网络空间安全实训 第七次实验报告

57118221 梅昊

Task 1

1. 在主机 V 上 ping VPN 服务器，可以看到通信正常。

```
[07/28/21]seed@VM:~$ docksh 0d
root@0d9a0f8de68c:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.086 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.044 ms
^C
--- 10.9.0.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1009ms
rtt min/avg/max/mdev = 0.044/0.065/0.086/0.021 ms
```

2. 在 VPN 服务器上 ping 主机 V，可以看到通信正常。

```
[07/28/21]seed@VM:~$ docksh e3
root@e360d57fc3a8:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.051 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.046 ms
^C
--- 10.9.0.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1027ms
```

3. 在主机 U 上 ping 主机 V，可以看到无法进行通信。

```
root@0d9a0f8de68c:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3062ms
```

4. 在路由器上运行 tcpdump，嗅探接口 eth0，网络流量都可以正常嗅探。

```
root@e360d57fc3a8:/# tcpdump -i eth0 -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
02:04:28.190912 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 27, seq 1, length
 64
02:04:28.190923 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 27, seq 1, length 6
4
02:04:29.214042 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 27, seq 2, length
 64
02:04:29.214114 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 27, seq 2, length 6
4
02:04:30.238589 IP 10.9.0.5 > 10.9.0.11: ICMP echo request, id 27, seq 3, length
 64
02:04:30.238663 IP 10.9.0.11 > 10.9.0.5: ICMP echo reply, id 27, seq 3, length 6
4
```

因此得到结论，实验的配置正常。

Task 2.A

1. 在代码中将端口名修改为 mh（梅昊的姓名首字母）。

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

2. 在主机 U 上运行程序。

```
root@0d9a0f8de68c:/volumes# python3 tun.py
Interface Name: mh0
```

3. 打开另一个终端查看，可以看到修改成功。

```
root@0d9a0f8de68c:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    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
2: mh0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group defau
lt qlen 500
    link/none
```

Task 2.B

1. 在程序中添加自动分配 ip 地址的命令。

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

2. 运行程序并执行 ip address，可以看到端口已经被成功分配了地址。

```
root@0d9a0f8de68c:/# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    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
3: mh0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state
UNKNOWN group default qlen 500
    link/none
    inet 192.168.53.99/24 scope global mh0
       valid lft forever preferred lft forever
```

Task 2.C

1. 修改程序中的 while 循环。

```
while True:
# Get a packet from the tun interface
    packet = os.read(tun, 2048)
    if packet:
        ip = IP(packet)
        print(ip.summary())
```

2. 再次运行程序，选择 192.168.53.5，可以看到 ICMP 的请求报文被端口捕获了。此时 ping 不通。

```
root@0d9a0f8de68c:/volumes# python3 tun.py
Interface Name: mh0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
```

Task 2.D

1. 首先对程序中的 while 循环进行修改。首先判断是否为 Echo request，然后构造相应的响应包。

```
while True:
# Get a packet from the tun interface
    packet = os.read(tun, 2048)
    if packet:
        pkt = IP(packet)
        print(pkt.summary())
        if ICMP in pkt:
            newip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl = pkt[IP].ihl)
            newip.ttl = 99
            newicmp = ICMP(type = 0, id = pkt[ICMP].id, seq = pkt[ICMP].seq)
            if pkt.haslayer(Raw):
                data = pkt[Raw].load
                newpkt = newip/newicmp/data
            else:
                newpkt = newip/newicmp
        os.write(tun, bytes(newpkt))
```

2. 运行程序，并尝试 ping 192.168.53.5，可以看到 ping 有了响应。但这个响应是我们伪造的回复包。

```
root@0d9a0f8de68c:/volumes# python3 tun.py
Interface Name: mh0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
```

```
root@0d9a0f8de68c:/# ping 192.168.53.5
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
64 bytes from 192.168.53.5: icmp_seq=1 ttl=99 time=1.75 ms
64 bytes from 192.168.53.5: icmp_seq=2 ttl=99 time=1.83 ms
64 bytes from 192.168.53.5: icmp_seq=3 ttl=99 time=3.35 ms
64 bytes from 192.168.53.5: icmp_seq=4 ttl=99 time=1.99 ms
64 bytes from 192.168.53.5: icmp_seq=5 ttl=99 time=1.71 ms
64 bytes from 192.168.53.5: icmp_seq=6 ttl=99 time=3.31 ms
```

Task 3

1. 首先编写 tun_server 程序。

```
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'mh%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 192.168.53.99/24 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))

IP_A = "0.0.0.0"
PORT = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

while True:
# Get a packet from the tun interface
    data, (ip, port) = sock.recvfrom(2048)
    print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
    pkt = IP(data)
    print("Inside: {} --> {}".format(pkt.src, pkt.dst))
```

2. 之后编写 tun_client 程序。

```
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'mh%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 192.168.53.99/24 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))

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
SERVER_IP = "10.9.0.11"
SERVER_PORT = 9090
while True:
# Get a packet from the tun interface
    packet = os.read(tun, 2048)
    if packet:
# Send the packet via the tunnel
        pkt = IP(packet)
        print(pkt.summary())
        sock.sendto(packet, (SERVER_IP, SERVER_PORT))
```

3. 在主机 U 上运行 client 程序，在 VPN 服务器上运行 server 程序。

然后在主机 U 上 ping 192.168.53.5。可以看到，此时并没有响应。

```
root@0d9a0f8de68c:/# ping 192.168.53.5
PING 192.168.53.5 (192.168.53.5) 56(84) bytes of data.
^C
--- 192.168.53.5 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4100ms
```

4. 但是 server 和 client 程序都有了响应的输出，证明两个程序都收到了通过隧道的报文。

```
root@e360d57fc3a8:/volumes# python3 tun_server.py
Interface Name: mh0
RTNETLINK answers: File exists
10.9.0.5:42275 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:42275 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:42275 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:42275 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.5
10.9.0.5:42275 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.5
```

```
root@0d9a0f8de68c:/volumes# python3 tun_client.py
Interface Name: mh0
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.53.5 echo-request 0 / Raw
```

Task 4

1. 首先确认 router 的 ip_forward 是开启的。

```
 Router:
     image: handsonsecurity/seed-ubuntu:large
     container_name: server-router1
     tty: true
     cap_add:
             - ALL
     devices:
             - "/dev/net/tun:/dev/net/tun"
     sysctls:
             - net.ipv4.ip_forward=1
```

2. 对 server 程序进行一些修改，建立一个 tun 接口，将数据包路由到最终的目的地。

```
os.system("ip addr add 192.168.53.100/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

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))
    os.write(tun,data)
```

3. 查看服务器的网络情况，确定该嗅探的网卡为 eth1.

```
root@e360d57fc3a8:/volumes# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.11  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:0b  txqueuelen 0  (Ethernet)
        RX packets 83  bytes 8202 (8.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 17  bytes 1218 (1.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.60.11  netmask 255.255.255.0  broadcast 192.168.60.255
        ether 02:42:c0:a8:3c:0b  txqueuelen 0  (Ethernet)
        RX packets 46  bytes 4811 (4.8 KB)
        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
```

4. 开启 ping 以后，server 和 client 程序都有了相应的输出。

```
root@e360d57fc3a8:/volumes# python3 tun_server.py
Interface Name: mh0
RTNETLINK answers: File exists
10.9.0.5:35326 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
write
10.9.0.5:35326 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
write
10.9.0.5:35326 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
write
10.9.0.5:35326 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
write
10.9.0.5:35326 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.5
write
```

```
root@0d9a0f8de68c:/volumes# python3 tun_client.py
Interface Name: mh0
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
IP / ICMP 192.168.53.99 > 192.168.60.5 echo-request 0 / Raw
```

5. 观察服务器上的嗅探结果。可以看出，ICMP 已经相应。但是由于隧道是单向的，因此响应包还无法到达主机 U。

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on qiu0, link-type RAW (Raw IP), capture size 262144 bytes
23:34:05.931687 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 319, seq
291, length 64
23:34:05.931796 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 319, seq 29
1, length 64
23:34:06.957167 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 319, seq
292, length 64
23:34:06.957347 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 319, seq 29
2, length 64
23:34:07.979812 IP 192.168.53.99 > 192.168.60.5: ICMP echo request, id 319, seq
293, length 64
23:34:07.979861 IP 192.168.60.5 > 192.168.53.99: ICMP echo reply, id 319, seq 29
3, length 64
```

Task 5

1. 为了建立另一个方向的隧道，首先修改 server 程序。如果数据包来自 tun，则转发给 V。如果来自 socket，则转发给 tun。

```
os.system("ip addr add 192.168.53.100/24 dev {}".format(ifname))
os.system("ip link set dev {} up".format(ifname))

IP_A = "0.0.0.0"
PORT = 9090
ip = '10.0.9.5'
port = 9090
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))

while True:
# this will block until at least one interface is ready
    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, bytes(pkt))
    if fd is tun:
        packet = os.read(tun, 2048)
        pkt = IP(packet)
        print("From tun==>: {} --> {}".format(pkt.src, pkt.dst))
        sock.sendto(packet,(ip,port))
```

2. 之后修改 client 程序。如果数据包来自 tun，则发给 U，如果来自 socket，则转发给 tun。

```
os.system("ip addr add 192.168.53.99/24 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))

# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
SERVER_IP = "10.9.0.11"
SERVER_PORT = 9090
fds = [sock, tun]

while True:
# this will block until at least one interface is ready
    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, bytes(pkt))
    if fd is tun:
        packet = os.read(tun, 2048)
        if packet:
            pkt = IP(packet)
            print("From tun==>: {} --> {}".format(pkt.src, pkt.dst))
            sock.sendto(packet,(SERVER_IP,SERVER_PORT))
```

3. 重复之前的操作，已经可以 ping 通 192.168.60.5.

```
root@2ea54231e9ff:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=7.79 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=1.94 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=5.21 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=8.21 ms
```

4. 此时 server 和 client 都有了响应的响应。

```
root@7c797851bee0:/volumes# tun_server.py
Interface Name: mh0
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun==>: 192.168.60.5 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.5
From tun==>: 192.168.60.5 --> 192.168.53.99
root@2ea54231e9ff:/volumes# tun_client.py
Interface Name: mh0
From tun==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
From tun==>: 192.168.53.99 --> 192.168.60.5
From socket <==: 192.168.60.5 --> 192.168.53.99
```

5. 同样也可以建立 telnet 连接。

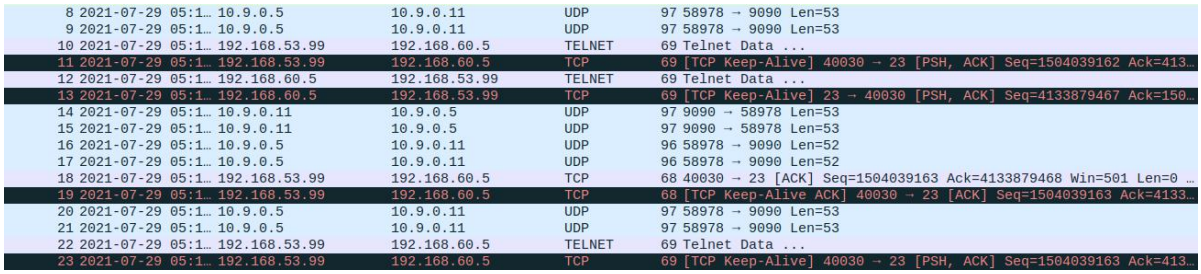
```
root@2ea54231e9ff:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
cbfae226a280 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
```

6. 使用 wireshark 捕获 telnet 过程中的数据包。可以看到数据先通过 tun 到达 VPN 服务器，然后 VPN 服务器再进行转发。反方向也是如此。从而完成了主机 U 和主机 V 之间的通信。



8	2021-07-29 05:1…	10.9.0.5	10.9.0.11	UDP	97 58978 → 9090 Len=53
9	2021-07-29 05:1…	10.9.0.5	10.9.0.11	UDP	97 58978 → 9090 Len=53
10	2021-07-29 05:1…	192.168.53.99	192.168.60.5	TELNET	69 Telnet Data ...
11	2021-07-29 05:1…	192.168.53.99	192.168.60.5	TCP	69 [TCP Keep-Alive] 40030 → 23 [PSH, ACK] Seq=1504039162 Ack=413…
12	2021-07-29 05:1…	192.168.60.5	192.168.53.99	TELNET	69 Telnet Data ...
13	2021-07-29 05:1…	192.168.60.5	192.168.53.99	TCP	69 [TCP Keep-Alive] 23 → 40030 [PSH, ACK] Seq=4133879467 Ack=150…
14	2021-07-29 05:1…	10.9.0.11	10.9.0.5	UDP	97 9090 → 58978 Len=53
15	2021-07-29 05:1…	10.9.0.11	10.9.0.5	UDP	97 9090 → 58978 Len=53
16	2021-07-29 05:1…	10.9.0.5	10.9.0.11	UDP	96 58978 → 9090 Len=52
17	2021-07-29 05:1…	10.9.0.5	10.9.0.11	UDP	96 58978 → 9090 Len=52
18	2021-07-29 05:1…	192.168.53.99	192.168.60.5	TCP	68 40030 → 23 [ACK] Seq=1504039163 Ack=4133879468 Win=501 Len=0 …
19	2021-07-29 05:1…	192.168.53.99	192.168.60.5	TCP	68 [TCP Keep-Alive ACK] 40030 → 23 [ACK] Seq=1504039163 Ack=4133…
20	2021-07-29 05:1…	10.9.0.5	10.9.0.11	UDP	97 58978 → 9090 Len=53
21	2021-07-29 05:1…	10.9.0.5	10.9.0.11	UDP	97 58978 → 9090 Len=53
22	2021-07-29 05:1…	192.168.53.99	192.168.60.5	TELNET	69 Telnet Data ...
23	2021-07-29 05:1…	192.168.53.99	192.168.60.5	TCP	69 [TCP Keep-Alive] 40030 → 23 [PSH, ACK] Seq=1504039163 Ack=413…

Task 6

1. 首先建立 telnet 连接，然后停止 server 程序。此时发现无法输入任何字符。这是因为停止 server 以后隧道中断了。因此数据包无法到达。

```
seed@cbfae226a280:/home$ ls
seed
seed@cbfae226a280:/home$
```

2. 如果在短时间内重新启动，telnet 连接会恢复。保存在缓冲区内的输入会重新显示。如果中断时间较长，程序就无法再恢复之前的内容了。

```
seed@cbfae226a280:/home$ ls^C
seed@cbfae226a280:/home$
```