Lab 7

57117221 姚远

VPN Tunneling Lab

Task1 Network Setup

Host U：10.0.2.7

VPN Server：10.0.2.6, 192.168.70.1

Host V：192.168.70.101

主机U连接VPN服务器：

```
[09/23/20]seed@VM:~$ ping 10.0.2.6
PING 10.0.2.6 (10.0.2.6) 56(84) bytes of data.
64 bytes from 10.0.2.6: icmp_seq=1 ttl=64 time=0.495 ms
64 bytes from 10.0.2.6: icmp_seq=2 ttl=64 time=0.483 ms
64 bytes from 10.0.2.6: icmp_seq=3 ttl=64 time=0.540 ms
```

连接成功。

VPN服务器连接主机V：

```
[09/23/20]seed@VM:~$ ping 192.168.70.101
PING 192.168.70.101 (192.168.70.101) 56(84) bytes of data.
64 bytes from 192.168.70.101: icmp_seq=1 ttl=64 time=0.575 ms
64 bytes from 192.168.70.101: icmp_seq=2 ttl=64 time=0.727 ms
64 bytes from 192.168.70.101: icmp_seq=3 ttl=64 time=0.559 ms
```

连接成功。

主机U连接主机V：

```
[09/23/20]seed@VM:~$ ping 192.168.70.101
PING 192.168.70.101 (192.168.70.101) 56(84) bytes of data.
^C
--- 192.168.70.101 ping statistics ---
9 packets transmitted, 0 received, 100% packet loss, time 8169ms
```

无法连接。

Task2 Create and Configure TUN Interface

Task2.a Name of the Interface

在主机U中运行tun.py：

```
[09/23/20]seed@VM:~$ vi tun.py
[09/23/20]seed@VM:~$ chmod a+x tun.py
[09/23/20]seed@VM:~$ sudo ./tun.py
Interface Name: tun0
```

在另一个终端输入ip address，看到新增了tun0：

```
[09/23/20]seed@VM:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOW
N group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_
fast state UP group default qlen 1000
    link/ether 08:00:27:5e:12:9d brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.7/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 550sec preferred_lft 550sec
    inet6 fe80::830e:8f20:9fb4:f8d2/64 scope link
       valid_lft forever preferred_lft forever
3: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state D
OWN group default qlen 500
    link/none
```

修改tun.py将接口名改为lzzk：

```
# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'lzzk%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun, TUNSETIFF, ifr)
```

Task2.b Set up the TUN Interface

按照题目中代码编写后，运行结果：

```
[09/23/20]seed@VM:~$ vi tun.py
[09/23/20]seed@VM:~$ chmod a+x tun.py
[09/23/20]seed@VM:~$ sudo ./tun.py
Interface Name: lzzk0
```

Lzzk0前的数字由4变成5Lzzk0的状态Flag多出了UP和LOWER_UPLzzk0有了inet的IP

Task2.c Read from the TUN Interface

修改代码后在主机另一个shell中：

```
[09/23/20]seed@VM:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNO
N group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_
fast state UP group default qlen 1000
    link/ether 08:00:27:5e:12:9d brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.7/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 444sec preferred_lft 444sec
    inet6 fe80::830e:8f20:9fb4:f8d2/64 scope link
       valid_lft forever preferred_lft forever
4: lzzk0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state
DOWN group default qlen 500
    link/none
```

查看到了ICMP报文，因为lzzk0在该子网中，所以ping命令就是用了lzzk0网口发出命令。

Task2.d Write to the TUN Interface

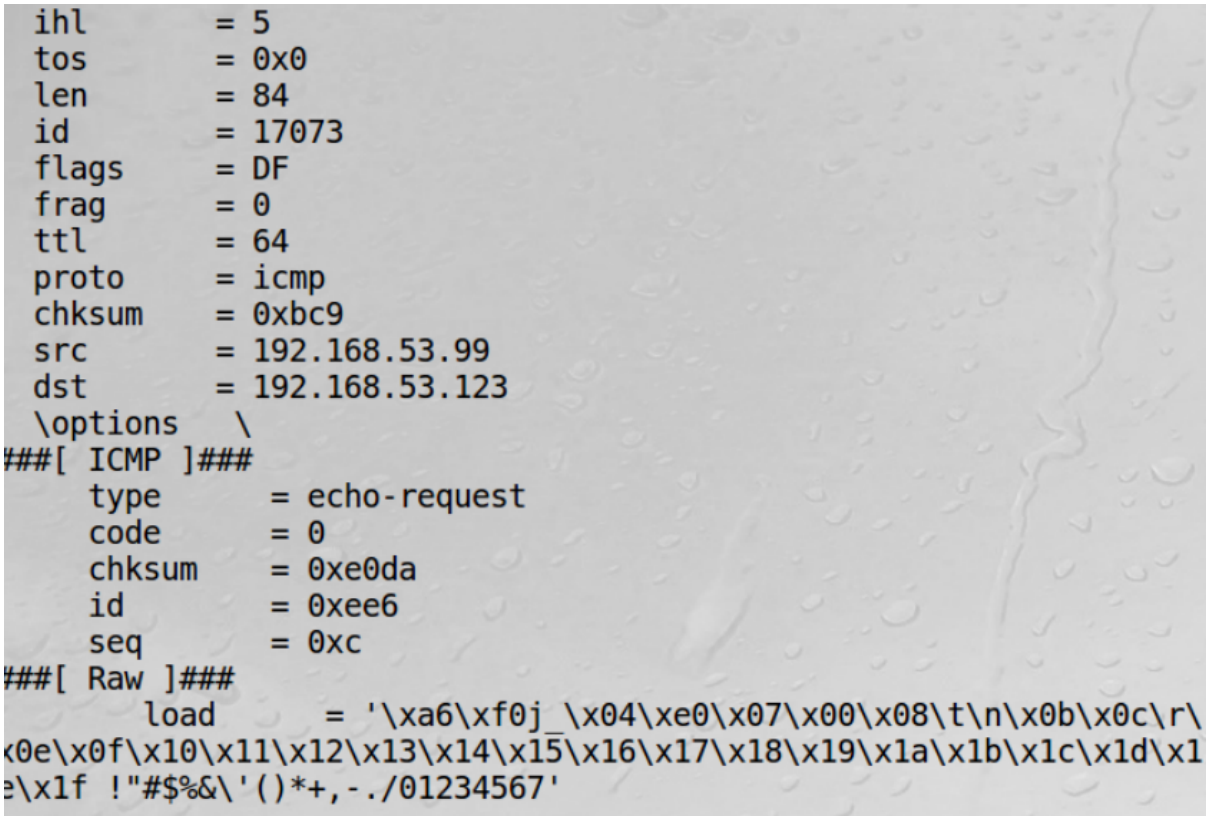
在结尾增加发送伪造数据包的代码：

```
[09/23/20]seed@VM:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOW
N group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_
fast state UP group default qlen 1000
    link/ether 08:00:27:5e:12:9d brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.7/24 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 532sec preferred_lft 532sec
    inet6 fe80::830e:8f20:9fb4:f8d2/64 scope link
       valid_lft forever preferred_lft forever
5: lzzk0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc
 pfifo_fast state UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.99/24 scope global lzzk0
       valid_lft forever preferred_lft forever
    inet6 fe80::16a6:bf3b:19cb:99c1/64 scope link flags 800
       valid_lft forever preferred_lft forever
```

再次运行ping 192.168.53.123会发现Wireshark不仅捕捉到lzzk0发往目标的ICMP报文，还捕捉到tun.py构造的源地址为1.2.3.4的ICMP报文，说明tun.py可以成功创建和发送报文。

```
[09/23/20]seed@VM:~$ ping 192.168.53.123
PING 192.168.53.123 (192.168.53.123) 56(84) bytes of data.
```

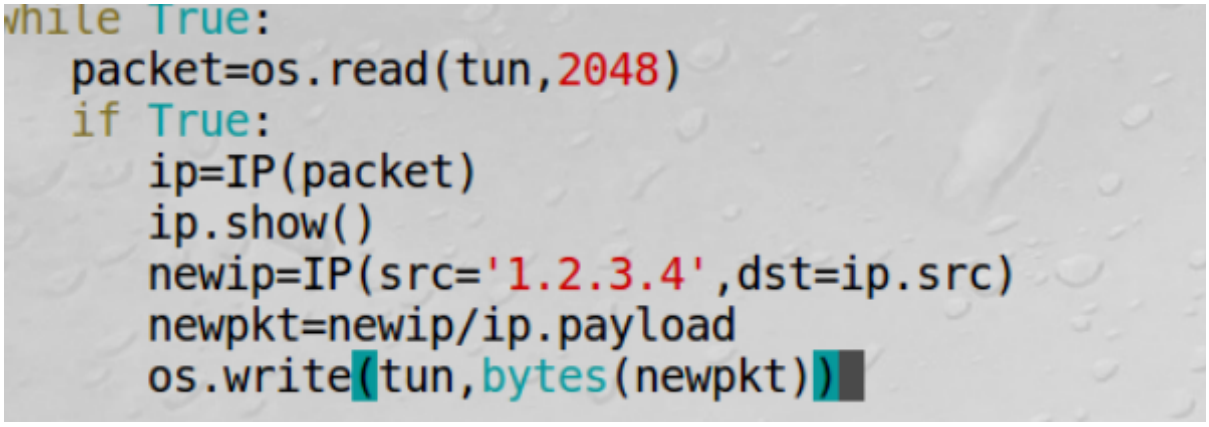
将发送到tun的数据改为hahaha，重复上述操作：



Wireshark捕捉到一些无意义的数据包。

Task3 Send the IP Packet to VPN Server Through a Tunnel

在VPN服务器上运行tun_server.py在主机U上运行tun_client.py在主机U上ping 192.168.56.123



可以看到VPN服务器上打印出了接受到的数据包信息，由于主机U和VPN服务器通过

10.0.2.0/24网段相连，因而外层UDP数据包的源IP地址为10.0.2.7，由于Ping的是192.168.53.0/24网段的IP地址，所以payload中的IP数据包为虚拟端口lzzk0的IP地址192.168.53.99。综上，实现了隧道的作用。在主机U上添加下图路由，实现将发往192.168.70.0/24网段的数据包从虚拟接口lzzk0发出。在主机U中ping 192.168.70.101。

```
.7785881… 192.168.53.99        192.168.53.123       ICMP      100 Echo (ping) requ
.7824696… 1.2.3.4              192.168.53.99        ICMP      100 Echo (ping) requ
.7953385… 192.168.53.99        192.168.53.123       ICMP      100 Echo (ping) requ
.8026154… 1.2.3.4              192.168.53.99        ICMP      100 Echo (ping) requ
```

可以看到VPN服务器打印出了对应的预期数据包信息，证明数据包确实是从lzzk0端口发过来的。

Task4 Set Up the VPN Server

首先开启VPN服务器的端口转发功能：

```
while True:
    packet=os.read(tun,2048)
    if True:
        ip=IP(packet)
        ip.show()
        newip=IP(src='1.2.3.4',dst=ip.src)
        newpkt=newip/ip.payload
        #os.write(tun,bytes(newpkt))
        os.write(tun,b'hahaha')
```

编写tun_server.py代码并运行：

```
77… 192.168.53.99        192.168.53.123       ICMP      100 Echo (ping) request
09…                                           IPv6       22 Invalid IPv6 header
28… 192.168.53.99        192.168.53.123       ICMP      100 Echo (ping) request
28…                                           IPv6       22 Invalid IPv6 header
60… 192.168.53.99        192.168.53.123       ICMP      100 Echo (ping) request
95…                                           IPv6       22 Invalid IPv6 header
```

在主机U中运行tun_client.py：

```
10.0.2.7:32951 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.123
10.0.2.7:32951 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.123
10.0.2.7:32951 --> 0.0.0.0:9090
```

尝试从主机U ping 主机V：

```
[09/23/20]seed@VM:~$ sudo ip route add 192.168.70.0/24 dev lzzk0
[09/23/20]seed@VM:~$ ping 192.168.70.101
PING 192.168.70.101 (192.168.70.101) 56(84) bytes of data.
```

在主机V的Wireshark中可以看到收到了来自192.168.53.99即主机U的ICMP报文。

Task5 Handling Traffic in Both Directions

修改tun_server.py：

```
10.0.2.7:32951 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.70.101
10.0.2.7:32951 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.70.101
```

修改tun_client.py：

```
[09/23/20]seed@VM:~$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

在主机U中ping主机V，成功看到返回。

```
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'lzzk%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")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.1/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

IP_A="0.0.0.0"
PORT=9090
```

在主机U的Wireshark上可以看到虚拟端口IP地址发往主机V的数据包及reply信息。

```
[09/23/20]seed@VM:~$ sudo ./tun_client.py
Interface Name: lzzk0
```

在VPN服务器上的Wireshark可以看到从主机U的10.0.2.7到VPN服务器的10.0.2.6隧道发送UDP数据包实现了VPN的功能。

Task6 Tunnel-Breaking Experiment

在主机U与V建立Telnet连接后，关闭tun_client.py，无论输入什么，主机U的Telnet界面都没有显示。但此时TCP连接并没有中断，再次运行tun_client.py，输入字符，等待一会之后发现之前输入的字符重新出现在了Telnet界面中。当关闭tun_client.py时，之前建立的TCP连接会将内容缓存进缓冲区进入重连状态。如果及时恢复tun_client.py，那么缓冲区的字符又会重新通过TCP连接发送出去。

Task7 Routing Experiment on Host V

清除主机V的默认路由并添加下图路由：

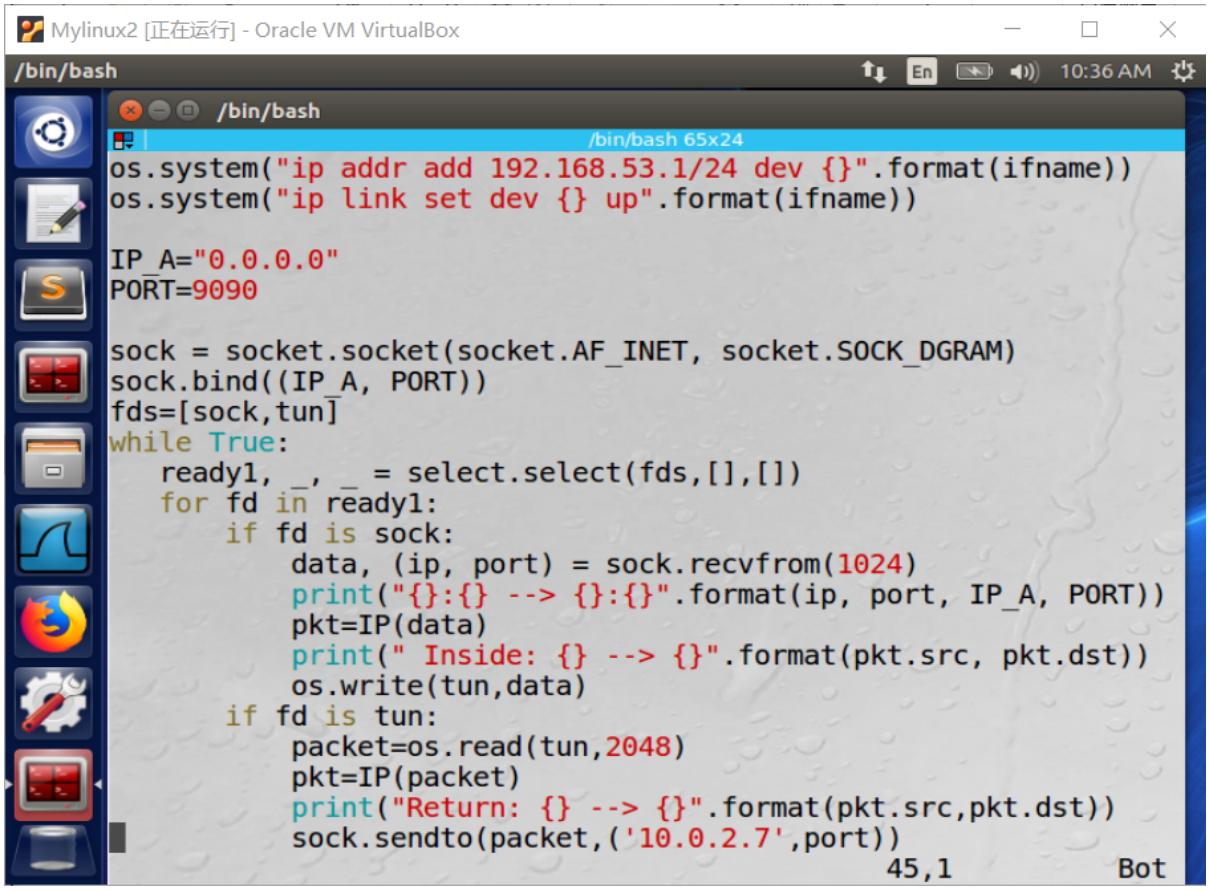
```
10.0.2.7:46036 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.70.101
10.0.2.7:46036 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.70.101
```

在主机U可以ping通主机V：

```
6:55:02.5637859… 192.168.53.99    192.168.70.101    ICMP    100
6:55:02.5638280… 192.168.70.101   192.168.53.99     ICMP    100
6:55:03.5878205… 192.168.53.99    192.168.70.101    ICMP    100
6:55:03.5878498… 192.168.70.101   192.168.53.99     ICMP    100
```

Task8 Experiment with the TUN IP Address

将主机U的虚拟端口lzzk0对应的IP地址改为192.168.30.99。重新运行tun_client.py：



主机V中收不到ICMP报文，说明在服务器处没有进行ICMP报文的发送。证明tun不在同一网段时不能接收和发送报文，这是由路由器的反向过滤安全机制导致的。当接收到的数据包源地址和宿地址调换时，若不能从接收端口发出回复包，则会丢弃收到的该数据包。为解决该问题，需手动添加反向路由到VPN服务器和主机V：

```
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.d
st))
        if fd is tun:
            packet=os.read(tun,2048)
            pkt=IP(packet)
            print("From tun <== {} --> {}".format(pkt.src,pkt.dst)
)
            sock.sendto(packet,('10.0.2.6',9090))
```

此时，主机U可以再次ping通主机V：

```
07:34:25.4667649… 192.168.53.99    192.168.70.101    ICMP
07:34:25.4667831… 192.168.53.99    192.168.70.101    ICMP
07:34:25.4677025… 192.168.70.101   192.168.53.99     ICMP
07:34:25.4677181… 192.168.70.101   192.168.53.99     ICMP
07:34:25.4688756… 10.0.2.6         10.0.2.7          UDP
07:34:26.4672468… 10.0.2.7         10.0.2.6          UDP
07:34:26.4681558… 192.168.53.99    192.168.70.101    ICMP
07:34:26.4681690… 192.168.53.99    192.168.70.101    ICMP
```

Task9 Experiment with the TAP Interface

在主机U中运行下图代码tap.py：

```
TUNSETIFF = 0x400454ca
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000

# Create the tun interface
tap = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'tap%d', IFF_TAP | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tap, TUNSETIFF, ifr)

# Get the interface name
ifname = ifname_bytes.decode('UTF-8')[:16].strip("\x00")
print("Interface Name: {}".format(ifname))

os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

while True:
    packet=os.read(tap,2048)
    if True:
        ether=Ether(packet)
        ether.show()
```

尝试Ping 192.168.53.0/24网段的IP地址：

```
[09/23/20]seed@VM:~$ sudo ip route del 0.0.0.0/0
[09/23/20]seed@VM:~$ ip route
169.254.0.0/16 dev enp0s3  scope link  metric 1000
192.168.70.0/24 dev enp0s3  proto kernel  scope link  src 192.168.
70.101  metric 100
[09/23/20]seed@VM:~$ sudo ip route add 192.168.53.0/24 dev enp0s3
via 192.168.60.1
RTNETLINK answers: Network is unreachable
[09/23/20]seed@VM:~$ sudo ip route add 192.168.53.0/24 dev enp0s3
via 192.168.70.1
[09/23/20]seed@VM:~$
```

可以看到在Ethernet层存在src MAC地址，证明了TAP接口与MAC地址绑定。

```
From tun <== 192.168.53.99 --> 192.168.70.101
From socket <== 192.168.70.101 --> 192.168.53.99
From tun <== 192.168.53.99 --> 192.168.70.101
From socket <== 192.168.70.101 --> 192.168.53.99
From tun <== 192.168.53.99 --> 192.168.70.101
From socket <== 192.168.70.101 --> 192.168.53.99
From tun <== 192.168.53.99 --> 192.168.70.101
From socket <== 192.168.70.101 --> 192.168.53.99
From tun <== 192.168.53.99 --> 192.168.70.101
From socket <== 192.168.70.101 --> 192.168.53.99
From tun <== 192.168.53.99 --> 192.168.70.101
From socket <== 192.168.70.101 --> 192.168.53.99
```