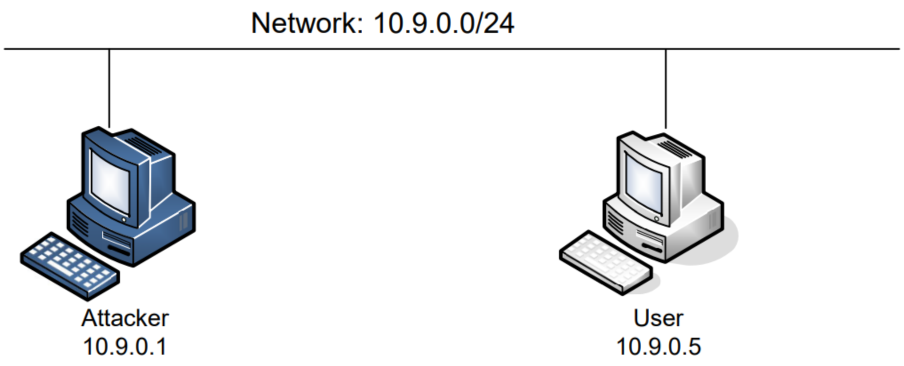
Lab1

陈盈 57118204

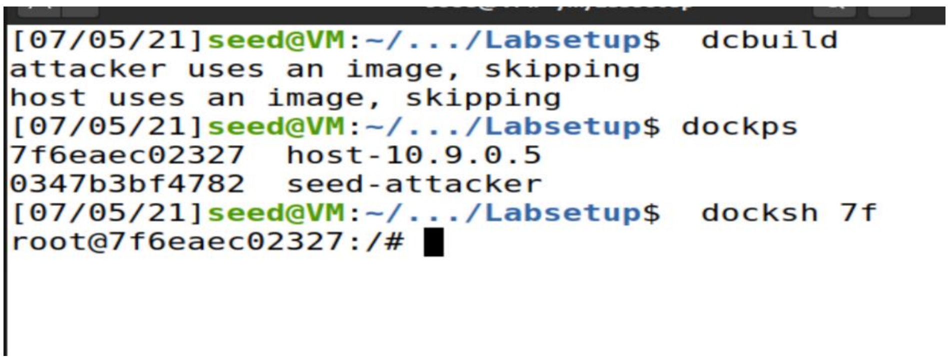
网络结构

我们可以使用两个 VM，下图描述了使用 VM 的实验室环境设置。我们会做所有的攻击在攻击者上，同时使用另一个 VM 作为用户。



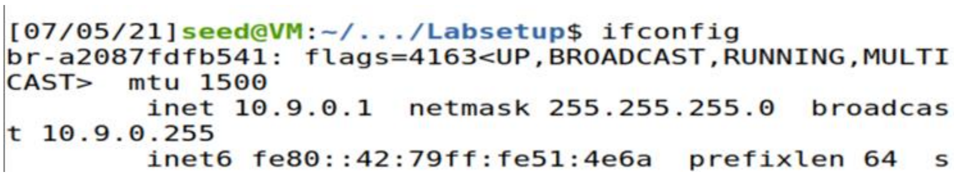
Container Setup and Commands

获取对应哈希值。



```
[07/05/21]seed@VM:~/.../Labsetup$  dcbuild
attacker uses an image, skipping
host uses an image, skipping
[07/05/21]seed@VM:~/.../Labsetup$ dockps
7f6eaec02327   host-10.9.0.5
0347b3bf4782   seed-attacker
[07/05/21]seed@VM:~/.../Labsetup$  docksh 7f
root@7f6eaec02327:/#
```

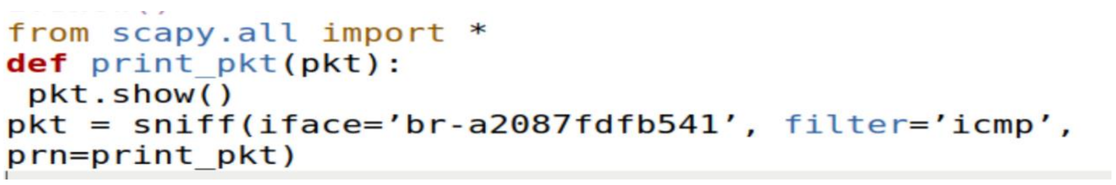
获取网卡信息。



```
[07/05/21]seed@VM:~/.../Labsetup$ ifconfig
br-a2087fdfb541: flags=4163<UP,BROADCAST,RUNNING,MULTI
CAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcas
t 10.9.0.255
        inet6 fe80::42:79ff:fe51:4e6a  prefixlen 64  s
```

Task 1.1: Sniffing Packets

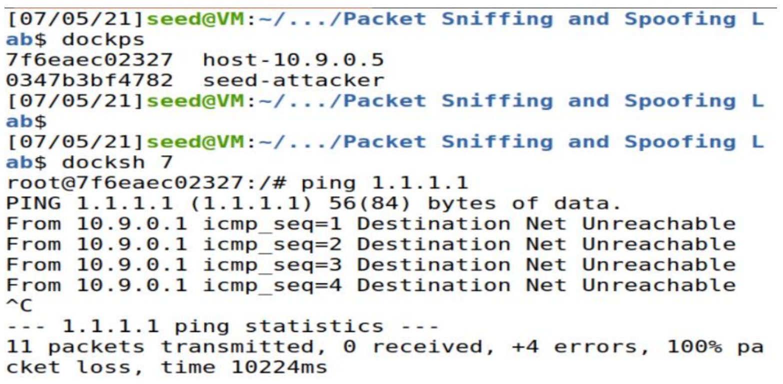
编写抓包程序 sniffer.py。



```
from scapy.all import *
def print_pkt(pkt):
 pkt.show()
pkt = sniff(iface='br-a2087fdfb541', filter='icmp',
prn=print_pkt)
```

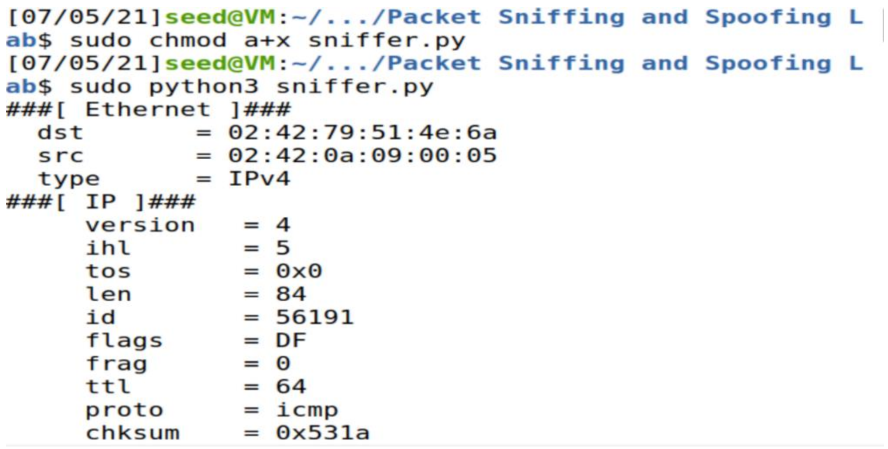
task 1.1.A

尝试向 1.1.1.1 发送报文。

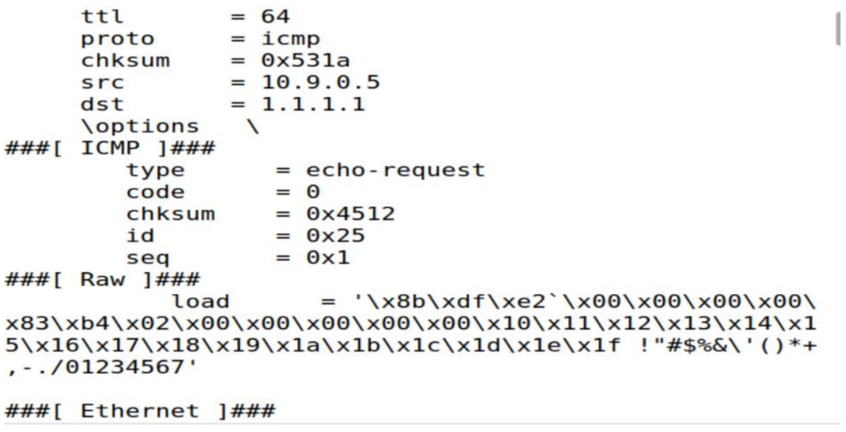


```
[07/05/21]seed@VM:~/.../Packet Sniffing and Spoofing L
ab$ dockps
7f6eaec02327  host-10.9.0.5
0347b3bf4782  seed-attacker
[07/05/21]seed@VM:~/.../Packet Sniffing and Spoofing L
ab$
[07/05/21]seed@VM:~/.../Packet Sniffing and Spoofing L
ab$ docksh 7
root@7f6eaec02327:/# ping 1.1.1.1
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Net Unreachable
From 10.9.0.1 icmp_seq=2 Destination Net Unreachable
From 10.9.0.1 icmp_seq=3 Destination Net Unreachable
From 10.9.0.1 icmp_seq=4 Destination Net Unreachable
^C
--- 1.1.1.1 ping statistics ---
11 packets transmitted, 0 received, +4 errors, 100% pa
cket loss, time 10224ms
```

Attacker 中，使用 root 执行#sniffer.py，抓取的报文如下

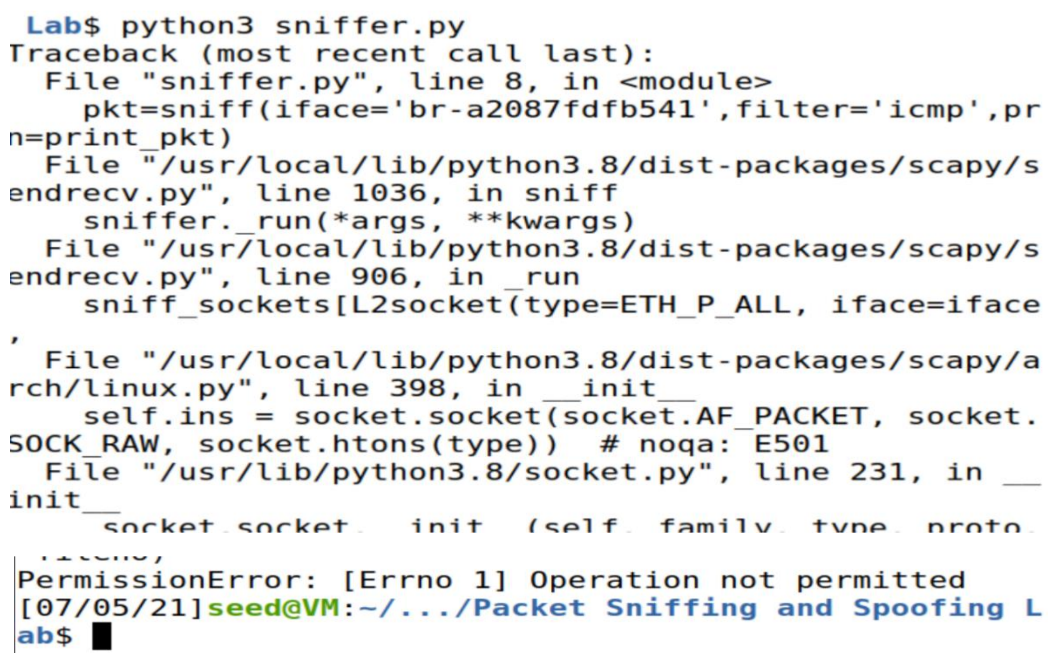


```
[07/05/21]seed@VM:~/.../Packet Sniffing and Spoofing L
ab$ sudo chmod a+x sniffer.py
[07/05/21]seed@VM:~/.../Packet Sniffing and Spoofing L
ab$ sudo python3 sniffer.py
###[ Ethernet ]###
  dst       = 02:42:79:51:4e:6a
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 56191
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x531a
```



```
     ttl       = 64
     proto     = icmp
     chksum    = 0x531a
     src       = 10.9.0.5
     dst       = 1.1.1.1
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x4512
        id        = 0x25
        seq       = 0x1
###[ Raw ]###
           load      = '\x8b\xdf\xe2`\x00\x00\x00\x00\
x83\xb4\x02\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x1
5\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+
,-./01234567'

###[ Ethernet ]###
```

非 root 权限下，运行 sniffer 抓包无效。

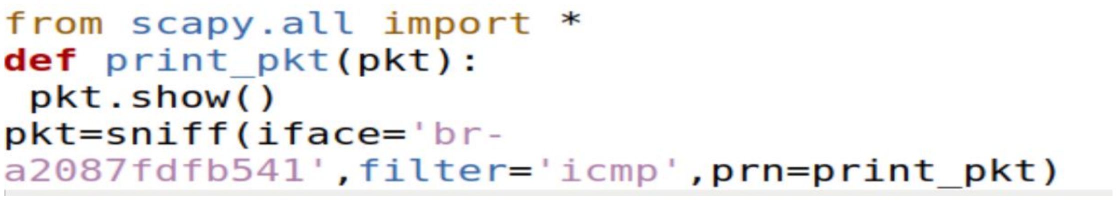


```
Lab$ python3 sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 8, in <module>
    pkt=sniff(iface='br-a2087fdfb541',filter='icmp',pr
n=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/s
endrecv.py", line 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/s
endrecv.py", line 906, in _run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface
,
  File "/usr/local/lib/python3.8/dist-packages/scapy/a
rch/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,
PermissionError: [Errno 1] Operation not permitted
[07/05/21]seed@VM:~/.../Packet Sniffing and Spoofing L
ab$ 
```

task 1.1.B

• Capture only the ICMP packet

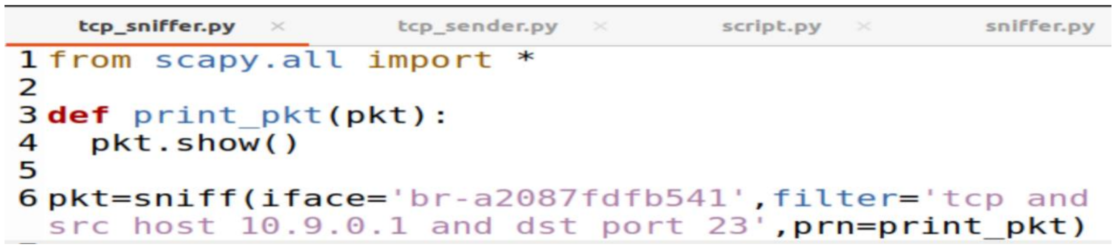
仍使用 sniffer.py 进行抓包，结果与 task1.1.A 相同。



```
from scapy.all import *
def print_pkt(pkt):
 pkt.show()
pkt=sniff(iface='br-
a2087fdfb541',filter='icmp',prn=print_pkt)
```

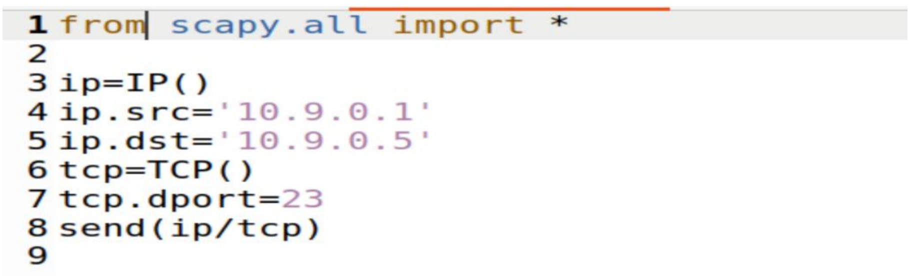
• Capture any TCP packet that comes from a particular IP and with a destination port number 23.

tcp_sniffer.py 用于抓取满足要求的 TCP 报文，源地址为 10.9.0.1，目的端口为 23。



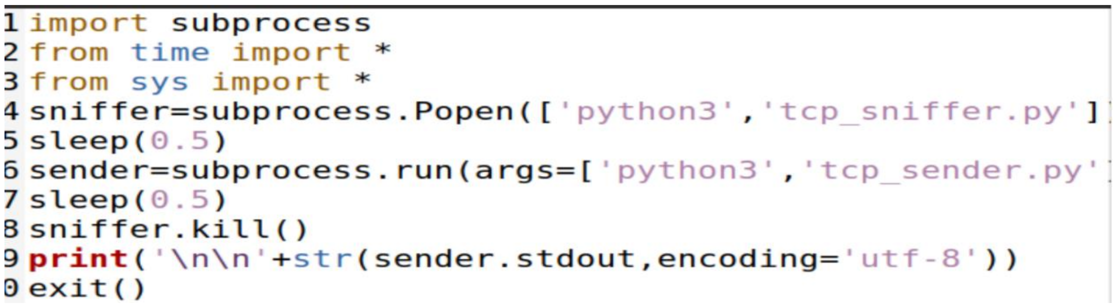
```
from scapy.all import *

def print_pkt(pkt):
  pkt.show()

pkt=sniff(iface='br-a2087fdfb541',filter='tcp and
 src host 10.9.0.1 and dst port 23',prn=print_pkt)
```

tcp_sender.py 用于发送一条满足要求的 TCP 报文。



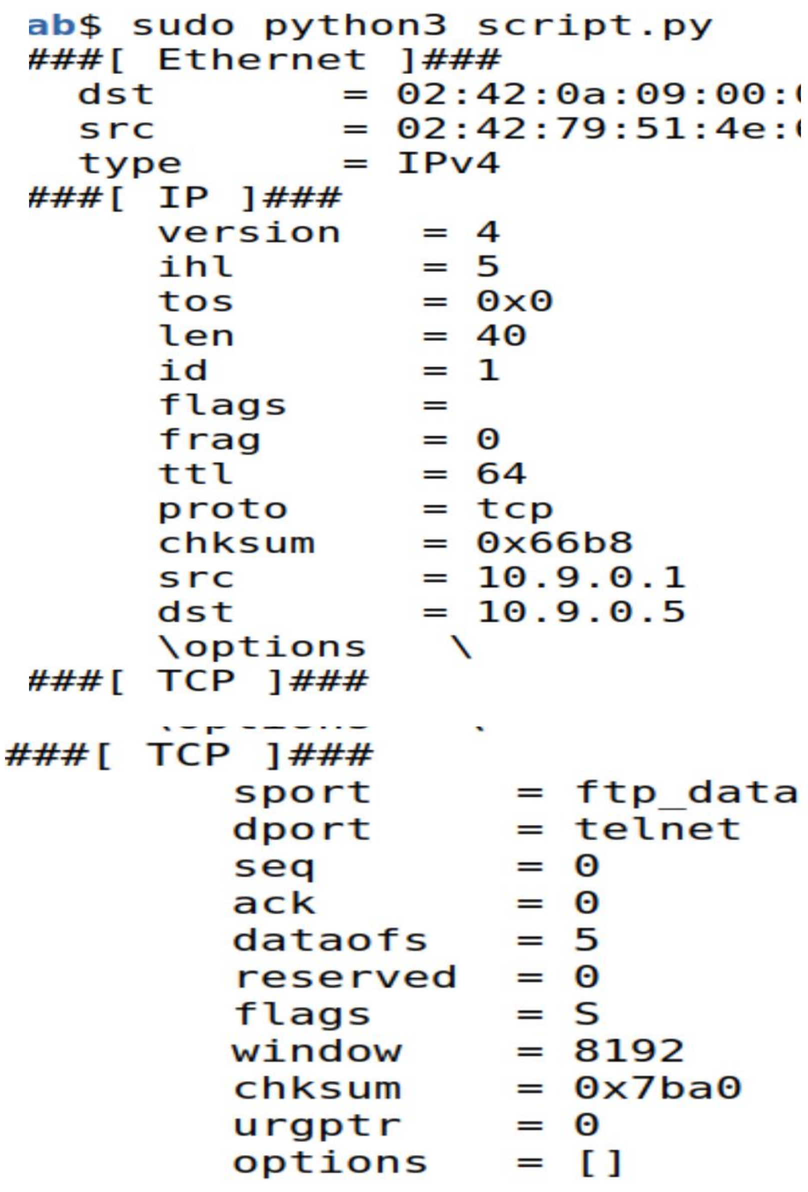
```
from scapy.all import *

ip=IP()
ip.src='10.9.0.1'
ip.dst='10.9.0.5'
tcp=TCP()
tcp.dport=23
send(ip/tcp)
```

script.py



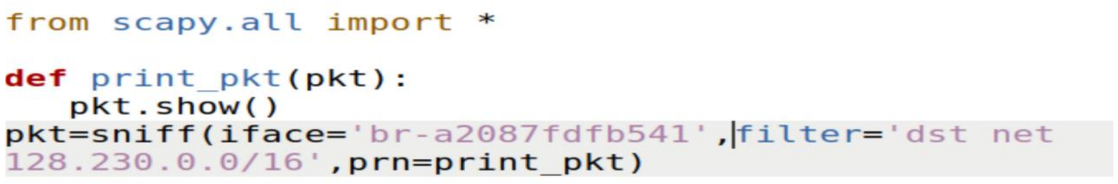
```
import subprocess
from time import *
from sys import *
sniffer=subprocess.Popen(['python3','tcp_sniffer.py']
sleep(0.5)
sender=subprocess.run(args=['python3','tcp_sender.py'
sleep(0.5)
sniffer.kill()
print('\n\n'+str(sender.stdout,encoding='utf-8'))
exit()
```

运行 script.py 进行 TCP 报文的发送和抓取。



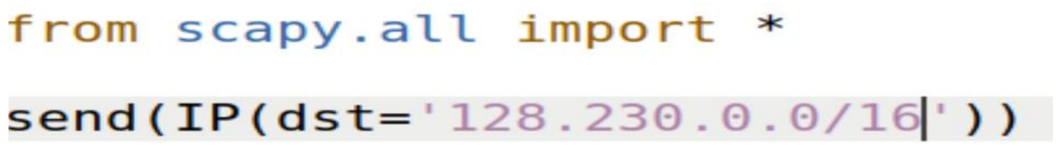
```
ab$ sudo python3 script.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:
  src       = 02:42:79:51:4e:
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 40
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x66b8
     src       = 10.9.0.1
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
###[ TCP ]###
        sport     = ftp_data
        dport     = telnet
        seq       = 0
        ack       = 0
        dataofs   = 5
        reserved  = 0
        flags     = S
        window    = 8192
        chksum    = 0x7ba0
        urgptr    = 0
        options   = []
```

• 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.
sniffer.py，用于抓取发往 128.230.0.0/16 子网的报文。



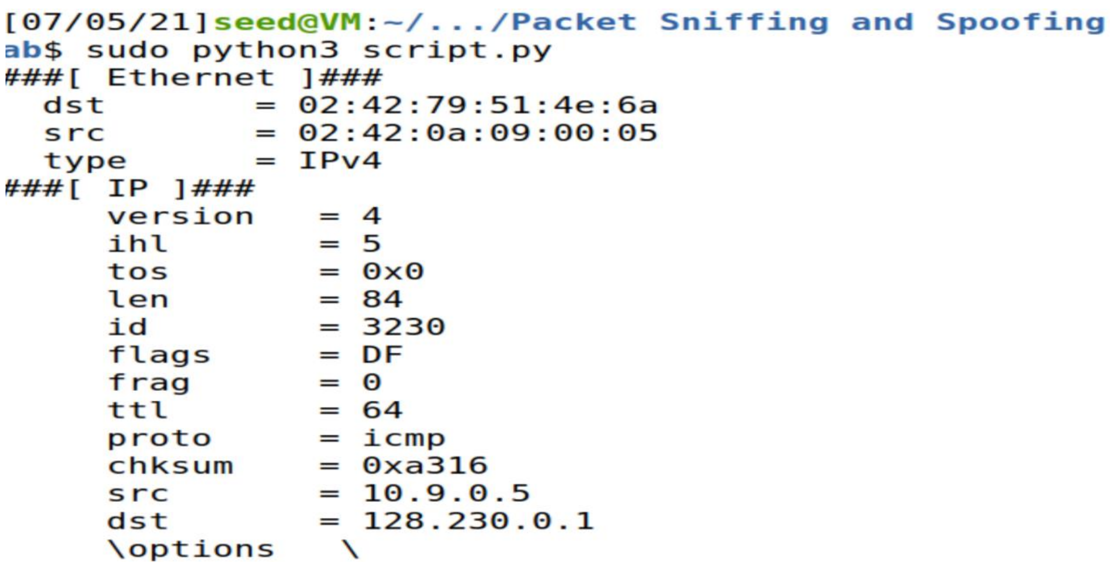
```
from scapy.all import *

def print_pkt(pkt):
   pkt.show()
pkt=sniff(iface='br-a2087fdfb541',filter='dst net
128.230.0.0/16',prn=print_pkt)
```

sender.py，用于发送报文给 128.230.0.0/16 子网。



```
from scapy.all import *

send(IP(dst='128.230.0.0/16'))
```

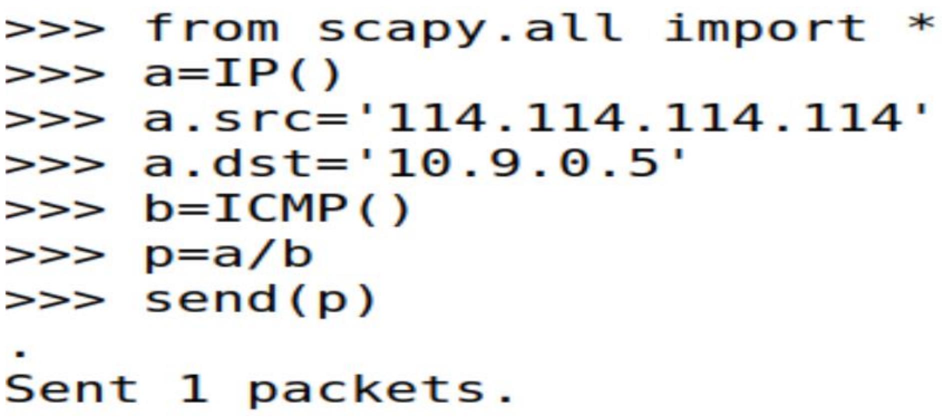
运行 script.py 进行发往子网 128.230.0.0/16 的报文的抓取。



```
[07/05/21]seed@VM:~/.../Packet Sniffing and Spoofing
ab$ sudo python3 script.py
###[ Ethernet ]###
  dst       = 02:42:79:51:4e:6a
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 3230
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xa316
     src       = 10.9.0.5
     dst       = 128.230.0.1
     \options   \
```

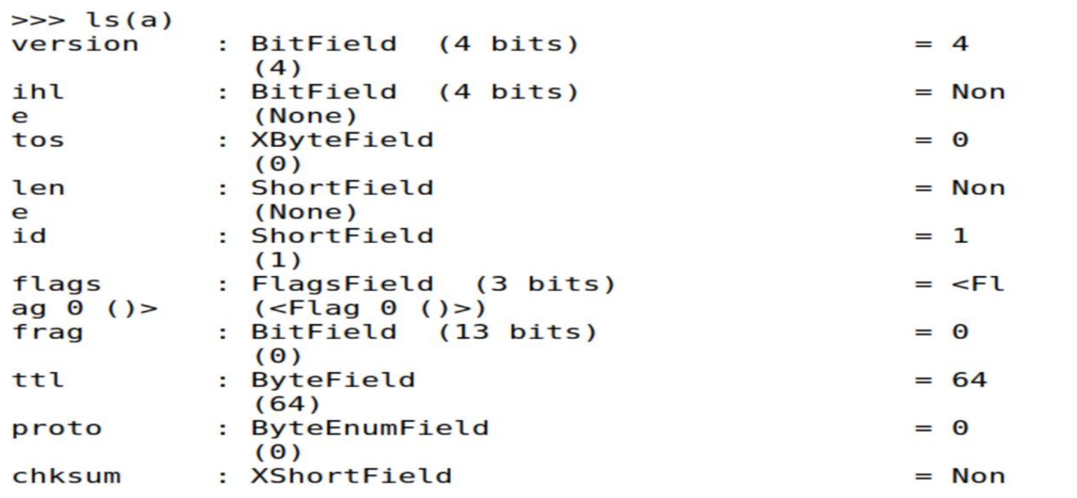
Task 1.2: Spoofing ICMP Packets

如下程序实现构造一个 ICMP echo-request 包，可以指定任意 IP 地址，本次实验指定 114.114.114.114 为源地址（伪造），10.1.0.5 为目的地址。



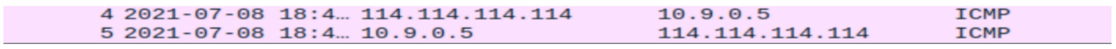
```
>>> from scapy.all import *
>>> a=IP()
>>> a.src='114.114.114.114'
>>> a.dst='10.9.0.5'
>>> b=ICMP()
>>> p=a/b
>>> send(p)
.
Sent 1 packets.
```

查看发送的 IP 报文。



```
>>> ls(a)
version      : BitField  (4 bits)                    = 4
               (4)
ihl          : BitField  (4 bits)                    = Non
e              (None)
tos          : XByteField                            = 0
               (0)
len          : ShortField                            = Non
e              (None)
id           : ShortField                            = 1
               (1)
flags        : FlagsField  (3 bits)                  = <Fl
ag 0 ()>       (<Flag 0 ()>)
frag         : BitField  (13 bits)                   = 0
               (0)
ttl          : ByteField                             = 64
               (64)
proto        : ByteEnumField                         = 0
               (0)
chksum       : XShortField                           = Non
```

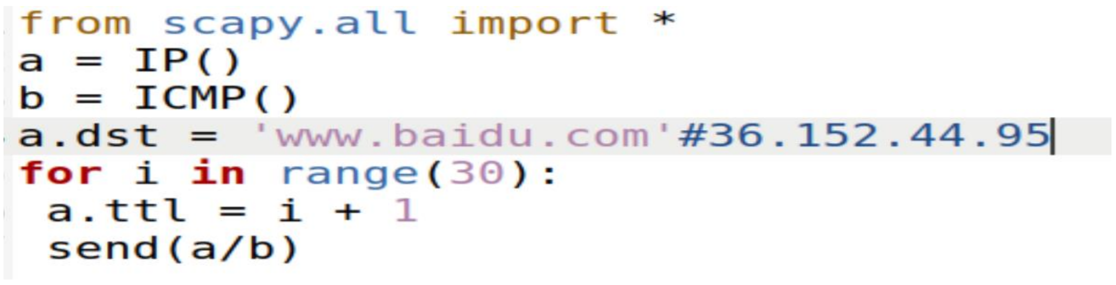
利用 wireshark 进行抓包，可以发现受害者主机 10.9.0.5 尝试回复伪冒的源地址。



```
     4 2021-07-08 18:4… 114.114.114.114       10.9.0.5              ICMP
     5 2021-07-08 18:4… 10.9.0.5              114.114.114.114       ICMP
```

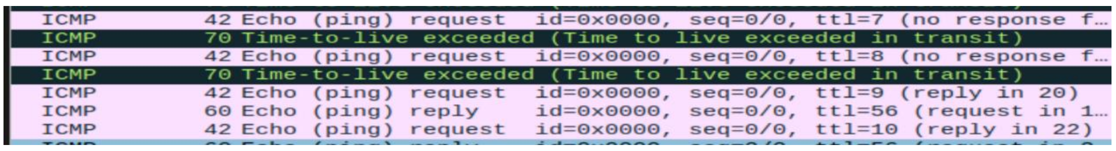
Task 1.3

使用桥接网络，运行 testTTL.py，尝试用不同的跳数将报文发送给 www.baidu.com。



```python
from scapy.all import *
a = IP()
b = ICMP()
a.dst = 'www.baidu.com'#36.152.44.95
for i in range(30):
  a.ttl = i + 1
  send(a/b)
```

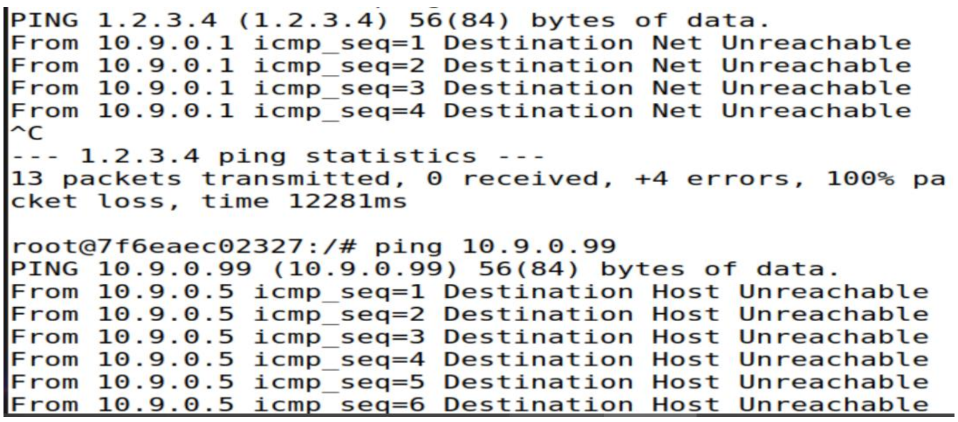
从 wireshark 看，最早在 TTL=9 时开始有来自百度的响应，所以 VM 和百度服务器之间的 Loop 为 9。

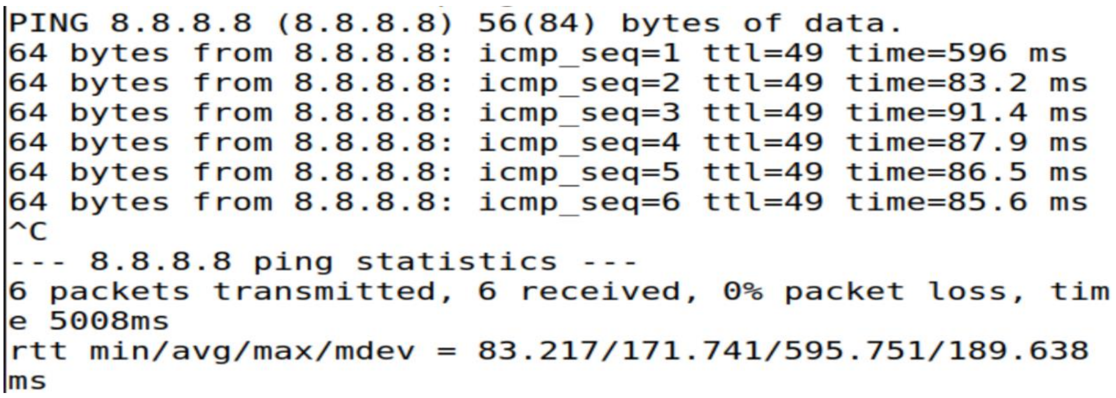


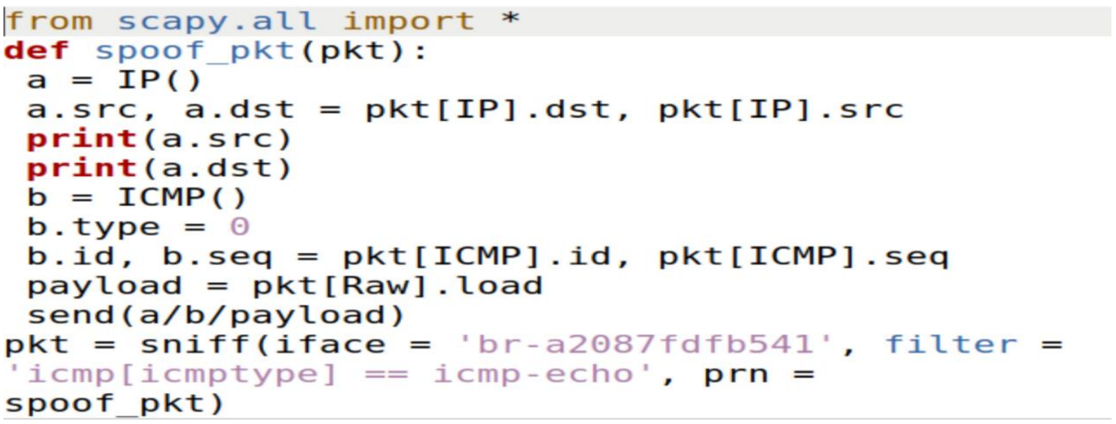
```
ICMP      42 Echo (ping) request  id=0x0000, seq=0/0, ttl=7 (no response f…
ICMP      70 Time-to-live exceeded (Time to live exceeded in transit)
ICMP      42 Echo (ping) request  id=0x0000, seq=0/0, ttl=8 (no response f…
ICMP      70 Time-to-live exceeded (Time to live exceeded in transit)
ICMP      42 Echo (ping) request  id=0x0000, seq=0/0, ttl=9 (reply in 20)
ICMP      60 Echo (ping) reply    id=0x0000, seq=0/0, ttl=56 (request in 1…
ICMP      42 Echo (ping) request  id=0x0000, seq=0/0, ttl=10 (reply in 22)
```

Task 1.4

直接 ping1.2.3.4，10.9.0.99,发现均无法 ping 通。这是因为 1.2.3.4 和 10.9.0.99 不存在。

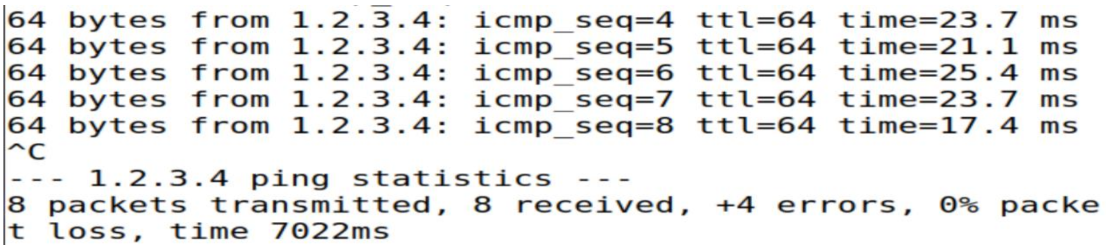

```
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Net Unreachable
From 10.9.0.1 icmp_seq=2 Destination Net Unreachable
From 10.9.0.1 icmp_seq=3 Destination Net Unreachable
From 10.9.0.1 icmp_seq=4 Destination Net Unreachable
^C
--- 1.2.3.4 ping statistics ---
13 packets transmitted, 0 received, +4 errors, 100% pa
cket loss, time 12281ms

root@7f6eaec02327:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
From 10.9.0.5 icmp_seq=4 Destination Host Unreachable
From 10.9.0.5 icmp_seq=5 Destination Host Unreachable
From 10.9.0.5 icmp_seq=6 Destination Host Unreachable
```
ping8.8.8.8 发现可以 ping 通，因为该主机存在于互联网上。

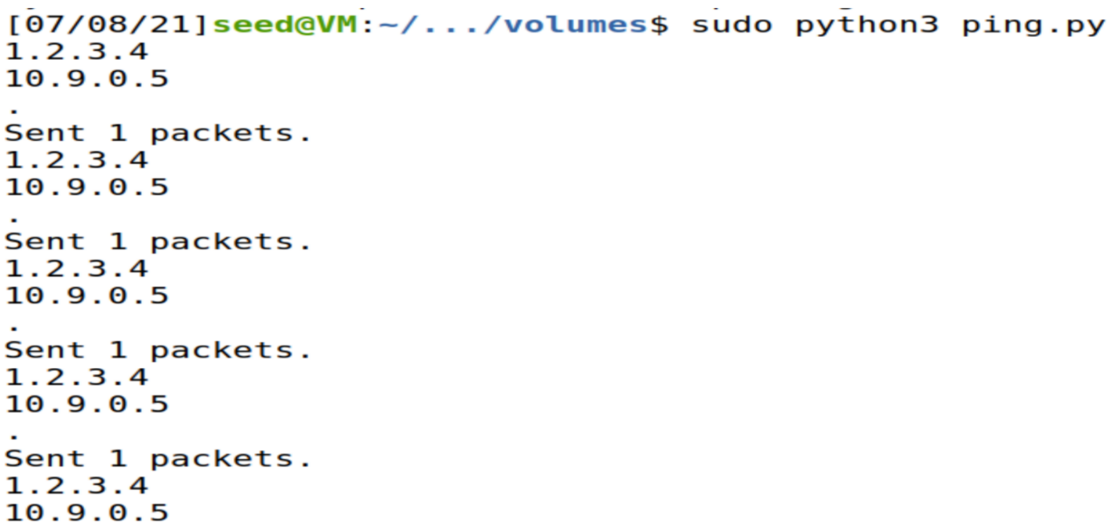

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=49 time=596 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=49 time=83.2 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=49 time=91.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=49 time=87.9 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=49 time=86.5 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=49 time=85.6 ms
^C
--- 8.8.8.8 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, tim
e 5008ms
rtt min/avg/max/mdev = 83.217/171.741/595.751/189.638
ms
```
ping.py 实现伪造不存在主机的 ICMP echo-reply 报文。


```
from scapy.all import *
def spoof_pkt(pkt):
 a = IP()
 a.src, a.dst = pkt[IP].dst, pkt[IP].src
 print(a.src)
 print(a.dst)
 b = ICMP()
 b.type = 0
 b.id, b.seq = pkt[ICMP].id, pkt[ICMP].seq
 payload = pkt[Raw].load
 send(a/b/payload)
pkt = sniff(iface = 'br-a2087fdfb541', filter =
'icmp[icmptype] == icmp-echo', prn =
spoof_pkt)
```

测试 1.2.3.4（不存在主机）

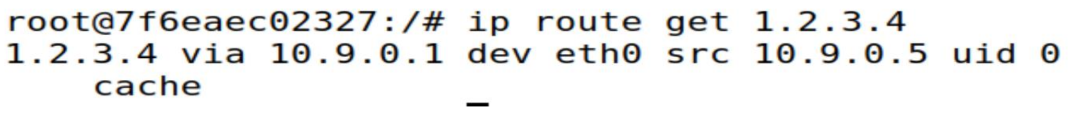
运行后，ping1.2.3.4 可以 ping 通，这是因为首先 user 查看该 ip 是否在局域网内，发现不在局域网内，然后将报文发送给网关，实现路由。


```
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=23.7 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=21.1 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=25.4 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=23.7 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=17.4 ms
^C
--- 1.2.3.4 ping statistics ---
8 packets transmitted, 8 received, +4 errors, 0% packe
t loss, time 7022ms
```



```
[07/08/21]seed@VM:~/.../volumes$ sudo python3 ping.py
1.2.3.4
10.9.0.5
.
Sent 1 packets.
1.2.3.4
10.9.0.5
.
Sent 1 packets.
1.2.3.4
10.9.0.5
.
Sent 1 packets.
1.2.3.4
10.9.0.5
.
Sent 1 packets.
1.2.3.4
10.9.0.5
```

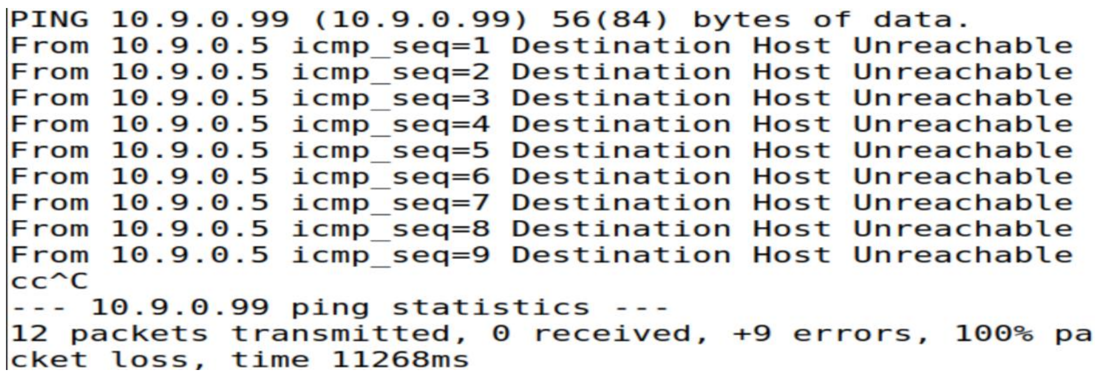
通过查看路由，发现 user 将 10.9.0.1 当做网关，所以将该报文传递给 10.9.0.1，而报文到了 10.9.0.1 后，因为实际上 1.2.3.4.不可达，所以只有 ping.py 会发回伪造报文给 10.9.0.5，造成 1.2.3.4 可以 ping 通的假象。



```
root@7f6eaec02327:/# ip route get 1.2.3.4
1.2.3.4 via 10.9.0.1 dev eth0 src 10.9.0.5 uid 0
    cache
```

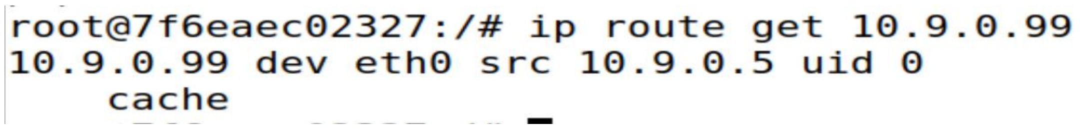
测试 10.9.0.99（本局域网内不存在）

ping10.9.0.99 仍然 ping 不通。



```
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
From 10.9.0.5 icmp_seq=4 Destination Host Unreachable
From 10.9.0.5 icmp_seq=5 Destination Host Unreachable
From 10.9.0.5 icmp_seq=6 Destination Host Unreachable
From 10.9.0.5 icmp_seq=7 Destination Host Unreachable
From 10.9.0.5 icmp_seq=8 Destination Host Unreachable
From 10.9.0.5 icmp_seq=9 Destination Host Unreachable
cc^C
--- 10.9.0.99 ping statistics ---
12 packets transmitted, 0 received, +9 errors, 100% pa
cket loss, time 11268ms
```

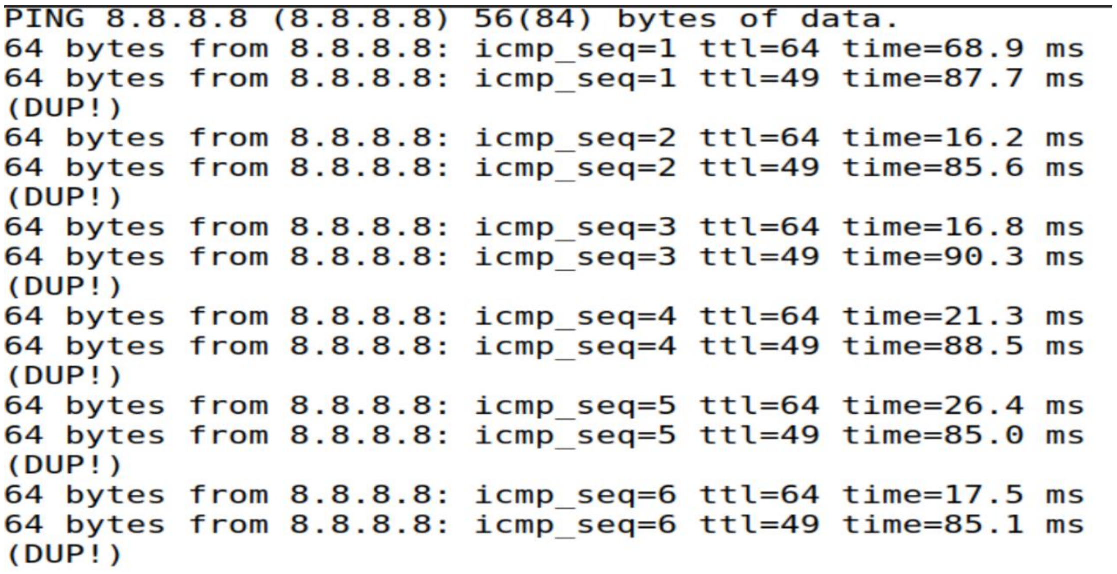
由于该地址在 10.9.0.0/24 局域网内，arp 广播后没有响应，说明该主机不存在。通过查看路由可以发现，由于目的地址同属于一个局域网，所以主机并没有把报文交给网关，ping.py 无法伪造报文发给 10.9.0.5。



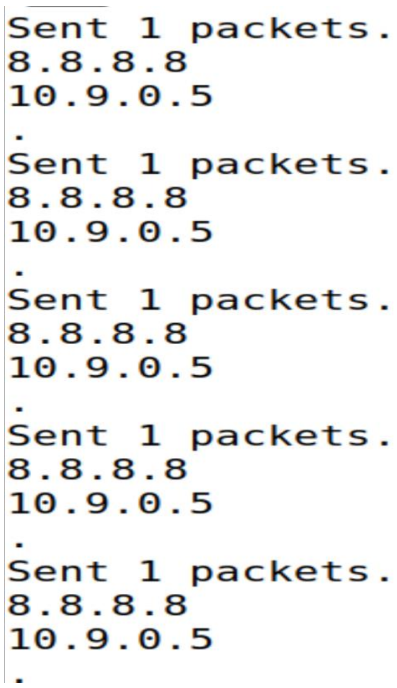
```
root@7f6eaec02327:/# ip route get 10.9.0.99
10.9.0.99 dev eth0 src 10.9.0.5 uid 0
    cache
```

测试 8.8.8.8（互联网上存在）

ping 8.8.8.8 可以 ping 通，可以发现收到 ICMP 响应报文重复，attacker 的返回和 8.8.8.8 的返回重复。

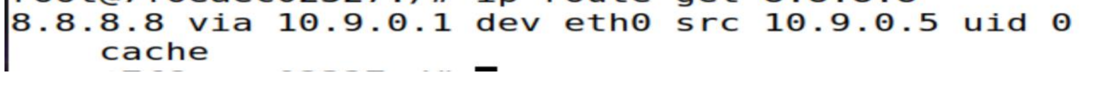


```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=68.9 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=49 time=87.7 ms
(DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=16.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=49 time=85.6 ms
(DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=16.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=49 time=90.3 ms
(DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=21.3 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=49 time=88.5 ms
(DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=26.4 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=49 time=85.0 ms
(DUP!)
64 bytes from 8.8.8.8: icmp_seq=6 ttl=64 time=17.5 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=49 time=85.1 ms
(DUP!)
```



```
Sent 1 packets.
8.8.8.8
10.9.0.5
.
Sent 1 packets.
8.8.8.8
10.9.0.5
.
Sent 1 packets.
8.8.8.8
10.9.0.5
.
Sent 1 packets.
8.8.8.8
10.9.0.5
.
Sent 1 packets.
8.8.8.8
10.9.0.5
.
```

通过查看路由，user 将报文发送给网关后，ping 命令的报文会被 10.9.0.1 发送到互联网上，经由互联网交给 8.8.8.8，由于 8.8.8.8 真实存在，所以最后会有两个 echo-reply，一个来自 8.8.8.8，另外一个由 piny.py 伪造。



```
root@7f6eaec02327:/# ip route get 8.8.8.8
8.8.8.8 via 10.9.0.1 dev eth0 src 10.9.0.5 uid 0
    cache
```