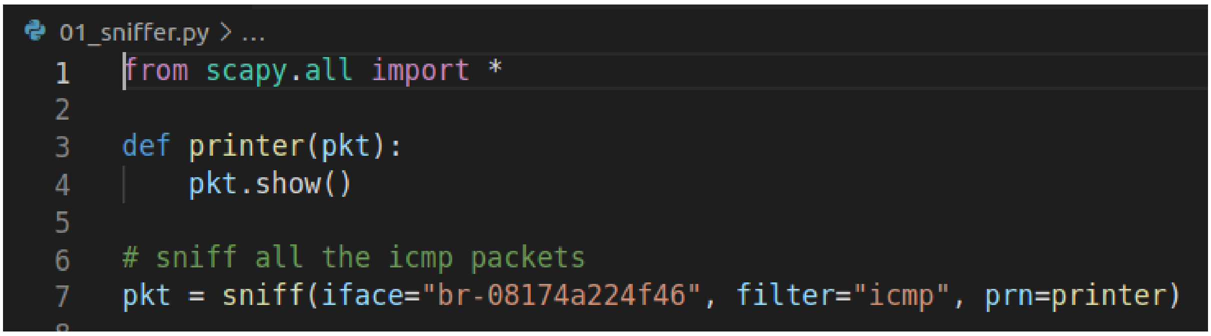
**Task 1.1A.Run the program with the root privilege and demonstrate that you can indeed capture packets.**
**After that, run the program again, but without using the root privilege; describe and explain your observations.**

We should run Python using the root privilege because the privilege is required for spoofing packets and to use scapy framework.Without super user privileges we cannot use scapy framework.

**Task 1.1B.**

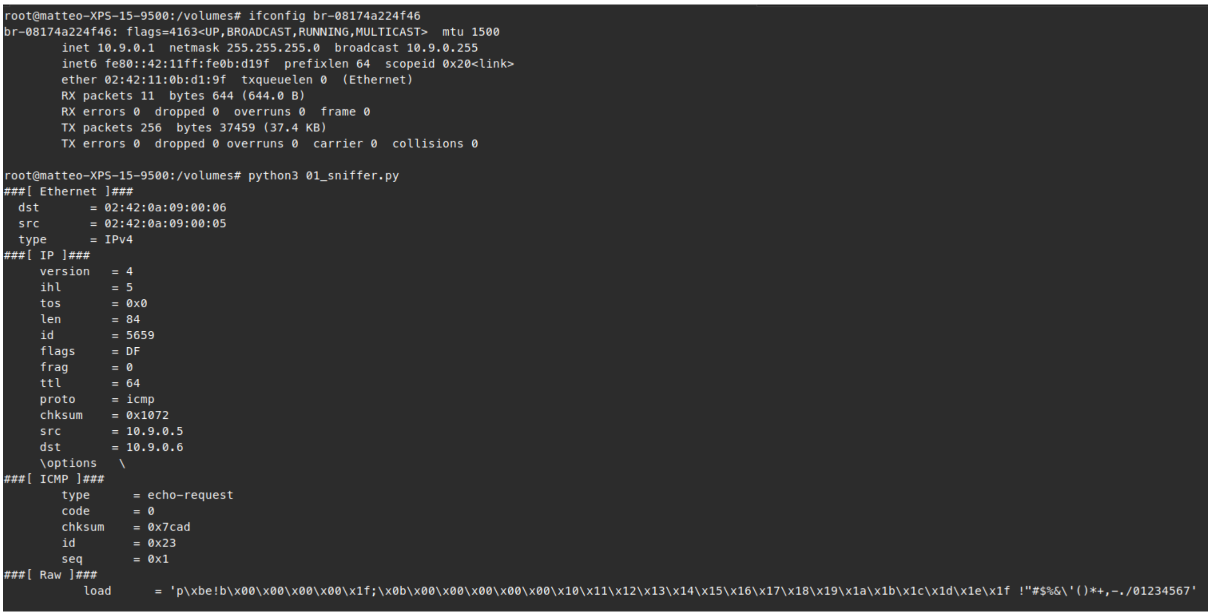
- **I. Capture only the ICMP packet**



01_sniffer.py > ...

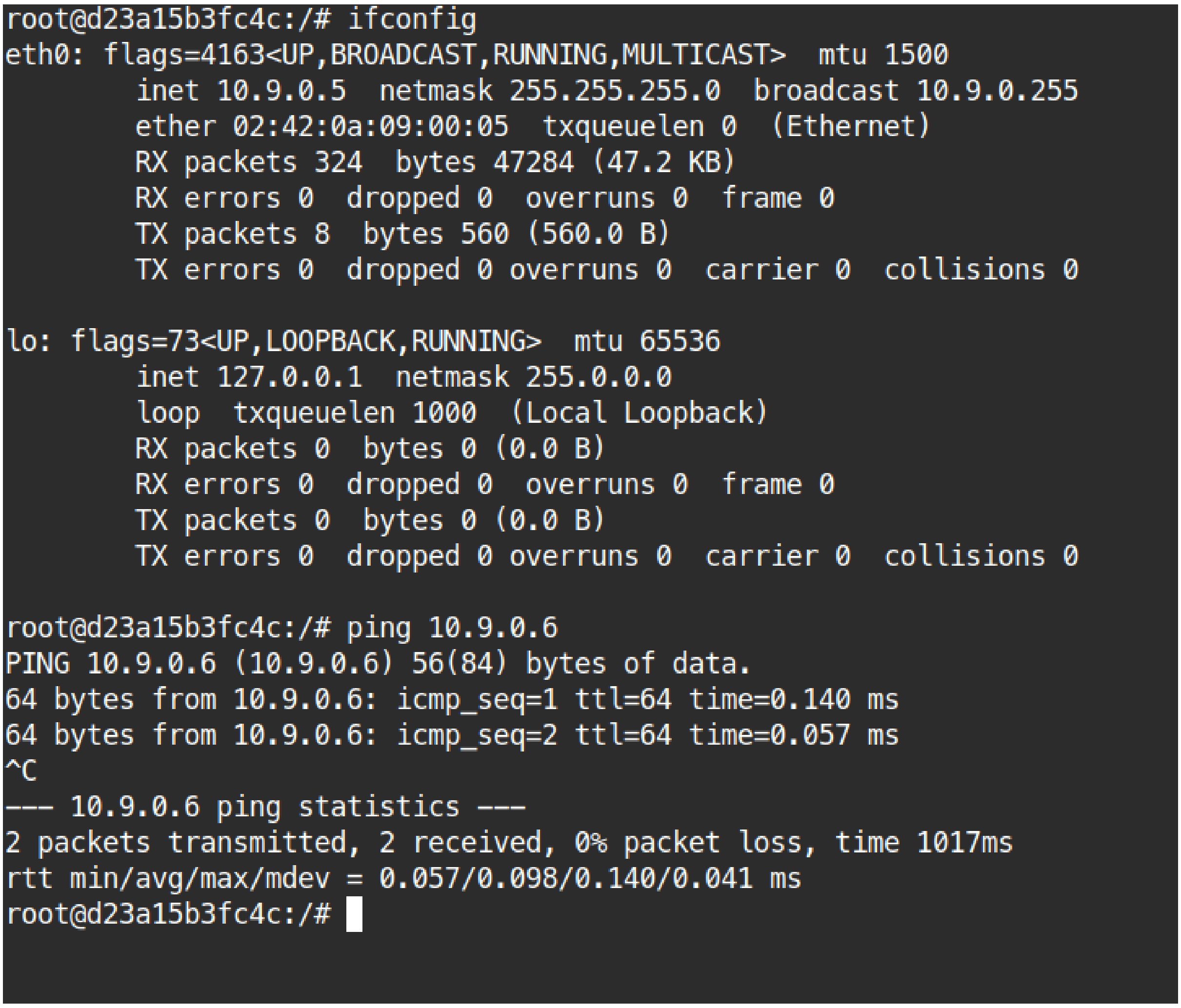
```python
from scapy.all import *

def printer(pkt):
    pkt.show()

# sniff all the icmp packets
pkt = sniff(iface="br-08174a224f46", filter="icmp", prn=printer)
```

In this piece of code we define a function **printer()** that prints out the packets and all the attributes of the packets. This function will be activated by function **sniff()** when a icmp packet will be matched in the defined interface.



```
root@matteo-XPS-15-9500:/volumes# ifconfig br-08174a224f46
br-08174a224f46: 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::42:11ff:fe0b:d19f  prefixlen 64  scopeid 0x20<link>
        ether 02:42:11:0b:d1:9f  txqueuelen 0  (Ethernet)
        RX packets 11  bytes 644 (644.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 256  bytes 37459 (37.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@matteo-XPS-15-9500:/volumes# python3 01_sniffer.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        = 5659
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x1072
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x7cad
        id        = 0x23
        seq       = 0x1
###[ Raw ]###
           load      = 'p\xbe!b\x00\x00\x00\x00\x1f;\x0b\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

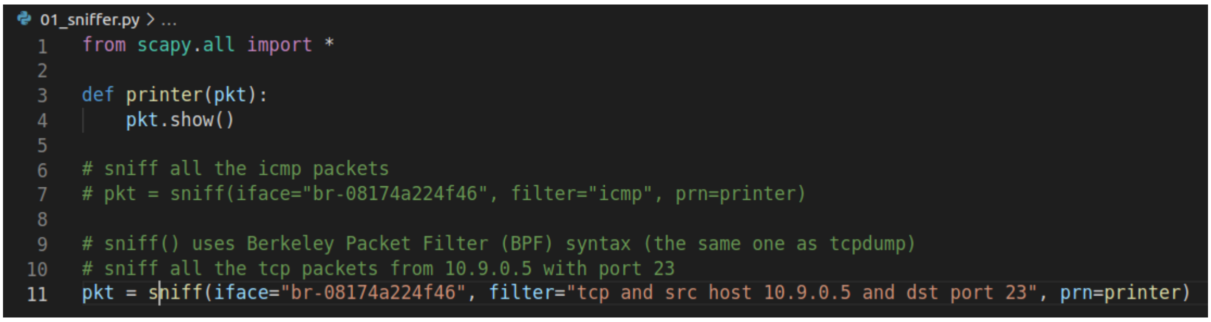
This is the output of the icmp packets sniffed by the attacker-machine.
Layer -> ethernet / ip / tcp / raw



```
root@d23a15b3fc4c:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 324  bytes 47284 (47.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 560 (560.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@d23a15b3fc4c:/# ping 10.9.0.6
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=0.140 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.057 ms
^C
--- 10.9.0.6 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1017ms
rtt min/avg/max/mdev = 0.057/0.098/0.140/0.041 ms
root@d23a15b3fc4c:/# 
```

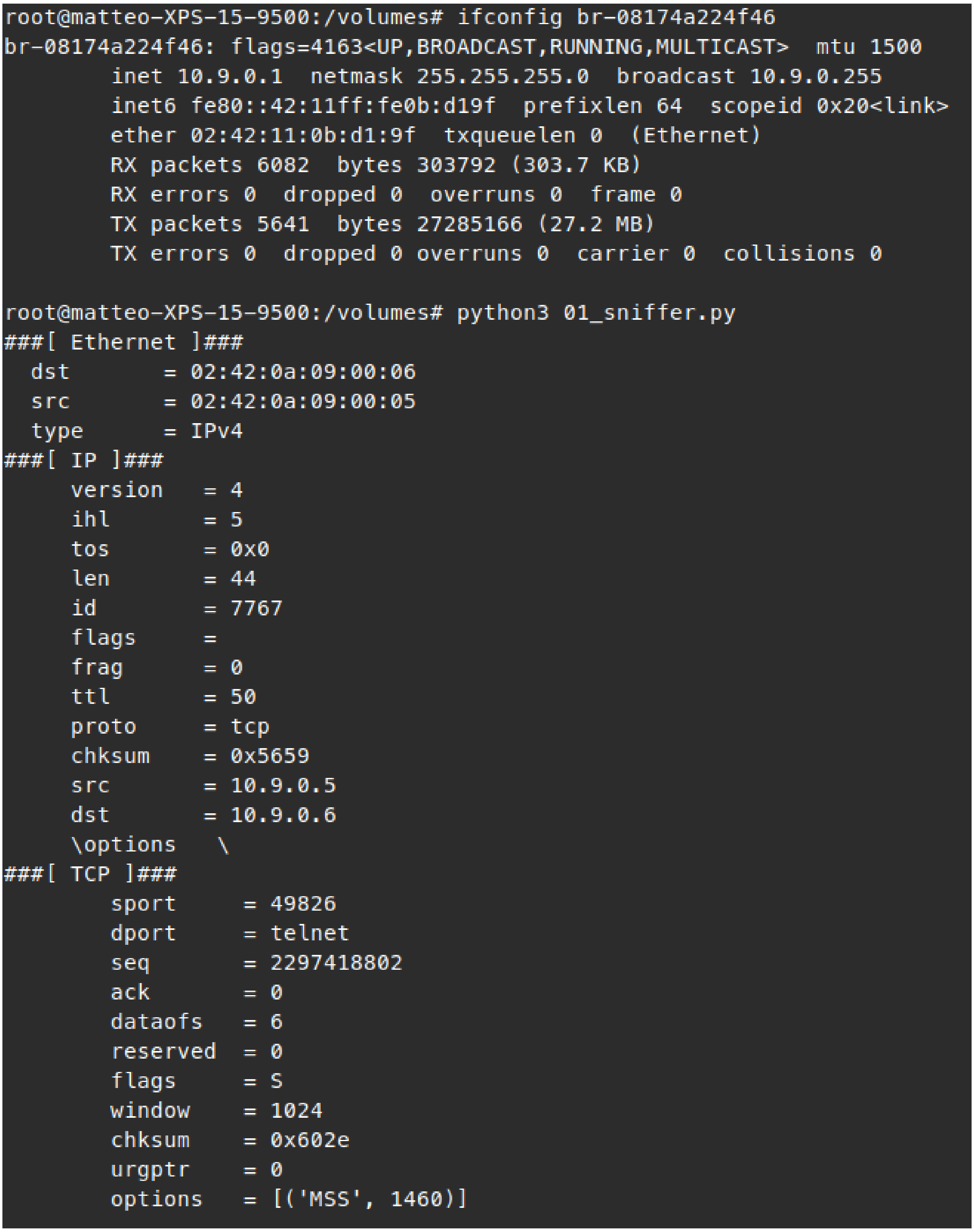
This is the output of the victim machine when a ping(ICMP) packet will be sent throw the network.

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



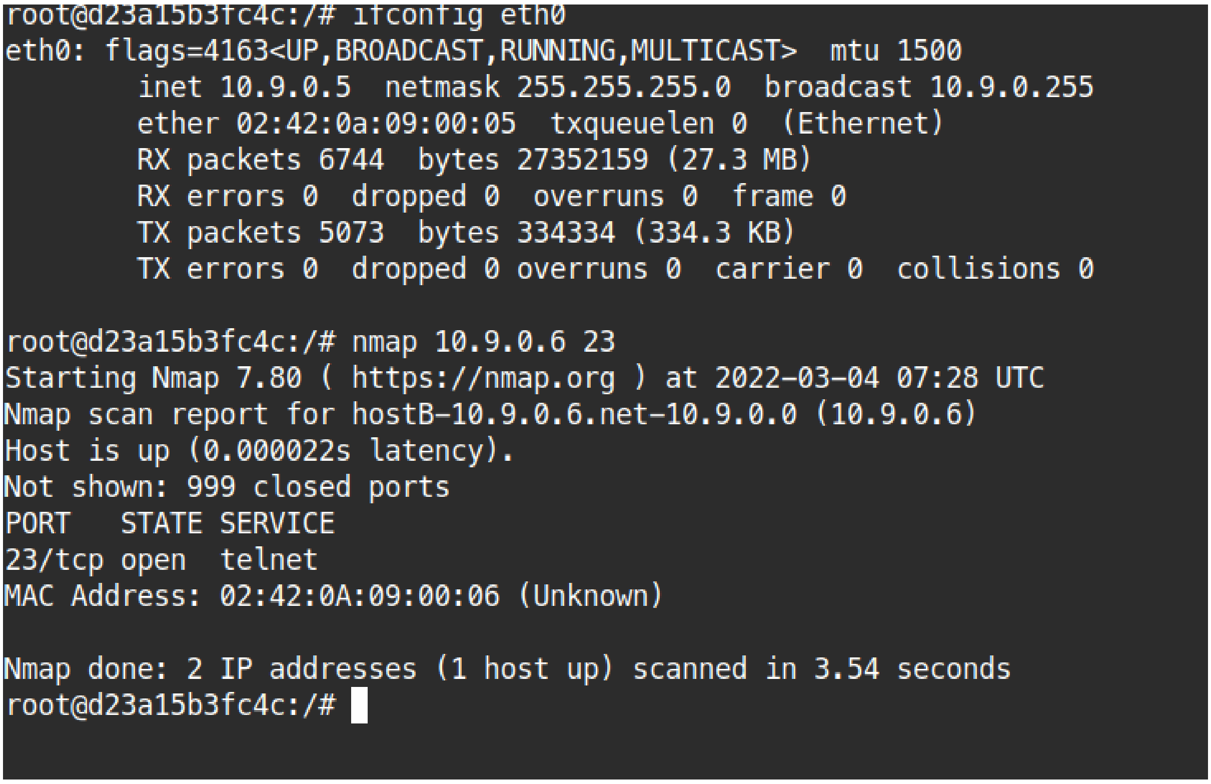
```python
from scapy.all import *

def printer(pkt):
    pkt.show()

# sniff all the icmp packets
# pkt = sniff(iface="br-08174a224f46", filter="icmp", prn=printer)

# sniff() uses Berkeley Packet Filter (BPF) syntax (the same one as tcpdump)
# sniff all the tcp packets from 10.9.0.5 with port 23
pkt = sniff(iface="br-08174a224f46", filter="tcp and src host 10.9.0.5 and dst port 23", prn=printer)
```

In this case we define a function **printer()** that prints out the packets and all the attributes of the packets. This function will be activated by the function **sniff()** when a tcp packet with source ip 10.9.0.5 and destination port 23 match in the interface defined.



```
root@matteo-XPS-15-9500:/volumes# ifconfig br-08174a224f46
br-08174a224f46: 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::42:11ff:fe0b:d19f  prefixlen 64  scopeid 0x20<link>
        ether 02:42:11:0b:d1:9f  txqueuelen 0  (Ethernet)
        RX packets 6082  bytes 303792 (303.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5641  bytes 27285166 (27.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@matteo-XPS-15-9500:/volumes# python3 01_sniffer.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       = 44
     id        = 7767
     flags     =
     frag      = 0
     ttl       = 50
     proto     = tcp
     chksum    = 0x5659
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ TCP ]###
        sport     = 49826
        dport     = telnet
        seq       = 2297418802
        ack       = 0
        dataofs   = 6
        reserved  = 0
        flags     = S
        window    = 1024
        chksum    = 0x602e
        urgptr    = 0
        options   = [('MSS', 1460)]
```

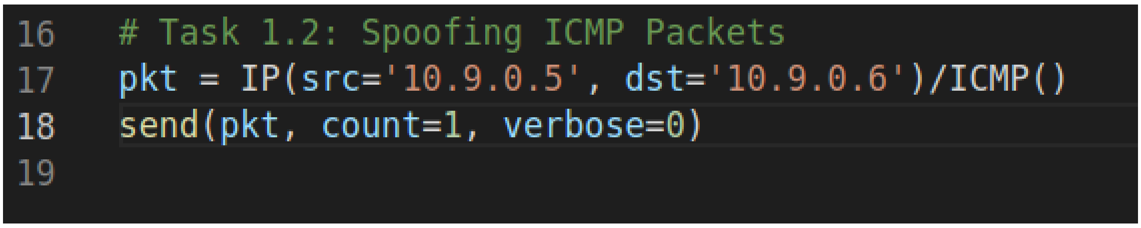
This is the output of one matched packet printed out by the attacker machine.
Layer -> ethernet / ip / tcp



```
root@d23a15b3fc4c:/# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 6744  bytes 27352159 (27.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5073  bytes 334334 (334.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@d23a15b3fc4c:/# nmap 10.9.0.6 23
Starting Nmap 7.80 ( https://nmap.org ) at 2022-03-04 07:28 UTC
Nmap scan report for hostB-10.9.0.6.net-10.9.0.0 (10.9.0.6)
Host is up (0.000022s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
23/tcp open  telnet
MAC Address: 02:42:0A:09:00:06 (Unknown)

Nmap done: 2 IP addresses (1 host up) scanned in 3.54 seconds
root@d23a15b3fc4c:/# 
```

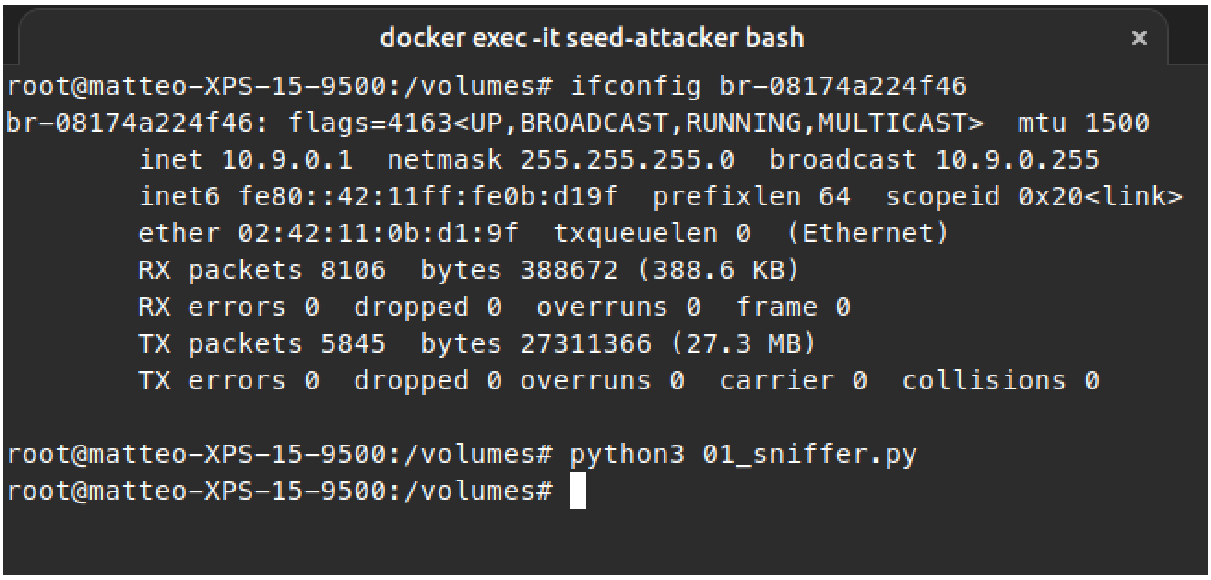
This is the output of the victim machine when ping, using nmap, the 10.9.0.6 host on port 23.

## Task 1.2: Spoofing ICMP Packets



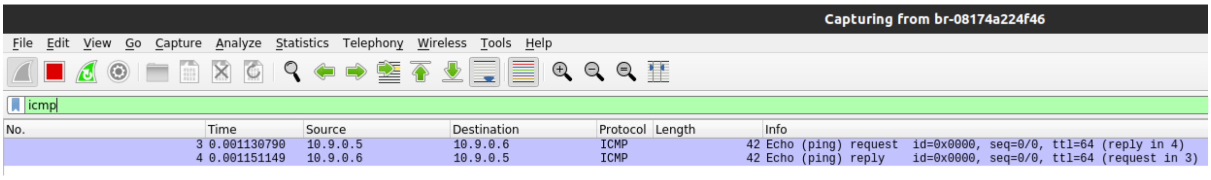
```
# Task 1.2: Spoofing ICMP Packets
pkt = IP(src='10.9.0.5', dst='10.9.0.6')/ICMP()
send(pkt, count=1, verbose=0)

```

This piece of code creates an ICMP packet setting up in IP layer source and destination ip, in order to spoof a packet.



```
root@matteo-XPS-15-9500:/volumes# ifconfig br-08174a224f46
br-08174a224f46: 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::42:11ff:fe0b:d19f  prefixlen 64  scopeid 0x20<link>
        ether 02:42:11:0b:d1:9f  txqueuelen 0  (Ethernet)
        RX packets 8106  bytes 388672 (388.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5845  bytes 27311366 (27.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@matteo-XPS-15-9500:/volumes# python3 01_sniffer.py
root@matteo-XPS-15-9500:/volumes# 
```

startup of the tool

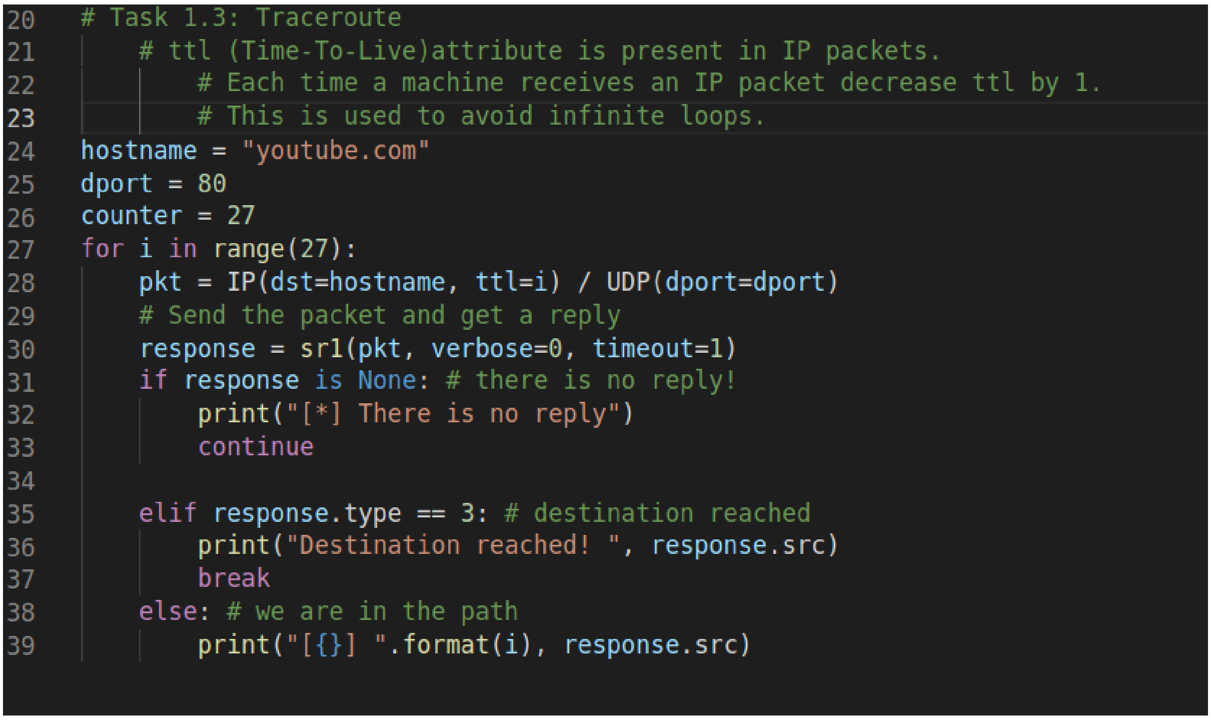


icmp

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 3 | 0.001130790 | 10.9.0.5 | 10.9.0.6 | ICMP | 42 | Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply in 4) |
| 4 | 0.001151149 | 10.9.0.6 | 10.9.0.5 | ICMP | 42 | Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (request in 3) |

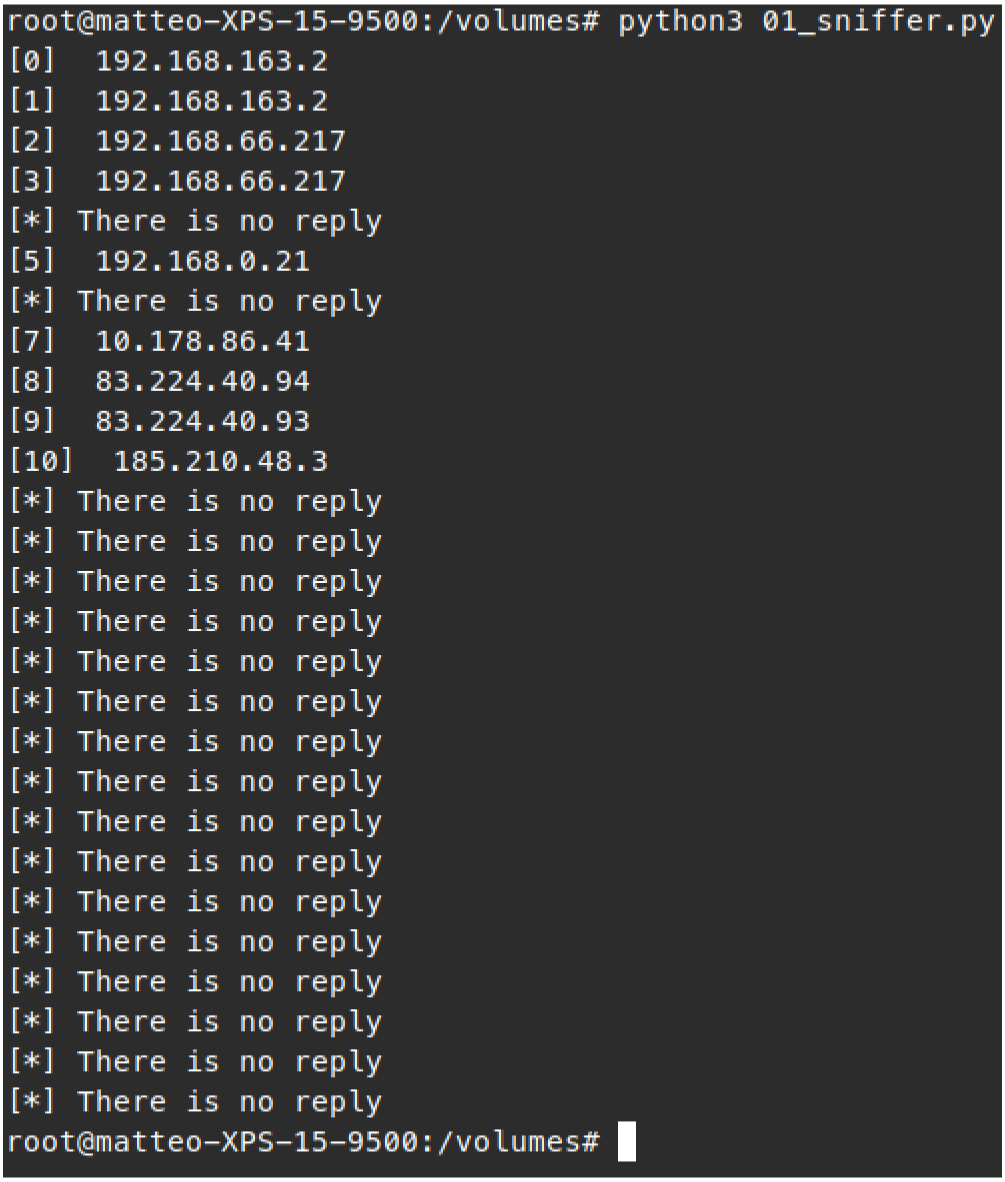
Packets captured by wireshark[the second one is spoofed by our tool].

## Task 1.3: Traceroute



```
# Task 1.3: Traceroute
    # ttl (Time-To-Live)attribute is present in IP packets.
        # Each time a machine receives an IP packet decrease ttl by 1.
        # This is used to avoid infinite loops.
hostname = "youtube.com"
dport = 80
counter = 27
for i in range(27):
    pkt = IP(dst=hostname, ttl=i) / UDP(dport=dport)
    # Send the packet and get a reply
    response = sr1(pkt, verbose=0, timeout=1)
    if response is None: # there is no reply!
        print("[*] There is no reply")
        continue

    elif response.type == 3: # destination reached
        print("Destination reached! ", response.src)
        break
    else: # we are in the path
        print("[{}] ".format(i), response.src)
```

We know that traceroute works by pinging each host through the network increasing the **ttl(time to live)** field. In this way is able to understand how many hops and which are the ip of the hosts along the path to connect to a specified host. The max number of hop is 30(loop), no more are possible. We can construct UDP packets over IP protocol on a specific destination port. The dns resolution of the uri(hostname) can be resolved directly from scapy framework, so we do not have to worry about it. We collect the response and we check if there is no response(None), or if the type of response is “destination reached” (value type 3) or if we get a response but without value type set to 3, so we get a packet from a host through the path.

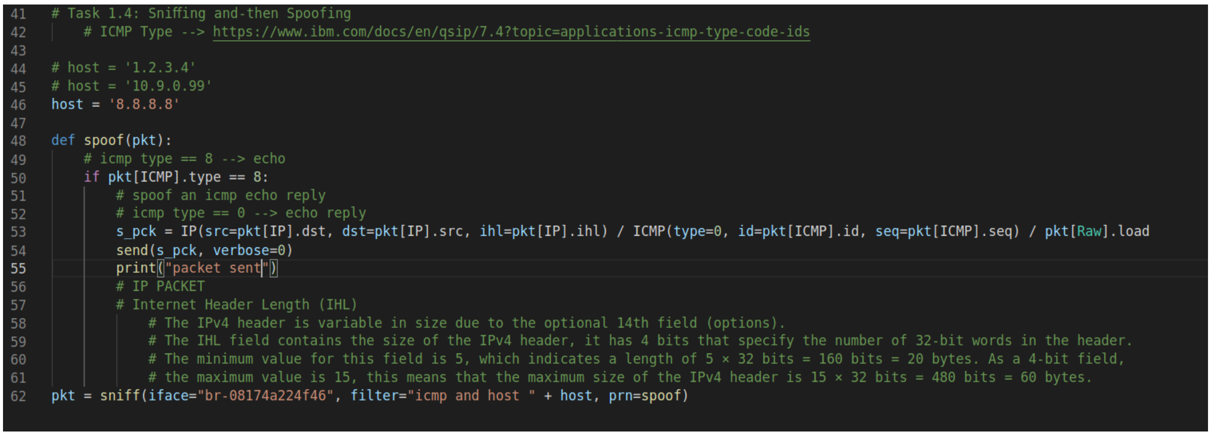


```
root@matteo-XPS-15-9500:/volumes# python3 01_sniffer.py
[0]  192.168.163.2
[1]  192.168.163.2
[2]  192.168.66.217
[3]  192.168.66.217
[*] There is no reply
[5]  192.168.0.21
[*] There is no reply
[7]  10.178.86.41
[8]  83.224.40.94
[9]  83.224.40.93
[10]  185.210.48.3
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
[*] There is no reply
root@matteo-XPS-15-9500:/volumes# 
```

output of the script.

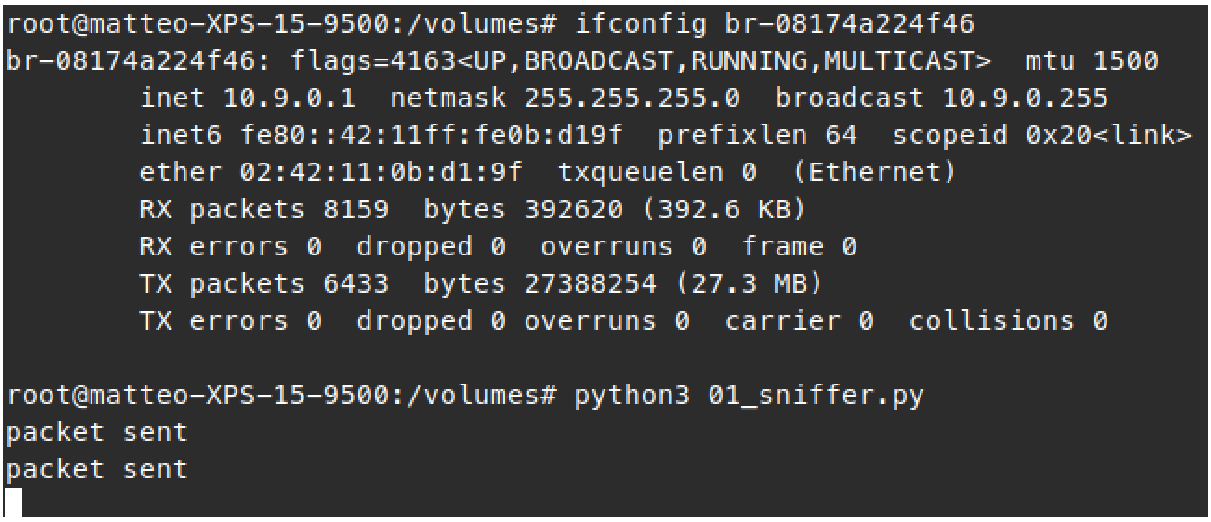
## Task 1.4: Sniffing and-then Spoofing

### PING 8.8.8.8



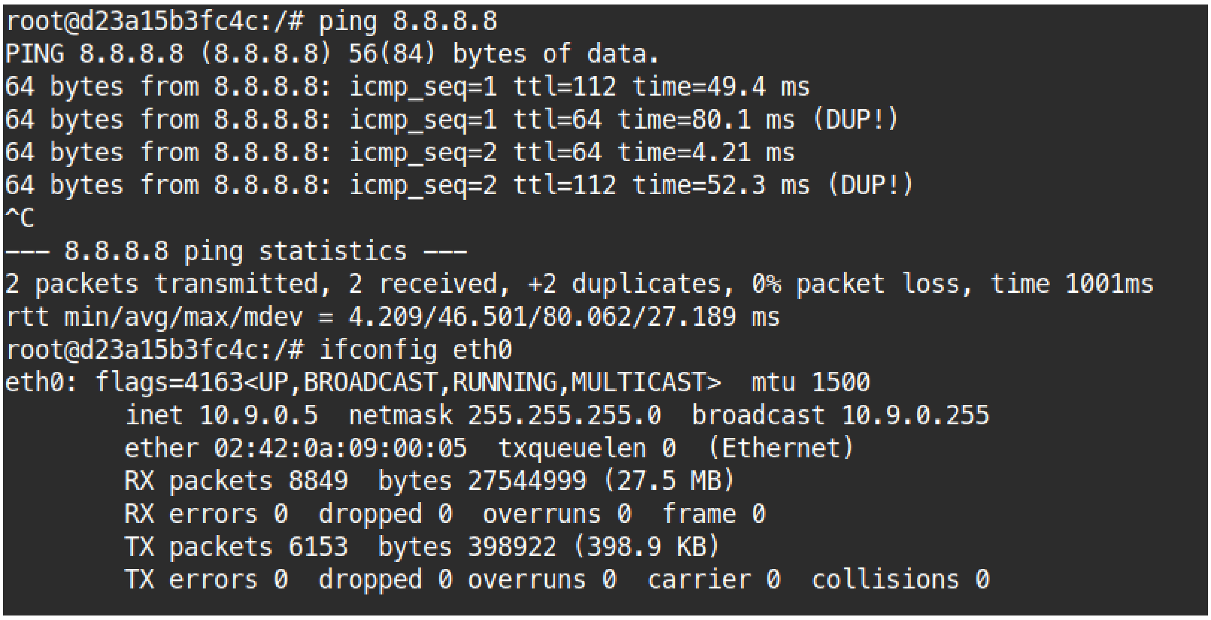
```python
# Task 1.4: Sniffing and-then Spoofing
    # ICMP Type --> https://www.ibm.com/docs/en/qsip/7.4?topic=applications-icmp-type-code-ids

# host = '1.2.3.4'
# host = '10.9.0.99'
host = '8.8.8.8'

def spoof(pkt):
    # icmp type == 8 --> echo
    if pkt[ICMP].type == 8:
        # spoof an icmp echo reply
        # icmp type == 0 --> echo reply
        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")
        # IP PACKET
        # Internet Header Length (IHL)
            # The IPv4 header is variable in size due to the optional 14th field (options).
            # The IHL field contains the size of the IPv4 header, it has 4 bits that specify the number of 32-bit words in the header.
            # The minimum value for this field is 5, which indicates a length of 5 × 32 bits = 160 bits = 20 bytes. As a 4-bit field,
            # the maximum value is 15, this means that the maximum size of the IPv4 header is 15 × 32 bits = 480 bits = 60 bytes.
pkt = sniff(iface="br-08174a224f46", filter="icmp and host " + host, prn=spoof)
```

In this piece of code the **sniff()** function is invoked on the interface indicated with a filter icmp and host 8.8.8.8, when sniff() function finds a match invokes the **spoof()** function passing as argument the packet that caused the match.



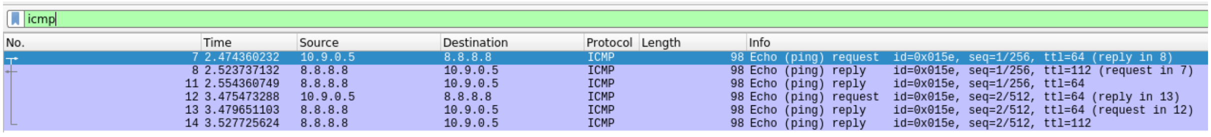
```
root@matteo-XPS-15-9500:/volumes# ifconfig br-08174a224f46
br-08174a224f46: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:11ff:fe0b:d19f  prefixlen 64  scopeid 0x20<link>
        ether 02:42:11:0b:d1:9f  txqueuelen 0  (Ethernet)
        RX packets 8159  bytes 392620 (392.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6433  bytes 27388254 (27.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@matteo-XPS-15-9500:/volumes# python3 01_sniffer.py
packet sent
packet sent
```

In this screen there is the tool started on the victim machine.



```
root@d23a15b3fc4c:/# 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=112 time=49.4 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=80.1 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=4.21 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=112 time=52.3 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, +2 duplicates, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 4.209/46.501/80.062/27.189 ms
root@d23a15b3fc4c:/# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 8849  bytes 27544999 (27.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6153  bytes 398922 (398.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

In this screen there is the ping output from 8.8.8.8 host.



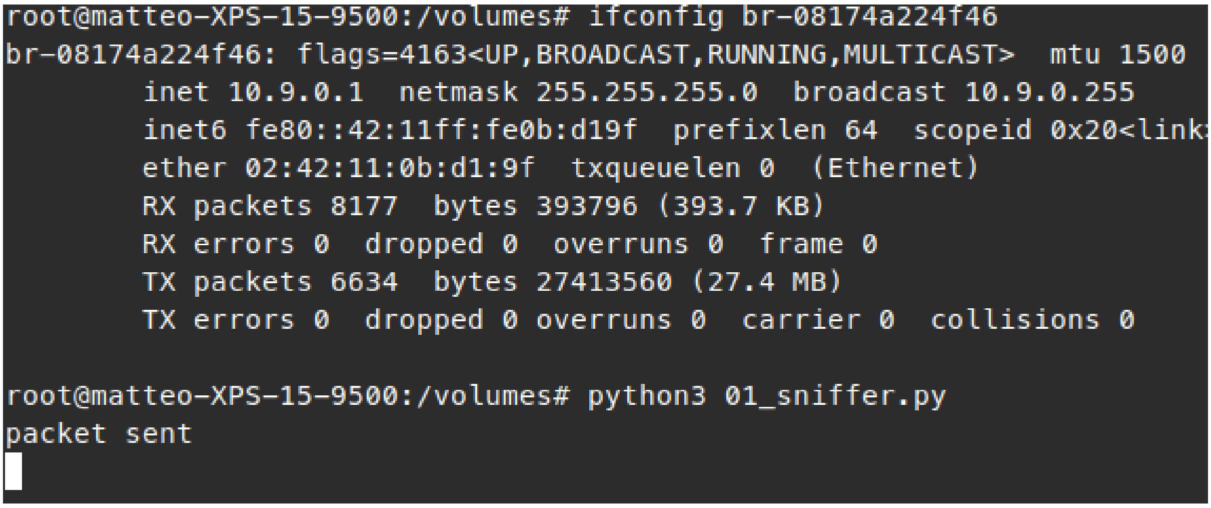
icmp

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 7 | 2.474360232 | 10.9.0.5 | 8.8.8.8 | ICMP | 98 | Echo (ping) request  id=0x015e, seq=1/256, ttl=64 (reply in 8) |
| 8 | 2.523737132 | 8.8.8.8 | 10.9.0.5 | ICMP | 98 | Echo (ping) reply    id=0x015e, seq=1/256, ttl=112 (request in 7) |
| 11 | 2.554360749 | 8.8.8.8 | 10.9.0.5 | ICMP | 98 | Echo (ping) reply    id=0x015e, seq=1/256, ttl=64 |
| 12 | 3.475473288 | 10.9.0.5 | 8.8.8.8 | ICMP | 98 | Echo (ping) request  id=0x015e, seq=2/512, ttl=64 (reply in 13) |
| 13 | 3.479651103 | 8.8.8.8 | 10.9.0.5 | ICMP | 98 | Echo (ping) reply    id=0x015e, seq=2/512, ttl=64 (request in 12) |
| 14 | 3.527725624 | 8.8.8.8 | 10.9.0.5 | ICMP | 98 | Echo (ping) reply    id=0x015e, seq=2/512, ttl=112 |

we can see that in this case, the icmp echo replies are duplicate, due to that the ping tool is saying to us “DUP!” to advise us of that! We can also see in wireshark about this problem, we can see that the packets of “icmp reply” are duplicate, one caming from the real server and the other spoofed by the attacker.
Opening the packet in wireshark we can see that the spoofed packets and the real packets have the same source mac address because they pass through the same interface.

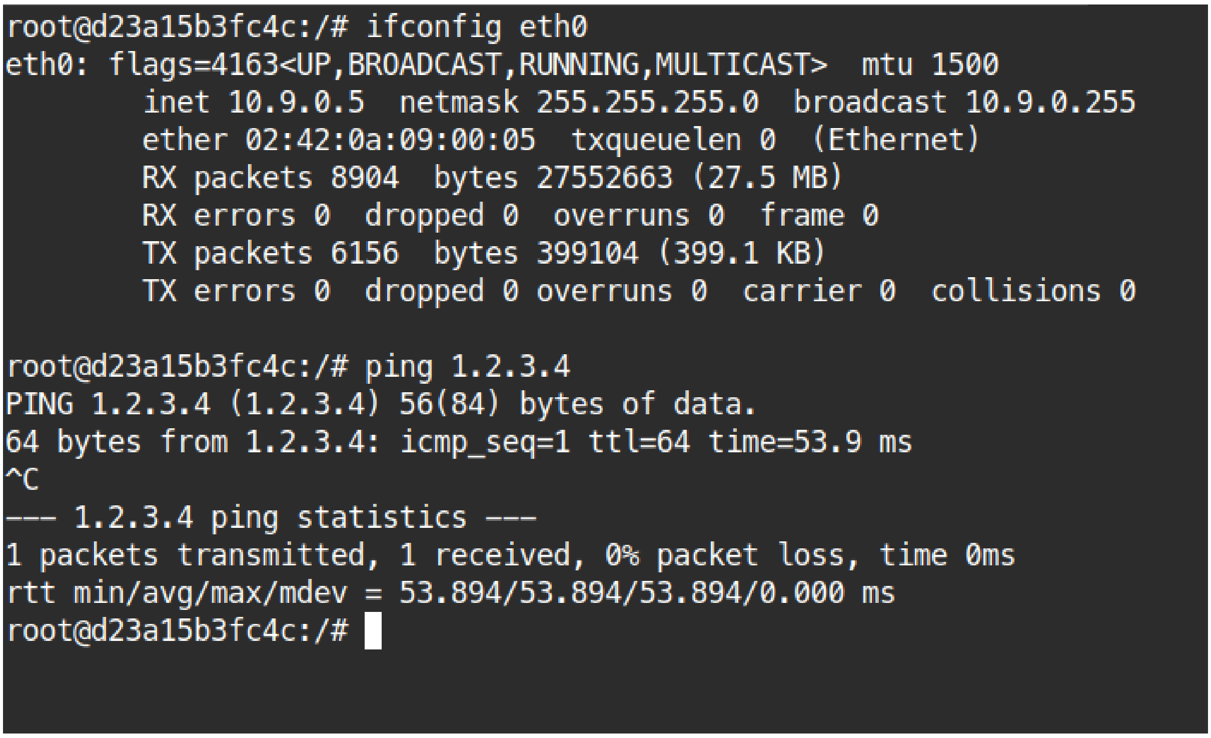
**PING 1.2.3.4**

The code is the same as before but we only change the host ip.



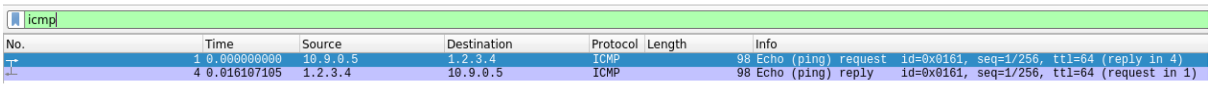
```
root@matteo-XPS-15-9500:/volumes# ifconfig br-08174a224f46
br-08174a224f46: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:11ff:fe0b:d19f  prefixlen 64  scopeid 0x20<link
        ether 02:42:11:0b:d1:9f  txqueuelen 0  (Ethernet)
        RX packets 8177  bytes 393796 (393.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6634  bytes 27413560 (27.4 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@matteo-XPS-15-9500:/volumes# python3 01_sniffer.py
packet sent
```

Screen of the tool running against host 1.2.3.4.



```
root@d23a15b3fc4c:/# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 8904  bytes 27552663 (27.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6156  bytes 399104 (399.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@d23a15b3fc4c:/# 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=53.9 ms
^C
--- 1.2.3.4 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 53.894/53.894/53.894/0.000 ms
root@d23a15b3fc4c:/# 
```

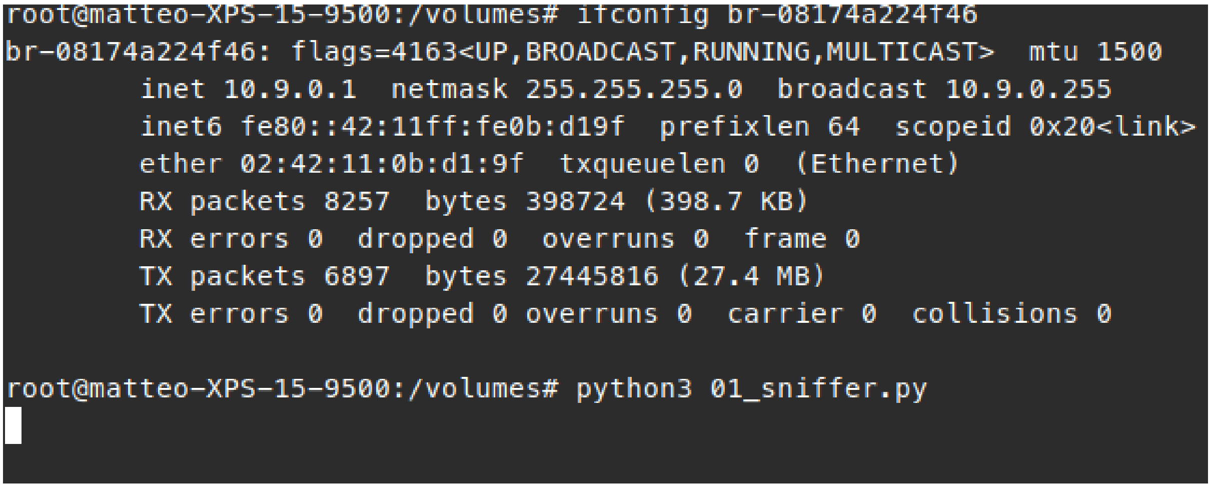
Important to note is that here, the victim thinks that this host is online also if it is not.



```
No.     Time          Source     Destination  Protocol  Length  Info
      1 0.000000000   10.9.0.5   1.2.3.4      ICMP          98  Echo (ping) request  id=0x0161, seq=1/256, ttl=64 (reply in 4)
      4 0.016107105   1.2.3.4    10.9.0.5     ICMP          98  Echo (ping) reply    id=0x0161, seq=1/256, ttl=64 (request in 1)
```

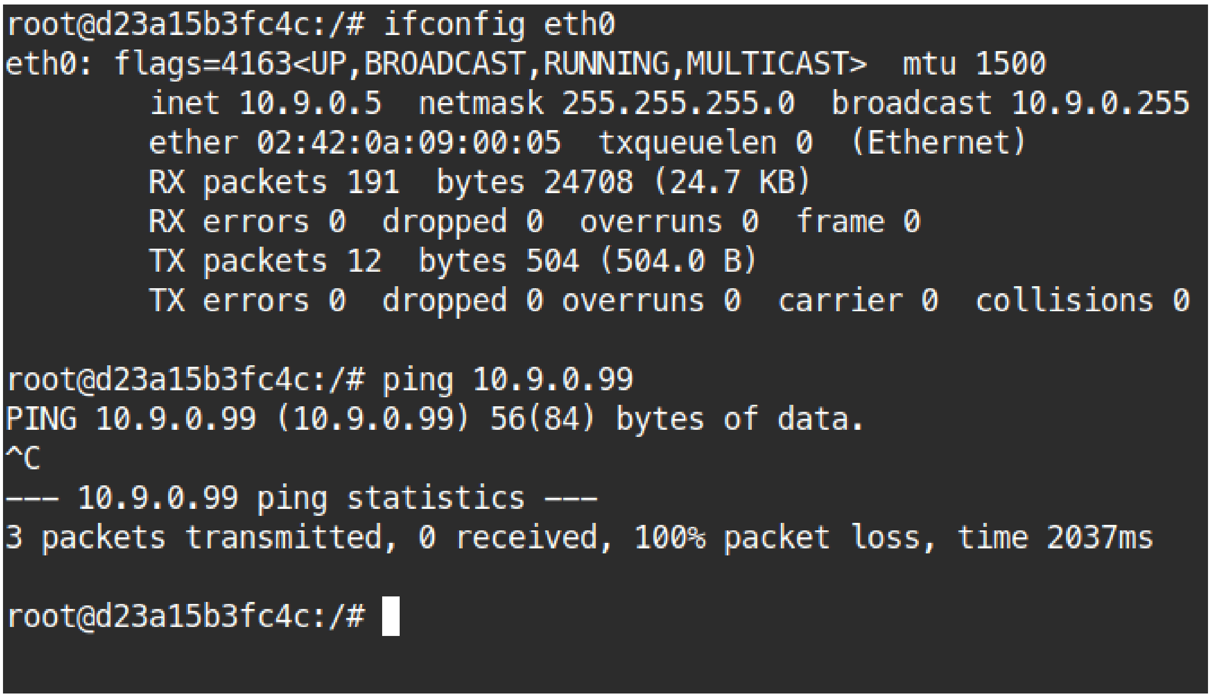
Wireshark shows the correct packet(spoofed) sent through the network.

**PING 10.9.0.99**



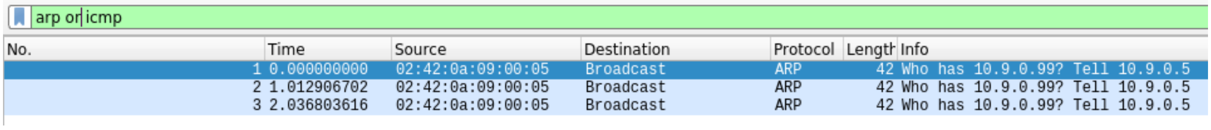
```
root@matteo-XPS-15-9500:/volumes# ifconfig br-08174a224f46
br-08174a224f46: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:11ff:fe0b:d19f  prefixlen 64  scopeid 0x20<link>
        ether 02:42:11:0b:d1:9f  txqueuelen 0  (Ethernet)
        RX packets 8257  bytes 398724 (398.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6897  bytes 27445816 (27.4 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@matteo-XPS-15-9500:/volumes# python3 01_sniffer.py
```

In this case the attacker cannot sniff any icmp packet due to the victim not the arp entry in the arp table corresponding to the ip 10.9.0.99.



```
root@d23a15b3fc4c:/# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 191  bytes 24708 (24.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 504 (504.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@d23a15b3fc4c:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
^C
--- 10.9.0.99 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2037ms

root@d23a15b3fc4c:/#
```

The victim knows that this ip is in the same subnetwork of its, so is looking for the resolution of the ip address to MAC address in a way to directly contact the machine.
As we can see there is no possibility for the victim to create an icmp packet without the arp entry of the ip 10.9.0.99



arp or icmp

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 02:42:0a:09:00:05 | Broadcast | ARP | 42 | Who has 10.9.0.99? Tell 10.9.0.5 |
| 2 | 1.012906702 | 02:42:0a:09:00:05 | Broadcast | ARP | 42 | Who has 10.9.0.99? Tell 10.9.0.5 |
| 3 | 2.036803616 | 02:42:0a:09:00:05 | Broadcast | ARP | 42 | Who has 10.9.0.99? Tell 10.9.0.5 |

The victim is asking the network who has the 10.9.0.99 ip, hoping for someone in the same subnetwork that knows which is the mac address associated with 10.9.0.99.

## Task 2.1A: Understanding How a Sniffer Works

- Question 1. Please use your own words to describe the sequence of the library calls that are essential for sniffer programs.
- 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?
- 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):

  ```
  # ip -d link show dev br-f2478ef5974
  ```