0023, seq=22/5632, ttl=64 (no response yet)
3577	2023-11-27 00:00:00.000000	10.9.0.5	1.2.3.4	ICMP	100	Echo (ping) request id=0x0023, seq=22/5632, ttl=64 (reply in progress)
3578	2023-11-27 00:00:00.000000	10.9.0.5	1.2.3.4	ICMP	100	Echo (ping) request id=0x0023, seq=22/5632, ttl=64 (reply in progress)
3579	2023-11-27 00:00:00.000000	1.2.3.4	10.9.0.5	ICMP	100	Echo (ping) reply id=0x0023, seq=22/5632, ttl=64 (request in progress)

- For ping 10.9.0.99 (A host which doesn't exist on the LAN)

- Run the code on the VM

```

seed@VM: ~/../volumes
[11/27/23]seed@VM:~/../volumes$ sudo python3 SniffThenSnoop.py

```

- Send a ping on a user container (Host A) to 10.9.0.99. Packets are not sent as host is unreachable

```

root@a7c843dd9f0e:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
From 10.9.0.5 icmp_seq=4 Destination Host Unreachable
From 10.9.0.5 icmp_seq=5 Destination Host Unreachable
From 10.9.0.5 icmp_seq=6 Destination Host Unreachable
From 10.9.0.5 icmp_seq=7 Destination Host Unreachable
From 10.9.0.5 icmp_seq=8 Destination Host Unreachable
From 10.9.0.5 icmp_seq=9 Destination Host Unreachable
From 10.9.0.5 icmp_seq=10 Destination Host Unreachable
From 10.9.0.5 icmp_seq=11 Destination Host Unreachable

```


- We can see that we receive a echo-reply to the echo-request on Wireshark as shown below –

SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Activities Wireshark Nov 27 00:40

[SEED Labs] Capturing from any

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
7385	2023-11-27 00:3...	8.8.8.8	10.0.2.15	ICMP	100	Echo (ping) reply id=0x0029, seq=234/59904, ttl=64
7386	2023-11-27 00:3...	10.9.0.5	8.8.8.8	ICMP	100	Echo (ping) request id=0x0029, seq=235/60160, ttl=64 (no res...
7387	2023-11-27 00:3...	10.9.0.5	8.8.8.8	ICMP	100	Echo (ping) request id=0x0029, seq=235/60160, ttl=64 (reply ...
7388	2023-11-27 00:3...	10.0.2.15	8.8.8.8	ICMP	100	Echo (ping) request id=0x0029, seq=235/60160, ttl=63 (reply ...
7389	2023-11-27 00:3...	8.8.8.8	10.0.2.15	ICMP	100	Echo (ping) reply id=0x0029, seq=235/60160, ttl=55 (request...
7390	2023-11-27 00:3...	8.8.8.8	10.9.0.5	ICMP	100	Echo (ping) reply id=0x0029, seq=235/60160, ttl=54 (request...
7391	2023-11-27 00:3...	8.8.8.8	10.9.0.5	ICMP	100	Echo (ping) reply id=0x0029, seq=235/60160, ttl=54
7392	2023-11-27 00:3...	8.8.8.8	10.0.2.15	ICMP	100	Echo (ping) reply id=0x0029, seq=235/60160, ttl=64
7393	2023-11-27 00:3...	10.9.0.5	8.8.8.8	ICMP	100	Echo (ping) request id=0x0029, seq=236/60416, ttl=64 (no res...
7394	2023-11-27 00:3...	10.9.0.5	8.8.8.8	ICMP	100	Echo (ping) request id=0x0029, seq=236/60416, ttl=64 (reply ...
7395	2023-11-27 00:3...	10.0.2.15	8.8.8.8	ICMP	100	Echo (ping) request id=0x0029, seq=236/60416, ttl=63 (reply ...