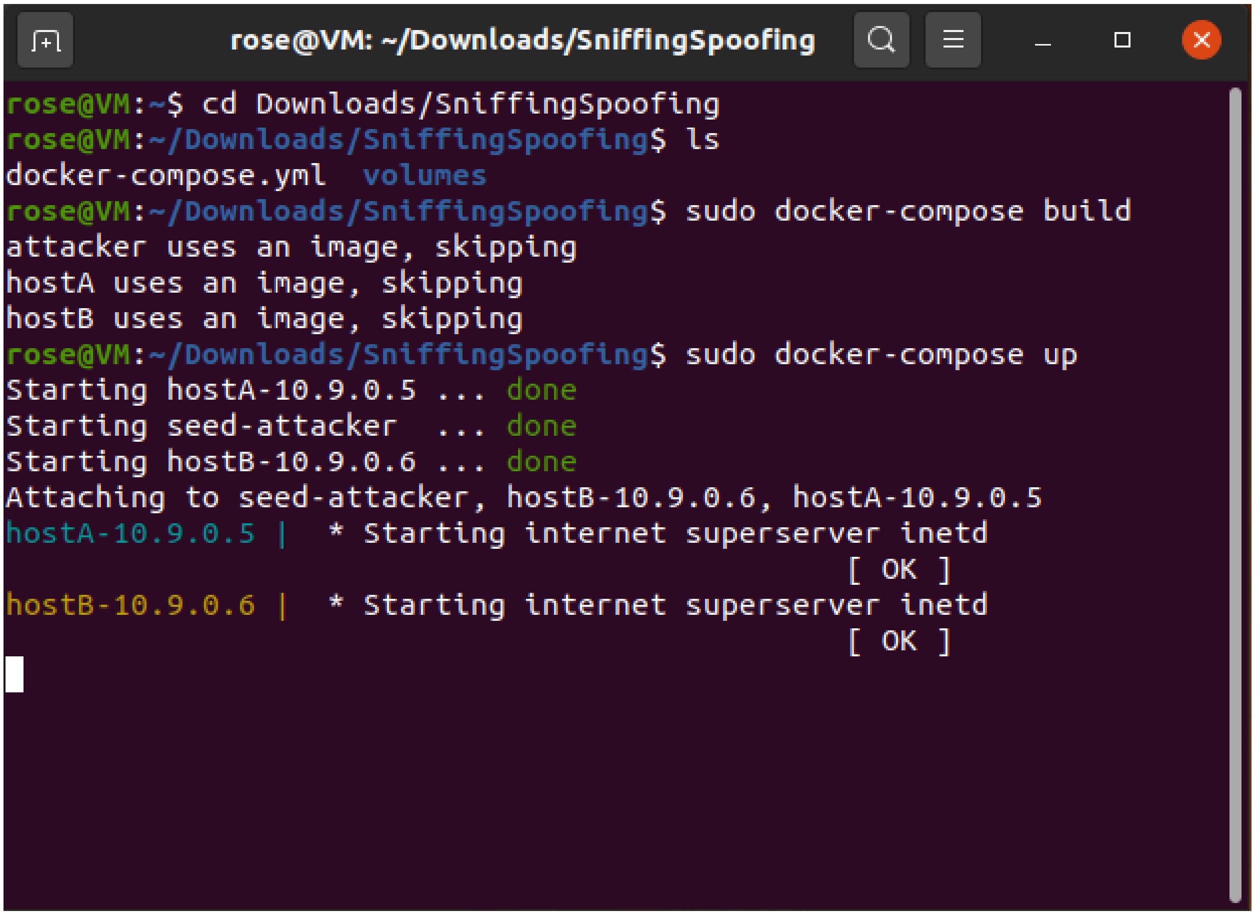
Lab5 Submission: Sniffing and Spoofing Demo

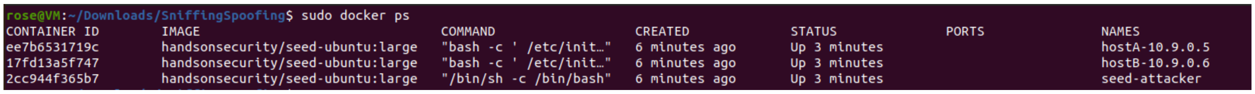
Samantha Jackson
CSCI 4321
Computer Security
March 1, 2025

Docker SniffingSpoofing

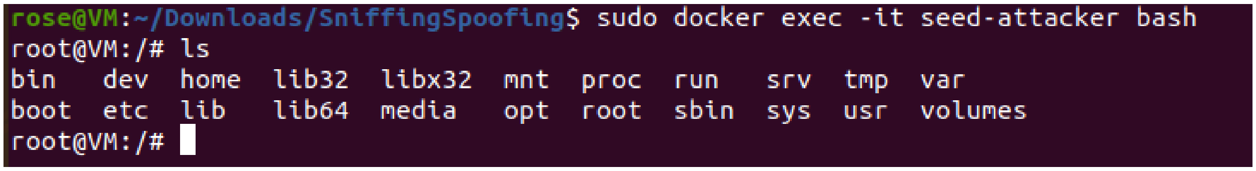
I set up the docker for SniffingSpoofing which I downloaded from SEEDLAB. I use sudo docker ps and note the containers for later on. I then execute an interactive bash shell.



```
rose@VM:~$ cd Downloads/SniffingSpoofing
rose@VM:~/Downloads/SniffingSpoofing$ ls
docker-compose.yml  volumes
rose@VM:~/Downloads/SniffingSpoofing$ sudo docker-compose build
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
rose@VM:~/Downloads/SniffingSpoofing$ sudo docker-compose up
Starting hostA-10.9.0.5 ... done
Starting seed-attacker  ... done
Starting 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 ]
```



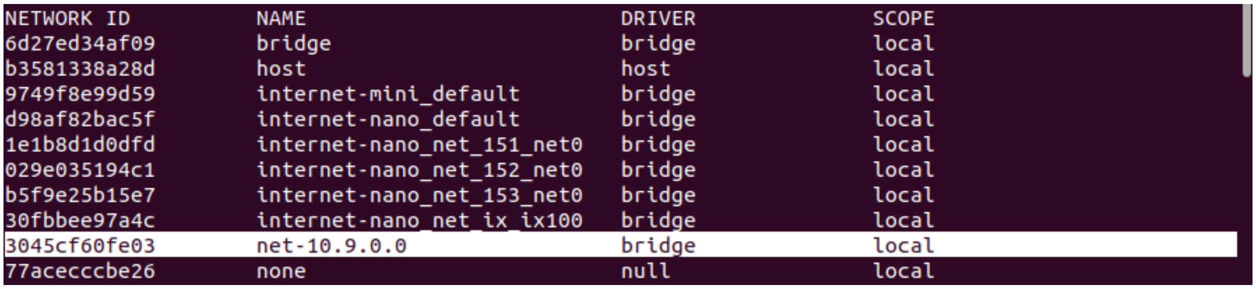
```
rose@VM:~/Downloads/SniffingSpoofing$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED         STATUS         PORTS     NAMES
ee7b6531719c   handsonsecurity/seed-ubuntu:large    "bash -c ' /etc/init…"   6 minutes ago   Up 3 minutes             hostA-10.9.0.5
17fd13a5f747   handsonsecurity/seed-ubuntu:large    "bash -c ' /etc/init…"   6 minutes ago   Up 3 minutes             hostB-10.9.0.6
2cc944f365b7   handsonsecurity/seed-ubuntu:large    "/bin/sh -c /bin/bash"   6 minutes ago   Up 3 minutes             seed-attacker
```



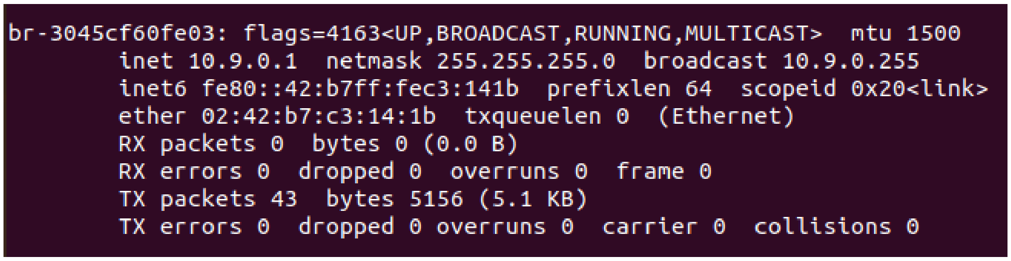
```
rose@VM:~/Downloads/SniffingSpoofing$ sudo docker exec -it seed-attacker bash
root@VM:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr  volumes
root@VM:/#
```

Finding the Bridge ID for Host IP

I found the Bridge ID for the host IP ‘br-3045cf60fe03’ using sudo docker network ls and ifconfig. We can use this for editing the *.py file in Lab5.



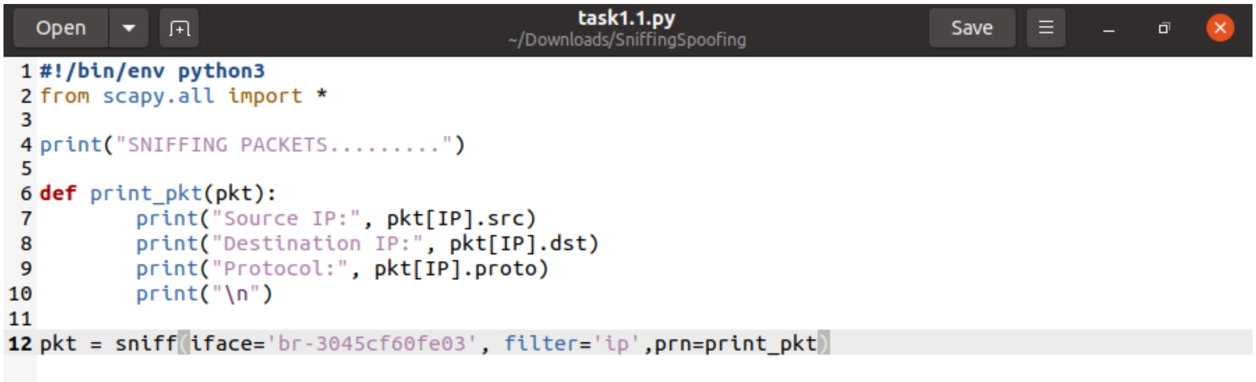
```
NETWORK ID     NAME                          DRIVER    SCOPE
6d27ed34af09   bridge                        bridge    local
b3581338a28d   host                          host      local
9749f8e99d59   internet-mini_default         bridge    local
d98af82bac5f   internet-nano_default         bridge    local
1e1b8d1d0dfd   internet-nano_net_151_net0    bridge    local
029e035194c1   internet-nano_net_152_net0    bridge    local
b5f9e25b15e7   internet-nano_net_153_net0    bridge    local
30fbbee97a4c   internet-nano_net_ix_ix100    bridge    local
3045cf60fe03   net-10.9.0.0                  bridge    local
77acecccbe26   none                          null      local
```



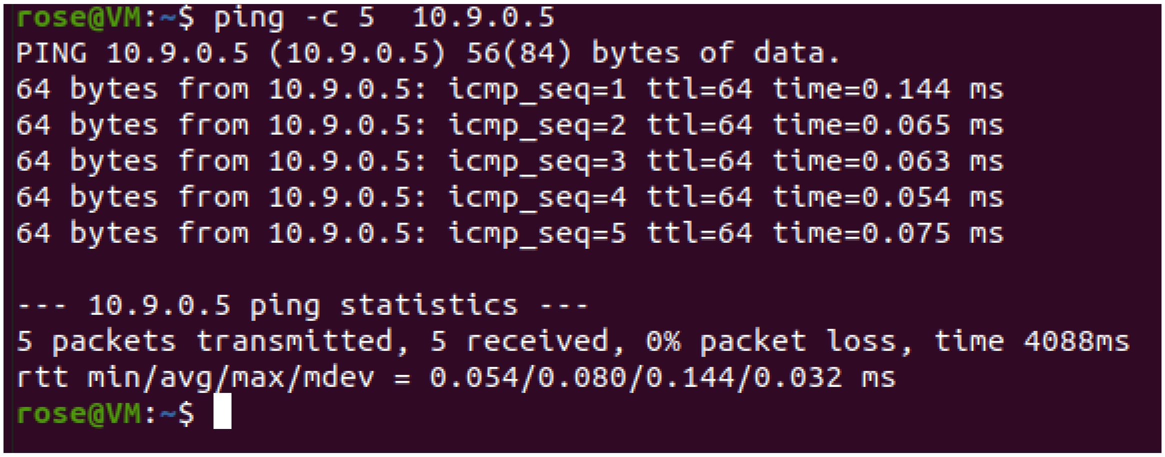
```
br-3045cf60fe03: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:b7ff:fec3:141b  prefixlen 64  scopeid 0x20<link>
        ether 02:42:b7:c3:14:1b  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 43  bytes 5156 (5.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

task1.1.py Sniffing Packets

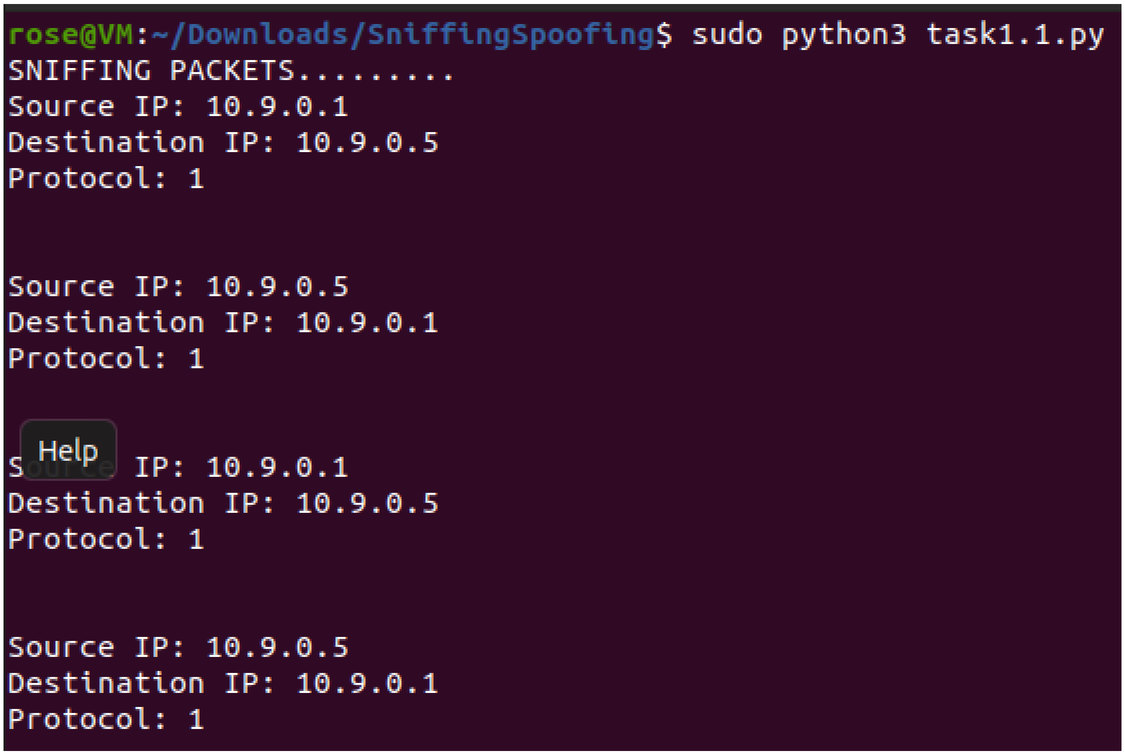
I set up the first py file. I couldn’t find anything in the volumes folder or the docker so this is the only task I’m aware of thus far. I start up the file in the terminal. Then I start pinging to make some network traffic from the 10.9.0.5 IP. After scraping through the web I eventually found the other tasks.



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

print("SNIFFING PACKETS.........")

def print_pkt(pkt):
	print("Source IP:", pkt[IP].src)
	print("Destination IP:", pkt[IP].dst)
	print("Protocol:", pkt[IP].proto)
	print("\n")

pkt = sniff(iface='br-3045cf60fe03', filter='ip',prn=print_pkt)
```



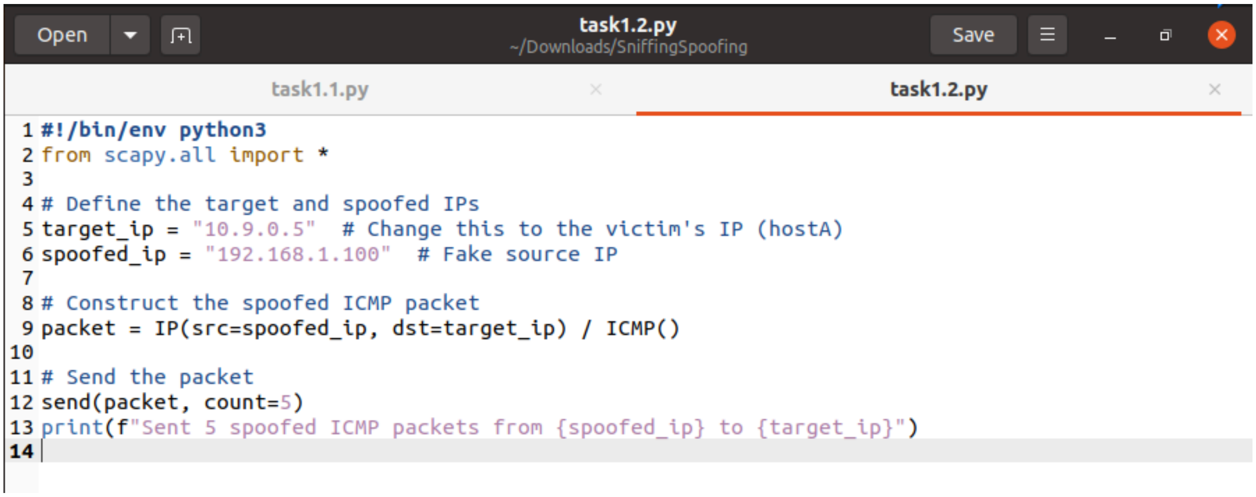
```
rose@VM:~$ ping -c 5  10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.144 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.065 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.063 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.054 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=64 time=0.075 ms

--- 10.9.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4088ms
rtt min/avg/max/mdev = 0.054/0.080/0.144/0.032 ms
rose@VM:~$ 
```



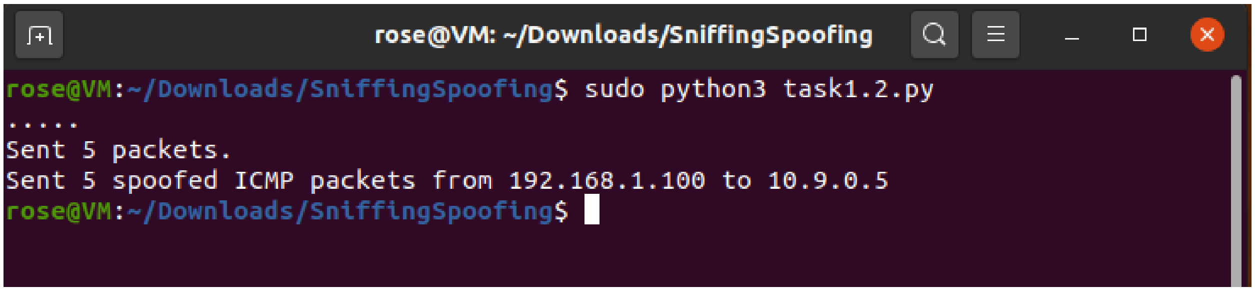
```
rose@VM:~/Downloads/SniffingSpoofing$ sudo python3 task1.1.py
SNIFFING PACKETS.........
Source IP: 10.9.0.1
Destination IP: 10.9.0.5
Protocol: 1


Source IP: 10.9.0.5
Destination IP: 10.9.0.1
Protocol: 1


Source IP: 10.9.0.1
Destination IP: 10.9.0.5
Protocol: 1


Source IP: 10.9.0.5
Destination IP: 10.9.0.1
Protocol: 1
```

task1.2.py Spoofing ICMP Packets

I set up a program for task two that spoofs ICMP packets. This program sends 5 ICMP packets from the spoofed IP to the target IP.



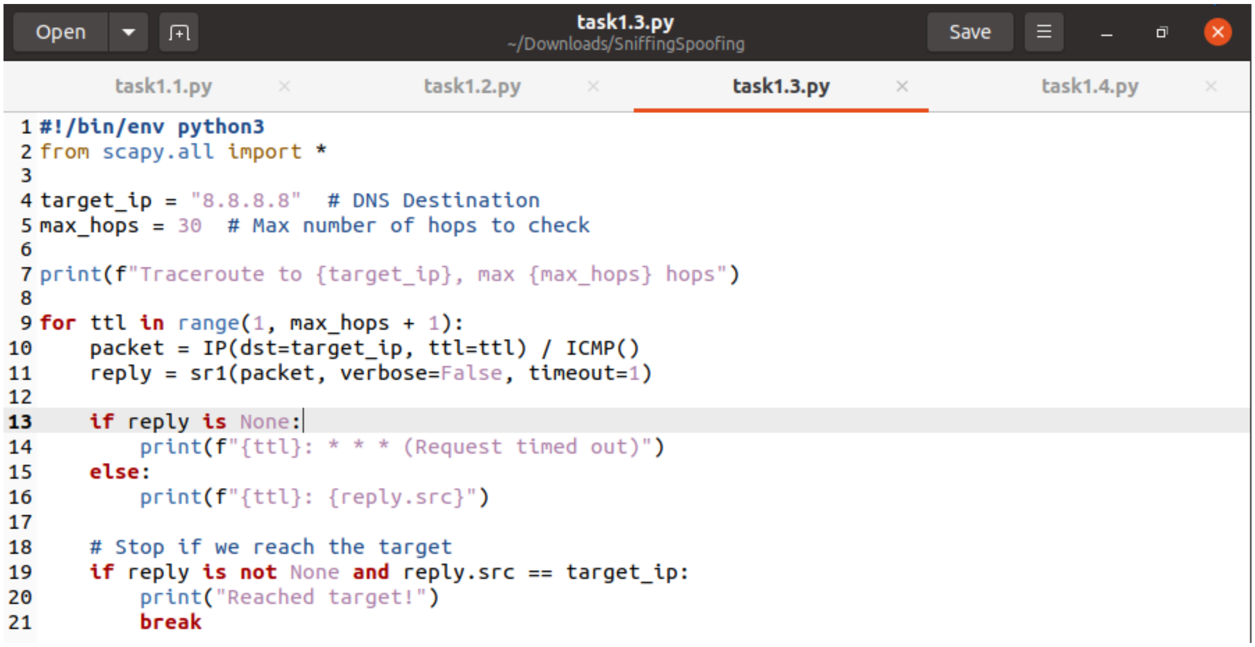
```
#!/bin/env python3
from scapy.all import *

# Define the target and spoofed IPs
target_ip = "10.9.0.5"  # Change this to the victim's IP (hostA)
spoofed_ip = "192.168.1.100"  # Fake source IP

# Construct the spoofed ICMP packet
packet = IP(src=spoofed_ip, dst=target_ip) / ICMP()

# Send the packet
send(packet, count=5)
print(f"Sent 5 spoofed ICMP packets from {spoofed_ip} to {target_ip}")

```



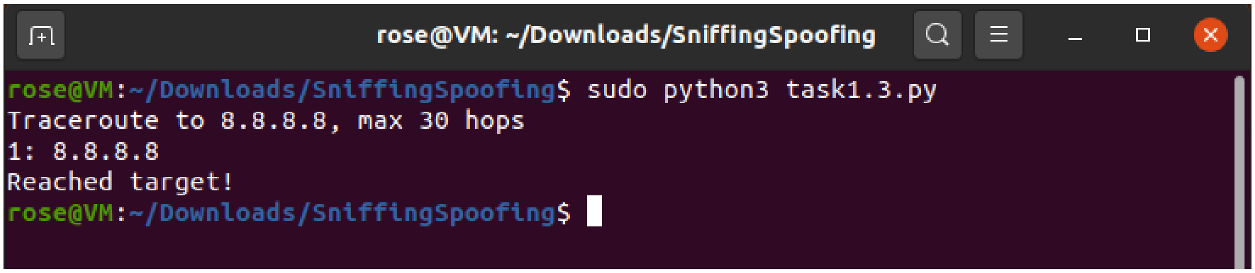
```
rose@VM:~/Downloads/SniffingSpoofing$ sudo python3 task1.2.py
.....
Sent 5 packets.
Sent 5 spoofed ICMP packets from 192.168.1.100 to 10.9.0.5
rose@VM:~/Downloads/SniffingSpoofing$ 
```

task1.3.py Implementing Traceroute

I set up a program for task two that implements traceroute. This program does a maximum of 30 hops to the target domain: 8.8.8.8. As we can see it is successful based on the terminal output. I tracked the network traffic from source to destination.



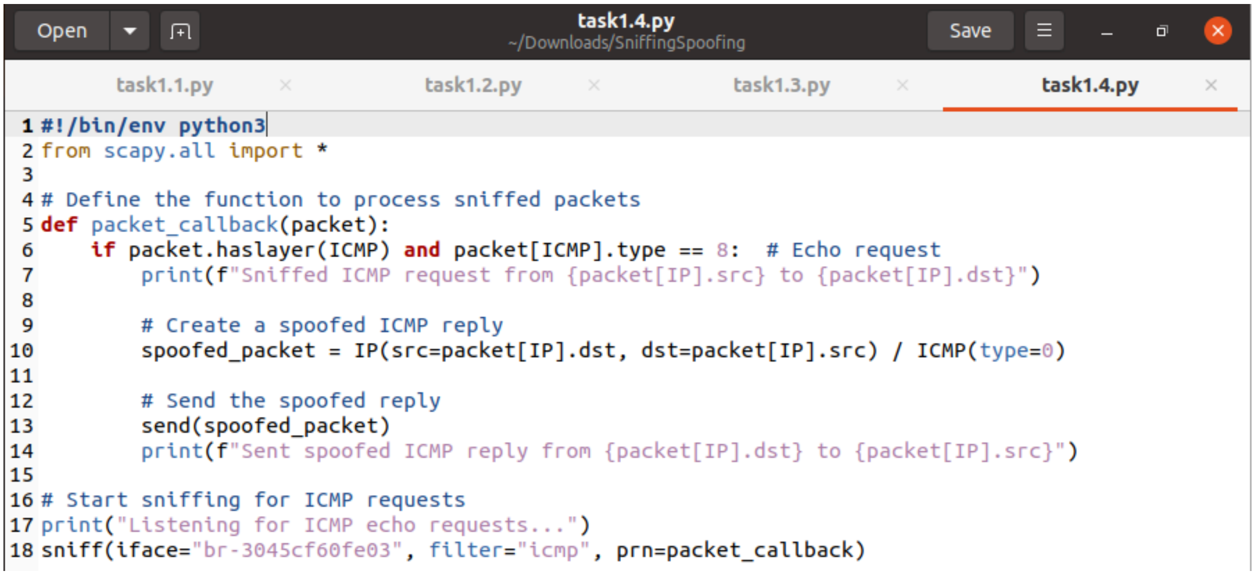
```python
#!/bin/env python3
from scapy.all import *

target_ip = "8.8.8.8"  # DNS Destination
max_hops = 30  # Max number of hops to check

print(f"Traceroute to {target_ip}, max {max_hops} hops")

for ttl in range(1, max_hops + 1):
    packet = IP(dst=target_ip, ttl=ttl) / ICMP()
    reply = sr1(packet, verbose=False, timeout=1)

    if reply is None:
        print(f"{ttl}: * * * (Request timed out)")
    else:
        print(f"{ttl}: {reply.src}")

    # Stop if we reach the target
    if reply is not None and reply.src == target_ip:
        print("Reached target!")
        break
```



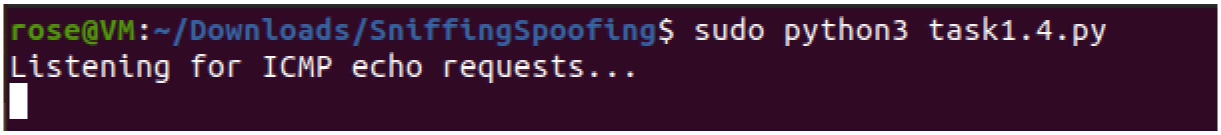
```
rose@VM:~/Downloads/SniffingSpoofing$ sudo python3 task1.3.py
Traceroute to 8.8.8.8, max 30 hops
1: 8.8.8.8
Reached target!
rose@VM:~/Downloads/SniffingSpoofing$ 
```

task1.4.py Sniffing And Spoofing

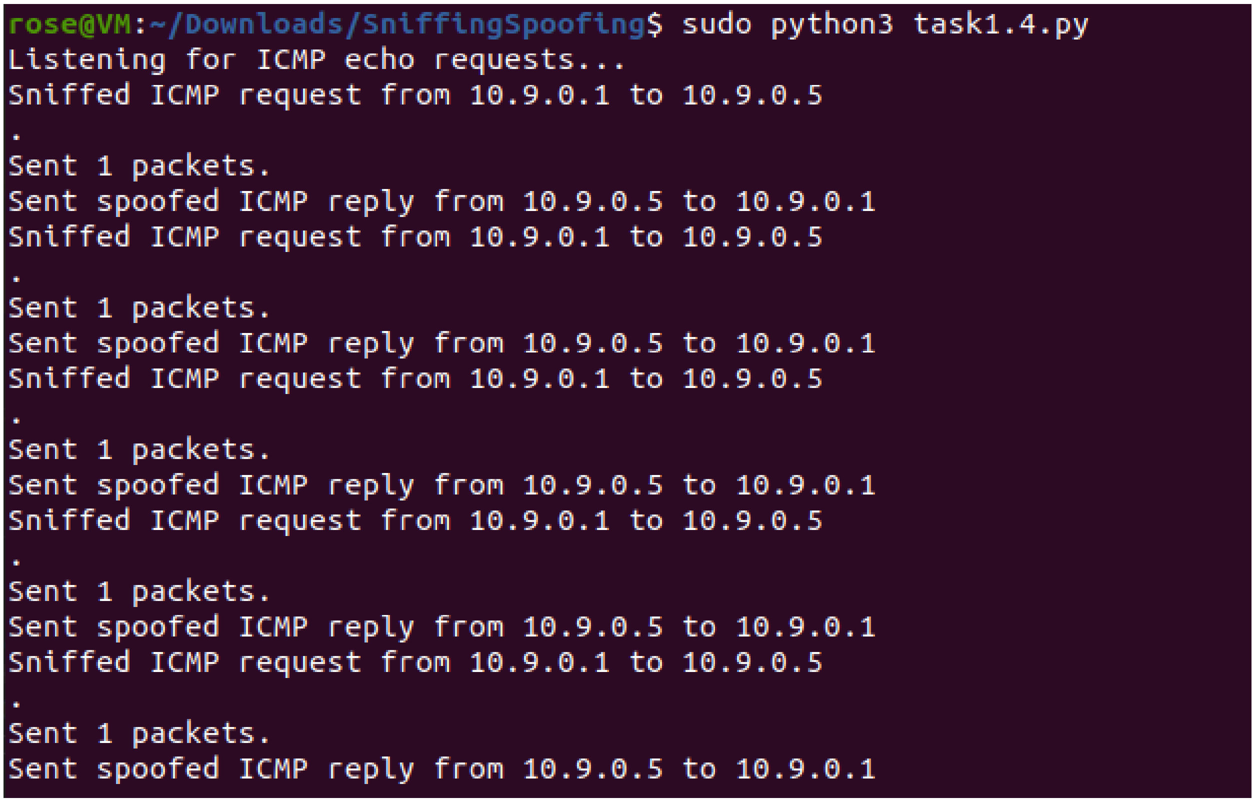
I set up a program for task two that implements both sniffing and spoofing. It checks for network traffic to our Bridge ID to host IP and automatically spoofs those packages. So we'll create some network traffic like we did for task1.1.py



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

# Define the function to process sniffed packets
def packet_callback(packet):
    if packet.haslayer(ICMP) and packet[ICMP].type == 8:  # Echo request
        print(f"Sniffed ICMP request from {packet[IP].src} to {packet[IP].dst}")

        # Create a spoofed ICMP reply
        spoofed_packet = IP(src=packet[IP].dst, dst=packet[IP].src) / ICMP(type=0)

        # Send the spoofed reply
        send(spoofed_packet)
        print(f"Sent spoofed ICMP reply from {packet[IP].dst} to {packet[IP].src}")

# Start sniffing for ICMP requests
print("Listening for ICMP echo requests...")
sniff(iface="br-3045cf60fe03", filter="icmp", prn=packet_callback)
```



```
rose@VM:~/Downloads/SniffingSpoofing$ sudo python3 task1.4.py
Listening for ICMP echo requests...
```



```
rose@VM:~/Downloads/SniffingSpoofing$ sudo python3 task1.4.py
Listening for ICMP echo requests...
Sniffed ICMP request from 10.9.0.1 to 10.9.0.5
.
Sent 1 packets.
Sent spoofed ICMP reply from 10.9.0.5 to 10.9.0.1
Sniffed ICMP request from 10.9.0.1 to 10.9.0.5
.
Sent 1 packets.
Sent spoofed ICMP reply from 10.9.0.5 to 10.9.0.1
Sniffed ICMP request from 10.9.0.1 to 10.9.0.5
.
Sent 1 packets.
Sent spoofed ICMP reply from 10.9.0.5 to 10.9.0.1
Sniffed ICMP request from 10.9.0.1 to 10.9.0.5
.
Sent 1 packets.
Sent spoofed ICMP reply from 10.9.0.5 to 10.9.0.1
Sniffed ICMP request from 10.9.0.1 to 10.9.0.5
.
Sent 1 packets.
Sent spoofed ICMP reply from 10.9.0.5 to 10.9.0.1
```

In this lab I learned about sniffing and spoofing ICMP packets. This allows me to sniff any network packages sent between IPs through my Bridge ID Host IP. Eventually by task 1.4 I learned how to automatically spoof the network traffic I sniffed. This sort of information is pretty relevant to my project so I see some value in it.