Packet Sniffing and Spoofing Lab

Ethical Hacking 2024/25

Nicola Ursino

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
MATEMATICA

Padua, October 16, 2024

Task 1: Using Scapy to Sniff and Spoof Packets

- Make sure the right **network interface** is used in the provided script
- Run it:
  - Without root privileges
  - With root priviledges
- Demonstrate it works

Provided initial script

```
#!/usr/bin/env python3
from scapy.all import *
defprint_pkt(pkt):
    pkt.show()
pkt = sniff(iface='eth0', filter='icmp', prn=print_pkt)
```

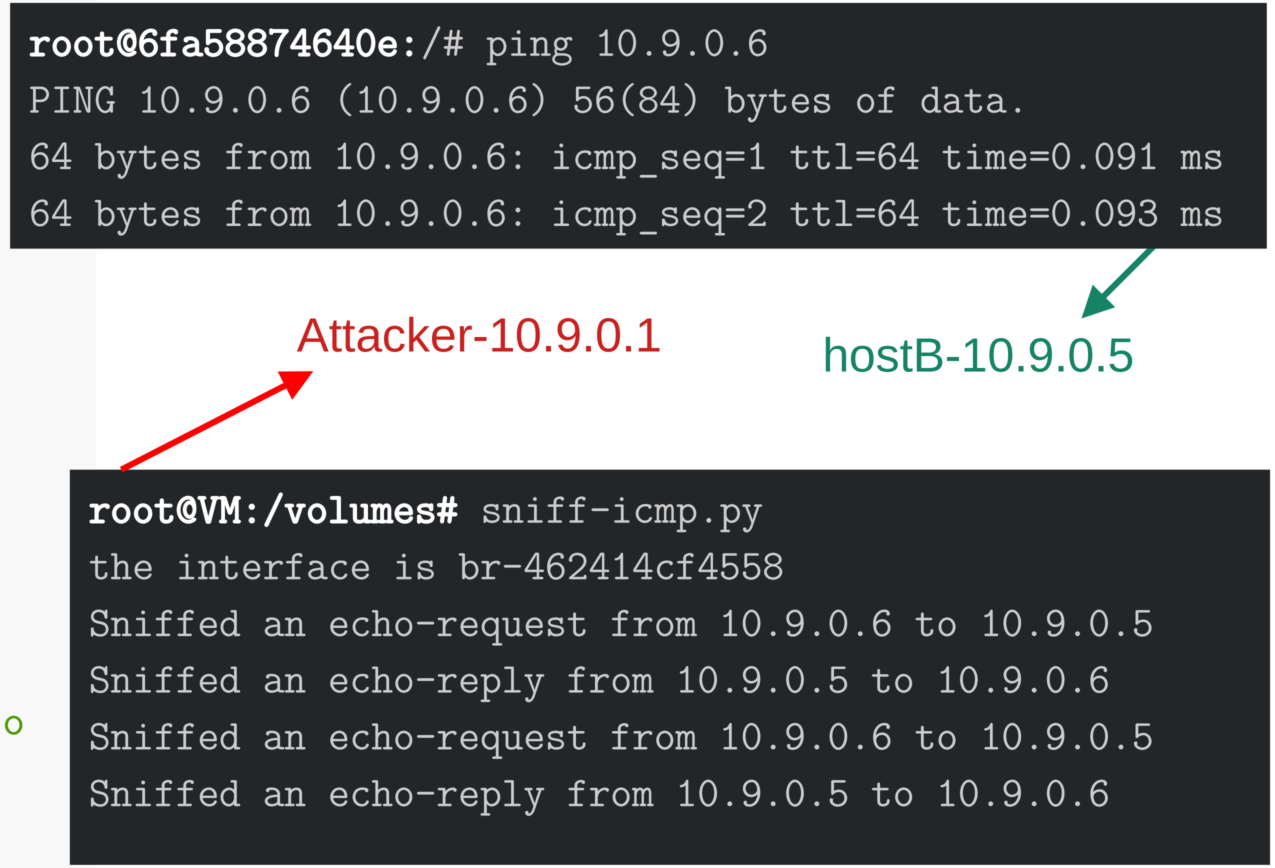
Task 1.1: Solution

```
def search_iface_by_prefx(search_string, string_list):
    matches = [item for item in string_list if search_string in item]
    return matches[0]
```

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

interface = search_iface_by_prefx("br-", get_if_list())
print (f'the interface is {interface}')

def print_pkt(pkt):
    icmp_type=""
    if (pkt[ICMP].type==8):
        icmp_type="echo-request"
    elif (pkt[ICMP].type==0):
        icmp_type="echo-reply"

    print(f'Received an {icmp_type} from {pkt[IP].src} to
{pkt[IP].dst}')

pkt = sniff(iface=interface, filter='icmp', prn=print_pkt)
```

Task 1.1: Solution: **Demonstration**

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

interface = search_iface_by_prefx("br-", get_if_list())
print (f'the interface is {interface}')

def print_pkt(pkt):
    icmp_type=""
    if (pkt[ICMP].type==8):
        icmp_type="echo-request"
    elif (pkt[ICMP].type==0):
        icmp_type="echo-reply"

    print(f'Received an {icmp_type} from {pkt[IP].src} to
{pkt[IP].dst}')

pkt = sniff(iface=interface, filter='icmp', prn=print_pkt)
```

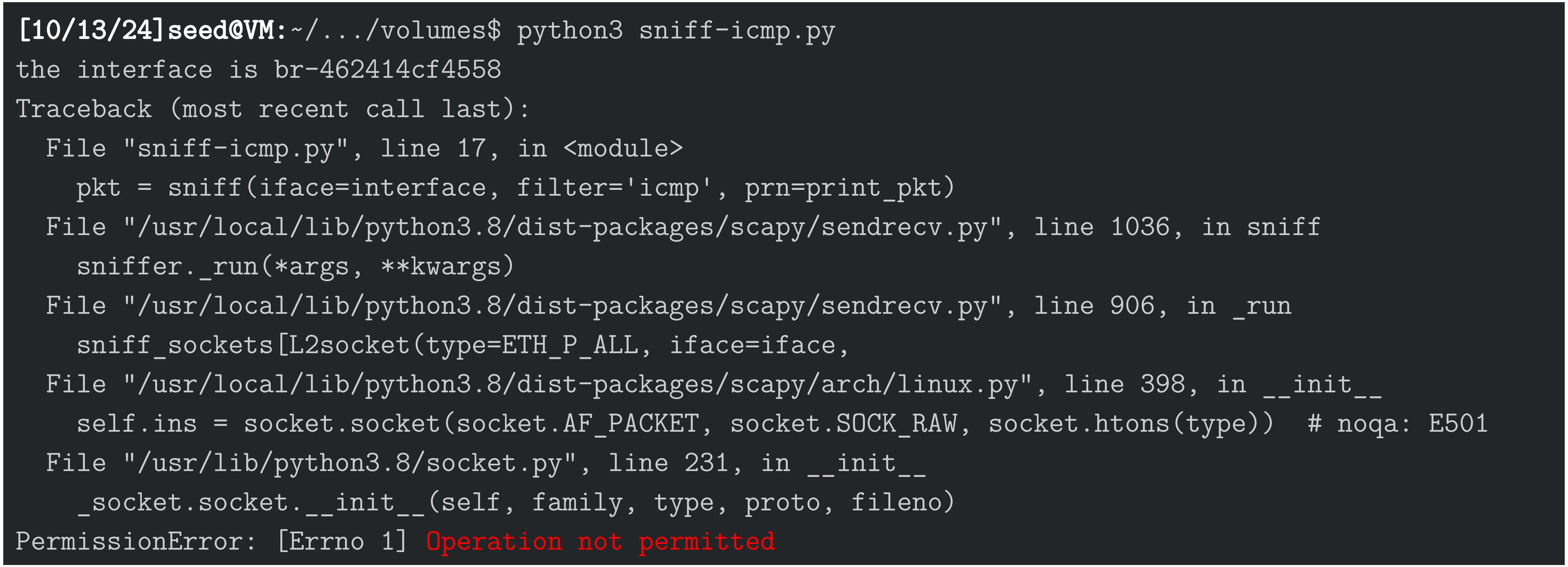


```
root@6fa58874640e:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.091 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.093 ms
```

```
root@VM:/volumes# sniff-icmp.py
the interface is br-462414cf4558
Sniffed an echo-request from 10.9.0.6 to 10.9.0.5
Sniffed an echo-reply from 10.9.0.5 to 10.9.0.6
Sniffed an echo-request from 10.9.0.6 to 10.9.0.5
Sniffed an echo-reply from 10.9.0.5 to 10.9.0.6
```

Task 1.1: Solution: **demonstration**

Without root priviledges



```
[10/13/24]seed@VM:~/.../volumes$ python3 sniff-icmp.py
the interface is br-462414cf4558
Traceback (most recent call last):
  File "sniff-icmp.py", line 17, in <module>
    pkt = sniff(iface=interface, 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
```

Task 1.1B: play with filters

Task

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

- Use Berkeley Packet Filters

Task 1.1B: play with filters

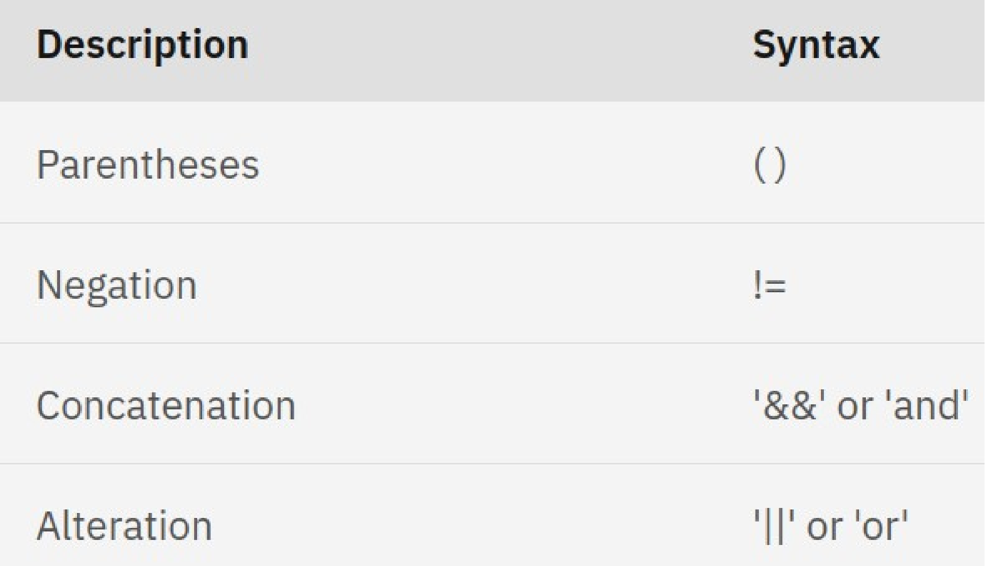
Task

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

- Use Berkeley Packet Filters

BPF syntax

- `[src|dst] host <host>`
- `[tcp|udp] [src|dst] port <port>`



Description	Syntax
Parentheses	( )
Negation	!=
Concatenation	'&&' or 'and'
Alteration	'||' or 'or'

Source:https://www.ibm.com/docs/en/qsip/7.4?topic=queries-berkeley-packet-filters

Task 1.1B: play with filters: **Solution**

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

interface = search_iface_by_prefx("br-", get_if_list())
print (f'the interface is {interface}')

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface=interface, filter='tcp dst port 23 and src host 10.9.0.6', prn=print_pkt)
```

Task 1.1B: play with filters: **Solution**

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

interface = search_iface_by_prefx("br-", get_if_list())
print (f'the interface is {interface}')

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface=interface, filter='tcp dst port 23 and src host 10.9.0.6', prn=print_pkt)
```

Demostrate it works

- How can we produce tcp packets using port 23?

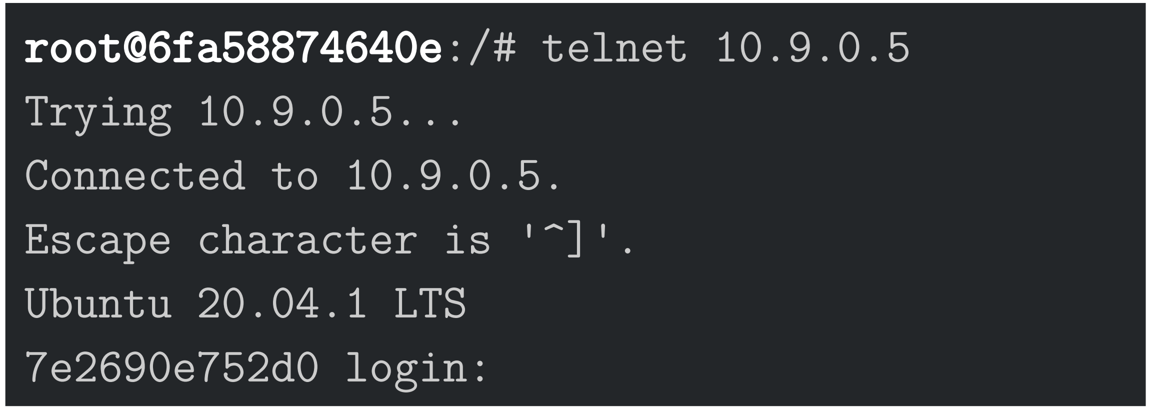
Task 1.1B: play with filters: **Solution**

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

interface = search_iface_by_prefx("br-", get_if_list())
print (f'the interface is {interface}')

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface=interface, filter='tcp dst port 23 and src host 10.9.0.6', prn=print_pkt)
```

Demostrate it works

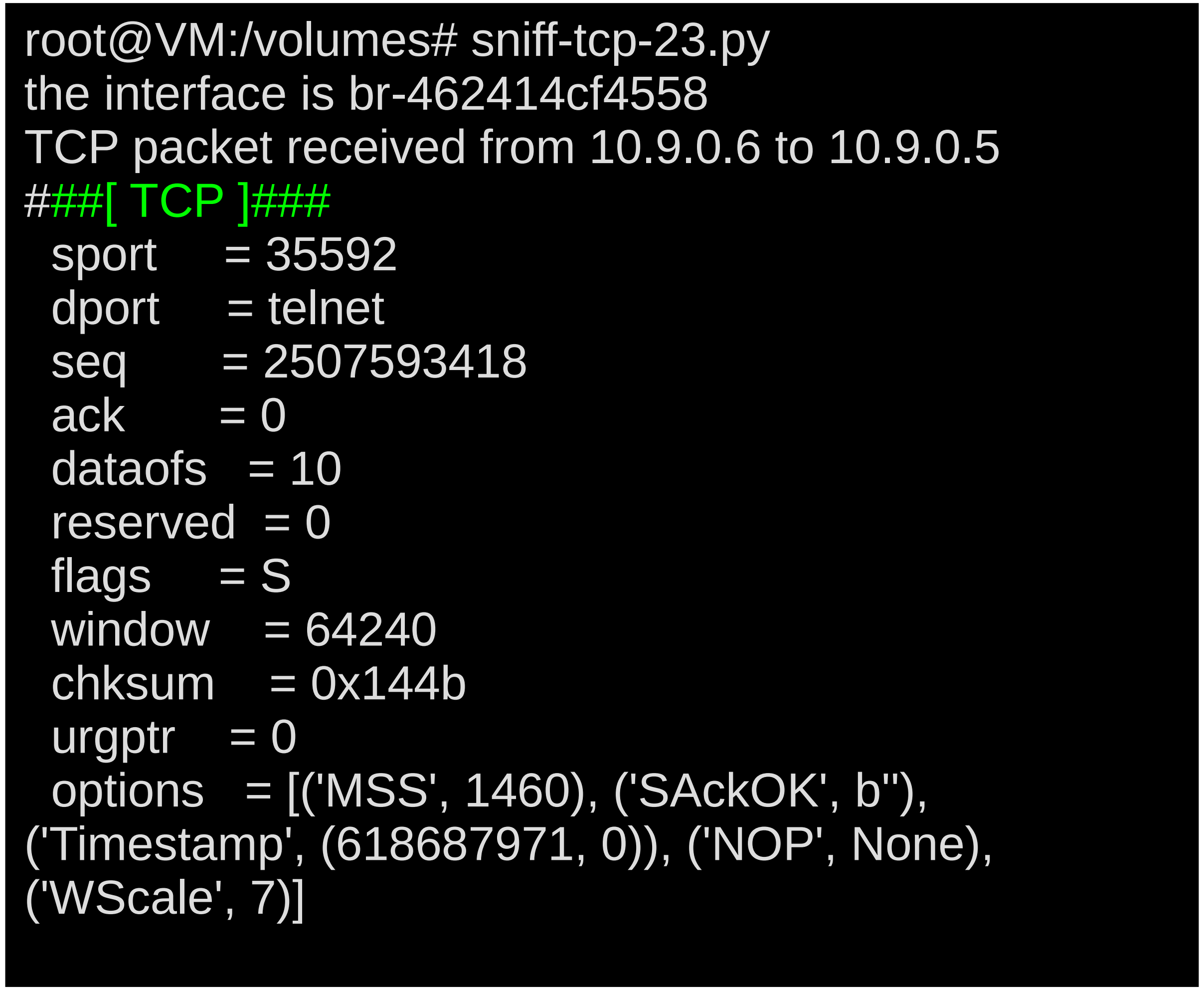
- How can we produce tcp packets using port 23?

What about using TELNET?

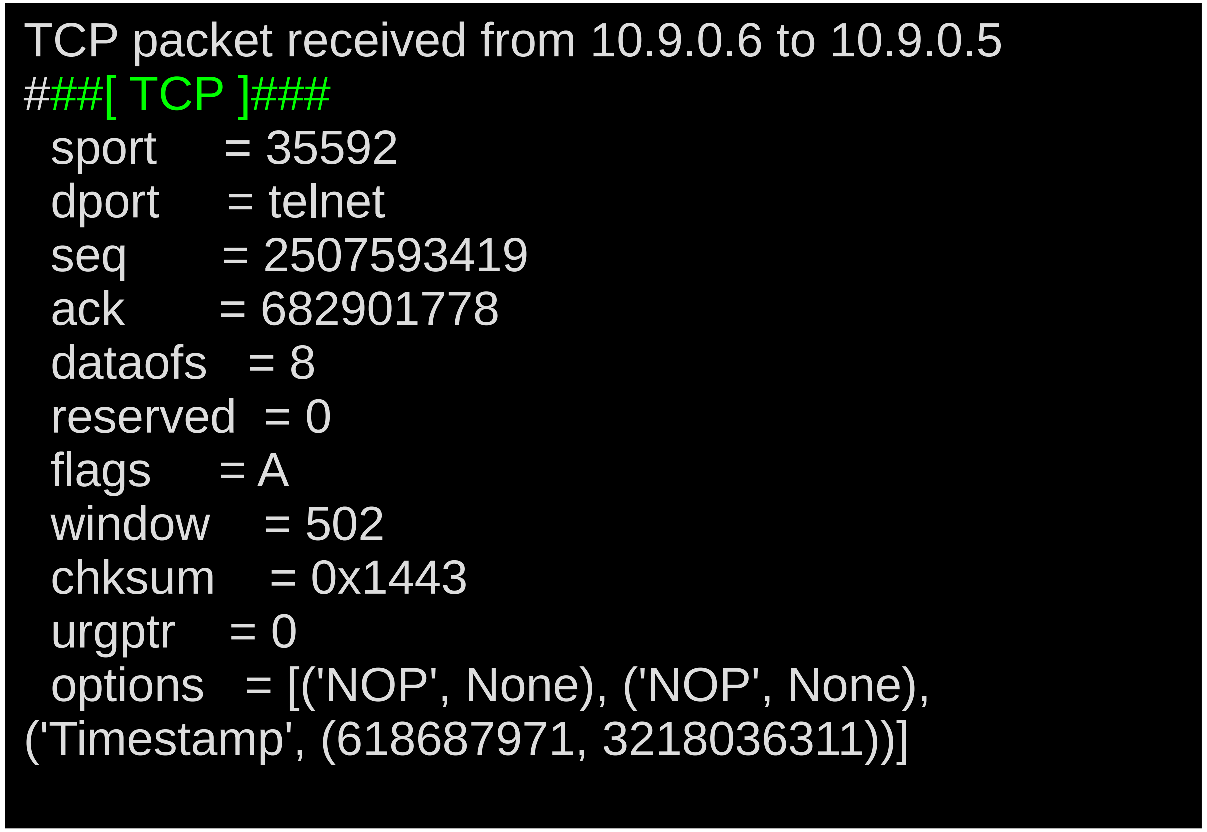


```
root@6fa58874640e:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
7e2690e752d0 login:
```

Task 1.1B: play with filters: **Demonstration**



```
root@VM:/volumes# sniff-tcp-23.py
the interface is br-462414cf4558
TCP packet received from 10.9.0.6 to 10.9.0.5
###[ TCP ]###
  sport     = 35592
  dport     = telnet
  seq       = 2507593418
  ack       = 0
  dataofs   = 10
  reserved  = 0
  flags     = S
  window    = 64240
  chksum    = 0x144b
  urgptr    = 0
  options   = [('MSS', 1460), ('SAckOK', b''),
('Timestamp', (618687971, 0)), ('NOP', None),
('WScale', 7)]
```



```
TCP packet received from 10.9.0.6 to 10.9.0.5
###[ TCP ]###
  sport     = 35592
  dport     = telnet
  seq       = 2507593419
  ack       = 682901778
  dataofs   = 8
  reserved  = 0
  flags     = A
  window    = 502
  chksum    = 0x1443
  urgptr    = 0
  options   = [('NOP', None), ('NOP', None),
('Timestamp', (618687971, 3218036311))]
```

18 messages just to get to the login prompt!

Task 1.2: Spoofing ICMP Packets

Task:

- spoof an ICMP echo request packet and send it to another VM on the same network
- use Wireshark to observe whether our request will be accepted by the receiver.

Provided initial script

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

ip = IP()
ip.dst = '10.0.0.1'
icmp = ip/ICMP()
sendp(icmp)
```

Task 1.2: Spoofing ICMP Packets: **First attempt**

Easy! Just add the **source** and **destination** IP addresses!

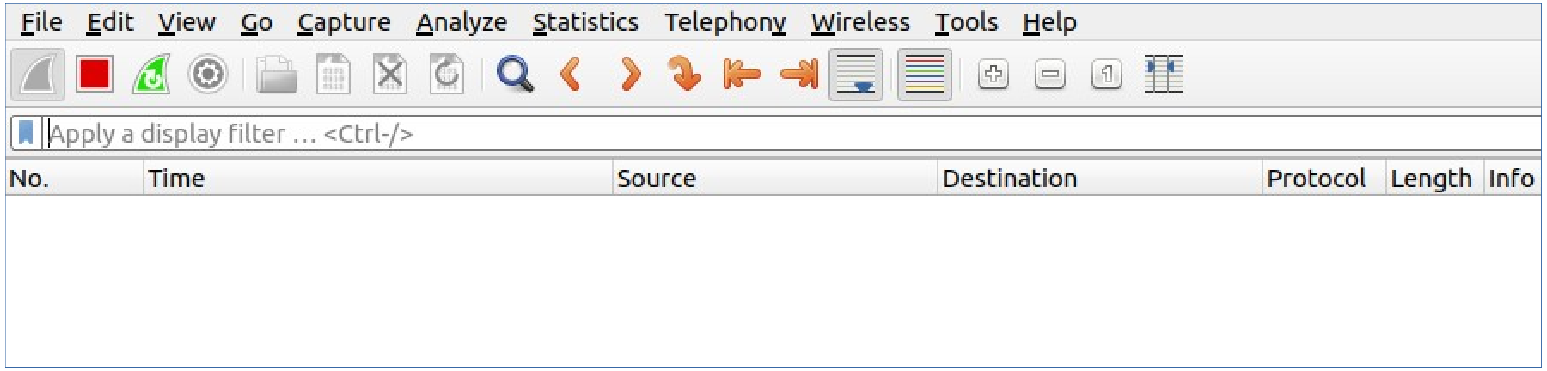
```
#!/usr/bin/env python3
from scapy.all import *

ip = IP()
ip.src = '10.9.0.6'
ip.dst = '10.9.0.5'

icmp = ICMP()
packet = ip/icmp
packet.show()

sendp(packet)
```

Task 1.2: Spoofing ICMP Packets: **First attempt**

Easy! Just add the **source** and **destination** IP addresses!

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

ip = IP()
ip.src = '10.9.0.6'
ip.dst = '10.9.0.5'

icmp = ICMP()
packet = ip/icmp
packet.show()

sendp(packet)
```

But it does not work... ☹



Task 1.2: Spoofing ICMP Packets: **Documentation**

- The **`send()`** function will send packets at layer 3

- The **`sendp()`** function will work at layer 2. It’s up to you to choose the right interface and the right link layer protocol

Task 1.2: Spoofing ICMP Packets: **Documentation**

- The **`send()`** function will send packets at layer 3

Fix 1

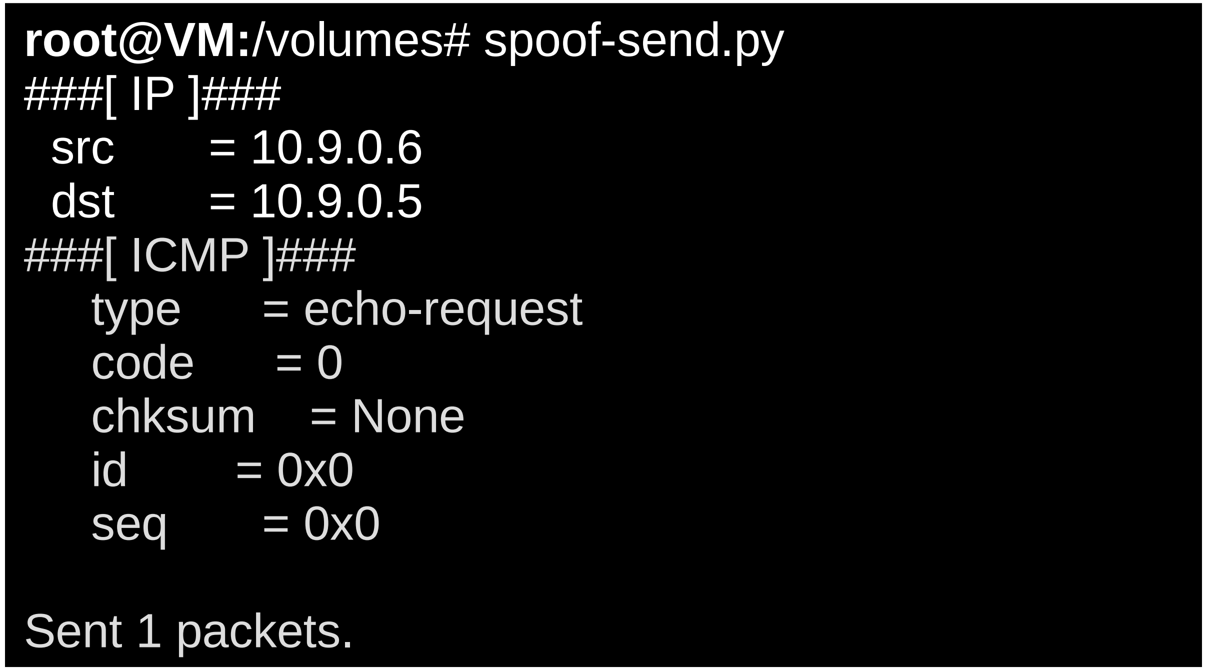
```
icmp = ICMP()
packet = ip/icmp
packet.show()

send(packet)
```

- The **`sendp()`** function will work at layer 2. It’s up to you to choose the right interface and the right link layer protocol

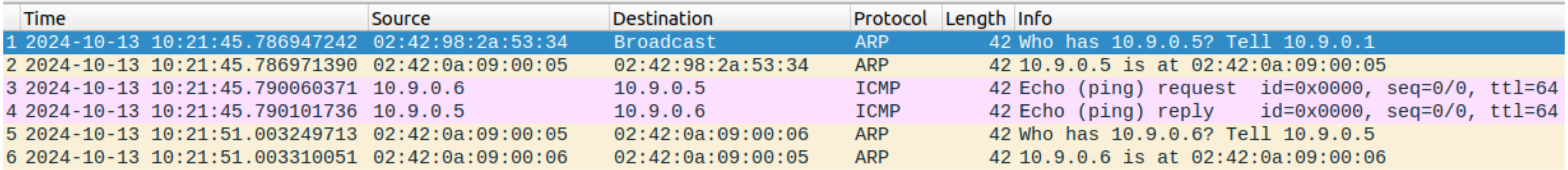
Fix 2

```
icmp = ICMP()
packet = Ether()/ip/icmp
packet.show()

sendp(packet, iface=interface)
```

Task 1.2: Spoofing ICMP Packets: **Demonstration**



```
root@VM:/volumes# spoof-send.py
###[ IP ]###
  src       = 10.9.0.6
  dst       = 10.9.0.5
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = None
     id        = 0x0
     seq       = 0x0

Sent 1 packets.
```



	Time	Source	Destination	Protocol	Length	Info
1	2024-10-13 10:21:45.786947242	02:42:98:2a:53:34	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
2	2024-10-13 10:21:45.786971390	02:42:0a:09:00:05	02:42:98:2a:53:34	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
3	2024-10-13 10:21:45.790060371	10.9.0.6	10.9.0.5	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64
4	2024-10-13 10:21:45.790101736	10.9.0.5	10.9.0.6	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=64
5	2024-10-13 10:21:51.003249713	02:42:0a:09:00:05	02:42:0a:09:00:06	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
6	2024-10-13 10:21:51.003310051	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06

Task 1.3: Traceroute

Task:

- Use Scapy to **estimate the distance**, in terms of **number of routers**, between your VM and a selected destination
- Keep sending icmp packets increasing the **ttl** each time.
- Every time the **ttl** expires, a server will answer with a notification reveiling its address

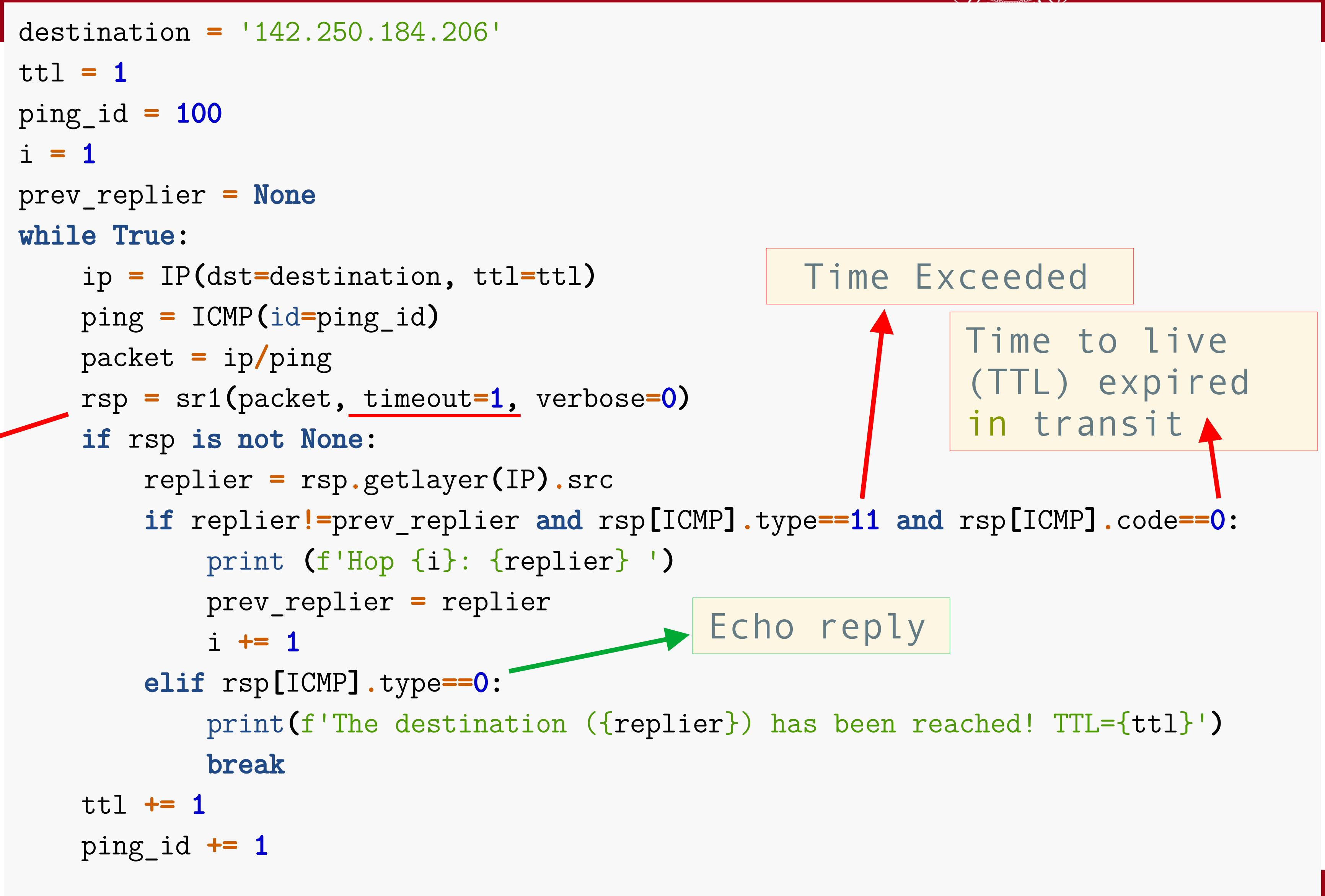
Provided initial script

```
a = IP()
a.dst = '1.2.3.4'
a.ttl = 3
b = ICMP()
send(a/b)
```

Task 1.3: Traceroute: **Solution**

- **ICMP id** is necessary, otherwise we will not get any ping reply from the server. It works without in the same LAN

- Sometimes, with certain **ttl**s the ping request gets lost and receives no response. This is why We need a timeout!



```
destination = '142.250.184.206'
ttl = 1
ping_id = 100
i = 1
prev_replier = None
while True:
    ip = IP(dst=destination, ttl=ttl)
    ping = ICMP(id=ping_id)
    packet = ip/ping
    rsp = sr1(packet, timeout=1, verbose=0)
    if rsp is not None:
        replier = rsp.getlayer(IP).src
        if replier!=prev_replier and rsp[ICMP].type==11 and rsp[ICMP].code==0:
            print (f'Hop {i}: {replier} ')
            prev_replier = replier
            i += 1
        elif rsp[ICMP].type==0:
            print(f'The destination ({replier}) has been reached! TTL={ttl}')
            break
    ttl += 1
    ping_id += 1
```

Time Exceeded

Time to live (TTL) expired in transit

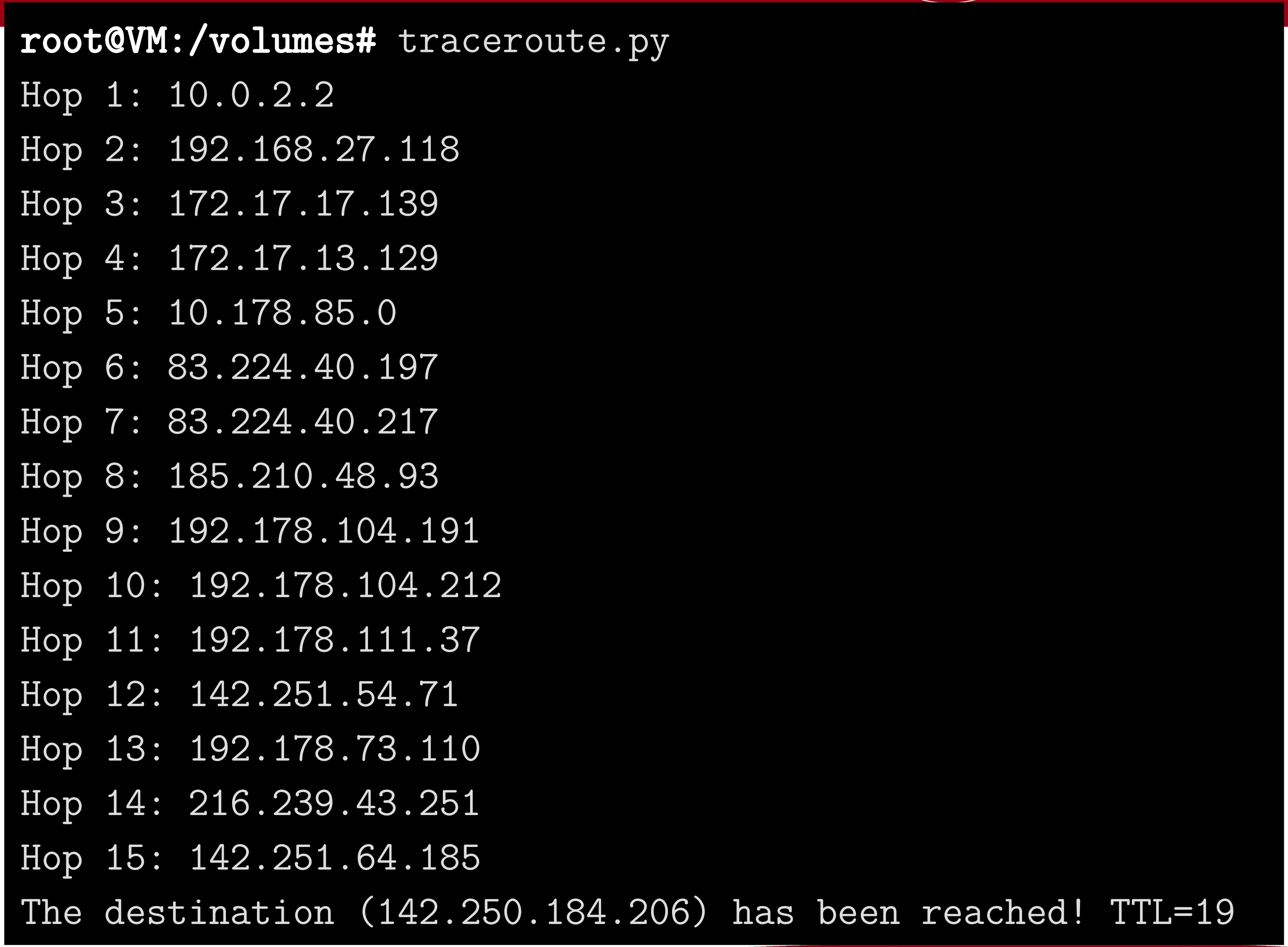
Echo reply

Task 1.3: Traceroute: **Demonstration**

Considerations:

- As also noted in the Lab instructions, Eduroam blocks consecutive ICMP requests

- Sharing my phone connection with the laptop fixed the problem



```
root@VM:/volumes# traceroute.py
Hop 1: 10.0.2.2
Hop 2: 192.168.27.118
Hop 3: 172.17.17.139
Hop 4: 172.17.13.129
Hop 5: 10.178.85.0
Hop 6: 83.224.40.197
Hop 7: 83.224.40.217
Hop 8: 185.210.48.93
Hop 9: 192.178.104.191
Hop 10: 192.178.104.212
Hop 11: 192.178.111.37
Hop 12: 142.251.54.71
Hop 13: 192.178.73.110
Hop 14: 216.239.43.251
Hop 15: 142.251.64.185
The destination (142.250.184.206) has been reached! TTL=19
```

Task 1.4: Sniffing and-then Spoofing

Task:

- From the **user container**, you ping an IP **X**
- Build a script that sniffs for ICMP packets and replies to ICMP requests spoofing the packets. **Run the script on the VM**
- Thanks to the script, the ping program will always receive a ping back, even if the IP **X** does not exist

Task 1.4: Sniffing and-then Spoofing

Task:

- From the **user container**, you ping an IP **X**
- Build a script that sniffs for ICMP packets and replies to ICMP requests spoofing the packets. **Run the script on the VM**
- Thanks to the script, the ping program will always receive a ping back, even if the IP **X** does not exist

But first…

Task 1.4: Sniffing and-then Spoofing

Task:

- From the **user container**, you ping an IP **X**
- Build a script that sniffs for ICMP packets and replies to ICMP requests spoofing the packets. **Run the script on the VM**
- Thanks to the script, the ping program will always receive a ping back, even if the IP **X** does not exist

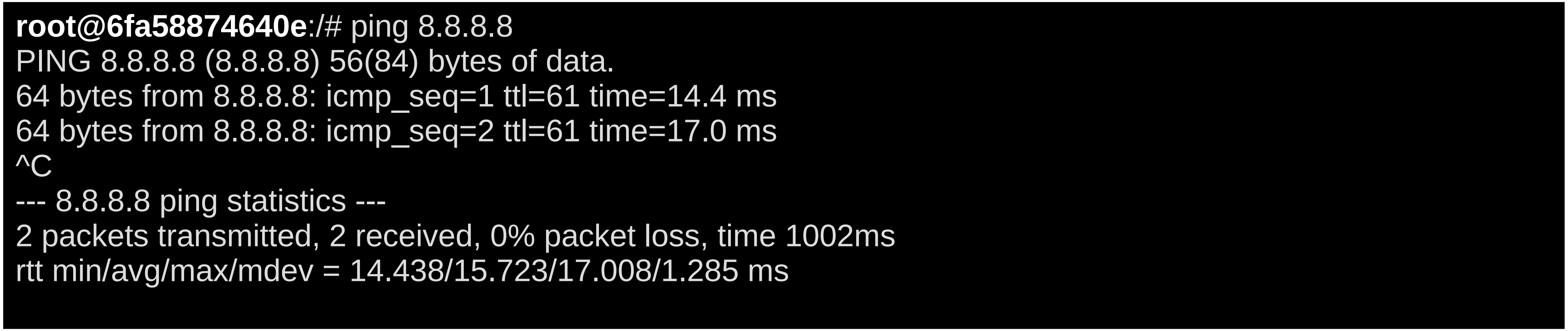
But first...

```
ping 8.8.8.8    # an existing host on the Internet
ping 1.2.3.4    # a non-existing host on the Internet
ping 10.9.0.99 # a non-existing host on the LAN
```

And see what happens...

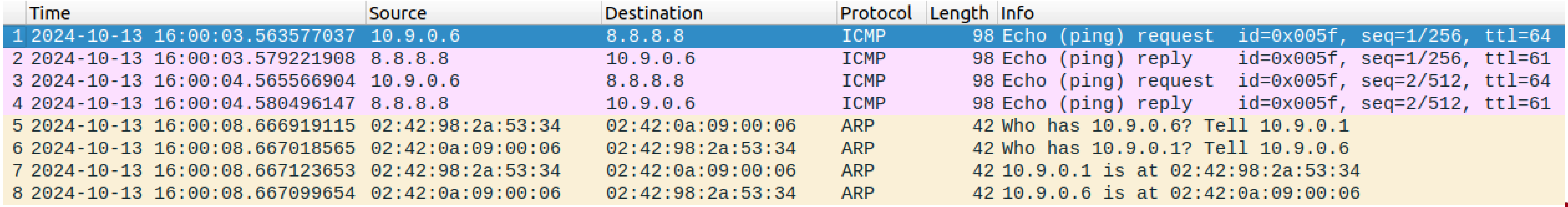
Task 1.4: Sniffing and-then Spoofing: **8.8.8.8**

Host B 10.9.0.6



```
root@6fa58874640e:/# 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=61 time=14.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=61 time=17.0 ms
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 14.438/15.723/17.008/1.285 ms
```

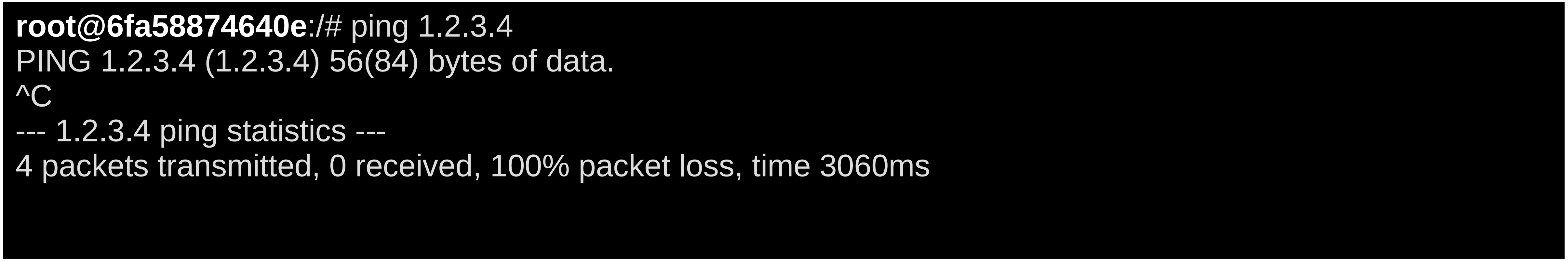
Wireshark



	Time	Source	Destination	Protocol	Length	Info
1	2024-10-13 16:00:03.563577037	10.9.0.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x005f, seq=1/256, ttl=64
2	2024-10-13 16:00:03.579221908	8.8.8.8	10.9.0.6	ICMP	98	Echo (ping) reply id=0x005f, seq=1/256, ttl=61
3	2024-10-13 16:00:04.565566904	10.9.0.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x005f, seq=2/512, ttl=64
4	2024-10-13 16:00:04.580496147	8.8.8.8	10.9.0.6	ICMP	98	Echo (ping) reply id=0x005f, seq=2/512, ttl=61
5	2024-10-13 16:00:08.666919115	02:42:98:2a:53:34	02:42:0a:09:00:06	ARP	42	Who has 10.9.0.6? Tell 10.9.0.1
6	2024-10-13 16:00:08.667018565	02:42:0a:09:00:06	02:42:98:2a:53:34	ARP	42	Who has 10.9.0.1? Tell 10.9.0.6
7	2024-10-13 16:00:08.667123653	02:42:98:2a:53:34	02:42:0a:09:00:06	ARP	42	10.9.0.1 is at 02:42:98:2a:53:34
8	2024-10-13 16:00:08.667099654	02:42:0a:09:00:06	02:42:98:2a:53:34	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06

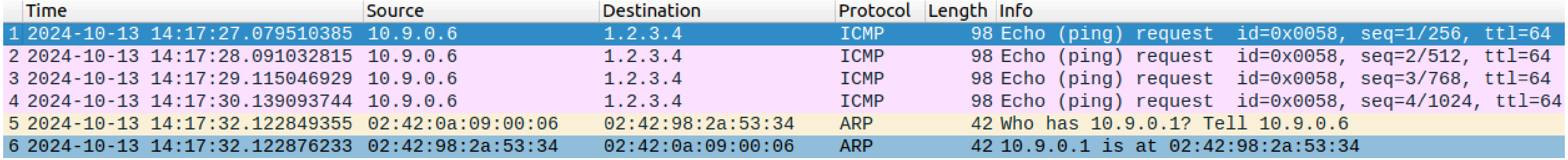
Task 1.4: Sniffing and-then Spoofing: **1.2.3.4**

Host B 10.9.0.6



```
root@6fa58874640e:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
^C
--- 1.2.3.4 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3060ms
```

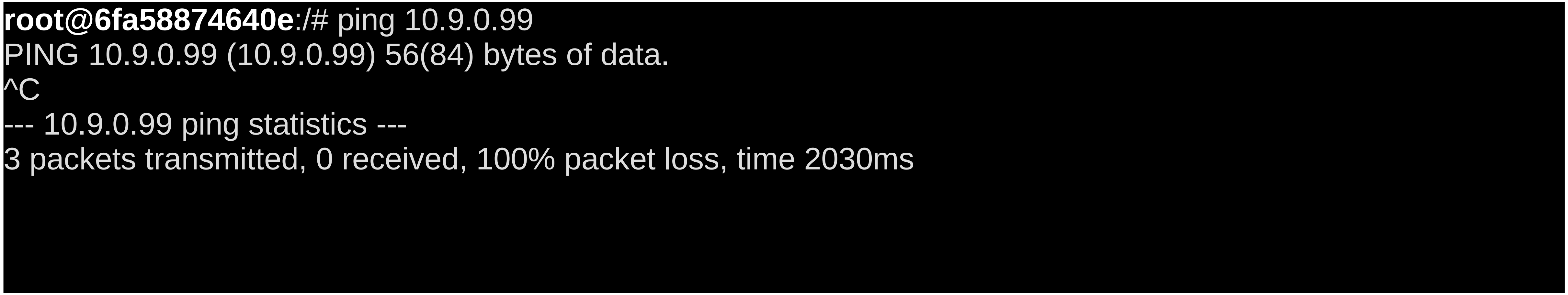
Wireshark



	Time	Source	Destination	Protocol	Length	Info
1	2024-10-13 14:17:27.079510385	10.9.0.6	1.2.3.4	ICMP	98	Echo (ping) request id=0x0058, seq=1/256, ttl=64
2	2024-10-13 14:17:28.091032815	10.9.0.6	1.2.3.4	ICMP	98	Echo (ping) request id=0x0058, seq=2/512, ttl=64
3	2024-10-13 14:17:29.115046929	10.9.0.6	1.2.3.4	ICMP	98	Echo (ping) request id=0x0058, seq=3/768, ttl=64
4	2024-10-13 14:17:30.139093744	10.9.0.6	1.2.3.4	ICMP	98	Echo (ping) request id=0x0058, seq=4/1024, ttl=64
5	2024-10-13 14:17:32.122849355	02:42:0a:09:00:06	02:42:98:2a:53:34	ARP	42	Who has 10.9.0.1? Tell 10.9.0.6
6	2024-10-13 14:17:32.122876233	02:42:98:2a:53:34	02:42:0a:09:00:06	ARP	42	10.9.0.1 is at 02:42:98:2a:53:34

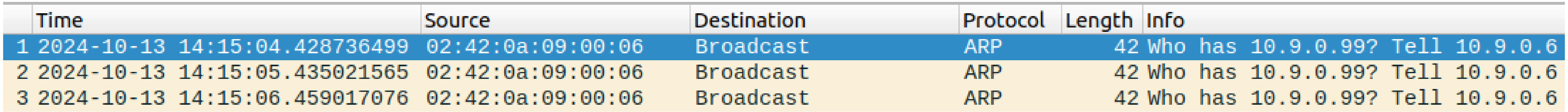
Task 1.4: Sniffing and-then Spoofing: **10.9.0.99**

Host B 10.9.0.6



```
root@6fa58874640e:/# 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 2030ms
```

Wireshark



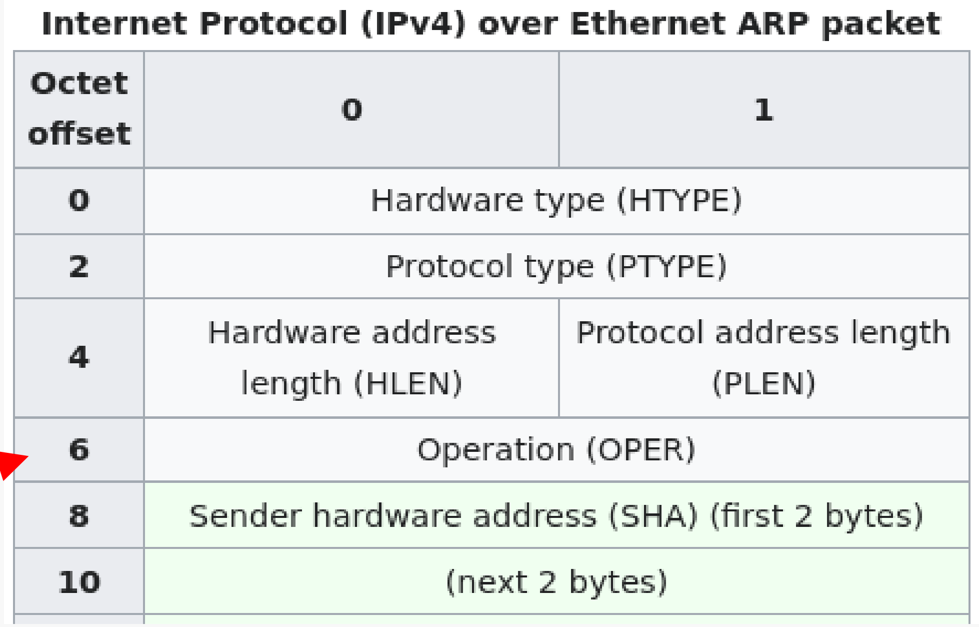
	Time	Source	Destination	Protocol	Length	Info
1	2024-10-13 14:15:04.428736499	02:42:0a:09:00:06	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.6
2	2024-10-13 14:15:05.435021565	02:42:0a:09:00:06	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.6
3	2024-10-13 14:15:06.459017076	02:42:0a:09:00:06	Broadcast	ARP	42	Who has 10.9.0.99? Tell 10.9.0.6

Task 1.4: Sniffing and-then Spoofing: **Solution**

```python
def handle_sniffing(pkt):
    if ARP in pkt:  # ARP request
        print ("an ARP request arrived!")
        pkt.show()
        arp_reply = ARP(op=2, hwsrc=mac_address, psrc=pkt[ARP].pdst,hwdst=pkt[ARP].hwsrc, pdst=pkt[ARP].psrc)
        send(arp_reply, verbose=False)
        print(f"Sent ARP reply for {pkt[ARP].pdst}")
        arp_reply.show()
    else: # ICMP request
        print ("an ICMP request arrived!")
        pkt.show()
        ip_reply = IP(src=pkt[IP].dst, dst=pkt[IP].src)
        icmp_reply = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        fullreply = ip_reply/icmp_reply/pkt[Raw]
        send(fullreply, verbose=False)
        print ("ICMP reply sent!")
        fullreply.show()

pkt = sniff(iface=interface, filter='arp[6:2] = 1 or icmp[icmptype] = 8', prn=handle_sniffing)
```

Task 1.4: Sniffing and-then Spoofing: **Solution**

```
def handle_sniffing(pkt):
    if ARP in pkt:  # ARP request
        print ("an ARP request arrived!")
        pkt.show()
        arp_reply = ARP(op=2, hwsrc=mac_address, psrc=pkt[ARP].pdst,hwdst=pkt[ARP].hwsrc, pdst=pkt[ARP].psrc)
        send(arp_reply, verbose=False)
        print(f"Sent ARP reply for {pkt[ARP].pdst}")
        arp_reply.show()
    else: # ICMP request
        print ("an ICMP request arrived!")
        pkt.show()
        ip_reply = IP(src=pkt[IP].dst, dst=pkt[IP].src)
        icmp_reply = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        fullreply = ip_reply/icmp_reply/pkt[Raw]
        send(fullreply, verbose=False)
        print ("ICMP reply sent!")
        fullreply.show()

pkt = sniff(iface=interface, filter='arp[6:2] = 1 or icmp[icmptype] = 8', prn=handle_sniffing)
```



Internet Protocol (IPv4) over Ethernet ARP packet

Octet offset	0	1
0	Hardware type (HTYPE)	
2	Protocol type (PTYPE)	
4	Hardware address length (HLEN)	Protocol address length (PLEN)
6	Operation (OPER)	
8	Sender hardware address (SHA) (first 2 bytes)	
10	(next 2 bytes)	

Task 1.4: Sniffing and-then Spoofing: **Solution**

```
def handle_sniffing(pkt):
    if ARP in pkt:  # ARP request
        print ("an ARP request arrived!")
        pkt.show()
        arp_reply = ARP(op=2, hwsrc=mac_address, psrc=pkt[ARP].pdst,hwdst=pkt[ARP].hwsrc, pdst=pkt[ARP].psrc)
        send(arp_reply, verbose=False)
        print(f"Sent ARP reply for {pkt[ARP].pdst}")
        arp_reply.show()
    else: # ICMP request
        print ("an ICMP request arrived!")
        pkt.show()
        ip_reply = IP(src=pkt[IP].dst, dst=pkt[IP].src)
        icmp_reply = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        fullreply = ip_reply/icmp_reply/pkt[Raw]
        send(fullreply, verbose=False)
        print ("ICMP reply sent!")
        fullreply.show()

pkt = sniff(iface=interface, filter='arp[6:2] = 1 or icmp[icmptype] = 8', prn=handle_sniffing)
```

```
interface = search_iface_by_prefx("br-", get_if_list())
print (f'the interface is {interface}')
mac_address = get_if_hwaddr(interface)
```

Task 1.4: Sniffing and-then Spoofing: **Demonstration**

I’ll give you a live demo for this one 🙂

```
sudo ip -s -s neigh flush all
```

→ You can use this command to flush the arp cache

THANK YOU FOR YOUR ATTENTION

Questions?

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
MATEMATICA