

# Lab 2 - Packet Sniffing and Spoofing Lab

● Graded

## Student

Alexander Lotero

## Total Points

105 / 100 pts

### Question 1

(no title)

10 / 10 pts

#### 1.1 Task 1.1A

10 / 10 pts

✓ - 0 pts Correct

### Question 2

#### Task 1.1B

15 / 15 pts

✓ - 0 pts Correct

### Question 3

#### Task 1.2

15 / 15 pts

✓ + 15 pts Correct

### Question 4

#### Task 1.3

20 / 20 pts

✓ - 0 pts Correct

### Question 5

#### Task 1.4

40 / 40 pts

✓ - 0 pts Correct

### Question 6

#### Early/Late Submission Bonus

5 / 0 pts

✓ + 5 pts Early submission

## **Q1**

**10 Points**

Lab PDF:

[https://seedsecuritylabs.org/Labs\\_20.04/Files/Sniffing\\_Spoofing/Sniffing\\_Spoofing.pdf](https://seedsecuritylabs.org/Labs_20.04/Files/Sniffing_Spoofing/Sniffing_Spoofing.pdf)

**Note: For this lab, we will only do up to task 1.4, the scapy portions. Stop on page 7.**

- Lab setup files: [Labsetup.zip](#) - Please use this Labsetup for this lab.
- [Docker Manual](#) (if more help is needed)

***Note: Make sure your environment is setup, especially docker, in section 2.1, before proceeding to the tasks***

Earliest acceptance date: 14 October (+1% per day, up to +5% bonus for five days early)

Normal due date: 19 October

Latest acceptance date: 29 October (-10% points)

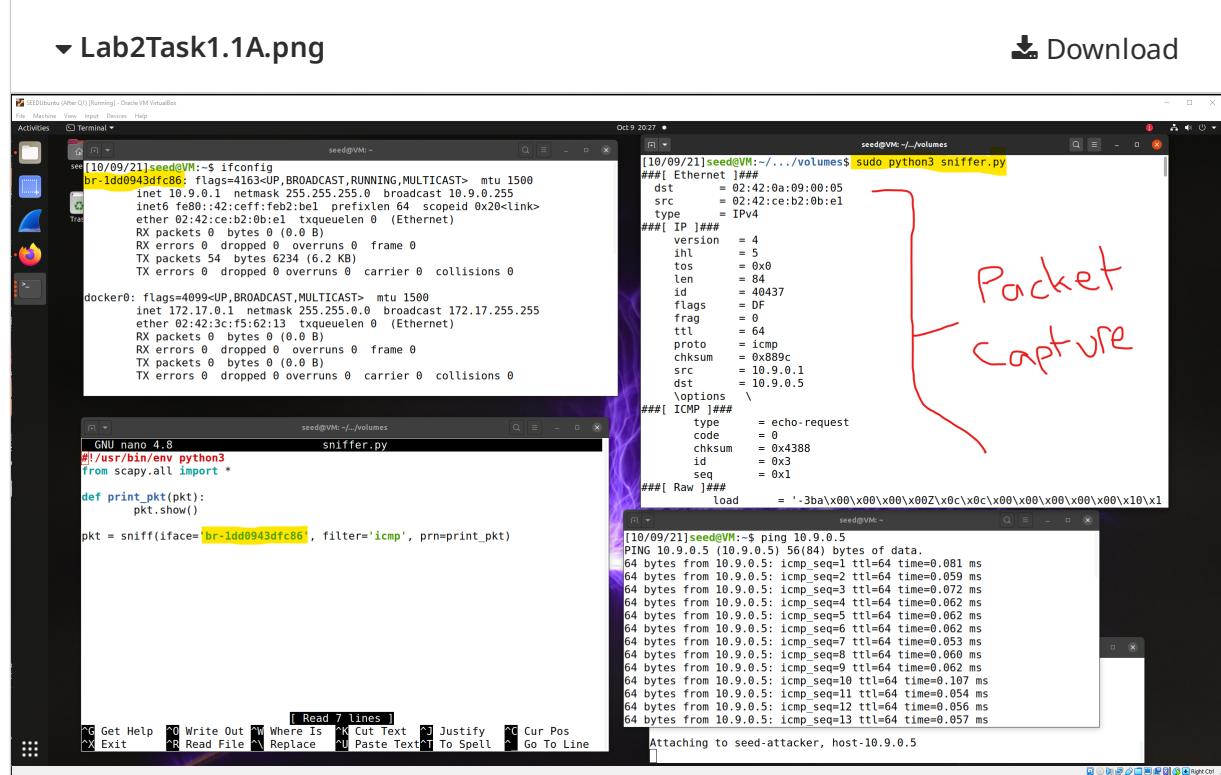
Please follow the instructions from the PDF document, but submit your work based on the instructions below.

## Q1.1 Task 1.1A

## 10 Points

Run the program with root privilege and demonstrate that you can indeed capture packets.

(1) Show a screenshot showing that packets are being captured.



## ▼ sniffer.py

 Download

```
1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 def print_pkt(pkt):
5     pkt.show()
6
7 pkt = sniff(filter='net 128.230.0.0/16', prn=print_pkt)
8
```

(2) Write a sentence to answer: What happens when you don't run with root privileges? Why?

Attempting to run sniffer.py to sniff packets without root privileges results in the following error: "PermissionError: [Errno 1] Operation not permitted."

This means that the current user does not have permission to sniff packets.

More specifically, when not run with Administrator privileges, the OS denies python/scapy's request to analyze traffic on the socket. This request is permitted when run with Administrator privileges.

## Q2 Task 1.1B

15 Points

(1) Capture only the ICMP packet. Submit your code and take a screenshot demonstrating that ICMP packets are captured.

▼ Lab2Task1.1B-1.png [Download](#)

The screenshot shows a Linux desktop environment with several windows open:

- A terminal window titled "seed@VM: ~" showing the output of "ifconfig". It lists two interfaces: "br-4137ac48b18a" and "dockero0".
- A terminal window titled "seed@VM: ~/.volumes" showing the output of "tcpdump -n". A yellow box highlights the "proto = icmp" line in the raw dump, with a blue star drawn next to it. Handwritten text above the star says "filtering for only ICMP". Another handwritten note below the star says "doesn't capture netcat (not icmp)".
- A terminal window titled "seed@VM: ~/.volumes" showing the output of "sniffer.py". It includes a Python script for sniffing ICMP. A yellow box highlights the "filter='icmp'" line in the script.
- A terminal window titled "seed@VM: ~" showing the output of "ping 10.9.0.5". Handwritten text at the bottom of this window says "Read 7 lines".
- A terminal window titled "seed@VM: ~" showing the output of "netstat -an | grep 4443". Handwritten text at the bottom of this window says "Read 7 lines".
- A terminal window titled "seed@VM: ~" showing the output of "docker-compose up". Handwritten text at the bottom of this window says "Read 7 lines".

(2) Capture any TCP packet that comes from a particular IP and with a destination port number 23. Submit your code and take a screenshot demonstrating that.

▼ Lab2Task1.1B-2.png [Download](#)

The screenshot shows a Linux desktop environment with several windows open:

- A terminal window titled "seed@VM: ~" showing the output of "ifconfig". It lists two interfaces: "br-4137ac48b18a" and "dockero0".
- A terminal window titled "seed@VM: ~/.volumes" showing the output of "tcpdump -n". A yellow box highlights the "proto = tcp" line in the raw dump, with a blue star drawn next to it. Handwritten text above the star says "=Port 23".
- A terminal window titled "seed@VM: ~/.volumes" showing the output of "sniffer.py". It includes a Python script for sniffing TCP port 23. A yellow box highlights the "filter='tcp and host 10.9.0.5 and dst port 23'" line in the script.
- A terminal window titled "seed@VM: ~" showing the output of "nc -l -p 23". Handwritten text at the bottom of this window says "Read 7 lines".
- A terminal window titled "seed@VM: ~" showing the output of "nc 10.9.0.5 23". Handwritten text at the bottom of this window says "Read 7 lines".
- A terminal window titled "seed@VM: ~" showing the output of "netstat -an | grep 4443". Handwritten text at the bottom of this window says "Read 7 lines".
- A terminal window titled "seed@VM: ~" showing the output of "docker-compose up". Handwritten text at the bottom of this window says "Read 7 lines".

(3) Capture packets coming from or to going to a particular subnet. Submit your code and take a screenshot demonstrating the traffic generated.

▼ Lab2Task1.1B-3.png

Download

The screenshot shows a desktop environment with several windows open:

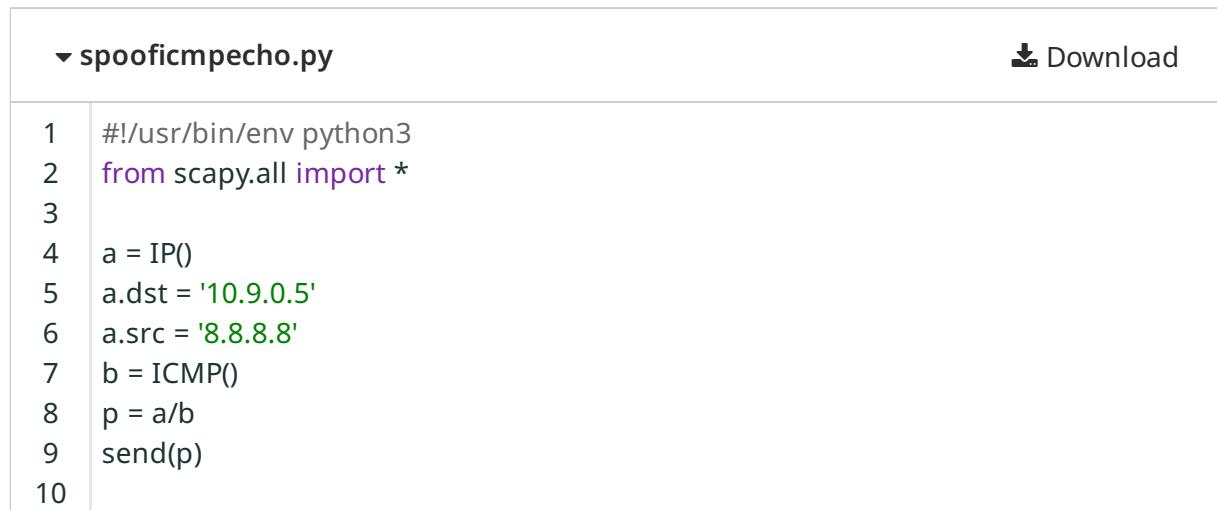
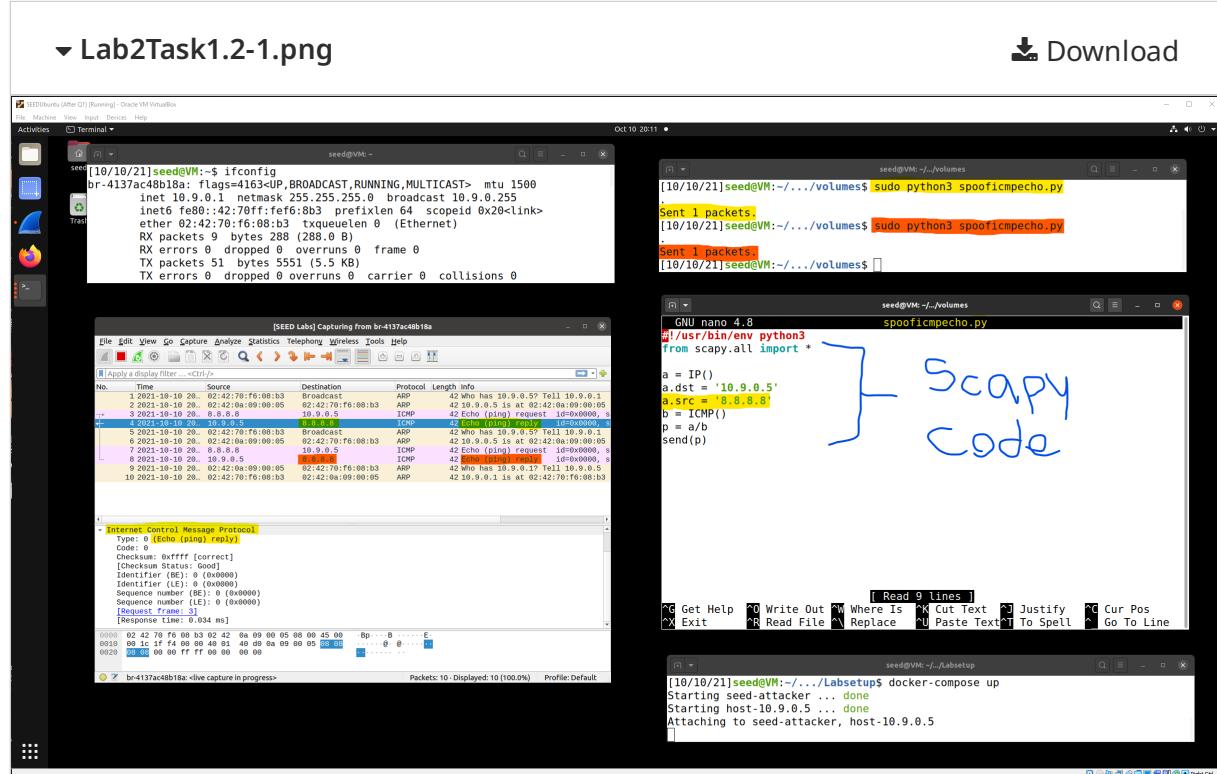
- A terminal window titled "seed@VM:~" showing the output of the command "ifconfig". It lists network interfaces "br-4137ac48b18a" and "eth0" with their respective details like MTU, broadcast address, and link layer information.
- A terminal window titled "seed@VM:~" showing the output of the command "ping 128.230.0.1". It displays a series of ICMP echo requests and replies between the local host and a target at 128.230.0.1.
- A terminal window titled "seed@VM:~/.volumes" showing the raw hex dump of an ICMP echo reply packet. The dump includes fields like seq, load, dst, src, type, version, ihl, tos, len, id, flags, frag, ttl, proto, checksum, and options.
- A terminal window titled "seed@VM:~/.volumes" showing the source code of a Python script named "sniffer.py". The script uses Scapy to filter for packets on the interface "net 128.230.0.0/16" and prints them.
- A terminal window titled "seed@VM:~/.Labsetup" showing the command "docker-compose up". It starts a container named "seed-attacker" and attaches to it.

Q3 Task 1.2

**15 Points**

Spoof an ICMP echo request packet with source IP address 8.8.8.8 from the first VM and send to the second VM. Use Wireshark on the second VM to show that it replies back with echo replies.

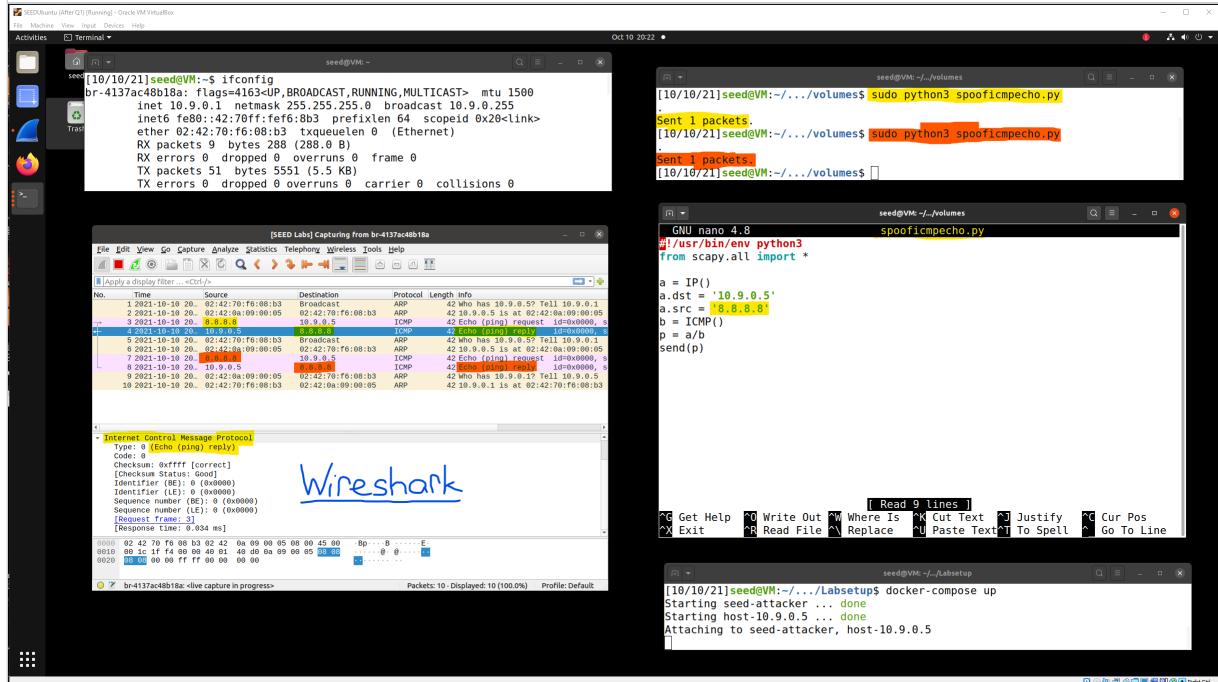
## (1) Submit your scapy code



(2) Submit a screenshot of Wireshark showing the spoof echo request from 8.8.8.8 and that the VM replied back to it with an echo reply.

## ▼ Lab2Task1.2-2.png

[Download](#)



## Q4 Task 1.3

20 Points

Implement TCP traceroute using `scapy`. Do NOT use the built-in `scapy` traceroute function. Perform a traceroute to 8.8.8.8. Submit your code and take screenshot of your program's output.

▼ Lab2Task1.3-Code.png [Download](#)

The screenshot shows a Linux desktop environment with several windows open:

- A terminal window titled "seed@VM: ~/.volumes" showing the output of running `sudo python3 traceroute.py`. The output lists 11 routers along the path to 8.8.8.8, ending with a message: "Your trace is complete. Have a nice day!"
- A terminal window titled "seed@VM: ~/.volumes" showing the source code for `traceroute.py` in a nano editor. The code uses scapy to construct ICMP packets and send them to destination ports to determine the route.
- A terminal window titled "seed@VM: ~/" showing network interface statistics for enp0s3.
- A terminal window titled "seed@VM: ~/.Labsetup" showing the command `docker-compose up` being run.

▼ Lab2Task1.3-Output.png [Download](#)

The screenshot shows a Linux desktop environment with several windows open:

- A terminal window titled "seed@VM: ~/.volumes" showing the output of running `sudo python3 traceroute.py`. The output lists 11 routers along the path to 8.8.8.8, ending with a message: "Your trace is complete. Have a nice day!" A blue bracket labeled "Program Output" points to this terminal window.
- A terminal window titled "seed@VM: ~/.volumes" showing the source code for `traceroute.py` in a nano editor. The code uses scapy to construct ICMP packets and send them to destination ports to determine the route.
- A terminal window titled "seed@VM: ~/" showing network interface statistics for enp0s3.
- A terminal window titled "seed@VM: ~/.Labsetup" showing the command `docker-compose up` being run.

▼ traceroute.py

 Download

```
1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 hops = 1
5 routenum = []
6 routeip = []
7 while True:
8     a = sr1(IP(dst = '8.8.8.8', ttl = hops)/ICMP(), verbose = 0)
9     routenum.append(hops)
10    routeip.append(a.src)
11    hops += 1
12    if a.src == '8.8.8.8':
13        break
14 for x in range(hops - 1):
15     print(routenum[x], end = ' ')
16     print(routeip[x])
17 print('Your trace is complete. Have a nice day!')
```

## Q5 Task 1.4

40 Points

Sniffing and-then Spoofing. You need two machines on the same LAN: the VM and the user container.

---

You will find that when you ping from the terminal, this IP will have a destination unreachable response. That is the expected result. Your program does not need to work for this IP. (You do not need to force it to work by performing ARP spoofing.) You will, however, need to explain why your program does not work for IP 10.9.0.99 (while it's suppose to work for 1.2.3.4 and 8.8.8.8).

---

(1) Submit your scapy code for sniffing and then spoofing.

▼ sniffandspoof.py

 Download

```
1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 def spoofer(a):
5     if a[0].getlayer(IP).type != 8:
6         return
7     ip = IP(src=a[0].getlayer(IP).dst, dst=a[0].getlayer(IP).src)
8     icmp = ICMP(type='echo-reply', id=a[0].getlayer(ICMP).id,
9      seq=a[0].getlayer(ICMP).seq)
10    payload = a[0].getlayer(IP).load
11    b = ip/icmp/payload
12
13    print('COMMENCING PACKET SPOOF...')
14    print()
15    print('Source: ' + a[0].getlayer(IP).dst + ' ' + 'Destination: ' +
16 a[0].getlayer(IP).src)
17
18    while True:
19        a = sniff(filter='icmp', prn=spoofer)
20
21
```

(2) Ping 1.2.3.4 and show screenshots of the output from your program and with ping from the terminal

### ▼ Lab2Task1.4-2.png

[Download](#)

The screenshot shows two terminal windows. The top window runs the command `sudo python3 sniffandspoof.py`, which sends ICMP echo-reply packets to 10.0.2.4. The bottom window runs `ping -c 3 1.2.3.4`, receiving three replies from 1.2.3.4 with increasing sequence numbers (seq=1, seq=2, seq=3) and decreasing times (29.5 ms, 25.2 ms, 26.2 ms). A red bracket highlights the sequence numbers in the ping output, and a green arrow points from the first sequence number to the first reply in the sniffing output.

```
[10/14/21]seed@VM:~/.../volumes$ sudo python3 sniffandspoof.py
COMMENCING PACKET SPOOF...
Source: 1.2.3.4 Destination: 10.0.2.4
Sent 1 packets.
COMMENCING PACKET SPOOF...
Source: 1.2.3.4 Destination: 10.0.2.4
.
Sent 1 packets.
COMMENCING PACKET SPOOF...
Source: 1.2.3.4 Destination: 10.0.2.4
Sent 1 packets.

[10/14/21]seed@VM:~/.../volumes$ ping -c 3 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=29.5 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=25.2 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=26.2 ms
--- 1.2.3.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2013ms
rtt min/avg/max/mdev = 25.238/26.961/29.495/1.829 ms
[10/14/21]seed@VM:~$
```

```
GNU nano 4.8
from scapy.all import *
sniffandspoof.py

def spoofer(a):
    if a[0].getlayer(IP).type != 8:
        return
    ip = IP(src=a[0].getlayer(IP).dst, dst=a[0].getlayer(IP).src)
    icmp = ICMP(type='echo-reply', id=a[0].getlayer(ICMP).id, seq=a[0].getlayer(ICMP).seq)
    payload = a[0].getlayer(IP).load
    b = ip/icmp/payload

    print('COMMENCING PACKET SPOOF...')
    print()
    print('Source: ' + a[0].getlayer(IP).dst + ' ' + 'Destination: ' + a[0].getlayer(IP).src)

    send(b)

while True:
    a = sniff(filter='icmp', prn=spoofer)
```

(3) Ping 10.9.0.99 and show screenshots of the output from your program and with ping from the terminal. If this does not work, please explain why.

### ▼ Lab2Task1.4-3.png

[Download](#)

The screenshot shows two terminal windows. The top window runs `sudo python3 sniffandspoof.py`, with a large red annotation in the center reading "nothing, see explanation on gradescope". The bottom window runs `ping -c 3 10.9.0.99`, showing three ICMP error messages: "Destination Host Unreachable" for seq=1, seq=2, and seq=3. A yellow box highlights the destination address 10.9.0.99 in the ping command.

```
[10/14/21]seed@VM:~/.../volumes$ sudo python3 sniffandspoof.py
nothing, see explanation
on gradescope
```

```
[10/14/21]seed@VM:~/.../volumes$ ping -c 3 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Host Unreachable
From 10.9.0.1 icmp_seq=2 Destination Host Unreachable
From 10.9.0.1 icmp_seq=3 Destination Host Unreachable
--- 10.9.0.99 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2031ms
pipe 3
[10/14/21]seed@VM:~$
```

```
GNU nano 4.8
from scapy.all import *
sniffandspoof.py

def spoofer(a):
    if a[0].getlayer(IP).type != 8:
        return
    ip = IP(src=a[0].getlayer(IP).dst, dst=a[0].getlayer(IP).src)
    icmp = ICMP(type='echo-reply', id=a[0].getlayer(ICMP).id, seq=a[0].getlayer(ICMP).seq)
    payload = a[0].getlayer(IP).load
    b = ip/icmp/payload

    print('COMMENCING PACKET SPOOF...')
    print()
    print('Source: ' + a[0].getlayer(IP).dst + ' ' + 'Destination: ' + a[0].getlayer(IP).src)

    send(b)

while True:
    a = sniff(filter='icmp', prn=spoofer)
```

(4) Ping 8.8.8.8 and show screenshots of the output from your program and with ping from the terminal

## ▼ Lab2Task1.4-4.png

[Download](#)

The screenshot shows a terminal window with two panes. The left pane displays the source code for `sniffandspoof.py`, which is a Python script using Scapy to intercept ICMP echo-reply packets and spoof them back to the source. The right pane shows the output of a ping command to the IP address 3.8.8.8. The output includes several ICMP echo-reply packets from the host 8.8.8.8, with some being marked as duplicates due to the real host also replying. A handwritten note on the right side of the terminal window states: "Duplicates b/c real host is also replying".

```
seed@VM:~/.../volumes$ sudo python3 sniffandspoof.py
COMMENCING PACKET SPOOF...
Source: 8.8.8.8 Destination: 10.0.2.4
Sent 1 packets.
COMMENCING PACKET SPOOF...
Source: 8.8.8.8 Destination: 10.0.2.4
Sent 1 packets.
COMMENCING PACKET SPOOF...
Source: 8.8.8.8 Destination: 10.0.2.4
Sent 1 packets.

[10/14/21]seed@VM:~/.../volumes$ ping -c 3 3.8.8.8
PING 3.8.8.8 (9.8.8.8) 36(84) bytes of data.
64 bytes from 9.8.8.8: icmp_seq=1 ttl=64 time=26.4 ms
64 bytes from 9.8.8.8: icmp_seq=1 ttl=64 time=28.2 ms (DUP!)
64 bytes from 9.8.8.8: icmp_seq=2 ttl=64 time=19.3 ms
64 bytes from 9.8.8.8: icmp_seq=2 ttl=64 time=32.4 ms (DUP!)
64 bytes from 9.8.8.8: icmp_seq=3 ttl=64 time=21.5 ms
--- 3.8.8.8 ping statistics ---
3 packets transmitted, 3 received, +2 duplicates, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 19.282/25.568/32.388/4.669 ms
[10/14/21]seed@VM:~$
```

```
GNU nano 4.8
from scapy.all import *
def spoofer(a):
    if a[0].getlayer(IP).type != 8:
        return
    ip = IP(src=a[0].getlayer(IP).dst, dst=a[0].getlayer(IP).src)
    icmp = ICMP(type='echo-reply', id=a[0].getlayer(ICMP).id, seq=a[0].getlayer(ICMP).seq)
    payload = a[0].getlayer(IP).load
    b = ip/icmp/payload
    print('COMMENCING PACKET SPOOF...')
    print()
    print('Source: ' + a[0].getlayer(IP).dst + ' ' + 'Destination: ' + a[0].getlayer(IP).src)
    send(b)
while True:
    a = sniff(filter='icmp', prn=spoofer)
```

(5) Explain the results for (2), (3), and (4).

Ping 1.2.3.4 is best example of the `sniffandspoof.py` program in action. The 1.2.3.4 host is not actually live on the internet, so without our program the ping is unable to reach the host and receives 100% packet loss. However, with our program running, the ping receives every icmp echo-reply packet that it is expecting. These packets are spoofed of course, but to the terminal running ping, our program makes it appear as though the 1.2.3.4 host is live and responding (see screenshot).

Ping 10.9.0.99 does not work because while the address is within the LAN, it is not actually an existing host. This result is explained by the ARP protocol. Firstly, our device will check its ARP cache for the 10.9.0.99 device's MAC address. There is no MAC address stored in our ARP cache for this address because we have not communicated with it before. Next, our device will issue an ARP request broadcast across the LAN via the router at 10.9.0.1 looking for the MAC address of the device at 10.9.0.99. No such device exists, so our machine will never resolve that address and therefore our ping will fail with the "Destination Host Unreachable" error (see screenshot).

As for 8.8.8.8, this ping, and our program's sniff and spoof work without a problem. That said, because 8.8.8.8 is an actual live host, we see duplicate reply packets in our ping output. This is because the ping terminal is first receiving a spoofed reply from our program followed by the real reply from the destination host (see screenshot).

**Q6 Early/Late Submission Bonus****0 Points**

Bonus points for early or late submission will be added here. You may submit up to five days early for an extra 5% bonus points added to the grade of this assignment, or up to 10% deducted for late submission.

Submissions more than 10 days late are not accepted without a medical or work approved reason.