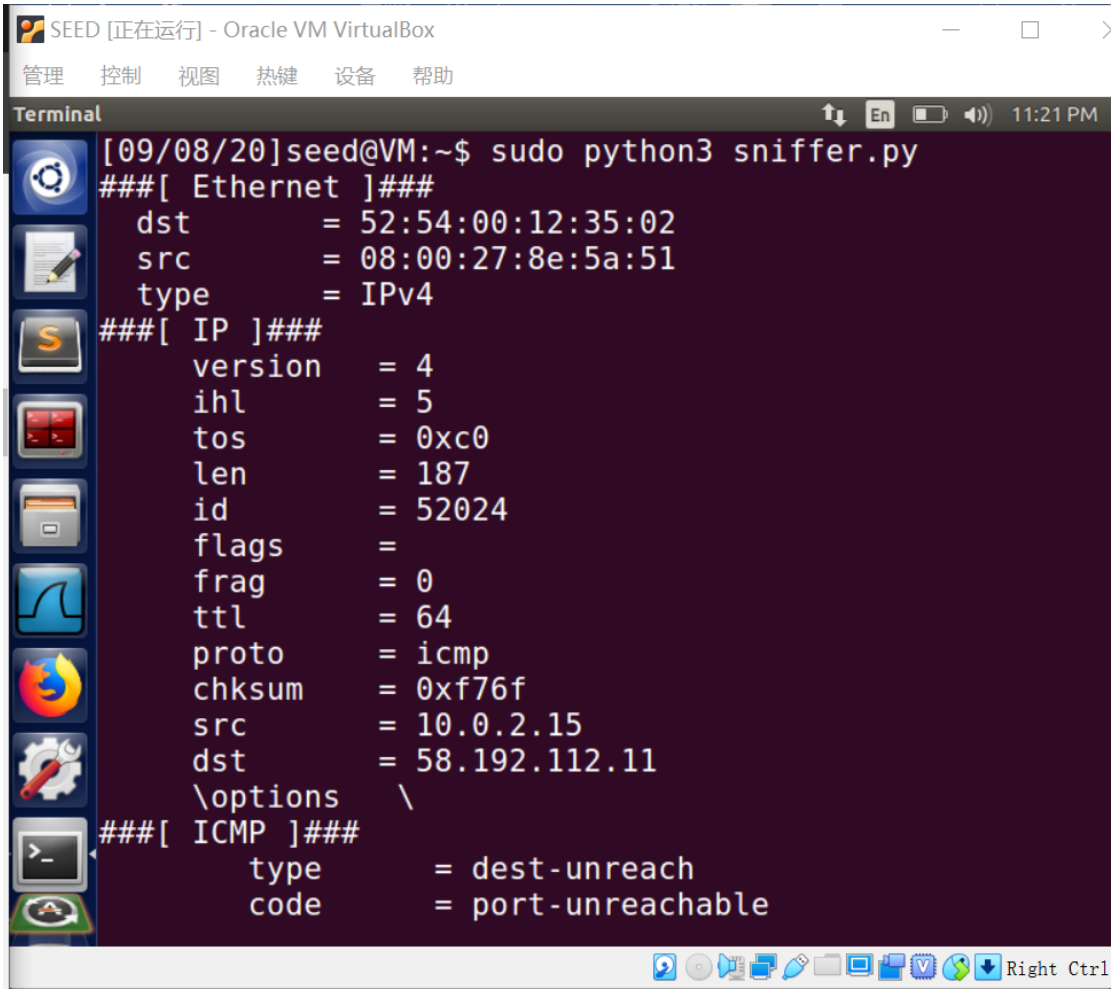
Task set 1

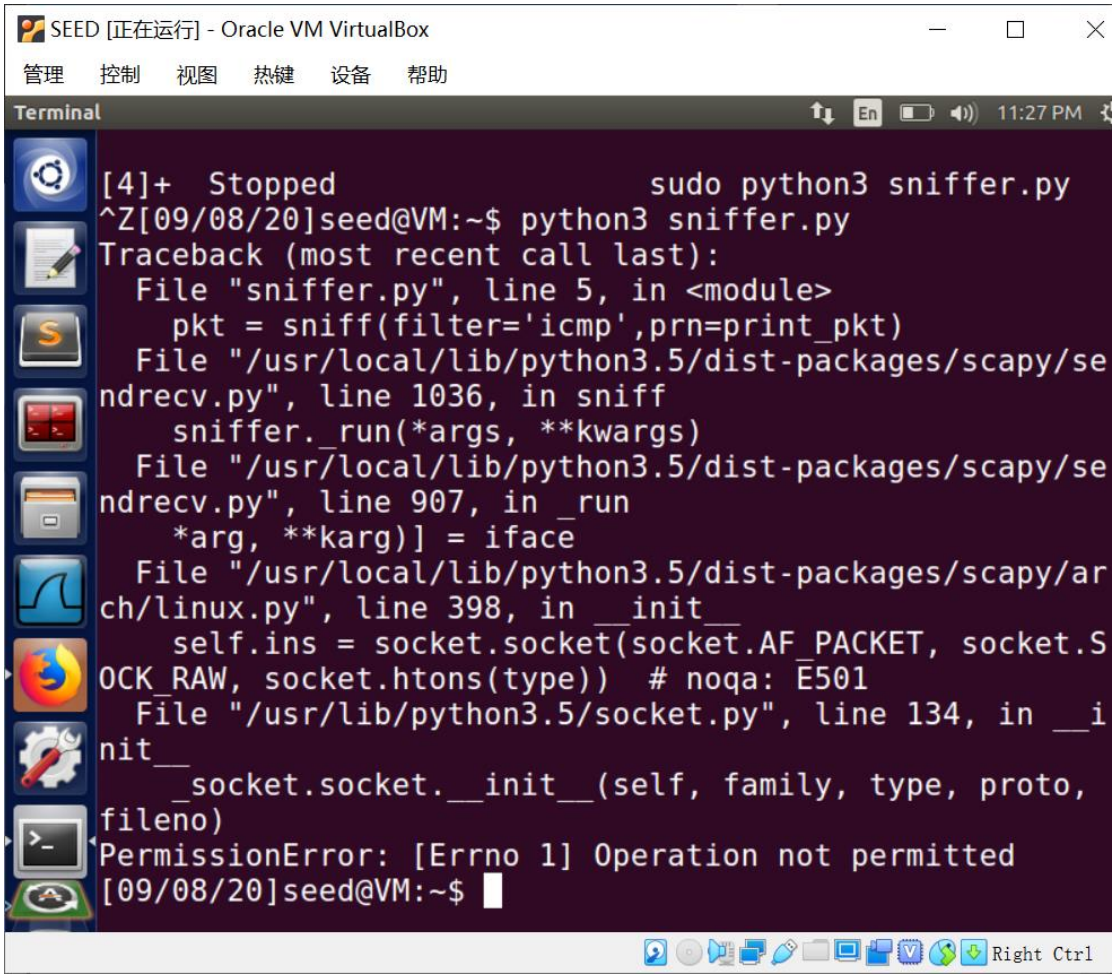
Task1.1

Task1.1A:

使用 root 权限执行时获得了如下结果，可以看到接收到了 IP 报文，所以确实捕获了报文

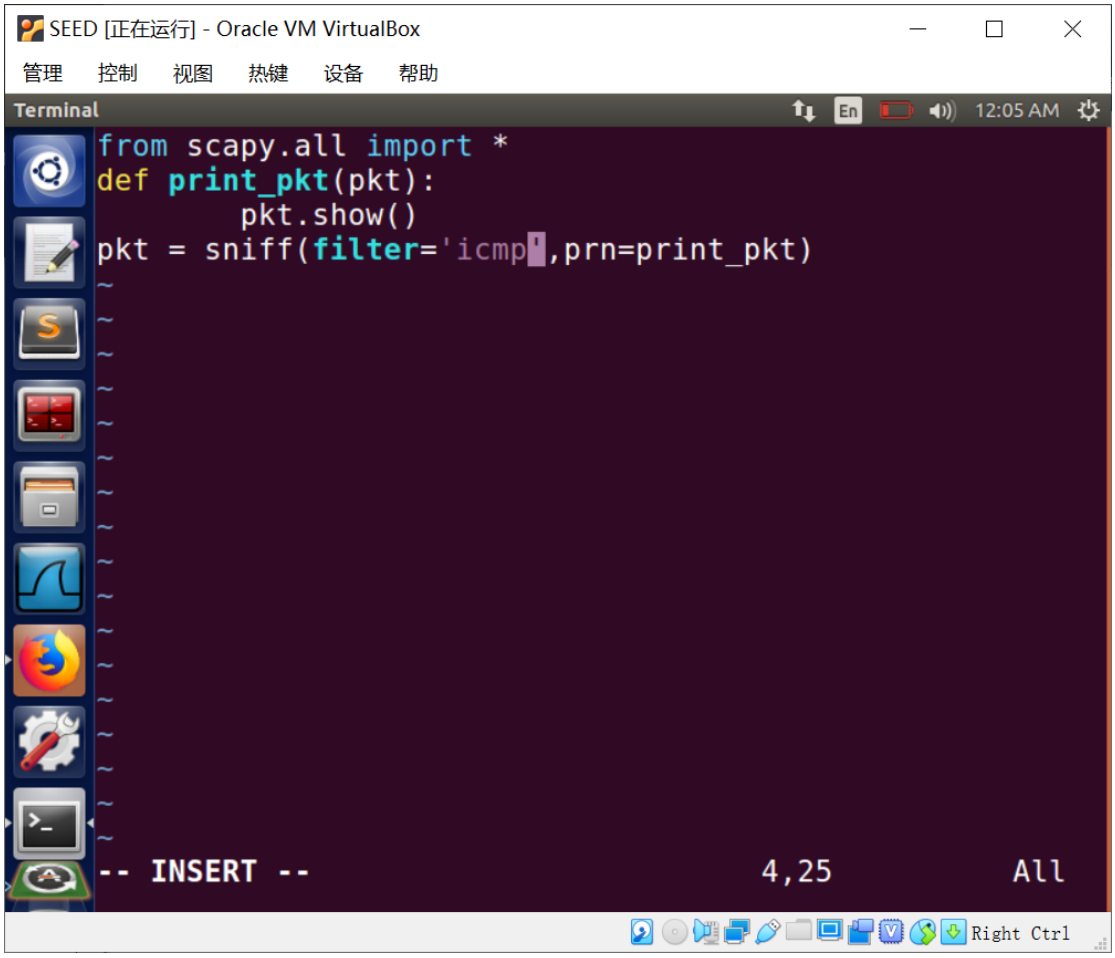


在不使用 root 权限再次执行时发现已经不能捕获到报文

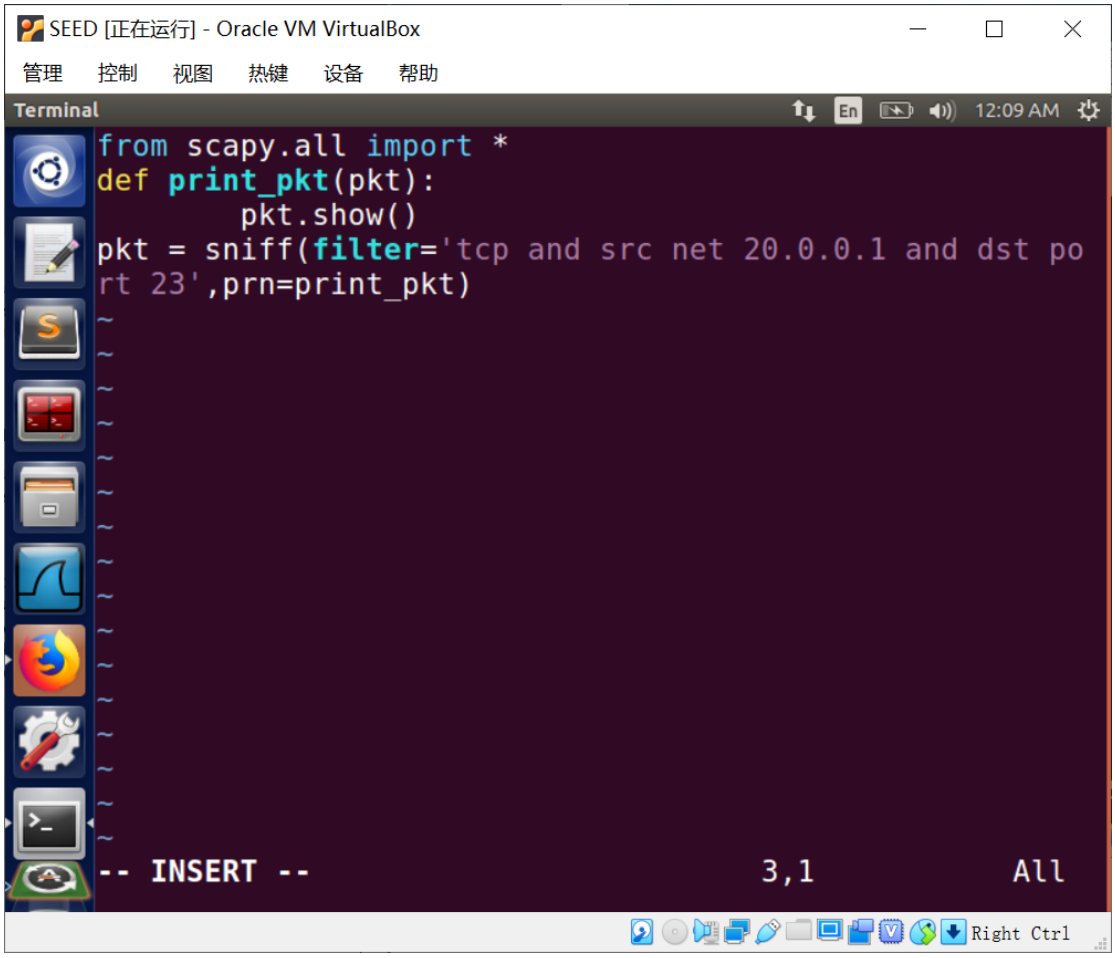


Task1.1B

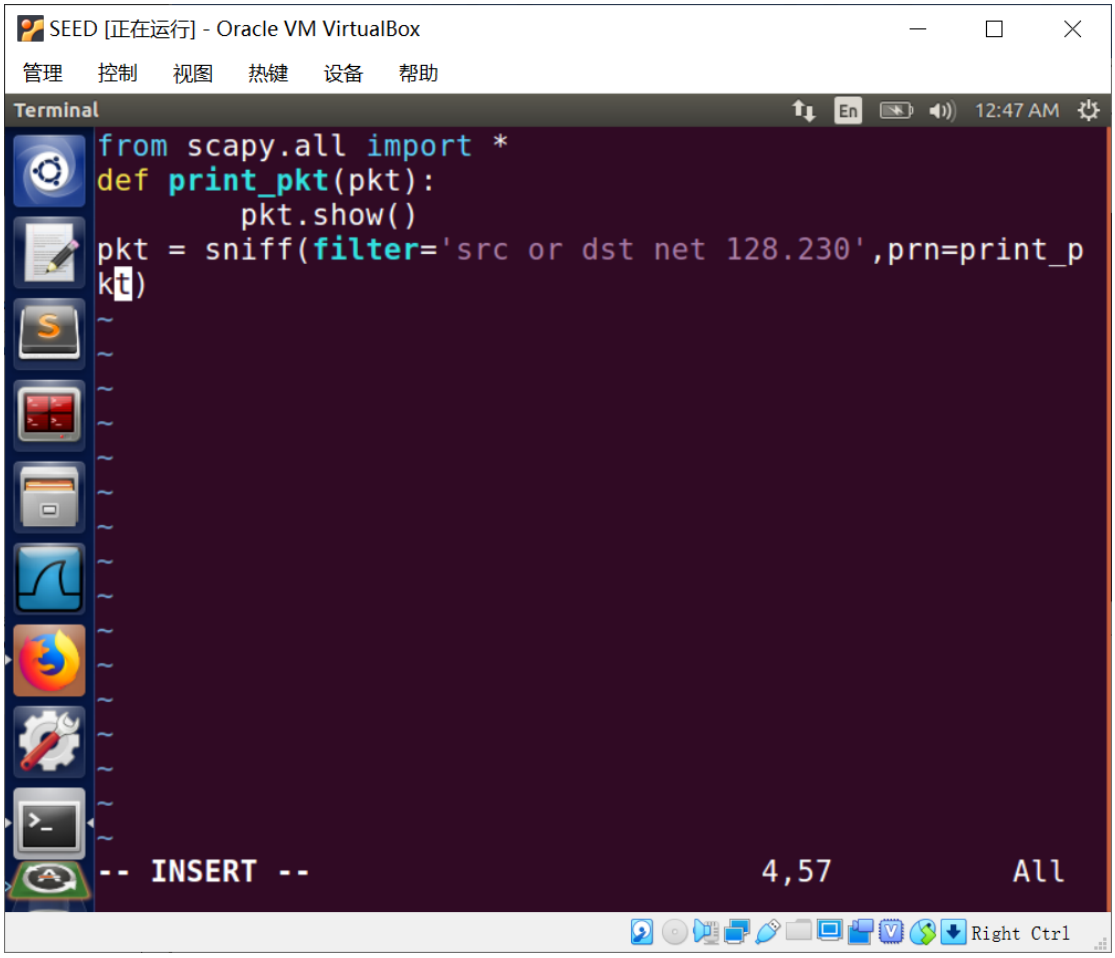
捕获 ICMP 报文:



捕获来自 IP 为 20.0.0.1，到达端口为 23 的 TCP 报文：

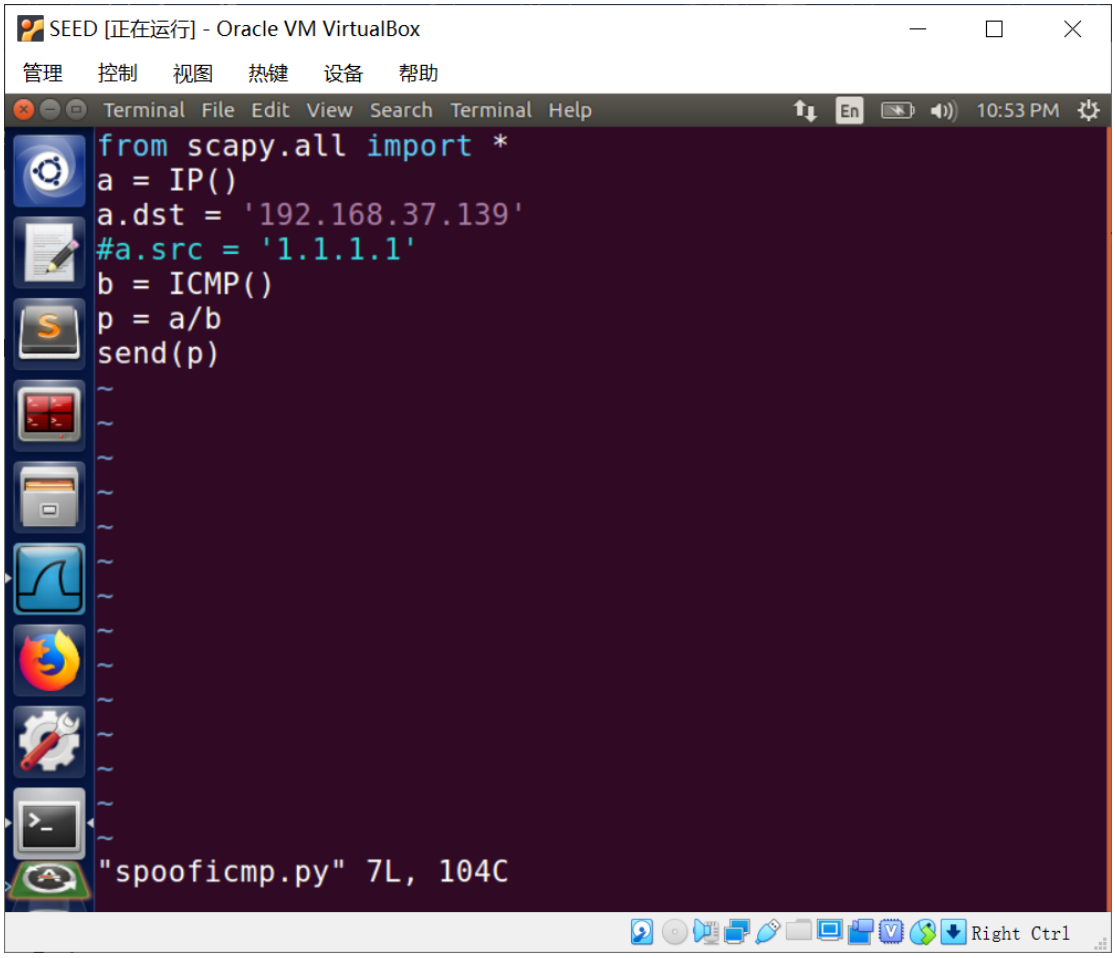


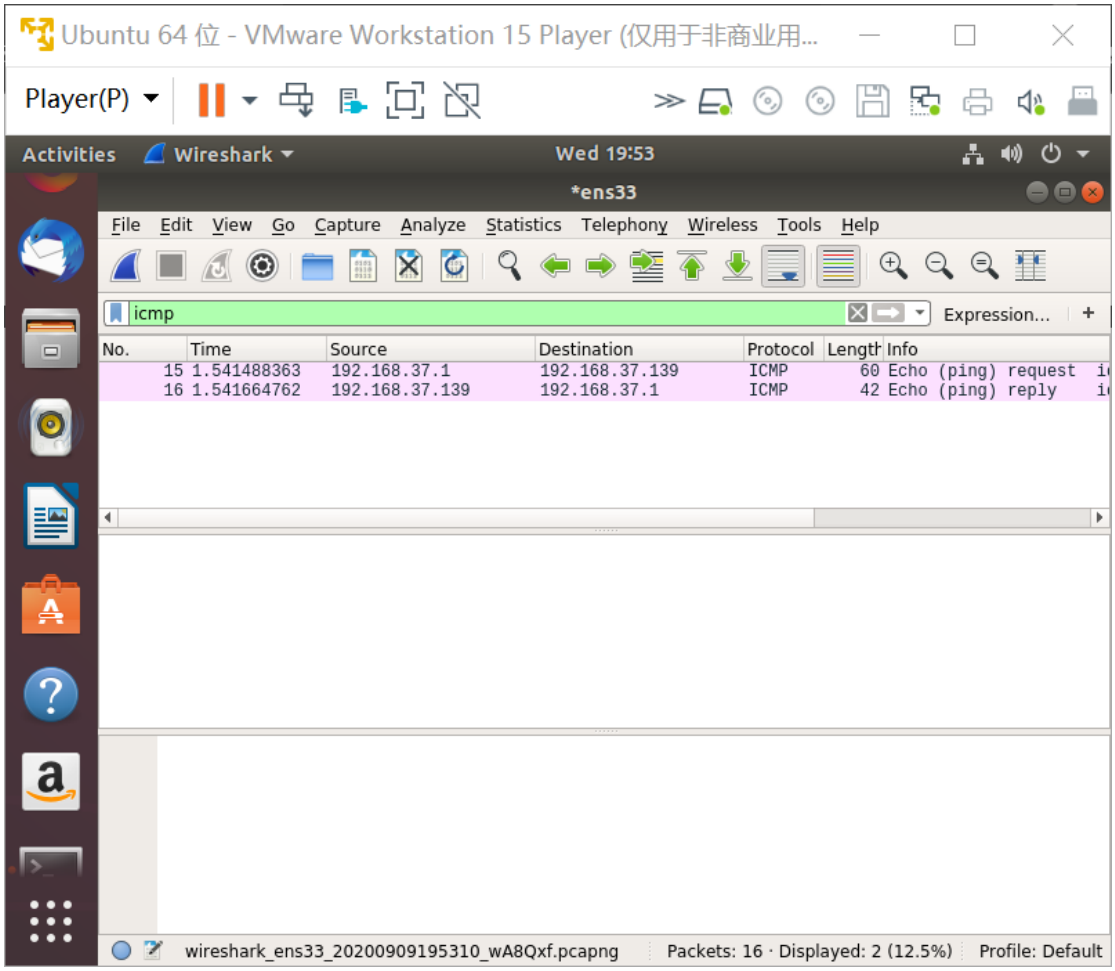
捕获来自或者到达某个子网的报文:



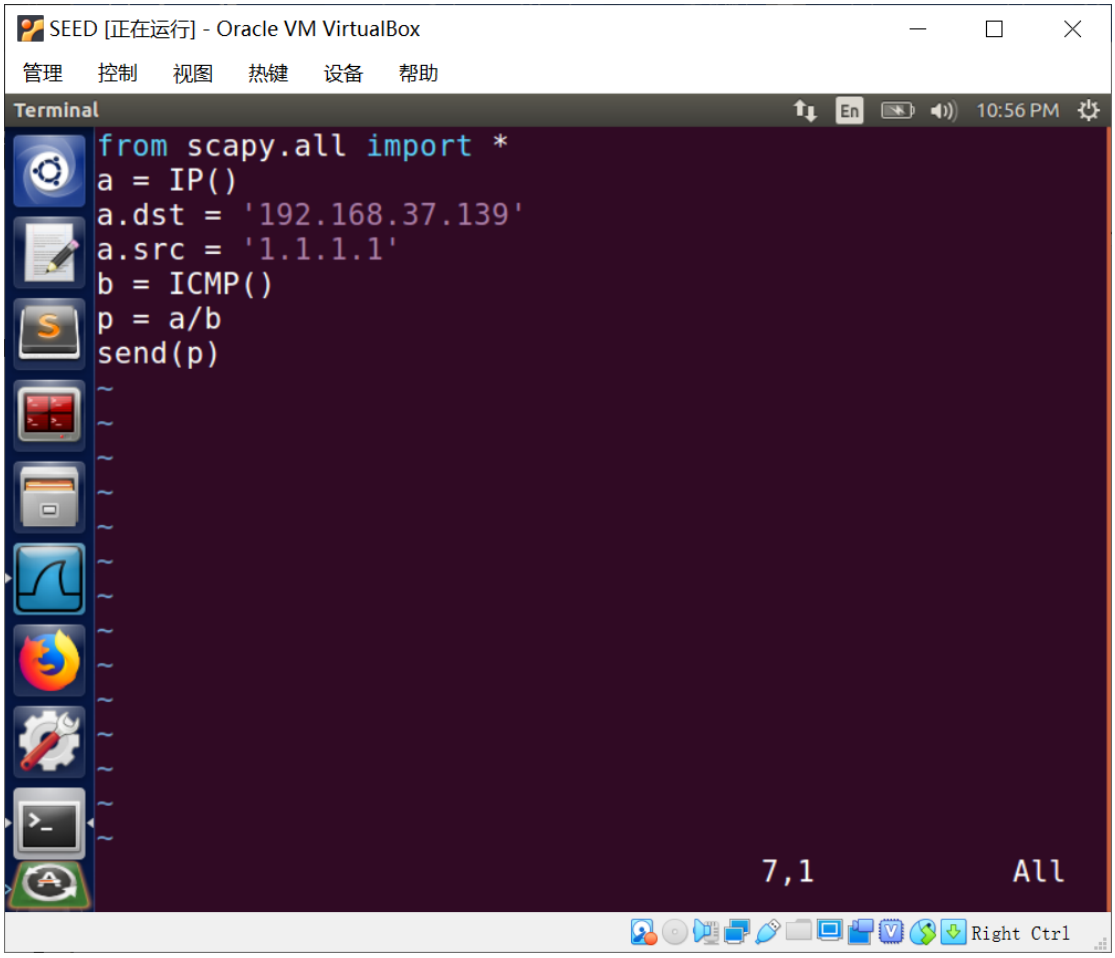
Task1.2:

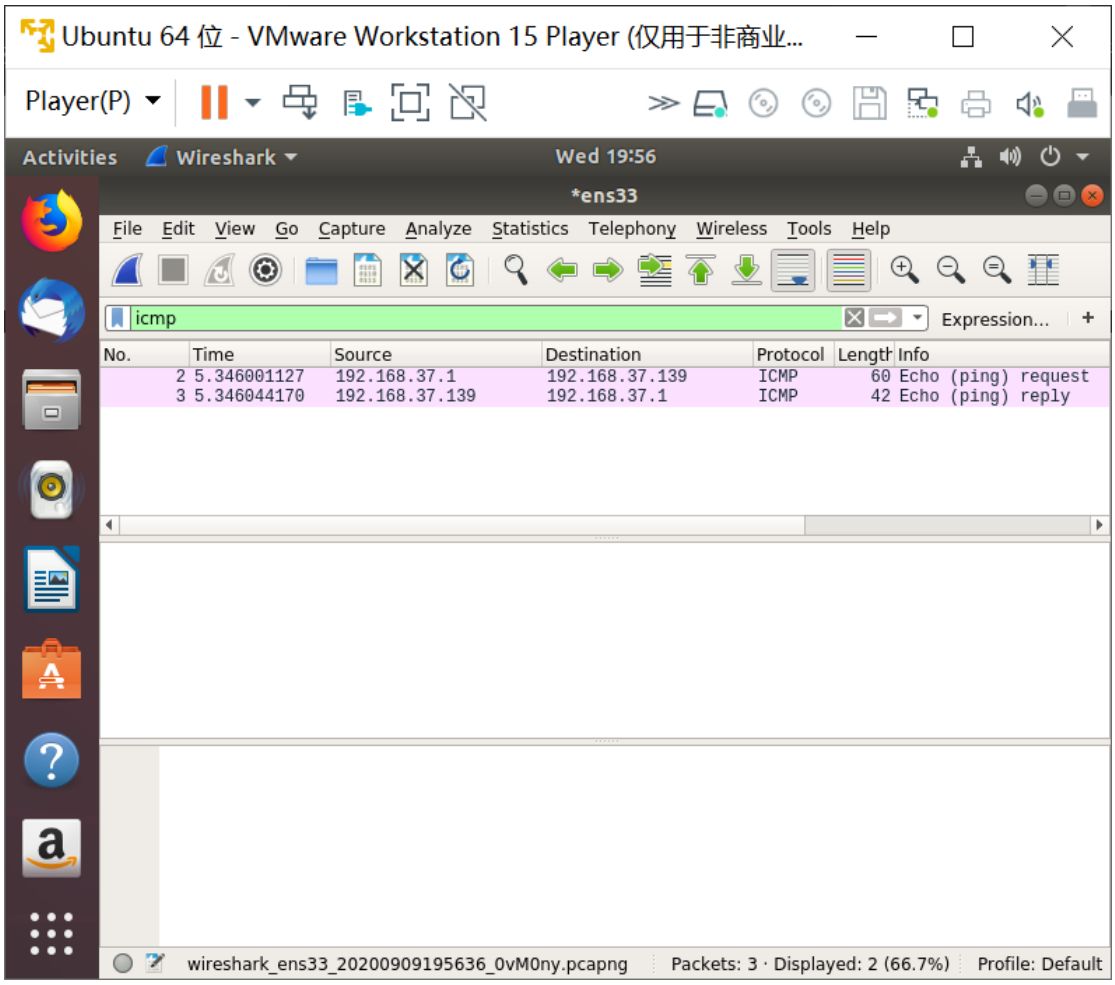
通过 scapy 构造 icmp 报文发送到另一台虚拟机，发现虚拟机上 wireshark 捕获到了相应报文:


SEED [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Terminal File Edit View Search Terminal Help
10:53 PM
from scapy.all import *
a = IP()
a.dst = '192.168.37.139'
#a.src = '1.1.1.1'
b = ICMP()
p = a/b
send(p)
"spooficmp.py" 7L, 104C
Right Ctrl



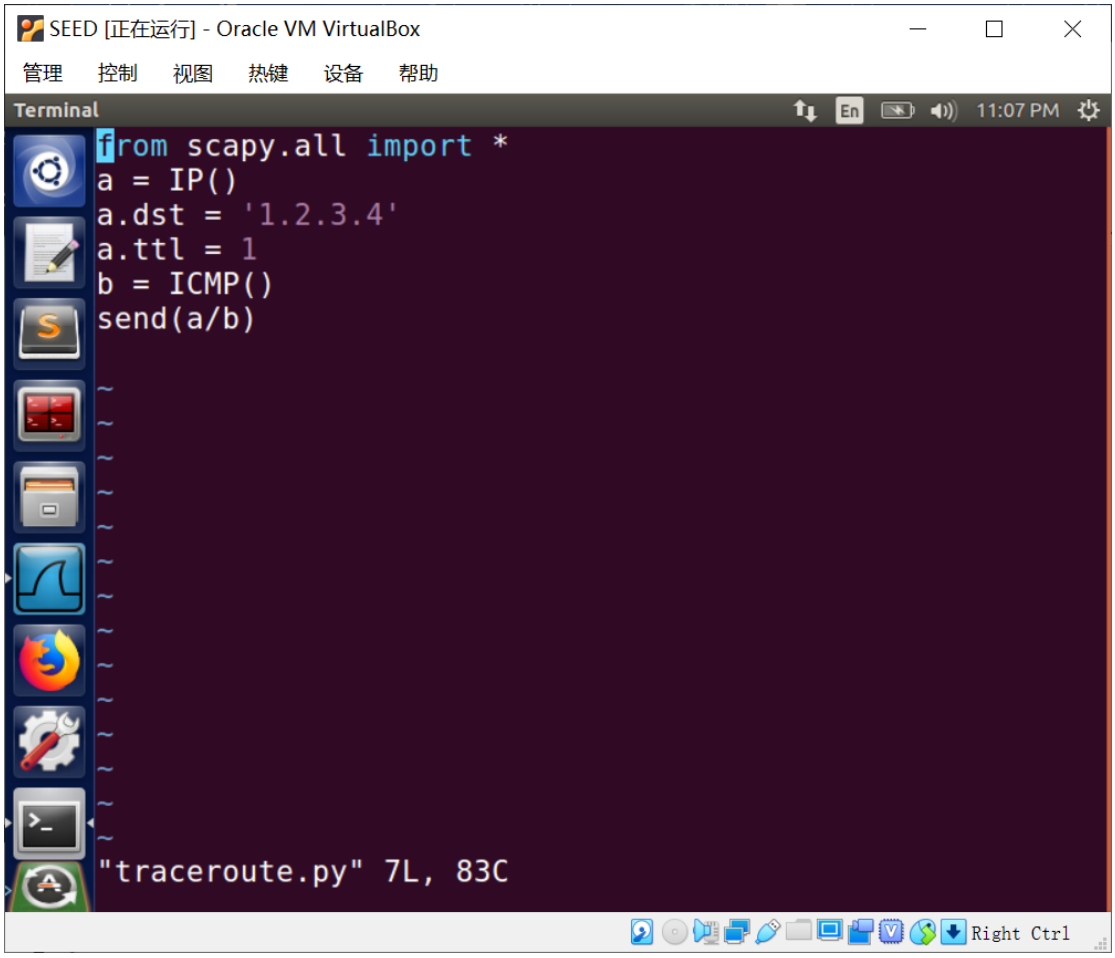
但是当使用 scapy 设置源地址时，不知道什么原因，设置成任何地址在另一台虚拟机上仍然接收为 192.168.37.1

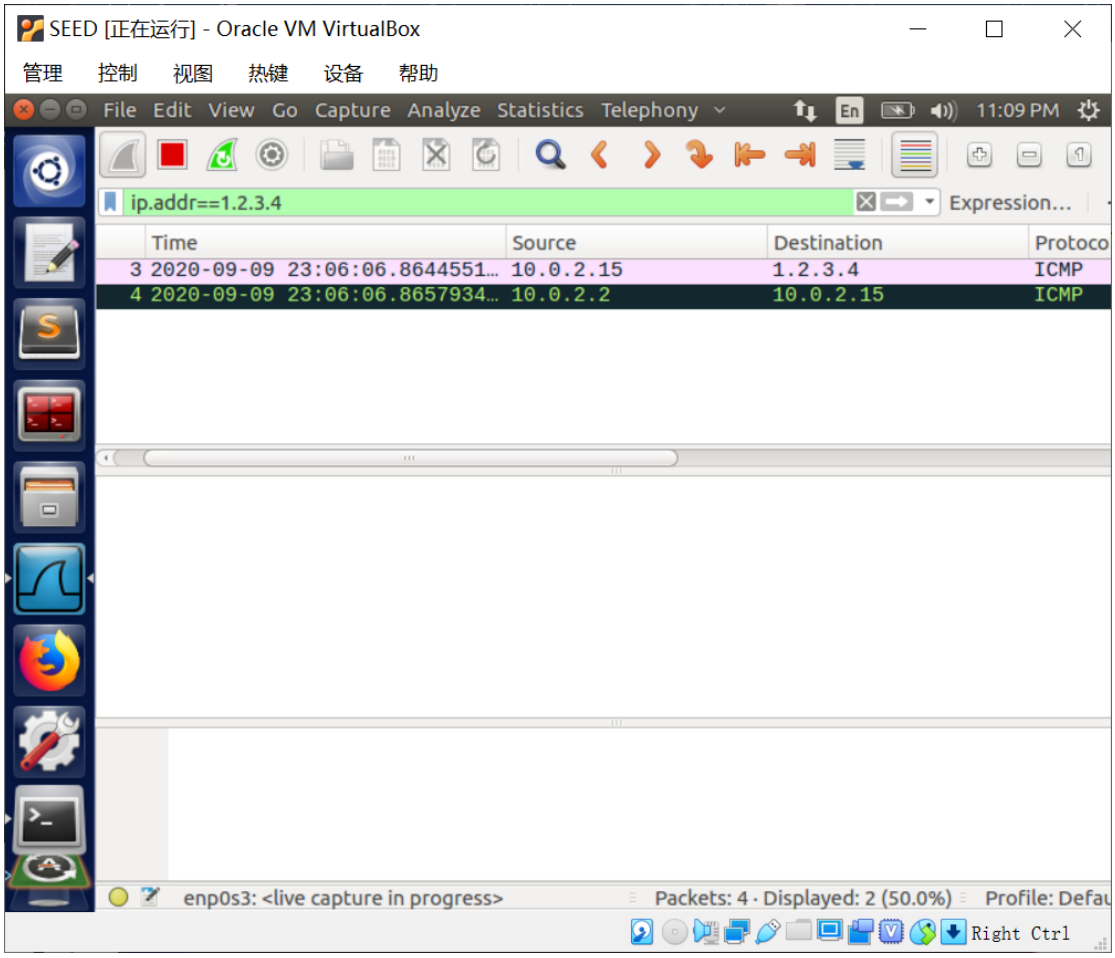

SEED [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Terminal
10:56 PM
from scapy.all import *
a = IP()
a.dst = '192.168.37.139'
a.src = '1.1.1.1'
b = ICMP()
p = a/b
send(p)
7,1 All
Right Ctrl



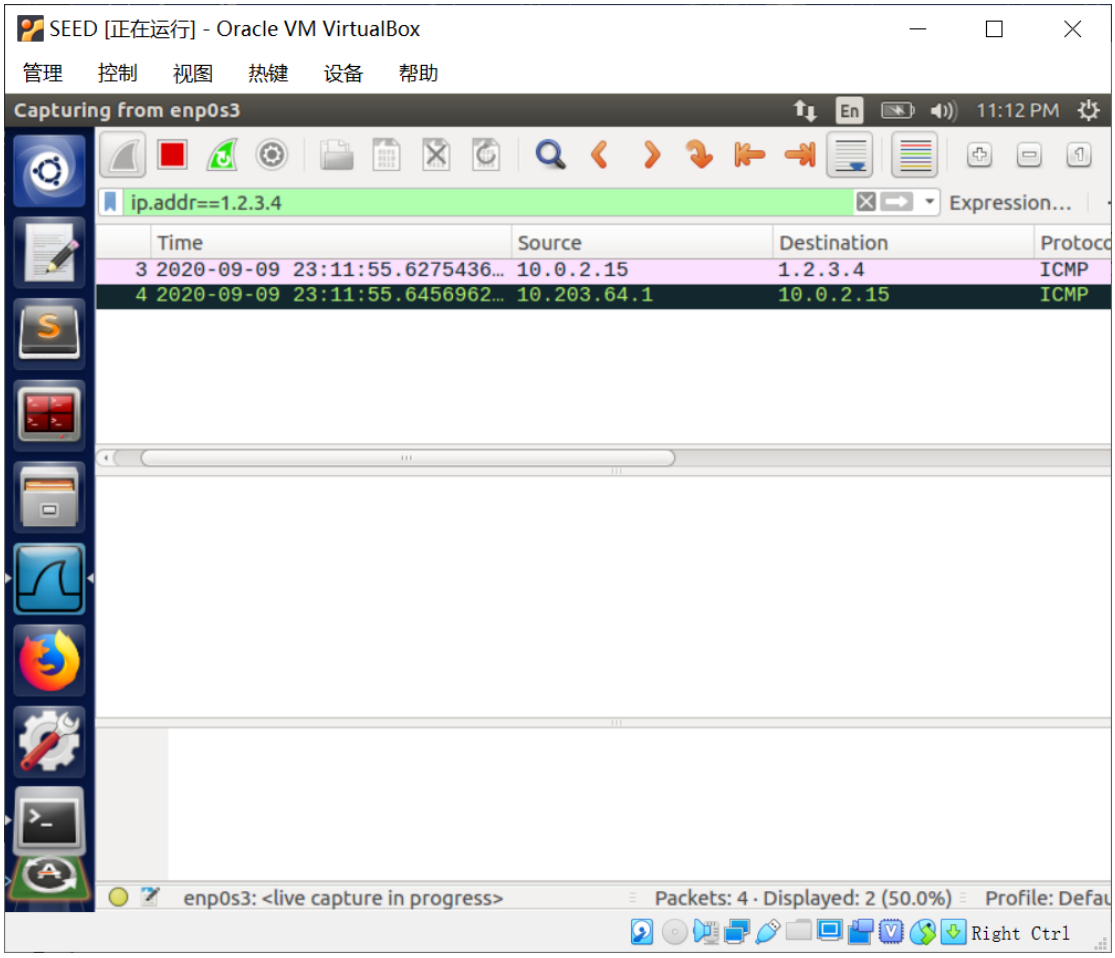
Task1.3:

设置 ttl 为 1 时，可以看到 wireshark 中 icmp 报文到达 10.0.2.2 就超时了：

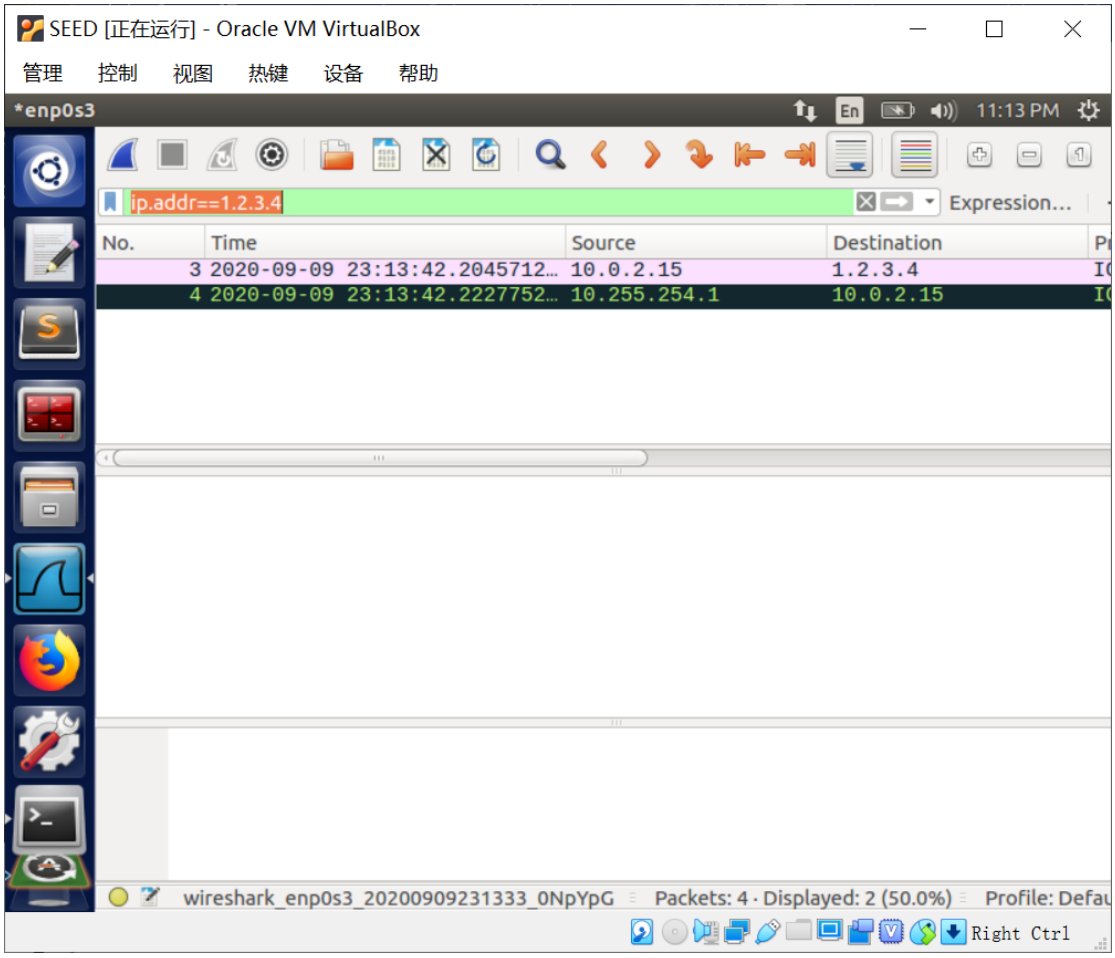

SEED [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Terminal
11:07 PM
from scapy.all import *
a = IP()
a.dst = '1.2.3.4'
a.ttl = 1
b = ICMP()
send(a/b)
"traceroute.py" 7L, 83C
Right Ctrl



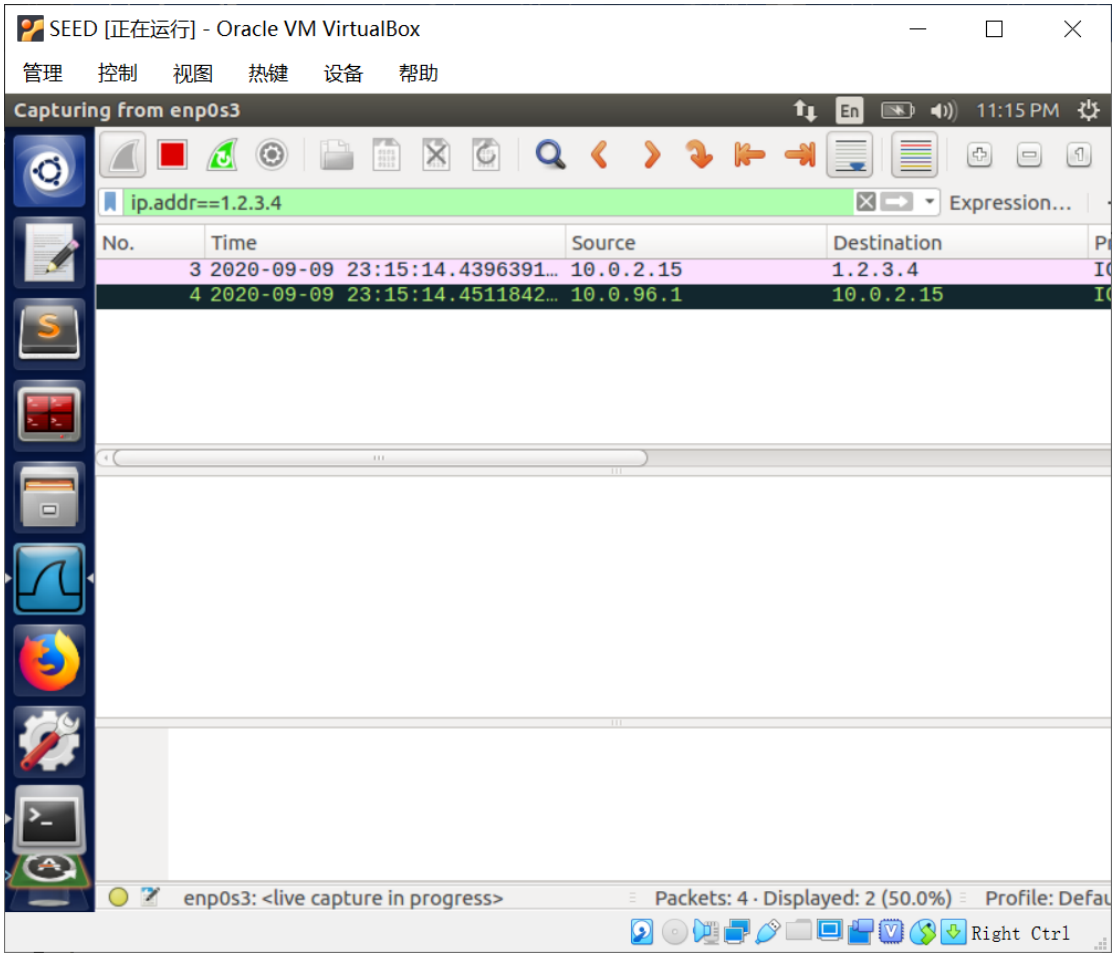
设置 ttl 为 2，可以看到这次到达 10.203.64.1 出现了超时：



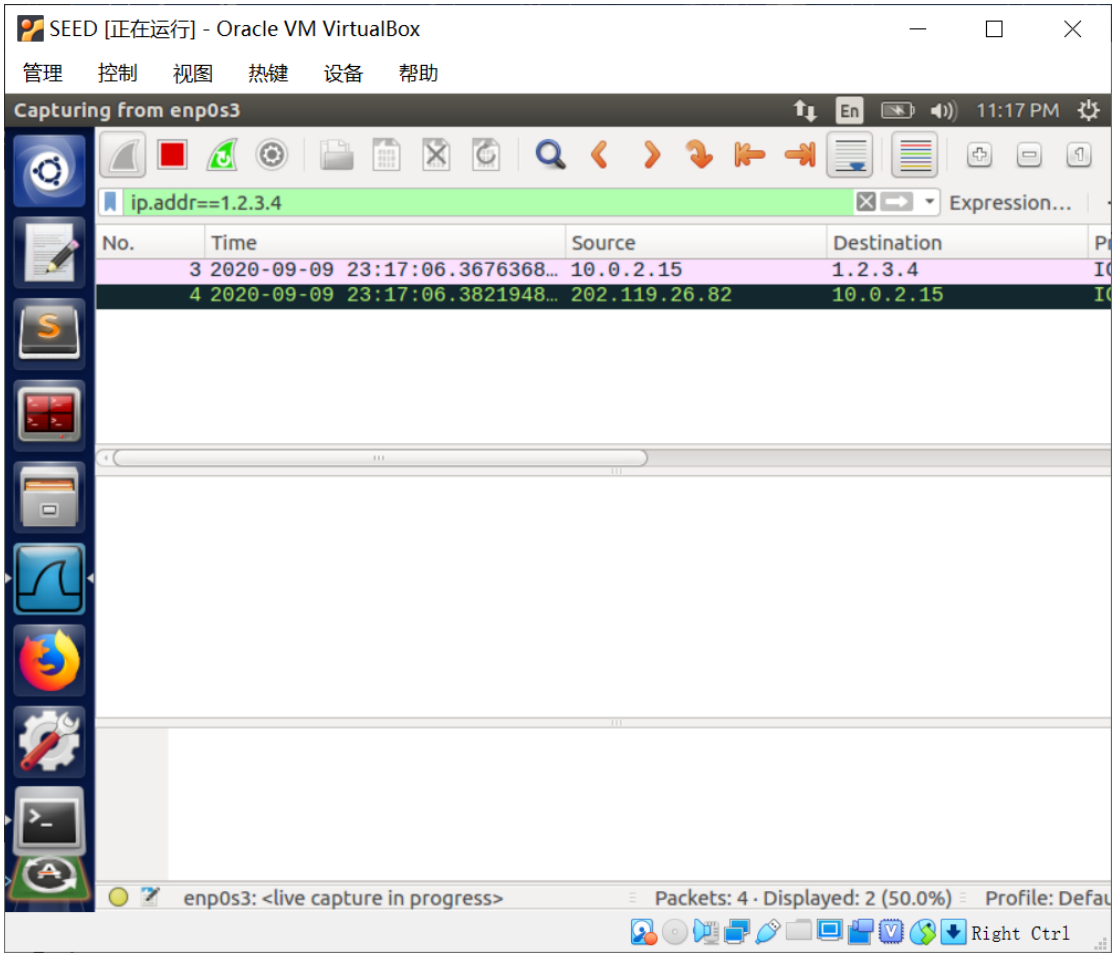
设置 ttl 为 3，可以看到这次在 10.255.254.1 出现超时：



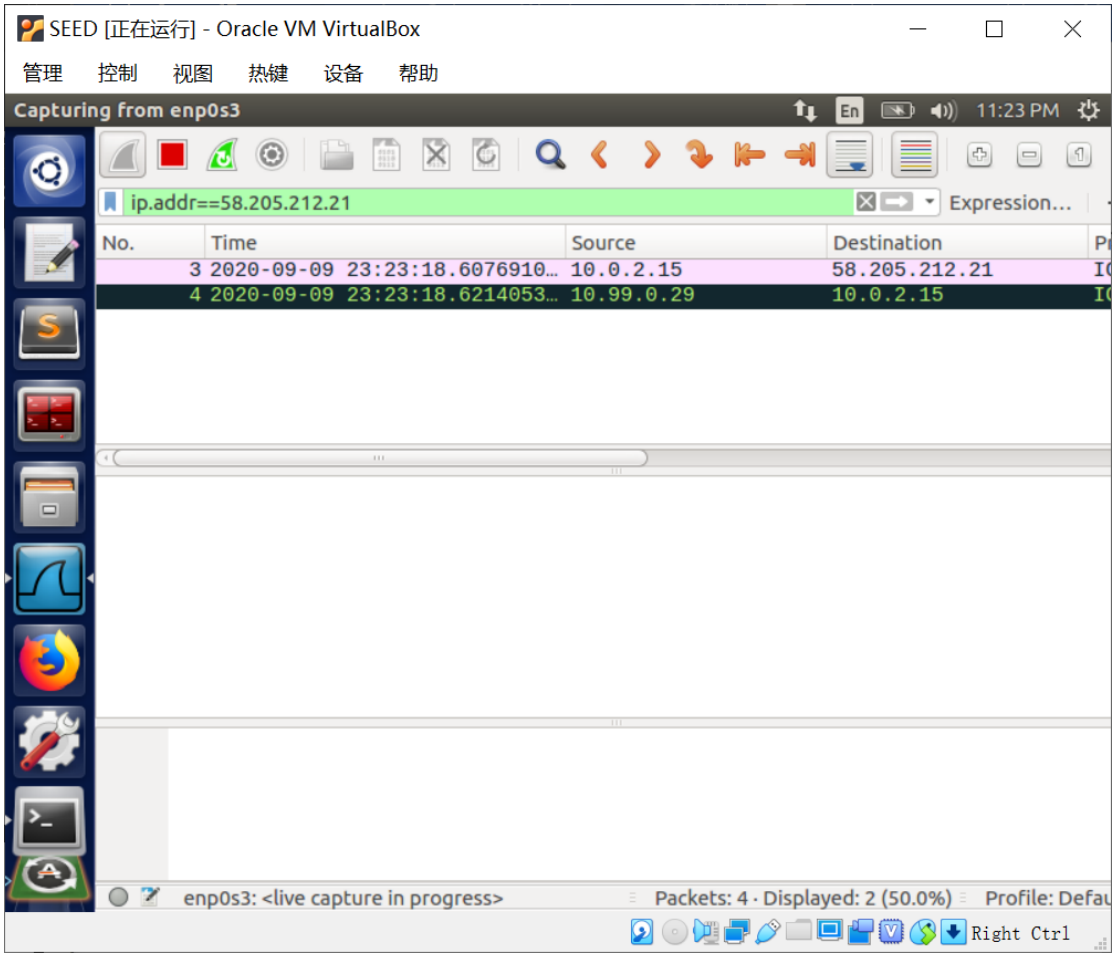
设置 ttl 为 4，可以看到这次在 10.0.96.1 处出现了超时：



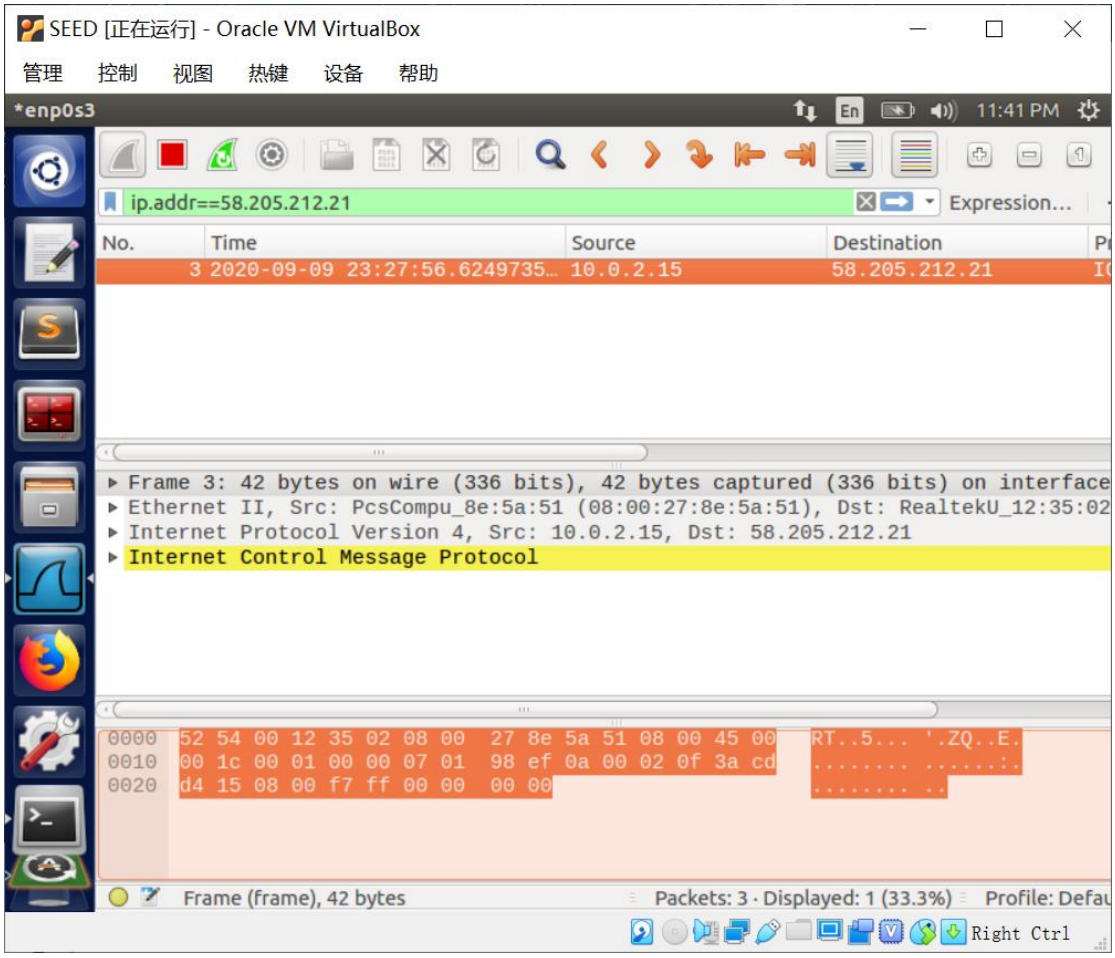
设置 ttl 为 5，看到这一次在 202.119.26.82 处出现了超时：



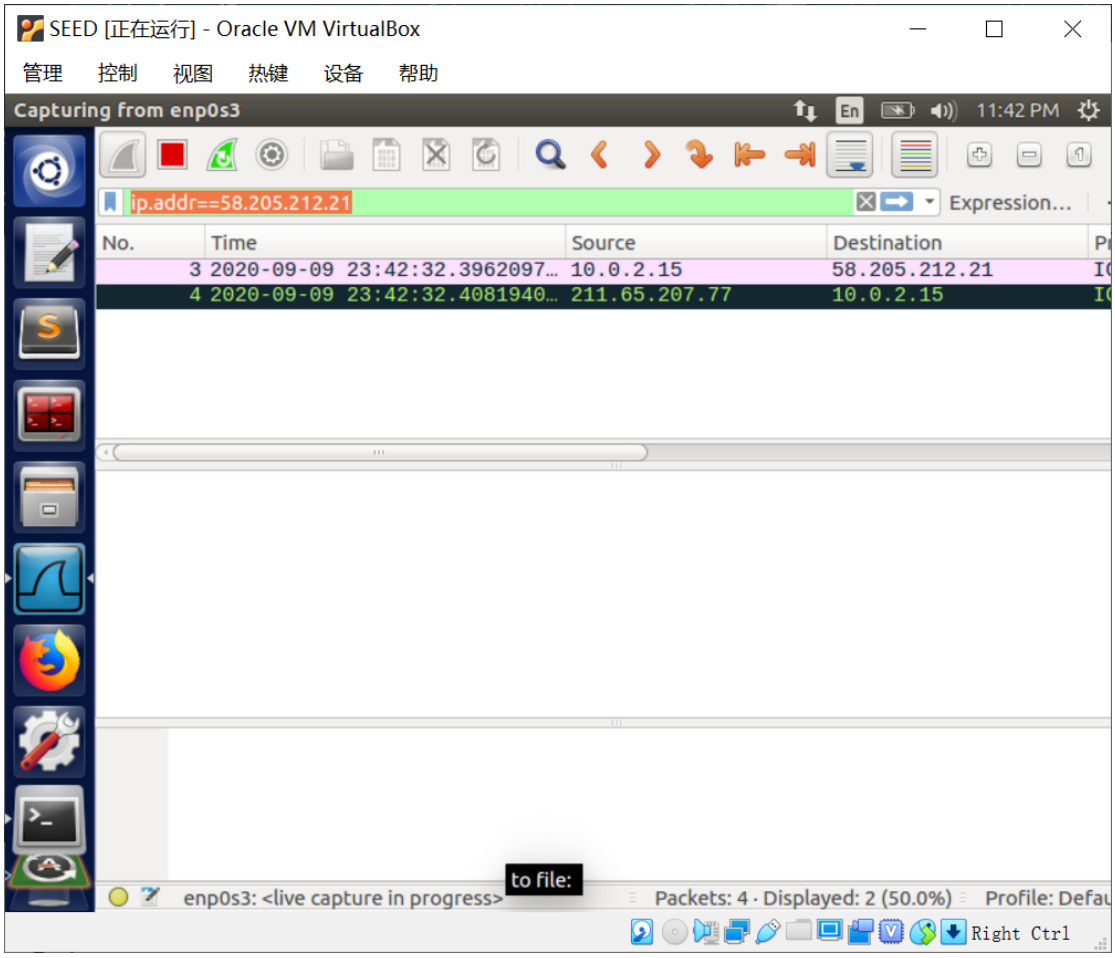
设置 ttl 为 6，可以看到在到达地址 10.99.0.29 处超时：



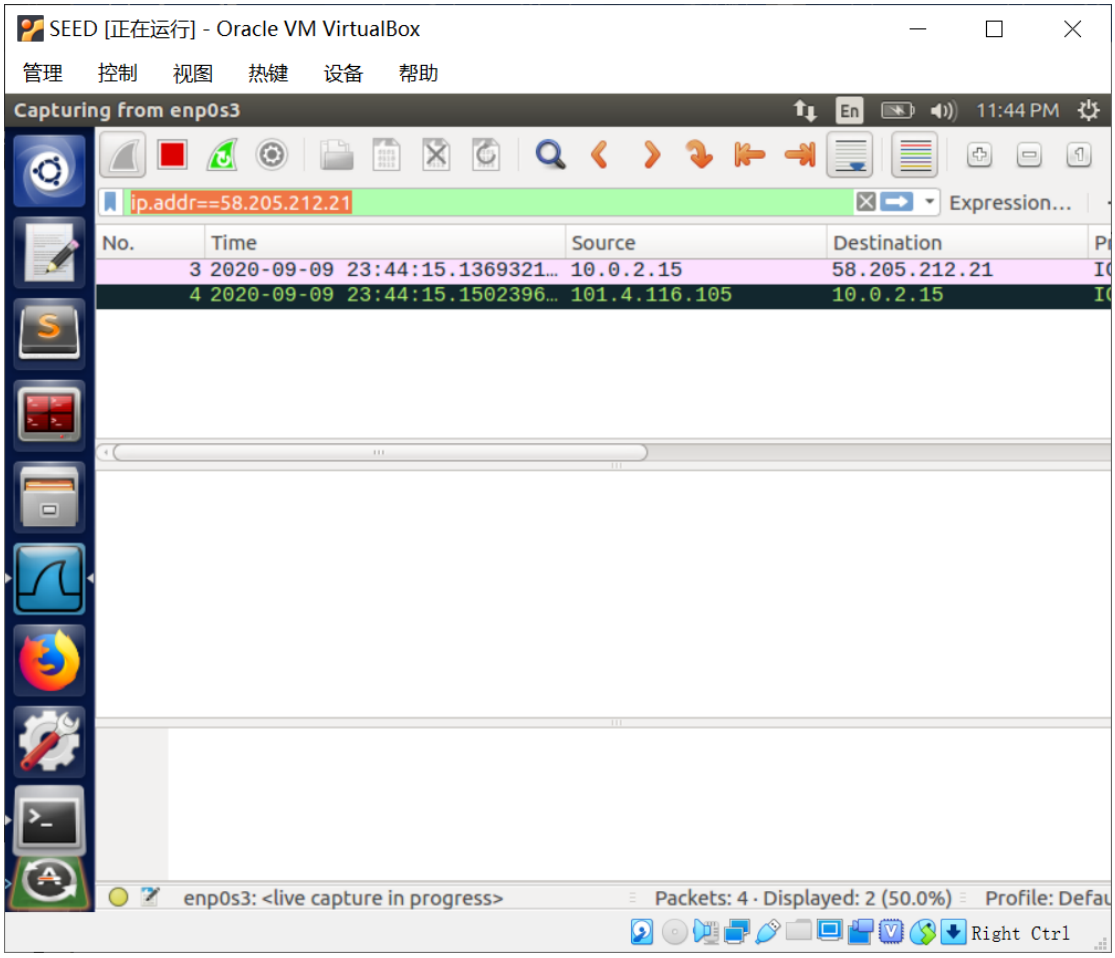
设置 ttl 为 7，报文丢失：



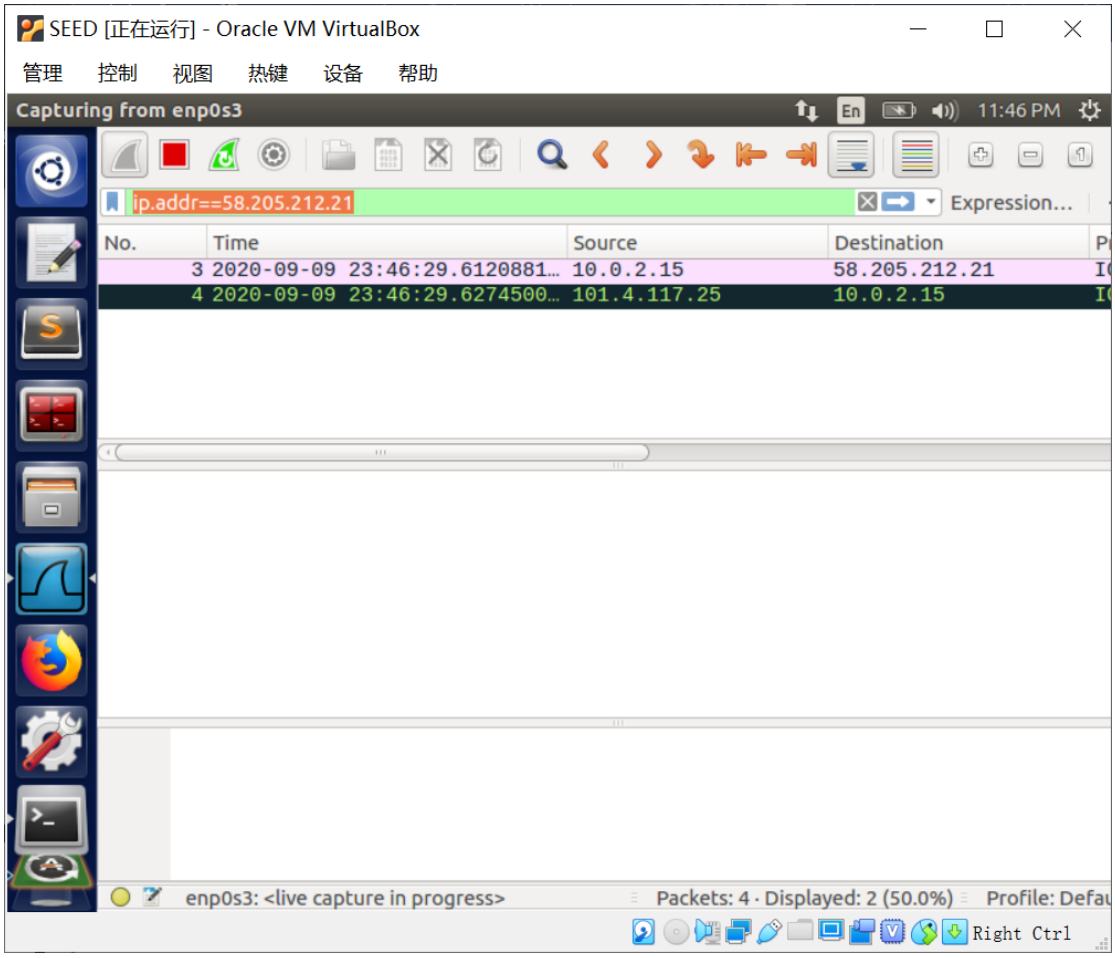
设置 ttl 为 8，到达 211.65.207.77 处出现超时：



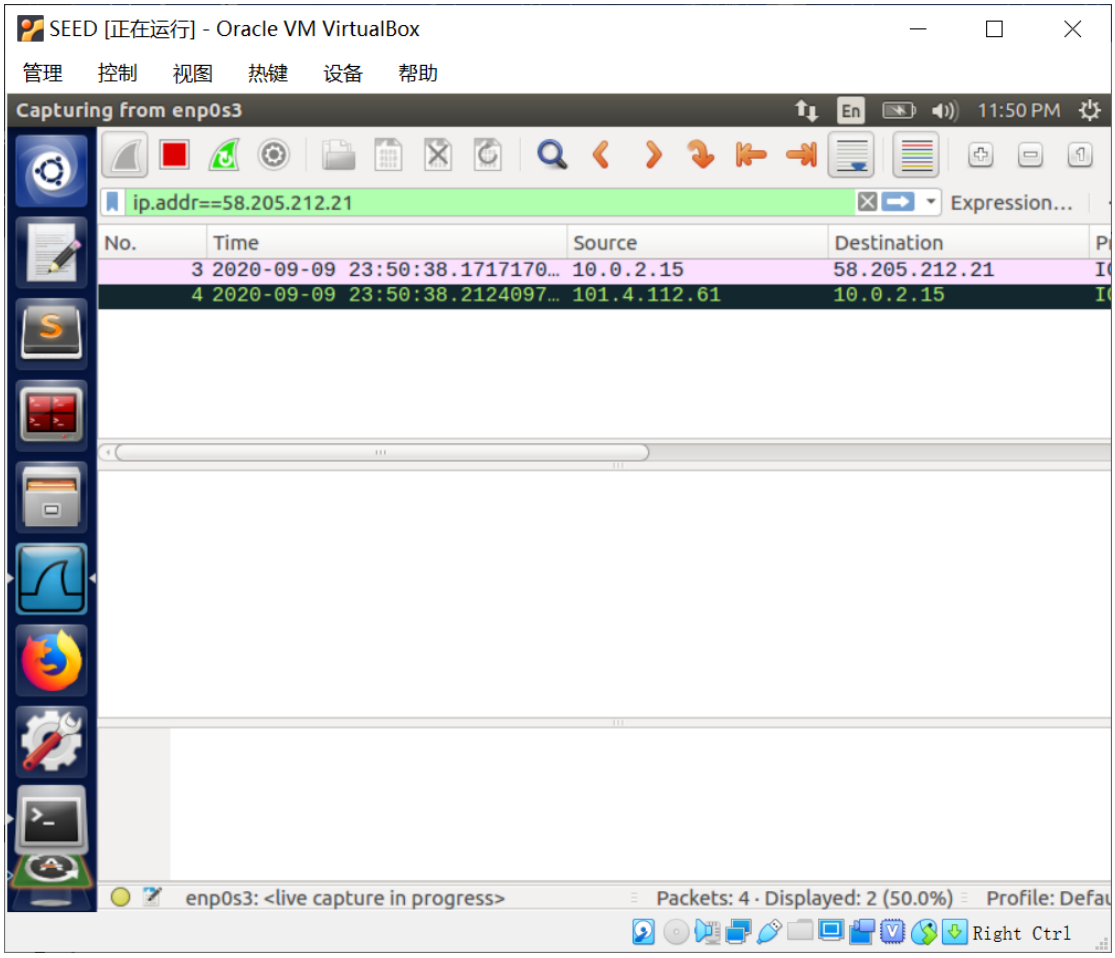
设置 ttl 为 9，达到 101.4.116.105 处出现超时：



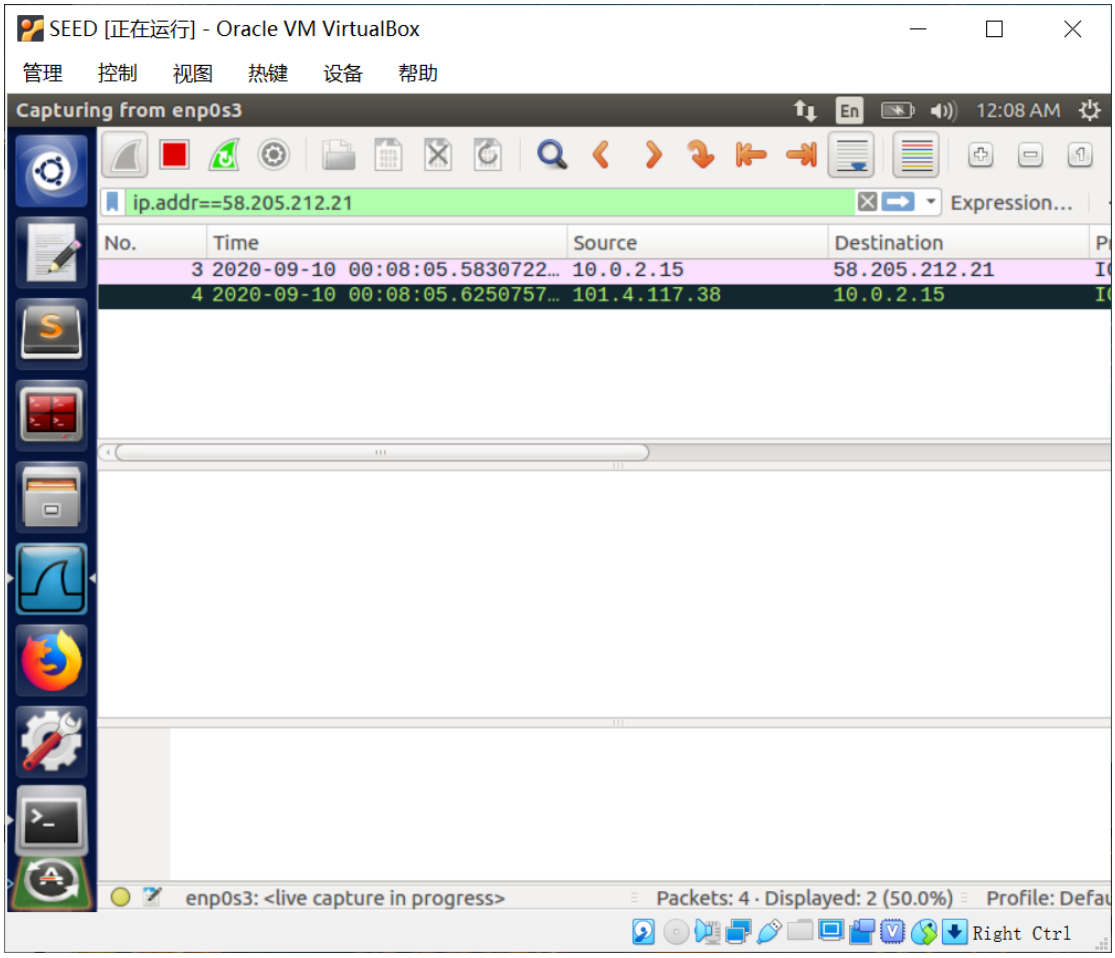
设置 ttl 为 10，到达 101.4.117.25 时出现超时：



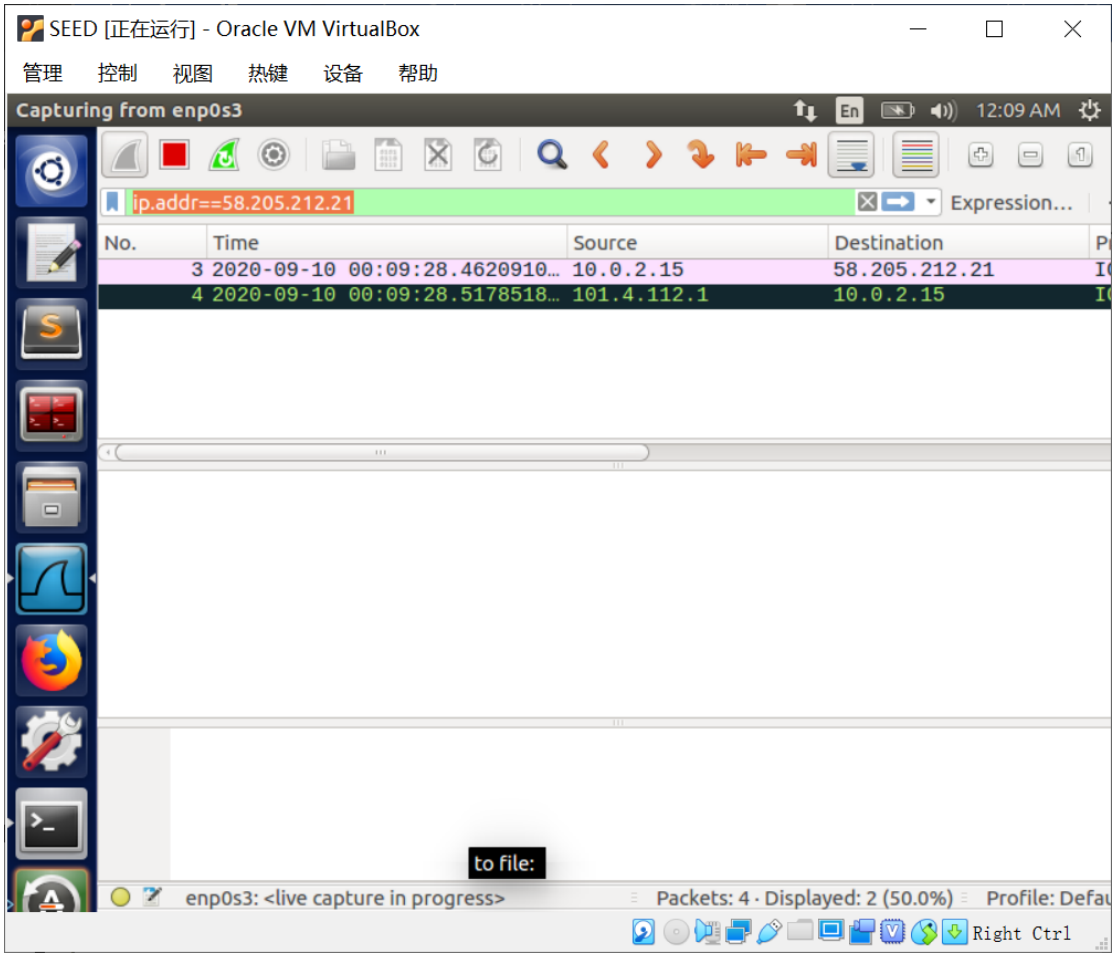
设置 ttl 为 11，到达 101.4.112.61 处出现超时：



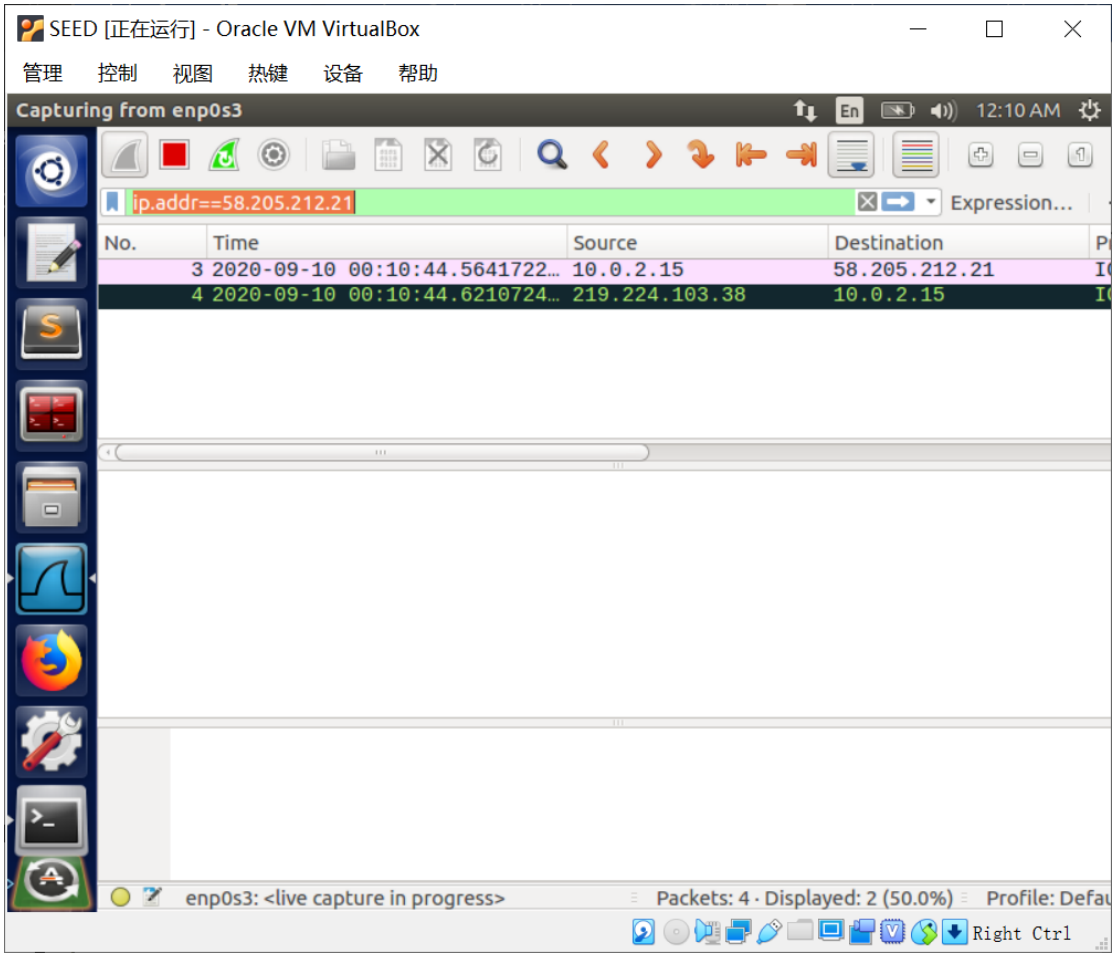
设置 ttl 为 12，可以看到在 101.4.117.38 处超时了：



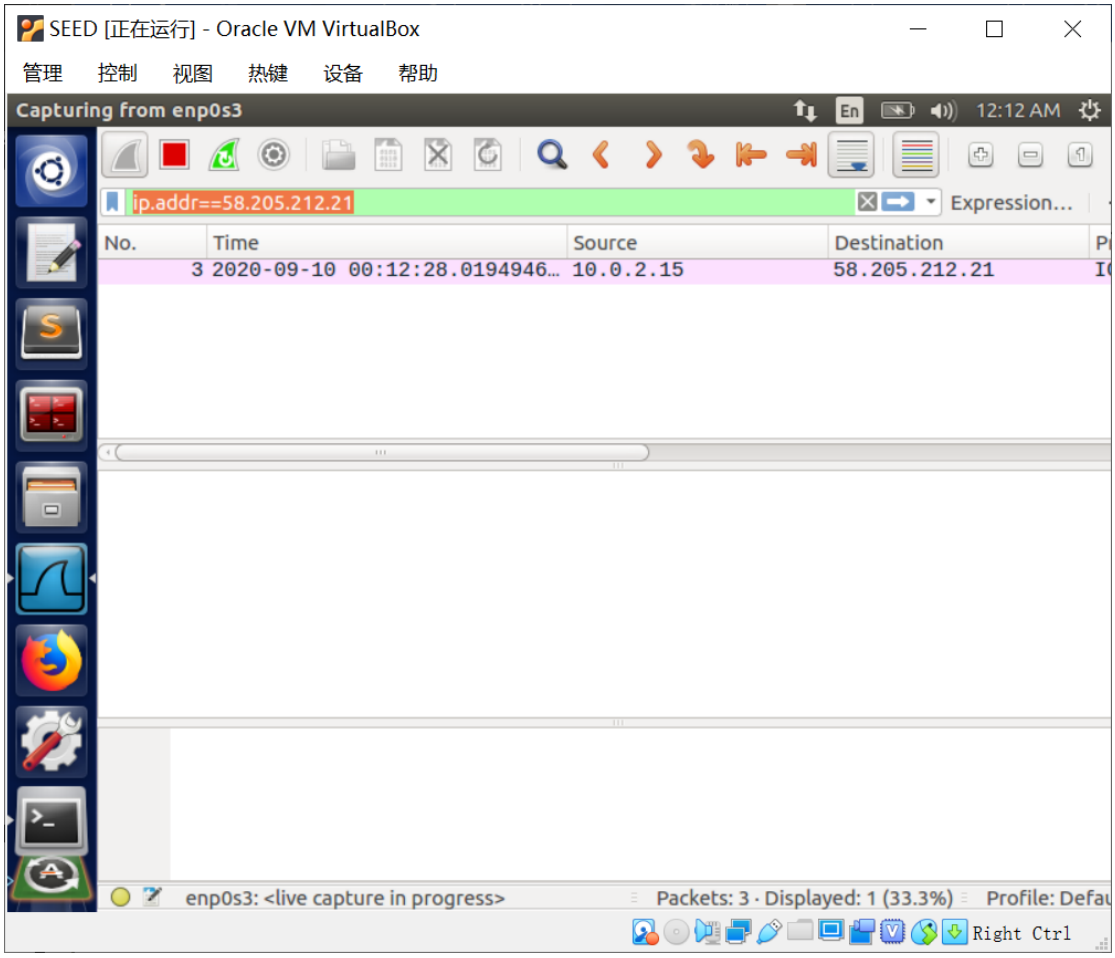
设置 ttl 为 13，可以看到在 101.4.112.1 处超时：



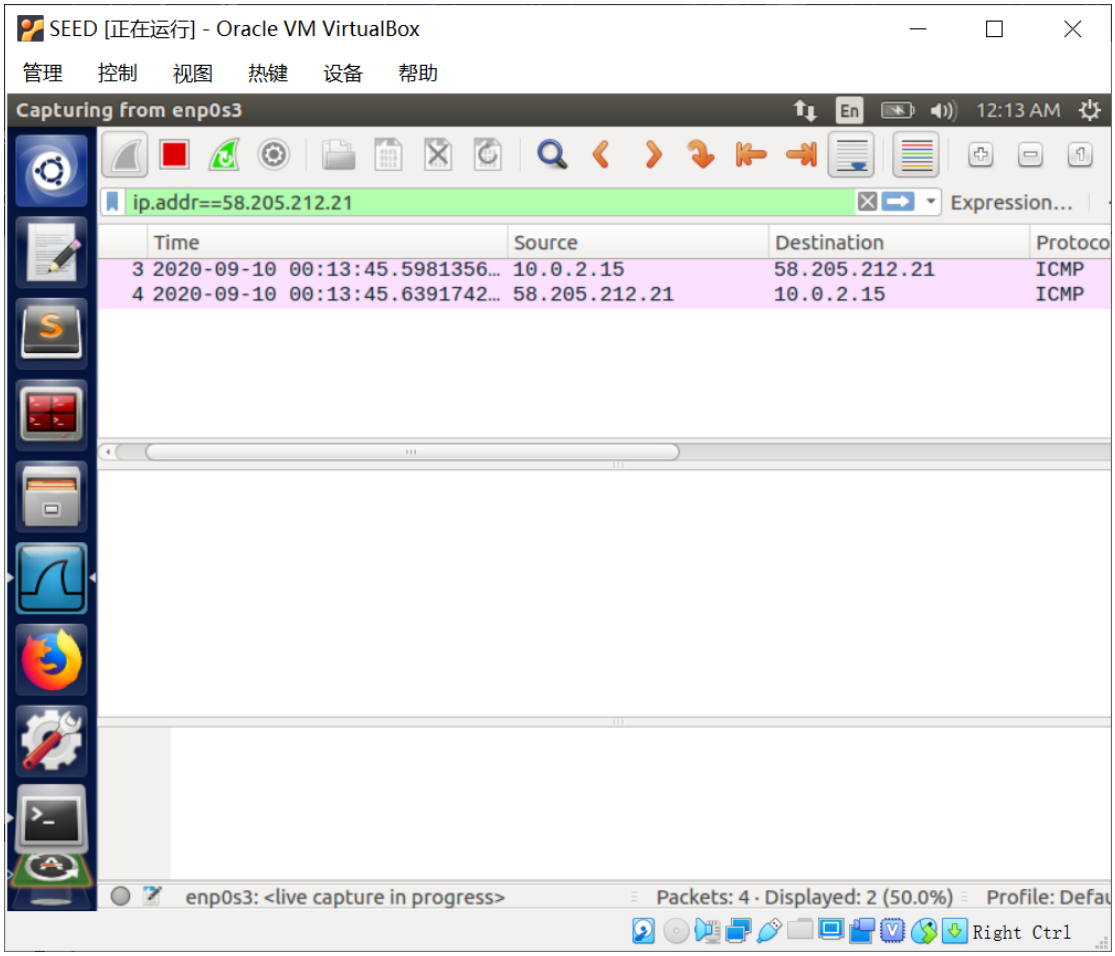
设置 ttl 为 14，可以看到在 219.224.103.38 处超时了：



设置 ttl 为 15，发现丢包了：



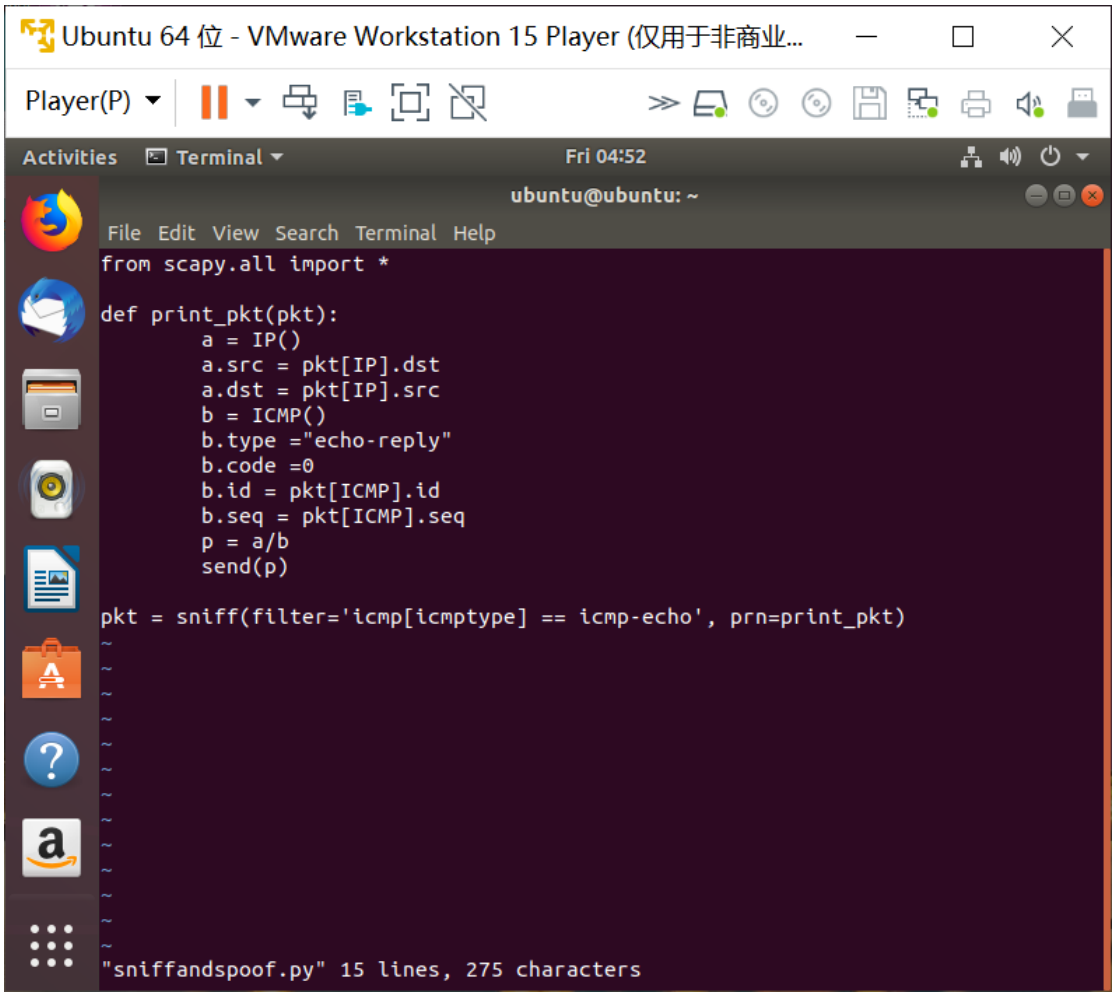
设置 ttl 为 16，看到收到了来自 58.205.212.21 的 icmp 回复报文，说明 icmp 报文到达了该地址：



综上所有的步骤得出：访问 58.205.212.21 地址需要经过 15 个路由器。

Task1.4

编写一个 sniffandspoof 程序，实现既嗅探报文又能发送伪造报文。将所有该局域网内的 icmp-echo 报文都嗅探下来并且根据源宿地址以及报文内容构造出响应报文并发送回请求方。



在一个虚拟机机上跑这个程序，使用该局域网中的另一个虚拟机进行 ping 操作：

```
ubuntu@ubuntu:~$ sudo python3 sniffandspoof.py
```

```
ubuntu-16@ubuntu:~$ ping www.baidu.com
PING www.a.shifen.com (182.61.200.6) 56(84) bytes of data.
64 bytes from 182.61.200.6: icmp_seq=1 ttl=128 time=60.5 ms
8 bytes from 182.61.200.6: icmp_seq=1 ttl=64 (truncated)
8 bytes from 182.61.200.6: icmp_seq=2 ttl=64 (truncated)
64 bytes from 182.61.200.6: icmp_seq=2 ttl=128 time=56.1 ms (DUP!)
8 bytes from 182.61.200.6: icmp_seq=3 ttl=64 (truncated)
64 bytes from 182.61.200.6: icmp_seq=3 ttl=128 time=62.8 ms (DUP!)
64 bytes from 182.61.200.6: icmp_seq=4 ttl=128 time=49.2 ms
8 bytes from 182.61.200.6: icmp_seq=4 ttl=64 (truncated)
8 bytes from 182.61.200.6: icmp_seq=5 ttl=64 (truncated)
64 bytes from 182.61.200.6: icmp_seq=5 ttl=128 time=49.1 ms (DUP!)
^Z
```

可以发现，正常的 ping 操作的 seq 应该是从 1 不断增加的，但是这里每一个 seq 号都出现了两次，这正是因为除了正常 ping 所得到的响应以外，sniffandspoof 函数在捕获 icmp 请求报文时也发出了一个响应。可以看到运行 sniffandspoof 操作时在另一台主机 ping 的时候该函数发出了报文：

```
ubuntu@ubuntu:~$ sudo python3 sniffandspoof.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

Lab2:

Task1:

Task1A:

在 a 和 b 通信后，查看 a 的 arp cache，可以看到 b 的 ip 和 mac 的映射:

```
ubuntu-16@ubuntu:~$ arp -a
? (192.168.37.139) at 00:0c:29:64:de:af [ether] on ens33
? (192.168.37.2) at 00:50:56:fa:e9:e7 [ether] on ens33
? (192.168.37.254) at 00:50:56:f7:78:9e [ether] on ens33
```

此时，我们在 m 中构造 arp 请求报文，将 m 的源 ip 地址构造成为 b 的 ip 地址，m 的 mac 地址还是写作 m 的 mac 地址，用此 arp 请求报文请求获得 a 的 mac 地址:

```
from scapy.all import *
E = Ether()
A = ARP()
A.psrc='192.168.37.139'
A.pdst='192.168.37.146'
A.hwsrc='08:00:27:8e:5a:51'
A.op=1
pkt = E/A
sendp(pkt)
~
```

再次查看发现 a 中 arp 缓存的 b 的 ip 对应的 mac 地址，发现 mac 地址已经被改成了 m 的 mac 地址:

```
? (192.168.37.139) at 08:00:27:8e:5a:51 [ether] on ens33
```

Task1B:
更改 arp 代码，将 op 从 1 改为 2，再次重复上述实验，发现这一次对应的 b 的 mac 地址没有发生改变：

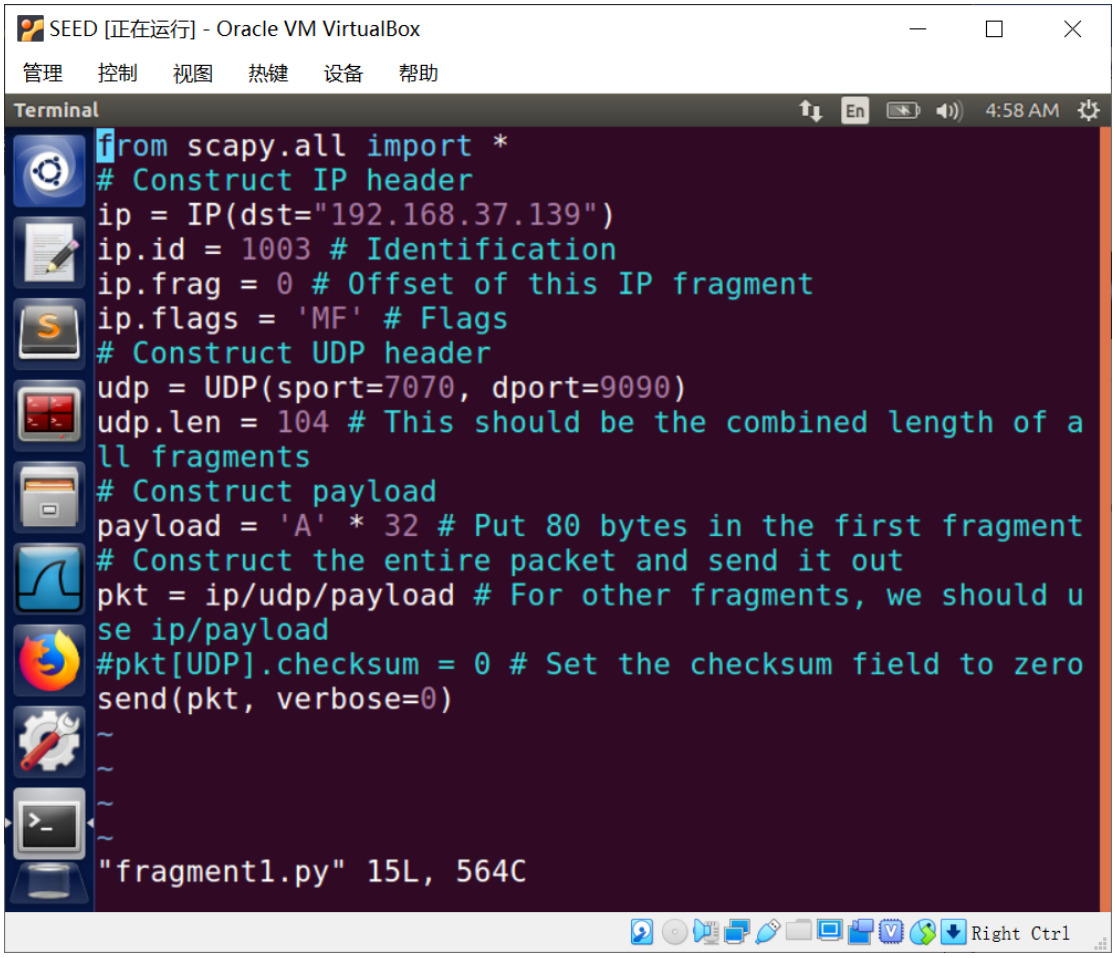
```
? (192.168.37.139) at 00:0c:29:64:de:af [ether] on ens33
```

Task1C:
更改 arp 代码，设置成为 arp gratuitous message，然后在次执行：

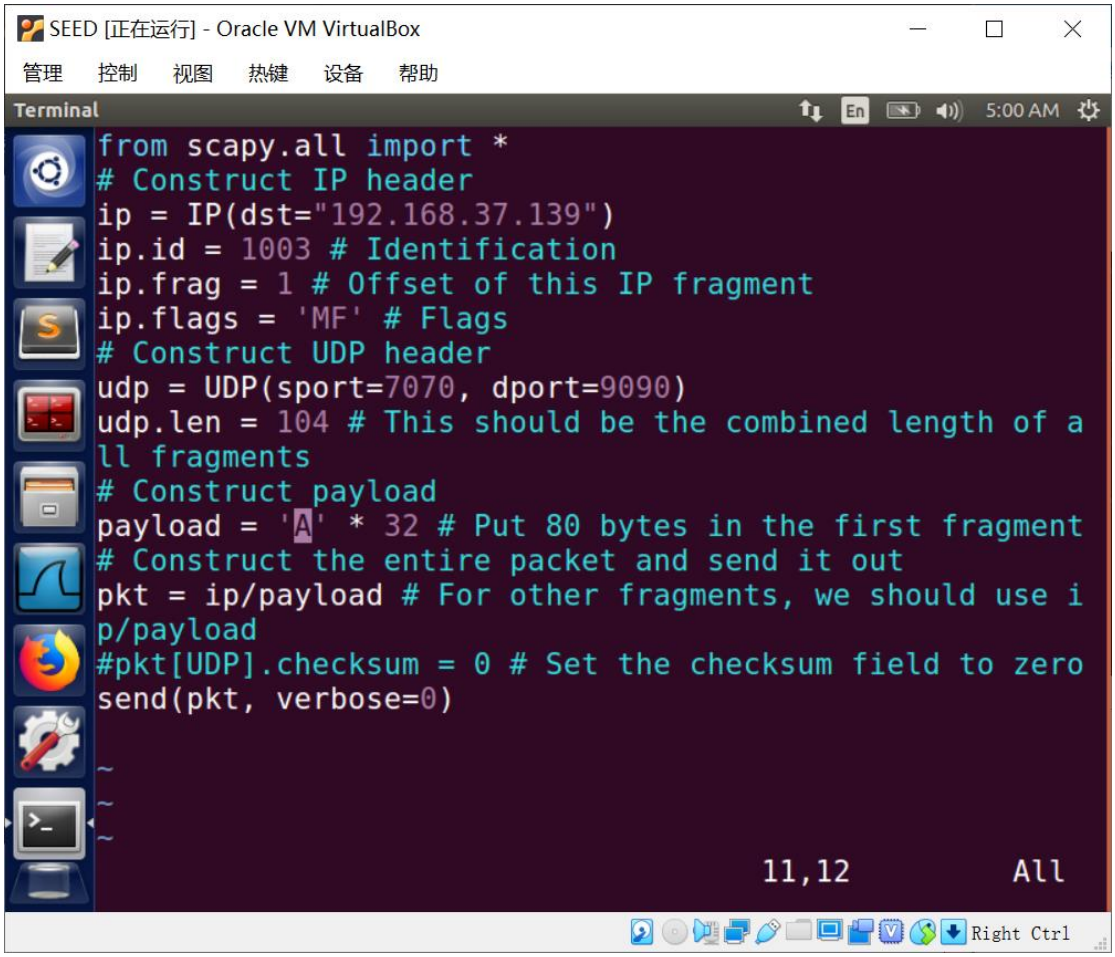
```
from scapy.all import *
E = Ether()
A = ARP()
A.psrc='192.168.37.1'
A.pdst='192.168.37.1'
A.hwsrc='08:00:27:8e:5a:51'
A.hwdst='ff:ff:ff:ff:ff:ff'
A.op=1
pkt = E/A
sendp(pkt)
```

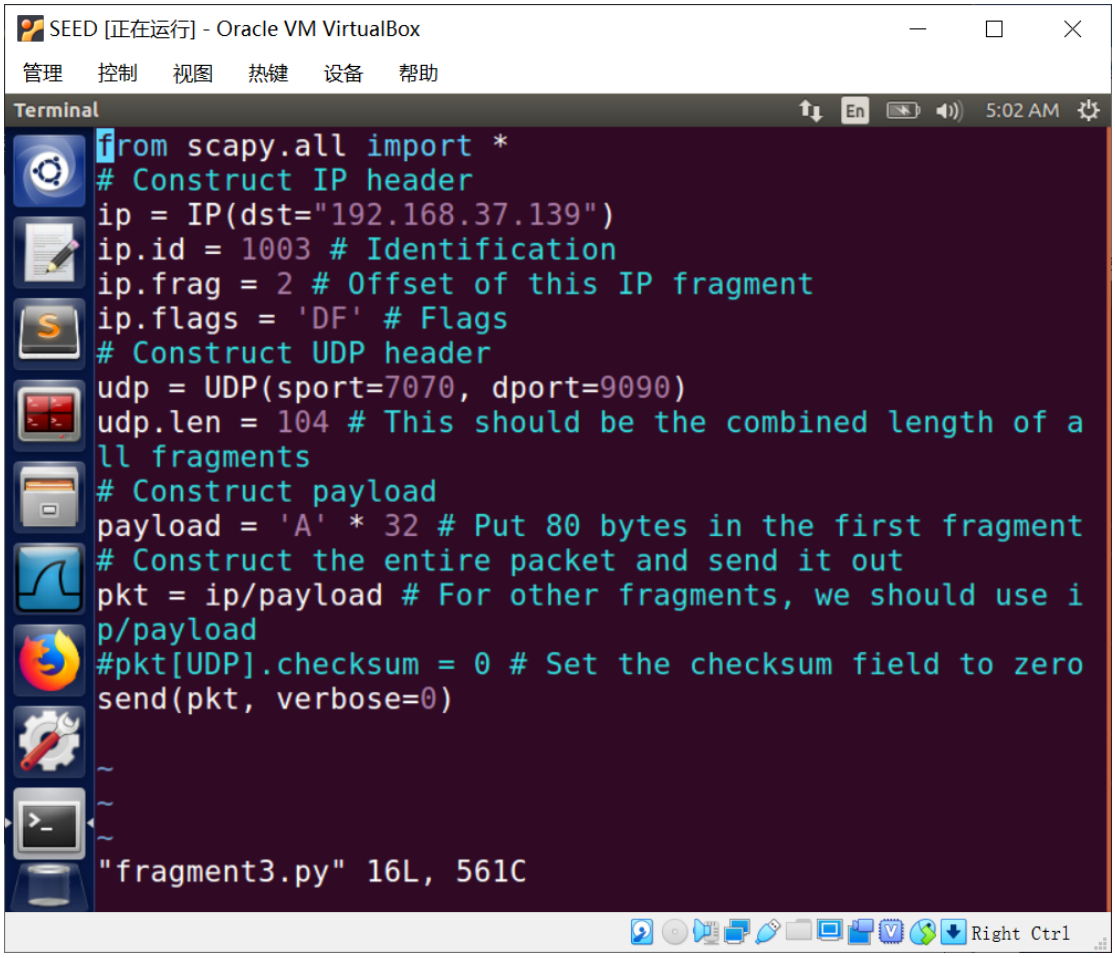
最后发现 b 的 mac 地址还是没有成功被修改。

Lab3
Task1:
Task1.a:
写三个片段的代码分别是：
第一个片段包括 ip 头，udp 头以及负载：



第二、第三个片段只有 ip 头以及负载，即将 pkt 中的/udp 去掉：


SEED [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Terminal
5:00 AM
from scapy.all import *
# Construct IP header
ip = IP(dst="192.168.37.139")
ip.id = 1003 # Identification
ip.frag = 1 # Offset of this IP fragment
ip.flags = 'MF' # Flags
# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len = 104 # This should be the combined length of a
ll fragments
# Construct payload
payload = 'A' * 32 # Put 80 bytes in the first fragment
# Construct the entire packet and send it out
pkt = ip/payload # For other fragments, we should use i
p/payload
#pkt[UDP].checksum = 0 # Set the checksum field to zero
send(pkt, verbose=0)
~
~
~
11,12 All
Right Ctrl



最终在服务器端结果为:

```
ubuntu@ubuntu:~$ nc -lu 9090
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA
```

可以看见服务器端接收到了三个片段并合并成了一个 udp 报文