Lab7-report

57117134-张家康

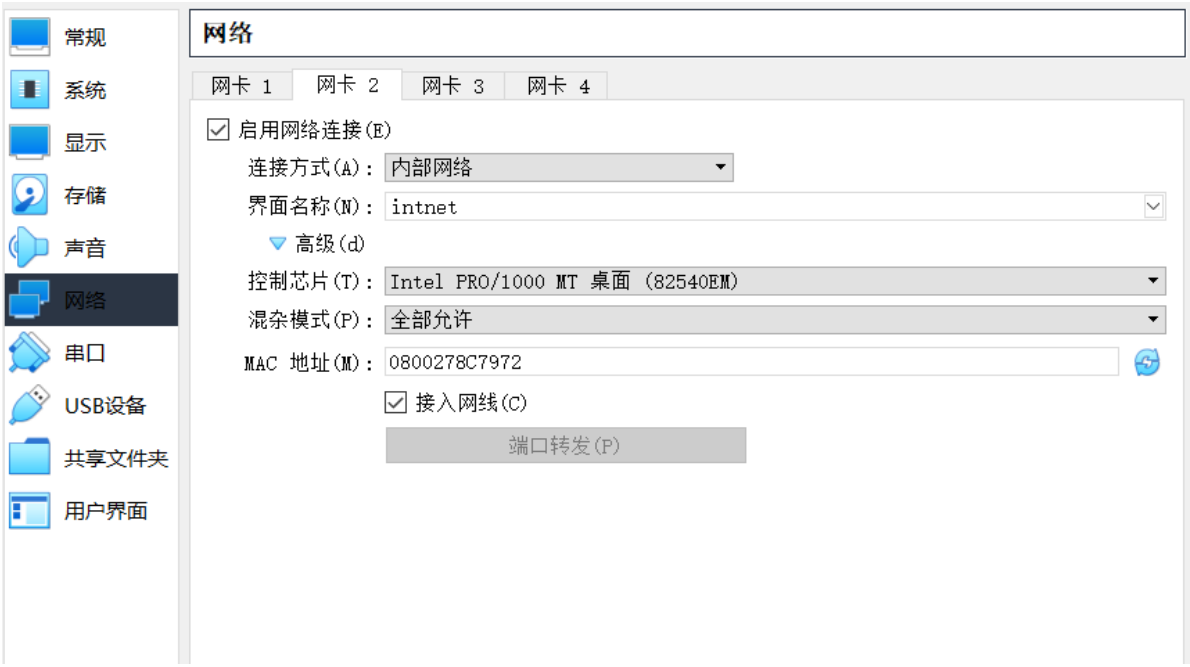
Task0：实验环境配置

本实验需要用到三台虚拟机：

```
10.0.2.7                //主机 U:  NAT 网络
10.0.2.8                //VPN 服务器G:  NAT 网络
192.168.60.1            //VPN 服务器G: 内部网络
192.168.60.101          //主机 V:  内部网络
```

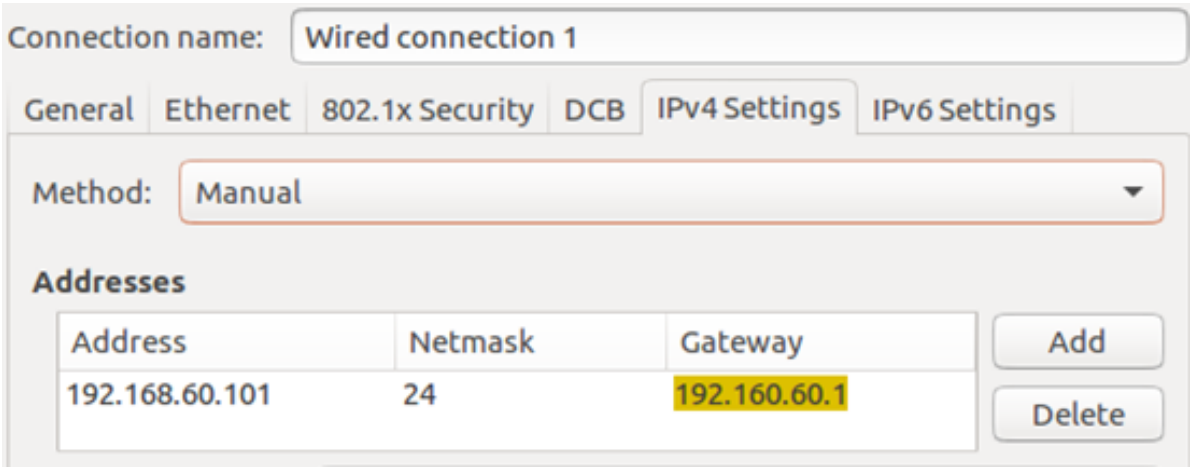
Task1: Network Setup

为VPN服务器G添加新的网卡，该网卡使用内部网络模式，作为连接VPN的网络接口设备。



打开 `虚拟机 G` 的 `Network` 设置面板，并将新网卡打开，设置 `VPN Server服务器G` 为192.168.60.1/24作为VPN网段：

打开 `Host V` 的 `Network` 设置面板，将其设置IP地址为192.168.60.101，网关采用VPN服务器IP 192.168.60.1：



然后，用 `ping` 测试三者之间的通信结果：

- `主机 U` 和 `VPN 服务器` 通信：

```
[09/22/20]seed@VM:~$ ping 10.0.2.8
PING 10.0.2.8 (10.0.2.8) 56(84) bytes of data.
64 bytes from 10.0.2.8: icmp seq=1 ttl=64 time=0.277 ms
```

- `主机 V` 和 `VPN 服务器` 通信：

```
[09/22/20]seed@VM:~$ ping 10.0.2.8
PING 10.0.2.8 (10.0.2.8) 56(84) bytes of data.
64 bytes from 10.0.2.8: icmp seq=1 ttl=64 time=0.277 ms
```

- `主机 V` 和 `主机 U` 无法通信：

```
[09/22/20]seed@VM:~$ ping 10.0.2.8
PING 10.0.2.8 (10.0.2.8) 56(84) bytes of data.
```

如图所示，长时间没有响应，通信失败。

Task 2: Create and Confifigure TUN Interface

- **Task 2.a: Name of the Interface**

在 `主机U` 上新建脚本 `tun.py`，输入代码：

```
#!/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'tun%d', IFF_TUN | IFF_NO_PI)
ifname_bytes = fcntl.ioctl(tun,TUNSETIFF, ifr)

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

while True:
    time.sleep(10)

```

在终端输入命令：

```
sudo chmod a+x tun.py
sudo ./tun.py
```

运行结果如下图所示：

```
[09/22/20]seed@VM:~$ sudo ./tun.py
Interface Name: tun0
```

打开另一个终端，输入命令 `ip address`，可以看到新增了 `tun0` 的接口。

将上述代码中第15行中的 `ifr= struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)`，改为 `ifr= struct.pack('16sH', b'zhang%d', IFF_TUN | IFF_NO_PI)`。重新运行程序，发现新增了一个名为 `zhang0` 的网卡，结果如下图所示：

```
[09/25/20]seed@VM:~/.../lab7$ sudo ./tun.py
Interface Name: zhang0
```

Task2.b: Set up the TUN Interface

在 `tun.py` 中加入以下代码，为新增的接口分配IP地址：

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

运行 `tun.py` ，此时 `zhang0` 接口拥有了IP地址，且状态变为了打开。

Task2.c: Read from the TUN Interface

在 `tun.py` 中加入以下代码，使其能够从 `zhang` 虚拟网卡中读取报文：

```
while True:
    packet = os.read(zhang, 2048)
    if True:
        ip = IP(packet)
        ip.show()
```

运行 `tun.py` ，通过 `ping` 进行测试：

- On Host U, ping a host in the `192.168.53.0/24` network. What are printed out by the `tun.py` program? What has happened? Why?

  `tun.py` 输出了icmp报文，这是因为 `192.168.53.0/24` 在 `zhang` 设备的网段中。

- On Host U, ping a host in the internal network `192.168.60.0/24` , Does `tun.py` print out anything? Why?

  没有输出，因为 `192.168.60.0/24` 不在 `zhang` 设备的网段中

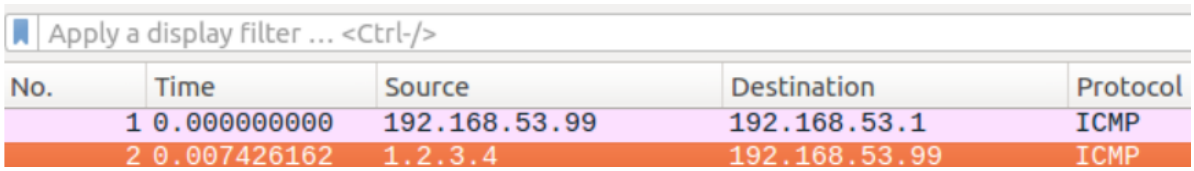
Task2.d: Write to the TUN Interface

根据从 `zhang` 读取的报文构造新的报文，并通过 `write` 的方式交付给 `zhang` .

修改 `tun.py` ，加入如下所示代码：

```
while True:
    packet = os.read(zhang, 2048)
    if True:
        ip = IP(packet)
        # Send out a spoof packet using the tun interface
        newip = IP(src='1.2.3.4', dst=ip.src)
        newpkt = newip/ip.payload
        os.write(tun, bytes(newpkt))
```

运行程序，并打开 `wireshark` 进行观察，如下图所示：

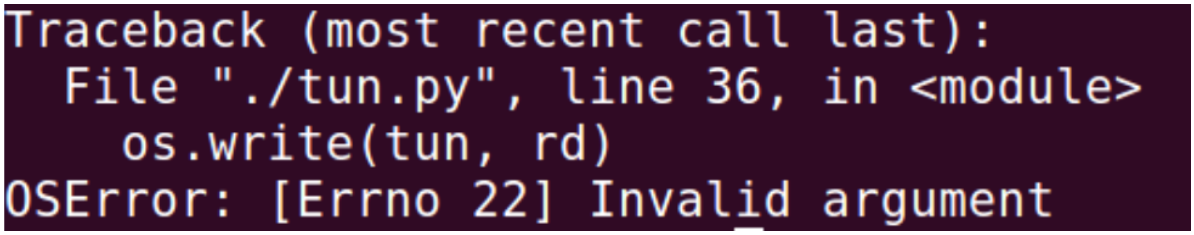


可见，报文成功发送了。

修改脚本，向 `zhang` 网卡写入随机的数据 ：

```
while True:
    packet = os.read(zhang, 2048)
    if True:
        ip = IP(packet)
        # Send out a spoof packet using the tun interface
        newip = IP(src='1.2.3.4', dst=ip.src)
        newpkt = ip.payload
        os.write(tun, bytes(newpkt))
```

运行程序后，提示出现错误，如下图所示：



Task3: Send the IP Packet to VPN Server Through a Tunnel

在 `VPN服务器G` 上，编写 VPN 程序 `tun_server.py` ：

```
#!/usr/bin/python3

from scapy.all import *

IP_A = "0.0.0.0"
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))
```

在 客户端U 编写 tun_client.py：

```
while True:
    packet = os.read(zhang, 2048)
    if True:
        sock.sendto('10.0.2.8', 9090)
        print("Send a packet\n")
```

分别在服务端和客户端运行 tun_server.py 和 tun_client.py

用 客户端U 向 192.168.53.1 发起 ping 请求，结果如下图所示：

```
10.0.2.8:35613 --> 0.0.0.0:9090
 Inside: 0.0.0.0 --> 238.147.237.222
10.0.2.8:35613 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.0.2.8:35613 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.0.2.8:35613 --> 0.0.0.0:9090
 Inside: 192.168.53.99 --> 192.168.53.1
10.0.2.8:35613 --> 0.0.0.0:9090
 Inside: 0.0.0.0 --> 238.147.237.222
```

Task 4: Set Up the VPN Server

修改 tun_server.py，使其创建 zhang 虚拟网卡，并将 sock 收到的报文的数据部分用 scapy 转
成 IP 报文，交给 zhang 网卡

```
#!/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
```

```
12
13  #create the tun interface
14  tun = os.open("/dev/net/tun", os.O_RDWR)
15  ifr= struct.pack('16sH', b'tun%d', IFF_TUN | IFF_NO_PI)
16  ifname_bytes = fcntl.ioctl(tun,TUNSETIFF, ifr)
17
18  #Get the interface nane
19  ifname= ifname_bytes.decode('UTF-8')[:16].strip("\x00")
20  print("Interface Name: {}".format(ifname)
21
22  os.system("ip addr add 192.168.53.99/24 dev {}".format(ifname))
23  os.system("ip link set dev {} up".format(ifname))
24
25  while True:
26      data, (ip, port) = sock.recvfrom(2048)
27      pkt = IP(data)
28      os.write(tun, bytes(pkt))
```

使用命令 `sudo sysctl net.ipv4.ip_forward=1` ，开启报文转发功能。

在 `V` 打开 `wireshark` 监听报文，用 `U` 向 `V` 发起 `ping` 请求 ，抓包结果如下图所示：

`主机 V` 试图回复 `主机 U` ， 这说明 `主机 V` 已经收到 `主机 U` 的 `ICMP` 报文， 单向隧道配置成功。但由于反向通道还未进行配置， 故 `ICMP` 响应不可达。

Task 5: Handling Traffific in Both Directions

1. 修改 `tun_client.py` ：

```
1   sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
2   sock.bind((IP_A, PORT))
3
4   while True:
5       # this will block until at least one interface is ready
6       ready, _, _ = select([sock, tun], [], [])
7
8       for fd in ready:
9           if fd is sock:
10              data, (ip, port) = sock.recvfrom(2048)
11              pkt = IP(data)
12              print("From socket <==: {} --> {}".format(pkt.src,
    pkt.dst))
13              os.write(tun, data)
```

```
14
15
16          if fd is tun:
17              packet = os.read(tun, 2048)
18              pkt = IP(packet)
19              print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
20              sock.sendto(packet, ('10.0.2.8', 9000))
```

2. 修改 `tun_server.py` ：

```
1   while True:
2       # this will block until at least one interface is ready
3       ready, _, _ = select([sock, tun], [], [])
4
5       for fd in ready:
6           if fd is sock:
7               data, (ip, port) = sock.recvfrom(2048)
8               pkt = IP(data)
9               print("From socket <==: {} --> {}".format(pkt.src,
    pkt.dst))
10              os.write(tun, data)
11
12          if fd is tun:
13              packet = os.read(tun, 2048)
14              pkt = IP(packet)
15              print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
16              sock.sendto(packet, ('10.0.2.7', 9000))
```

运行并测试，使用 `主机V` `ping` `主机U` ， 成功连通 ， 如下图所示：

```
PING 10.0.2.7 (10.0.2.7) 56(84) bytes of data.
64 bytes from 10.0.2.7: icmp_seq=1 ttl=63 time=4.42 ms
```

Task6: Tunning-Breaking Experiment

首先，打开隧道，并进行让 `主机U` 和 `主机V` 进行 `telnet` 通信，如下图所示：

```
Trying 10.0.2.7...
Connected to 10.0.2.7.
Escape character is '^]'.
```

然后，关闭 `tun_server.py` ，此时在 `telnet` 界面输入任何命令都没有反应。

最后，重新打开 `tun_server.py` ， 又可以正常输入 `telnet` 命令了， 同时还会出现之前输入过的命令，

Task7: Routing Experiment on Host V

配置 `主机V` 的路由：

```
sudo ip route add 192.168.53.0/24 dev enp0s3 via 192.168.60.1
sudo ip route add 10.0.2.0/24 dev enp0s3 via 192.168.60.1
```

连通 VPN 隧道后， `主机 U` 与 `主机 V` 建立 `Telnet` 连接，可以正常连接，如下图所示：

```
Trying 10.0.2.7...
Connected to 10.0.2.7.
Escape character is '^]'.
```

Task 8: Experiment with the TUN IP Address

修改隧道接口地址：

```
os.system("ip addr add 192.168.88.99/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))
```

重新运行客户端， 并用 `主机U` 对 `主机V` 发起 `ping` 请求， 此时 `ping` 命令长时间没有收到回复：

```
PING 10.0.2.7 (10.0.2.7) 56(84) bytes of data.
```

Task9: Experiment with the TAP Interface

`TUN` 网卡只能捕捉 IP 层的数据， 而 `TAP` 能捕捉数据链路层的数据。 创建 `TAP` 虚拟网卡和创建 `TUN` 虚拟网卡的方法相同， 只需要将 `IFF_TUN` 改成 `IFF_TAP` 即可。 在 `tun_server.py` 的基础上修改代码：

```
#!/usr/bin/python3
import fcntl
```

```python
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
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 nane
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()
```

在客户机上 `ping 192.168.53.2`，结果如下图所示：

```
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 76:2f:ea:84:6b:31
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = 76:2f:ea:84:6b:31
     psrc      = 192.168.53.99
     hwdst     = 00:00:00:00:00:00
     pdst      = 192.168.53.2
```

可见，`TAP` 网卡成功捕捉到了链路层的数据。