57117121 聂榕

VPN Tunneling Lab

Task1

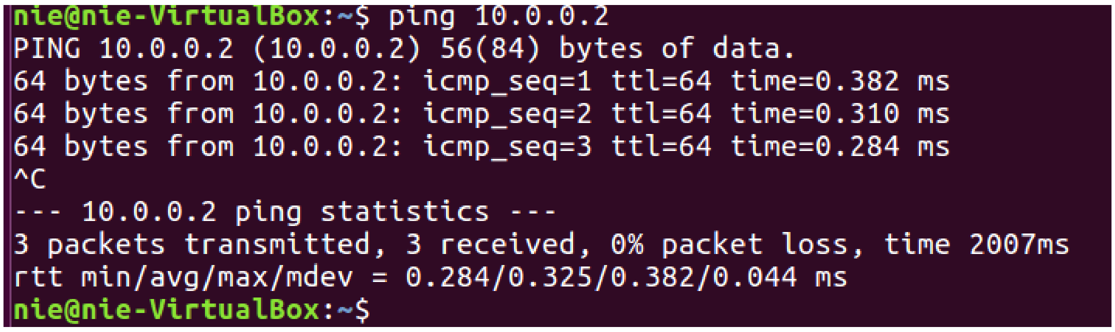
虚拟机 A(seed)一张网卡，桥接至宿主机无线网卡

虚拟机 B（普通 ubuntu）两张网卡，一张桥接至宿主机无线网卡，另一张连接至内部网络（内部网络名称为 intnet）并配置 IP 为 10.0.0.1/24，网关也为 10.0.0.1

虚拟机 C（security onion）一张网卡连接至内部网络（内部网络名称为 intnet）并配置 IP 为 10.0.0.2/24，网关为 10.0.0.1

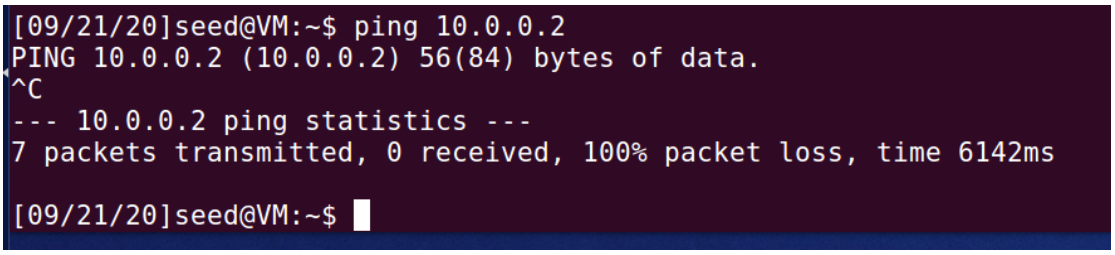
虚拟机 A 和 B 桥接至宿主机的网卡 ip 为自动分配，不过都在 192.168.1.0/24 内。

B ping C:



```
nie@nie-VirtualBox:~$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.382 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.310 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.284 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2007ms
rtt min/avg/max/mdev = 0.284/0.325/0.382/0.044 ms
nie@nie-VirtualBox:~$
```

A ping C



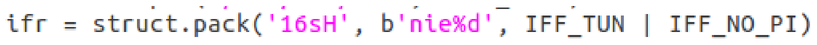
```
[09/21/20]seed@VM:~$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6142ms

[09/21/20]seed@VM:~$
```

Task2

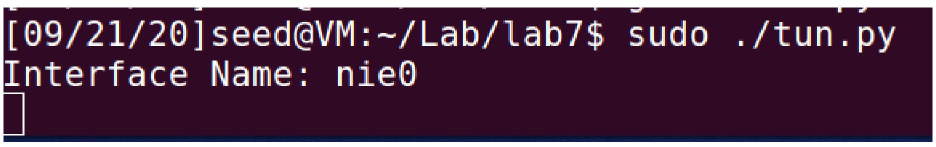
Task2.a

修改示例程序为:

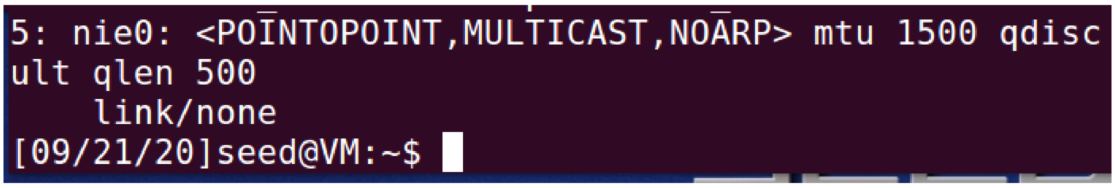


```
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
```

运行后:



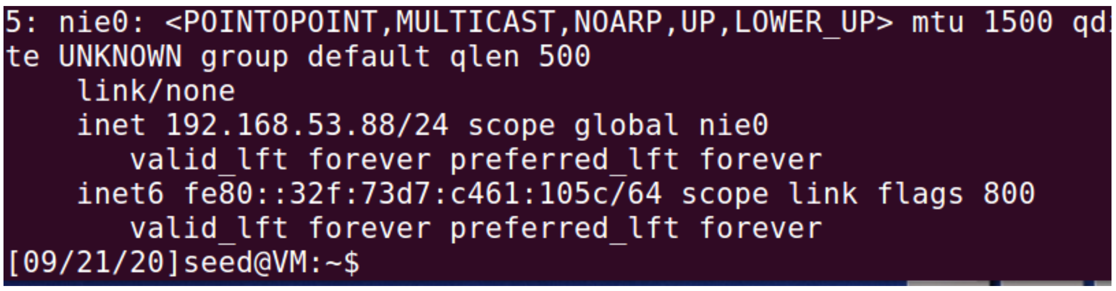
```
[09/21/20]seed@VM:~/Lab/lab7$ sudo ./tun.py
Interface Name: nie0
```



```
5: nie0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc
ult qlen 500
    link/none
[09/21/20]seed@VM:~$
```

Task2.b

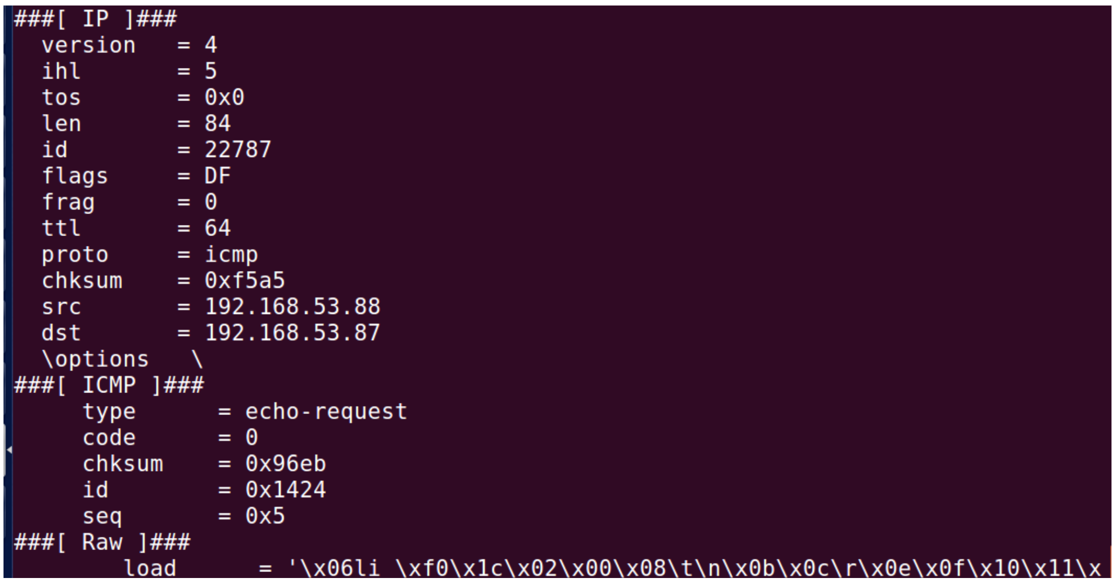
ip 设置为 192.168.153.88/24



```
5: nie0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qd
te UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.88/24 scope global nie0
       valid_lft forever preferred_lft forever
    inet6 fe80::32f:73d7:c461:105c/64 scope link flags 800
       valid_lft forever preferred_lft forever
[09/21/20]seed@VM:~$
```

Task2.c

尝试 ping192.168.53.87：

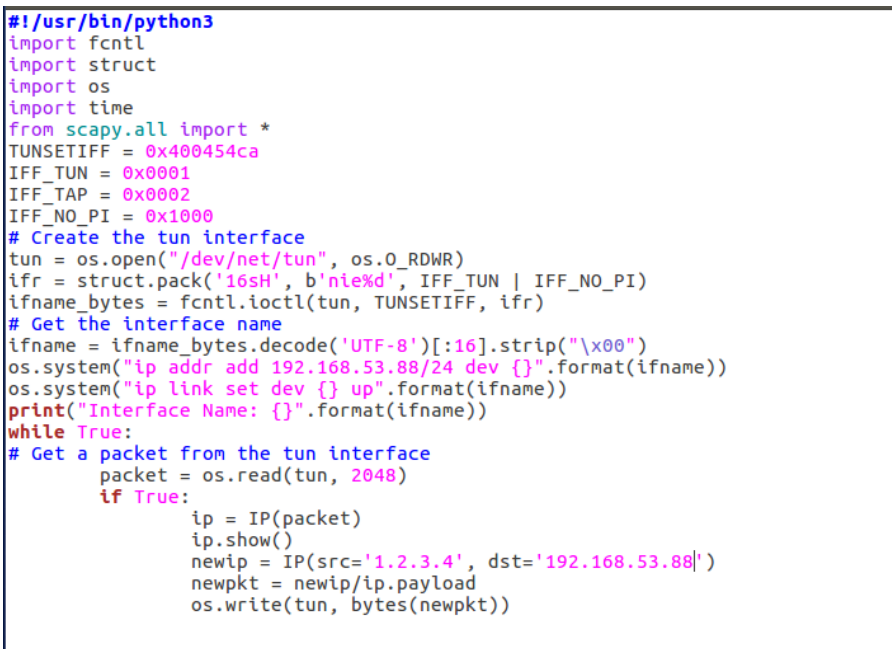


```
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = 84
  id        = 22787
  flags     = DF
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = 0xf5a5
  src       = 192.168.53.88
  dst       = 192.168.53.87
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = 0x96eb
     id        = 0x1424
     seq       = 0x5
###[ Raw ]###
        load      = '\x06li \xf0\x1c\x02\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x
```

尝试 ping 10.0.0.2，什么都没有显示

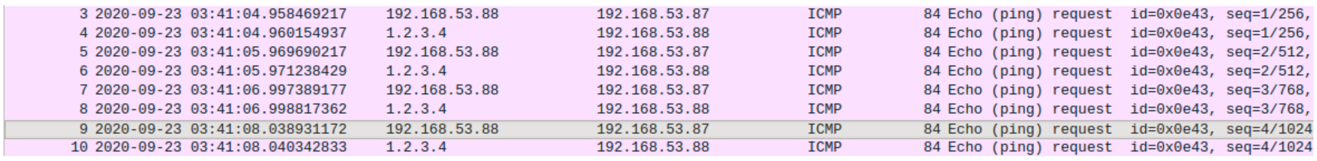
Task2.d

程序如下时：



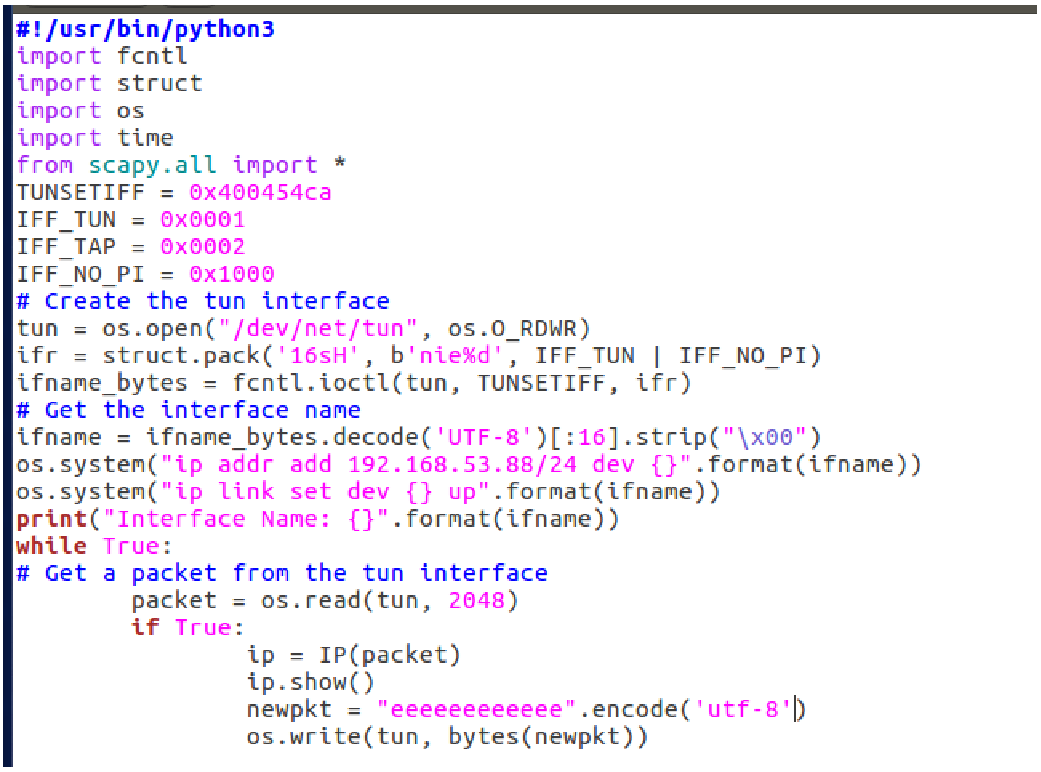
```python
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.88/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
print("Interface Name: {}".format(ifname))
while True:
# Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if True:
                ip = IP(packet)
                ip.show()
                newip = IP(src='1.2.3.4', dst='192.168.53.88')
                newpkt = newip/ip.payload
                os.write(tun, bytes(newpkt))
```

即收到一个包，以 1.2.3.4 为源向 tun（192.168.53.88）发送一个包
终端 ping 192.168.53.87，wireshark 如下：



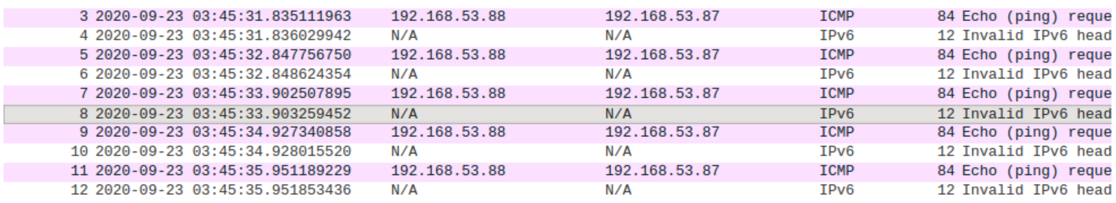
```
 3 2020-09-23 03:41:04.958469217   192.168.53.88   192.168.53.87   ICMP   84 Echo (ping) request  id=0x0e43, seq=1/256,
 4 2020-09-23 03:41:04.960154937   1.2.3.4         192.168.53.88   ICMP   84 Echo (ping) request  id=0x0e43, seq=1/256,
 5 2020-09-23 03:41:05.969690217   192.168.53.88   192.168.53.87   ICMP   84 Echo (ping) request  id=0x0e43, seq=2/512,
 6 2020-09-23 03:41:05.971238429   1.2.3.4         192.168.53.88   ICMP   84 Echo (ping) request  id=0x0e43, seq=2/512,
 7 2020-09-23 03:41:06.997389177   192.168.53.88   192.168.53.87   ICMP   84 Echo (ping) request  id=0x0e43, seq=3/768,
 8 2020-09-23 03:41:06.998817362   1.2.3.4         192.168.53.88   ICMP   84 Echo (ping) request  id=0x0e43, seq=3/768,
 9 2020-09-23 03:41:08.038931172   192.168.53.88   192.168.53.87   ICMP   84 Echo (ping) request  id=0x0e43, seq=4/1024
10 2020-09-23 03:41:08.040342833   1.2.3.4         192.168.53.88   ICMP   84 Echo (ping) request  id=0x0e43, seq=4/1024
```

程序改为如下：

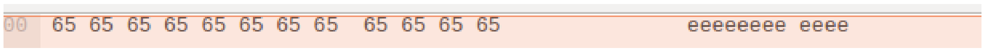


```python
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.88/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
print("Interface Name: {}".format(ifname))
while True:
# Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if True:
                ip = IP(packet)
                ip.show()
                newpkt = "eeeeeeeeeeee".encode('utf-8')
                os.write(tun, bytes(newpkt))
```

即写入 12 个 e
运行程序并 ping192.168.53.87
Wireshark：



```
 3 2020-09-23 03:45:31.835111963   192.168.53.88   192.168.53.87   ICMP   84 Echo (ping) reque
 4 2020-09-23 03:45:31.836029942   N/A             N/A             IPv6   12 Invalid IPv6 head
 5 2020-09-23 03:45:32.847756750   192.168.53.88   192.168.53.87   ICMP   84 Echo (ping) reque
 6 2020-09-23 03:45:32.848624354   N/A             N/A             IPv6   12 Invalid IPv6 head
 7 2020-09-23 03:45:33.902507895   192.168.53.88   192.168.53.87   ICMP   84 Echo (ping) reque
 8 2020-09-23 03:45:33.903259452   N/A             N/A             IPv6   12 Invalid IPv6 head
 9 2020-09-23 03:45:34.927340858   192.168.53.88   192.168.53.87   ICMP   84 Echo (ping) reque
10 2020-09-23 03:45:34.928015520   N/A             N/A             IPv6   12 Invalid IPv6 head
11 2020-09-23 03:45:35.951189229   192.168.53.88   192.168.53.87   ICMP   84 Echo (ping) reque
12 2020-09-23 03:45:35.951853436   N/A             N/A             IPv6   12 Invalid IPv6 head
```

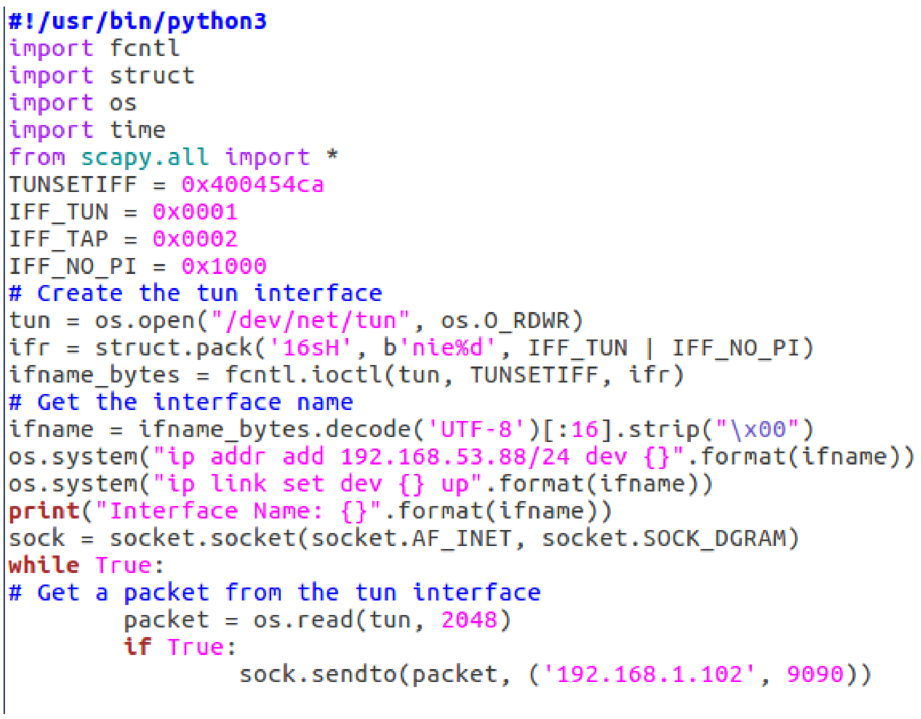


```
00  65 65 65 65 65 65 65 65  65 65 65 65          eeeeeeee eeee
```

可以收到包，还是 12 个 e，没有添加任何头。

Task3

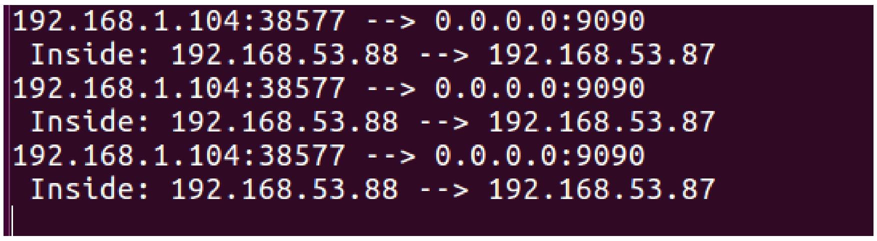
Server 端程序使用实验指导的即可
Client 程序如下：



```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.88/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
print("Interface Name: {}".format(ifname))
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
# Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if True:
                sock.sendto(packet, ('192.168.1.102', 9090))
```

192.168.1.102 为 vpn server 所在虚拟机的对外的网卡地址

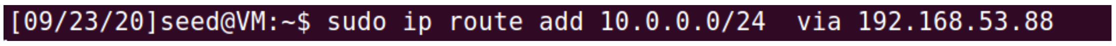
两边执行程序后，在 client 端虚拟机上 ping192.168.53.87，server 端如下：



```
192.168.1.104:38577 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 192.168.53.87
192.168.1.104:38577 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 192.168.53.87
192.168.1.104:38577 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 192.168.53.87
```

报告一开始说明了实验中内网地址分配为 10.0.0.0/24

所以在 client 中添加路由表项：

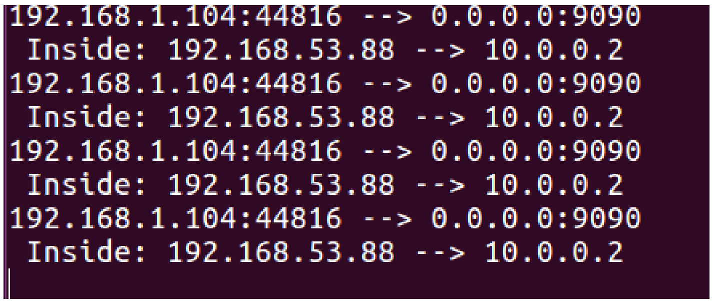


```
[09/23/20]seed@VM:~$ sudo ip route add 10.0.0.0/24  via 192.168.53.88
```

192.168.53.88 即为 tun 卡的 ip

再 ping10.0.0.2

server 端如下：



```
192.168.1.104:44816 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 10.0.0.2
192.168.1.104:44816 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 10.0.0.2
192.168.1.104:44816 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 10.0.0.2
192.168.1.104:44816 --> 0.0.0.0:9090
 Inside: 192.168.53.88 --> 10.0.0.2
```

成功收到

Task4：

因为以之前的三台虚拟机的配置（与实验指导的有区别）总是无法成功，在没有头绪的情况下选择了完全按照实验指导手册重新设置三台虚拟机。

新的配置如下

三台 seed

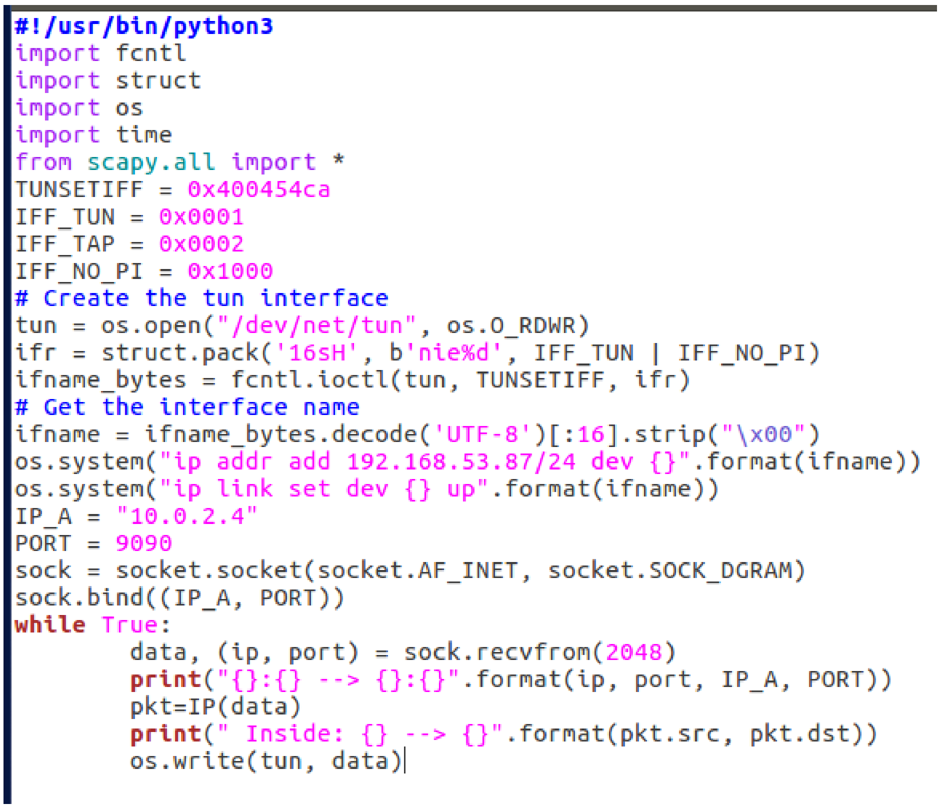
seed 与 seed3（Host V）连接至同一内部网络，ip 分别为 192.168.60.1/24，192.168.60.2/24

seed 与 seed2 (Host U) 连接至同一 NAT 网络 (模拟外网)，ip 分别为 10.0.2.4/24，10.0.2.7/24

然后就是最关键的，关于 seed2 和 seed 上面的两张 tun 网卡，经过一些小实验之后我得到了如下条件，即这两张网卡需要在同一网段内且不能同 ip 实验才能成功，即 seed 上的 tun 卡会把包传给内核，内核再传至连接内部网络的网卡再传至另一主机。主要是对 tun 卡的工作原理不太清楚，如果同 ip 的话似乎包根本没有经过我们的程序（程序中的输出部分根本没有输出），也就是在更下层就处理掉了，如果不是同网段倒是经过我们的 tun 程序，但不会传给内核（似乎是）。以上是一些猜想。

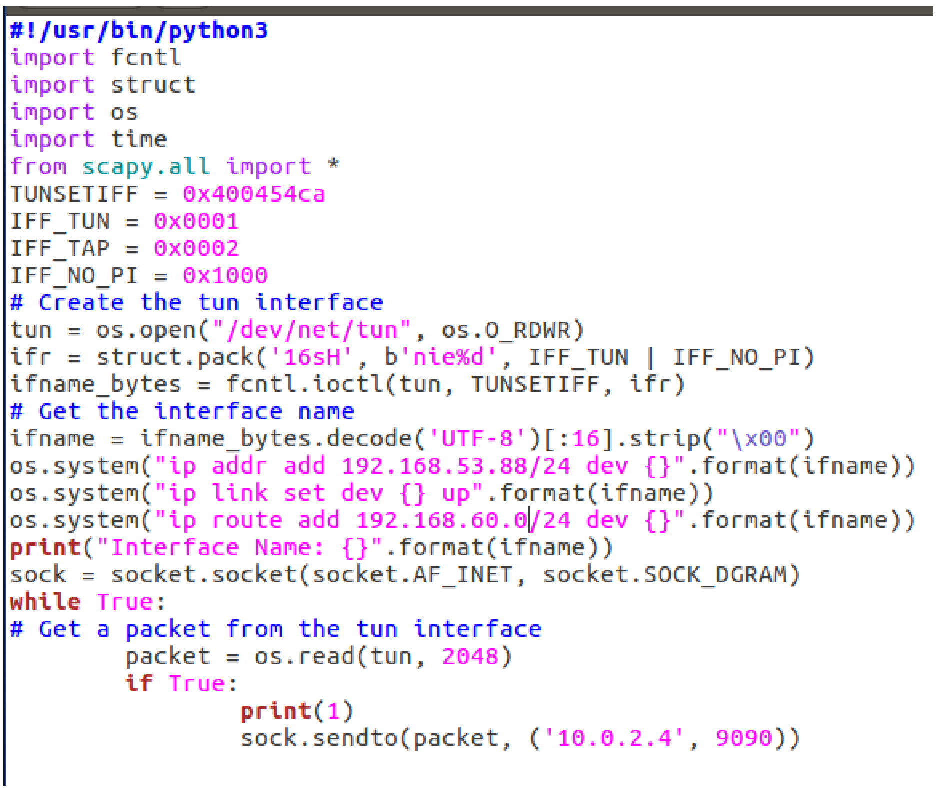
seed 上的 tun 卡 ip 为 192.168.53.87/24，seed2 上的 tun 卡 ip 为.88/24

tun_server.py 如下：



```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.87/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
IP_A = "10.0.2.4"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
while True:
        data, (ip, port) = sock.recvfrom(2048)
        print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
        pkt=IP(data)
        print(" Inside: {} --> {}".format(pkt.src, pkt.dst))
        os.write(tun, data)
```

tun.py（seed2 上的客户端程序）如下：

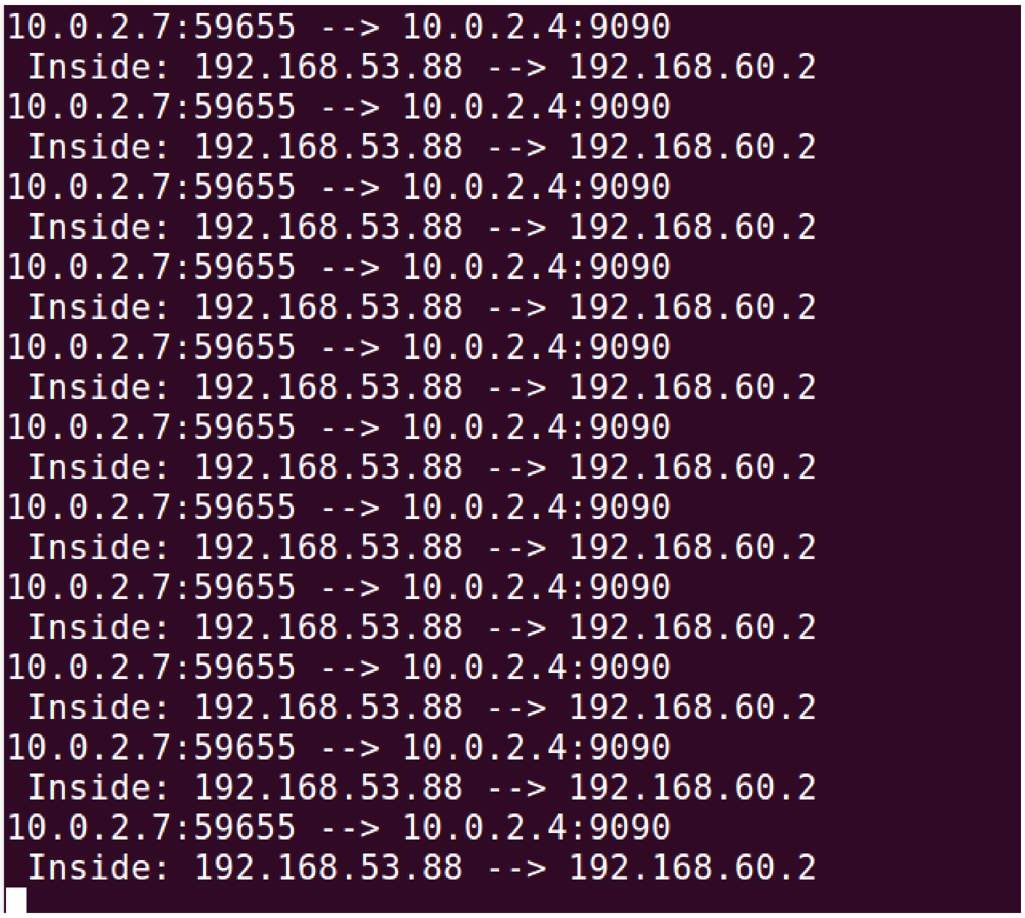


```
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.88/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))
print("Interface Name: {}".format(ifname))
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
while True:
# Get a packet from the tun interface
        packet = os.read(tun, 2048)
        if True:
                print(1)
                sock.sendto(packet, ('10.0.2.4', 9090))
```

把配置路由表的语句也写入程序，这样就不用再输命令配置了。

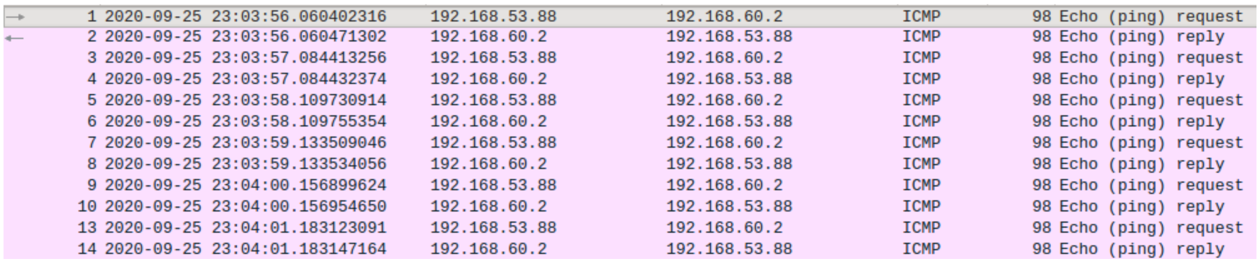
在 seed2 （Host U）ping seed3（Host V）后

Seed 上：



```
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
10.0.2.7:59655 --> 10.0.2.4:9090
 Inside: 192.168.53.88 --> 192.168.60.2
```

Host V 上：

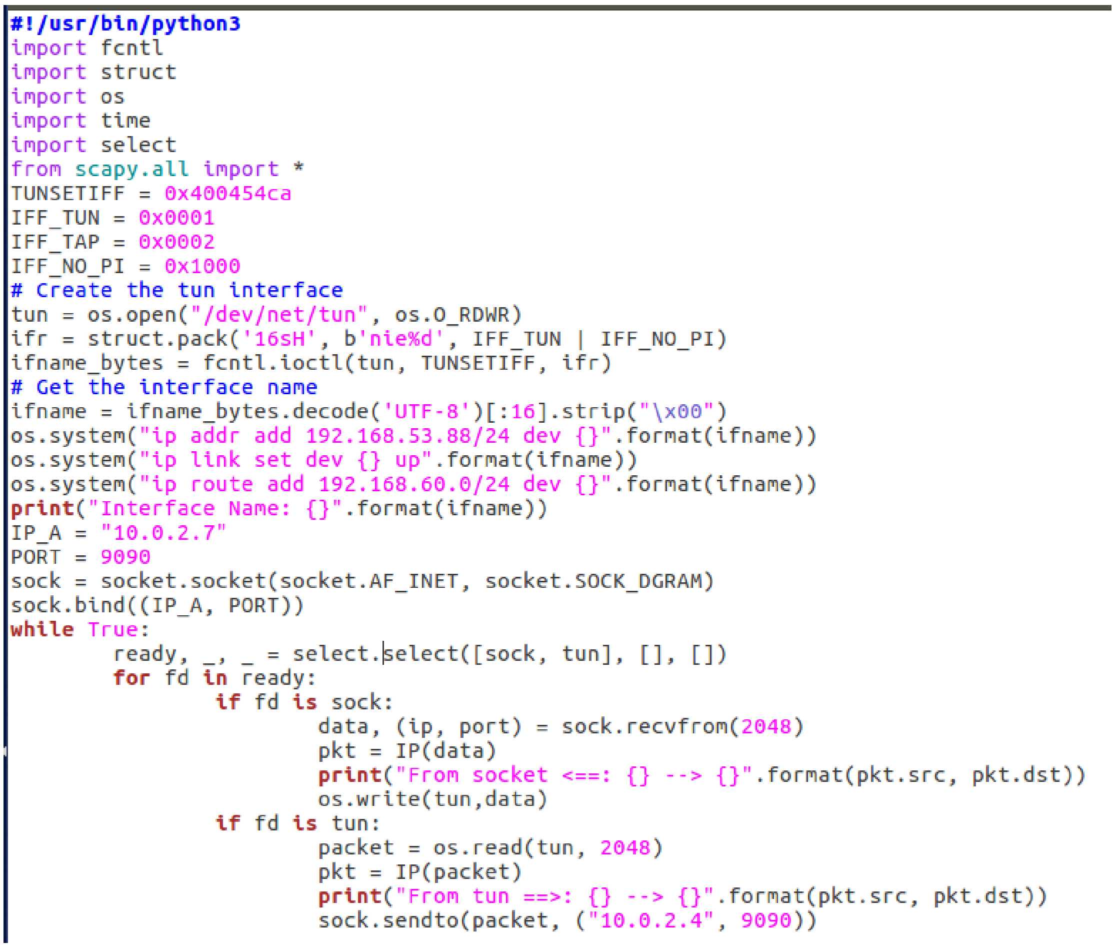


```
 1 2020-09-25 23:03:56.060402316   192.168.53.88   192.168.60.2    ICMP   98 Echo (ping) request
 2 2020-09-25 23:03:56.060471302   192.168.60.2    192.168.53.88   ICMP   98 Echo (ping) reply
 3 2020-09-25 23:03:57.084413256   192.168.53.88   192.168.60.2    ICMP   98 Echo (ping) request
 4 2020-09-25 23:03:57.084432374   192.168.60.2    192.168.53.88   ICMP   98 Echo (ping) reply
 5 2020-09-25 23:03:58.109730914   192.168.53.88   192.168.60.2    ICMP   98 Echo (ping) request
 6 2020-09-25 23:03:58.109755354   192.168.60.2    192.168.53.88   ICMP   98 Echo (ping) reply
 7 2020-09-25 23:03:59.133509046   192.168.53.88   192.168.60.2    ICMP   98 Echo (ping) request
 8 2020-09-25 23:03:59.133534056   192.168.60.2    192.168.53.88   ICMP   98 Echo (ping) reply
 9 2020-09-25 23:04:00.156899624   192.168.53.88   192.168.60.2    ICMP   98 Echo (ping) request
10 2020-09-25 23:04:00.156954650   192.168.60.2    192.168.53.88   ICMP   98 Echo (ping) reply
13 2020-09-25 23:04:01.183123091   192.168.53.88   192.168.60.2    ICMP   98 Echo (ping) request
14 2020-09-25 23:04:01.183147164   192.168.60.2    192.168.53.88   ICMP   98 Echo (ping) reply
```

看到了 U（53.88）传至 V（60.2）的包

Task5

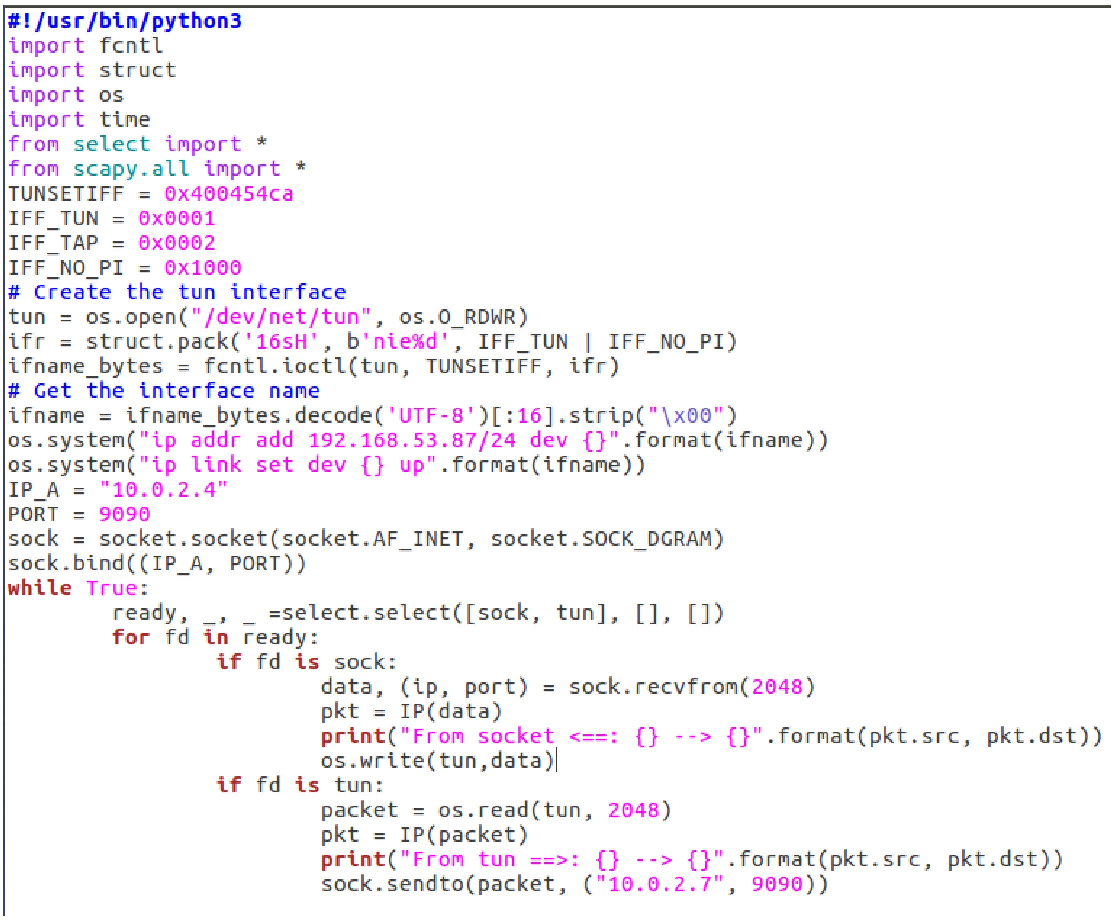
tun_client.py 如下：



```python
#!/usr/bin/python3
import fcntl
import struct
import os
import time
import select
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.88/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
os.system("ip route add 192.168.60.0/24 dev {}".format(ifname))
print("Interface Name: {}".format(ifname))
IP_A = "10.0.2.7"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
while True:
        ready, _, _ = select.select([sock, tun], [], [])
        for fd in ready:
                if fd is sock:
                        data, (ip, port) = sock.recvfrom(2048)
                        pkt = IP(data)
                        print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
                        os.write(tun,data)
                if fd is tun:
                        packet = os.read(tun, 2048)
                        pkt = IP(packet)
                        print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
                        sock.sendto(packet, ("10.0.2.4", 9090))
```

即在 host U 上也创建一个绑定到 9090 端口上的 udp 服务以接收 host V 产生的经由 server 到达 U 的包。

tun_server.py 如下：

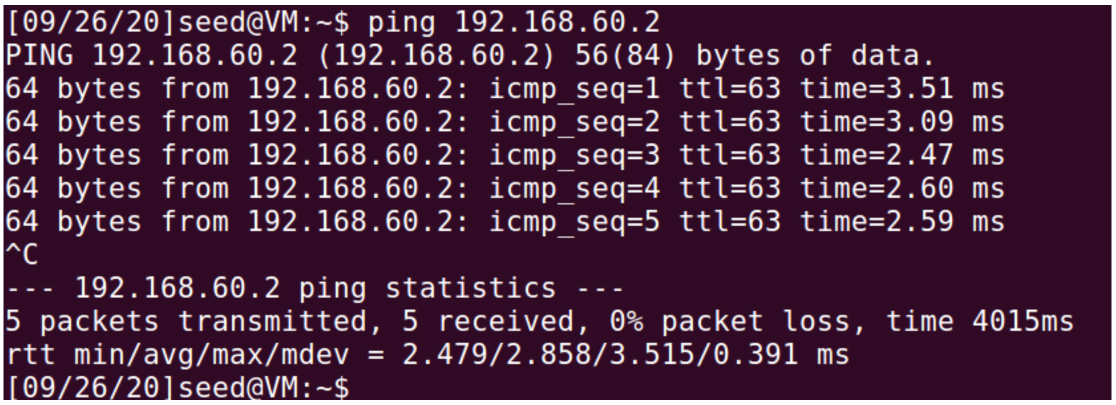


```python
#!/usr/bin/python3
import fcntl
import struct
import os
import time
from select import *
from scapy.all import *
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'nie%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
os.system("ip addr add 192.168.53.87/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
IP_A = "10.0.2.4"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
while True:
        ready, _, _ =select.select([sock, tun], [], [])
        for fd in ready:
                if fd is sock:
                        data, (ip, port) = sock.recvfrom(2048)
                        pkt = IP(data)
                        print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
                        os.write(tun,data)
                if fd is tun:
                        packet = os.read(tun, 2048)
                        pkt = IP(packet)
                        print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
                        sock.sendto(packet, ("10.0.2.7", 9090))
```

收到 tun 卡的包后将其通过 socket 传到 U 的 9090 端口。

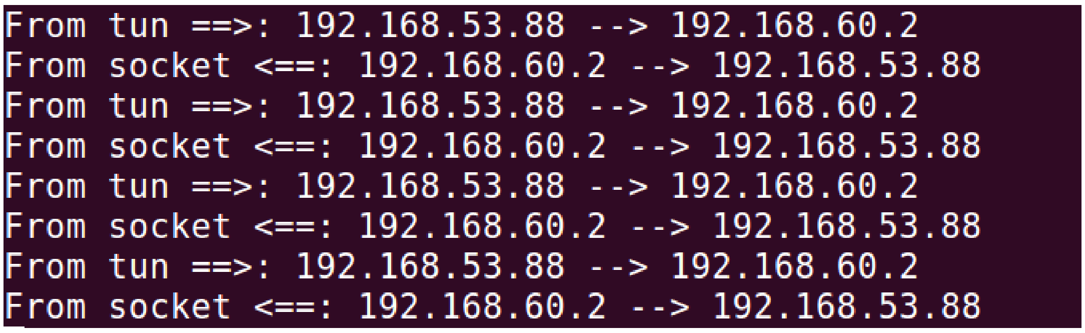
因为创建 tun 卡后系统会默认添加到 192.168.53.0/24 的路由（通过 tun 卡发送），所以不用有添加路由的语句。

从 U ping V：

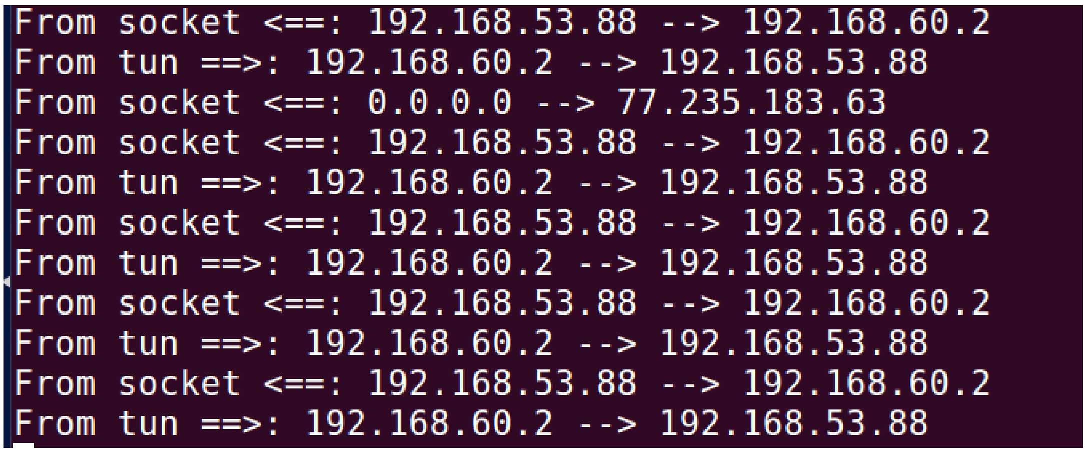


```
[09/26/20]seed@VM:~$ ping 192.168.60.2
PING 192.168.60.2 (192.168.60.2) 56(84) bytes of data.
64 bytes from 192.168.60.2: icmp_seq=1 ttl=63 time=3.51 ms
64 bytes from 192.168.60.2: icmp_seq=2 ttl=63 time=3.09 ms
64 bytes from 192.168.60.2: icmp_seq=3 ttl=63 time=2.47 ms
64 bytes from 192.168.60.2: icmp_seq=4 ttl=63 time=2.60 ms
64 bytes from 192.168.60.2: icmp_seq=5 ttl=63 time=2.59 ms
^C
--- 192.168.60.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4015ms
rtt min/avg/max/mdev = 2.479/2.858/3.515/0.391 ms
[09/26/20]seed@VM:~$
```

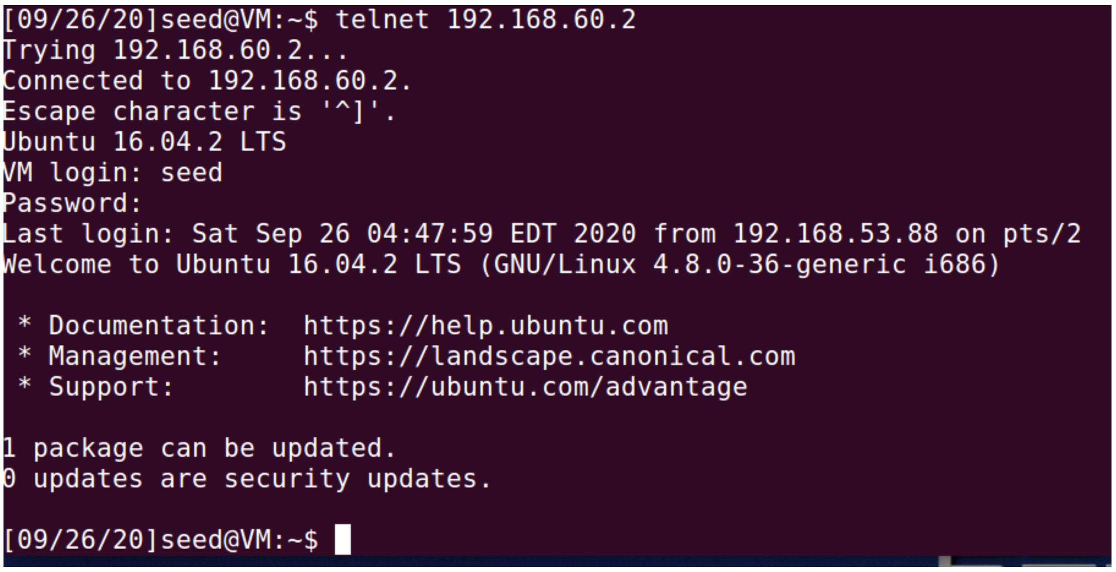
tun_client.py:


```
From tun ==>: 192.168.53.88 --> 192.168.60.2
From socket <==: 192.168.60.2 --> 192.168.53.88
From tun ==>: 192.168.53.88 --> 192.168.60.2
From socket <==: 192.168.60.2 --> 192.168.53.88
From tun ==>: 192.168.53.88 --> 192.168.60.2
From socket <==: 192.168.60.2 --> 192.168.53.88
From tun ==>: 192.168.53.88 --> 192.168.60.2
From socket <==: 192.168.60.2 --> 192.168.53.88
```

tun_server.py:


```
From socket <==: 192.168.53.88 --> 192.168.60.2
From tun ==>: 192.168.60.2 --> 192.168.53.88
From socket <==: 0.0.0.0 --> 77.235.183.63
From socket <==: 192.168.53.88 --> 192.168.60.2
From tun ==>: 192.168.60.2 --> 192.168.53.88
From socket <==: 192.168.53.88 --> 192.168.60.2
From tun ==>: 192.168.60.2 --> 192.168.53.88
From socket <==: 192.168.53.88 --> 192.168.60.2
From tun ==>: 192.168.60.2 --> 192.168.53.88
From socket <==: 192.168.53.88 --> 192.168.60.2
From tun ==>: 192.168.60.2 --> 192.168.53.88
```

telnet:


```
[09/26/20]seed@VM:~$ telnet 192.168.60.2
Trying 192.168.60.2...
Connected to 192.168.60.2.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sat Sep 26 04:47:59 EDT 2020 from 192.168.53.88 on pts/2
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

[09/26/20]seed@VM:~$ 
```

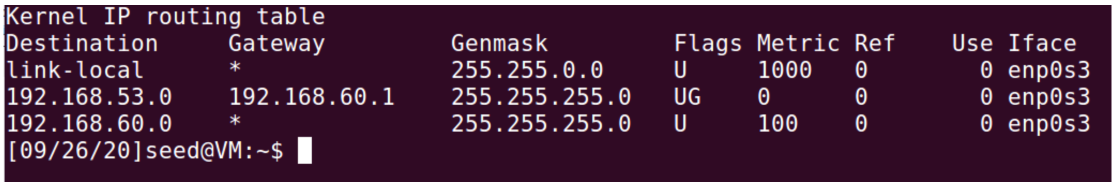
Task6

如果停止 tun_client.py，通过已连接的 telnet 输入一串“zzzzzz”，什么都不会发生，然后再启动 tun_client.py，这一串 z 会同时跳出来。

应该是 telnet 所在进程的 socket 一直在利用 tcp 特性向 tun 重发，未超时前 client 程序启动，那么这些重发就被 U 的 tun 卡接收到，然后继续进行。

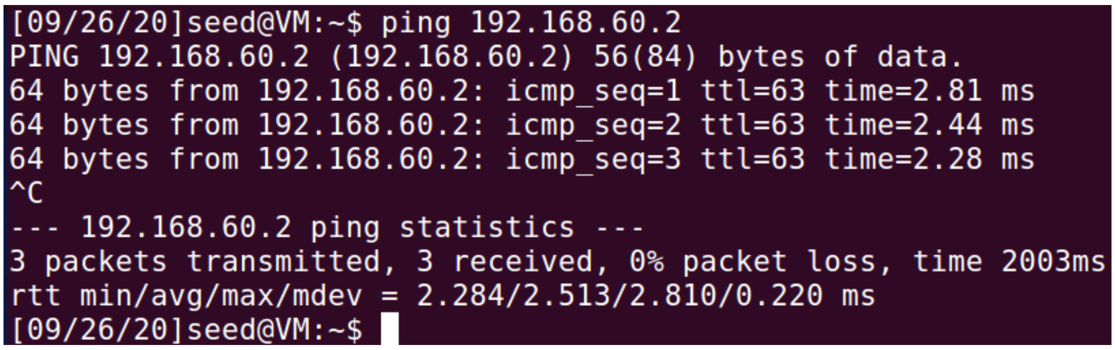
Task7

将 V 的路由表改为如下：



```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
link-local      *               255.255.0.0     U     1000   0        0 enp0s3
192.168.53.0    192.168.60.1    255.255.255.0   UG    0      0        0 enp0s3
192.168.60.0    *               255.255.255.0   U     100    0        0 enp0s3
[09/26/20]seed@VM:~$ 
```

U 仍然可以 ping 通 V：



```
[09/26/20]seed@VM:~$ ping 192.168.60.2
PING 192.168.60.2 (192.168.60.2) 56(84) bytes of data.
64 bytes from 192.168.60.2: icmp_seq=1 ttl=63 time=2.81 ms
64 bytes from 192.168.60.2: icmp_seq=2 ttl=63 time=2.44 ms
64 bytes from 192.168.60.2: icmp_seq=3 ttl=63 time=2.28 ms
^C
--- 192.168.60.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 2.284/2.513/2.810/0.220 ms
[09/26/20]seed@VM:~$ 
```

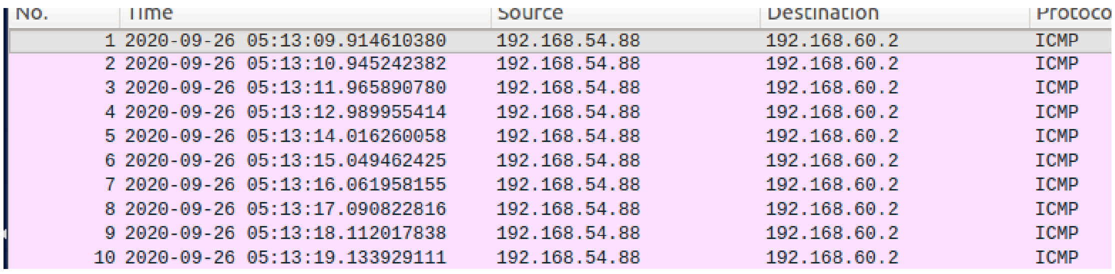
Task8

这个问题就对应了上面 Task4 中我遇到的问题。

将 client 中的 tun 卡的 ip 改为 192.168.54.88/24（原来两张卡都是 192.168.53.0/24 网段），再尝试 ping V

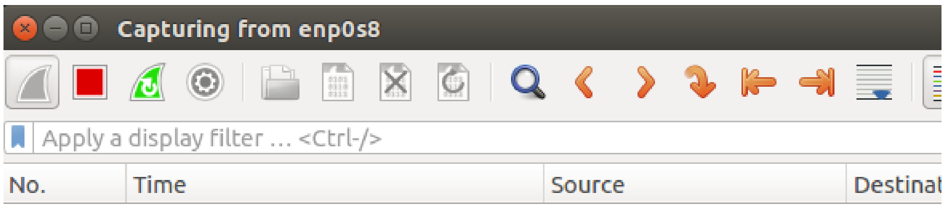
可以看到：

Vpn server 的 tun 卡：



No.	Time	Source	Destination	Protoco
1	2020-09-26 05:13:09.914610380	192.168.54.88	192.168.60.2	ICMP
2	2020-09-26 05:13:10.945242382	192.168.54.88	192.168.60.2	ICMP
3	2020-09-26 05:13:11.965890780	192.168.54.88	192.168.60.2	ICMP
4	2020-09-26 05:13:12.989955414	192.168.54.88	192.168.60.2	ICMP
5	2020-09-26 05:13:14.016260058	192.168.54.88	192.168.60.2	ICMP
6	2020-09-26 05:13:15.049462425	192.168.54.88	192.168.60.2	ICMP
7	2020-09-26 05:13:16.061958155	192.168.54.88	192.168.60.2	ICMP
8	2020-09-26 05:13:17.090822816	192.168.54.88	192.168.60.2	ICMP
9	2020-09-26 05:13:18.112017838	192.168.54.88	192.168.60.2	ICMP
10	2020-09-26 05:13:19.133929111	192.168.54.88	192.168.60.2	ICMP

Vpn server 连接内网的卡：



Capturing from enp0s8

Apply a display filter … <Ctrl-/>

No.	Time	Source	Destinat

什么也没有。

说明包没有被连接内网的卡拿到，也就无法传给 V

　　根据反向路径过滤机制的介绍，内核会检查一个包其 ip 源地址在路由表中是如何路由

的，如果该源地址或源地址所在网段对于收到该包的卡是可达的，那么该包被保留，进行正常操作，若该 ip 源地址所在网段是收到该包的网卡不可达的，那么包就被丢弃

Write 函数是将包经由 tun 卡发给内核的。内核收到一个源为 192.168.54.88，目的为 192.168.60.2 的包，该包是 tun 卡收到的。源地址网段为 192.168.54.0/24，在路由表中，没有关于该网段任何项，那么毫无疑问会被丢弃，因为该网段 tun 卡不可达。那么我们应该设置路由 192.168.54.0/24 通过 tun 传输。

而这个路由又恰恰是 V 的回复包应该经过的路由。所以这一个路由命令兼顾了来回。

这么设置以后，确实可以 ping 通了

内核的该功能使用如下命令关闭：

sudo sysctl net.ipv4.conf.default.rp_filter=0

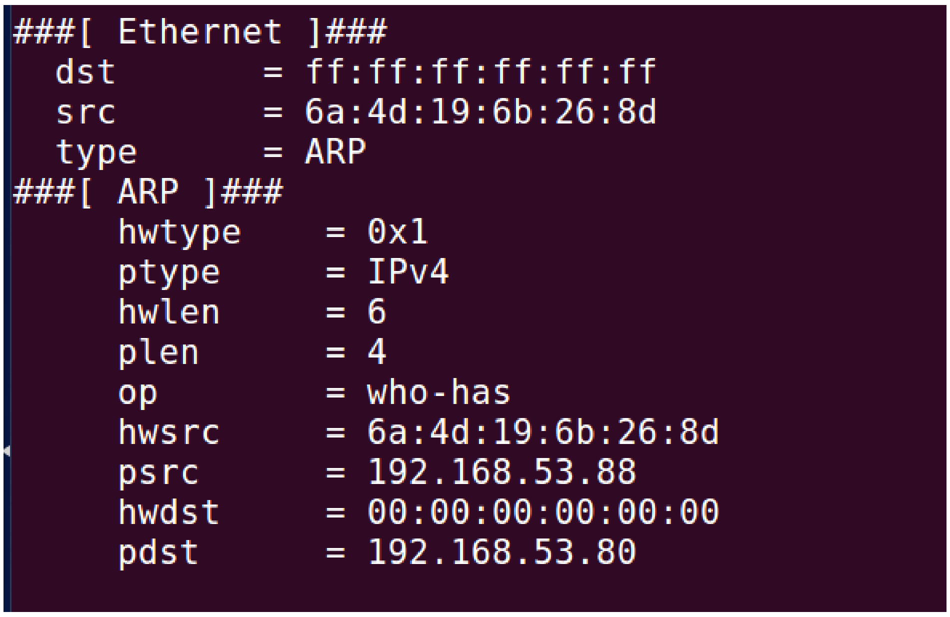
但是因为我们总要配置 192.168.54.0/24 的路由通过 tun 传输，这本来就说是回送的方向，而我们上面又看到了这个参数不设置为 0 也能 ping 通，所以就为了 ping 通 V 来说的话该参数设置与否没什么关系。但是，理论上讲，在该参数设置为 0 且没有配置路由的情况下，U ping V 的 icmp request 包应该被发向 V，只不过回复无法被正确转发。但是事实上是requst 还是无法发给 V。所以说是否设置该参数没有什么不同。当然，因为该参数是网上找到的，所以到底这个是不是控制反向过滤的参数，可能是有疑点的。

但如果该参数是控制过滤机制的参数，那么这里可以说是一个疑问点，目前尚不清楚。

Task9

尝试 ping 192.168.53.80

可以看到接收到的包是 arp 包：



```
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 6a:4d:19:6b:26:8d
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = 6a:4d:19:6b:26:8d
     psrc      = 192.168.53.88
     hwdst     = 00:00:00:00:00:00
     pdst      = 192.168.53.80
```

程序中改为 IP(packet)，解码出来的包也都是错的，ipversion 都是 15

说明 tun 卡接收到包时以太帧头部已经没有了。

但 tap 卡收到的包数据中以太帧头部分还在