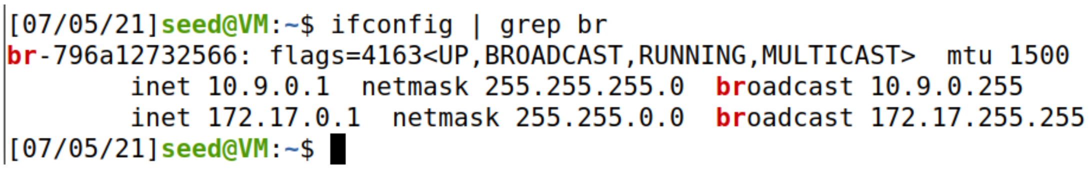
TASK 1.1

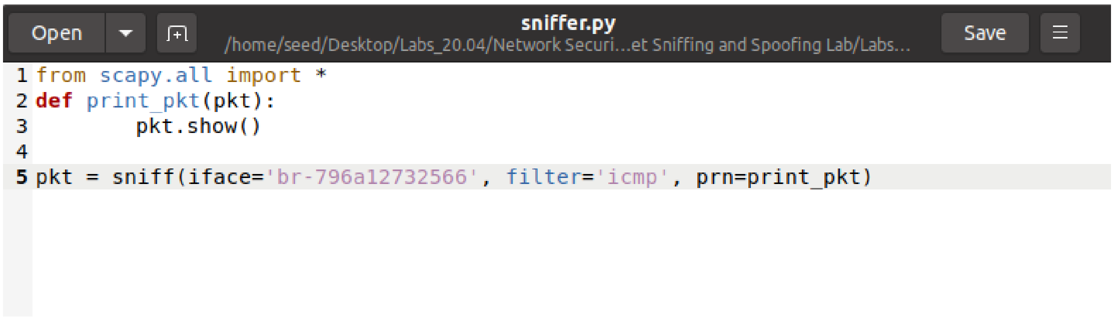
1.1A

用 ifconfig 查询自己的 iface，为 br-796a12732566



```
[07/05/21]seed@VM:~$ ifconfig | grep br
br-796a12732566: 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
[07/05/21]seed@VM:~$ 
```

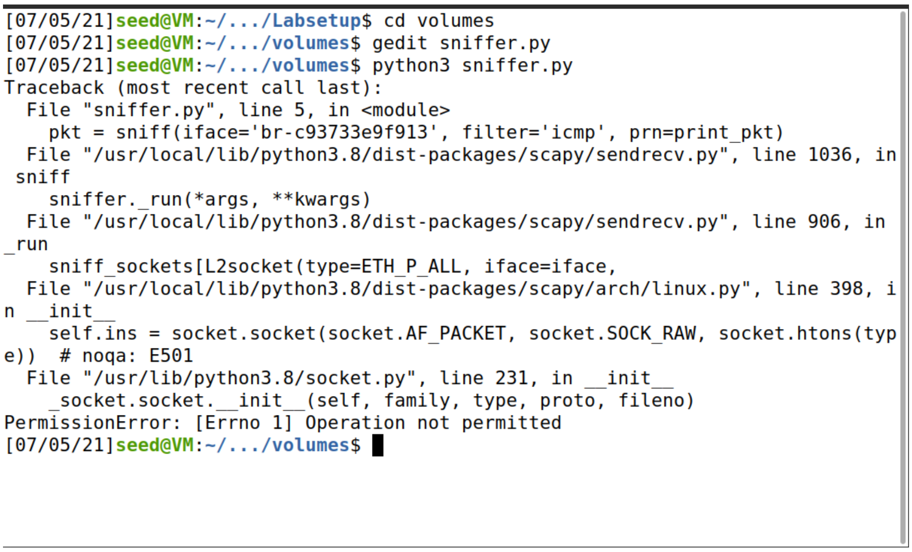
在 volumes 目录下新建 sniffer.py，写入下图：



Open
sniffer.py
/home/seed/Desktop/Labs_20.04/Network Securi...et Sniffing and Spoofing Lab/Labs...
Save

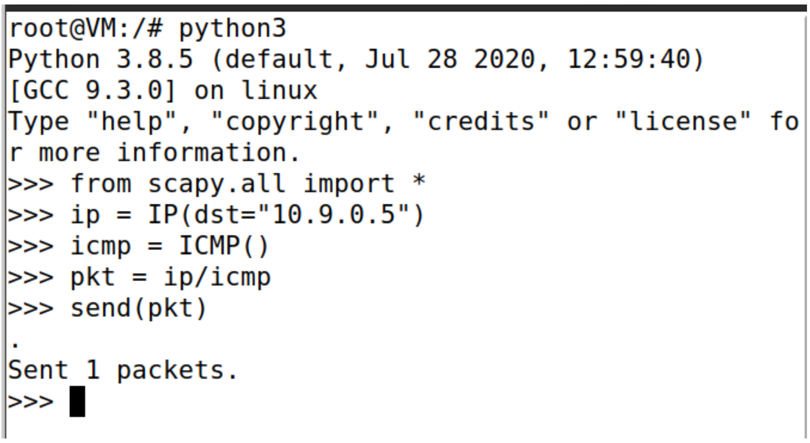
```
from scapy.all import *
def print_pkt(pkt):
        pkt.show()

pkt = sniff(iface='br-796a12732566', filter='icmp', prn=print_pkt)
```

首先在用户态下运行发现运行失败，因为没有相应权限。如图。



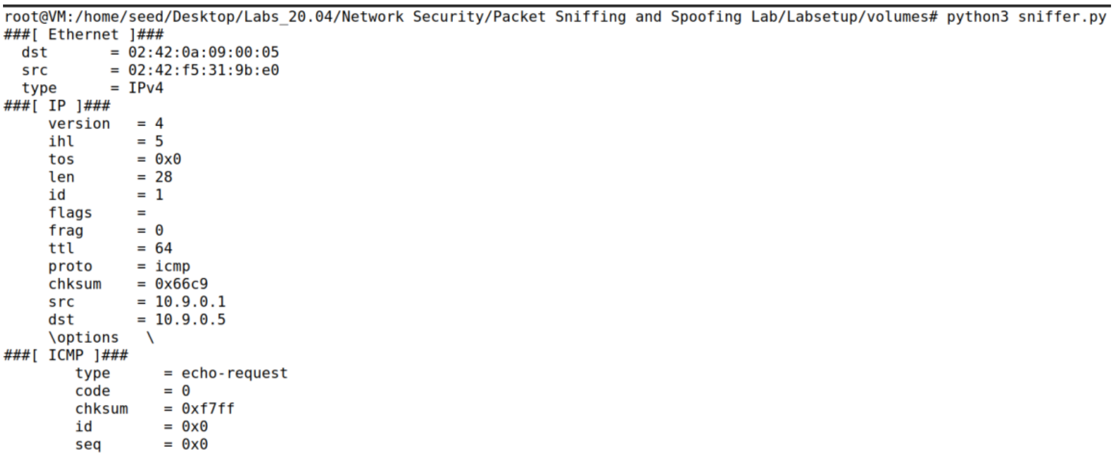
```
[07/05/21]seed@VM:~/.../Labsetup$ cd volumes
[07/05/21]seed@VM:~/.../volumes$ gedit sniffer.py
[07/05/21]seed@VM:~/.../volumes$ python3 sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 5, in <module>
    pkt = sniff(iface='br-c93733e9f913', filter='icmp', prn=print_pkt)
  File "/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, i
n __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(typ
e))  # 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
[07/05/21]seed@VM:~/.../volumes$ 
```

进入 root 后再次运行 sniffer.py，接着在 docker 上构造并发送如下报文：



```
root@VM:/# python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" fo
r more information.
>>> from scapy.all import *
>>> ip = IP(dst="10.9.0.5")
>>> icmp = ICMP()
>>> pkt = ip/icmp
>>> send(pkt)
.
Sent 1 packets.
>>> 
```

sniffer.py 成功捕获如下信息：

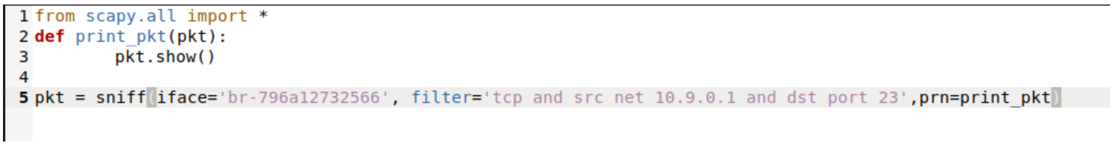


```
root@VM:/home/seed/Desktop/Labs_20.04/Network Security/Packet Sniffing and Spoofing Lab/Labsetup/volumes# python3 sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:f5:31:9b:e0
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 28
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x66c9
     src       = 10.9.0.1
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xf7ff
        id        = 0x0
        seq       = 0x0
```

1.1B

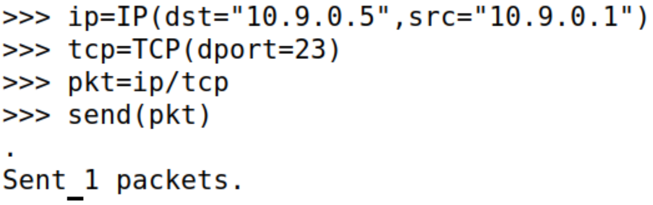
捕获 ICMP 报文，sniffer.py 代码和捕获结果同 1.1A。

捕获特定源地址和目的端口号为 23 的 TCP 报文时，更改 sniffer 的 filter，如下：



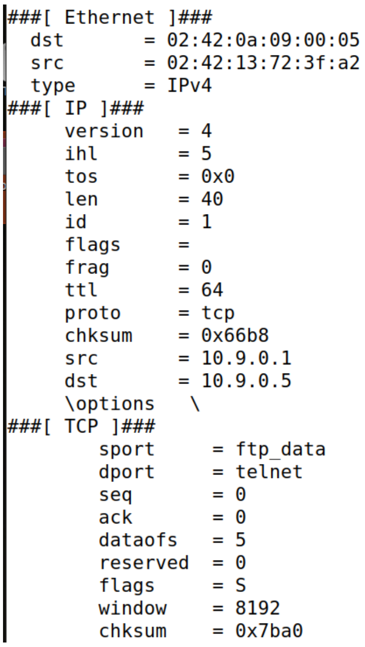
```python
from scapy.all import *
def print_pkt(pkt):
        pkt.show()

pkt = sniff(iface='br-796a12732566', filter='tcp and src net 10.9.0.1 and dst port 23',prn=print_pkt)
```

运行 sniffer.py，在 docker 中重新构造并发送报文，如下：



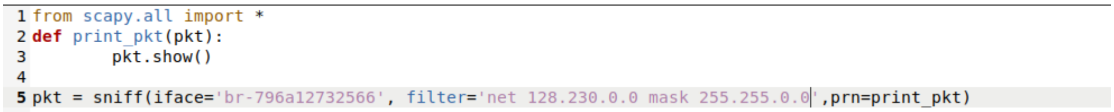
```
>>> ip=IP(dst="10.9.0.5",src="10.9.0.1")
>>> tcp=TCP(dport=23)
>>> pkt=ip/tcp
>>> send(pkt)
.
Sent 1 packets.
```

sniffer.py 捕获到的结果如下，其中 dport 端口为 telnet，默认为 23。



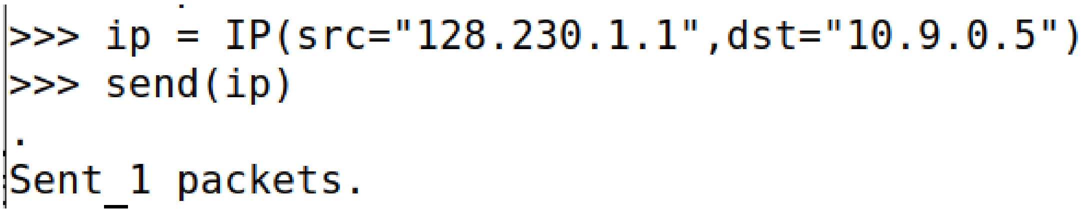
```
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:13:72:3f:a2
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 40
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x66b8
     src       = 10.9.0.1
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
        sport     = ftp_data
        dport     = telnet
        seq       = 0
        ack       = 0
        dataofs   = 5
        reserved  = 0
        flags     = S
        window    = 8192
        chksum    = 0x7ba0
```

捕获来自任意子网或去往任意子网的报文，重复上述步骤，其中 sniffer.py 改为：



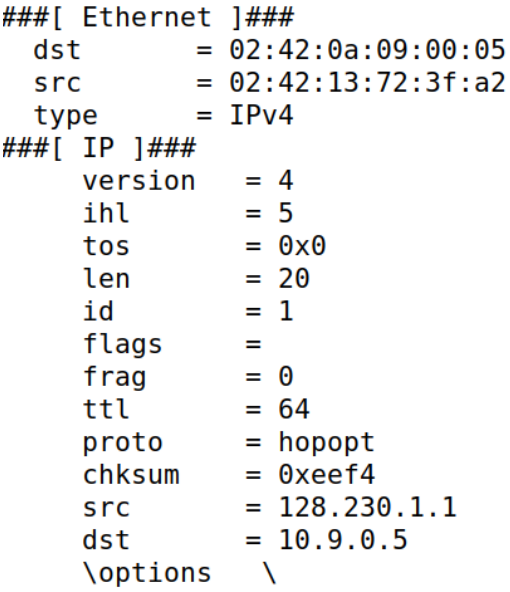
```
from scapy.all import *
def print_pkt(pkt):
        pkt.show()

pkt = sniff(iface='br-796a12732566', filter='net 128.230.0.0 mask 255.255.0.0',prn=print_pkt)
```

构造的报文为：

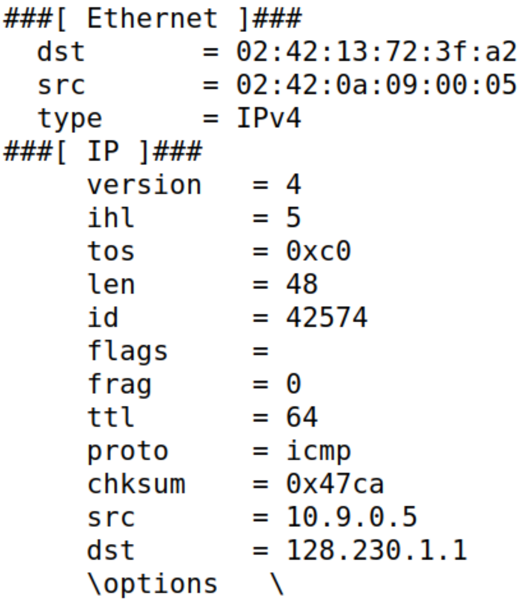


```
>>> ip = IP(src="128.230.1.1",dst="10.9.0.5")
>>> send(ip)
.
Sent 1 packets.
```

捕获的结果为：



```
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:13:72:3f:a2
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 20
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = hopopt
     chksum    = 0xeef4
     src       = 128.230.1.1
     dst       = 10.9.0.5
     \options   \
```

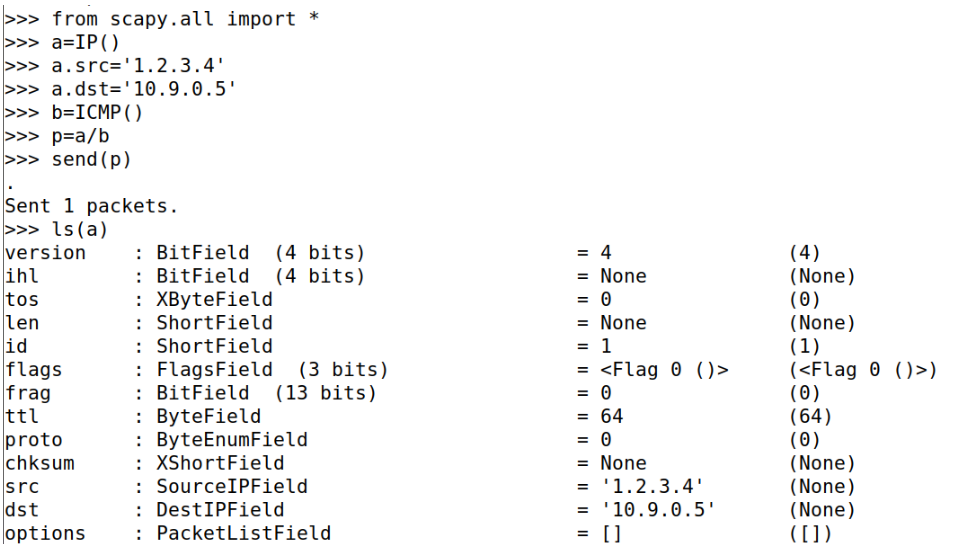


```
###[ Ethernet ]###
  dst       = 02:42:13:72:3f:a2
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0xc0
     len       = 48
     id        = 42574
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x47ca
     src       = 10.9.0.5
     dst       = 128.230.1.1
     \options   \
```

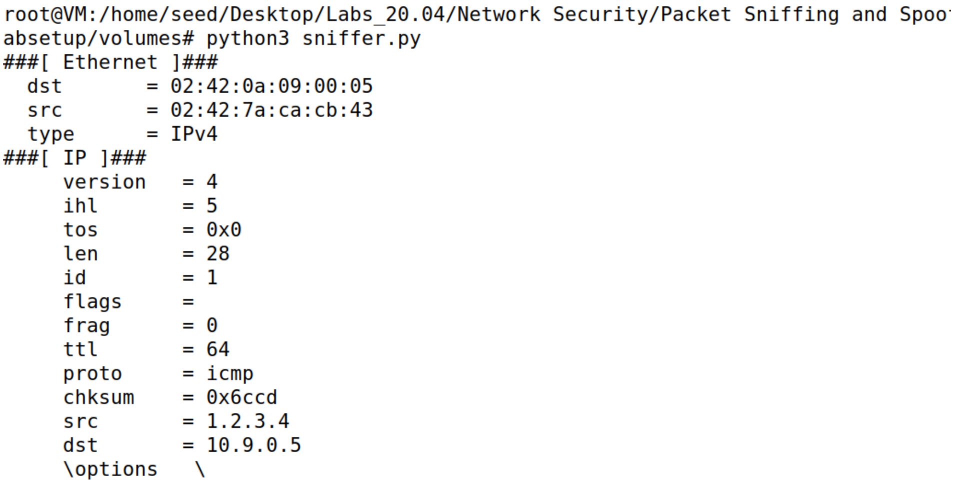
可以看到无论是 src 为 128.230.1.1 还是 dst 为 128.230.1.1，都能成功捕获到。

TASK1.2

实验过程和结果如下



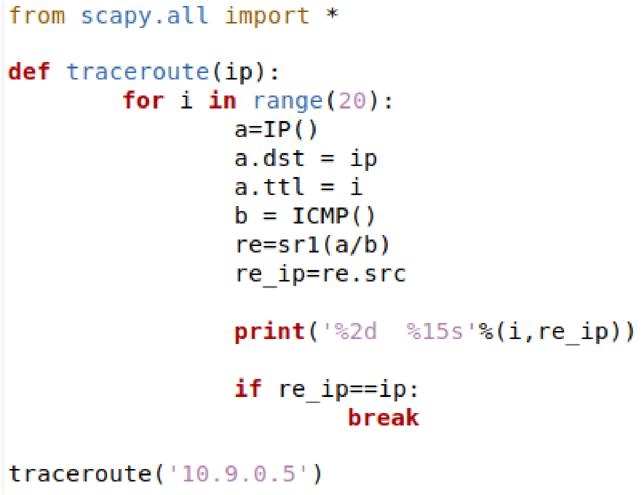
```
>>> from scapy.all import *
>>> a=IP()
>>> a.src='1.2.3.4'
>>> a.dst='10.9.0.5'
>>> b=ICMP()
>>> p=a/b
>>> send(p)
.
Sent 1 packets.
>>> ls(a)
version    : BitField  (4 bits)                  = 4               (4)
ihl        : BitField  (4 bits)                  = None            (None)
tos        : 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                       = '1.2.3.4'       (None)
dst        : DestIPField                         = '10.9.0.5'      (None)
options    : PacketListField                     = []              ([])
```



```
root@VM:/home/seed/Desktop/Labs_20.04/Network Security/Packet Sniffing and Spoo
absetup/volumes# python3 sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:7a:ca:cb:43
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 28
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x6ccd
     src       = 1.2.3.4
     dst       = 10.9.0.5
     \options   \
```

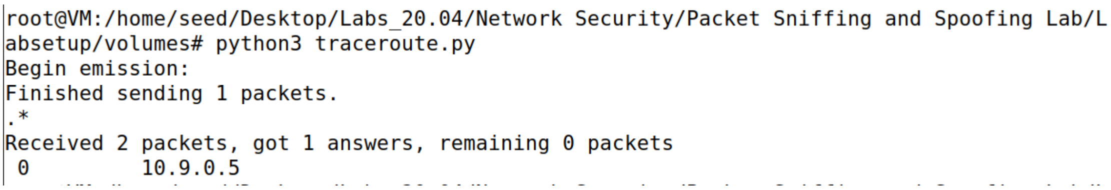
TASK1.3

traceroute.py 代码如下



```
from scapy.all import *

def traceroute(ip):
        for i in range(20):
                a=IP()
                a.dst = ip
                a.ttl = i
                b = ICMP()
                re=sr1(a/b)
                re_ip=re.src

                print('%2d  %15s'%(i,re_ip))

                if re_ip==ip:
                        break

traceroute('10.9.0.5')
```

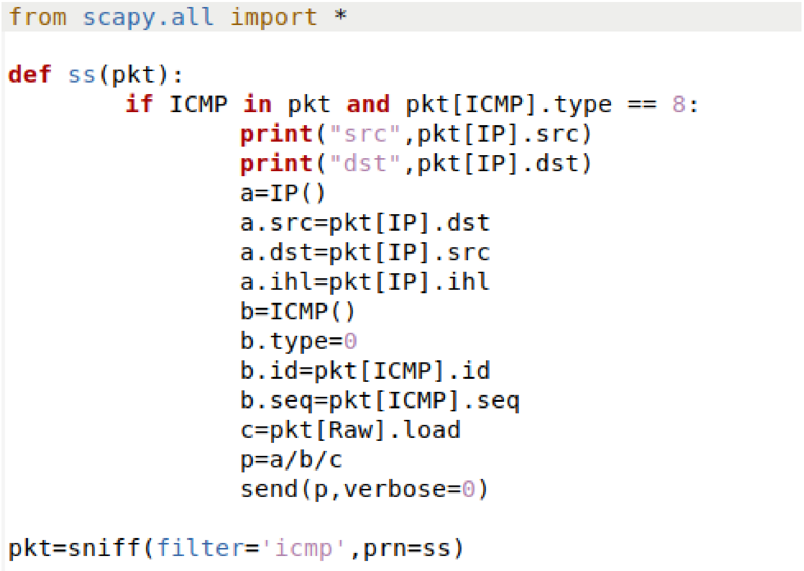
可以看到经过一跳到达了目的地址。



```
root@VM:/home/seed/Desktop/Labs_20.04/Network Security/Packet Sniffing and Spoofing Lab/L
absetup/volumes# python3 traceroute.py
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
 0         10.9.0.5
```

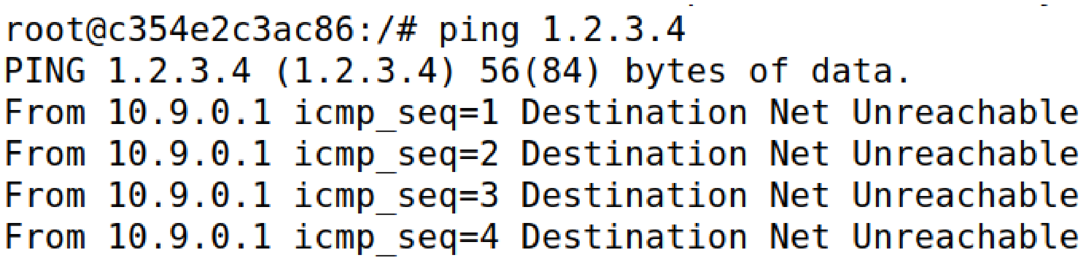
TASK1.4

ss.py 的代码如下

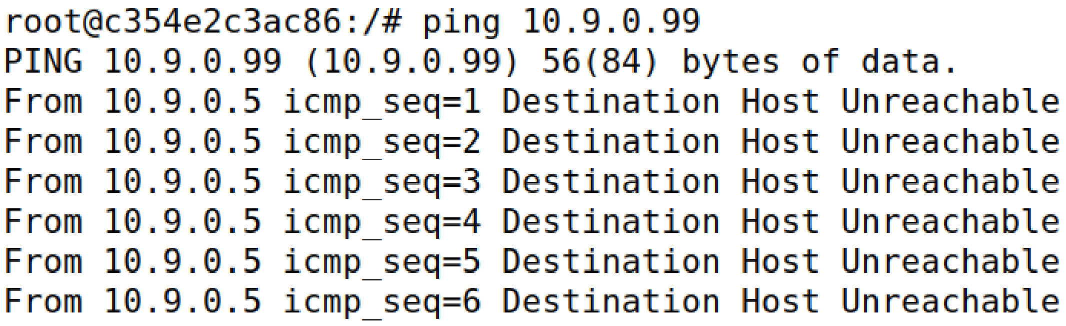


```
from scapy.all import *

def ss(pkt):
        if ICMP in pkt and pkt[ICMP].type == 8:
                print("src",pkt[IP].src)
                print("dst",pkt[IP].dst)
                a=IP()
                a.src=pkt[IP].dst
                a.dst=pkt[IP].src
                a.ihl=pkt[IP].ihl
                b=ICMP()
                b.type=0
                b.id=pkt[ICMP].id
                b.seq=pkt[ICMP].seq
                c=pkt[Raw].load
                p=a/b/c
                send(p,verbose=0)

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

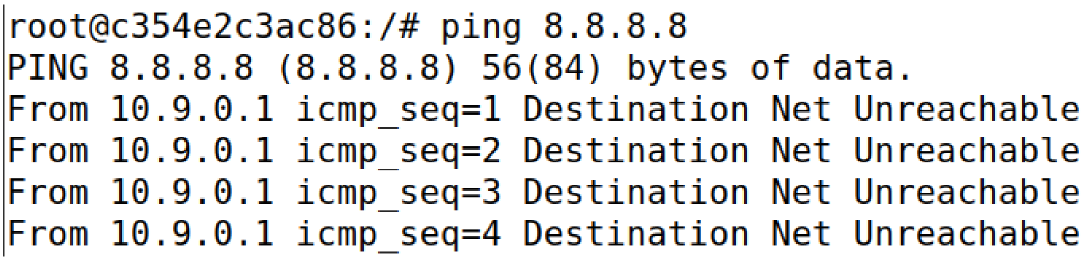
在未运行 ss.py 时，我们 ping 一下三个地址分别得到如下情况，可见三个地址都是不可到达的，但是外网的两个地址不可达是 10.9.0.1 告诉我们的，而内网地址不可到达是 10.9.0.5 告诉我们的，也就是说 10.9.0.1 是出内网的网关。



```
root@c354e2c3ac86:/# ping 1.2.3.4
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
```

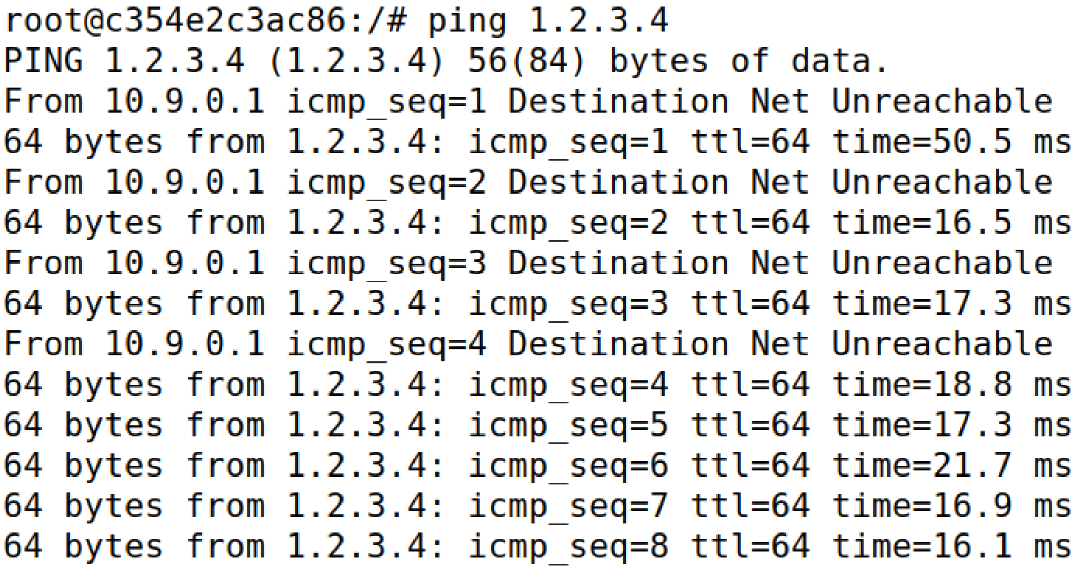


```
root@c354e2c3ac86:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
From 10.9.0.5 icmp_seq=4 Destination Host Unreachable
From 10.9.0.5 icmp_seq=5 Destination Host Unreachable
From 10.9.0.5 icmp_seq=6 Destination Host Unreachable
```

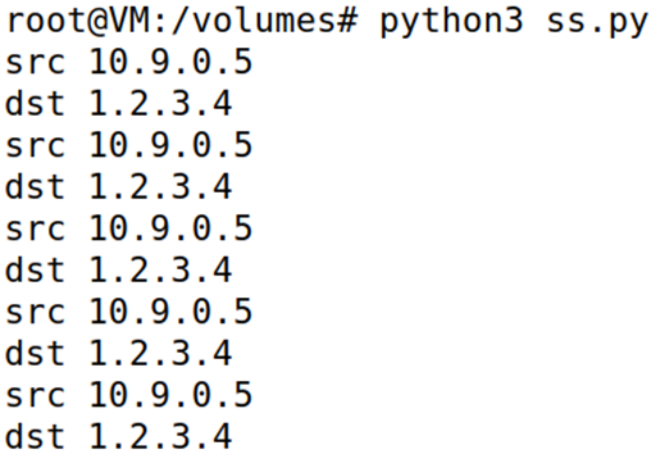


```
root@c354e2c3ac86:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Net Unreachable
From 10.9.0.1 icmp_seq=2 Destination Net Unreachable
From 10.9.0.1 icmp_seq=3 Destination Net Unreachable
From 10.9.0.1 icmp_seq=4 Destination Net Unreachable
```

运行 ss.py 后再次 ping 这三个地址，下图是 ping 1.2.3.4 时的结果和 ss.py 的输出，可以看到此时 1.2.3.4 可达。

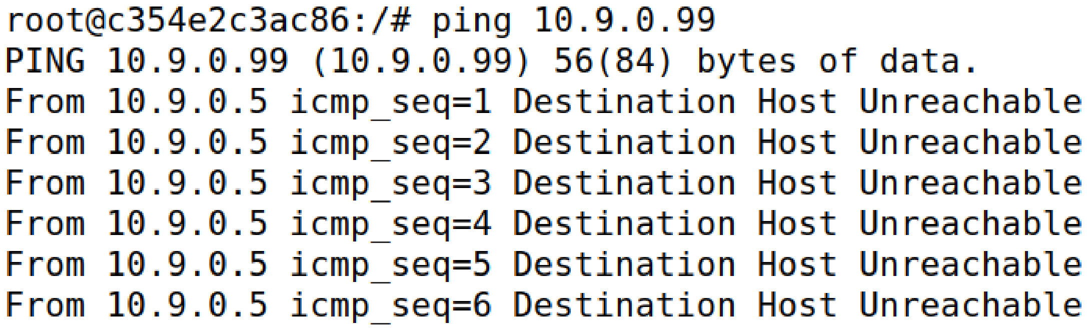


```
root@c354e2c3ac86:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Net Unreachable
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=50.5 ms
From 10.9.0.1 icmp_seq=2 Destination Net Unreachable
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=16.5 ms
From 10.9.0.1 icmp_seq=3 Destination Net Unreachable
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=17.3 ms
From 10.9.0.1 icmp_seq=4 Destination Net Unreachable
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=18.8 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=17.3 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=21.7 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=16.9 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=16.1 ms
```



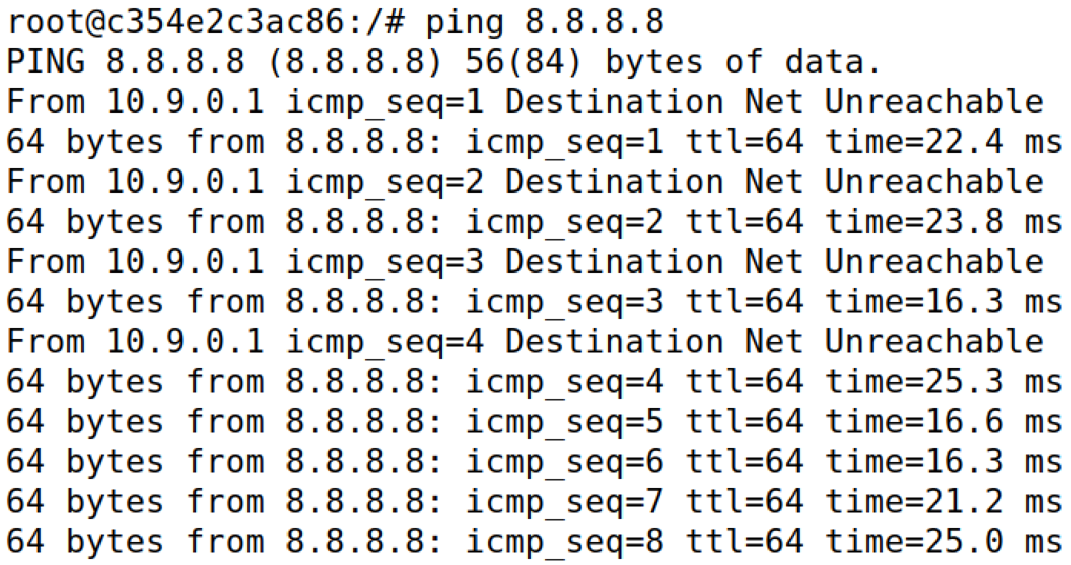
```
root@VM:/volumes# python3 ss.py
src 10.9.0.5
dst 1.2.3.4
src 10.9.0.5
dst 1.2.3.4
src 10.9.0.5
dst 1.2.3.4
src 10.9.0.5
dst 1.2.3.4
src 10.9.0.5
dst 1.2.3.4
```

下面是 ping 10.9.0.99 时的结果，此时 ss.py 并没有任何输出，不可达。

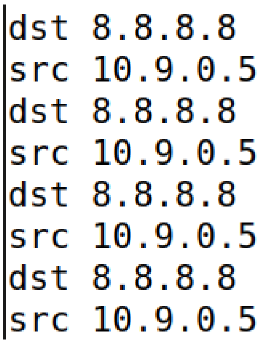


```
root@c354e2c3ac86:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
From 10.9.0.5 icmp_seq=4 Destination Host Unreachable
From 10.9.0.5 icmp_seq=5 Destination Host Unreachable
From 10.9.0.5 icmp_seq=6 Destination Host Unreachable
```

下面是 ping 8.8.8.8 时的结果和 ss.py 的输出，地址可达。



```
root@c354e2c3ac86:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Net Unreachable
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=22.4 ms
From 10.9.0.1 icmp_seq=2 Destination Net Unreachable
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=23.8 ms
From 10.9.0.1 icmp_seq=3 Destination Net Unreachable
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=16.3 ms
From 10.9.0.1 icmp_seq=4 Destination Net Unreachable
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=25.3 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=16.6 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=64 time=16.3 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=64 time=21.2 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=64 time=25.0 ms
```



```
dst 8.8.8.8
src 10.9.0.5
dst 8.8.8.8
src 10.9.0.5
dst 8.8.8.8
src 10.9.0.5
dst 8.8.8.8
src 10.9.0.5
```

根据上面的情况可知attacker机也就是上面说的10.9.0.1是出内网的网关，而本机是10.9.0.5，在未运行ss.py时，都不能ping通，而运行之后，外网的两个地址能 ping 通是因为，他们两个的报文要经过 attacker 机出去，所以被attacker检测到，并伪造了返回报文，让本机误以为可以ping通，但是ping内网的地址时，不需要经过 attacker，所以 attacker 没有返回伪造报文，而10.9.0.99这个内网地址又是不存在的，结果就和没运行ss.py时一样ping不通。