CSC 223

Title and Experiment #	Lab 6: Packet Sniffing and Spoofing
Name	Gianna Galard
Date Performed	28-Nov-22
Date Submitted	28-Nov-22

The student pledges this work to be their own *Gianna Galard*

Overview:

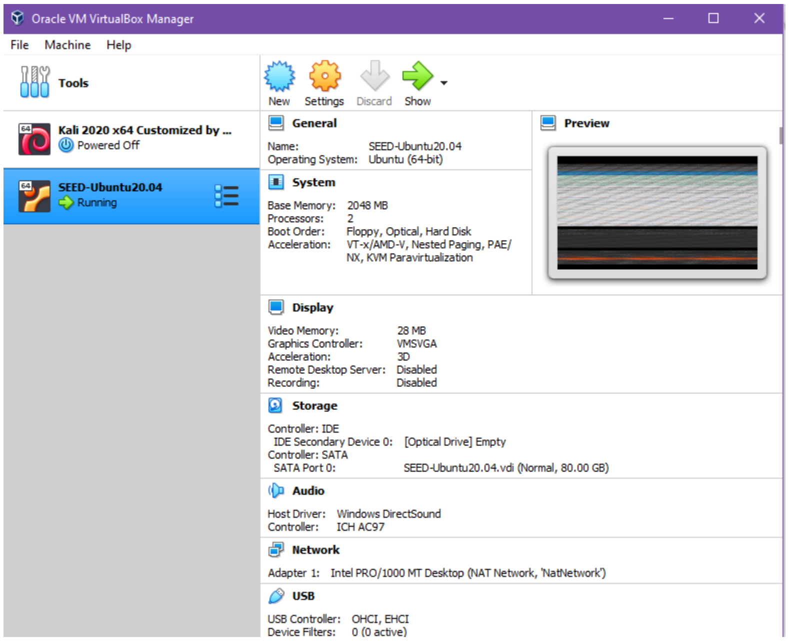
Packet sniffing and spoofing are two crucial concepts in network security; they are two significant threats to network communication. Understanding these two threats is essential for understanding security measures in networking. Many packet sniffing and spoofing tools, such as Wireshark, Tcpdump, Netwox, Scapy, etc. Some of these tools are widely used by security experts, as well as by attackers. Using these tools is essential for students, but what is more important for students in a network security course is to understand how these tools work, i.e., how packet sniffing and spoofing are implemented in software.

The objective of this lab is two-fold: learning to use the tools and understanding the technologies underlying these tools. For the second object, students will write simple sniffer and spoofing programs and gain an in-depth understanding of the technical aspects of these programs. This lab covers the following topics:

- How the sniffing and spoofing work
- Packet sniffing using the pcap library and Scapy
- Packet spoofing using raw socket and Scapy
- Manipulating packets using Scapy

Lab Environment Setup:

- Launched the Virtual Machine



General

Name: SEED-Ubuntu20.04
Operating System: Ubuntu (64-bit)

System

Base Memory: 2048 MB
Processors: 2
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization

Display

Video Memory: 28 MB
Graphics Controller: VMSVGA
Acceleration: 3D
Remote Desktop Server: Disabled
Recording: Disabled

Storage

Controller: IDE
IDE Secondary Device 0: [Optical Drive] Empty
Controller: SATA
SATA Port 0: SEED-Ubuntu20.04.vdi (Normal, 80.00 GB)

Audio

Host Driver: Windows DirectSound
Controller: ICH AC97

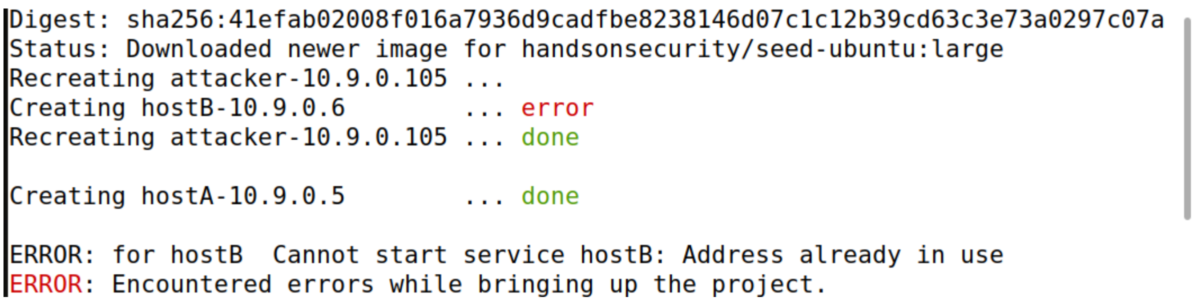
Network

Adapter 1: Intel PRO/1000 MT Desktop (NAT Network, 'NatNetwork')

USB

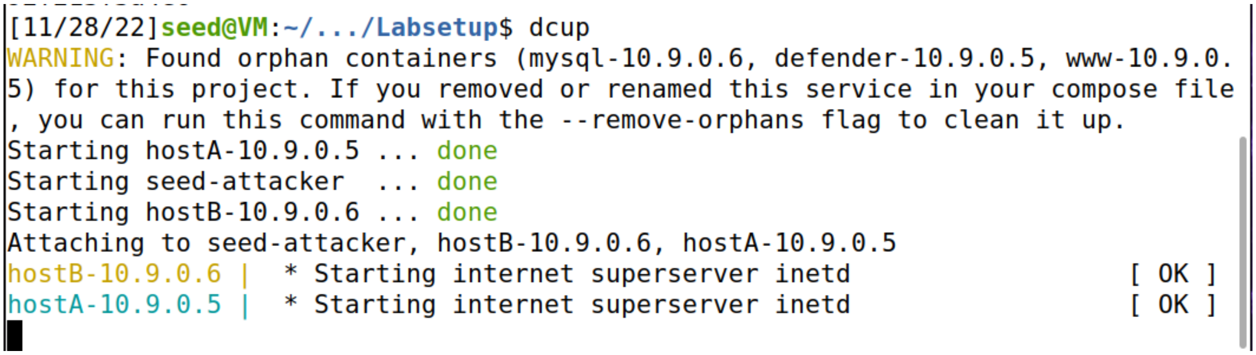
USB Controller: OHCI, EHCI
Device Filters: 0 (0 active)

- Downloaded and used Labsetup as a shared folder between Linux VM and Host PC
- Ran the following commands in the following order in terminal:
  - dcbuild to build the container
  - dcup to start the container
  - ***note* -> when running dcup, I received an error that the address was already being used. After further research on Google, the solution was to simply stop any containers running which can be seen by using the docker container ls, and then stopping all containers using docker container stop $(docker ps -aq)**



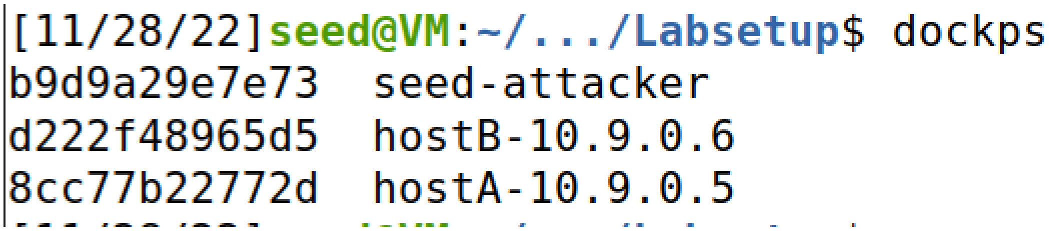
```
Digest: sha256:41efab02008f016a7936d9cadfbe8238146d07c1c12b39cd63c3e73a0297c07a
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:large
Recreating attacker-10.9.0.105 ...
Creating hostB-10.9.0.6        ... error
Recreating attacker-10.9.0.105 ... done

Creating hostA-10.9.0.5        ... done

ERROR: for hostB  Cannot start service hostB: Address already in use
ERROR: Encountered errors while bringing up the project.
```

- Afterwards, running dcup worked correctly:



```
[11/28/22]seed@VM:~/.../Labsetup$ dcup
WARNING: Found orphan containers (mysql-10.9.0.6, defender-10.9.0.5, www-10.9.0.
5) for this project. If you removed or renamed this service in your compose file
, you can run this command with the --remove-orphans flag to clean it 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
hostB-10.9.0.6 |  * Starting internet superserver inetd                    [ OK ]
hostA-10.9.0.5 |  * Starting internet superserver inetd                    [ OK ]
```

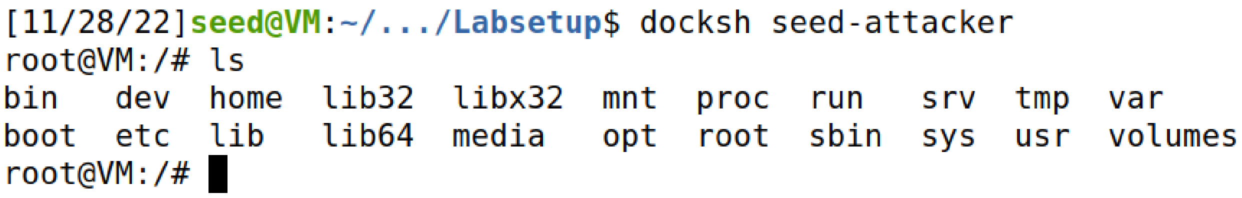
- To verify that it’s working correctly, I ran dockps:



```
[11/28/22]seed@VM:~/.../Labsetup$ dockps
b9d9a29e7e73  seed-attacker
d222f48965d5  hostB-10.9.0.6
8cc77b22772d  hostA-10.9.0.5
```

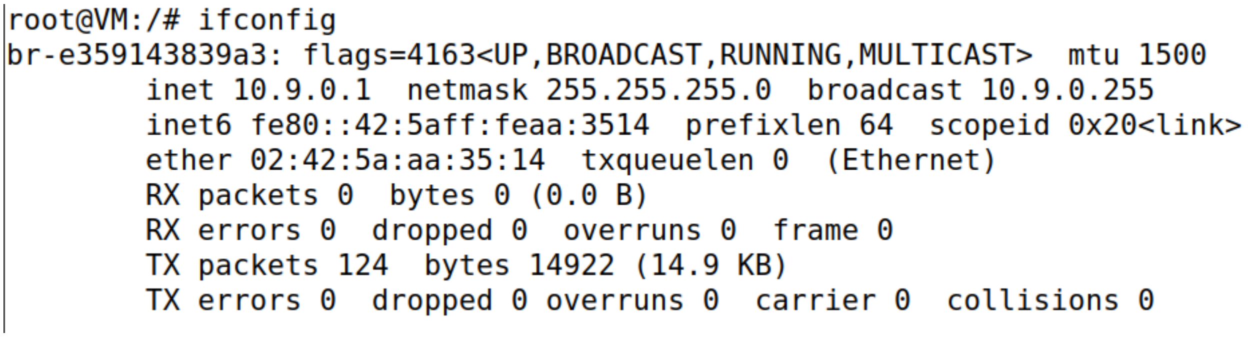
2.2 About the Attacker Container

We will go into the attacker container using docksh seed-attacker



```
[11/28/22]seed@VM:~/.../Labsetup$ docksh seed-attacker
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:/# 
```

Then, we will run the ifconfig command in the terminal inside the attack container, then scroll up to the interface that starts with br and note it down:

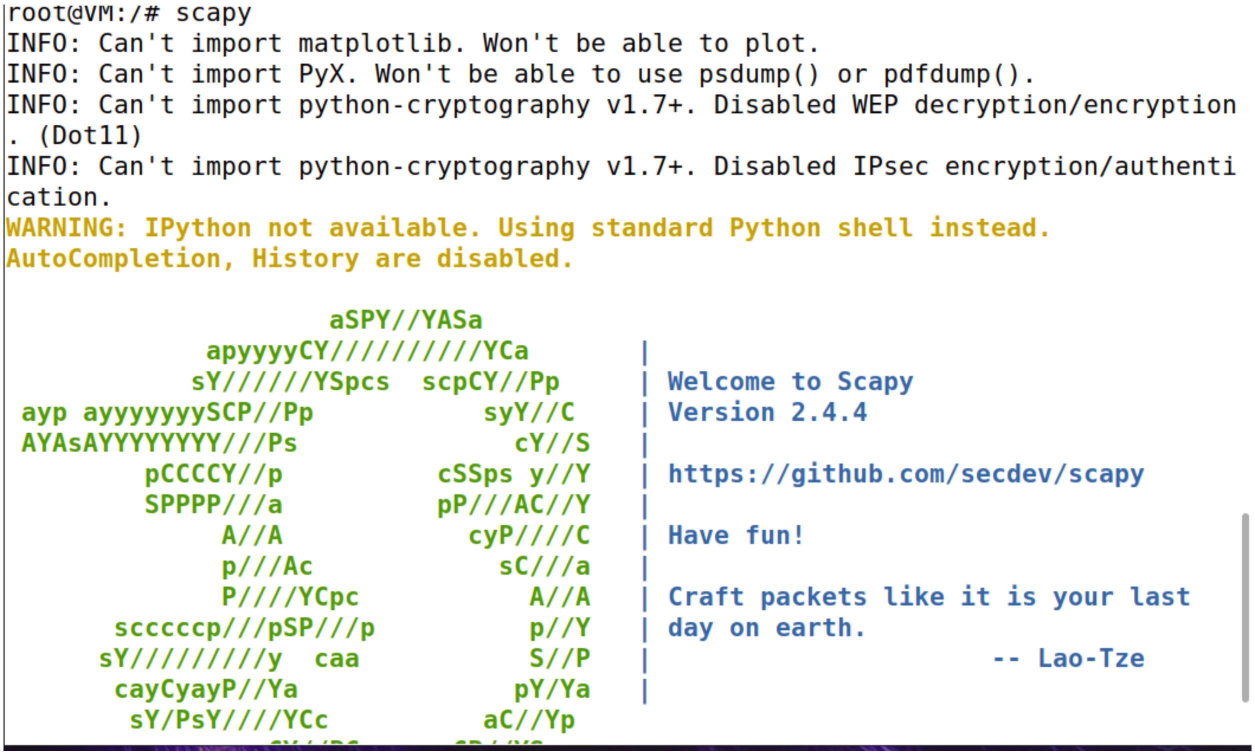


```
root@VM:/# ifconfig
br-e359143839a3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:5aff:feaa:3514  prefixlen 64  scopeid 0x20<link>
        ether 02:42:5a:aa:35:14  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 124  bytes 14922 (14.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Br-e359143839a3

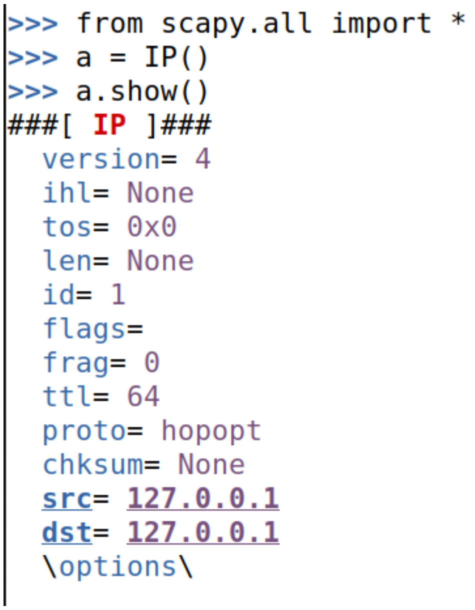
Lab Task Set 1: Using Scapy to Sniff and Spoof Packets

We can go into the scapy interface using the scapy command:



```
root@VM:/# scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
INFO: Can't import python-cryptography v1.7+. Disabled WEP decryption/encryption
. (Dot11)
INFO: Can't import python-cryptography v1.7+. Disabled IPsec encryption/authenti
cation.
WARNING: IPython not available. Using standard Python shell instead.
AutoCompletion, History are disabled.

                     aSPY//YASa
             apyyyyCY//////////YCa       |
            sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C    | Version 2.4.4
 AYAsAYYYYYYYY///Ps              cY//S   |
         pCCCCY//p          cSSps y//Y   | https://github.com/secdev/scapy
         SPPPP///a          pP///AC//Y   |
              A//A            cyP////C   | Have fun!
              p///Ac            sC///a   |
              P////YCpc           A//A   | Craft packets like it is your last
       scccccp///pSP///p          p//Y   | day on earth.
      sY/////////y  caa           S//P   |                       -- Lao-Tze
       cayCyayP//Ya              pY/Ya   |
        sY/PsY////YCc          aC//Yp
```

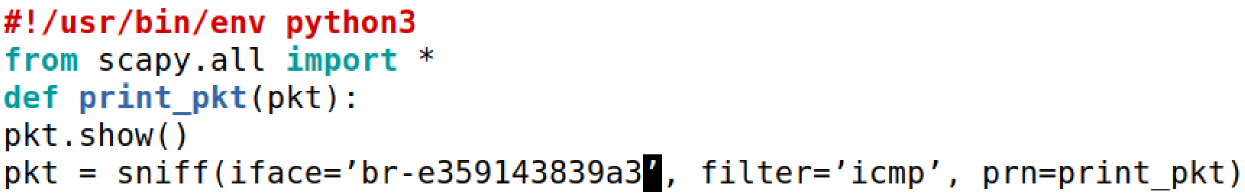
Then we can run the following code in the terminal:



```
>>> from scapy.all import *
>>> a = IP()
>>> a.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= hopopt
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
```

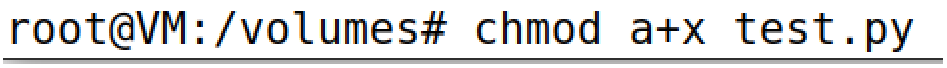
Task 1.1: Sniffing Packets

I created a new file called code.py inside the volumes folder (shared folder) and pasted the code in there and changed the interface name to match the one I noted down previously:



```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
pkt.show()
pkt = sniff(iface=’br-e359143839a3’, filter=’icmp’, prn=print_pkt)
```

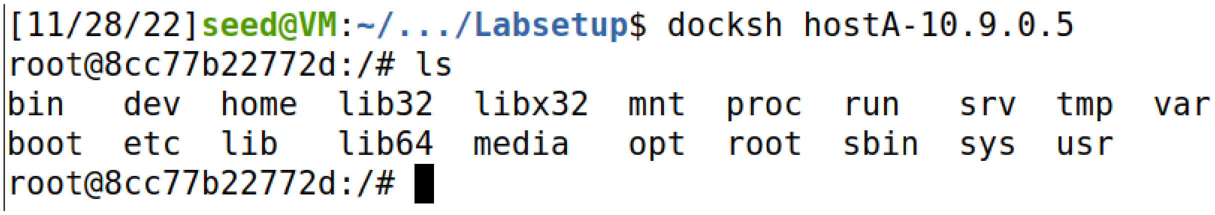
Now I can access the code in the attacker container, where I ran the chmod command:



```
root@VM:/volumes# chmod a+x test.py
```

Now we can run the file by simply using ./test.py, which will not show anything yet

We will now switch over to the hostA:

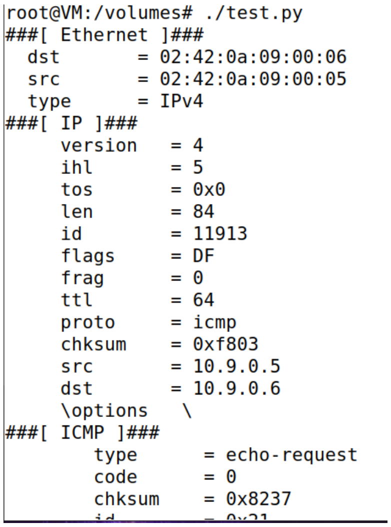


```
[11/28/22]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@8cc77b22772d:/# ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
root@8cc77b22772d:/# 
```

We can ping 10.9.0.6 to give us results when we run ./test.py

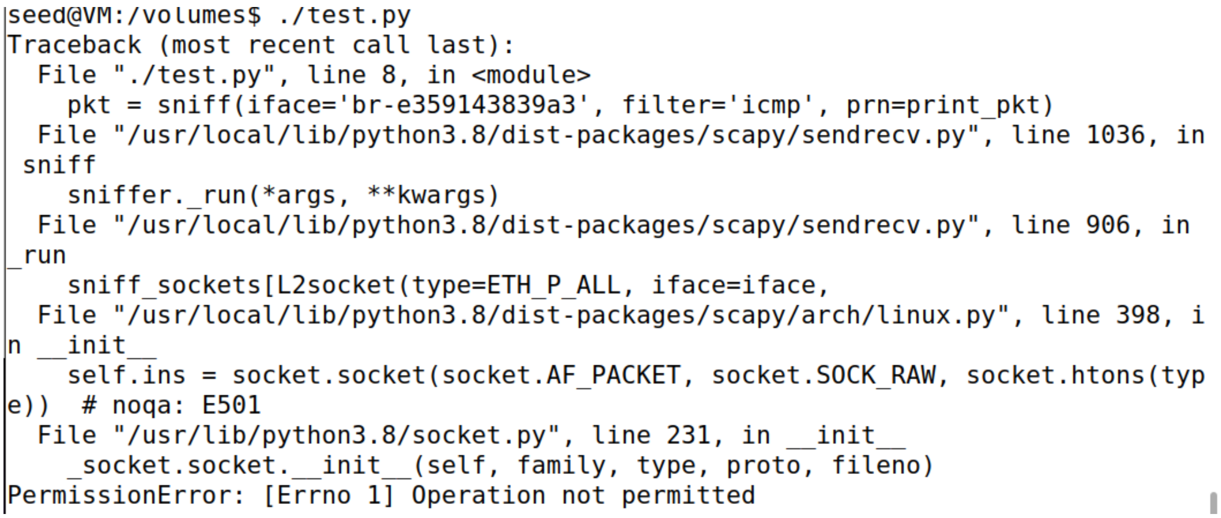
Task 1.1a

Running ./test.py in root after pinging:



```
root@VM:/volumes# ./test.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 11913
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xf803
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x8237
        id        = 0x21
```

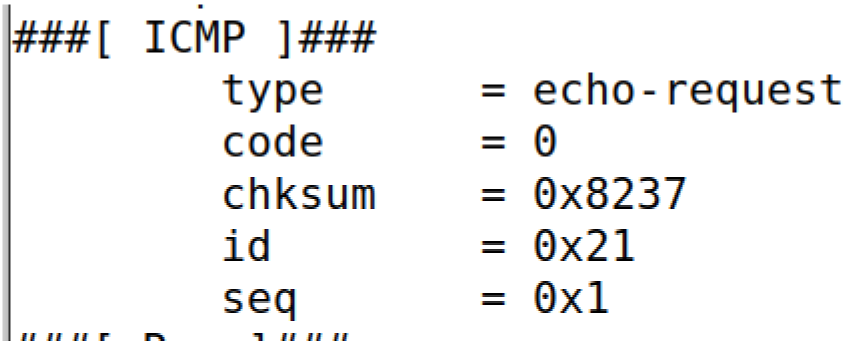
Running su seed and then ./test.py gives me an operation not permitted error:



```
seed@VM:/volumes$ ./test.py
Traceback (most recent call last):
  File "./test.py", line 8, in <module>
    pkt = sniff(iface='br-e359143839a3', filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1036, in
 sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 906, in
_run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 398, i
n __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(typ
e))  # noqa: E501
  File "/usr/lib/python3.8/socket.py", line 231, in __init__
    _socket.socket.__init__(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
```

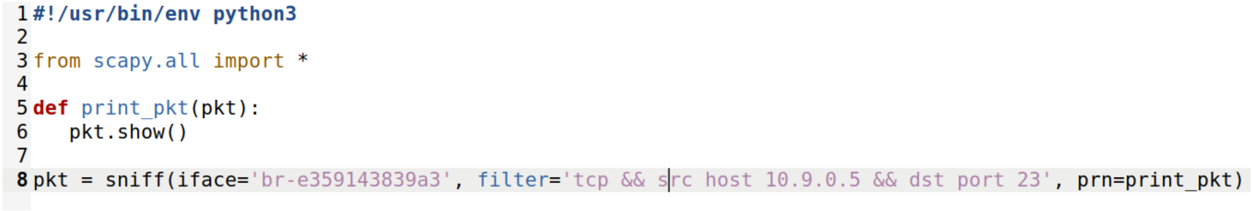
Task 1.1b

For ICMP, we are already filtering ICMP from task 1.1A:



```
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x8237
        id        = 0x21
        seq       = 0x1
```

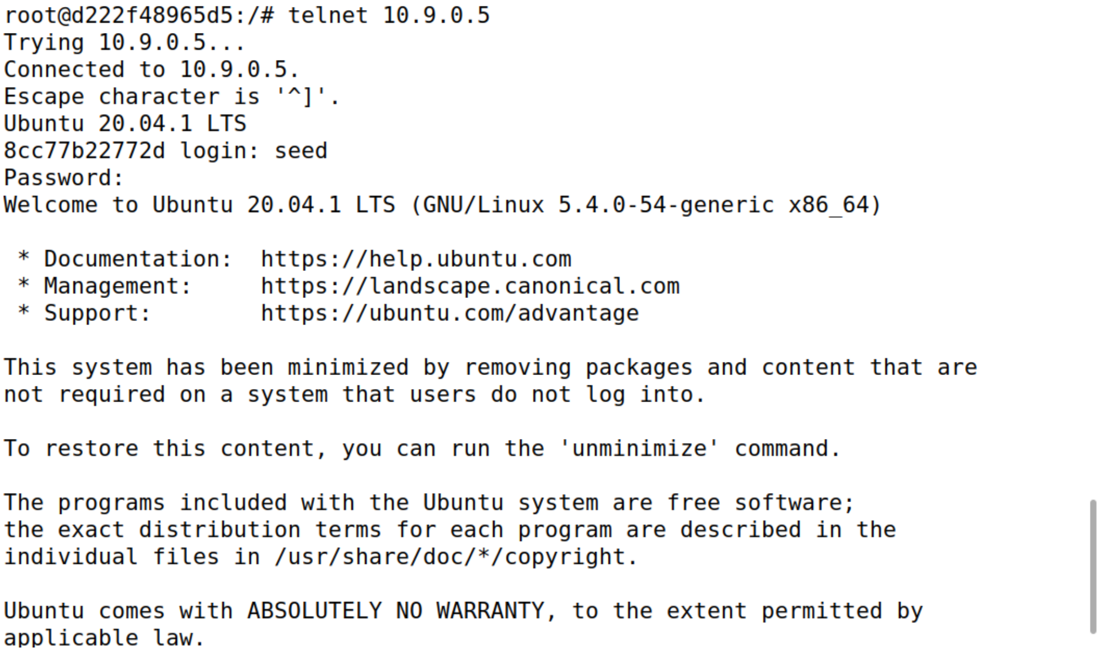
TCP code (we can use the following syntax) and results:



```
#!/usr/bin/env python3

from scapy.all import *

def print_pkt(pkt):
   pkt.show()

pkt = sniff(iface='br-e359143839a3', filter='tcp && src host 10.9.0.5 && dst port 23', prn=print_pkt)
```

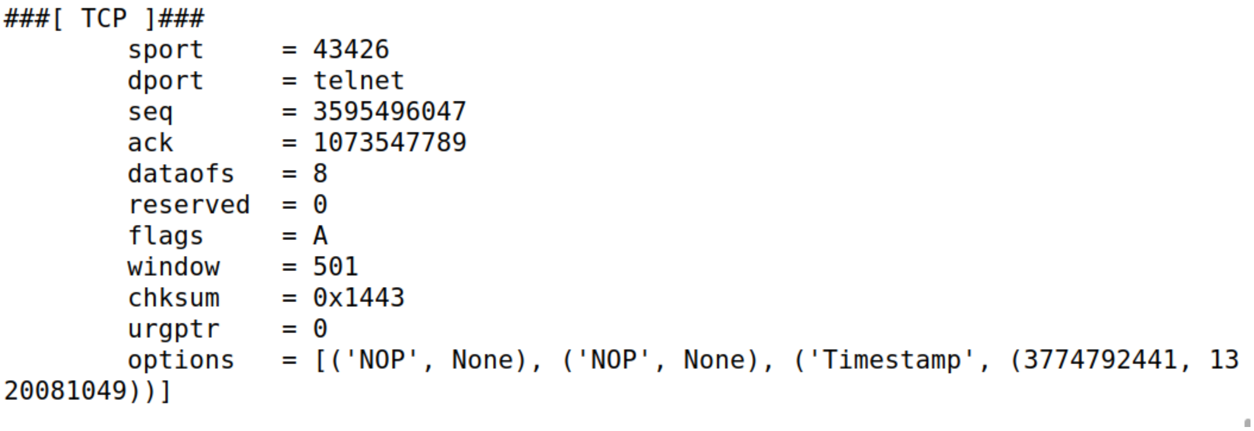
Looking up port 23 online, we now know that that refers to telnet.

We go into hostB container, then we can use telnet 10.9.0.5 to connect with hostA:



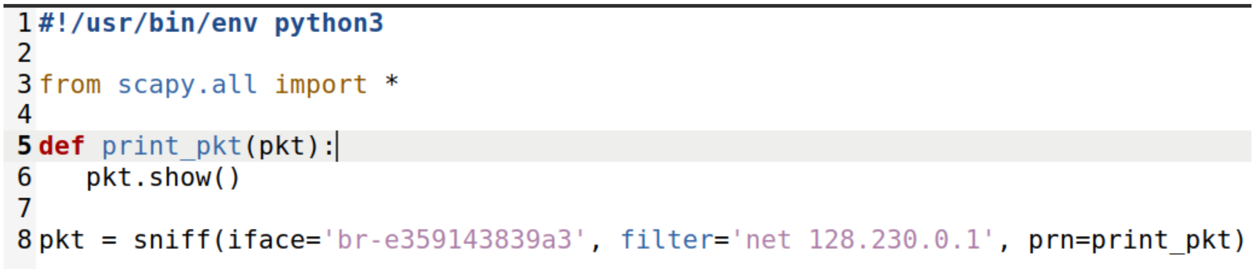
```
root@d222f48965d5:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
8cc77b22772d login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

If we go back to the attacker container:



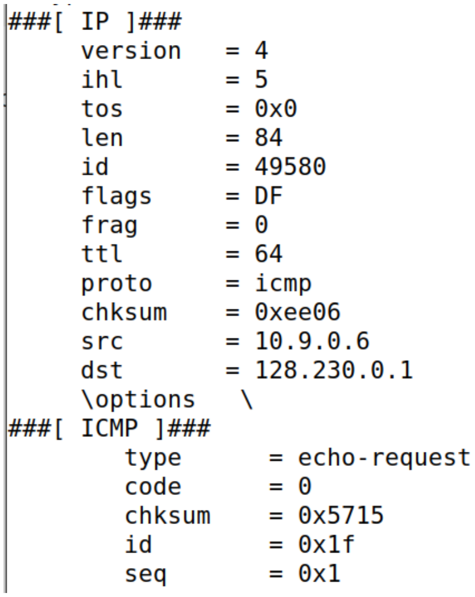
```
###[ TCP ]###
        sport     = 43426
        dport     = telnet
        seq       = 3595496047
        ack       = 1073547789
        dataofs   = 8
        reserved  = 0
        flags     = A
        window    = 501
        chksum    = 0x1443
        urgptr    = 0
        options   = [('NOP', None), ('NOP', None), ('Timestamp', (3774792441, 13
20081049))]
```

Subnets:



```python
#!/usr/bin/env python3

from scapy.all import *

def print_pkt(pkt):
   pkt.show()

pkt = sniff(iface='br-e359143839a3', filter='net 128.230.0.1', prn=print_pkt)
```

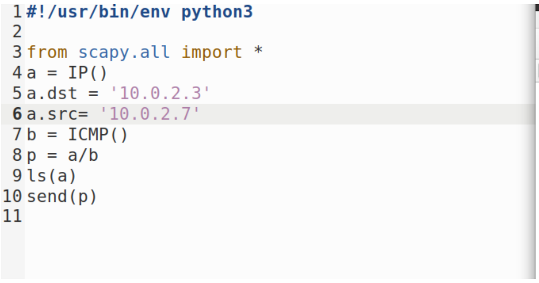
After running ./test.py in the attacker container, we can go back to hostB and then ping 128.230.0.1 which will give us the following result:



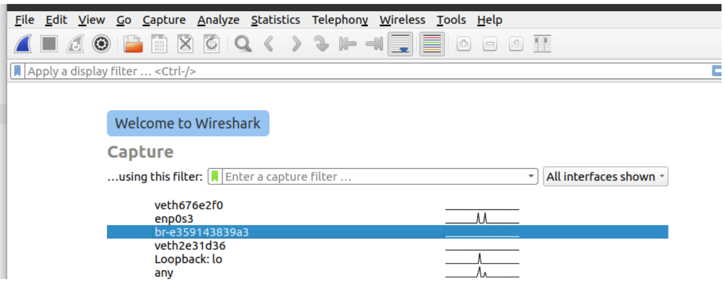
```
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 49580
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xee06
     src       = 10.9.0.6
     dst       = 128.230.0.1
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x5715
        id        = 0x1f
        seq       = 0x1
```

Task 1.2: Spoofing ICMP Packets

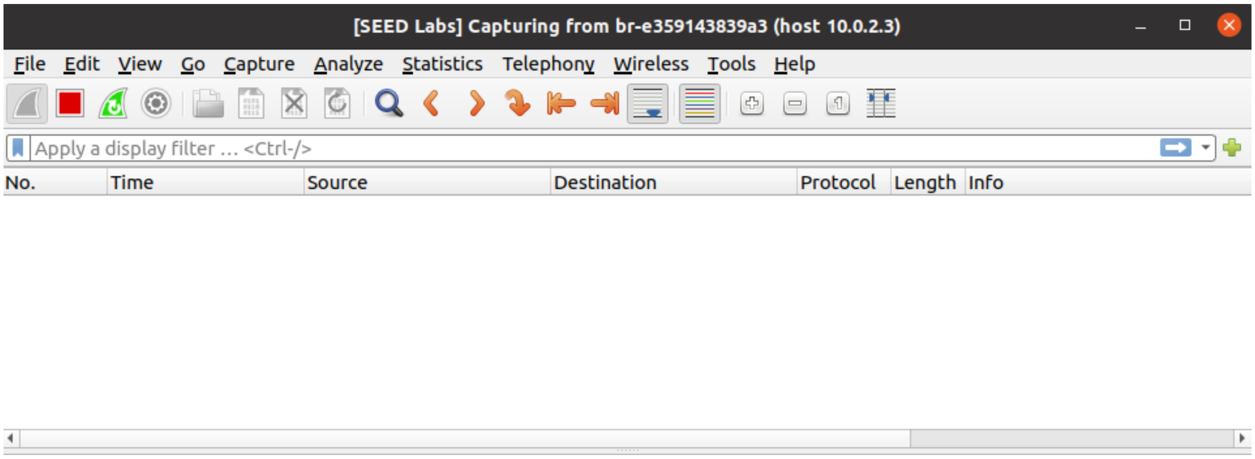
I added the code in a file where I also added the src below the dst and opened Wireshark, and we can see the interface name:



```
#!/usr/bin/env python3

from scapy.all import *
a = IP()
a.dst = '10.0.2.3'
a.src= '10.0.2.7'
b = ICMP()
p = a/b
ls(a)
send(p)

```



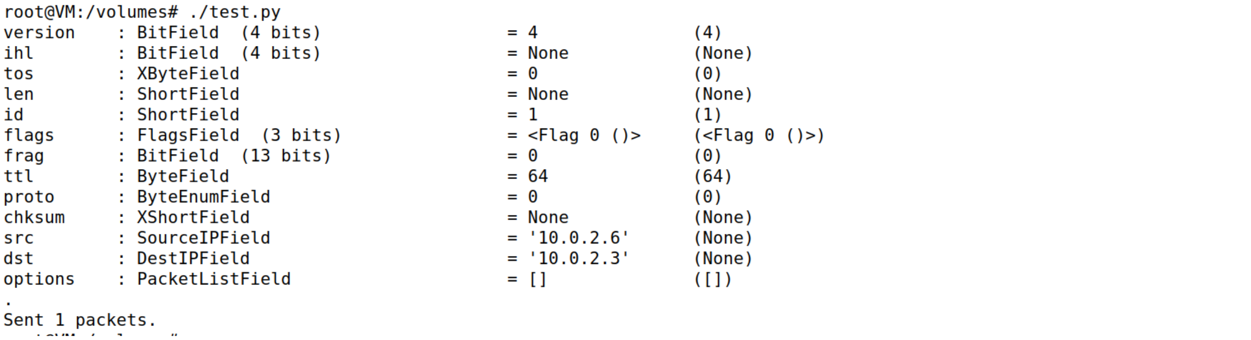
I imputed the host 10.0.2.3 to capture then hit enter in Wireshark



[SEED Labs] Capturing from br-e359143839a3 (host 10.0.2.3)

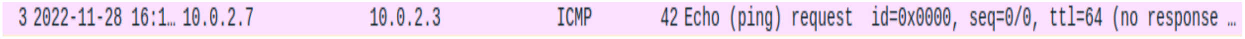
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
No.
Time
Source
Destination
Protocol
Length
Info

When we run ./test.py:



```
root@VM:/volumes# ./test.py
version    : BitField  (4 bits)                    = 4              (4)
ihl        : BitField  (4 bits)                    = None           (None)
tos        : XByteField                            = 0              (0)
len        : ShortField                            = None           (None)
id         : ShortField                            = 1              (1)
flags      : FlagsField  (3 bits)                  = <Flag 0 ()>    (<Flag 0 ()>)
frag       : BitField  (13 bits)                   = 0              (0)
ttl        : ByteField                             = 64             (64)
proto      : ByteEnumField                         = 0              (0)
chksum     : XShortField                           = None           (None)
src        : SourceIPField                         = '10.0.2.6'     (None)
dst        : DestIPField                           = '10.0.2.3'     (None)
options    : PacketListField                       = []             ([])
.
Sent 1 packets.
```

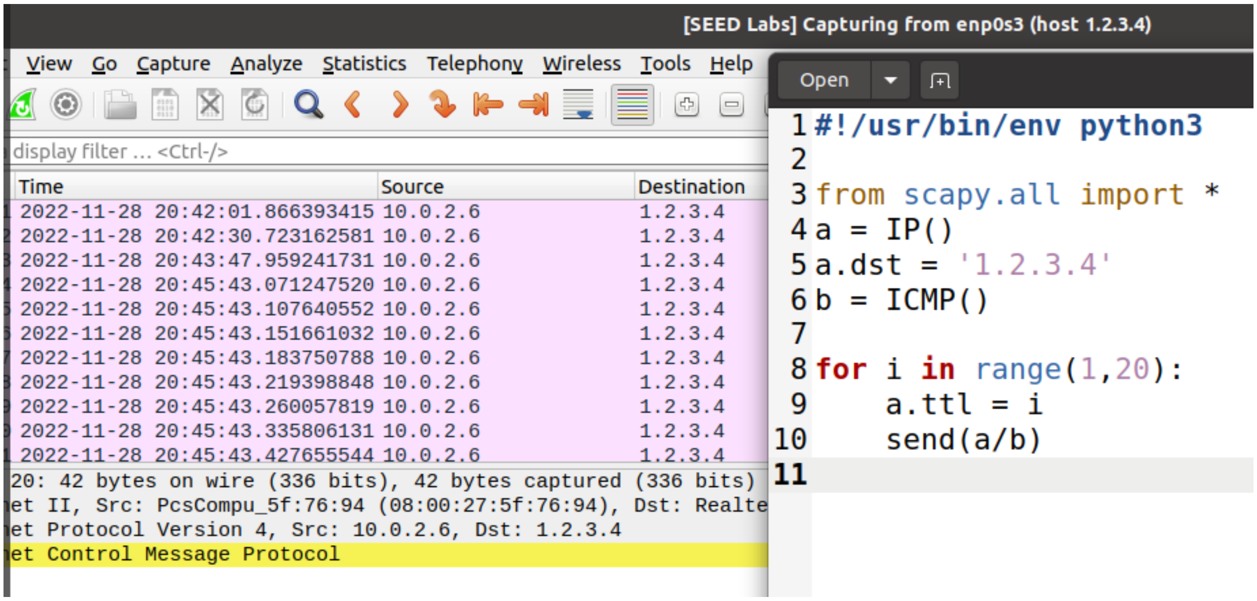
In wireshark we received the packet:



```
3 2022-11-28 16:1… 10.0.2.7          10.0.2.3          ICMP      42 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response …
```

Task 1.3: Traceroute

I ran the code in a for loop where I am basically sending with a TTL from 1 to 20:


[SEED Labs] Capturing from enp0s3 (host 1.2.3.4)
View Go Capture Analyze Statistics Telephony Wireless Tools Help
display filter … <Ctrl-/>
Open

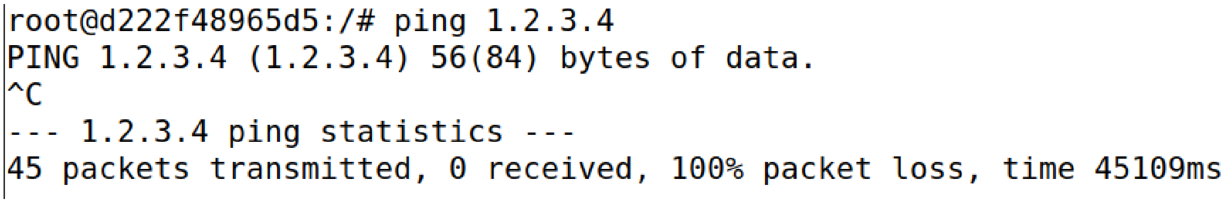
```
Time                          Source     Destination
2022-11-28 20:42:01.866393415 10.0.2.6   1.2.3.4
2022-11-28 20:42:30.723162581 10.0.2.6   1.2.3.4
2022-11-28 20:43:47.959241731 10.0.2.6   1.2.3.4
2022-11-28 20:45:43.071247520 10.0.2.6   1.2.3.4
2022-11-28 20:45:43.107640552 10.0.2.6   1.2.3.4
2022-11-28 20:45:43.151661032 10.0.2.6   1.2.3.4
2022-11-28 20:45:43.183750788 10.0.2.6   1.2.3.4
2022-11-28 20:45:43.219398848 10.0.2.6   1.2.3.4
2022-11-28 20:45:43.260057819 10.0.2.6   1.2.3.4
2022-11-28 20:45:43.335806131 10.0.2.6   1.2.3.4
2022-11-28 20:45:43.427655544 10.0.2.6   1.2.3.4
20: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
et II, Src: PcsCompu_5f:76:94 (08:00:27:5f:76:94), Dst: Realte
et Protocol Version 4, Src: 10.0.2.6, Dst: 1.2.3.4
et Control Message Protocol
```

```python
#!/usr/bin/env python3

from scapy.all import *
a = IP()
a.dst = '1.2.3.4'
b = ICMP()

for i in range(1,20):
    a.ttl = i
    send(a/b)

```

Task 1.4: Sniffing and-then Spoofing

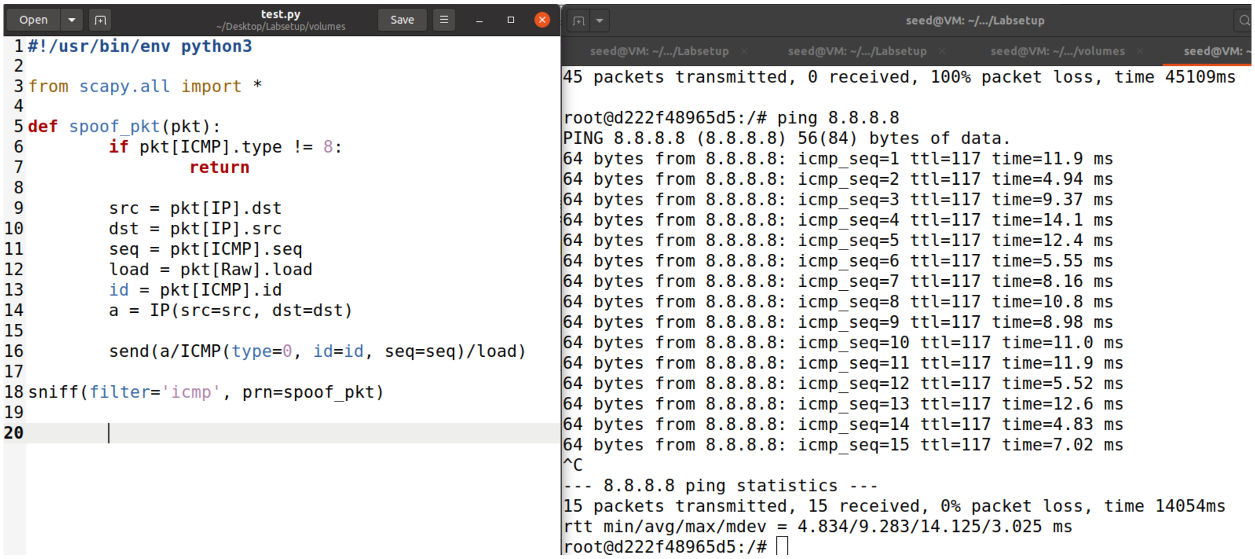
We create our own function by retrieving some crucial information for our purposes such as the src, dst, seq, load and id, which all can be retrieved from the IP, ICMP and Raw.

Pinging 1.2.3.4 from the user container has the following effect:



```
root@d222f48965d5:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
^C
--- 1.2.3.4 ping statistics ---
45 packets transmitted, 0 received, 100% packet loss, time 45109ms
```

However, pinging 8.8.8.8 from the user container gets us the following output: (output alongside the code)



```python
#!/usr/bin/env python3

from scapy.all import *

def spoof_pkt(pkt):
        if pkt[ICMP].type != 8:
                return

        src = pkt[IP].dst
        dst = pkt[IP].src
        seq = pkt[ICMP].seq
        load = pkt[Raw].load
        id = pkt[ICMP].id
        a = IP(src=src, dst=dst)

        send(a/ICMP(type=0, id=id, seq=seq)/load)

sniff(filter='icmp', prn=spoof_pkt)

```

```
45 packets transmitted, 0 received, 100% packet loss, time 45109ms

root@d222f48965d5:/# 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=117 time=11.9 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=117 time=4.94 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=117 time=9.37 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=117 time=14.1 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=117 time=12.4 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=117 time=5.55 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=117 time=8.16 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=117 time=10.8 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=117 time=8.98 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=117 time=11.0 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=117 time=11.9 ms
64 bytes from 8.8.8.8: icmp_seq=12 ttl=117 time=5.52 ms
64 bytes from 8.8.8.8: icmp_seq=13 ttl=117 time=12.6 ms
64 bytes from 8.8.8.8: icmp_seq=14 ttl=117 time=4.83 ms
64 bytes from 8.8.8.8: icmp_seq=15 ttl=117 time=7.02 ms
^C
--- 8.8.8.8 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14054ms
rtt min/avg/max/mdev = 4.834/9.283/14.125/3.025 ms
root@d222f48965d5:/# 
```