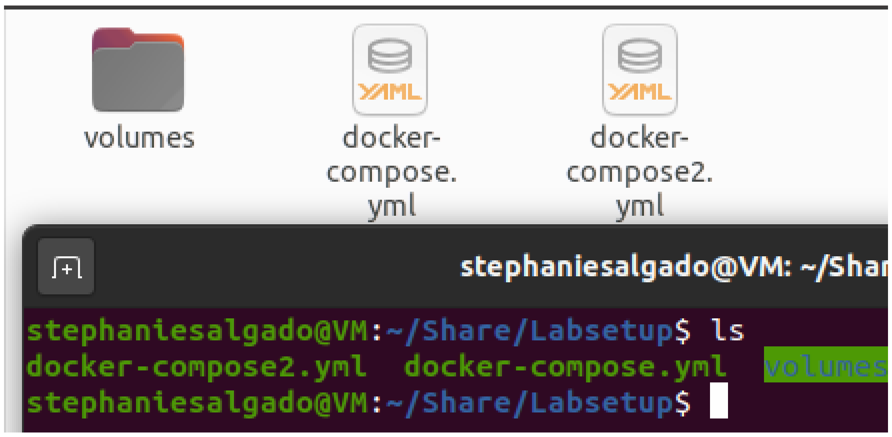
Computer Security

Lab 9 Demo: VPN Lab - The Container Version

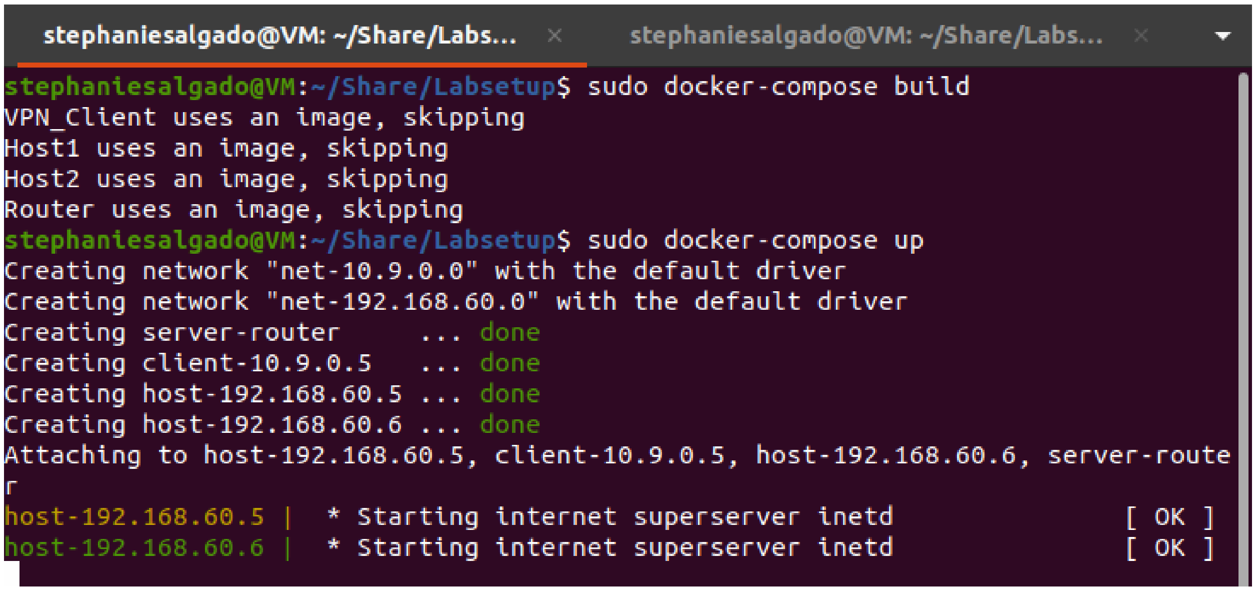
Stephanie Salgado

Set up:



```
stephaniesalgado@VM:~/Share/Labsetup$ ls
docker-compose2.yml  docker-compose.yml  volumes
stephaniesalgado@VM:~/Share/Labsetup$
```

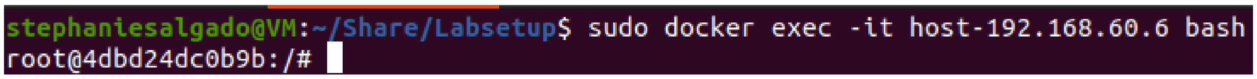
Located lab set up files.


stephaniesalgado@VM: ~/Share/Labs...
stephaniesalgado@VM: ~/Share/Labs...

```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker-compose build
VPN_Client uses an image, skipping
Host1 uses an image, skipping
Host2 uses an image, skipping
Router uses an image, skipping
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating network "net-192.168.60.0" with the default driver
Creating server-router      ... done
Creating client-10.9.0.5    ... done
Creating host-192.168.60.5 ... done
Creating host-192.168.60.6 ... done
Attaching to host-192.168.60.5, client-10.9.0.5, host-192.168.60.6, server-route
r
host-192.168.60.5 |  * Starting internet superserver inetd                [ OK ]
host-192.168.60.6 |  * Starting internet superserver inetd                [ OK ]
```

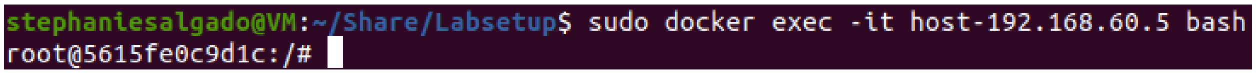
I started the containers.

host-192.168.60.6:



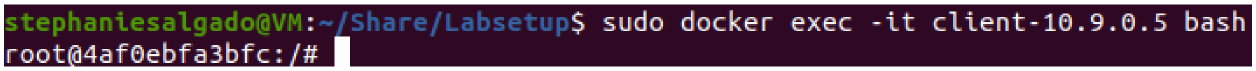
```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker exec -it host-192.168.60.6 bash
root@4dbd24dc0b9b:/#
```

host-192.168.60.5:



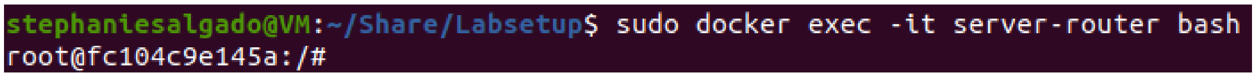
```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker exec -it host-192.168.60.5 bash
root@5615fe0c9d1c:/#
```

client-10.9.0.5:



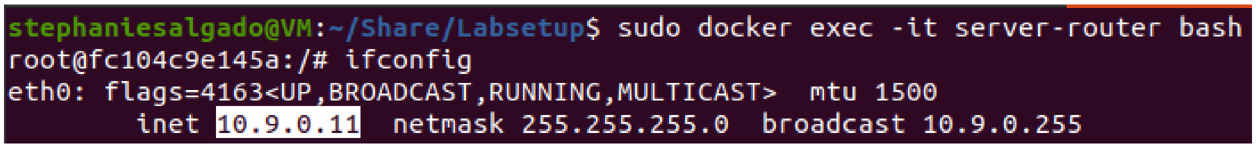
```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker exec -it client-10.9.0.5 bash
root@4af0ebfa3bfc:/#
```

server-router:



```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker exec -it server-router bash
root@fc104c9e145a:/#
```

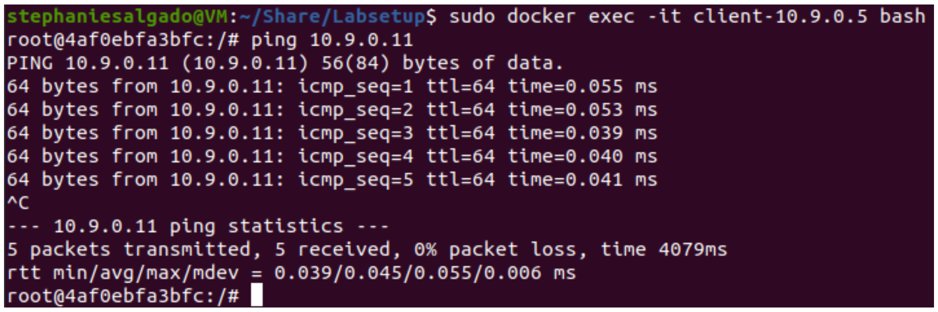
Then, I start a shell on each.



```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker exec -it server-router bash
root@fc104c9e145a:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.11  netmask 255.255.255.0  broadcast 10.9.0.255
```

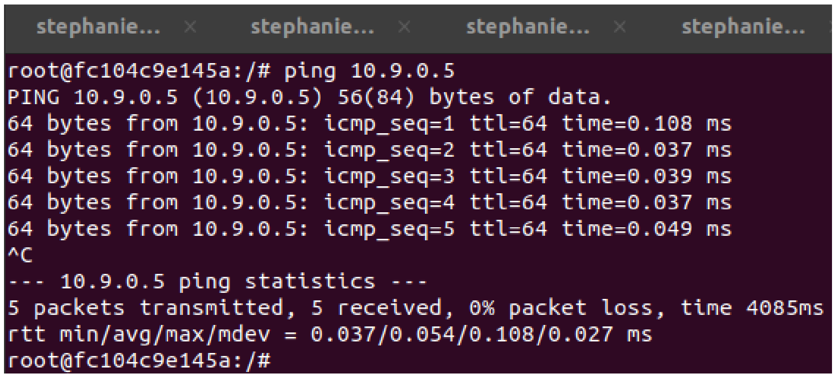
Finally, I use “ifconfig” from “server-router” to find its IP address.

Task 1: Network Setup



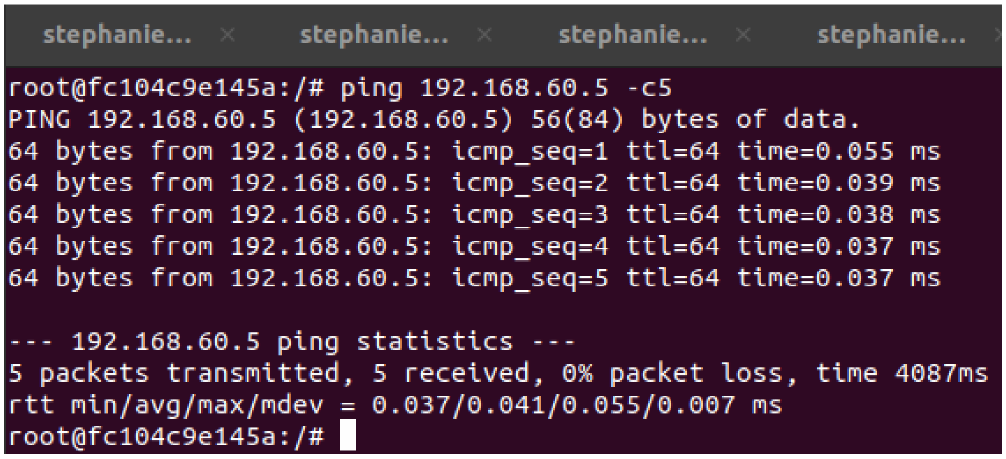
```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker exec -it client-10.9.0.5 bash
root@4af0ebfa3bfc:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.055 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.053 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.039 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.040 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.041 ms
^C
--- 10.9.0.11 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4079ms
rtt min/avg/max/mdev = 0.039/0.045/0.055/0.006 ms
root@4af0ebfa3bfc:/#
```

From “client-10.9.0.5”, I ping “server-router” using the command “ping 10.9.0.11”. This successfully transmitted 5 packets before I stopped the ping.



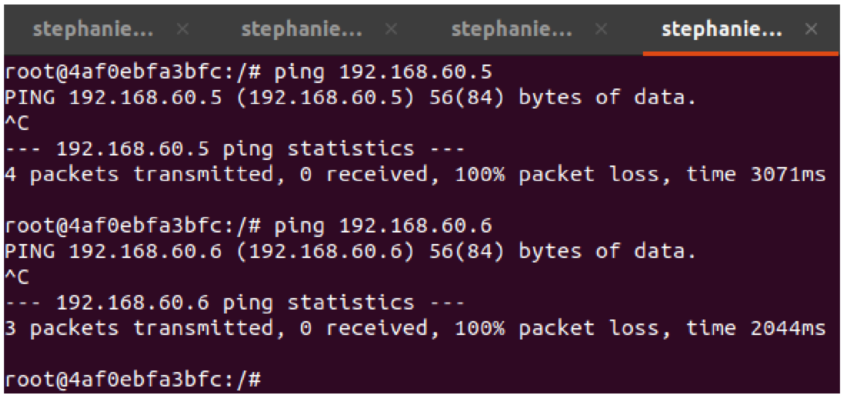
```
root@fc104c9e145a:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.108 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.037 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.039 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.037 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=64 time=0.049 ms
^C
--- 10.9.0.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4085ms
rtt min/avg/max/mdev = 0.037/0.054/0.108/0.027 ms
root@fc104c9e145a:/#
```

Then, from “server-router”, I ping “client-10.9.0.5”; once again transmitting 5 packets.



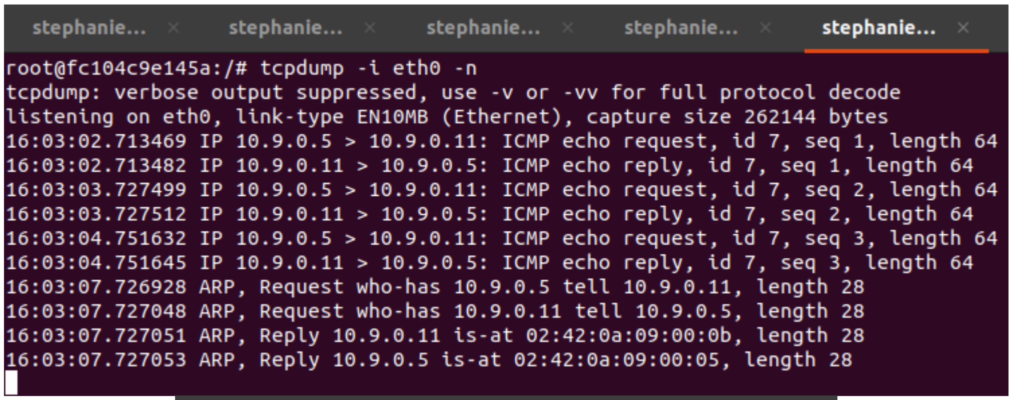
```
root@fc104c9e145a:/# ping 192.168.60.5 -c5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=64 time=0.055 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=64 time=0.039 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=64 time=0.038 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=64 time=0.037 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=64 time=0.037 ms

--- 192.168.60.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4087ms
rtt min/avg/max/mdev = 0.037/0.041/0.055/0.007 ms
root@fc104c9e145a:/#
```

Similarly, I use “ping 192.168.60.5” to ping “host-192.168.60.5” from “server-router”.

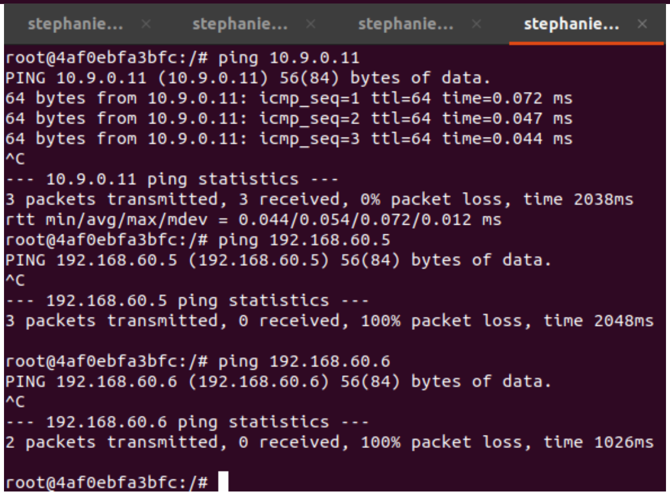


```
root@4af0ebfa3bfc:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3071ms

root@4af0ebfa3bfc:/# ping 192.168.60.6
PING 192.168.60.6 (192.168.60.6) 56(84) bytes of data.
^C
--- 192.168.60.6 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2044ms

root@4af0ebfa3bfc:/#
```

I go to “client” and ping both hosts. Notice there were 0 received packets and 100% packet loss, meaning the client is unable to ping either host.



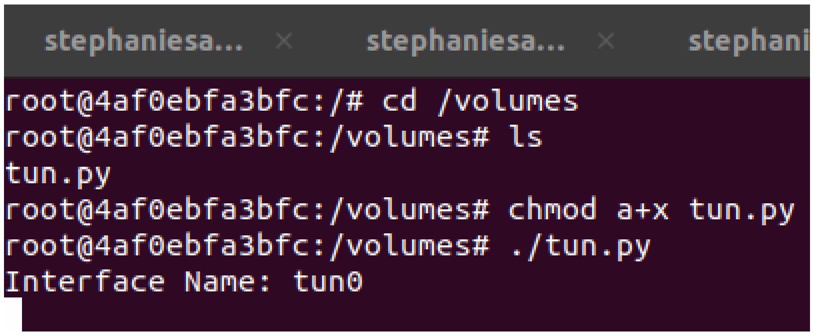
```
root@fc104c9e145a:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
16:03:02.713469 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 7, seq 1, length 64
16:03:02.713482 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 7, seq 1, length 64
16:03:03.727499 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 7, seq 2, length 64
16:03:03.727512 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 7, seq 2, length 64
16:03:04.751632 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 7, seq 3, length 64
16:03:04.751645 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 7, seq 3, length 64
16:03:07.726928 ARP, Request who-has 10.9.0.5 tell 10.9.0.11, length 28
16:03:07.727048 ARP, Request who-has 10.9.0.11 tell 10.9.0.5, length 28
16:03:07.727051 ARP, Reply 10.9.0.11 is-at 02:42:0a:09:00:0b, length 28
16:03:07.727053 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, length 28
```



```
root@4af0ebfa3bfc:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.072 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.047 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.044 ms
^C
--- 10.9.0.11 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2038ms
rtt min/avg/max/mdev = 0.044/0.054/0.072/0.012 ms
root@4af0ebfa3bfc:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2048ms

root@4af0ebfa3bfc:/# ping 192.168.60.6
PING 192.168.60.6 (192.168.60.6) 56(84) bytes of data.
^C
--- 192.168.60.6 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1026ms

root@4af0ebfa3bfc:/#
```

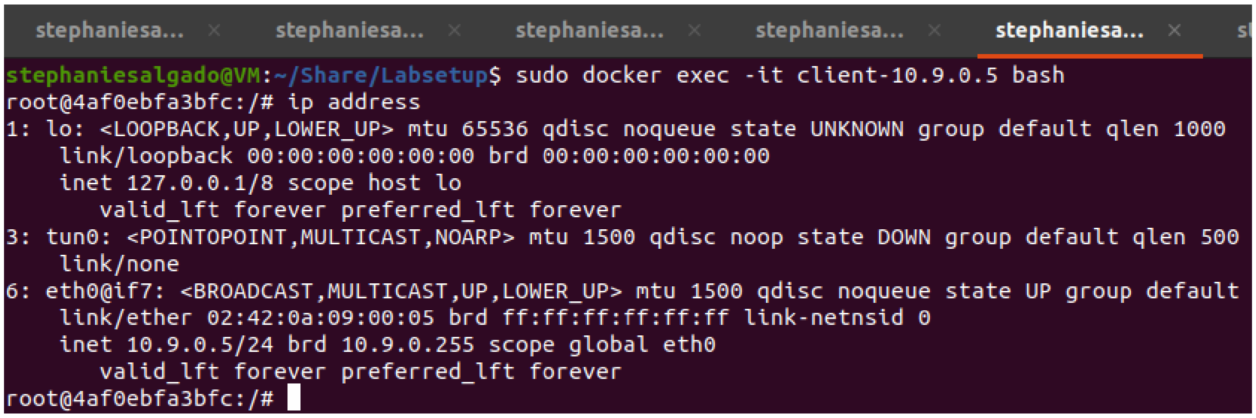
The command “tcpdump -i eth0 -n” from “server-router” can be used to listen on eth0. This once again proved that the client is able to communicate with the server, but not “host-192.168.60.5” or “host-192.168.60.6”.

Task 2: Create and Configure TUN Interface



```
root@4af0ebfa3bfc:/# cd /volumes
root@4af0ebfa3bfc:/volumes# ls
tun.py
root@4af0ebfa3bfc:/volumes# chmod a+x tun.py
root@4af0ebfa3bfc:/volumes# ./tun.py
Interface Name: tun0
```

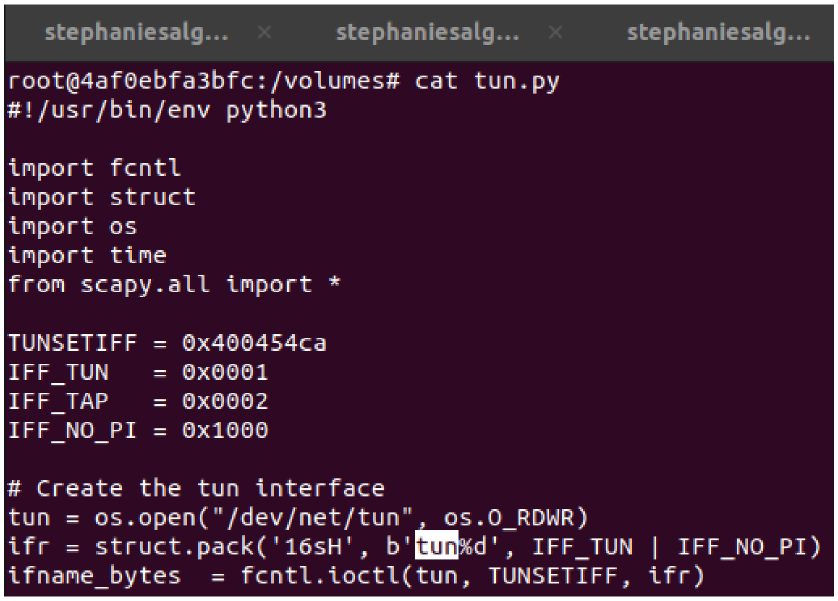
I navigate to the “volumes” directory then use “ls” to check if the provided “tun.py” file was there. Once I saw it was there, I used “chmod a+x tun.py” to make it executable and then ran the program using “./tun.py”. The program printed the default interface name “tun0” and then continued running.



```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker exec -it client-10.9.0.5 bash
root@4af0ebfa3bfc:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
6: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
root@4af0ebfa3bfc:/#
```

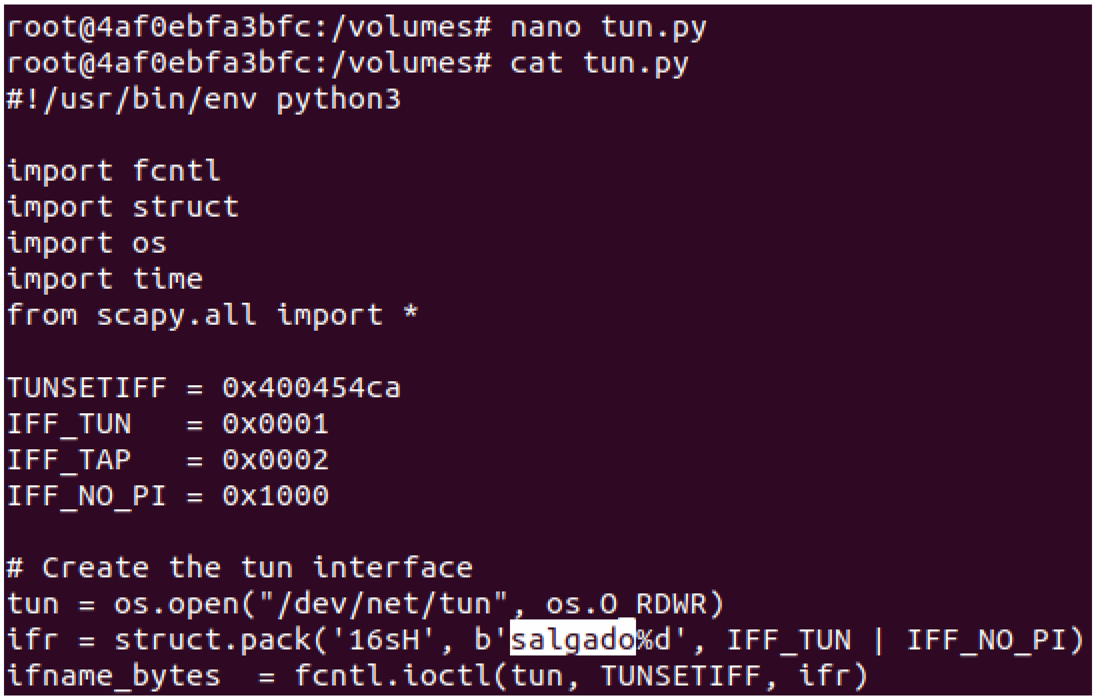
I started a shell on “client” in a different tab and then used “ip address”. There, I was able to find the new “tun0” interface that the program created.

Task 2.a: Name of the Interface



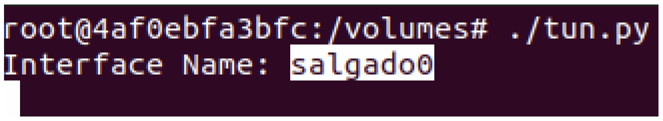
```
root@4af0ebfa3bfc:/volumes# cat tun.py
#!/usr/bin/env python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)
```

I inspect "tun.py" using "cat" then notice I can easily change the name of the interface by modifying the program.

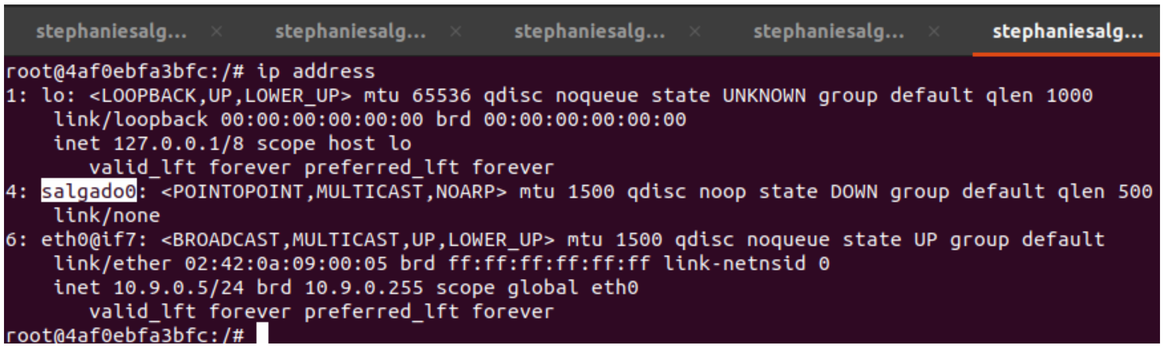


```
root@4af0ebfa3bfc:/volumes# nano tun.py
root@4af0ebfa3bfc:/volumes# cat tun.py
#!/usr/bin/env python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'salgado%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)
```

The command "nano tun.py" allows me to change "tun" to "salgado" (my last name).



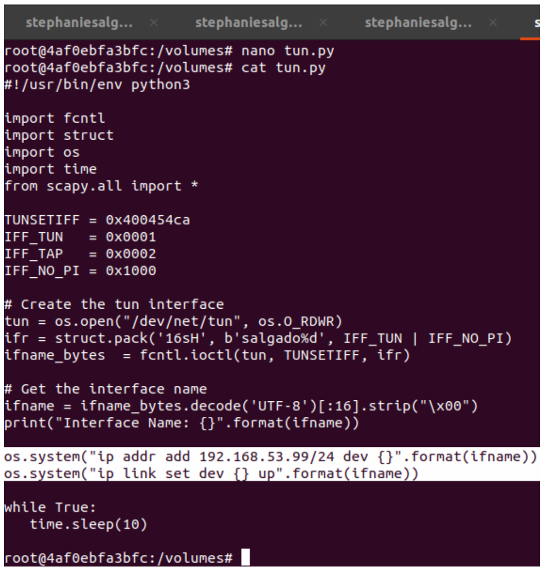
```
root@4af0ebfa3bfc:/volumes# ./tun.py
Interface Name: salgado0
```



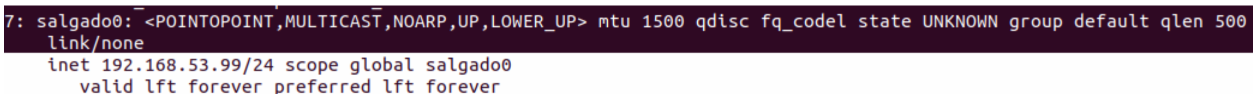
```
root@4af0ebfa3bfc:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
4: salgado0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
6: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:09:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.9.0.5/24 brd 10.9.0.255 scope global eth0
       valid_lft forever preferred_lft forever
root@4af0ebfa3bfc:/#
```

I run the program and this time after checking the interface name, it is now my last name.

Task 2.b: Set up the TUN Interface



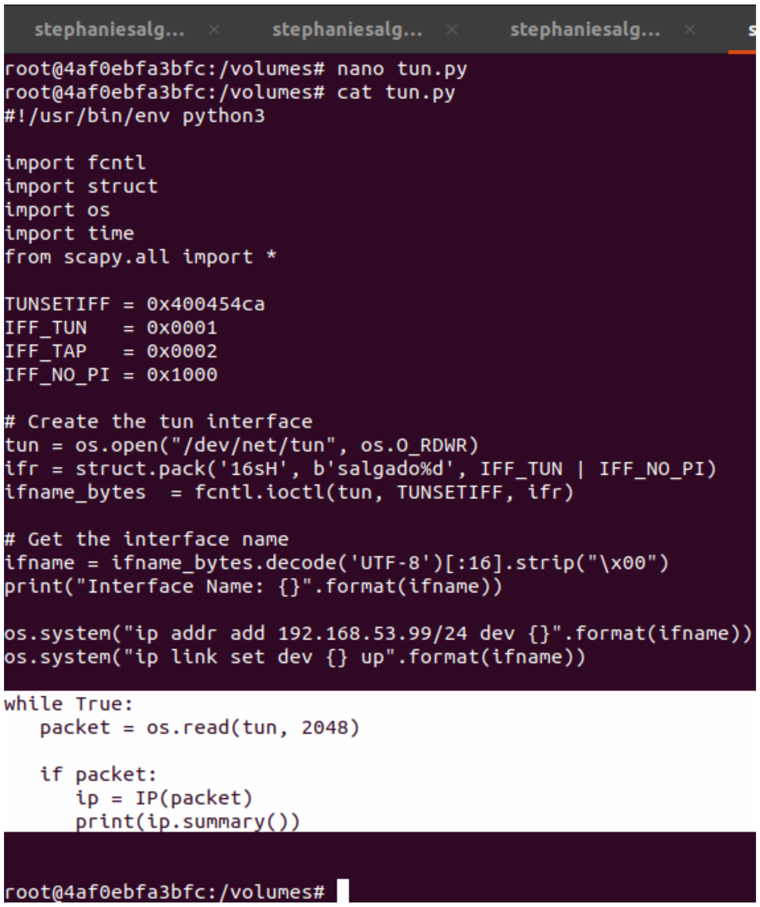
```
root@4af0ebfa3bfc:/volumes# nano tun.py
root@4af0ebfa3bfc:/volumes# cat tun.py
#!/usr/bin/env python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'salgado%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

while True:
   time.sleep(10)

root@4af0ebfa3bfc:/volumes#
```



```
7: salgado0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.99/24 scope global salgado0
       valid_lft forever preferred_lft forever
```

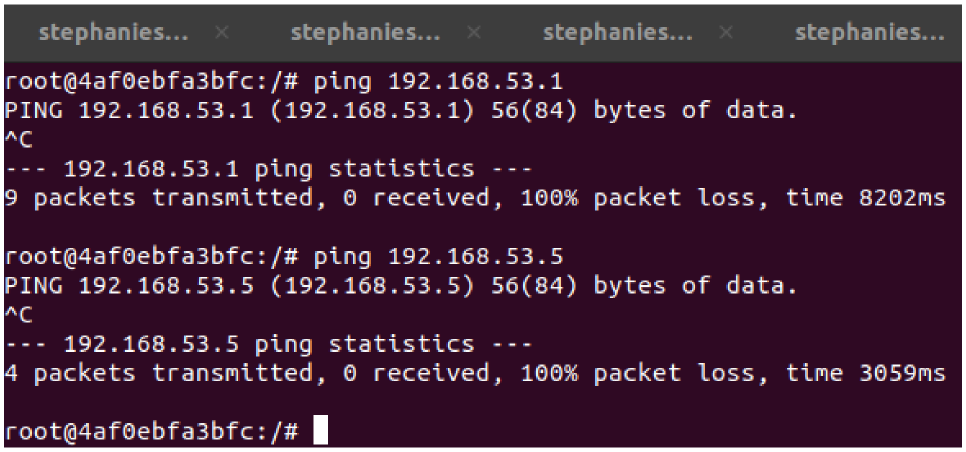
I modify the code and add the highlighted lines in order to automatically assign an IP address to it and to bring up the interface every time the program runs. This is reflected by the highlighted lines when I run “ip addr”.

Task 2.c: Read from the TUN Interface



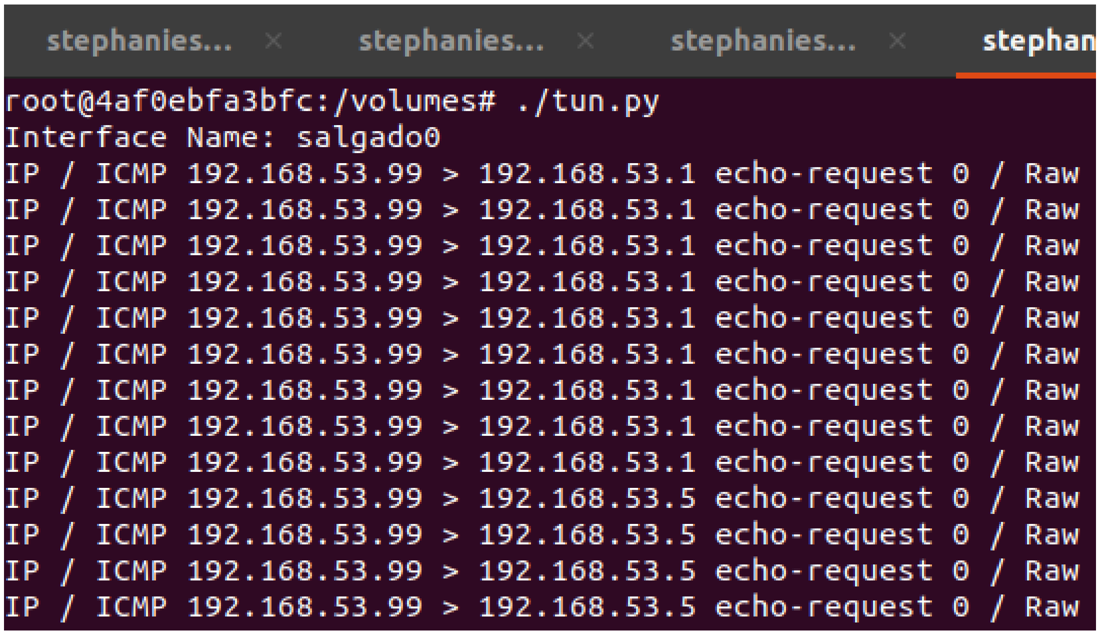
```
root@4af0ebfa3bfc:/volumes# nano tun.py
root@4af0ebfa3bfc:/volumes# cat tun.py
#!/usr/bin/env python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'salgado%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

while True:
   packet = os.read(tun, 2048)

   if packet:
      ip = IP(packet)
      print(ip.summary())

root@4af0ebfa3bfc:/volumes# 
```

I use “nano tun.py” to modify the code, changing the while loop to the highlighted text.



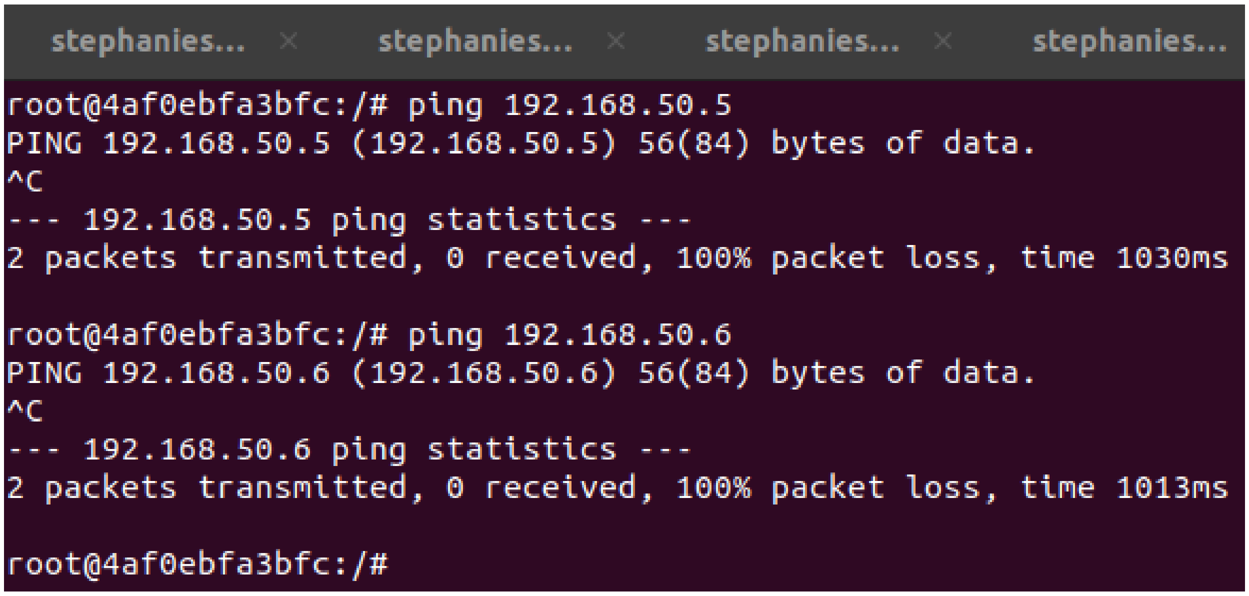
```
root@4af0ebfa3bfc:/# ping 192.168.53.1
PING 192.168.53.1 (192.168.53.1) 56(84) bytes of data.
^C
--- 192.168.53.1 ping statistics ---
9 packets transmitted, 0 received, 100% packet loss, time 8202ms

root@4af0ebfa3bfc:/# ping 192.168.53.5
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
^C
--- 192.168.53.5 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3059ms

root@4af0ebfa3bfc:/# 
```

I ping two IP addresses in the 192.168.53.0/24 network.



```
root@4af0ebfa3bfc:/volumes# ./tun.py
Interface Name: salgado0
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.1 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
```

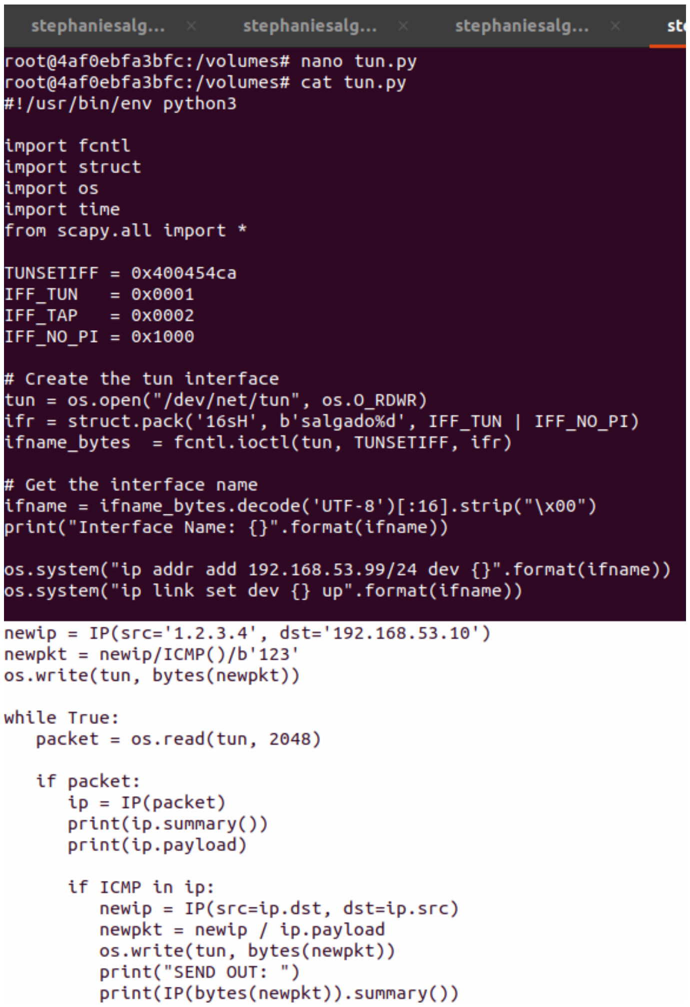
The program prints the captured ICMP echo requests.



```
root@4af0ebfa3bfc:/# ping 192.168.50.5
PING 192.168.50.5 (192.168.50.5) 56(84) bytes of data.
^C
--- 192.168.50.5 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1030ms

root@4af0ebfa3bfc:/# ping 192.168.50.6
PING 192.168.50.6 (192.168.50.6) 56(84) bytes of data.
^C
--- 192.168.50.6 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1013ms

root@4af0ebfa3bfc:/#
```

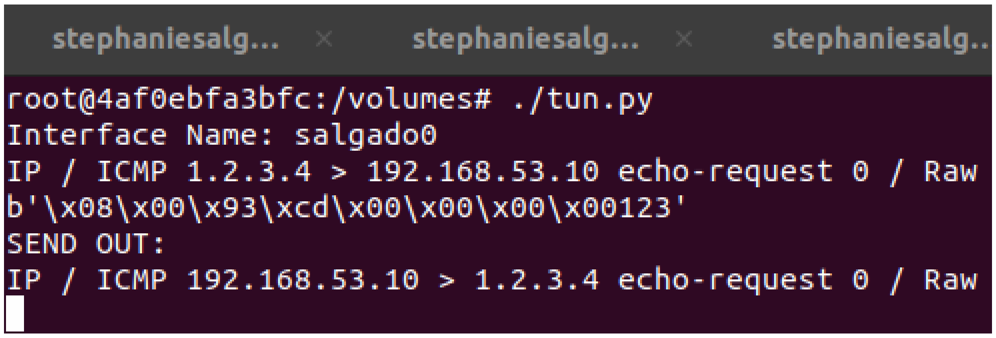
I ping both hosts and there is 100% packet loss again, when I checked the output of "tun.py", there were no changes.

Task 2.d: Write to the TUN Interface



```
root@4af0ebfa3bfc:/volumes# nano tun.py
root@4af0ebfa3bfc:/volumes# cat tun.py
#!/usr/bin/env python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'salgado%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
newip = IP(src='1.2.3.4', dst='192.168.53.10')
newpkt = newip/ICMP()/b'123'
os.write(tun, bytes(newpkt))

while True:
   packet = os.read(tun, 2048)

   if packet:
      ip = IP(packet)
      print(ip.summary())
      print(ip.payload)

      if ICMP in ip:
         newip = IP(src=ip.dst, dst=ip.src)
         newpkt = newip / ip.payload
         os.write(tun, bytes(newpkt))
         print("SEND OUT: ")
         print(IP(bytes(newpkt)).summary())
```

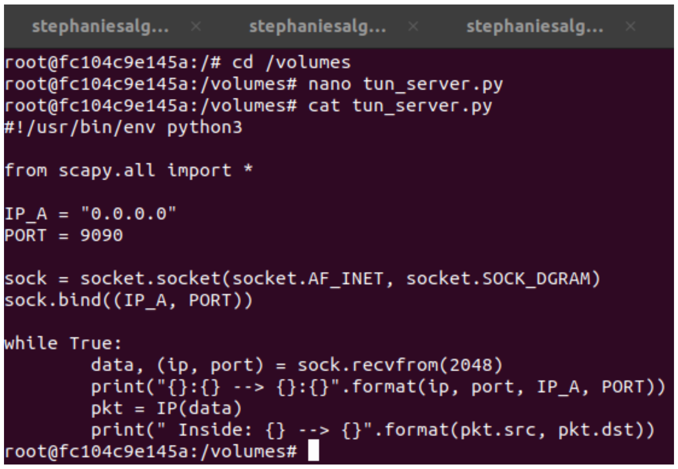
I modified “tun.py” once again.



```
root@4af0ebfa3bfc:/volumes# ./tun.py
Interface Name: salgado0
IP / ICMP 1.2.3.4 > 192.168.53.10 echo-request 0 / Raw
b'\x08\x00\x93\xcd\x00\x00\x00\x00123'
SEND OUT:
IP / ICMP 192.168.53.10 > 1.2.3.4 echo-request 0 / Raw
```

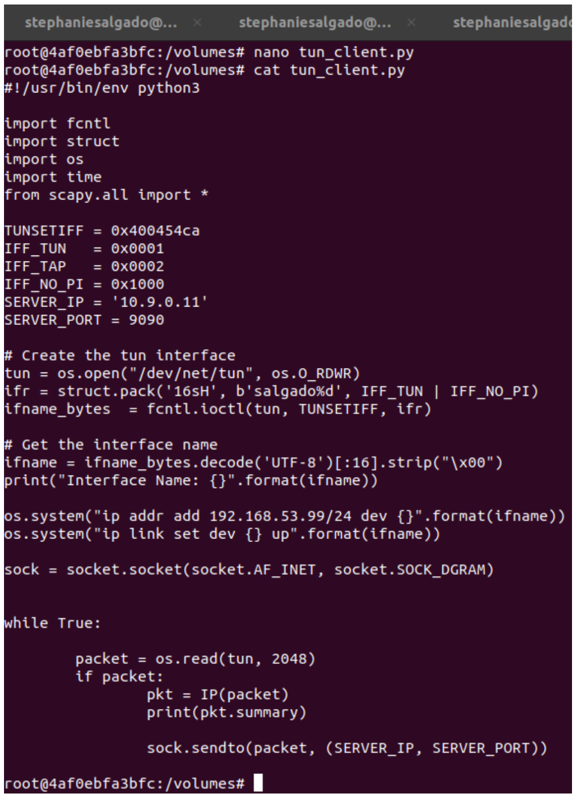
This was the output.

Task 3: Send the IP Packet to VPN Server Through a Tunnel



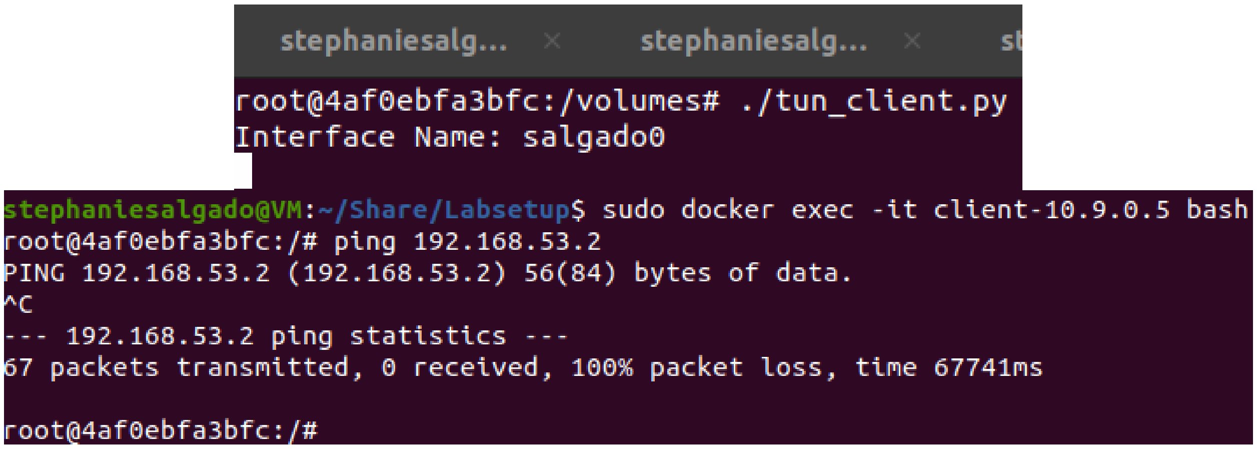
```
root@fc104c9e145a:/# cd /volumes
root@fc104c9e145a:/volumes# nano tun_server.py
root@fc104c9e145a:/volumes# cat tun_server.py
#!/usr/bin/env python3

from scapy.all import *

IP_A = "0.0.0.0"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

while True:
        data, (ip, port) = sock.recvfrom(2048)
        print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
        pkt = IP(data)
        print(" Inside: {} --> {}".format(pkt.src, pkt.dst))
root@fc104c9e145a:/volumes#
```

From “server-router” I navigate to volumes then use “nano tun_server.py” to create the “tun_server.py” file. Then I add the contents provided in the lab documentation.



```
root@4af0ebfa3bfc:/volumes# nano tun_client.py
root@4af0ebfa3bfc:/volumes# cat tun_client.py
#!/usr/bin/env python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000
SERVER_IP = '10.9.0.11'
SERVER_PORT = 9090

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'salgado%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)


while True:

        packet = os.read(tun, 2048)
        if packet:
                pkt = IP(packet)
                print(pkt.summary)

                sock.sendto(packet, (SERVER_IP, SERVER_PORT))

root@4af0ebfa3bfc:/volumes#
```

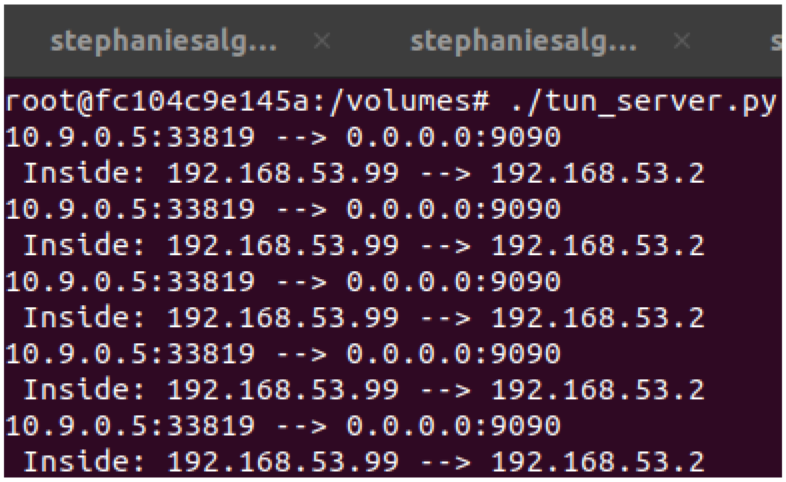
I go back to “client” and create “tun_client.py” and add the contents from “tun.py” but modify them.



```
root@4af0ebfa3bfc:/volumes# ./tun_client.py
Interface Name: salgado0
```

```
stephaniesalgado@VM:~/Share/Labsetup$ sudo docker exec -it client-10.9.0.5 bash
root@4af0ebfa3bfc:/# ping 192.168.53.2
PING 192.168.53.2 (192.168.53.2) 56(84) bytes of data.
^C
--- 192.168.53.2 ping statistics ---
67 packets transmitted, 0 received, 100% packet loss, time 67741ms

root@4af0ebfa3bfc:/#
```

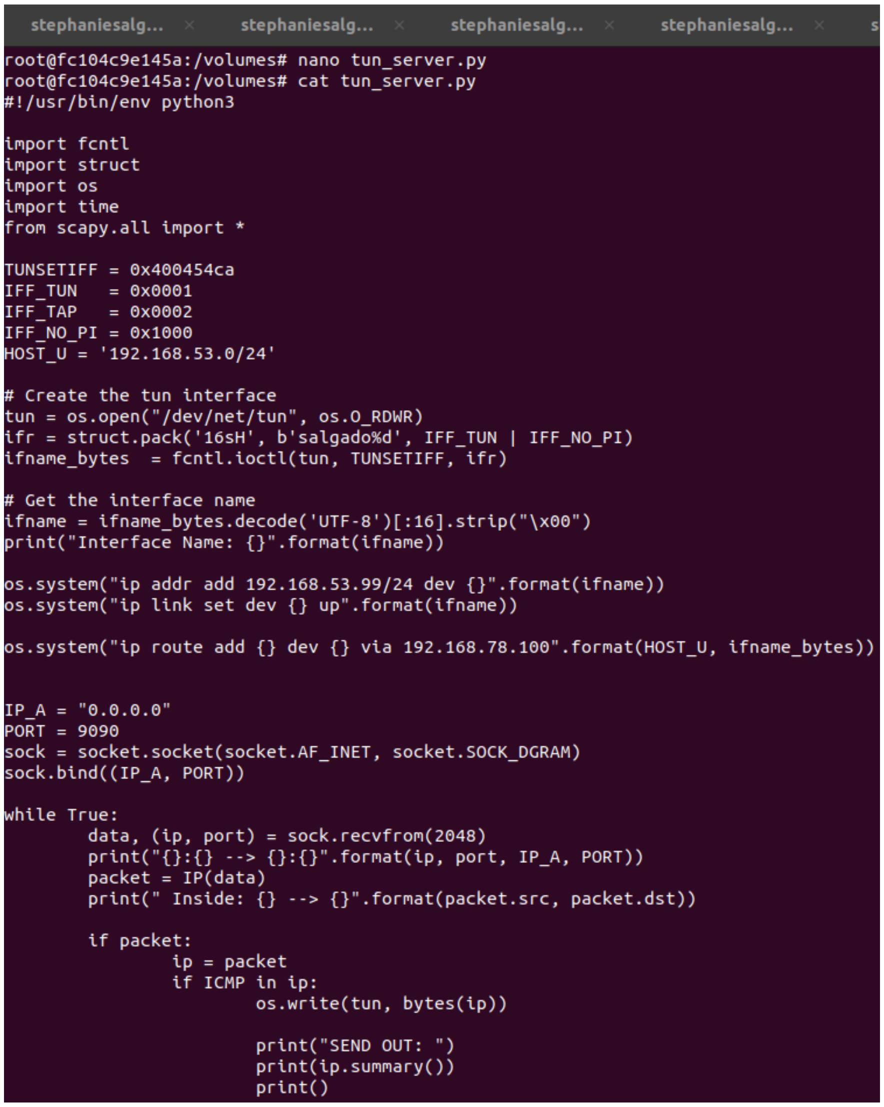
I run both “tun_client.py” and “tun_server.py” while pinging “193.168.53.2”.



```
root@fc104c9e145a:/volumes# ./tun_server.py
10.9.0.5:33819 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:33819 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:33819 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:33819 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
10.9.0.5:33819 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.2
```

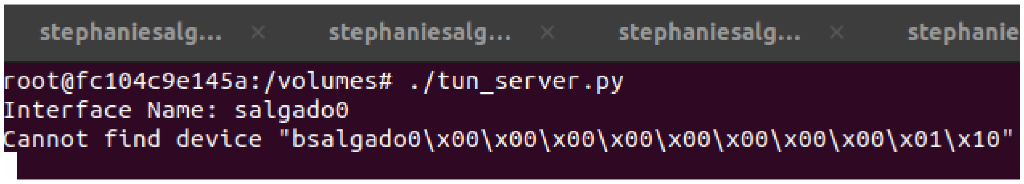
Here is what the “tun_server.py” file outputs.

Task 4: Set Up the VPN Server



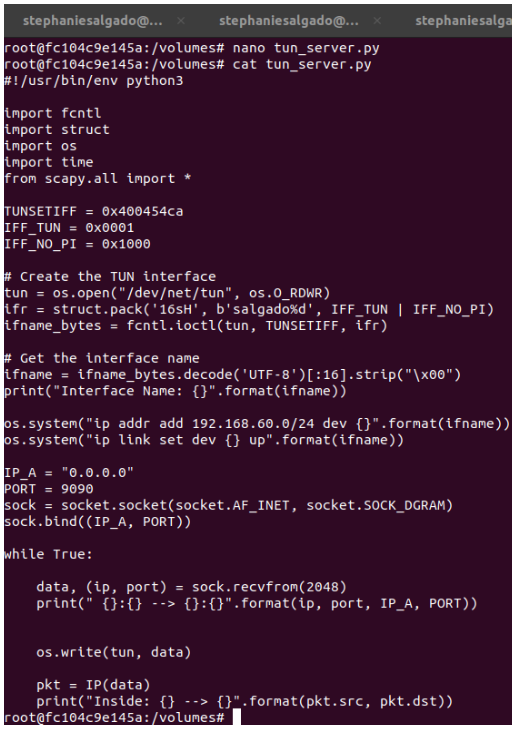
```
root@fc104c9e145a:/volumes# nano tun_server.py
root@fc104c9e145a:/volumes# cat tun_server.py
#!/usr/bin/env python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN   = 0x0001
IFF_TAP   = 0x0002
IFF_NO_PI = 0x1000
HOST_U = '192.168.53.0/24'

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'salgado%d', IFF_TUN | IFF_NO_PI)
ifname_bytes  = fcntl.ioctl(tun, TUNSETIFF, ifr)

# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

os.system("ip route add {} dev {} via 192.168.78.100".format(HOST_U, ifname_bytes))


IP_A = "0.0.0.0"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

while True:
        data, (ip, port) = sock.recvfrom(2048)
        print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
        packet = IP(data)
        print(" Inside: {} --> {}".format(packet.src, packet.dst))

        if packet:
                ip = packet
                if ICMP in ip:
                        os.write(tun, bytes(ip))

                        print("SEND OUT: ")
                        print(ip.summary())
                        print()
```

I modify “tun_server.py” to include



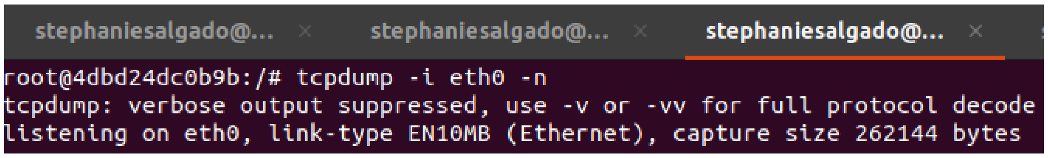
```
root@fc104c9e145a:/volumes# ./tun_server.py
Interface Name: salgado0
Cannot find device "bsalgado0\x00\x00\x00\x00\x00\x00\x00\x00\x01\x10"
```

I ran it the first time and I got this error. I decided to try a different approach.



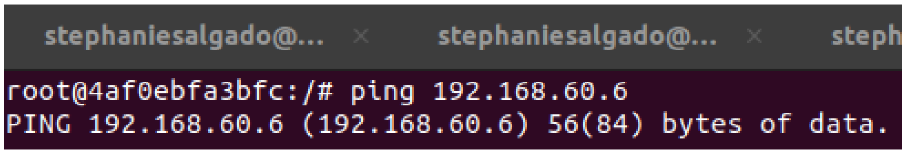
```
root@fc104c9e145a:/volumes# nano tun_server.py
root@fc104c9e145a:/volumes# cat tun_server.py
#!/usr/bin/env python3

import fcntl
import struct
import os
import time
from scapy.all import *

TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_NO_PI = 0x1000

# Create the TUN interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'salgado%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)

# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.60.0/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

IP_A = "0.0.0.0"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

while True:

    data, (ip, port) = sock.recvfrom(2048)
    print(" {}:{} --> {}:{}".format(ip, port, IP_A, PORT))


    os.write(tun, data)

    pkt = IP(data)
    print("Inside: {} --> {}".format(pkt.src, pkt.dst))
root@fc104c9e145a:/volumes# 
```

Here is a new version of the “tun_server.py” file.



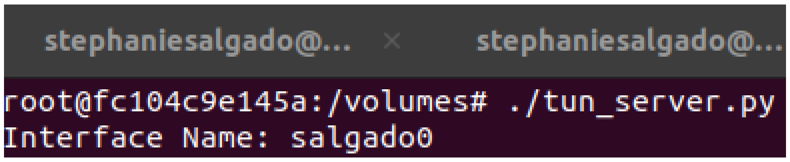
```
root@4dbd24dc0b9b:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

I typed this command in “host-192.168.60.6”.



```
root@4af0ebfa3bfc:/# ping 192.168.60.6
PING 192.168.60.6 (192.168.60.6) 56(84) bytes of data.
```

And I pinged it from “client”.



```
root@fc104c9e145a:/volumes# ./tun_server.py
Interface Name: salgado0
```

And I ran the new version, but I never got an output.

Summary:

In this lab I was able to set up the network and create the TUN interface with no problem. I only started having issues around task 4, and since I was unable to complete the task successfully, I didn’t get to move onto task 5. However, I was still able to learn more from this lab. I have enjoyed working with scapy for all of these labs, and this lab was no different, since it allowed me to learn even more about the library. This lab was even more challenging than the last. It left me somewhat frustrated because I was unable to find what was wrong with the program. I feel like I was very close to solving my issue but I had already spent an extensive amount of time working on this lab and I started to feel like I wouldn’t be able to find success. Ultimately, this lab burnt me out, but I can confidently say I tried my very best.