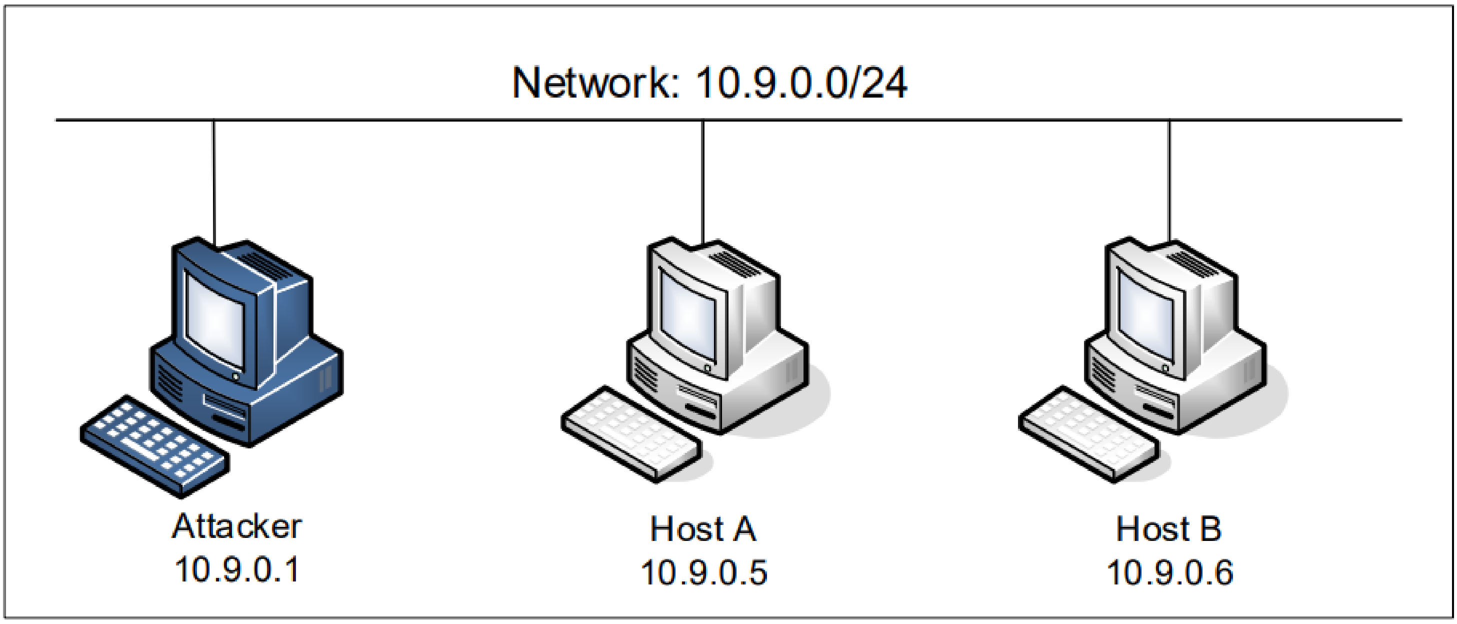
מטלה 6

סטודנט ראשון : אילן מאיר סופיר, ת"ז : 342615648

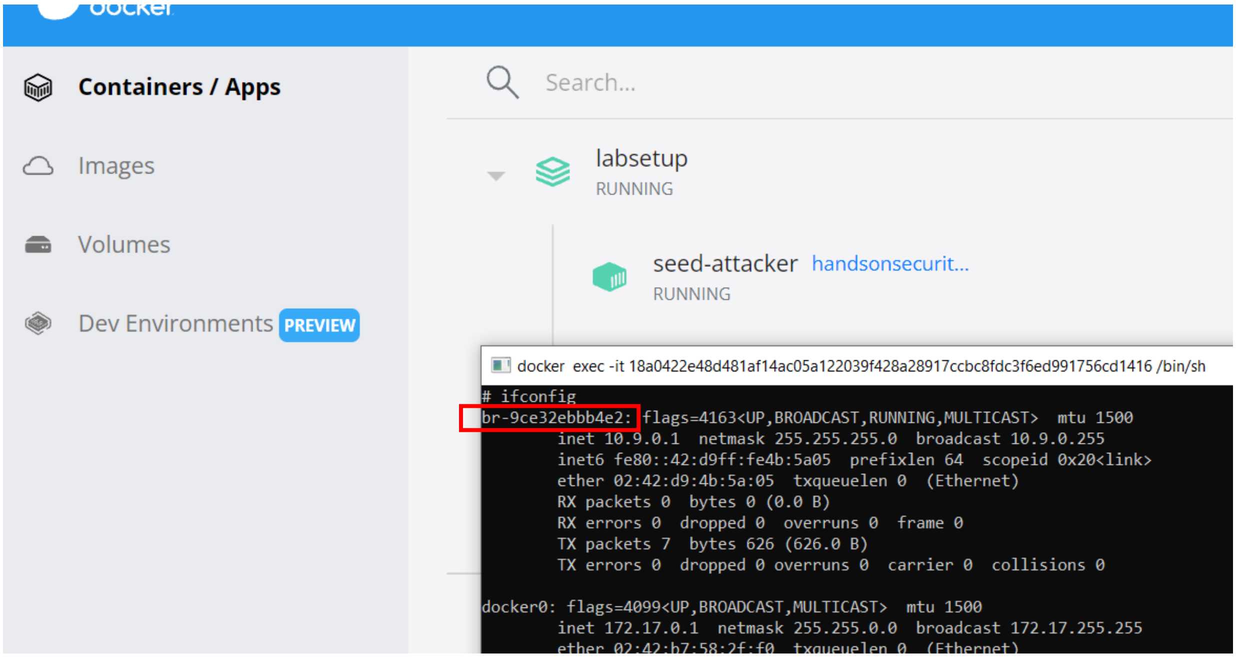
סטודנט שני : בן כהן , ת"ז : 207029786



Task 1.1: Sniffing Packets

Task1.1A :

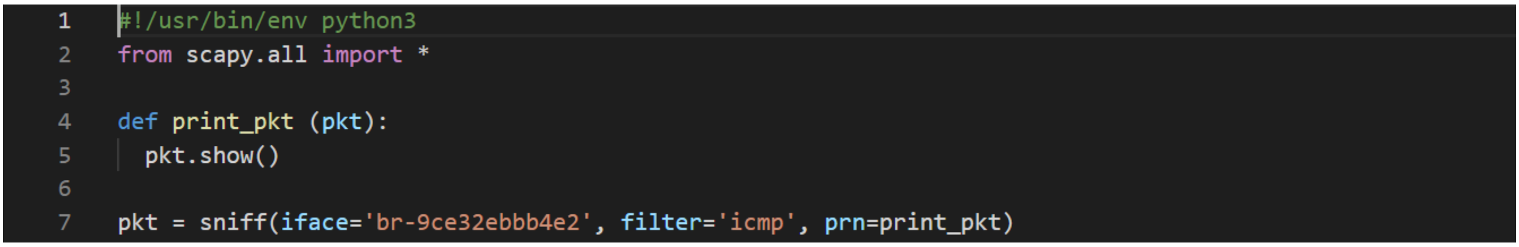
To know the network’s name, we use « ifconfig » on the attacker and then we have the network’s name : **br-9ce32ebbb4e2**.



docker exec -it 18a0422e48d481af14ac05a122039f428a28917ccbc8fdc3f6ed991756cd1416 /bin/sh

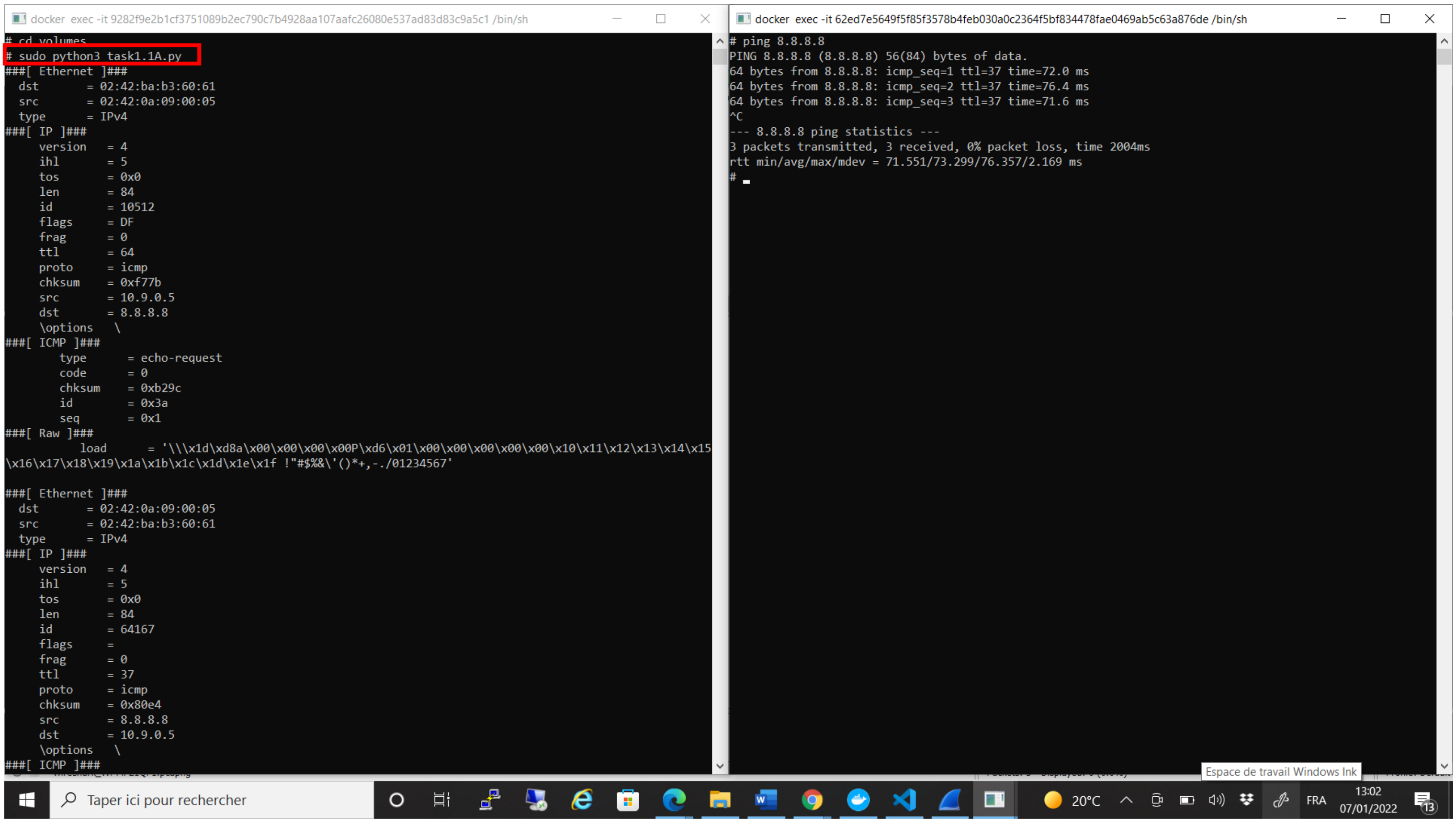
```
# ifconfig
br-9ce32ebbb4e2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:d9ff:fe4b:5a05  prefixlen 64  scopeid 0x20<link>
        ether 02:42:d9:4b:5a:05  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7  bytes 626 (626.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:b7:58:2f:f0  txqueuelen 0  (Ethernet)
```

Python code to sniff icmp packets :



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

def print_pkt (pkt):
    pkt.show()

pkt = sniff(iface='br-9ce32ebbb4e2', filter='icmp', prn=print_pkt)
```

With the root (sudo) :

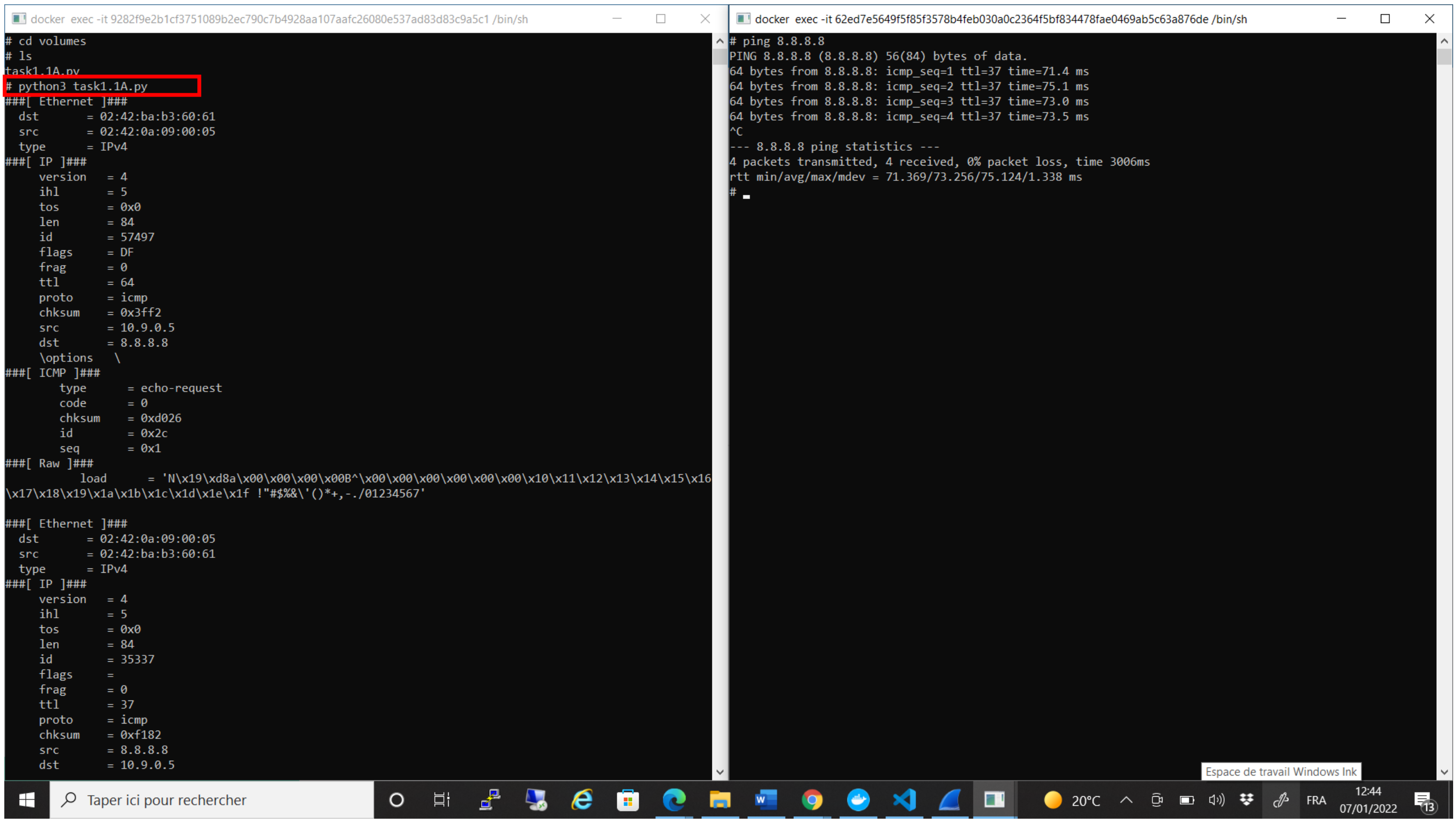

docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh
docker exec -it 62ed7e5649f5f85f3578b4feb030a0c2364f5bf834478fae0469ab5c63a876de /bin/sh
Espace de travail Windows Ink
Taper ici pour rechercher
20°C
FRA
13:02
07/01/2022

```
# cd volumes
# sudo python3 task1.1A.py
###[ Ethernet ]###
  dst       = 02:42:ba:b3:60:61
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 10512
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xf77b
     src       = 10.9.0.5
     dst       = 8.8.8.8
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xb29c
        id        = 0x3a
        seq       = 0x1
###[ Raw ]###
           load      = '\\\x1d\xd8a\x00\x00\x00\x00P\xd6\x01\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15
\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:ba:b3:60:61
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 64167
     flags     =
     frag      = 0
     ttl       = 37
     proto     = icmp
     chksum    = 0x80e4
     src       = 8.8.8.8
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
```

```
# 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=37 time=72.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=37 time=76.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=37 time=71.6 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 71.551/73.299/76.357/2.169 ms
#
```

All the ICMP packets sent with the command ping 8.8.8.8 (Google DNS) are sniffed successfully.

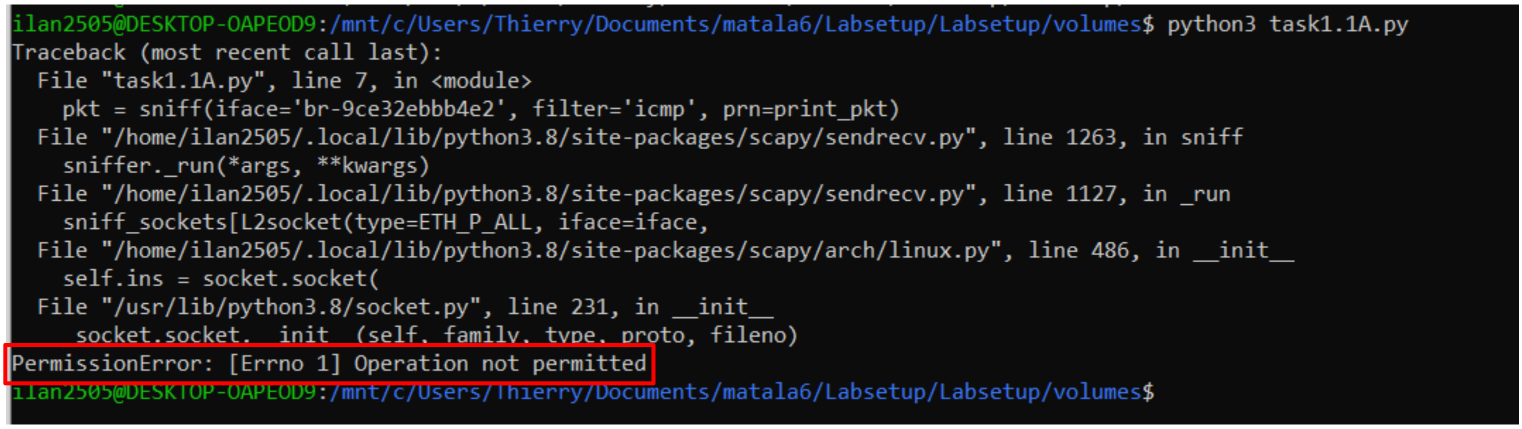
Without the root :


docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh
docker exec -it 62ed7e5649f5f85f3578b4feb030a0c2364f5bf834478fae0469ab5c63a876de /bin/sh
Espace de travail Windows Ink
Taper ici pour rechercher
20°C
FRA
12:44
07/01/2022
13

```
# cd volumes
# ls
task1.1A.py
# python3 task1.1A.py
###[ Ethernet ]###
  dst       = 02:42:ba:b3:60:61
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 57497
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x3ff2
     src       = 10.9.0.5
     dst       = 8.8.8.8
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xd026
        id        = 0x2c
        seq       = 0x1
###[ Raw ]###
           load      = 'N\x19\xd8a\x00\x00\x00\x00B^\x00\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16
\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:ba:b3:60:61
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 35337
     flags     =
     frag      = 0
     ttl       = 37
     proto     = icmp
     chksum    = 0xf182
     src       = 8.8.8.8
     dst       = 10.9.0.5
```

```
# 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=37 time=71.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=37 time=75.1 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=37 time=73.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=37 time=73.5 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 71.369/73.256/75.124/1.338 ms
# 
```

We can see that the python code task1.1A.py works, but in the ubuntu terminal when I run the code I have :



```
ilan2505@DESKTOP-OAPEOD9:/mnt/c/Users/Thierry/Documents/matala6/Labsetup/Labsetup/volumes$ python3 task1.1A.py
Traceback (most recent call last):
  File "task1.1A.py", line 7, in <module>
    pkt = sniff(iface='br-9ce32ebbb4e2', filter='icmp', prn=print_pkt)
  File "/home/ilan2505/.local/lib/python3.8/site-packages/scapy/sendrecv.py", line 1263, in sniff
    sniffer._run(*args, **kwargs)
  File "/home/ilan2505/.local/lib/python3.8/site-packages/scapy/sendrecv.py", line 1127, in _run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/home/ilan2505/.local/lib/python3.8/site-packages/scapy/arch/linux.py", line 486, in __init__
    self.ins = socket.socket(
  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
ilan2505@DESKTOP-OAPEOD9:/mnt/c/Users/Thierry/Documents/matala6/Labsetup/Labsetup/volumes$
```

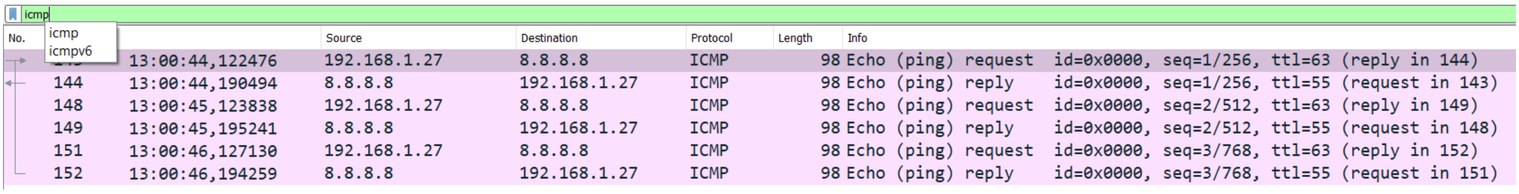
We have this message error : PermissionError: [Errno 1] Operation not permitted.

Why in the docker it’s work for the two mode : with and without the root ? I think it’s beacause when i wrote the command « sudo » for the first time, and then we don’t need to write « sudo » each time.

Captures Wireshark :

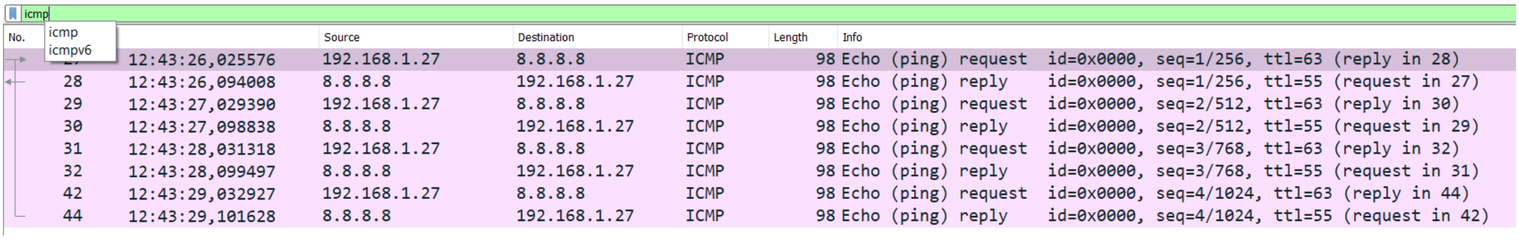
I work with the docker in Windows. So in Wireshark I don't know why but I have the IP adress 192.168.1.27 and not the IP adress 10.9.0.5 which is the adress of the HostA.

With sudo : (3 pings)



```
No.                         Source          Destination     Protocol  Length  Info
143     13:00:44,122476     192.168.1.27    8.8.8.8         ICMP      98 Echo (ping) request  id=0x0000, seq=1/256, ttl=63 (reply in 144)
144     13:00:44,190494     8.8.8.8         192.168.1.27    ICMP      98 Echo (ping) reply    id=0x0000, seq=1/256, ttl=55 (request in 143)
148     13:00:45,123838     192.168.1.27    8.8.8.8         ICMP      98 Echo (ping) request  id=0x0000, seq=2/512, ttl=63 (reply in 149)
149     13:00:45,195241     8.8.8.8         192.168.1.27    ICMP      98 Echo (ping) reply    id=0x0000, seq=2/512, ttl=55 (request in 148)
151     13:00:46,127130     192.168.1.27    8.8.8.8         ICMP      98 Echo (ping) request  id=0x0000, seq=3/768, ttl=63 (reply in 152)
152     13:00:46,194259     8.8.8.8         192.168.1.27    ICMP      98 Echo (ping) reply    id=0x0000, seq=3/768, ttl=55 (request in 151)
```

Without sudo : (4 pings)



```
No.                         Source          Destination     Protocol  Length  Info
27      12:43:26,025576     192.168.1.27    8.8.8.8         ICMP      98 Echo (ping) request  id=0x0000, seq=1/256, ttl=63 (reply in 28)
28      12:43:26,094008     8.8.8.8         192.168.1.27    ICMP      98 Echo (ping) reply    id=0x0000, seq=1/256, ttl=55 (request in 27)
29      12:43:27,029390     192.168.1.27    8.8.8.8         ICMP      98 Echo (ping) request  id=0x0000, seq=2/512, ttl=63 (reply in 30)
30      12:43:27,098838     8.8.8.8         192.168.1.27    ICMP      98 Echo (ping) reply    id=0x0000, seq=2/512, ttl=55 (request in 29)
31      12:43:28,031318     192.168.1.27    8.8.8.8         ICMP      98 Echo (ping) request  id=0x0000, seq=3/768, ttl=63 (reply in 32)
32      12:43:28,099497     8.8.8.8         192.168.1.27    ICMP      98 Echo (ping) reply    id=0x0000, seq=3/768, ttl=55 (request in 31)
42      12:43:29,032927     192.168.1.27    8.8.8.8         ICMP      98 Echo (ping) request  id=0x0000, seq=4/1024, ttl=63 (reply in 44)
44      12:43:29,101628     8.8.8.8         192.168.1.27    ICMP      98 Echo (ping) reply    id=0x0000, seq=4/1024, ttl=55 (request in 42)
```

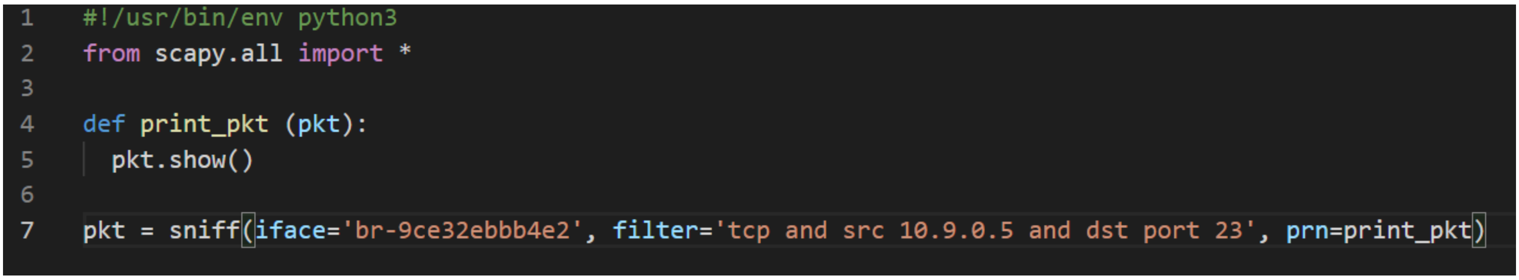
Task1.1B :

- Capture only the ICMP packet :

-> see the task 1.1A.

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

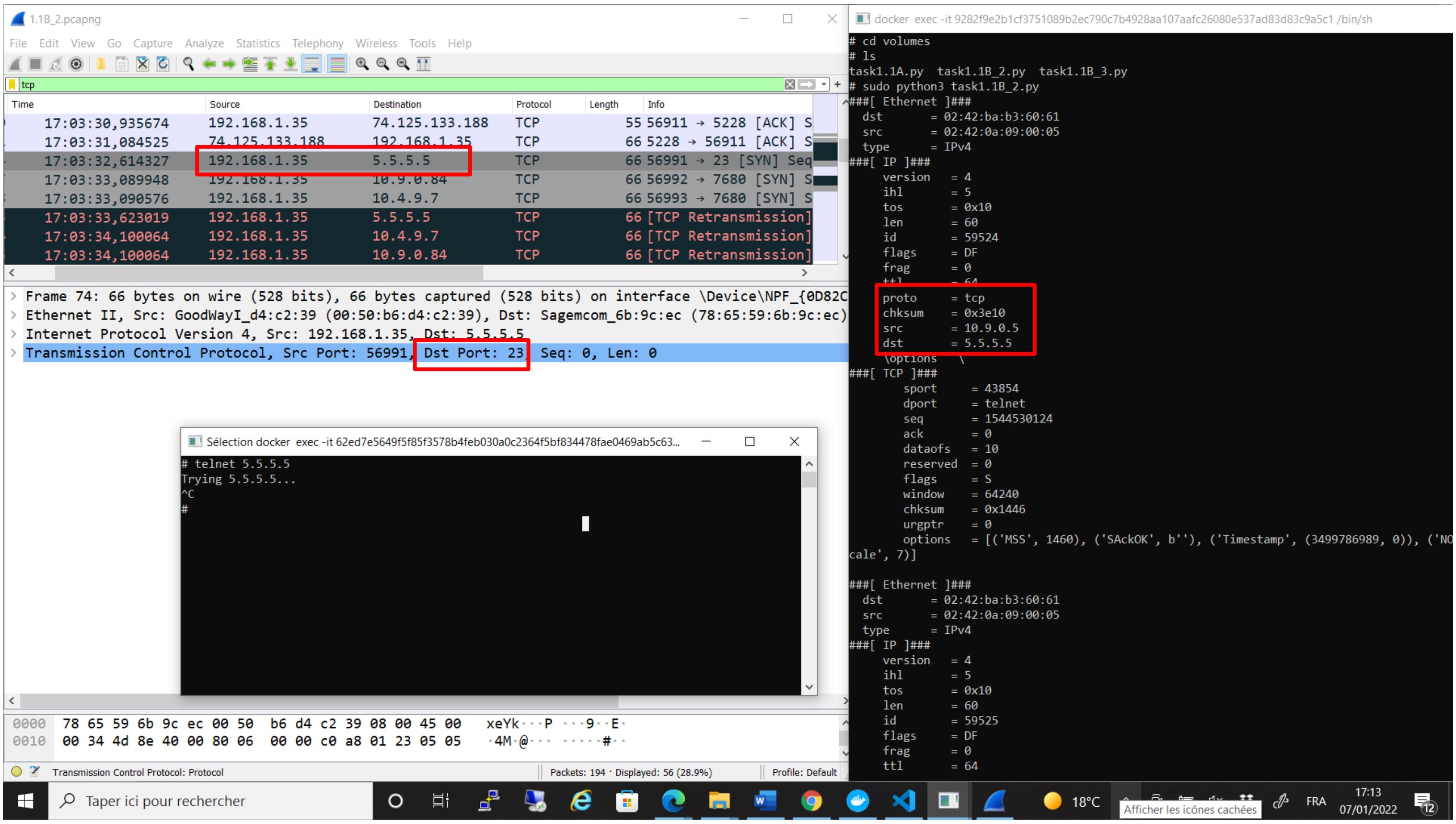
Code python :



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

def print_pkt (pkt):
    pkt.show()

pkt = sniff(iface='br-9ce32ebbb4e2', filter='tcp and src 10.9.0.5 and dst port 23', prn=print_pkt)
```

telnet : works on port 23, telnet is a tool for working exclusively through the terminal, it is a tool for connecting with a far system. It allows you to edit documents, run features, check whether a particular port in the remote system is available or not.

Wireshark capture 1.1B_2 :



1.1B_2.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp

```
Time              Source            Destination       Protocol  Length  Info
17:03:30,935674   192.168.1.35      74.125.133.188    TCP       55 56911 → 5228 [ACK] S
17:03:31,084525   74.125.133.188    192.168.1.35      TCP       66 5228 → 56911 [ACK] S
17:03:32,614327   192.168.1.35      5.5.5.5           TCP       66 56991 → 23 [SYN] Seq
17:03:33,089948   192.168.1.35      10.9.0.84         TCP       66 56992 → 7680 [SYN] S
17:03:33,090576   192.168.1.35      10.4.9.7          TCP       66 56993 → 7680 [SYN] S
17:03:33,623019   192.168.1.35      5.5.5.5           TCP       66 [TCP Retransmission]
17:03:34,100064   192.168.1.35      10.4.9.7          TCP       66 [TCP Retransmission]
17:03:34,100064   192.168.1.35      10.9.0.84         TCP       66 [TCP Retransmission]
```

```
> Frame 74: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{0D82C
> Ethernet II, Src: GoodWayI_d4:c2:39 (00:50:b6:d4:c2:39), Dst: Sagemcom_6b:9c:ec (78:65:59:6b:9c:ec)
> Internet Protocol Version 4, Src: 192.168.1.35, Dst: 5.5.5.5
> Transmission Control Protocol, Src Port: 56991, Dst Port: 23, Seq: 0, Len: 0
```

```
0000  78 65 59 6b 9c ec 00 50  b6 d4 c2 39 08 00 45 00   xeYk···P ···9··E·
0010  00 34 4d 8e 40 00 80 06  00 00 c0 a8 01 23 05 05   ·4M·@··· ·····#··
```

Transmission Control Protocol: Protocol
Packets: 194 · Displayed: 56 (28.9%)
Profile: Default
Sélection docker exec -it 62ed7e5649f5f85f3578b4feb030a0c2364f5bf834478fae0469ab5c63...

```
# telnet 5.5.5.5
Trying 5.5.5.5...
^C
#
```

docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh

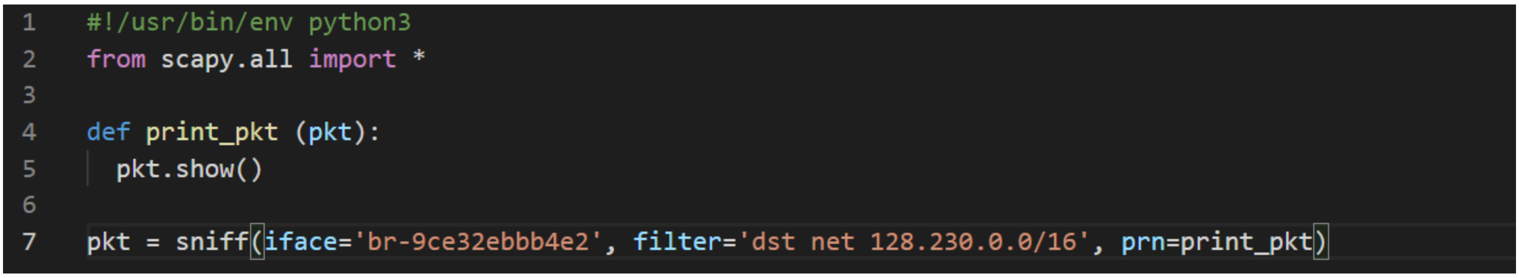
```
# cd volumes
# ls
task1.1A.py  task1.1B_2.py  task1.1B_3.py
# sudo python3 task1.1B_2.py
###[ Ethernet ]###
  dst       = 02:42:ba:b3:60:61
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 60
     id        = 59524
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x3e10
     src       = 10.9.0.5
     dst       = 5.5.5.5
     \options   \
###[ TCP ]###
        sport     = 43854
        dport     = telnet
        seq       = 1544530124
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x1446
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (3499786989, 0)), ('NO
cale', 7)]

###[ Ethernet ]###
  dst       = 02:42:ba:b3:60:61
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 60
     id        = 59525
     flags     = DF
     frag      = 0
     ttl       = 64
```

- 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 :

With ping from 10.9.0.5 to one of the subnet addresses I mentioned:

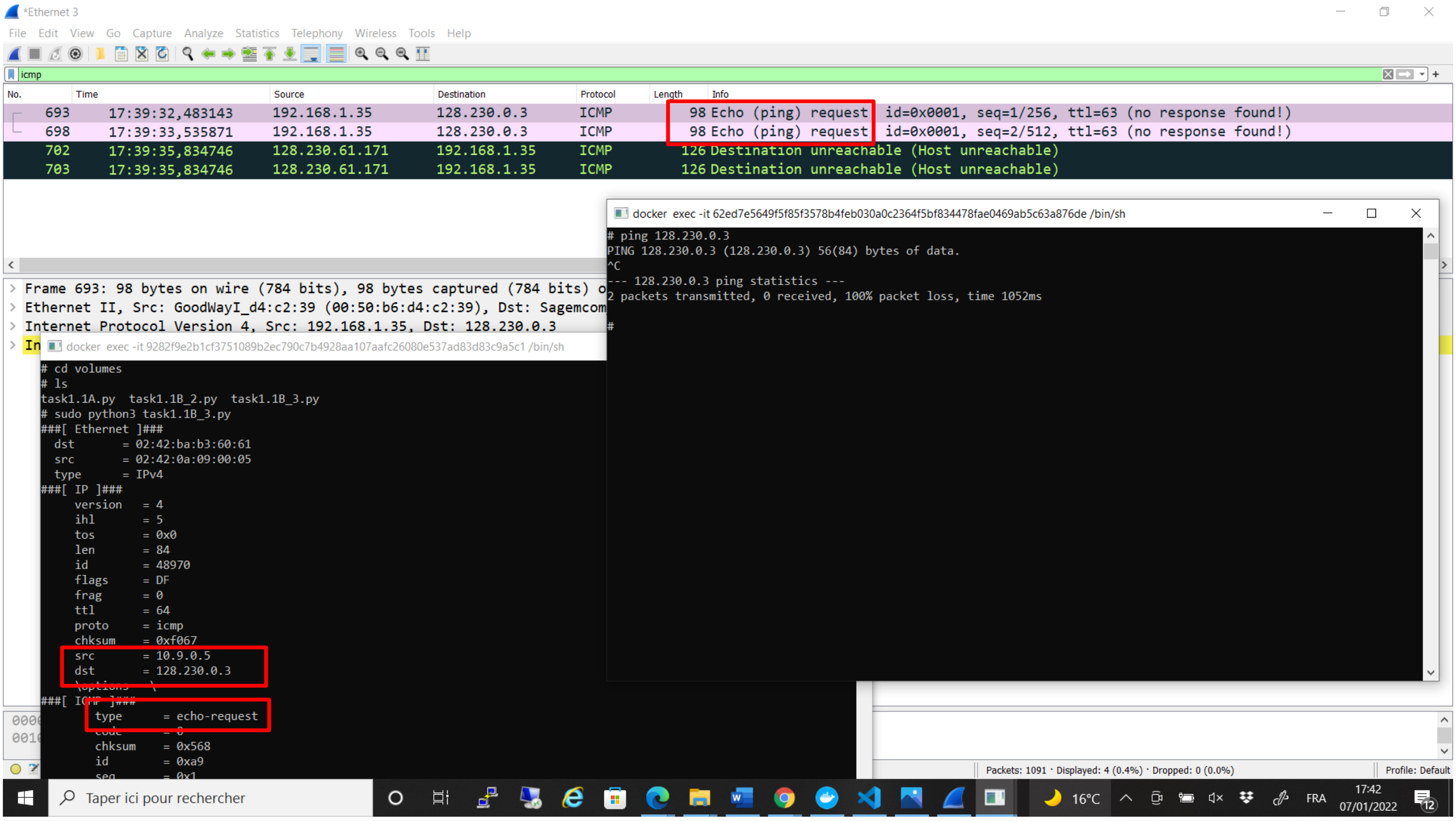
I chose to pingto 128.230.0.3, there is no answer from this address, these are just request-echo (look wireshark) icmp requests.

Python code :



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

def print_pkt (pkt):
    pkt.show()

pkt = sniff(iface='br-9ce32ebbb4e2', filter='dst net 128.230.0.0/16', prn=print_pkt)
```

Wireshark capture 1.1B_3 :



*Ethernet 3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmp

```
No.   Time             Source          Destination   Protocol  Length  Info
693   17:39:32,483143  192.168.1.35    128.230.0.3   ICMP      98 Echo (ping) request  id=0x0001, seq=1/256, ttl=63 (no response found!)
698   17:39:33,535871  192.168.1.35    128.230.0.3   ICMP      98 Echo (ping) request  id=0x0001, seq=2/512, ttl=63 (no response found!)
702   17:39:35,834746  128.230.61.171  192.168.1.35  ICMP      126 Destination unreachable (Host unreachable)
703   17:39:35,834746  128.230.61.171  192.168.1.35  ICMP      126 Destination unreachable (Host unreachable)
```

```
> Frame 693: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) o
> Ethernet II, Src: GoodWayI_d4:c2:39 (00:50:b6:d4:c2:39), Dst: Sagemcom
> Internet Protocol Version 4, Src: 192.168.1.35, Dst: 128.230.0.3
> In
```

docker exec -it 62ed7e5649f5f85f3578b4feb030a0c2364f5bf834478fae0469ab5c63a876de /bin/sh

```
# ping 128.230.0.3
PING 128.230.0.3 (128.230.0.3) 56(84) bytes of data.
^C
--- 128.230.0.3 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1052ms

#
```

docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh

```
# cd volumes
# ls
task1.1A.py  task1.1B_2.py  task1.1B_3.py
# sudo python3 task1.1B_3.py
###[ Ethernet ]###
  dst       = 02:42:ba:b3:60:61
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 48970
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xf067
     src       = 10.9.0.5
     dst       = 128.230.0.3
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x568
        id        = 0xa9
        seq       = 0x1
```

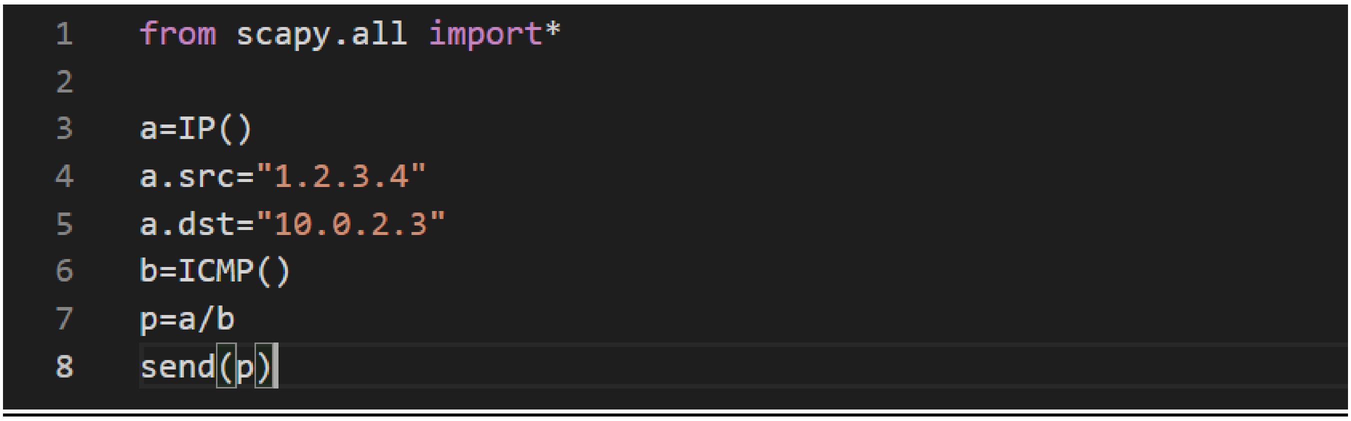
Packets: 1091 · Displayed: 4 (0.4%) · Dropped: 0 (0.0%)

Profile: Default

Task 1.2: Spoofing ICMP Packets

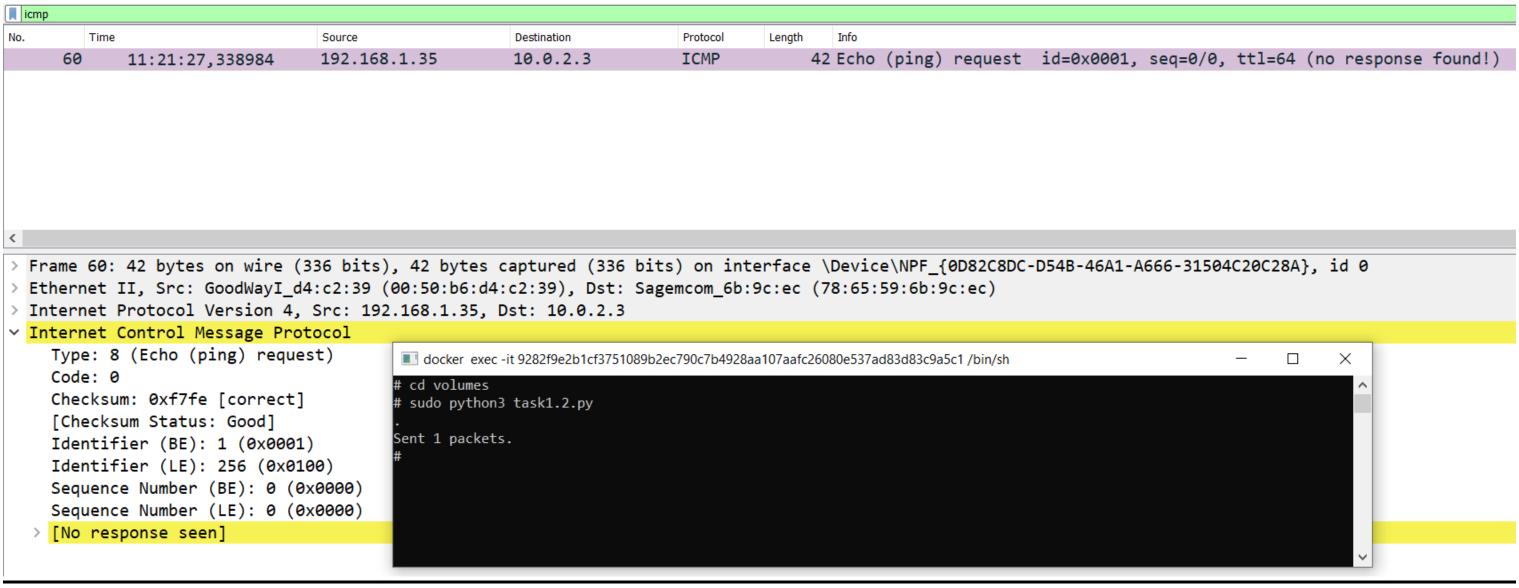
In this task we were asked to create our own fact and send it to the address we decide and from the address we decide, I chose the address given in the example of the exercise: 10.0.2.3, the packet does come from an IP (1.2.3.4) that does not exist in the subnet of the containers. We have only ICMP request.

Python code :



```python
from scapy.all import*

a=IP()
a.src="1.2.3.4"
a.dst="10.0.2.3"
b=ICMP()
p=a/b
send(p)
```

Wireshark capture 1.2 :



No.	Time	Source	Destination	Protocol	Length	Info
60	11:21:27,338984	192.168.1.35	10.0.2.3	ICMP	42	Echo (ping) request id=0x0001, seq=0/0, ttl=64 (no response found!)

```
> Frame 60: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{0D82C8DC-D54B-46A1-A666-31504C20C28A}, id 0
> Ethernet II, Src: GoodWayI_d4:c2:39 (00:50:b6:d4:c2:39), Dst: Sagemcom_6b:9c:ec (78:65:59:6b:9c:ec)
> Internet Protocol Version 4, Src: 192.168.1.35, Dst: 10.0.2.3
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xf7fe [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 0 (0x0000)
    Sequence Number (LE): 0 (0x0000)
  > [No response seen]
```

docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh

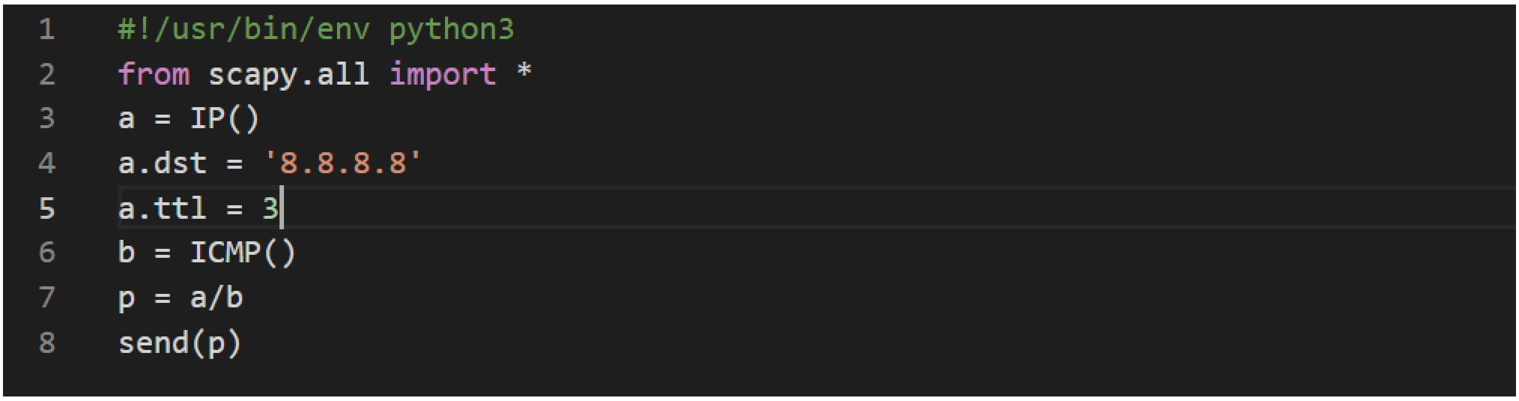
```
# cd volumes
# sudo python3 task1.2.py
.
Sent 1 packets.
#
```

Task 1.3: Traceroute

In this question we were asked to know how many routers move from our personal network to a specific IP address when sending Ping to this address. We checked this by sending Ping to google's address (8.8.8.8) and found that as much as TTL (To Time Live) Larger. This way you can get to the router further away until you reach the desired address.

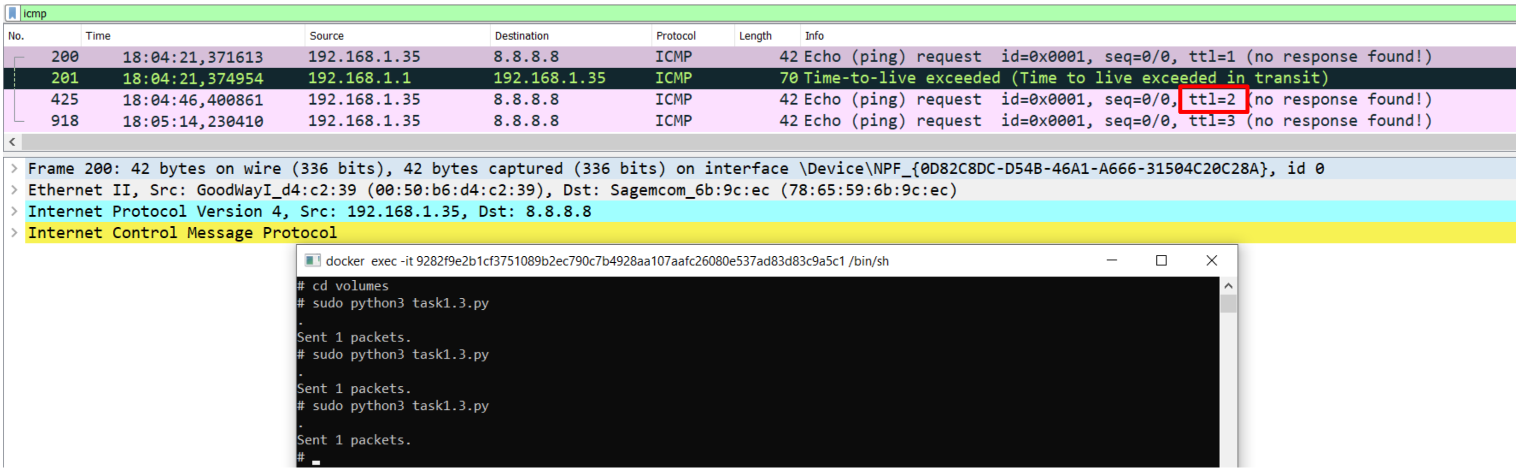
From the picture you can see that there were 2 transitions between routers until we reached the google's address.

Python code :



```python
#!/usr/bin/env python3
from scapy.all import *
a = IP()
a.dst = '8.8.8.8'
a.ttl = 3
b = ICMP()
p = a/b
send(p)
```

Wireshark capture 1.3 :



```
icmp
No.   Time              Source         Destination    Protocol  Length  Info
200   18:04:21,371613   192.168.1.35   8.8.8.8        ICMP      42 Echo (ping) request  id=0x0001, seq=0/0, ttl=1 (no response found!)
201   18:04:21,374954   192.168.1.1    192.168.1.35   ICMP      70 Time-to-live exceeded (Time to live exceeded in transit)
425   18:04:46,400861   192.168.1.35   8.8.8.8        ICMP      42 Echo (ping) request  id=0x0001, seq=0/0, ttl=2 (no response found!)
918   18:05:14,230410   192.168.1.35   8.8.8.8        ICMP      42 Echo (ping) request  id=0x0001, seq=0/0, ttl=3 (no response found!)

> Frame 200: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{0D82C8DC-D54B-46A1-A666-31504C20C28A}, id 0
> Ethernet II, Src: GoodWayI_d4:c2:39 (00:50:b6:d4:c2:39), Dst: Sagemcom_6b:9c:ec (78:65:59:6b:9c:ec)
> Internet Protocol Version 4, Src: 192.168.1.35, Dst: 8.8.8.8
> Internet Control Message Protocol
```

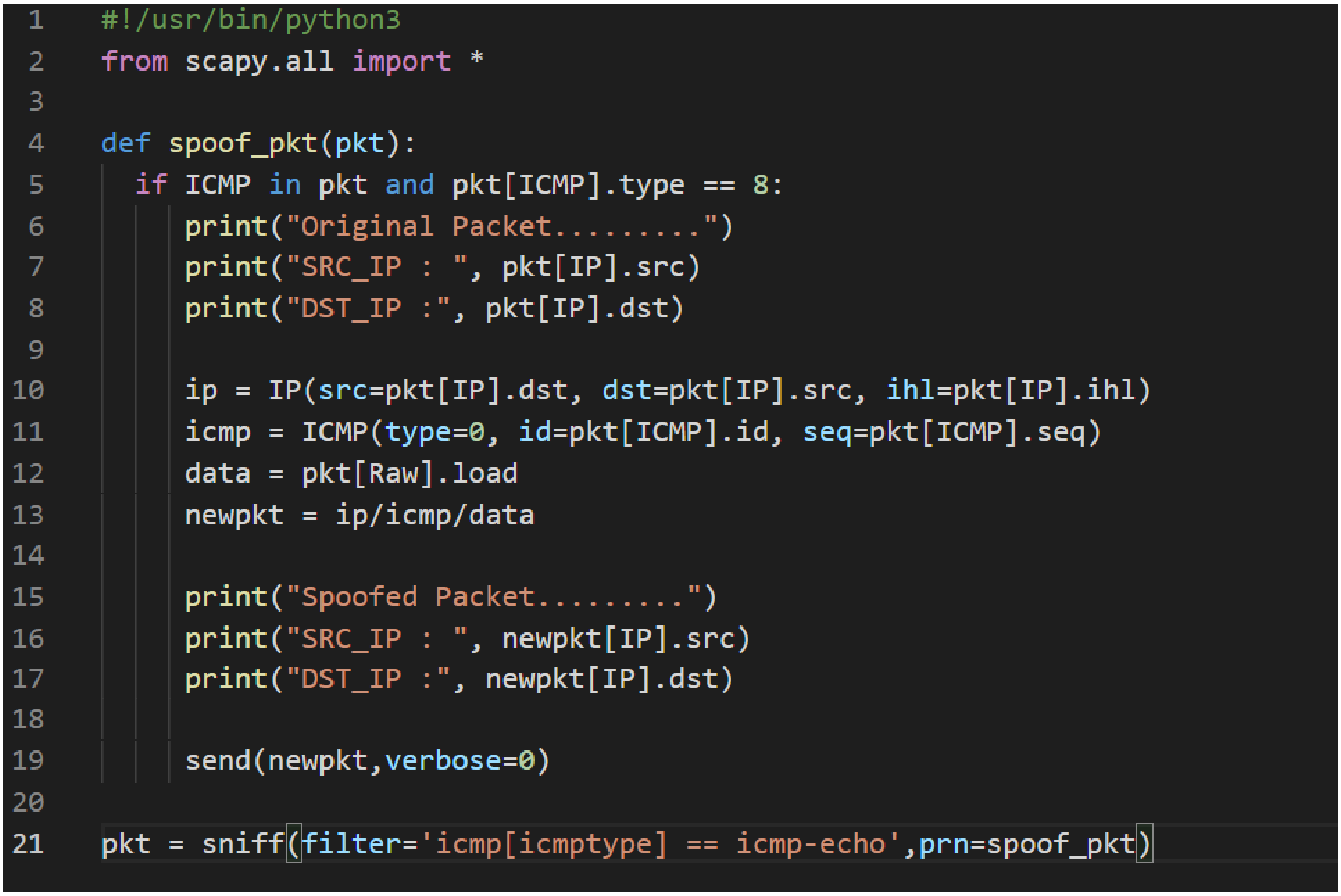
docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh

```
# cd volumes
# sudo python3 task1.3.py
.
Sent 1 packets.
# sudo python3 task1.3.py
.
Sent 1 packets.
# sudo python3 task1.3.py
.
Sent 1 packets.
#
```

Task 1.4: Sniffing and-then Spoofing

To answer this question we have developed two computers that simulate attack and user validity (the victim). The attacker runs the program to steal information from the victim.
At first the attacker pings to two non-existent addresses (1.2.3.4 and 10.9.0.99), the attacker branches the passing ICMP facts on the network, and sends the attacker a reply message from the same ping as if the address exists.
In addition, the attacker also sends a Ping to an existing address ("8.8.8.8)" which sends him a reply message back. The victim i think received a reply from "8.8.8.8" but the truth is that the attacker was the one who sent the reply message. An address that already exists on the Internet, a reply will be sent twice - both from the attacker and from the address itself (the original), can be see it in the image below - a message will be printed for us (DUP !) at the end of the double message line that alerts the user that this fact is sent from two different places.

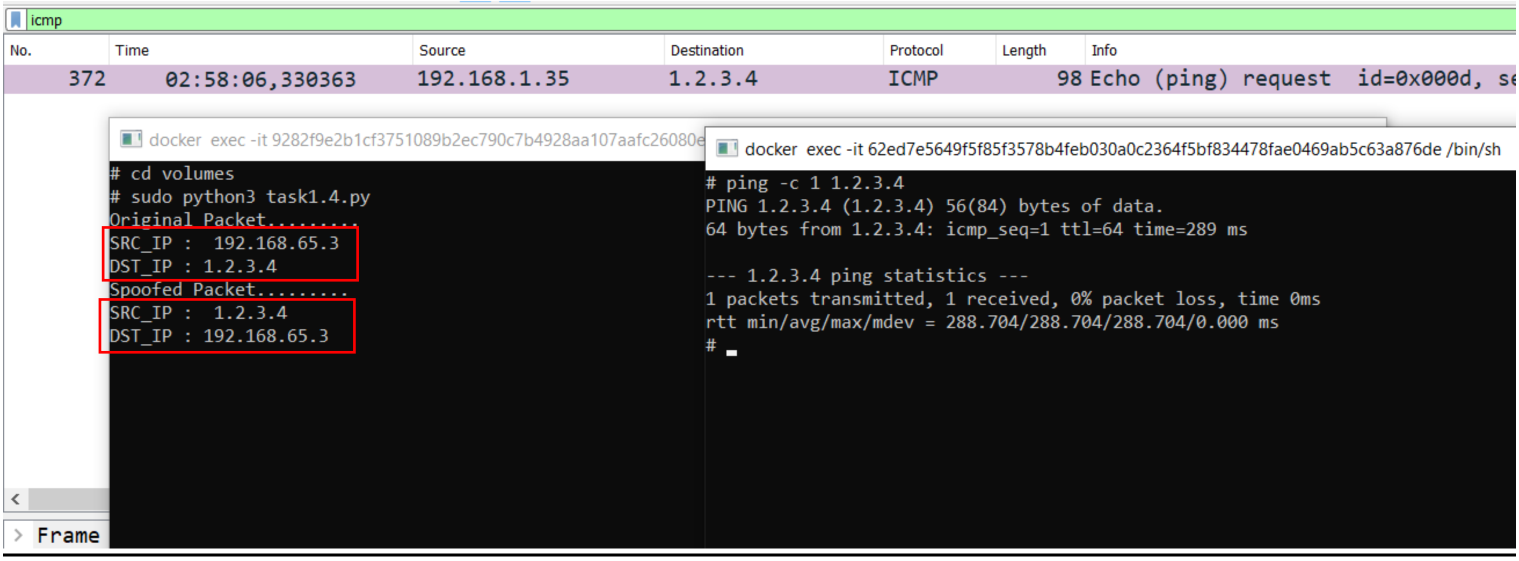
Python code :



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

def spoof_pkt(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.........")
        print("SRC_IP : ", pkt[IP].src)
        print("DST_IP :", pkt[IP].dst)

        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        data = pkt[Raw].load
        newpkt = ip/icmp/data

        print("Spoofed Packet.........")
        print("SRC_IP : ", newpkt[IP].src)
        print("DST_IP :", newpkt[IP].dst)

        send(newpkt,verbose=0)

pkt = sniff(filter='icmp[icmptype] == icmp-echo',prn=spoof_pkt)
```

1.2.3.4 :

We can see that in wireshark (1.4_1) i don’t have a reply but in my terminal yes.



icmp

No.	Time	Source	Destination	Protocol	Length	Info
372	02:58:06,330363	192.168.1.35	1.2.3.4	ICMP	98	Echo (ping) request id=0x000d, se

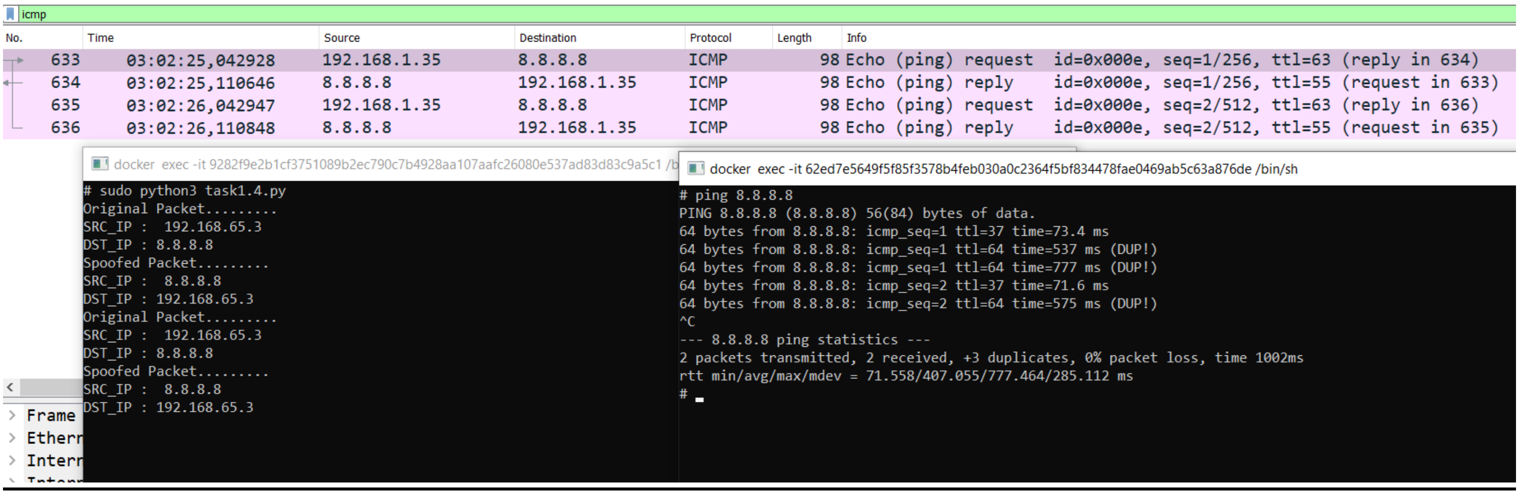
docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e

```
# cd volumes
# sudo python3 task1.4.py
Original Packet.........
SRC_IP :  192.168.65.3
DST_IP : 1.2.3.4
Spoofed Packet.........
SRC_IP :  1.2.3.4
DST_IP : 192.168.65.3
```

docker exec -it 62ed7e5649f5f85f3578b4feb030a0c2364f5bf834478fae0469ab5c63a876de /bin/sh

```
# ping -c 1 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=289 ms

--- 1.2.3.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 288.704/288.704/288.704/0.000 ms
# 
```

8.8.8.8 :



icmp

No.	Time	Source	Destination	Protocol	Length	Info
633	03:02:25,042928	192.168.1.35	8.8.8.8	ICMP	98	Echo (ping) request id=0x000e, seq=1/256, ttl=63 (reply in 634)
634	03:02:25,110646	8.8.8.8	192.168.1.35	ICMP	98	Echo (ping) reply id=0x000e, seq=1/256, ttl=55 (request in 633)
635	03:02:26,042947	192.168.1.35	8.8.8.8	ICMP	98	Echo (ping) request id=0x000e, seq=2/512, ttl=63 (reply in 636)
636	03:02:26,110848	8.8.8.8	192.168.1.35	ICMP	98	Echo (ping) reply id=0x000e, seq=2/512, ttl=55 (request in 635)

docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /b

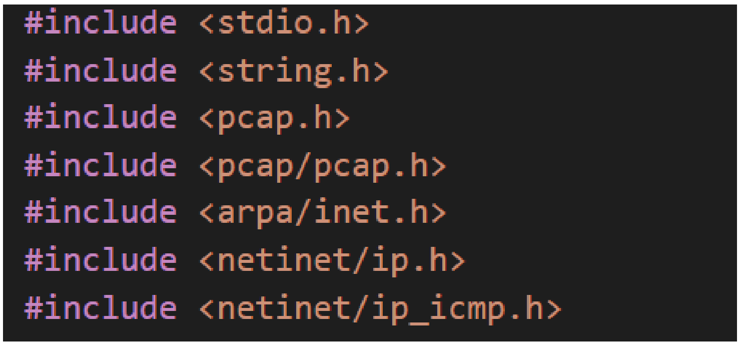
```
# sudo python3 task1.4.py
Original Packet.........
SRC_IP :  192.168.65.3
DST_IP : 8.8.8.8
Spoofed Packet.........
SRC_IP :  8.8.8.8
DST_IP : 192.168.65.3
Original Packet.........
SRC_IP :  192.168.65.3
DST_IP : 8.8.8.8
Spoofed Packet.........
SRC_IP :  8.8.8.8
DST_IP : 192.168.65.3
```

docker exec -it 62ed7e5649f5f85f3578b4feb030a0c2364f5bf834478fae0469ab5c63a876de /bin/sh

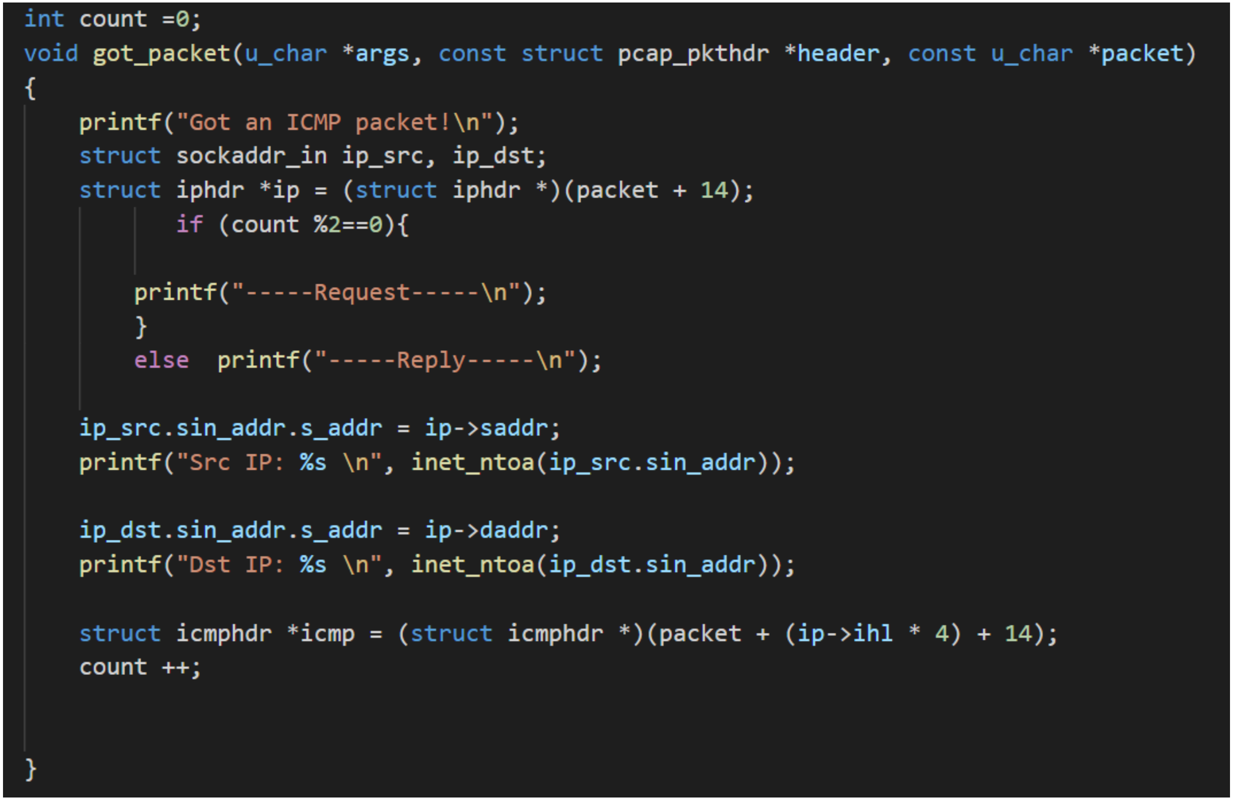
```
# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=37 time=73.4 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=537 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=777 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=37 time=71.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=575 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, +3 duplicates, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 71.558/407.055/777.464/285.112 ms
# 
```

Task 2.1A: Understanding How a Sniffer Works

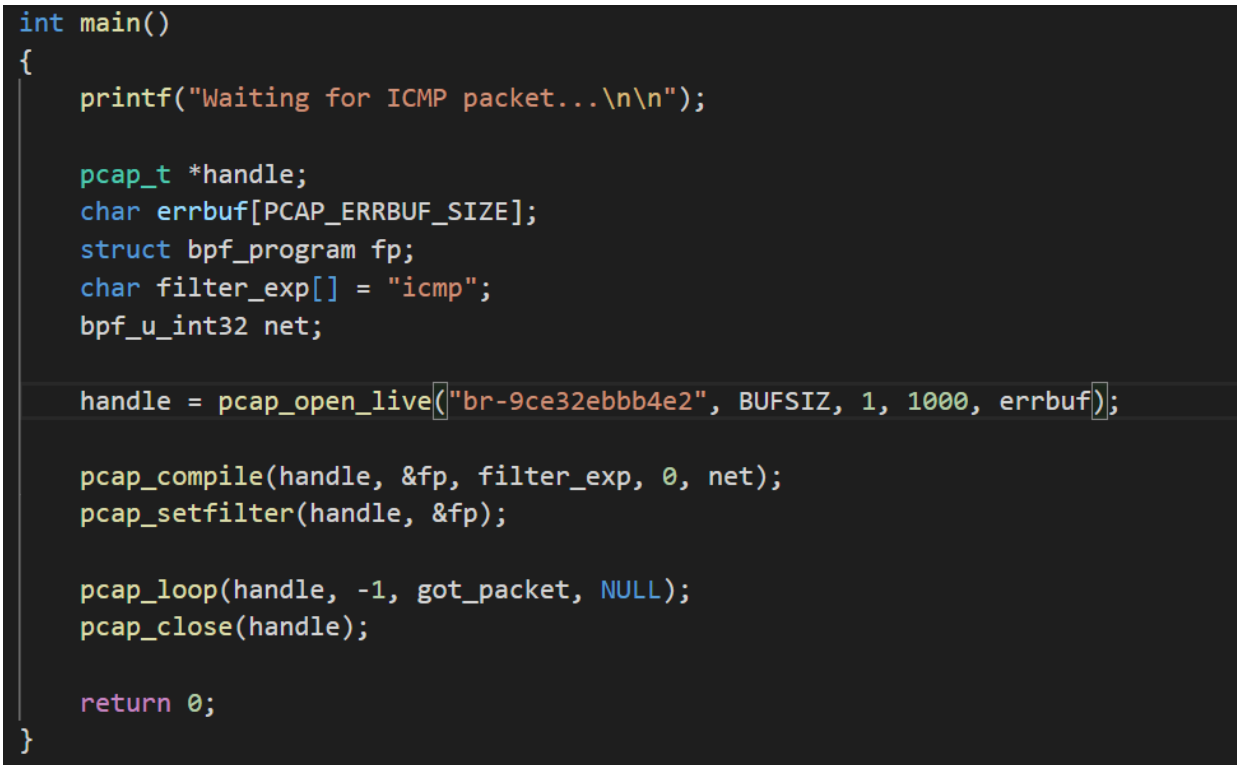
Code C :



```
#include <stdio.h>
#include <string.h>
#include <pcap.h>
#include <pcap/pcap.h>
#include <arpa/inet.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
```



```
int count =0;
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{
    printf("Got an ICMP packet!\n");
    struct sockaddr_in ip_src, ip_dst;
    struct iphdr *ip = (struct iphdr *)(packet + 14);
           if (count %2==0){

        printf("-----Request-----\n");
        }
        else  printf("-----Reply-----\n");

    ip_src.sin_addr.s_addr = ip->saddr;
    printf("Src IP: %s \n", inet_ntoa(ip_src.sin_addr));

    ip_dst.sin_addr.s_addr = ip->daddr;
    printf("Dst IP: %s \n", inet_ntoa(ip_dst.sin_addr));

    struct icmphdr *icmp = (struct icmphdr *)(packet + (ip->ihl * 4) + 14);
    count ++;


}
```



```
int main()
{
    printf("Waiting for ICMP packet...\n\n");

    pcap_t *handle;
    char errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char filter_exp[] = "icmp";
    bpf_u_int32 net;

    handle = pcap_open_live("br-9ce32ebbb4e2", BUFSIZ, 1, 1000, errbuf);

    pcap_compile(handle, &fp, filter_exp, 0, net);
    pcap_setfilter(handle, &fp);

    pcap_loop(handle, -1, got_packet, NULL);
    pcap_close(handle);

    return 0;
}
```

Question 1 : Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not detailed explanation like the one in the tutorial or book.

A :
The following calls are essential for this sniffer program :

- pcap_lookupdev : Find the default device on which to capture.
- pcap_lookupnet : Is used to determine the IPv4 network number and mask associated with the network device.
- pcap_open_live : Open a device to start sniffing.
- pcap_datalink : Get the link-layer header type to know what type.
- pcap_compile : Compile a filter expression.
- pcap_setfilter : Sets the compiled filter.
- pcap_loop : Process packets from the capture.
- pcap_freecode : Frees up allocated memory generated.
- pcap_close : Closes the sniffing session.

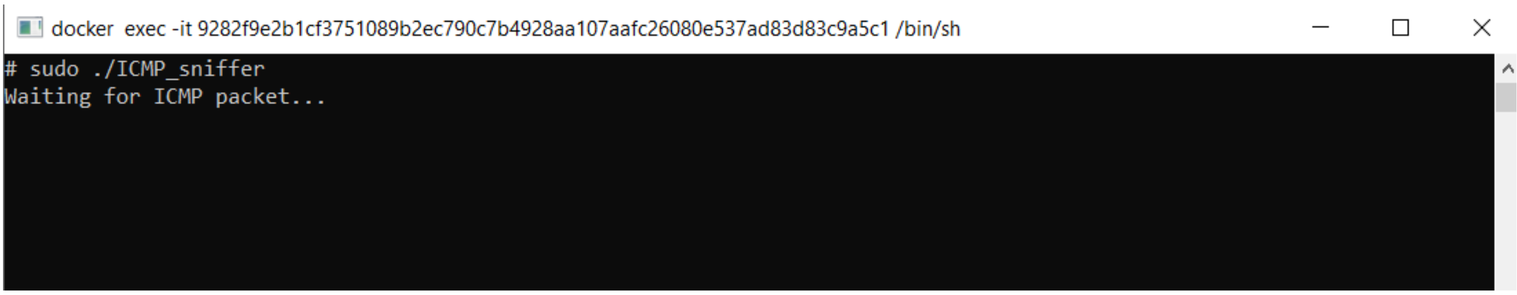
Question 2 : Why do you need the root privilege to run a sniffer program ? Where does the program fail if it is executed without the root privilege ?

A : We need root privilege to access to the Network Interface Card which needs admin authorization.

Question 3 : Please turn on and turn off the promiscuous mode in your sniffer program. The value 1 of the third parameter in pcap open live() turns on the promiscuous mode (use 0 to turn it off). Can you demonstrate the difference when this mode is on and off ? Please describe how you can demonstrate this. You can use the following command to check whether an interface's promiscuous mode is on or off (look at the promiscuity's value).

A : It can be deduced from the pictures that when Promiscuous mode is off in "0," mode our branch software does not manages to detect any facts that pass through Internet traffic from the second (attacked) VM.
On the other hand, when Promiscuous mode is on in "1" mode, our branch software manages to branch information from users on the network.

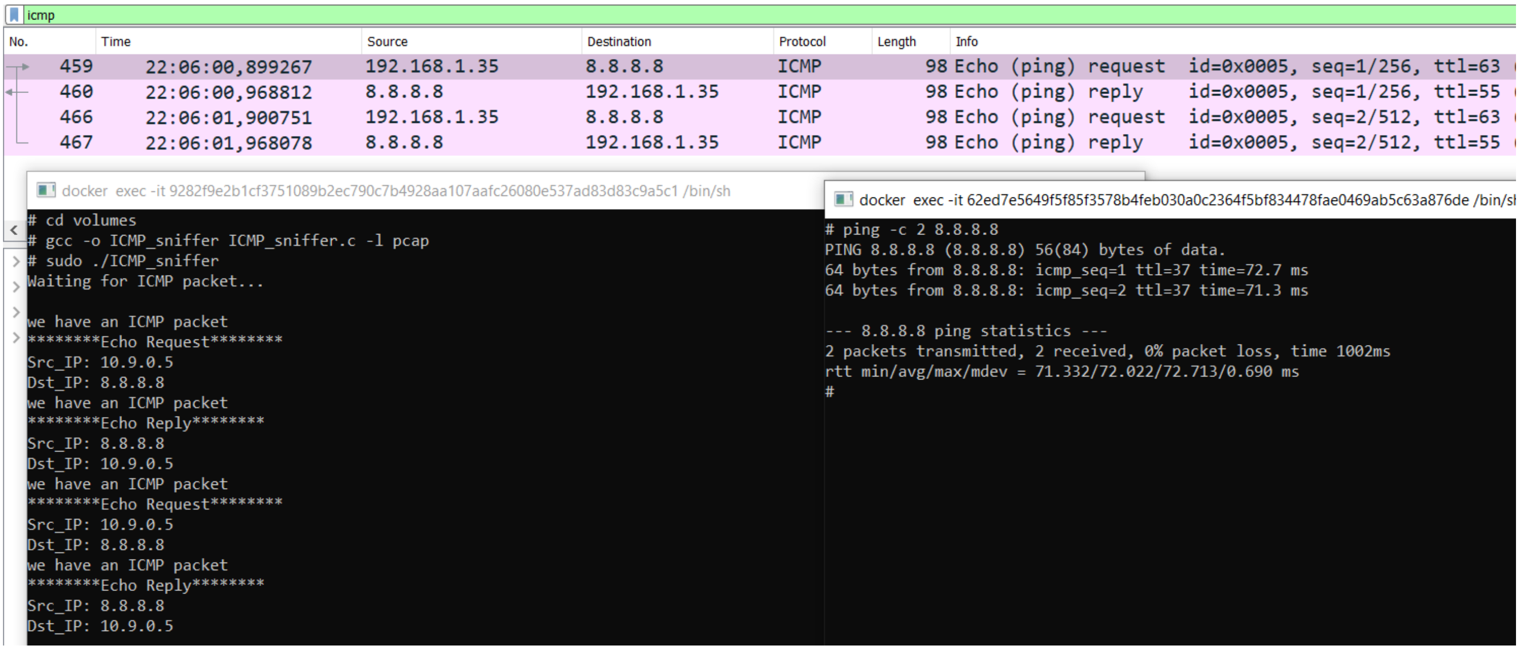
Promiscuous mode OFF :



docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh

```
# sudo ./ICMP_sniffer
Waiting for ICMP packet...
```

Promiscuous mode ON :



```
No.   Time              Source          Destination     Protocol  Length  Info
 459  22:06:00,899267   192.168.1.35    8.8.8.8         ICMP         98 Echo (ping) request  id=0x0005, seq=1/256, ttl=63
 460  22:06:00,968812   8.8.8.8         192.168.1.35    ICMP         98 Echo (ping) reply    id=0x0005, seq=1/256, ttl=55
 466  22:06:01,900751   192.168.1.35    8.8.8.8         ICMP         98 Echo (ping) request  id=0x0005, seq=2/512, ttl=63
 467  22:06:01,968078   8.8.8.8         192.168.1.35    ICMP         98 Echo (ping) reply    id=0x0005, seq=2/512, ttl=55
```

docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh

```
# cd volumes
# gcc -o ICMP_sniffer ICMP_sniffer.c -l pcap
# sudo ./ICMP_sniffer
Waiting for ICMP packet...

we have an ICMP packet
********Echo Request********
Src_IP: 10.9.0.5
Dst_IP: 8.8.8.8
we have an ICMP packet
********Echo Reply********
Src_IP: 8.8.8.8
Dst_IP: 10.9.0.5
we have an ICMP packet
********Echo Request********
Src_IP: 10.9.0.5
Dst_IP: 8.8.8.8
we have an ICMP packet
********Echo Reply********
Src_IP: 8.8.8.8
Dst_IP: 10.9.0.5
```

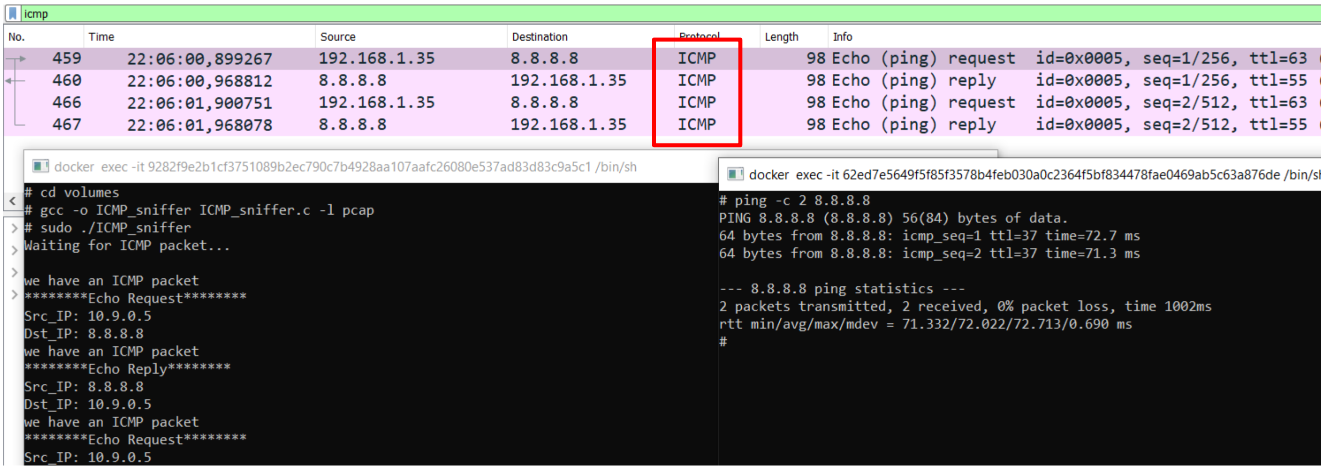
docker exec -it 62ed7e5649f5f85f3578b4feb030a0c2364f5bf834478fae0469ab5c63a876de /bin/s

```
# ping -c 2 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=37 time=72.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=37 time=71.3 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 71.332/72.022/72.713/0.690 ms
#
```

Task 2.1B: Writing Filters :

- Capture the ICMP packets between two specific hosts.

-> look task 2.1A :



```
icmp
No.   Time              Source         Destination    Protocol  Length Info
459   22:06:00,899267   192.168.1.35   8.8.8.8        ICMP      98 Echo (ping) request  id=0x0005, seq=1/256, ttl=63
460   22:06:00,968812   8.8.8.8        192.168.1.35   ICMP      98 Echo (ping) reply    id=0x0005, seq=1/256, ttl=55
466   22:06:01,900751   192.168.1.35   8.8.8.8        ICMP      98 Echo (ping) request  id=0x0005, seq=2/512, ttl=63
467   22:06:01,968078   8.8.8.8        192.168.1.35   ICMP      98 Echo (ping) reply    id=0x0005, seq=2/512, ttl=55
```

docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh

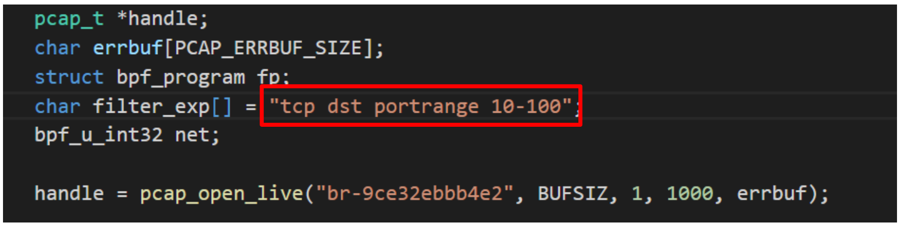
```
# cd volumes
# gcc -o ICMP_sniffer ICMP_sniffer.c -l pcap
# sudo ./ICMP_sniffer
Waiting for ICMP packet...

we have an ICMP packet
********Echo Request********
Src_IP: 10.9.0.5
Dst_IP: 8.8.8.8
we have an ICMP packet
********Echo Reply********
Src_IP: 8.8.8.8
Dst_IP: 10.9.0.5
we have an ICMP packet
********Echo Request********
Src_IP: 10.9.0.5
```

docker exec -it 62ed7e5649f5f85f3578b4feb030a0c2364f5bf834478fae0469ab5c63a876de /bin/sh

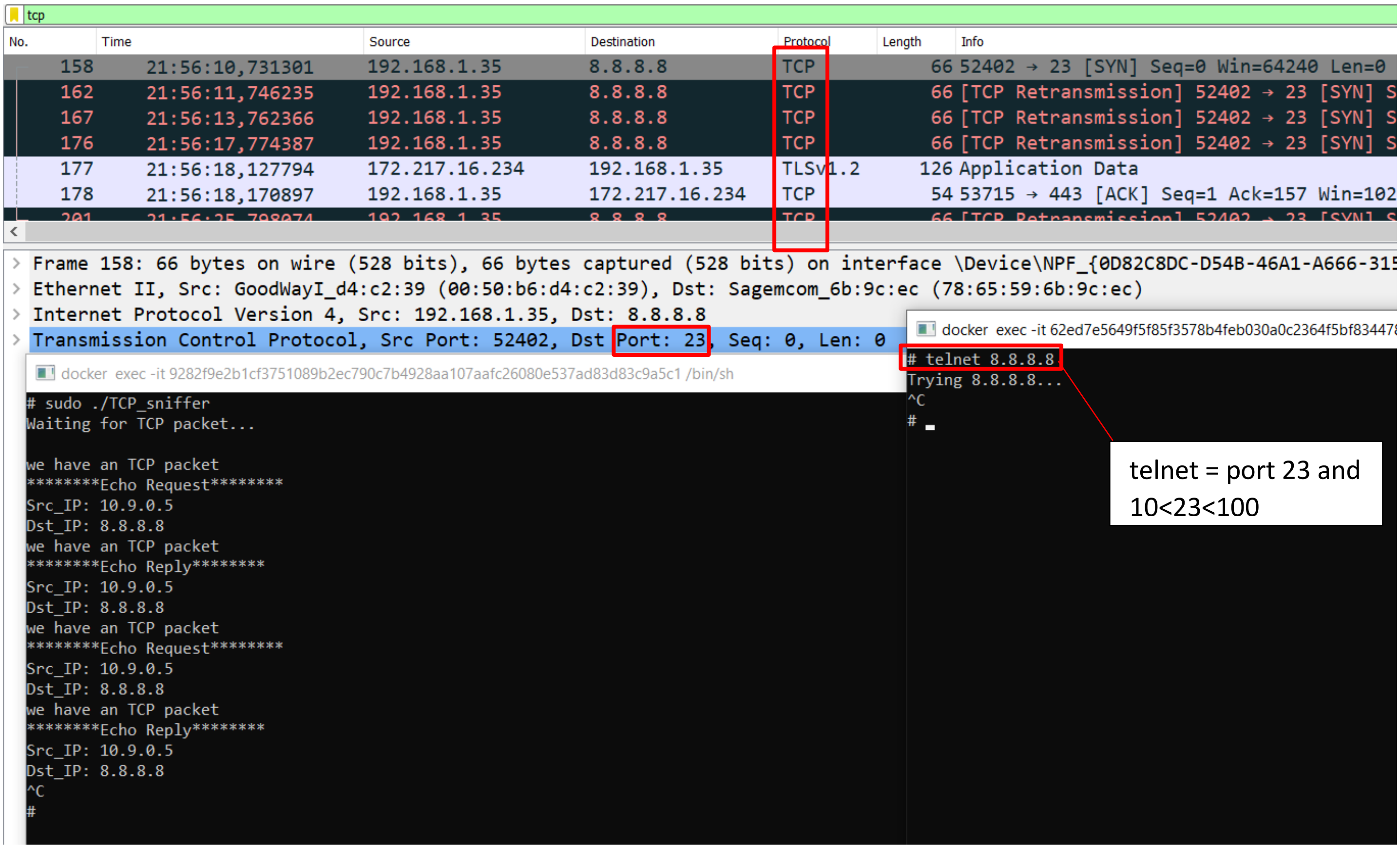
```
# ping -c 2 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=37 time=72.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=37 time=71.3 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 71.332/72.022/72.713/0.690 ms
#
```

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

We can capture only the TCP packets with a destination port number in the given range from 10 to 100 by modifying the filter_exp[] string in the code ICMP_sniffer:



```
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
struct bpf_program fp;
char filter_exp[] = "tcp dst portrange 10-100";
bpf_u_int32 net;

handle = pcap_open_live("br-9ce32ebbb4e2", BUFSIZ, 1, 1000, errbuf);
```



```
tcp
No.   Time              Source          Destination     Protocol  Length Info
158   21:56:10,731301   192.168.1.35    8.8.8.8         TCP       66 52402 → 23 [SYN] Seq=0 Win=64240 Len=0
162   21:56:11,746235   192.168.1.35    8.8.8.8         TCP       66 [TCP Retransmission] 52402 → 23 [SYN] S
167   21:56:13,762366   192.168.1.35    8.8.8.8         TCP       66 [TCP Retransmission] 52402 → 23 [SYN] S
176   21:56:17,774387   192.168.1.35    8.8.8.8         TCP       66 [TCP Retransmission] 52402 → 23 [SYN] S
177   21:56:18,127794   172.217.16.234  192.168.1.35    TLSv1.2   126 Application Data
178   21:56:18,170897   192.168.1.35    172.217.16.234  TCP       54 53715 → 443 [ACK] Seq=1 Ack=157 Win=102
201   21:56:25,798074   192.168.1.35    8.8.8.8         TCP       66 [TCP Retransmission] 52402 → 23 [SYN] S

> Frame 158: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{0D82C8DC-D54B-46A1-A666-315
> Ethernet II, Src: GoodWayI_d4:c2:39 (00:50:b6:d4:c2:39), Dst: Sagemcom_6b:9c:ec (78:65:59:6b:9c:ec)
> Internet Protocol Version 4, Src: 192.168.1.35, Dst: 8.8.8.8
> Transmission Control Protocol, Src Port: 52402, Dst Port: 23, Seq: 0, Len: 0
```

docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh

```
# sudo ./TCP_sniffer
Waiting for TCP packet...

we have an TCP packet
********Echo Request********
Src_IP: 10.9.0.5
Dst_IP: 8.8.8.8
we have an TCP packet
********Echo Reply********
Src_IP: 10.9.0.5
Dst_IP: 8.8.8.8
we have an TCP packet
********Echo Request********
Src_IP: 10.9.0.5
Dst_IP: 8.8.8.8
we have an TCP packet
********Echo Reply********
Src_IP: 10.9.0.5
Dst_IP: 8.8.8.8
^C
#
```

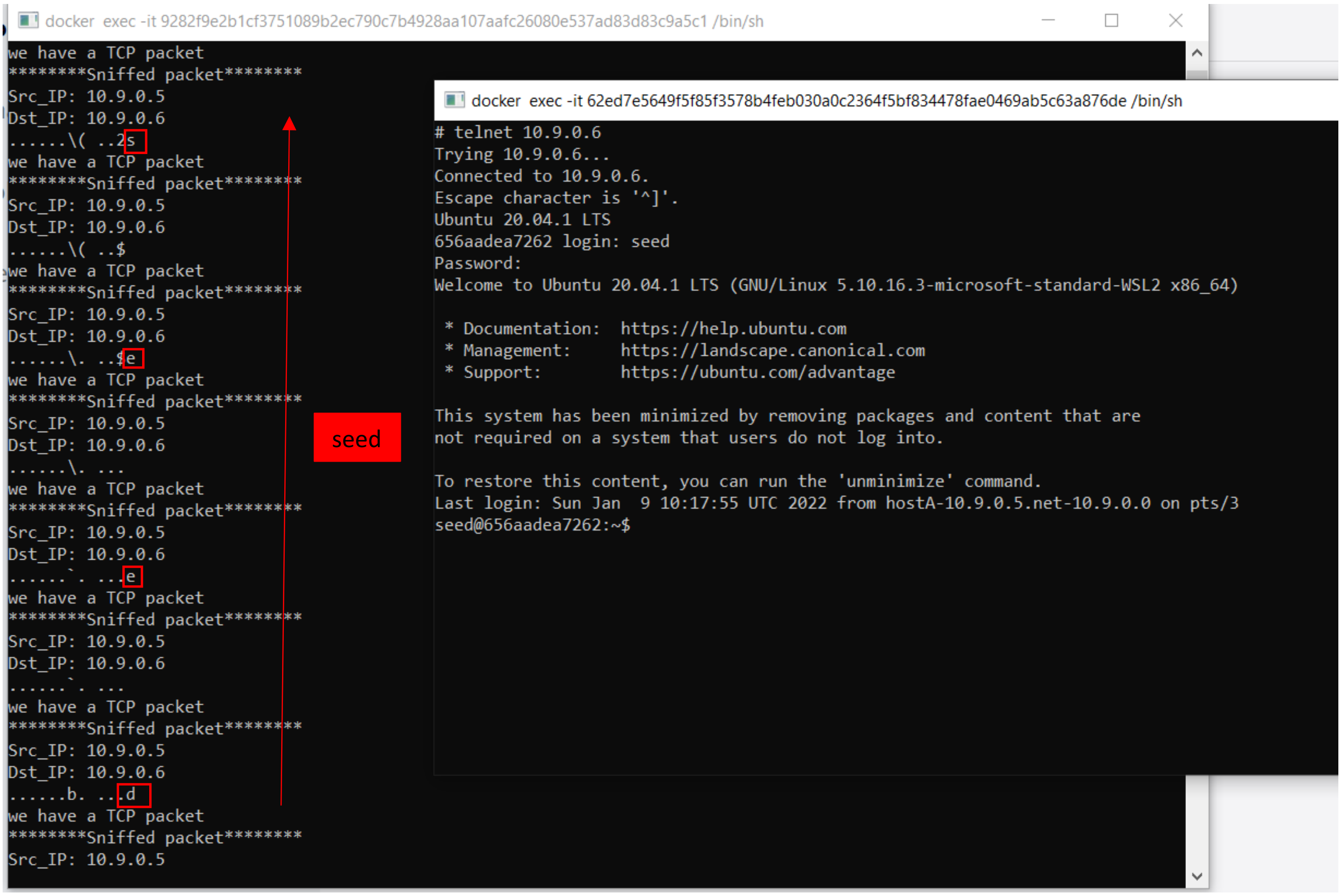
docker exec -it 62ed7e5649f5f85f3578b4feb030a0c2364f5bf83447

```
# telnet 8.8.8.8
Trying 8.8.8.8...
^C
#
```

telnet = port 23 and 10<23<100

Task 2.1C: Sniffing Passwords :

You need to run the command : sudo ./out2
And ping : telnet 10.9.0.6
Then you enter the login : seed
And the password : dees



docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh

```
we have a TCP packet
********Sniffed packet********
Src_IP: 10.9.0.5
Dst_IP: 10.9.0.6
......\( ..2s
we have a TCP packet
********Sniffed packet********
Src_IP: 10.9.0.5
Dst_IP: 10.9.0.6
......\( ..$
we have a TCP packet
********Sniffed packet********
Src_IP: 10.9.0.5
Dst_IP: 10.9.0.6
......\. ..$e
we have a TCP packet
********Sniffed packet********
Src_IP: 10.9.0.5
Dst_IP: 10.9.0.6
......\. ...
we have a TCP packet
********Sniffed packet********
Src_IP: 10.9.0.5
Dst_IP: 10.9.0.6
......`. ...e
we have a TCP packet
********Sniffed packet********
Src_IP: 10.9.0.5
Dst_IP: 10.9.0.6
......`. ...
we have a TCP packet
********Sniffed packet********
Src_IP: 10.9.0.5
Dst_IP: 10.9.0.6
......b. ...d
we have a TCP packet
********Sniffed packet********
Src_IP: 10.9.0.5
```

docker exec -it 62ed7e5649f5f85f3578b4feb030a0c2364f5bf834478fae0469ab5c63a876de /bin/sh

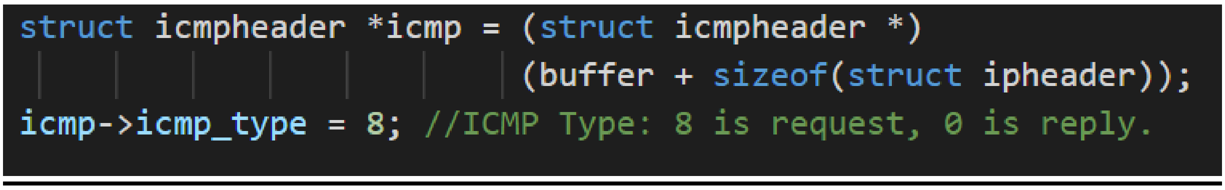
```
# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
656aadea7262 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.10.16.3-microsoft-standard-WSL2 x86_64)

 * 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: Sun Jan  9 10:17:55 UTC 2022 from hostA-10.9.0.5.net-10.9.0.0 on pts/3
seed@656aadea7262:~$
```

In this question the attacker wants to branch out the password of the user (victim), the attacker tries to access the VM by Operation : 10.9.0.6. After that, the user is asked to enter his password for login.
The attacker branches the TCP packets that pass through the Internet traffic and discovers the user's password by the information provided from the data of the facts.
The attacker sees the password at the end of te data. When we read them in a sequence from bottom to the top, the password will be revealed.
In the image shown above, the user's password is « dees ».

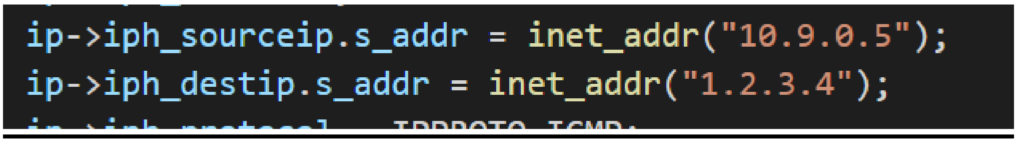
Task 2.2A: Write a spoofing program :

I wrote the code but I can’t see in wireshark like all my matala, I can’t see the reply

The important part of the code :



```
struct icmpheader *icmp = (struct icmpheader *)
                          (buffer + sizeof(struct ipheader));
icmp->icmp_type = 8; //ICMP Type: 8 is request, 0 is reply.
```

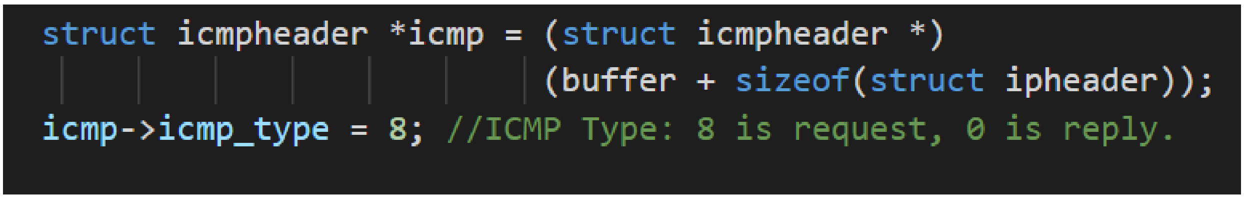


```
ip->iph_sourceip.s_addr = inet_addr("10.9.0.5");
ip->iph_destip.s_addr = inet_addr("1.2.3.4");
ip->iph_protocol = IPPROTO_ICMP;
```

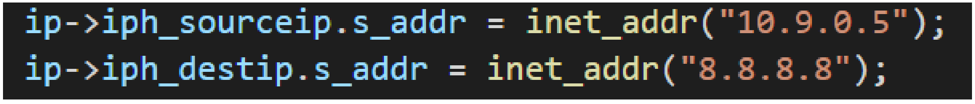
Task 2.2B: Spoof an ICMP Echo Request :

A big part is take from the tirgoul.
In this section we were asked to return an ICMP reply to an ICMP request from any machine.
I issue an unreal request spoofedon 10.9.0.5 to the google DNS (8.8.8.8).
And he gives me a reply.
It can be seen that a request was issued to 8.8.8.8 google DNS server, it can be seen that the request came from the IP of me and this is because when creating the containers then the attacker also gets the address 10.9.0.1 and he also gets the address of my so that he can branch because otherwise he will only be able to branch the traffic on his LAN so it came out here that even when he issued a fake fact it came out on the IP of the VM but the answer ICMP reply as requested by the code back to 10.9.0.5 .

The important part of the code :

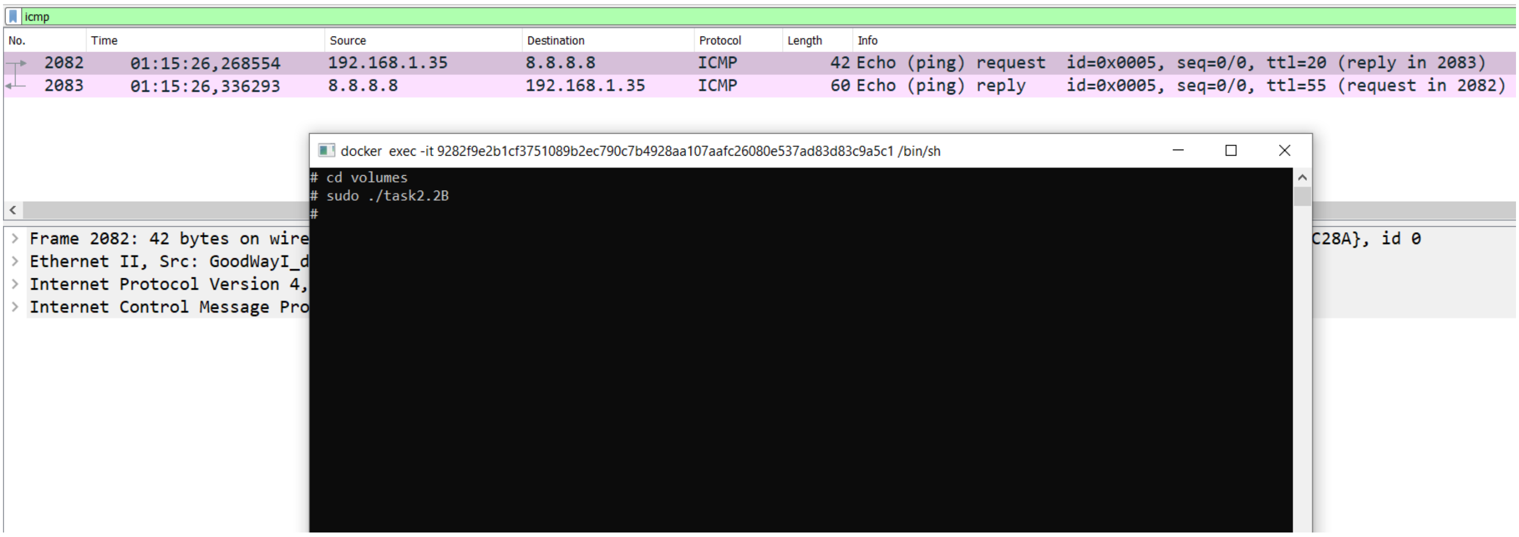


```
struct icmpheader *icmp = (struct icmpheader *)
                          (buffer + sizeof(struct ipheader));
icmp->icmp_type = 8; //ICMP Type: 8 is request, 0 is reply.
```



```
ip->iph_sourceip.s_addr = inet_addr("10.9.0.5");
ip->iph_destip.s_addr = inet_addr("8.8.8.8");
```

Capture Wireshark 2.2B :



```
No.    Time              Source          Destination     Protocol  Length  Info
2082   01:15:26,268554   192.168.1.35    8.8.8.8         ICMP          42  Echo (ping) request  id=0x0005, seq=0/0, ttl=20 (reply in 2083)
2083   01:15:26,336293   8.8.8.8         192.168.1.35    ICMP          60  Echo (ping) reply    id=0x0005, seq=0/0, ttl=55 (request in 2082)
```

```
> Frame 2082: 42 bytes on wire
> Ethernet II, Src: GoodWayI_d
> Internet Protocol Version 4,
> Internet Control Message Pro
C28A}, id 0
```

docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh

```
# cd volumes
# sudo ./task2.2B
#
```

- Question 4 : Can you set the IP packet length field to an arbitrary value, regardless of how big the actual packet is ?

The length field in this case should be the length of the IP packet, if not the sendto method (or function) will show an error ‘Invalid Argument’. So yes, it is important that it is the length of the IP packet.

- Question 5 : Using the raw socket programming, do you have to calculate the checksum for the IP header ?

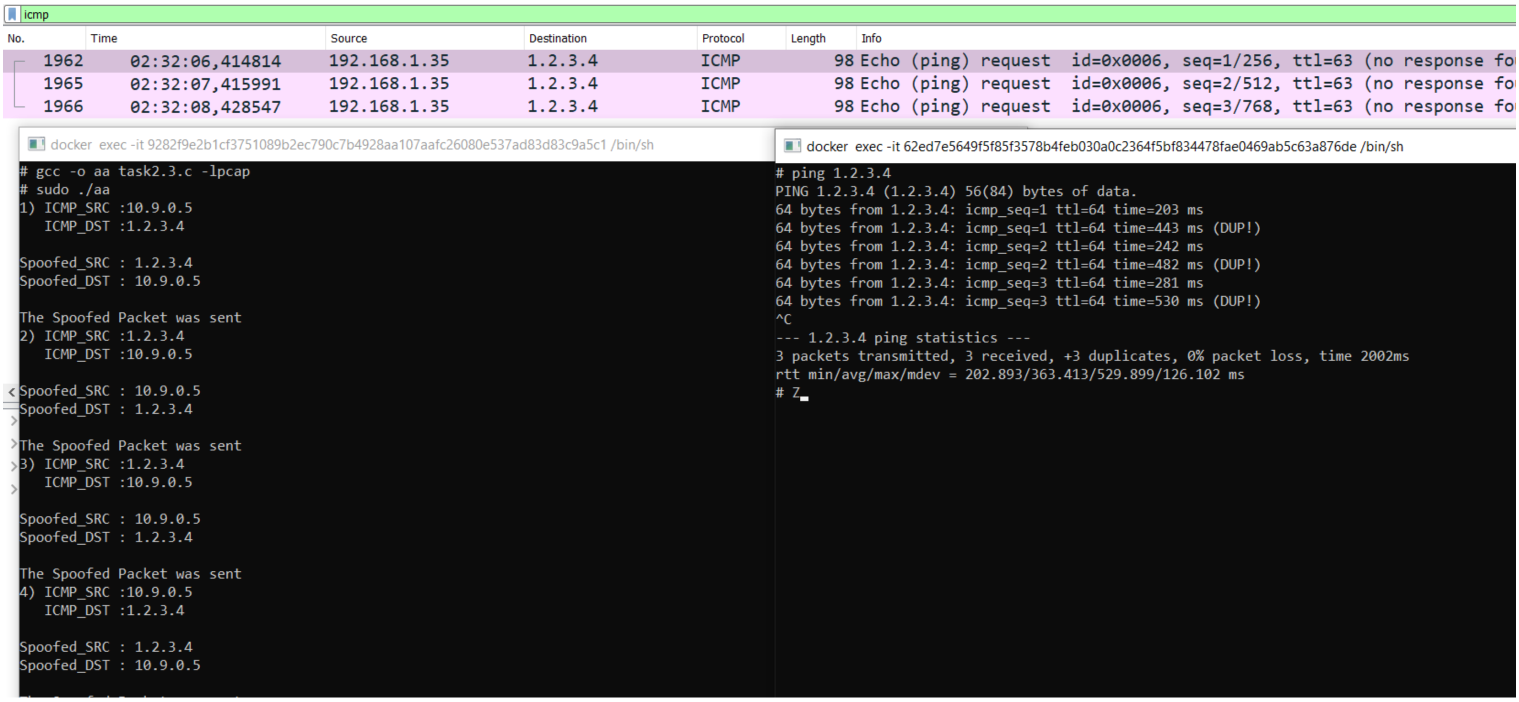
No you don’t need to calculate the checksum, it will be filled by the system.

- Question 6 : Why do you need the root privilege to run the programs that use raw sockets ? Where does the program fail if executed without the root privilege ?

It's restricted to root because it would break some rules for networking that are in place, without root you can't bind a port lower than 1024. With raw sockets you can simulate a server on any port and spoof custom packets which can interfere with inbound traffic.

Task 2.3: Sniff and then Spoof :

We can see that Wireshark displays a successful ping request even though no packet were really sent (100% packet loss), and in the attacker terminal we can also see that the source IP and the destination IP have been successfully swapped, meaning that the spoofing was successful. I don’t know why in wireshark I can’t see this swap.



icmp

No.	Time	Source	Destination	Protocol	Length	Info
1962	02:32:06,414814	192.168.1.35	1.2.3.4	ICMP	98	Echo (ping) request id=0x0006, seq=1/256, ttl=63 (no response fo
1965	02:32:07,415991	192.168.1.35	1.2.3.4	ICMP	98	Echo (ping) request id=0x0006, seq=2/512, ttl=63 (no response fo
1966	02:32:08,428547	192.168.1.35	1.2.3.4	ICMP	98	Echo (ping) request id=0x0006, seq=3/768, ttl=63 (no response fo

docker exec -it 9282f9e2b1cf3751089b2ec790c7b4928aa107aafc26080e537ad83d83c9a5c1 /bin/sh

```
# gcc -o aa task2.3.c -lpcap
# sudo ./aa
1) ICMP_SRC :10.9.0.5
   ICMP_DST :1.2.3.4

Spoofed_SRC : 1.2.3.4
Spoofed_DST : 10.9.0.5

The Spoofed Packet was sent
2) ICMP_SRC :1.2.3.4
   ICMP_DST :10.9.0.5

Spoofed_SRC : 10.9.0.5
Spoofed_DST : 1.2.3.4

The Spoofed Packet was sent
3) ICMP_SRC :1.2.3.4
   ICMP_DST :10.9.0.5

Spoofed_SRC : 10.9.0.5
Spoofed_DST : 1.2.3.4

The Spoofed Packet was sent
4) ICMP_SRC :10.9.0.5
   ICMP_DST :1.2.3.4

Spoofed_SRC : 1.2.3.4
Spoofed_DST : 10.9.0.5
```

docker exec -it 62ed7e5649f5f85f3578b4feb030a0c2364f5bf834478fae0469ab5c63a876de /bin/sh

```
# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=203 ms
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=443 ms (DUP!)
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=242 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=482 ms (DUP!)
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=281 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=530 ms (DUP!)
^C
--- 1.2.3.4 ping statistics ---
3 packets transmitted, 3 received, +3 duplicates, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 202.893/363.413/529.899/126.102 ms
# Z
```