

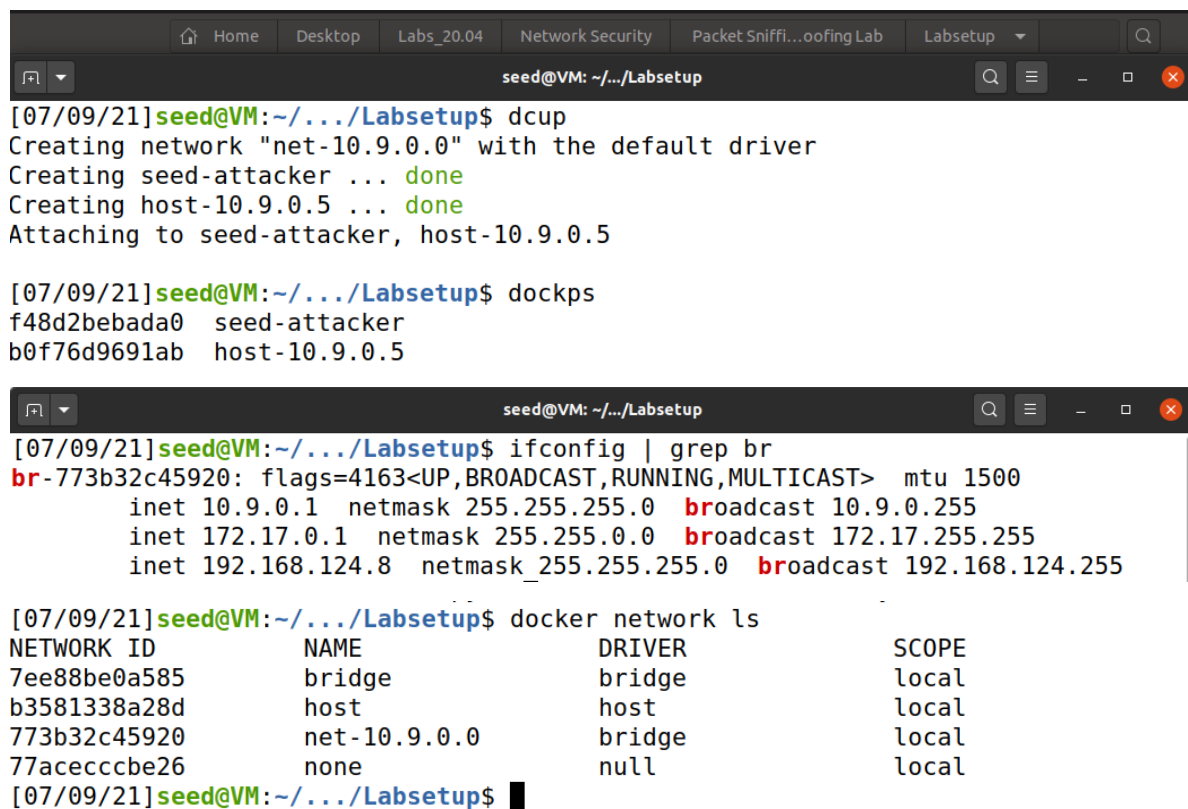
# Lab1

57118202 付孜

## Task 1.1: Sniffing Packets

### Task 1.1A

启动docker, 查看网络ID以及容器ID。



The image shows two terminal windows from a VM named 'seed@VM'. The first window shows the execution of 'dcup' to create a network 'net-10.9.0.0' and attach to the 'seed-attacker' container. The second window shows the output of 'dockps' and 'ifconfig | grep br', displaying network details for the bridge 'br-773b32c45920' and the Docker network 'net-10.9.0.0'.

```
[07/09/21]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating seed-attacker ... done
Creating host-10.9.0.5 ... done
Attaching to seed-attacker, host-10.9.0.5

[07/09/21]seed@VM:~/.../Labsetup$ dockps
f48d2bebada0  seed-attacker
b0f76d9691ab  host-10.9.0.5

[07/09/21]seed@VM:~/.../Labsetup$ ifconfig | grep br
br-773b32c45920: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
    inet 172.17.0.1 netmask 255.255.0.0  broadcast 172.17.255.255
    inet 192.168.124.8 netmask 255.255.255.0  broadcast 192.168.124.255

[07/09/21]seed@VM:~/.../Labsetup$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
7ee88be0a585        bridge              bridge              local
b3581338a28d        host                host                local
773b32c45920        net-10.9.0.0        bridge              local
77acecccb26         none                null                local
[07/09/21]seed@VM:~/.../Labsetup$
```

使用Scapy在Python程序中进行包嗅探的代码 `sniffer.py` :

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

def print_pkt(pkt):
    pkt.show()

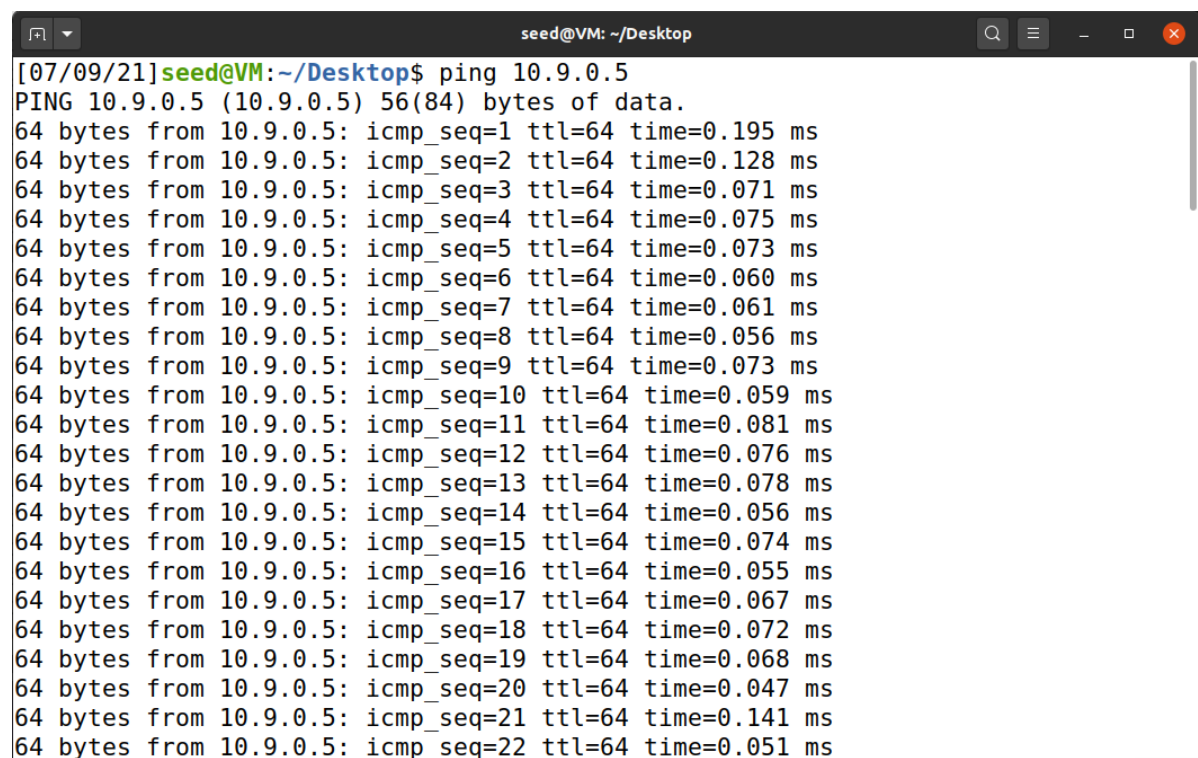
pkt = sniff(iface='br-773b32c45920', filter='icmp', prn=print_pkt)
```

使用root权限运行该程序, 并在另一个terminal中对主机IP进行ping命令, 运行结果如下图所示。

```

[07/09/21]seed@VM:~/Desktop$ touch sniffer.py
[07/09/21]seed@VM:~/Desktop$ chmod a+x sniffer.py
[07/09/21]seed@VM:~/Desktop$ sudo python3 sniffer.py
#### Ethernet ####
    dst      = 02:42:0a:09:00:05
    src      = 02:42:f2:e6:b5:75
    type     = IPv4
#### IP ####
    version  = 4
    ihl      = 5
    tos      = 0x0
    len      = 84
    id       = 63946
    flags    = DF
    frag     = 0
    ttl      = 64
    proto    = icmp
    chksum   = 0x2cc7
    src      = 10.9.0.1
    dst      = 10.9.0.5
    \options \
#### ICMP ####

```



```

[07/09/21]seed@VM:~/Desktop$ ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.195 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.128 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=64 time=0.071 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=64 time=0.075 ms
64 bytes from 10.9.0.5: icmp_seq=5 ttl=64 time=0.073 ms
64 bytes from 10.9.0.5: icmp_seq=6 ttl=64 time=0.060 ms
64 bytes from 10.9.0.5: icmp_seq=7 ttl=64 time=0.061 ms
64 bytes from 10.9.0.5: icmp_seq=8 ttl=64 time=0.056 ms
64 bytes from 10.9.0.5: icmp_seq=9 ttl=64 time=0.073 ms
64 bytes from 10.9.0.5: icmp_seq=10 ttl=64 time=0.059 ms
64 bytes from 10.9.0.5: icmp_seq=11 ttl=64 time=0.081 ms
64 bytes from 10.9.0.5: icmp_seq=12 ttl=64 time=0.076 ms
64 bytes from 10.9.0.5: icmp_seq=13 ttl=64 time=0.078 ms
64 bytes from 10.9.0.5: icmp_seq=14 ttl=64 time=0.056 ms
64 bytes from 10.9.0.5: icmp_seq=15 ttl=64 time=0.074 ms
64 bytes from 10.9.0.5: icmp_seq=16 ttl=64 time=0.055 ms
64 bytes from 10.9.0.5: icmp_seq=17 ttl=64 time=0.067 ms
64 bytes from 10.9.0.5: icmp_seq=18 ttl=64 time=0.072 ms
64 bytes from 10.9.0.5: icmp_seq=19 ttl=64 time=0.068 ms
64 bytes from 10.9.0.5: icmp_seq=20 ttl=64 time=0.047 ms
64 bytes from 10.9.0.5: icmp_seq=21 ttl=64 time=0.141 ms
64 bytes from 10.9.0.5: icmp_seq=22 ttl=64 time=0.051 ms

```

如果以seed用户而不用root权限运行 `sniffer.py`，系统会报错，如下图所示。

```

[07/09/21]seed@VM:~/Desktop$ python3 sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 7, in <module>
    pkt = sniff(iface='br-773b32c45920', 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))
  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

- 只抓取ICMP报文，见Task 1.1A所示。
- 捕获任何来自特定IP的TCP数据包，目的端口为23。

将 `sniffer.py` 修改为：

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-773b32c45920', filter='tcp port 23 and host 10.9.0.5',
prn=print_pkt)
```

利用docksh获取host的shell，telnet任意一个IP地址建立连接，并在运行 `sniffer.py` 的terminal中查看捕获的tcp数据包。

```
[07/09/21]seed@VM:~/.../Labsetup$ docksh b0
root@b0f76d9691ab:/# telnet 1.1.1.1
Trying 1.1.1.1...
telnet: Unable to connect to remote host: Connection timed out
```

```
seed@VM: ~/Desktop
[07/09/21]seed@VM:~/Desktop$ sudo python3 sniffer.py
###[ Ethernet ]###
  dst      = 02:42:f2:e6:b5:75
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x10
  len      = 60
  id       = 22747
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  chksum   = 0xd5c1
  src      = 10.9.0.5
  dst      = 1.1.1.1
  \options \
###[ TCP ]###
  sport    = 49620
  dport    = telnet
  seq      = 324818138
  ack      = 0
```

- 捕获来自或去特定子网的数据包。可以选择任何子网，如128.230.0.0/16；不应该选择VM所连接的子网。

将 `sniffer.py` 修改为：

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

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-773b32c45920', filter='host 10.9.0.3', prn=print_pkt)
```

ping 10.9.0.3, 并在运行 `sniffer.py` 的 terminal 中查看捕获的 tcp 数据包。

```
seed@VM: ~/Desktop
[07/09/21]seed@VM:~/Desktop$ ping 10.9.0.3
PING 10.9.0.3 (10.9.0.3) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=4 Destination Host Unreachable
From 10.9.0.1 icmp_seq=5 Destination Host Unreachable
From 10.9.0.1 icmp_seq=6 Destination Host Unreachable
From 10.9.0.1 icmp_seq=7 Destination Host Unreachable
From 10.9.0.1 icmp_seq=8 Destination Host Unreachable
From 10.9.0.1 icmp_seq=9 Destination Host Unreachable
From 10.9.0.1 icmp_seq=10 Destination Host Unreachable
From 10.9.0.1 icmp_seq=11 Destination Host Unreachable
From 10.9.0.1 icmp_seq=12 Destination Host Unreachable
From 10.9.0.1 icmp_seq=13 Destination Host Unreachable
From 10.9.0.1 icmp_seq=14 Destination Host Unreachable
From 10.9.0.1 icmp_seq=15 Destination Host Unreachable
From 10.9.0.1 icmp_seq=16 Destination Host Unreachable
From 10.9.0.1 icmp_seq=17 Destination Host Unreachable
From 10.9.0.1 icmp_seq=18 Destination Host Unreachable
From 10.9.0.1 icmp_seq=19 Destination Host Unreachable
From 10.9.0.1 icmp_seq=20 Destination Host Unreachable
From 10.9.0.1 icmp_seq=21 Destination Host Unreachable
From 10.9.0.1 icmp_seq=22 Destination Host Unreachable
From 10.9.0.1 icmp_seq=23 Destination Host Unreachable
From 10.9.0.1 icmp_seq=24 Destination Host Unreachable
From 10.9.0.1 icmp_seq=25 Destination Host Unreachable

^C[07/09/21]seed@VM:~/Desktop$
[07/09/21]seed@VM:~/Desktop$ sudo python3 sniffer.py
###[ Ethernet ]###
  dst      = ff:ff:ff:ff:ff:ff
  src      = 02:42:f2:e6:b5:75
  type     = ARP
###[ ARP ]###
  hwtype   = 0x1
  ptype    = IPv4
  hwlen    = 6
  plen     = 4
  op       = who-has
  hwsrc    = 02:42:f2:e6:b5:75
  psrc     = 10.9.0.1
  hwdst    = 00:00:00:00:00:00
  pdst     = 10.9.0.3

###[ Ethernet ]###
  dst      = ff:ff:ff:ff:ff:ff
  src      = 02:42:f2:e6:b5:75
  type     = ARP
###[ ARP ]###
  hwtype   = 0x1
  ptype    = IPv4
```

## Task 1.2: Spoofing ICMP Packets

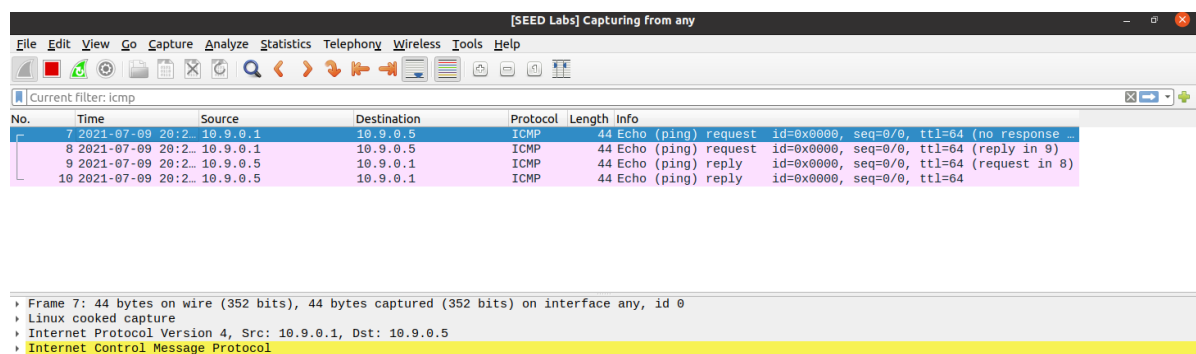
欺骗 ICMP 数据包的代码 `spoofer.py` :

```
#!/usr/bin/env python3
from scapy.all import *
a = IP()                               //line 1
a.dst = '10.9.0.5'
b = ICMP()
p = a/b
send(p)
ls(a)
```

第一行创建一个IP对象，第二行设置目标IP地址字段。第三行创建了一个ICMP对象，默认类型为echo request。在第四行中，我们将a和b堆叠在一起形成了一个新对象，“/”操作符被重载，不再表示除法，它意味着将b添加为a的有效负载字段，并相应地修改a的字段。

最终我们得到一个表示ICMP数据包的新对象，报文重组后，向子网内的一个IP发送数据包，打开Wireshark可观测到发送数据包和响应数据包。

```
[07/09/21]seed@VM:~/Desktop$ sudo python3 spoofer.py
Sent 1 packets.
version      : BitField  (4 bits)      = 4          (4)
ihl          : BitField  (4 bits)      = None       (None)
tos          : XByteField              = 0          (0)
len          : ShortField              = None       (None)
id           : ShortField              = 1          (1)
flags        : FlagsField  (3 bits)    = <Flag 0 (>) (<Flag 0 (>))
frag         : BitField  (13 bits)     = 0          (0)
ttl          : ByteField               = 64         (64)
proto        : ByteEnumField           = 0          (0)
chksum       : XShortField             = None       (None)
src          : SourceIPField           = '10.9.0.1' (None)
dst          : DestIPField             = '10.9.0.5' (None)
options      : PacketListField         = []         ([])
```



## Task 1.3: Traceroute

使用Scapy来估计虚拟机与选定目标之间的路由器数量之间的距离。编写工具 `trace.py`：

```
#!/usr/bin/env python3
from scapy.all import *
a = IP()
b = ICMP()
a.dst = '1.2.3.4'
for i in range(30):
    a.ttl = i + 1
    p = a / b
    send(p)
```

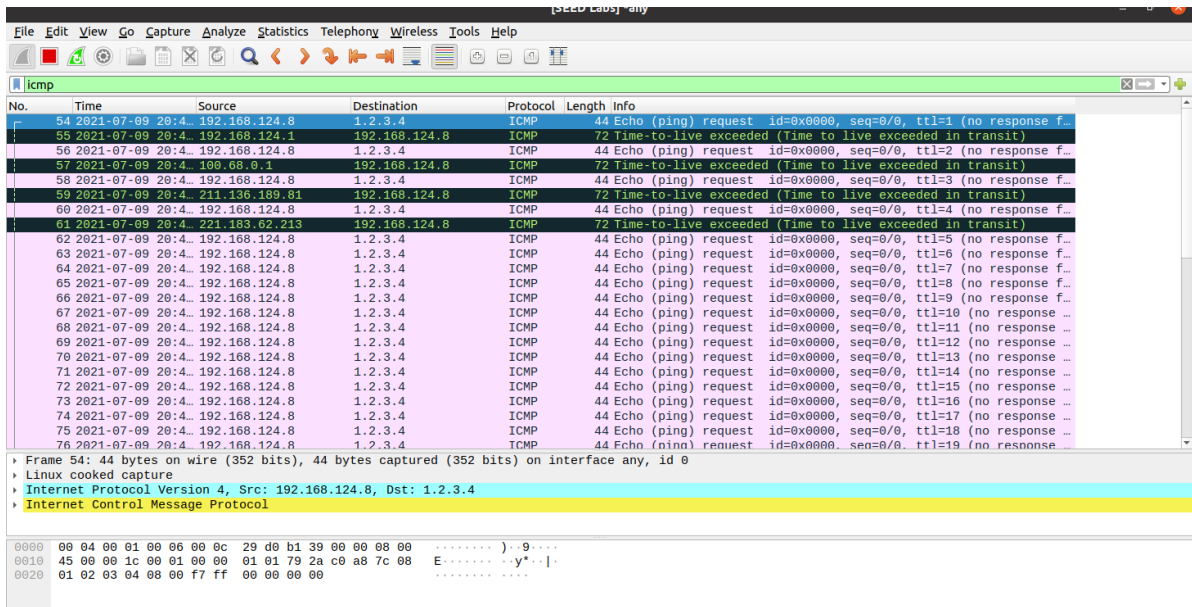
这个程序向目标IP发送 ICMP 数据包，一开始设置 TTL 值为 1，这个数据包将被第一个路由器丢弃，它将向我们发送一个ICMP错误消息，告诉我们生存时间已经超过了时间。这就是我们如何获得第一个路由器的IP地址。然后，不断增加 TTL 的值，直到使得数据包到达目的地。

用Wireshark观测数据包发送情况。

seed@VM: ~/Desktop

```
[07/09/21]seed@VM:~/Desktop$ touch trace.py
[07/09/21]seed@VM:~/Desktop$ sudo python3 trace.py
```

·  
Sent 1 packets.  
·  
Sent 1 packets.  
·  
Sent 1 packets.  
·  
Sent 1 packets.  
·  
Sent 1 packets.  
·  
Sent 1 packets.  
·  
Sent 1 packets.  
·  
Sent 1 packets.  
·  
Sent 1 packets.  
·  
Sent 1 packets.



No.	Time	Source	Destination	Protocol	Length	Info
54	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=1 (no response f...
55	2021-07-09 20:4...	192.168.124.1	192.168.124.8	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
56	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=2 (no response f...
57	2021-07-09 20:4...	100.68.0.1	192.168.124.8	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
58	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=3 (no response f...
59	2021-07-09 20:4...	211.136.189.81	192.168.124.8	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
60	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=4 (no response f...
61	2021-07-09 20:4...	221.183.62.213	192.168.124.8	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
62	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=5 (no response f...
63	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=6 (no response f...
64	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=7 (no response f...
65	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=8 (no response f...
66	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=9 (no response f...
67	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=10 (no response ...
68	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=11 (no response ...
69	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=12 (no response ...
70	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=13 (no response ...
71	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=14 (no response ...
72	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=15 (no response ...
73	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=16 (no response ...
74	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=17 (no response ...
75	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=18 (no response ...
76	2021-07-09 20:4...	192.168.124.8	1.2.3.4	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=19 (no response ...

Frame 54: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface any, id 0  
Linux cooked capture  
Internet Protocol Version 4, Src: 192.168.124.8, Dst: 1.2.3.4  
Internet Control Message Protocol

0000 00 04 00 01 00 06 00 0c 29 d0 b1 39 00 00 08 00 ..... )..9....  
0010 45 00 00 1c 00 01 00 00 01 01 79 2a c0 a8 7c 08 E..... .y\*...|  
0020 01 02 03 04 08 00 f7 ff 00 00 00 00 ..... ..

从Wireshark中我们可以观察到，途径的IP有192.168.124.1，100.68.0.1，211.136.189.81，221.183.62.213，最后到达1.2.3.4，即目的地。

## Task 1.4: Sniffing and-then Spoofing

结合嗅探和欺骗技术，编写 sniff-and-then-spoof.py：

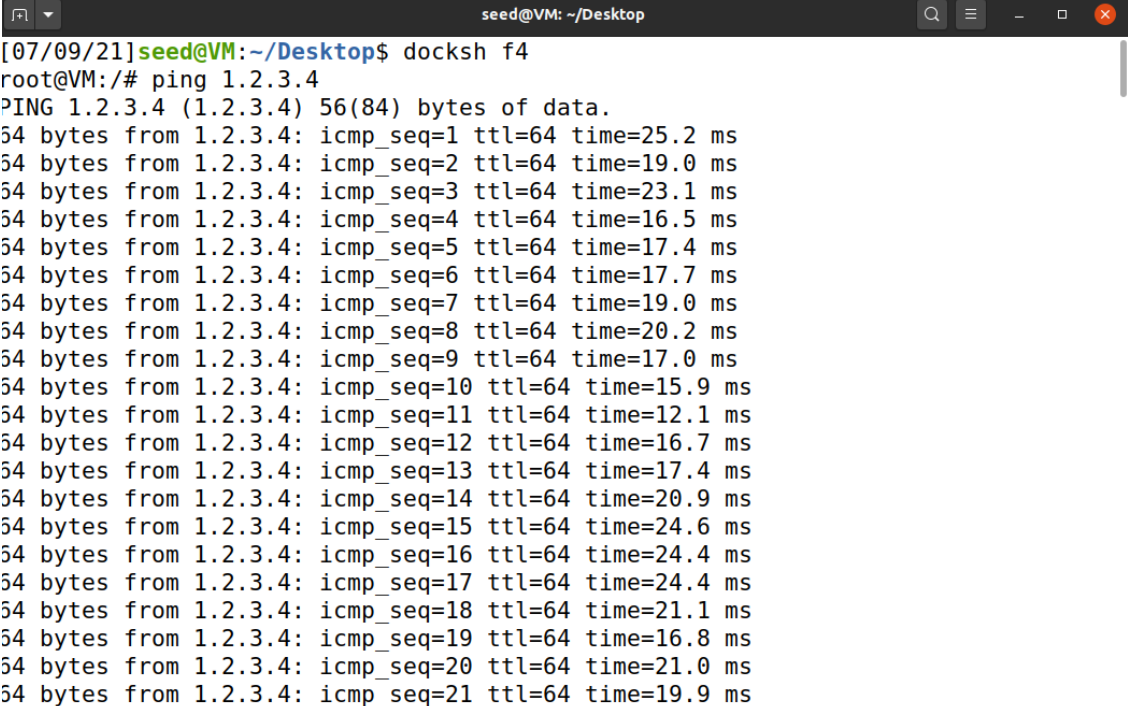


```
#!/usr/bin/evn python3
from scapy.all import *

def spoof_pkt(pkt):
    if ICMP in pkt and pkt[ICMP].type == 8:
        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
        send(newpkt)

pkt = sniff(filter='icmp', prn=spoof_pkt)
```

- ping 1.2.3.4 # a non-existing host on the Internet



```
[07/09/21]seed@VM:~/Desktop$ docksh f4
root@VM:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
54 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=25.2 ms
54 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=19.0 ms
54 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=23.1 ms
54 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=16.5 ms
54 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=17.4 ms
54 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=17.7 ms
54 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=19.0 ms
54 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=20.2 ms
54 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=17.0 ms
54 bytes from 1.2.3.4: icmp_seq=10 ttl=64 time=15.9 ms
54 bytes from 1.2.3.4: icmp_seq=11 ttl=64 time=12.1 ms
54 bytes from 1.2.3.4: icmp_seq=12 ttl=64 time=16.7 ms
54 bytes from 1.2.3.4: icmp_seq=13 ttl=64 time=17.4 ms
54 bytes from 1.2.3.4: icmp_seq=14 ttl=64 time=20.9 ms
54 bytes from 1.2.3.4: icmp_seq=15 ttl=64 time=24.6 ms
54 bytes from 1.2.3.4: icmp_seq=16 ttl=64 time=24.4 ms
54 bytes from 1.2.3.4: icmp_seq=17 ttl=64 time=24.4 ms
54 bytes from 1.2.3.4: icmp_seq=18 ttl=64 time=21.1 ms
54 bytes from 1.2.3.4: icmp_seq=19 ttl=64 time=16.8 ms
54 bytes from 1.2.3.4: icmp_seq=20 ttl=64 time=21.0 ms
54 bytes from 1.2.3.4: icmp_seq=21 ttl=64 time=19.9 ms
```

可以看到当我们ping一个网络上不存在的IP时，由于伪造报文，我们仍可以接收到响应。

- ping 10.9.0.99 # a non-existing host on the LAN

```
seed@VM: ~/Desktop
root@VM:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Host Unreachable
From 10.9.0.1 icmp_seq=2 Destination Host Unreachable
From 10.9.0.1 icmp_seq=3 Destination Host Unreachable
From 10.9.0.1 icmp_seq=4 Destination Host Unreachable
From 10.9.0.1 icmp_seq=5 Destination Host Unreachable
From 10.9.0.1 icmp_seq=6 Destination Host Unreachable
From 10.9.0.1 icmp_seq=7 Destination Host Unreachable
From 10.9.0.1 icmp_seq=8 Destination Host Unreachable
From 10.9.0.1 icmp_seq=9 Destination Host Unreachable
From 10.9.0.1 icmp_seq=10 Destination Host Unreachable
From 10.9.0.1 icmp_seq=11 Destination Host Unreachable
From 10.9.0.1 icmp_seq=12 Destination Host Unreachable
From 10.9.0.1 icmp_seq=13 Destination Host Unreachable
From 10.9.0.1 icmp_seq=14 Destination Host Unreachable
From 10.9.0.1 icmp_seq=15 Destination Host Unreachable
From 10.9.0.1 icmp_seq=16 Destination Host Unreachable
From 10.9.0.1 icmp_seq=17 Destination Host Unreachable
From 10.9.0.1 icmp_seq=18 Destination Host Unreachable
From 10.9.0.1 icmp_seq=19 Destination Host Unreachable
From 10.9.0.1 icmp_seq=20 Destination Host Unreachable
From 10.9.0.1 icmp_seq=21 Destination Host Unreachable
From 10.9.0.1 icmpn seq=22 Destination Host Unreachable
```

对于局域网内不存在的主机，先利用ARP进行MAC地址询问，由于一直得不到结果，所以没有ICMP报文，也就不存在报文欺骗。

- ping 8.8.8.8 # an existing host on the Internet

```
seed@VM: ~/Desktop
pipe 4
root@VM:/# 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=13.5 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=53 time=39.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=16.8 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=53 time=38.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=24.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=53 time=39.4 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=25.5 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=53 time=39.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=25.2 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=53 time=39.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=6 ttl=64 time=21.3 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=53 time=39.8 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=7 ttl=64 time=20.5 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=53 time=39.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=8 ttl=64 time=19.4 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=53 time=40.9 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=9 ttl=64 time=16.9 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=53 time=38.6 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=10 ttl=64 time=16.6 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=53 time=40.4 ms (DUP!)
64 bytes from 8.8.8.8: icmp seq=11 ttl=64 time=25.0 ms
```

对于网络上存在的主机，我们可以看到每个序列号的报文都存在一个重复报文，我们可以知道TTL=64，且时间较短的那个报文是伪造的报文。