Task 1: (A) First, we created the python program as such:

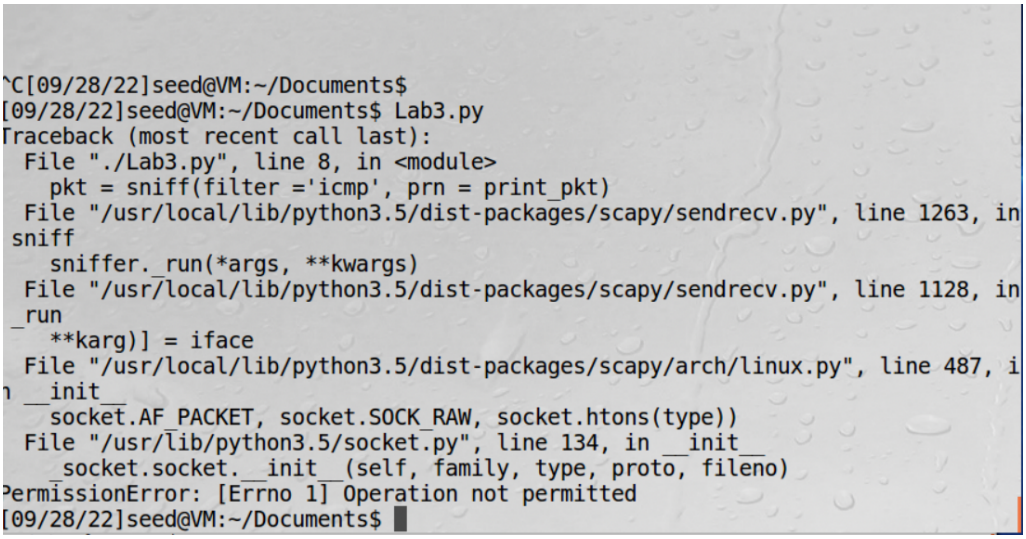
Lab3.py | sample.py

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

def print_pkt(pkt):
    pkt.show()


pkt = sniff(filter ='tcp dst port 23', prn = print_pkt)

```

Then we made the program executable by running 'chmod a+x Lab3.py'
When we ran the program with the root user permission, it started capturing packets upon opening a webpage.

```
|###[ DNS Resource Record ]###
|  rrname    = 'ns-cloud-c1.googledomains.com.'
|  type      = AAAA
|  rclass    = IN
|  ttl       = 21139
|  rdlen     = None
|  rdata     = 2001:4860:4802:32::6c
|###[ DNS Resource Record ]###
|  rrname    = 'ns-cloud-c2.googledomains.com.'
|  type      = AAAA
|  rclass    = IN
|  ttl       = 21139
|  rdlen     = None
|  rdata     = 2001:4860:4802:34::6c
|###[ DNS Resource Record ]###
|  rrname    = 'ns-cloud-c3.googledomains.com.'
|  type      = AAAA
|  rclass    = IN
|  ttl       = 21139
|  rdlen     = None
|  rdata     = 2001:4860:4802:36::6c
|###[ DNS Resource Record ]###
|  rrname    = 'ns-cloud-c4.googledomains.com.'
|  type      = AAAA
```

But when we ran without the root permission, we got an 'Operation not permitted' warning:



```
^C[09/28/22]seed@VM:~/Documents$
[09/28/22]seed@VM:~/Documents$ Lab3.py
Traceback (most recent call last):
  File "./Lab3.py", line 8, in <module>
    pkt = sniff(filter ='icmp', prn = print_pkt)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 1263, in
 sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.5/dist-packages/scapy/sendrecv.py", line 1128, in
 _run
    **karg)] = iface
  File "/usr/local/lib/python3.5/dist-packages/scapy/arch/linux.py", line 487, i
n __init__
    socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type))
  File "/usr/lib/python3.5/socket.py", line 134, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
[09/28/22]seed@VM:~/Documents$
```

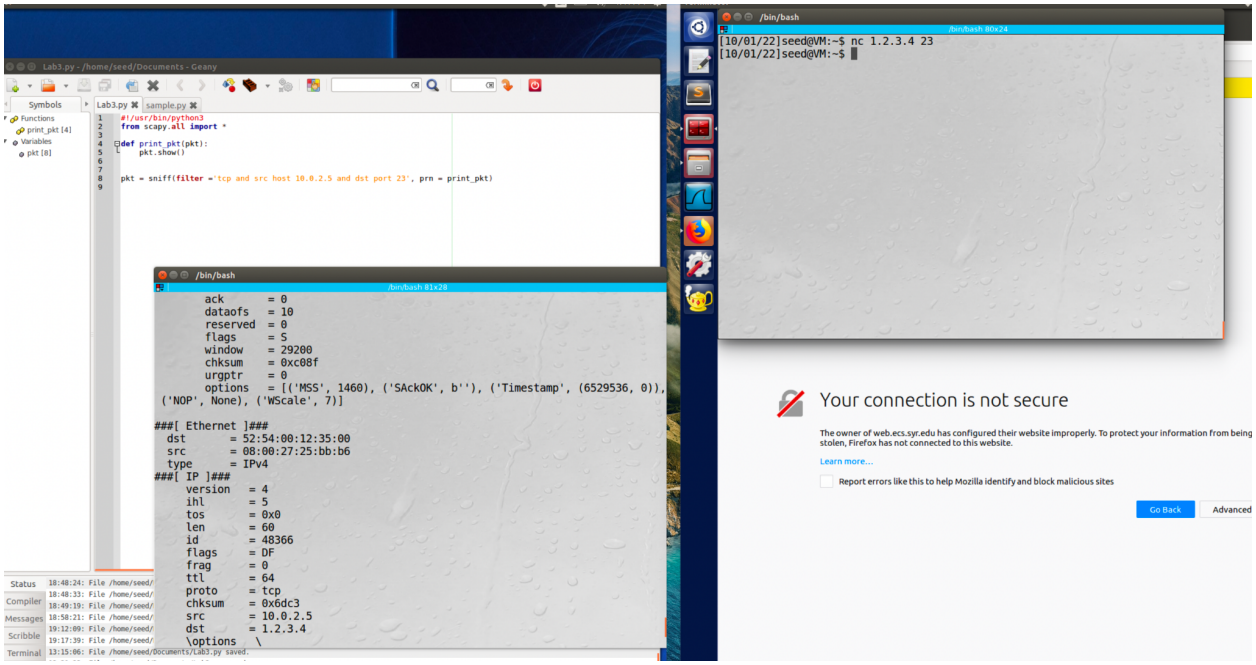
Task 1(B)- We changed the filter in the function arguments in order to only capture those packets that we specified. The first filter we applied is the ICMP filter. The terminal seemed to be stuck, until we opened up the web browser, then we saw the packet information displayed in the terminal.

We change the arguments inside the sniff function in order to set the filter to what is listed in Task 1(B). We were able to capture packets on ICMP filter, but in order to proceed with the others, we will be using a second virtual machine. Since our second machine's IP address is 10.0.2.5, we use this in our filter for machine 1 python program. Thus, the program for the second instruction is:

Lab3.py ✖ sample.py ✖

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

def print_pkt(pkt):
    pkt.show()


pkt = sniff(filter ='tcp and src host 10.0.2.5 and dst port 23', prn = print_pkt)

```

No packets were captured when I ran this program, I suspect because there are no packets going to port 23. But then I used the second machine and opened a TCP connection with netcat with a port 23 destination (command nc 1.2.3.4 23) to see if the first machine would capture the packets, and it did.

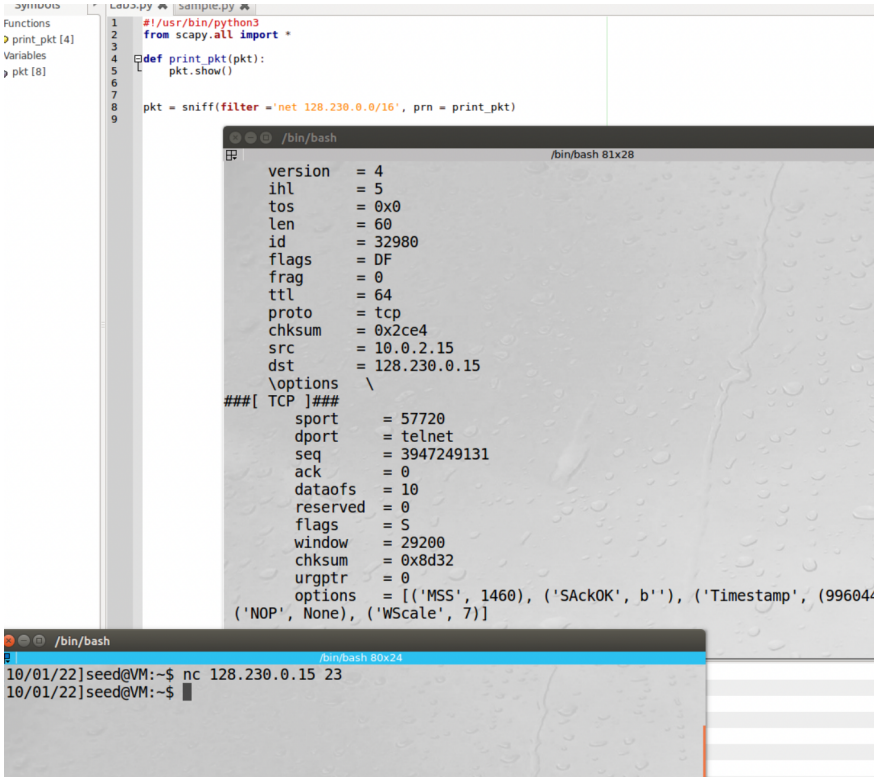


Next, we attempt to capture packets that come and go from subnet 128.230.0.0/16, so we once again alter our python program, to the following:

Lab3.py sample.py

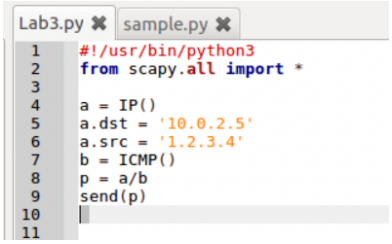
```
#!/usr/bin/python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()


pkt = sniff(filter ='net 128.230.0.0/16', prn = print_pkt)

```

Once I run the program, I open another terminal window and use the netcat command nc 128.230.0.15 23 to send a tcp packet that is part of the same subnet as 128.230.0.0/16. The packet is captured! As a test, I try to send another tcp packet, this time with a much different subnet using the netcat command nc 165.34.0.18 23. No packets are captured, thus our initial test worked.

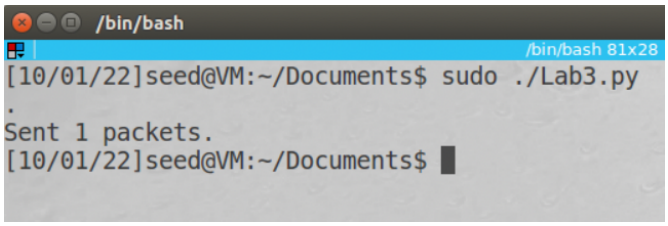


Task 2: Spoofing ICMP Packets

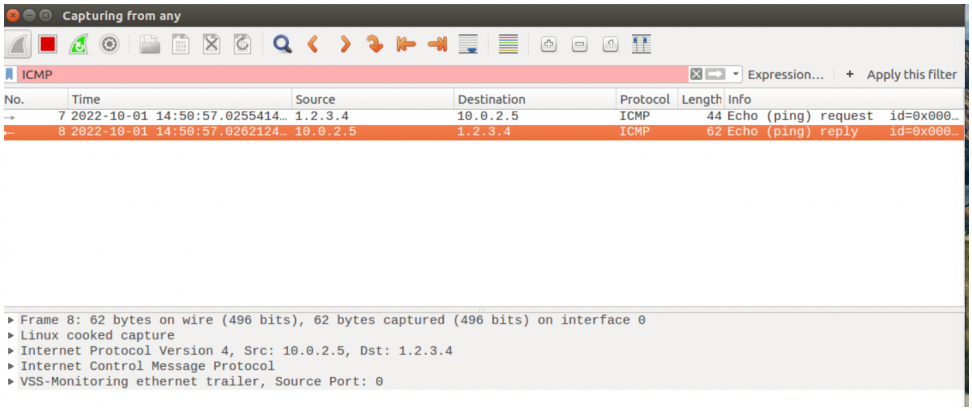
We start this task by writing out the python program, and use the IP address of machine 2 as the destination IP, as well as an arbitrary source IP 1.2.3.4:



I then open wireshark on machine 1, and use the ICMP filter to capture any packets. Next we run the Lab3.py program in the terminal using sudo ./Lab3.py , and we get the response that says “Sent 1 packets”:



We then notice that this packet was captured on Wireshark:



As we can see here, the source IP matches the spoofed (arbitrary) IP address we provided in the Lab3 python program. The request was sent to our second machine’s IP address (10.0.2.5) from 1.2.3.4 (arbitrary IP), which then replied to the same address as the destination. Thus we can conclude that the spoof was successful!

Task 3: Traceroute

The purpose of this task is to use Scapy to estimate the distance between our virtual machine and the chosen destination, in this case 8.8.8.8 which is Google's primary DNS.

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

hostname = '8.8.8.8'

for i in range(1, 15):
    pkt = IP(dst=hostname, ttl=i)
    b = ICMP()
    reply = sr1(pkt/b)
    print(reply.src)
```

In our program, we created a for loop that only increased ttl, which is the time-to-live, by one. We sent 15 packages to the target destination, our packages traveled through 11 hops until the 11th package reached the desired destinationation 8.8.8.8. We can see that the traceroute tool worked effectively.

```
[10/03/22]seed@VM:~$ sudo ./task3.py
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
10.0.2.2
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
192.168.1.1
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
142.254.236.93
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
24.30.169.45
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
72.129.19.2
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
72.129.17.2
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
66.109.3.232
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
72.14.221.120
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
142.250.236.213
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
142.251.60.99
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
8.8.8.8
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
8.8.8.8
Begin emission:
Finished sending 1 packets.
```

Task 4: Sniffing and-then Spoofing

In task 4, we combined sniffing and spoofing techniques to create the sniff-and-then-spoof program shown below. When the program detects an ICMP echo request, it should instantly send an echo reply using the packet spoofing technique.

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

def spoof_pkt(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("---Sniffed Packet---")
        print("Source IP:", pkt[IP].src)
        print("Destination IP:", pkt[IP].dst)
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load
        new_pkt = ip/icmp/data
        print("---Spoofed Packet---")
        print("Source IP:", new_pkt[IP].src)
        print("Destination IP:", new_pkt[IP].dst)
        send(new_pkt, verbose=0)

pkt = sniff(filter='icmp and src host 10.0.2.5', prn=spoof_pkt)
```

We needed two VMs to complete this task, on VM A we ping the IP 8.8.8.8 and generated an ICMP echo request.

```
[10/03/22]seed@VM:~$ 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=115 time=17.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=16.6 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=6.14 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=16.4 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=3.90 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=15.8 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=3.98 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=18.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=6 ttl=115 time=12.1 ms
```

Then on VM B we runned our program which sniffs packets in the same LAN network. The application will start sending out a spoofed ICMP echo reply if the type of the ICMP packet is an echo request (when the type value is 8).

```
[10/03/22]seed@VM:~$ sudo ./task4.py
---Sniffed Packet---
Source IP: 10.0.2.5
Destination IP: 8.8.8.8
---Spoofed Packet---
Source IP: 8.8.8.8
Destination IP: 10.0.2.5
---Sniffed Packet---
Source IP: 10.0.2.5
Destination IP: 8.8.8.8
---Spoofed Packet---
Source IP: 8.8.8.8
Destination IP: 10.0.2.5
---Sniffed Packet---
Source IP: 10.0.2.5
Destination IP: 8.8.8.8
---Spoofed Packet---
Source IP: 8.8.8.8
Destination IP: 10.0.2.5
```

As shown, our program was capable of sniffing and spoofing packets coming from VM A.