Justin Guerrero

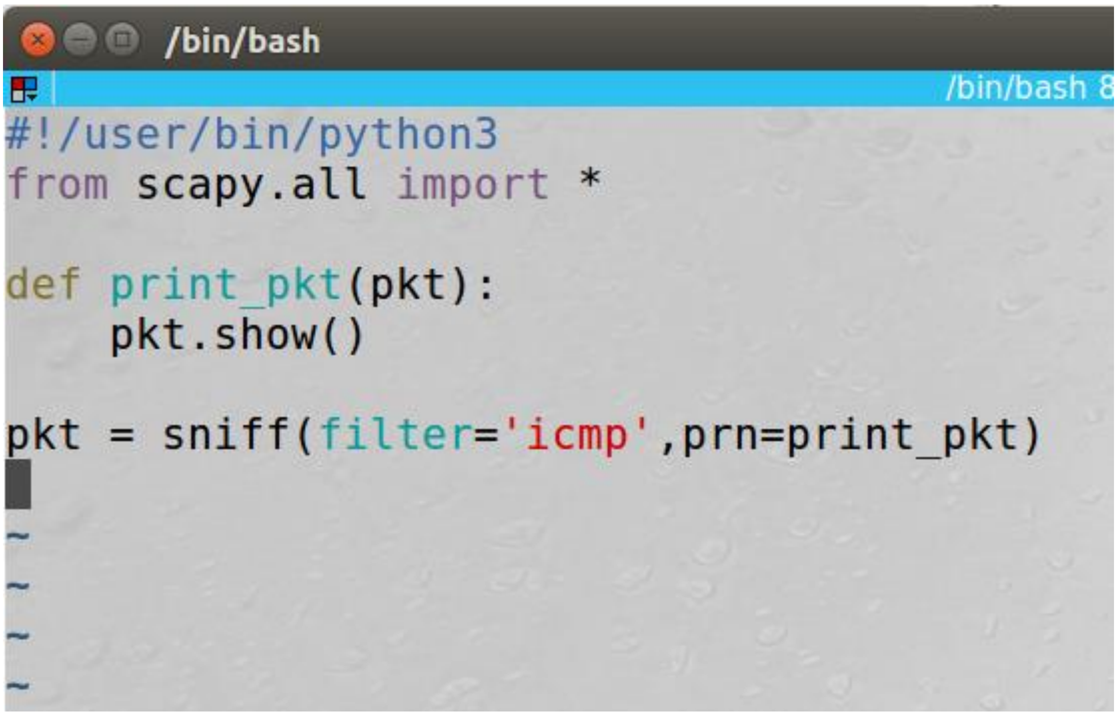
Lab 07 Sniffing and Spoofing

Task 1: Sniffing Packets

Task 1A: simple sniffer

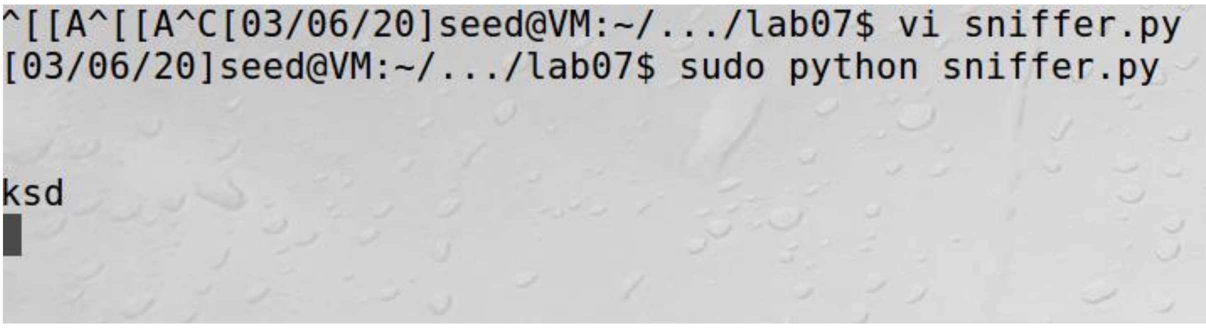
What we want to learn in task 1 is how to sniff packets using Python Programs.

We begin by using a simple python script.

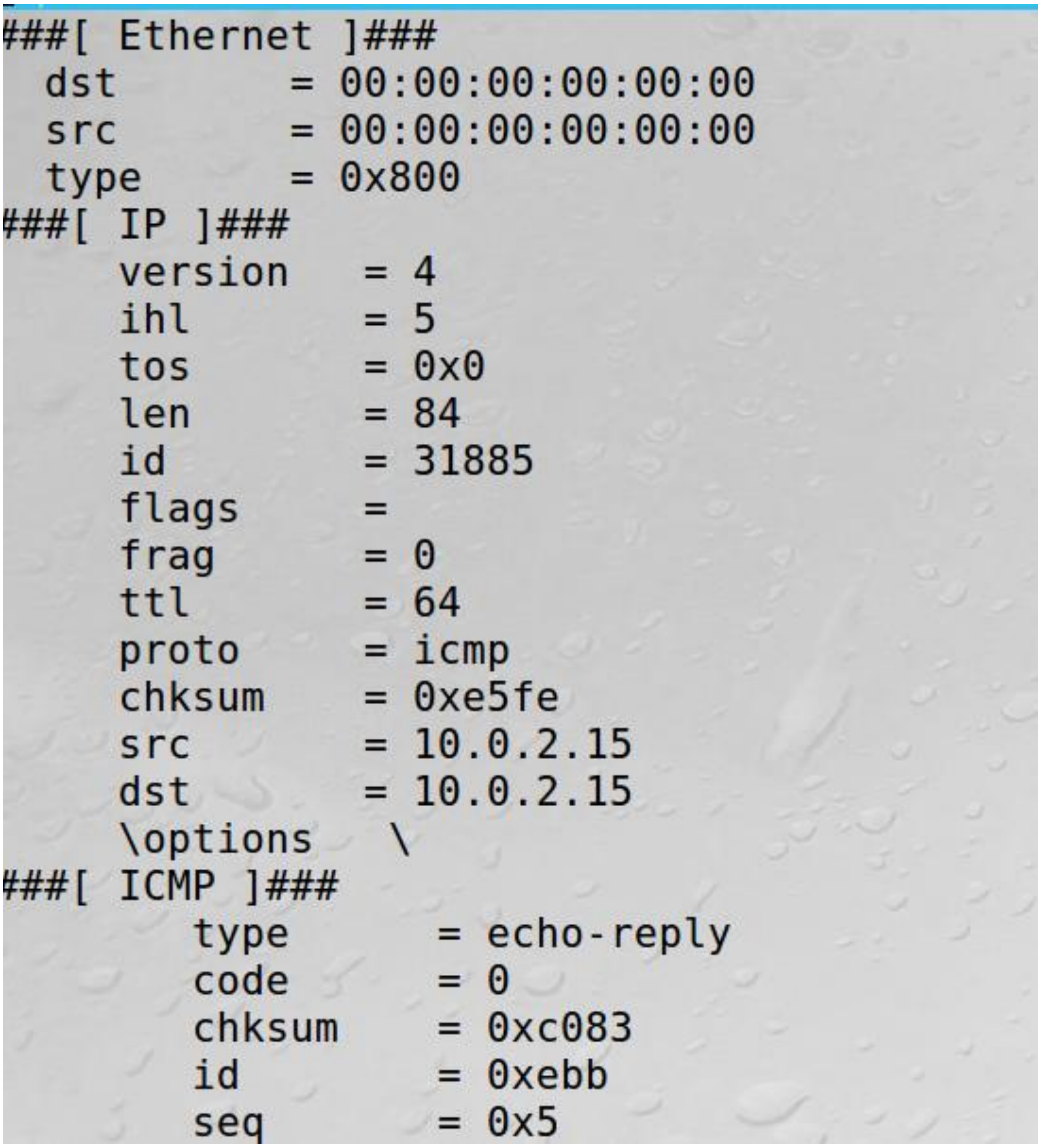


```
#!/user/bin/python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter='icmp',prn=print_pkt)

~
~
~
~
```

We then run our sniffer by using sudo python sniffer.py and see that we get the desired result, and see the incoming packets

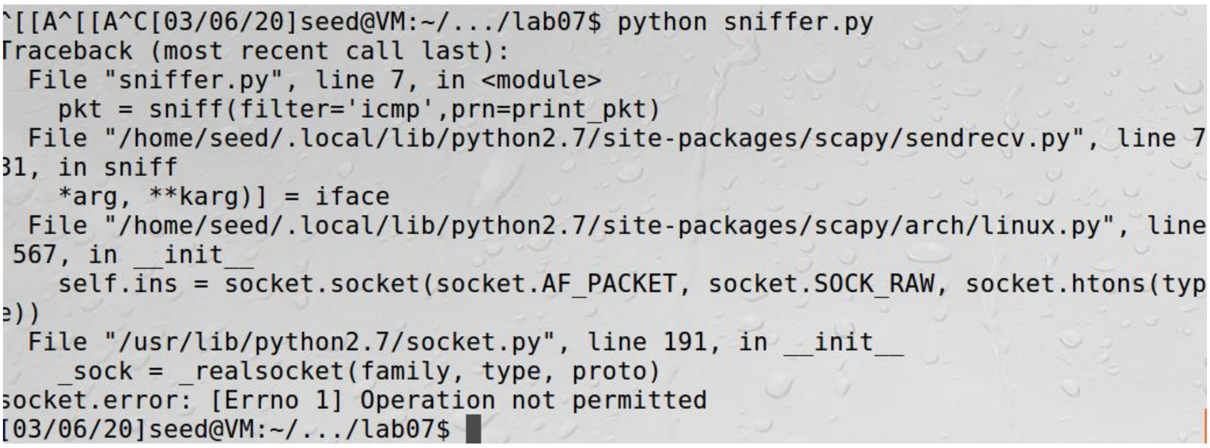


```
^[[A^[[A^C[03/06/20]seed@VM:~/.../lab07$ vi sniffer.py
[03/06/20]seed@VM:~/.../lab07$ sudo python sniffer.py


ksd
```



```
###[ Ethernet ]###
  dst       = 00:00:00:00:00:00
  src       = 00:00:00:00:00:00
  type      = 0x800
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 31885
     flags     = 
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xe5fe
     src       = 10.0.2.15
     dst       = 10.0.2.15
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0xc083
        id        = 0xebb
        seq       = 0x5
```

We are curious if this program must be run with root permissions, so we decided to run without sudo this time.

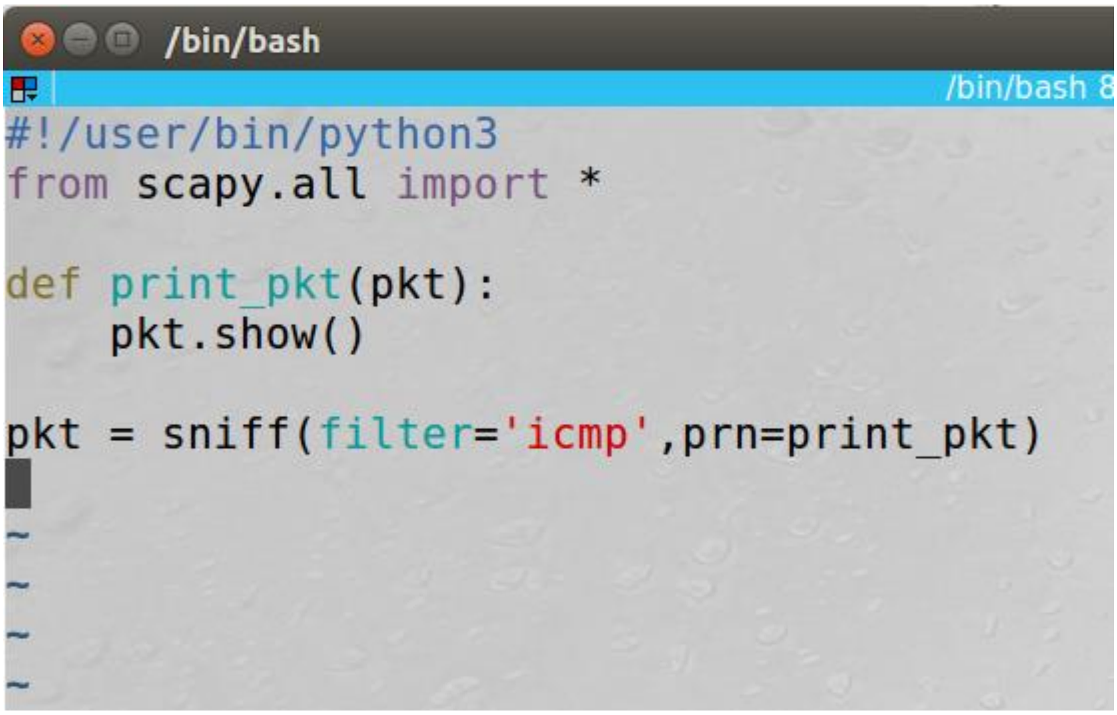


```
^[[A^[[A^C[03/06/20]seed@VM:~/.../lab07$ python sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 7, in <module>
    pkt = sniff(filter='icmp',prn=print_pkt)
  File "/home/seed/.local/lib/python2.7/site-packages/scapy/sendrecv.py", line 7
31, in sniff
    *arg, **karg)] = iface
  File "/home/seed/.local/lib/python2.7/site-packages/scapy/arch/linux.py", line
 567, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(typ
e))
  File "/usr/lib/python2.7/socket.py", line 191, in __init__
    _sock = _realsocket(family, type, proto)
socket.error: [Errno 1] Operation not permitted
[03/06/20]seed@VM:~/.../lab07$ 
```

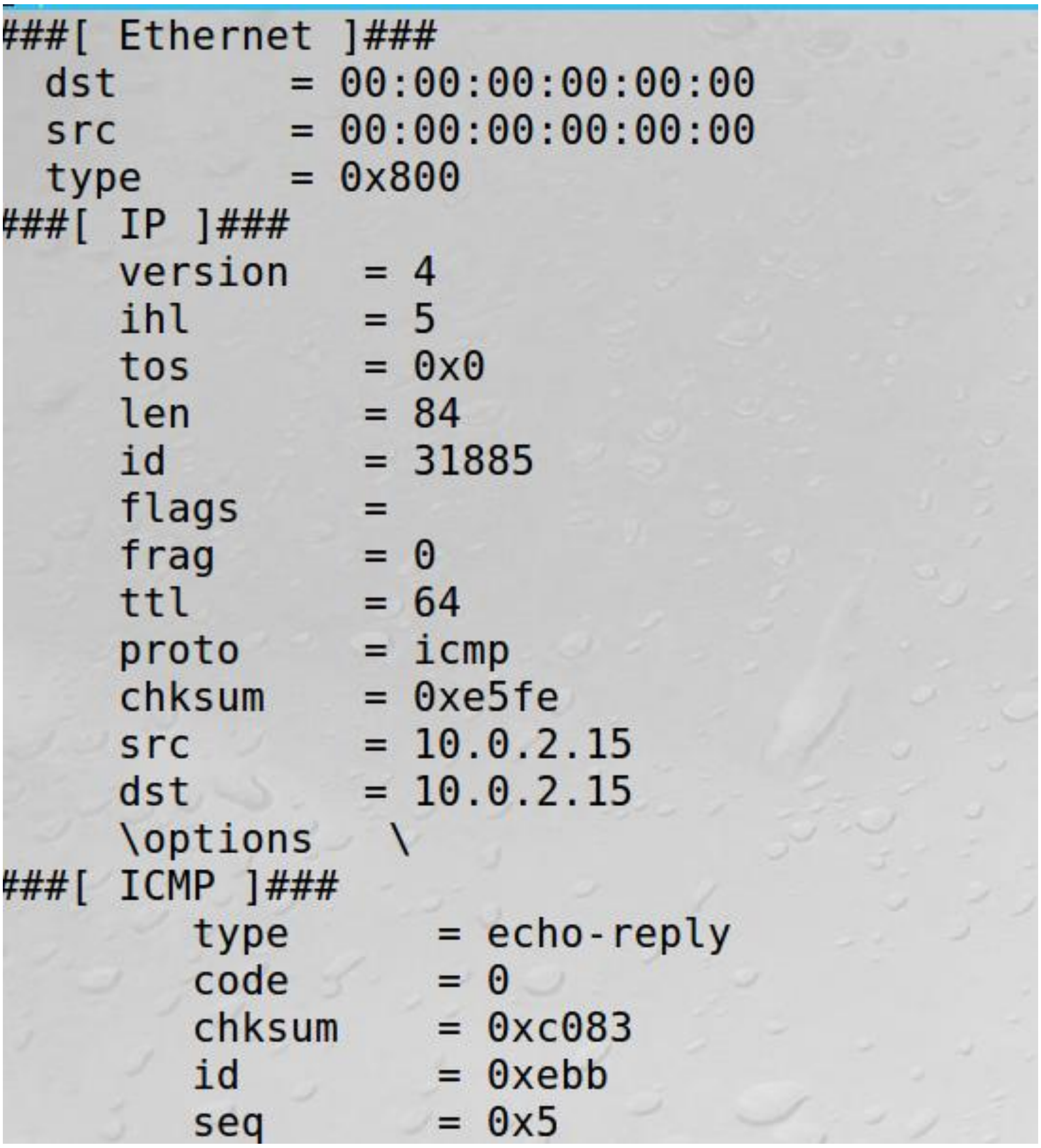
We notice we get errors that state we're not allowed to run this program. This is because the python libraries require super user permissions to run the socket function that is used in scapys sniff.

Task 1B:

In task 1B we want to get familiar with sniffing packets and get familiar with the python tool Scapy, similar to what we did in task 1A. This will help us in later steps of the lab. First, we set a filter from the BPF (Berkley Packet Filter) set to ICMP only, and we ping our virtual machines IP address.

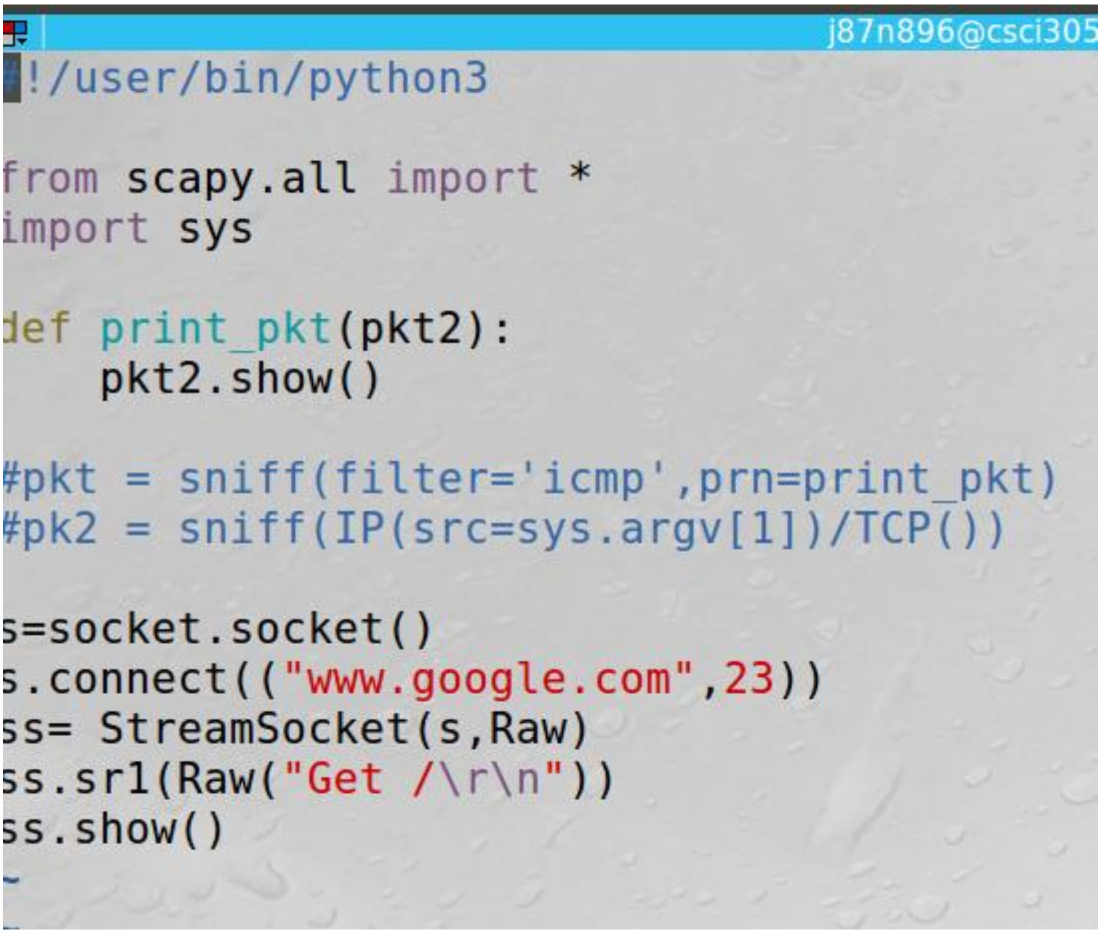


```
#!/user/bin/python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter='icmp',prn=print_pkt)

~
~
~
~
```

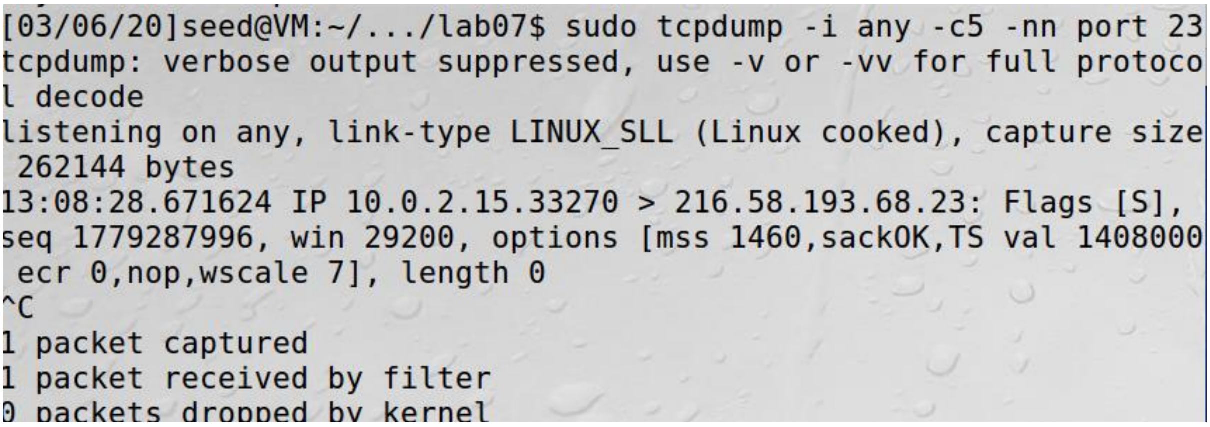


```
###[ Ethernet ]###
  dst       = 00:00:00:00:00:00
  src       = 00:00:00:00:00:00
  type      = 0x800
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 31885
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xe5fe
     src       = 10.0.2.15
     dst       = 10.0.2.15
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0xc083
        id        = 0xebb
        seq       = 0x5
```

Next, we want to set our sniffer to only receive things from a certain port and filter. This time we use TCP only and set our port to port 23.



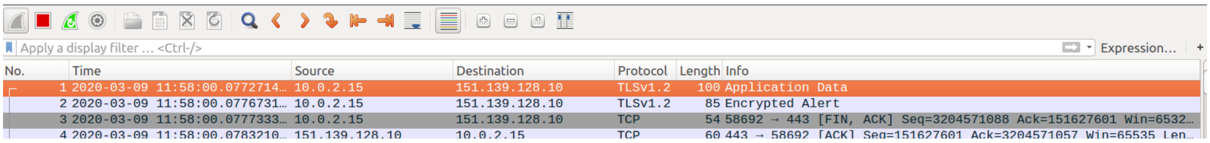
```
#!/user/bin/python3

from scapy.all import *
import sys

def print_pkt(pkt2):
    pkt2.show()

#pkt = sniff(filter='icmp',prn=print_pkt)
#pk2 = sniff(IP(src=sys.argv[1])/TCP())

s=socket.socket()
s.connect(("www.google.com",23))
ss= StreamSocket(s,Raw)
ss.sr1(Raw("Get /\r\n"))
ss.show()
~
~
```



```
[03/06/20]seed@VM:~/.../lab07$ sudo tcpdump -i any -c5 -nn port 23
tcpdump: verbose output suppressed, use -v or -vv for full protoco
l decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size
 262144 bytes
13:08:28.671624 IP 10.0.2.15.33270 > 216.58.193.68.23: Flags [S],
seq 1779287996, win 29200, options [mss 1460,sackOK,TS val 1408000
 ecr 0,nop,wscale 7], length 0
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
```

Here you can see that we were able to send packets out and receive them back from www.google.com on port 23. The code I took examples from the Scapy documentation.

In the next part we want to check out packages from other subnets other than our own, a more real world approach if you will. We open wireshark and start running the program by clicking the "any" connection. The next thing we did was go to the internet and just start refreshing tabs and navigating around the web. This allowed us to catch a lot of web traffic, shown below.

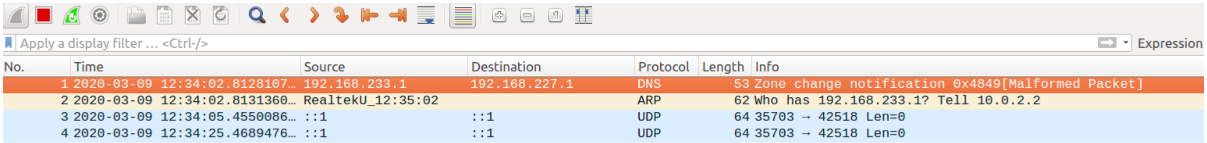


```
No. Time                          Source          Destination     Protocol Length Info
  1 2020-03-09 11:58:00.0772714…  10.0.2.15       151.139.128.10  TLSv1.2     100 Application Data
  2 2020-03-09 11:58:00.0776731…  10.0.2.15       151.139.128.10  TLSv1.2      85 Encrypted Alert
  3 2020-03-09 11:58:00.0777333…  10.0.2.15       151.139.128.10  TCP          54 58692 → 443 [FIN, ACK] Seq=3204571088 Ack=151627601 Win=6532…
  4 2020-03-09 11:58:00.0783210…  151.139.128.10  10.0.2.15       TCP          60 443 → 58692 [ACK] Seq=151627601 Ack=3204571057 Win=65535 Len…
```

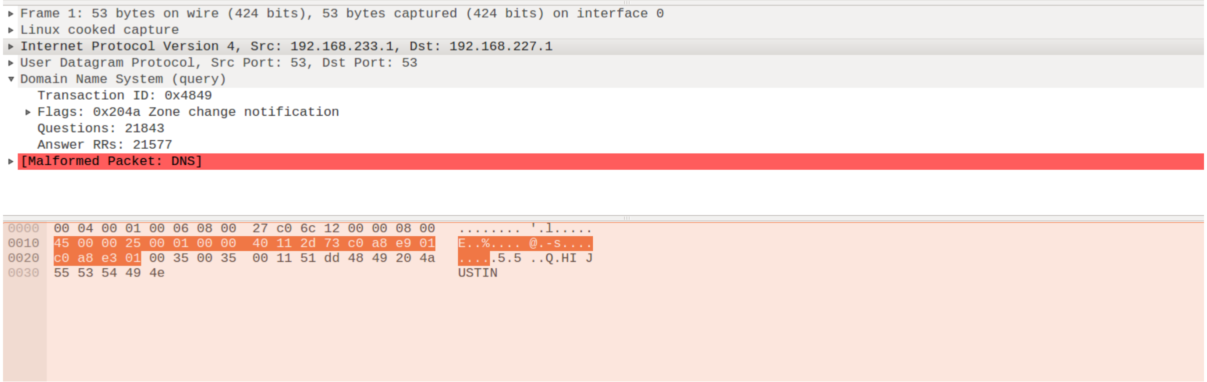
Task 2: Spoofing ICMP Packets

The task here is to send a "spoofed" packet to ourselves. We do this with the Scapy tool by typing in the attack string send (IP (dst='192.168.227.1',src='192.168.233.1')/UDP()/"HI JUSTIN")

this effectively allowed us to "spoof" a package.



```
No. Time                          Source             Destination    Protocol Length Info
  1 2020-03-09 12:34:02.8128107…  192.168.233.1      192.168.227.1  DNS          53 Zone change notification 0x4849[Malformed Packet]
  2 2020-03-09 12:34:02.8131360…  RealtekU_12:35:02                 ARP          62 Who has 192.168.233.1? Tell 10.0.2.2
  3 2020-03-09 12:34:05.4550086…  ::1                ::1            UDP          64 35703 → 42518 Len=0
  4 2020-03-09 12:34:25.4689476…  ::1                ::1            UDP          64 35703 → 42518 Len=0
```

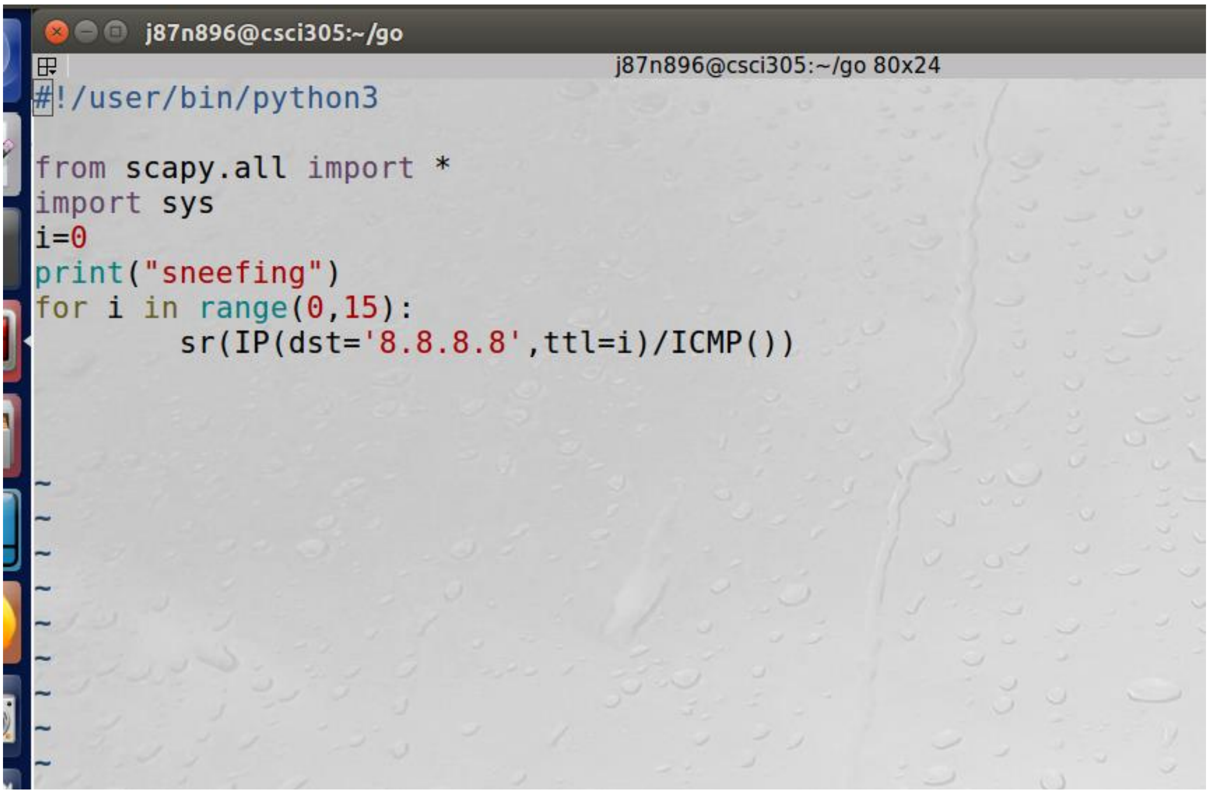


```
▸ Frame 1: 53 bytes on wire (424 bits), 53 bytes captured (424 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 192.168.233.1, Dst: 192.168.227.1
▸ User Datagram Protocol, Src Port: 53, Dst Port: 53
▾ Domain Name System (query)
    Transaction ID: 0x4849
  ▸ Flags: 0x204a Zone change notification
    Questions: 21843
    Answer RRs: 21577
▸ [Malformed Packet: DNS]
```

```
0000  00 04 00 01 00 06 08 00  27 c0 6c 12 00 00 08 00   ........ '.l.....
0010  45 00 00 25 00 01 00 00  40 11 2d 73 c0 a8 e9 01   E..%.... @.-s....
0020  c0 a8 e3 01 00 35 00 35  00 11 51 dd 48 49 20 4a   .....5.5 ..Q.HI J
0030  55 53 54 49 4e                                     USTIN
```

Task 3: Traceback

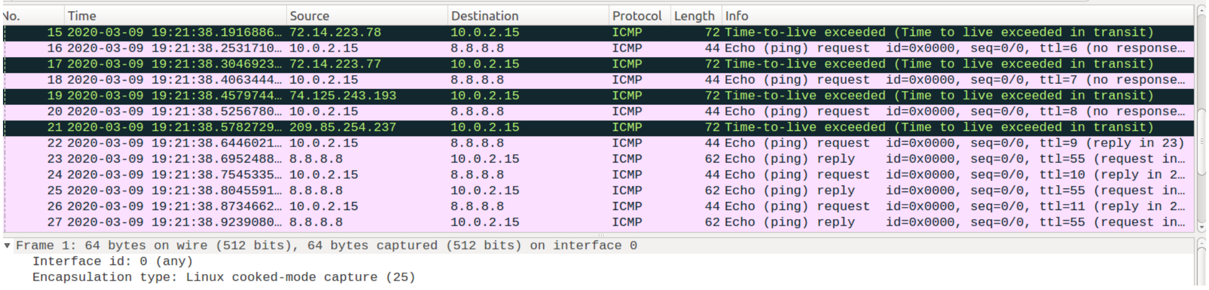
What we are tasked to do is find the distance in terms of routers between my VM and google. We set a for loop to run 15 times sending a ping to our target ip address. We find that the "Time-to-live has exceeded" error runs 9 times before we get a response from google.



j87n896@csci305:~/go

j87n896@csci305:~/go 80x24

```
#!/user/bin/python3

from scapy.all import *
import sys
i=0
print("sneefing")
for i in range(0,15):
        sr(IP(dst='8.8.8.8',ttl=i)/ICMP())
~
~
~
~
~
~
~
~
~
```

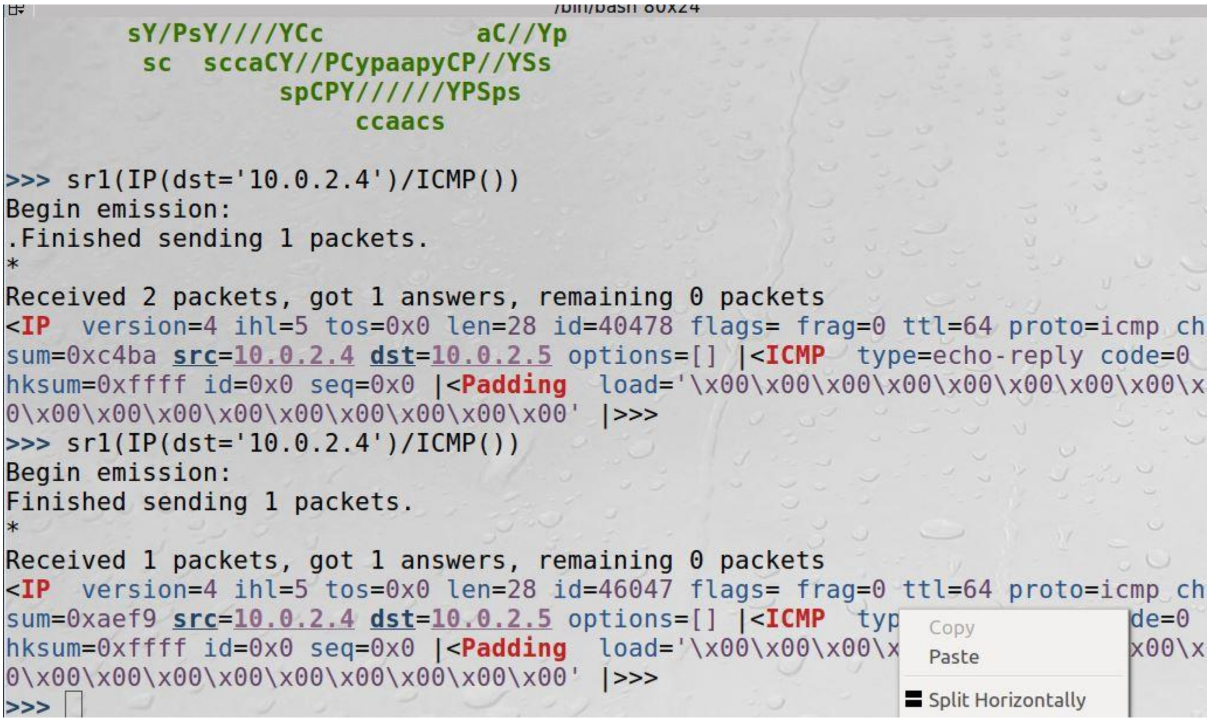


No.	Time	Source	Destination	Protocol	Length	Info
15	2020-03-09 19:21:38.1916886…	72.14.223.78	10.0.2.15	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
16	2020-03-09 19:21:38.2531710…	10.0.2.15	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=6 (no response…
17	2020-03-09 19:21:38.3046923…	72.14.223.77	10.0.2.15	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
18	2020-03-09 19:21:38.4063444…	10.0.2.15	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=7 (no response…
19	2020-03-09 19:21:38.4579744…	74.125.243.193	10.0.2.15	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
20	2020-03-09 19:21:38.5256780…	10.0.2.15	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=8 (no response…
21	2020-03-09 19:21:38.5782729…	209.85.254.237	10.0.2.15	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
22	2020-03-09 19:21:38.6446021…	10.0.2.15	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=9 (reply in 23)
23	2020-03-09 19:21:38.6952488…	8.8.8.8	10.0.2.15	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=55 (request in…
24	2020-03-09 19:21:38.7545335…	10.0.2.15	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=10 (reply in 2…
25	2020-03-09 19:21:38.8045591…	8.8.8.8	10.0.2.15	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=55 (request in…
26	2020-03-09 19:21:38.8734662…	10.0.2.15	8.8.8.8	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=11 (reply in 2…
27	2020-03-09 19:21:38.9239080…	8.8.8.8	10.0.2.15	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=55 (request in…

```
▾ Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface 0
    Interface id: 0 (any)
    Encapsulation type: Linux cooked-mode capture (25)
```

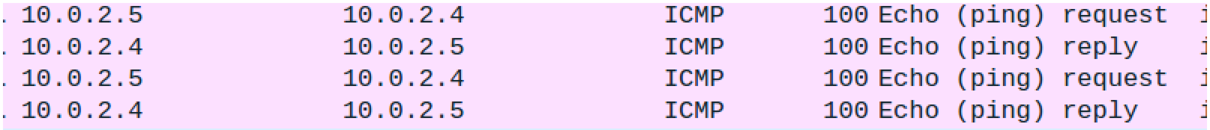
Task 4: Sniffing and then spoofing

We set up two virtual machines on the same network that we can talk back and forth with. This allows us to practice sniffing and spoofing safely without harming others in the process. We begin by setting up our scapy packet to send. I decided to send a simple ICMP packet to test the response.



```
        sY/PsY////YCc          aC//Yp
         sc  sccaCY//PCypaapyCP//YSs
                  spCPY//////YPSps
                       ccaacs

>>> sr1(IP(dst='10.0.2.4')/ICMP())
Begin emission:
.Finished sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
<IP  version=4 ihl=5 tos=0x0 len=28 id=40478 flags= frag=0 ttl=64 proto=icmp ch
sum=0xc4ba src=10.0.2.4 dst=10.0.2.5 options=[] |<ICMP  type=echo-reply code=0
hksum=0xffff id=0x0 seq=0x0 |<Padding  load='\x00\x00\x00\x00\x00\x00\x00\x00\x
0\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>
>>> sr1(IP(dst='10.0.2.4')/ICMP())
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
<IP  version=4 ihl=5 tos=0x0 len=28 id=46047 flags= frag=0 ttl=64 proto=icmp ch
sum=0xaef9 src=10.0.2.4 dst=10.0.2.5 options=[] |<ICMP  typ               de=0
hksum=0xffff id=0x0 seq=0x0 |<Padding  load='\x00\x00\x00\x               x00\x
0\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>
>>> 
```

The next thing we do is open Wireshark in the other VM to monitor the traffic that is going to our VM. You can see we have successfully spoofed our VM in the photo below by showing we receive a pin and send a response back.



```
. 10.0.2.5          10.0.2.4          ICMP      100 Echo (ping) request   i
. 10.0.2.4          10.0.2.5          ICMP      100 Echo (ping) reply     i
. 10.0.2.5          10.0.2.4          ICMP      100 Echo (ping) request   i
. 10.0.2.4          10.0.2.5          ICMP      100 Echo (ping) reply     i
```

The final takeaway from this lab is to get familiar with how internet package interception and spoofing is handled and how to be on the watch for these types of action. Packet sending can lead to a lot of common hacks if you spoof the right way!