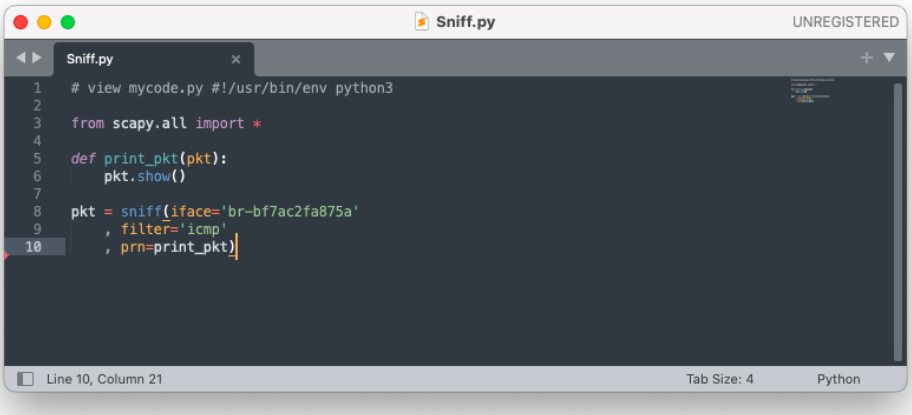
# Report Packet Sniffing and Spoofing Lab - Fatjon Freskina

## Task 1.1 : Sniffing Packets

After setting everything up, I opened a bash inside each container.

To show that I can sniff packets, I ran *ping* from HostA to HostB and try to sniff it from the attacker container running the Python program.

The Python program:




Output:

```
fatjonfreskina — com.docker.cli • docker exec -it c4 /bin/bash — 115×29
root@docker-desktop:/volumes# python3 Sniff.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 38010
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x9212
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x2ffd
        id        = 0x7
        seq       = 0xe
###[ Raw ]###
           load      = '\xba\xa7Jb\x00\x00\x00\x00\x02\x11\x02\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\
x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

We can see that the packet sniffed has *src* = *10.9.0.5* (HostA) and *dst* = *10.9.0.6* (HostB).

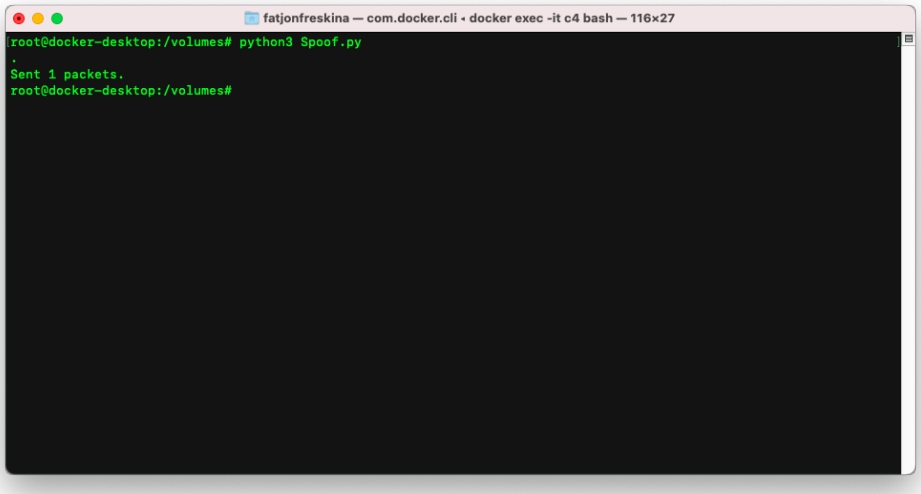
# Task 1.2: Spoofing ICMP Packets

Since on MacOS it is not easy to detect the traffic from bridge interfaces (the containers' ones) using Wireshark - they are not detected - , I will open 2 bash from in the attacker container and use Sniff.py to sniff the traffic and see if the ICMP echo reply arrives and another one to send the spoofed packet.

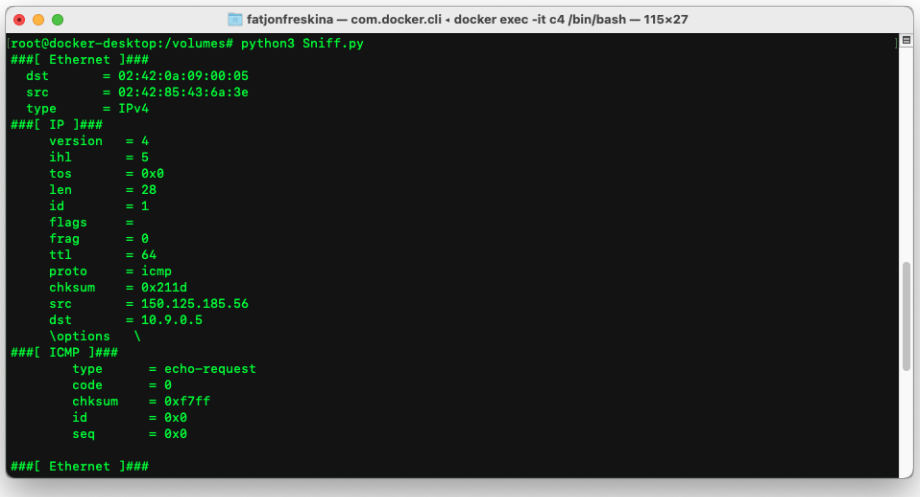
Python program:

```
Spoof.py                                                    UNREGISTERED
Spoof.py
1  from scapy.all import *
2
3  ip = IP(src=RandIP(),dst='10.9.0.5') #Host A
4  icmp = ip/ICMP()
5  send(icmp)
6
7
8
9
10
Line 1, Column 1                    Tab Size: 4                Python
```

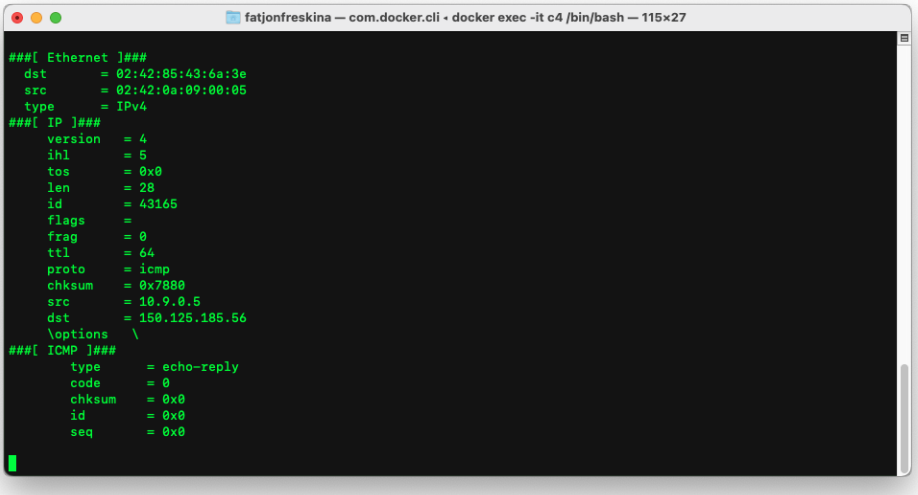
Output of Spoof.py




Output of Sniff.py:




Notes: this is the sent packet, we can see how it is an ICMP echo-request packet from a random IP address to HostA




And this is the ICMP echo-reply (from the victim to the attacker)

# Task 1.3: Traceroute

Python program to implement Traceroute:

Traceroute.py UNREGISTERED

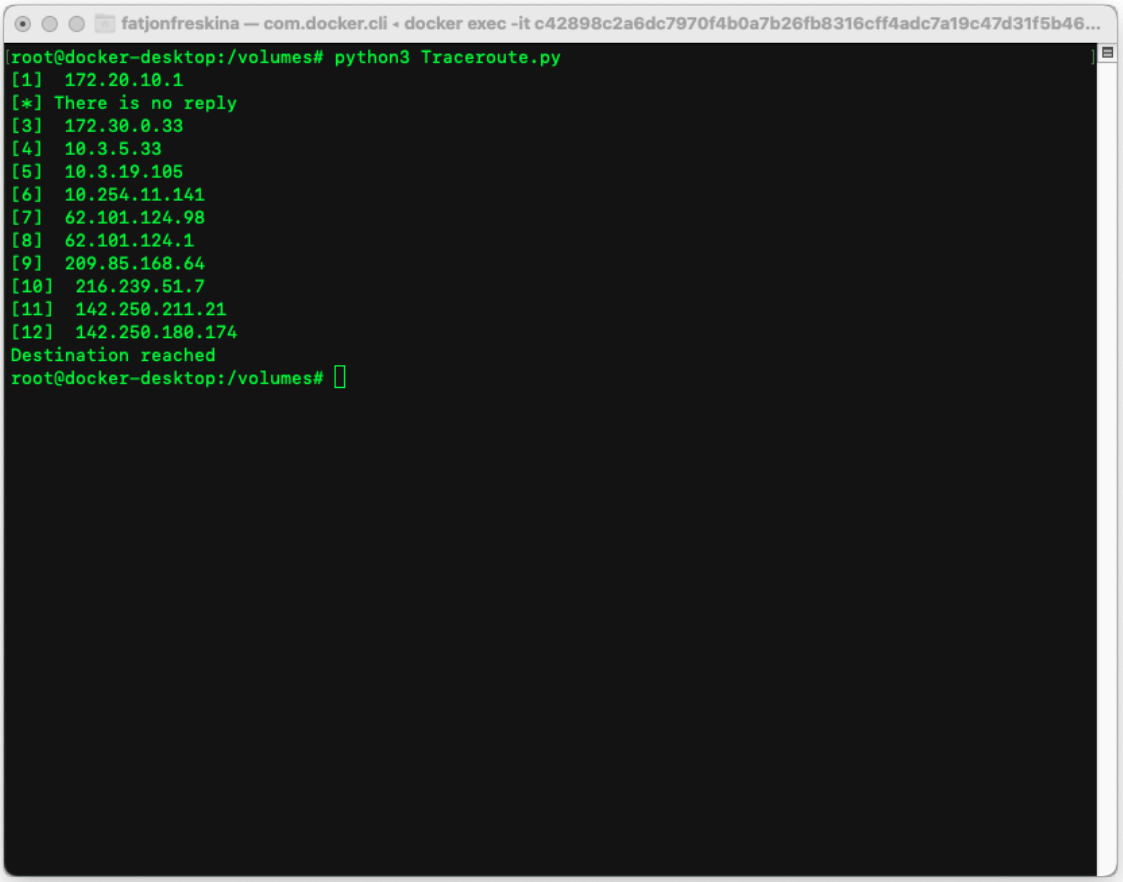
Traceroute.py

```python
from scapy.all import *

dport = 80
dst = 'google.com'
time_to_live = 0
new = 0 #used to check if destination reached

while (True):

    time_to_live += 1
    pkt = IP(dst=dst, ttl=time_to_live) / ICMP()
    response = sr1(pkt, verbose=0, timeout=5)

    try:
        if (new == response.src):
            exit("Destination reached")
    except AttributeError: pass

    try:
        new = response.src
    except AttributeError: pass

    if response is None:
        print("[*] There is no reply")
        continue

    else:
        print("[{}] ".format(time_to_live), response.src)
```

Line 20, Column 9; Saved ~/Desktop/EthicalHacking/Packet Sniffng and Spoofing Lab-20220331/Labsetup/volumes/Traceroute.py (UTF-8

The function sr1() is a variant that only returns one packet that answered the packet (or the packet set) sent.

Output:




142.250.180.174 is in fact the IP address of google.com

# Task 1.4: Sniffing and-then Spoofing

sniff_and_spoof.py UNREGISTERED

```python
from scapy.all import *

# host = "1.2.3.4"      # a non-existing host on the Internet
# host = "10.9.0.99"    # a non-existing host on the LAN
host = "8.8.8.8"        # an existing host on the Internet

def spoof(pkt):
    if pkt[ICMP].type == 8: #echo request
        s_pck = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl) / \
        ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq) / \
        pkt[Raw].load

        send(s_pck, verbose = 0)
        print("packet sent")
        s_pck.show()

pkt = sniff(iface='br-bf7ac2fa875a', filter = "icmp and host " + host , prn = spoof)
```

Line 1, Column 1 Tab Size: 4 Python

The program sniffs the given interface and if the packet sniffed is an ICMP echo request then creates a packet spoofing the src and destination, and setting the type as icmp echo reply.

fatjonfreskina — com.docker.cli ◂ docker exec -it c42898c2a6dc7970f4b0a7b26fb8316cff4adc7a19c47d31f5...

```
packet sent
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 8.8.8.8
  dst       = 10.9.0.5
  \options   \
###[ ICMP ]###
     type      = echo-reply
     code      = 0
     chksum    = None
     id        = 0x1
     seq       = 0x1
###[ Raw ]###
        load      = '\xe8\x10Tb\x00\x00\x00\x00\xbb8\x0c\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\
x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

Output of the program from the attacker side.

```
fatjonfreskina — com.docker.cli ◂ docker exec -it fd5783482cd47dc0ae1b6726e8529c3712956bc1b961c3907d14478...
root@fd5783482cd4:/# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:28:40.801047 IP fd5783482cd4 > dns.google: ICMP echo request, id 1, seq 1, length 64
11:28:40.803675 IP fd5783482cd4.42458 > 192.168.65.5.domain: 20232+ PTR? 8.8.8.8.in-addr.arpa. (38)
11:28:40.817139 IP 192.168.65.5.domain > fd5783482cd4.42458: 20232 1/0/0 PTR dns.google. (82)
11:28:40.820402 IP fd5783482cd4.37511 > 192.168.65.5.domain: 3862+ PTR? 5.65.168.192.in-addr.arpa. (43)
11:28:40.827602 ARP, Request who-has fd5783482cd4 tell 10.9.0.1, length 28
11:28:40.827637 ARP, Reply fd5783482cd4 is-at 02:42:0a:09:00:05 (oui Unknown), length 28
11:28:40.830061 IP 192.168.65.5.domain > fd5783482cd4.37511: 3862 NXDomain 0/0/0 (43)
11:28:40.834164 IP fd5783482cd4.46481 > 192.168.65.5.domain: 17572+ PTR? 1.0.9.10.in-addr.arpa. (39)
11:28:40.838717 IP dns.google > fd5783482cd4: ICMP echo reply, id 1, seq 1, length 64
11:28:40.842132 IP 192.168.65.5.domain > fd5783482cd4.46481: 17572 NXDomain 0/0/0 (39)
11:28:41.805839 IP fd5783482cd4 > dns.google: ICMP echo request, id 1, seq 2, length 64
11:28:41.821177 IP dns.google > fd5783482cd4: ICMP echo reply, id 1, seq 2, length 64
11:28:42.810760 IP fd5783482cd4 > dns.google: ICMP echo request, id 1, seq 3, length 64
11:28:42.823723 IP dns.google > fd5783482cd4: ICMP echo reply, id 1, seq 3, length 64
11:28:43.815250 IP fd5783482cd4 > dns.google: ICMP echo request, id 1, seq 4, length 64
11:28:43.832272 IP dns.google > fd5783482cd4: ICMP echo reply, id 1, seq 4, length 64
11:28:46.190825 ARP, Request who-has 10.9.0.1 tell fd5783482cd4, length 28
11:28:46.191095 ARP, Reply 10.9.0.1 is-at 02:42:a7:6f:1f:17 (oui Unknown), length 28
^C
```

Sniffing the interface with tcpdump at the victim side (wireshark not working very well with MacOs and Docker).

```
fatjonfreskina — com.docker.cli ◂ docker exec -it fd5783482cd47dc0a...
root@fd5783482cd4:/# 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=37.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=15.4 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=13.0 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=17.1 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3014ms
rtt min/avg/max/mdev = 13.014/20.808/37.762/9.893 ms
root@fd5783482cd4:/#
```

Pinging google from victim side.