lab7-report

57117227 邵长捷

VPN Tunneling Lab

Task 1: Network Setup

实验环境

Host U 192.168.43.88

VPN Server 192.168.43.150　　192.168.60.1

Host V 192.168.60.101

在 VM Fusion 中为 VPN Server 主机添加一个网卡，注意所有网络适配器都要设置为 NAT 才可以 ping 通。

```
[09/16/20]seed@VM:~$ ifconfig
ens33     Link encap:以太网  硬件地
          inet 地址:192.168.60.101
[09/16/20]seed@VM:~$ ping 192.168.60.1
PING 192.168.60.1 (192.168.60.1) 56(84) bytes of data.
64 bytes from 192.168.60.1: icmp_seq=1 ttl=64 time=0.800 ms
64 bytes from 192.168.60.1: icmp_seq=2 ttl=64 time=1.02 ms
^Z
[8]+  已停止               ping 192.168.60.1
[09/16/20]seed@VM:~$ ping 192.168.43.88
PING 192.168.43.88 (192.168.43.88) 56(84) bytes of data.
```

Host V 可以 ping 通 VPN Server，但是 ping 不通 Host U。

```
[09/16/20]seed@VM:~$ ifconfig
ens33     Link encap:以太网  硬件地
          inet 地址:192.168.43.88
[09/16/20]seed@VM:~$ ping 192.168.43.150
PING 192.168.43.150 (192.168.43.150) 56(84) bytes of data.
64 bytes from 192.168.43.150: icmp_seq=1 ttl=64 time=0.633 ms
64 bytes from 192.168.43.150: icmp_seq=2 ttl=64 time=1.17 ms
^Z
[6]+  已停止               ping 192.168.43.150
[09/16/20]seed@VM:~$ ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
```

Host U 无法 ping 通 Host V，但可以 ping 通 VPN Server。

```
[09/16/20]seed@VM:~$ ifconfig
ens33     Link encap:以太网  硬件地址
          inet 地址:192.168.43.150  广
          inet6 地址: 2408:84ec:3009:
          inet6 地址: 2408:84ec:3009:
          inet6 地址: fe80::197f:1ffc
          UP BROADCAST RUNNING MULTIC
          接收数据包:497 错误:0 丢弃:
          发送数据包:222 错误:0 丢弃:
          碰撞:0 发送队列长度:1000
          接收字节:89014 (89.0 KB)  发
          中断:19 基本地址:0x2000

ens38     Link encap:以太网  硬件地址
          inet 地址:192.168.60.1  广播
          inet6 地址: fe80::88e4:ae67
```

```
[09/16/20]seed@VM:~$ ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
64 bytes from 192.168.60.101: icmp_seq=1 ttl=64 time=0.777 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=64 time=1.32 ms
^Z
[3]+  已停止               ping 192.168.60.101
[09/16/20]seed@VM:~$ ping 192.168.43.88
PING 192.168.43.88 (192.168.43.88) 56(84) bytes of data.
64 bytes from 192.168.43.88: icmp_seq=1 ttl=64 time=0.740 ms
64 bytes from 192.168.43.88: icmp_seq=2 ttl=64 time=1.24 ms
^Z
[4]+  已停止               ping 192.168.43.88
```

VPN Server 可以 ping 通 Host U 和 Host V。

Task 2: Create and Configure TUN Interface

Task 2.a: Name of the Interface

```
[09/16/20]seed@VM:~/.../exp7$ chmod a+x tun.py
[09/16/20]seed@VM:~/.../exp7$ sudo ./tun.py
Interface Name: tun0
```

在主机 U 中运行 lab 中的 tun.py。

```
[09/16/20]seed@VM:~/.../exp7$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN 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: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen
1000
    link/ether 00:0c:29:3c:ad:20 brd ff:ff:ff:ff:ff:ff
    inet 192.168.43.88/24 brd 192.168.43.255 scope global dynamic ens33
       valid_lft 3044sec preferred_lft 3044sec
    inet6 2408:84ec:3009:e5b9:4427:8626:7fe2:8f2f/64 scope global temporary dynamic
       valid_lft 3488sec preferred_lft 3488sec
    inet6 2408:84ec:3009:e5b9:8125:9a04:df7d:335d/64 scope global mngtmpaddr noprefixroute dynamic
       valid_lft 3488sec preferred_lft 3488sec
    inet6 fe80::4211:f68c:c1c5:d702/64 scope link
       valid_lft forever preferred_lft forever
3: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
```

在另一个终端中输入 ip address，可以看到新增加了 tun0。

```
[09/16/20]seed@VM:~/.../exp7$ vim tun.py
[09/16/20]seed@VM:~/.../exp7$ sudo ./tun.py
Interface Name: scj
```

修改 tun.py，将接口名字改为 scj。

```
[09/16/20]seed@VM:~/.../exp7$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc
    link/loopback 00:00:00:00:00:00 brd 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: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mt
1000
    link/ether 00:0c:29:3c:ad:20 brd ff:ff:ff:
    inet 192.168.43.88/24 brd 192.168.43.255 s
       valid_lft 2817sec preferred_lft 2817sec
    inet6 2408:84ec:3009:e5b9:4427:8626:7fe2:8
       valid_lft 3261sec preferred_lft 3261sec
    inet6 2408:84ec:3009:e5b9:8125:9a04:df7d:3
       valid_lft 3261sec preferred_lft 3261sec
    inet6 fe80::4211:f68c:c1c5:d702/64 scope l
       valid_lft forever preferred_lft forever
3: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 150
    link/none
4: scj: <POINTOPOINT,MULTICAST,NOARP> mtu 1500
    link/none
```

ip address 可以看到新增的接口 scj。

Task 2.b: Set up the TUN Interface

```
[09/16/20]seed@VM:~/.../exp7$ sudo ip addr add 192.168.43.99/24 dev scj
[09/16/20]seed@VM:~/.../exp7$ sudo ip link set dev scj up
```

在主机 U 中输入上图命令，将接口 scj 绑定到 IP 地址 192.168.43.99 并开启。

```
os.system("ip addr add 192.168.43.99/24 dev {}".format(ifname
))
os.system("ip link set dev {} up".format(ifname))
```

可以在 tun.py 中添加上图两行，调用 system()自动执行上述功能。

```
3: scj: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qd
isc pfifo_fast state UNKNOWN group default qlen 500
    link/none
    inet 192.168.43.99/24 scope global scj
       valid_lft forever preferred_lft forever
    inet6 fe80::4f7b:43e:fe6e:5978/64 scope link flags 800
       valid_lft forever preferred_lft forever
```

ip address 中可以看到接口 scj 的信息，增加了 ip 地址。

Task 2.c: Read from the TUN Interface

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

将 tun.py 中的 while 循环替换为上图代码。

```
[09/16/20]seed@VM:~$ ping 192.168.53.123
PING 192.168.53.123 (192.168.53.123) 56(84) bytes of data.
```

在主机 U 中 ping VPN Server 的 IP 地址，无回应。原因是该主机不存在。

```
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = 84
  id        = 55110
  flags     = DF
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = 0x7733
  src       = 192.168.53.99
  dst       = 192.168.53.123
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = 0xd297
     id        = 0x174c
     seq       = 0x7
###[ Raw ]###
        load      = 'P\xd4a_b\xde\x0e\x00\x08\t\n\x0b\x0c\r\x
0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1
d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

tun.py 会打印出 src 为 192.168.53.99，dst 为 192.168.53.123 的数据包。

Task 2.d: Write to the TUN Interface

```
if True:
        ip = IP(packet)
        ip.show()
        # Send out a spoof packet using the tun inter

        newip = IP(src='1.2.3.4', dst=ip.src)
        newpkt = newip/ip.payload
        os.write(tun, bytes(newpkt))
```

在结尾增加发送伪造数据包的代码块。

Source	Destination	Protocol	Length	I
192.168.53.99	192.168.53.123	ICMP	84	E
1.2.3.4	192.168.53.99	ICMP	84	E
192.168.53.99	192.168.53.123	ICMP	84	E
1.2.3.4	192.168.53.99	ICMP	84	E

再次运行 ping 192.168.53.123 会发现 Wireshark 中不仅捕捉到由 scj 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。

```
newpkt = newip/ip.payload
#os.write(tun, bytes(newpkt))
os.write(tun,b'hahaha')
```

将发送到 tun 的数据改为 hahaha，重复上述操作。

192.168.53.99	192.168.53.123	ICMP	84	Echo (ping) request id=0x1939, seq=1
N/A	N/A	N/A	6	Raw packet data[Malformed Packet]
192.168.53.99	192.168.53.123	ICMP	84	Echo (ping) request id=0x1939, seq=2
N/A	N/A	N/A	6	Raw packet data[Malformed Packet]
192.168.53.99	192.168.53.123	ICMP	84	Echo (ping) request id=0x1939, seq=3
N/A	N/A	N/A	6	Raw packet data[Malformed Packet]

Wireshark 捕捉到一些无意义的数据包。

Task 3: Send the IP Packet to VPN Server Through a Tunnel

```
[09/16/20]seed@VM:~/.../exp7$ sudo ./tun_server.py
```

在 VPN Server 主机上运行 tun_server.py。

```
[09/16/20]seed@VM:~/.../exp7$ sudo ./tun_client.py
Interface Name: scj
```

在主机 U 上运行 tun_client.py。

```
[09/16/20]seed@VM:~/.../exp7$ ping 192.168.53.123
PING 192.168.53.123 (192.168.53.123) 56(84) bytes of data.
```

在主机 U 上 ping 192.168.53.123。

```
[09/16/20]seed@VM:~/.../exp7$ sudo ./tun_server.py
192.168.43.88:51006 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.123
192.168.43.88:51006 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.123
192.168.43.88:51006 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.53.123
```

此时可以看到 VPN Server 上打印出了接收到的数据包信息，由于主机 U 和 VPN Server 通过 192.168.43.0/24 网段相连，因而外层 UDP 数据包的源 IP 地址为 192.168.43.88，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 scj 的 IP 地址 192.168.53.99。综上，实现了隧道的作用。

```
[09/16/20]seed@VM:~/.../exp7$ sudo ip route add 192.168.60.0/
24 dev scj
```

在主机 U 添加上图路由，实现将发往 192.168.60.0/24 网段的数据包从虚拟接口 scj 发出。

```
[09/16/20]seed@VM:~/.../exp7$ ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
```

在主机 U 中 ping 192.168.60.101。

```
Inside: 192.168.53.99 --> 192.168.60.101
192.168.43.88:51006 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.101
192.168.43.88:51006 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.101
```

此时可以看到 VPN Server 打印出了对应的预期数据包信息，证明数据包确实是从 scj 端口发过来的。

Task 4: Set Up the VPN Server

```
[09/16/20]seed@VM:~/.../exp7$ sudo sysctl net.ipv4.ip_forward
=1
net.ipv4.ip_forward = 1
```

首先开启 VPN Server 的端口转发功能。

```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

IP_A = "0.0.0.0"
PORT = 9090

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'scj',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))

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_server.py" 35L, 888C                    35,1-8        全部
```

编写上图的 tun_server.py 代码并运行。

```
[09/16/20]seed@VM:~/.../exp7$ sudo ./tun_client.py
Interface Name: scj
```

在主机 U 运行 tun_client.py。

```
[09/16/20]seed@VM:~/.../exp7$ ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
```

尝试从主机 Uping 主机 V。

```
[09/16/20]seed@VM:~/.../exp7$ sudo ./tun_server.py
Interface Name: scj
192.168.43.88:49385 --> 0.0.0.0:9090
Inside: 192.168.53.99 --> 192.168.60.101
192.168.43.88:49385 --> 0.0.0.0:9090
```

可以看到 VPN Server 打印出了数据包信息。

```
.60… 192.168.53.99      192.168.60.101      ICMP     98 Echo (ping) request  id=0x18e2, seq=323/17153, ttl=63 (reply in 2414)
181… 192.168.60.101     192.168.53.99       ICMP     98 Echo (ping) reply    id=0x18e2, seq=323/17153, ttl=64 (request in 2413)
142… 192.168.53.99      192.168.60.101      ICMP     98 Echo (ping) request  id=0x18e2, seq=324/17409, ttl=63 (reply in 2427)
148… 192.168.60.101     192.168.53.99       ICMP     98 Echo (ping) reply    id=0x18e2, seq=324/17409, ttl=64 (request in 2426)
197… 192.168.53.99      192.168.60.101      ICMP     98 Echo (ping) request  id=0x18e2, seq=325/17665, ttl=63 (reply in 2430)
177… 192.168.60.101     192.168.53.99       ICMP     98 Echo (ping) reply    id=0x18e2, seq=325/17665, ttl=64 (request in 2429)
116… 192.168.53.99      192.168.60.101      ICMP     98 Echo (ping) request  id=0x18e2, seq=326/17921, ttl=63 (reply in 2436)
113… 192.168.60.101     192.168.53.99       ICMP     98 Echo (ping) reply    id=0x18e2, seq=326/17921, ttl=64 (request in 2435)
```

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。

Task 5: Handling Traffic in Both Directions

```python
#!/usr/bin/python3

import fcntl
import struct
import os
import time
from scapy.all import *
import select

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

IP_A = "0.0.0.0"
PORT = 9090
port = 10000

# Create the tun interface
tun = os.open("/dev/net/tun", os.O_RDWR)
ifr = struct.pack('16sH', b'scj',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))

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP_A, PORT))
fds = [sock,tun]
while True:
    ready1, _, _= select.select(fds, [], [])
    for fd in ready1:
        if fd is sock:
            data, (ip, port) = sock.recvfrom(1024)
            print("{}:{} --> {}:{}".format(ip, port, IP_A, PORT))
            pkt = IP(data)
            print("Inside: {} --> {}".format(pkt.src, pkt.dst))
            os.write(tun, data)
        if fd is tun:
            packet = os.read(tun,2048)
            pkt = IP(packet)
            print("Return: {} --> {}".format(pkt.src,pkt.dst))
            sock.sendto(packet,('192.168.43.88',port))
```

修改 tun_server.py 如上图。

```python
#!/usr/bin/python3

import fcntl
import struct
import os
import time
from scapy.all import *
import select

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'scj',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("ifconfig {} 192.168.53.99/24 up".format(ifname))
# Create UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
# Get a packet from the tun interface
    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))
        if fd is tun:
            packet = os.read(tun, 2048)
            pkt = IP(packet)
            print("From tun ==> {} --> {}".format(pkt.src, pkt.dst))
            sock.sendto(packet, ('192.168.43.150', 9090))
```

修改 tun_client.py 如上图。

```
[09/16/20]seed@VM:~/.../exp7$ ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
64 bytes from 192.168.60.101: icmp_seq=539 ttl=63 time=4.96 ms
64 bytes from 192.168.60.101: icmp_seq=540 ttl=63 time=4.71 ms
64 bytes from 192.168.60.101: icmp_seq=541 ttl=63 time=7.15 ms
64 bytes from 192.168.60.101: icmp_seq=542 ttl=63 time=5.42 ms
64 bytes from 192.168.60.101: icmp_seq=543 ttl=63 time=3.35 ms
64 bytes from 192.168.60.101: icmp_seq=544 ttl=63 time=4.18 ms
64 bytes from 192.168.60.101: icmp_seq=545 ttl=63 time=5.23 ms
64 bytes from 192.168.60.101: icmp_seq=546 ttl=63 time=4.36 ms
```

在主机 U 中尝试 ping 主机 V，成功看到返回！

```
531… 192.168.53.99    192.168.60.101   ICMP   98 Echo (ping) request  id=0x1755, seq=779/2819, ttl=63 (reply in 8288)
793… 192.168.60.101   192.168.53.99    ICMP   98 Echo (ping) reply    id=0x1755, seq=779/2819, ttl=64 (request in 8287)
142… 192.168.53.99    192.168.60.101   ICMP   98 Echo (ping) request  id=0x1755, seq=780/3075, ttl=63 (reply in 8302)
437… 192.168.60.101   192.168.53.99    ICMP   98 Echo (ping) reply    id=0x1755, seq=780/3075, ttl=64 (request in 8301)
389… 192.168.53.99    192.168.60.101   ICMP   98 Echo (ping) request  id=0x1755, seq=781/3331, ttl=63 (reply in 8314)
681… 192.168.60.101   192.168.53.99    ICMP   98 Echo (ping) reply    id=0x1755, seq=781/3331, ttl=64 (request in 8313)
347… 192.168.53.99    192.168.60.101   ICMP   98 Echo (ping) request  id=0x1755, seq=782/3587, ttl=63 (reply in 8321)
864… 192.168.60.101   192.168.53.99    ICMP   98 Echo (ping) reply    id=0x1755, seq=782/3587, ttl=64 (request in 8320)
```

Wireshark 可以看到由主机 U 的虚拟端口 IP 地址发往主机 V 的数据包及 reply 信息。

```
… 192.168.43.150   192.168.43.88    UDP    126 9090 → 59304 Len=84
… 192.168.43.88    192.168.43.150   UDP    126 59304 → 9090 Len=84
… 192.168.53.99    192.168.60.101   ICMP    98 Echo (ping) request  i
… 192.168.60.101   192.168.53.99    ICMP    98 Echo (ping) reply    i
… 192.168.43.150   192.168.43.88    UDP    126 9090 → 59304 Len=84
… 192.168.43.172   61.181.204.106   TCP     60 57998 → 80 [ACK] Seq=2
… 192.168.43.172   61.181.204.106   TCP     60 57998 → 80 [ACK] Seq=2
… 192.168.43.88    192.168.43.150   UDP    126 59304 → 9090 Len=84
… 192.168.53.99    192.168.60.101   ICMP    98 Echo (ping) request  i
… 192.168.60.101   192.168.53.99    ICMP    98 Echo (ping) reply    i
```

在 Server VPN 中的 Wireshark 可以看到利用从主机 U 的 192.168.43.88 到 VPN Server 的 192.168.43.150 隧道发送 UDP 数据包实现了 VPN 的功能。

```
From tun ==> 192.168.53.99 --> 192.168.60.101
From socket <== 192.168.60.101 --> 192.168.53.99
From tun ==> 192.168.53.99 --> 192.168.60.101
From socket <== 192.168.60.101 --> 192.168.53.99
From tun ==> 192.168.53.99 --> 192.168.60.101
```

```
[09/17/20]seed@VM:~/.../exp7$ telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login:
```

Telnet 也可以使用。

```
020-09-17 01:13:42.5313709… 192.168.43.150   192.168.43.88    UDP    131 9090 → 60719 Len=89
020-09-17 01:13:42.5321610… 192.168.43.150   192.168.43.88    UDP     94 9090 → 60719 Len=52
020-09-17 01:13:42.5337519… 192.168.43.88    192.168.43.150   UDP     94 60719 → 9090 Len=52
020-09-17 01:13:42.5341646… 192.168.43.88    192.168.43.150   UDP     94 60719 → 9090 Len=52
020-09-17 01:13:42.5348777… 192.168.53.99    192.168.60.101   TCP     66 41696 → 23 [ACK] Seq=1183604653 Ack
020-09-17 01:13:42.5355598… 192.168.53.99    192.168.60.101   TCP     66 41696 → 23 [FIN, ACK] Seq=118360465
020-09-17 01:13:42.5355732… 192.168.60.101   192.168.53.99    TCP     66 23 → 41696 [ACK] Seq=1730672788 Ack
020-09-17 01:13:42.5367206… 192.168.43.150   192.168.43.88    UDP     94 9090 → 60719 Len=52
```

可以看到主机 U 通过 UDP 隧道传送的 TCP 数据包（telnet）成功与内网主机 V 通信。

Task 6: Tunnel-Breaking Experiment

```
[09/17/20]seed@VM:~/.../exp7$ ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
64 bytes from 192.168.60.101: icmp_seq=1 ttl=63 time=4.00 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=5.31 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=4.94 ms
64 bytes from 192.168.60.101: icmp_seq=4 ttl=63 time=4.02 ms
64 bytes from 192.168.60.101: icmp_seq=5 ttl=63 time=4.45 ms
64 bytes from 192.168.60.101: icmp_seq=6 ttl=63 time=3.39 ms
From 192.168.43.88 icmp_seq=7 Destination Host Unreachable
From 192.168.43.88 icmp_seq=8 Destination Host Unreachable
From 192.168.43.88 icmp_seq=9 Destination Host Unreachable
64 bytes from 192.168.60.101: icmp_seq=12 ttl=63 time=6.60 ms
64 bytes from 192.168.60.101: icmp_seq=13 ttl=63 time=5.51 ms
From 192.168.43.88 icmp_seq=10 Destination Host Unreachable
From 192.168.43.88 icmp_seq=11 Destination Host Unreachable
64 bytes from 192.168.60.101: icmp_seq=14 ttl=63 time=3.06 ms
64 bytes from 192.168.60.101: icmp_seq=15 ttl=63 time=5.42 ms
```

当断开 tun 断开和重启时，会出现上述情况，即 VPN 隧道中断，导致无法 ping 通内网主机，重新开启 tun 后，则恢复连接。

```
$ lslsls
-dash: 2: lslsls: not found
$
```

在断开 tun 时，主机 U 终端无法输入，即无法显示来自内网主机 V 的终端信息，重新开启 tun 后即可恢复功能，且在主机 U 中输入的内容也一并出现。

Task 7: Routing Experiment on Host V

```
[09/17/20]seed@VM:~$ sudo ip route del 0.0.0.0/0
[09/17/20]seed@VM:~$ ip route
169.254.0.0/16 dev ens33  scope link  metric 1000
192.168.60.0/24 dev ens33  proto kernel  scope link  src 192.168.60.101  metric
100
[09/17/20]seed@VM:~$
```

清除主机 V 的默认路由。

```
[09/17/20]seed@VM:~$ sudo ip route add 192.168.53.0/24 dev ens33 via 192.168.60.
1
```

在主机 V 中添加上图的路由。

```
From tun ==> 0.0.0.0 --> 97.192.131.164
From tun ==> 192.168.53.99 --> 192.168.60.101
From socket <== 192.168.60.101 --> 192.168.53.99
From tun ==> 192.168.53.99 --> 192.168.60.101
From socket <== 192.168.60.101 --> 192.168.53.99
From tun ==> 192.168.53.99 --> 192.168.60.101
From socket <== 192.168.60.101 --> 192.168.53.99
From tun ==> 192.168.53.99 --> 192.168.60.101
```

```
[09/17/20]seed@VM:~/.../exp7$ ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
64 bytes from 192.168.60.101: icmp_seq=1 ttl=63 time=4.09 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=5.32 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=4.85 ms
64 bytes from 192.168.60.101: icmp_seq=4 ttl=63 time=4.82 ms
64 bytes from 192.168.60.101: icmp_seq=5 ttl=63 time=5.34 ms
64 bytes from 192.168.60.101: icmp_seq=6 ttl=63 time=5.33 ms
```

在主机 U 可以 ping 通主机 V。

Task 8: Experiment with the TUN IP Address

```
os.system("ip addr add 192.168.30.99/24 dev {}".format(ifname))
```

将主机 U 的虚拟端口 scj 对应的 IP 地址改为 192.168.30.99。

重新运行 tun_client.py。

```
192.168.43.88:46737 --> 0.0.0.0:9090
Inside: 192.168.30.99 --> 192.168.60.101
192.168.43.88:46737 --> 0.0.0.0:9090
Inside: 192.168.30.99 --> 192.168.60.101
192.168.43.88:46737 --> 0.0.0.0:9090
Inside: 192.168.30.99 --> 192.168.60.101
```

在 VPN Server 中可以正常解 UDP 包。

在主机 V 中收不到 ICMP 报文，说明在 Server 处并没有进行 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，这是由于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（即回复包）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。

为了解决该问题，需要手动添加反向路由到 VPN Server 和主机 V。

```
[09/17/20]seed@VM:~/.../exp7$ sudo ip route add 192.168.30.0/
24 dev scj
```

向 VPN Server 添加上图路由。

```
[09/17/20]seed@VM:~$ sudo ip route add 192.168.30.0/24 dev ens33 via 192.168.60.1
```

向主机 V 添加上图路由。

```
[09/17/20]seed@VM:~$ ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
64 bytes from 192.168.60.101: icmp_seq=1 ttl=63 time=3.75 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=3.98 ms
```

此时，主机 U 可以再次 ping 通主机 V。

Task 9: Experiment with the TAP Interface

```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 tap 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:
    # Get a packet from the tun interface
    packet = os.read(tap, 2048)
    if True:
        ether = Ether(packet)
        ether.show()
```

在主机 U 中运行上图代码 tap.py。

```
[09/17/20]seed@VM:~$ ping 192.168.53.123
PING 192.168.53.123 (192.168.53.123) 56(84) bytes of data.
From 192.168.53.99 icmp_seq=1 Destination Host Unreachable
From 192.168.53.99 icmp_seq=2 Destination Host Unreachable
From 192.168.53.99 icmp_seq=3 Destination Host Unreachable
From 192.168.53.99 icmp_seq=4 Destination Host Unreachable
```

尝试 ping 192.168.53.0/24 网段的 IP 地址。

```
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = d6:7c:56:76:87:6e
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = d6:7c:56:76:87:6e
     psrc      = 192.168.53.99
     hwdst     = 00:00:00:00:00:00
     pdst      = 192.168.53.123
```

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。

Virtual Private Network (VPN) Lab

Task 1: VM Setup

与前面的实验配置一致，实验环境如下：

Host U 192.168.43.88

VPN Server 192.168.43.150　　192.168.60.1

Host V 192.168.60.101

Task 2: Creating a VPN Tunnel using TUN/TAP

Step 1: Run VPN Server

```
7/20]seed@VM:~/.../vpn$ sudo ./vpnserver
```

在 VPN Server 主机上编译并启动 vpnserver 程序。

```
tun0      Link encap:未指定  硬件地址 00-00-00-00-00-00-00-
00-00
          POINTOPOINT NOARP MULTICAST  MTU:1500  跃点数:1
          接收数据包:0 错误:0 丢弃:0 过载:0 帧数:0
          发送数据包:0 错误:0 丢弃:0 过载:0 载波:0
          碰撞:0 发送队列长度:500
          接收字节:0 (0.0 B)  发送字节:0 (0.0 B)

[09/17/20]seed@VM:~$ sudo ifconfig tun0 192.168.53.1/24 up
[09/17/20]seed@VM:~$ sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
```

ifconfig -a 可以查看新增的虚拟端口 tun0，为其绑定 IP 地址 192.168.53.1 并打开端口转发功能。

Step 2: Run VPN Client

```
[09/17/20]seed@VM:~/.../vpn$ sudo ./vpnclient
```

在主机 U 上编译并运行 vpnclient 程序。

```
[09/17/20]seed@VM:~$ sudo ifconfig tun0 192.168.53.5/24 up
```

为虚拟端口 tun0 绑定 IP 地址 192.168.53.5。

Step 3: Set Up Routing on Client and Server VMs

```
[09/17/20]seed@VM:~$ sudo route add -net 192.168.60.0/24 tun0
[09/17/20]seed@VM:~$ ip route
default via 192.168.43.1 dev ens33  proto static  metric 100
169.254.0.0/16 dev ens33  scope link  metric 1000
192.168.43.0/24 dev ens33  proto kernel  scope link  src 192.168.43.88  metric 1
00
192.168.53.0/24 dev tun0  proto kernel  scope link  src 192.168.53.5
192.168.60.0/24 dev tun0  scope link
```

在主机 U 上添加上图的路由信息并确认。

```
[09/17/20]seed@VM:~$ ip route
default via 192.168.60.1 dev ens38  proto static  metric 100
default via 192.168.43.1 dev ens33  proto static  metric 101
169.254.0.0/16 dev ens38  scope link  metric 1000
192.168.43.0/24 dev ens33  proto kernel  scope link  src 192.168.43.150  metric 100
192.168.53.0/24 dev tun0  proto kernel  scope link  src 192.168.53.1
192.168.60.0/24 dev ens38  proto kernel  scope link  src 192.168.60.1  metric 100
```

确认 VPN Server 主机的路由表配置正确。

Step 4: Set Up Routing on Host V

```
[09/17/20]seed@VM:~$ sudo ip route add 192.168.53.0/24 dev ens33 via 192.168.60.1
[09/17/20]seed@VM:~$ ip route
default via 192.168.60.1 dev ens33  proto static  metric 100
169.254.0.0/16 dev ens33  scope link  metric 1000
192.168.53.0/24 via 192.168.60.1 dev ens33
192.168.60.0/24 dev ens33  proto kernel  scope link  src 192.168.60.101  metric 100
```

在主机 V 中添加上图路由，事实上由于默认路由已经起到了作用，所以不添加这条路由也是可以正常连通的。

Step 5: Test the VPN Tunnel

```
[09/17/20]seed@VM:~$ ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
64 bytes from 192.168.60.101: icmp_seq=1 ttl=63 time=1.89 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=3.33 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=2.81 ms
64 bytes from 192.168.60.101: icmp_seq=4 ttl=63 time=2.62 ms
64 bytes from 192.168.60.101: icmp_seq=5 ttl=63 time=1.62 ms
```

```
Got a packet from the tunnel
Got a packet from TUN
Got a packet from the tunnel
Got a packet from TUN
Got a packet from the tunnel
Got a packet from TUN
Got a packet from the tunnel
```

在主机 U 可以 ping 通主机 V。

192.168.43.88	192.168.43.150	UDP	126	35548 → 55555 Len=84
192.168.53.5	192.168.60.101	ICMP	98	Echo (ping) request id=0x
192.168.60.101	192.168.53.5	ICMP	98	Echo (ping) reply id=0x
192.168.43.150	192.168.43.88	UDP	126	55555 → 35548 Len=84

Wireshark 可以看到通过主机 U 和 VPN Server 的 UDP 隧道传送 UDP 数据包，解包之后为 ICMP 包，即实现了主机 U 的 ICMP 请求发送到内网主机 V 及原路回复。

```
[09/17/20]seed@VM:~$ telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Thu Sep 17 01:19:02 EDT 2020 from 192.168.53.99 on pts/18
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

0 packages can be updated.
0 updates are security updates.

$
```

在主机 U 中对主机 V 进行 Telnet 测试，可以正常使用主机 V 的 Telnet 服务。

Step 6: Tunnel-Breaking Test

```
Got a packet from the tunnel
Got a packet from TUN
^C
[09/17/20]seed@VM:~/.../vpn$
```

```
0 updates are security updates.

$
```

当断开主机 U 中的 vpnclient 程序时，连接中断，右侧的终端无法输入字符，即无法显示希望传送到主机 V 的命令。

```
sudo ifconfig tun0 192.168.53.5/24 up
sudo ip route add 192.168.60.0/24 dev tun0
```

重启 vpnclient 并重新添加配置，Telnet 服务仍然无法恢复，这与之前 Python 程序的结果不一致，可能是此 C 语言程序存在无法重连的瑕疵。

Task 3: Encrypting the Tunnel

```
127.0.0.1       localhost
127.0.1.1       VM
192.168.43.150  vpnlabserver.com
```

在主机 U 的/etc/hosts 文件中添加上图第三行。

运行 tlsclient 和 tlsserver 程序，注意使用 root 权限运行。

```
[09/17/20]seed@VM:~/.../tls$ sudo date -s 20190101
2019年 01月 01日 星期二 00:00:00 EST
[01/01/19]seed@VM:~/.../tls$ sudo ./tlsclient vpnlabserver.com  4433
SSL connection is successful
SSL connection using AES256-GCM-SHA384
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html><html><head><title>Hello World</title></head><style>body {backgro
und-color: black}h1 {font-size:3cm; text-align: center; color: white;text-shadow
: 0 0 3mm yellow}</style></head><body><h1>Hello, world!</h1></body></html>
```

```
[09/17/20]seed@VM:~/.../tls$ sudo ./tlsserver
3073955520:error:14094415:SSL routines:ssl3_read_bytes:sslv3 alert certificate e
xpired:s3_pkt.c:1487:SSL alert number 45
SSL connection established!
Received: GET / HTTP/1.1
Host: vpnlabserver.com
```

可以看到在客户端和服务端都打印出了连接成功的信息。

```
[09/17/20]seed@VM:~/.../tls$ sudo ./tlsclient vpnlabserver.com  4433
3073406656:error:14090086:SSL routines:ssl3_get_server_certificate:certificate v
erify failed:s3_clnt.c:1264:
```

注意：由于证书过期，需要先把本地时间往回调，否则会报上图的认证错误。

```
20-09-17 06:06:42.2797248… 192.168.43.88    192.168.43.150   TCP      74
20-09-17 06:06:42.2797475… 192.168.43.150   192.168.43.88    TCP      74
20-09-17 06:06:42.2802669… 192.168.43.88    192.168.43.150   TCP      66
20-09-17 06:06:42.2805965… 192.168.43.88    192.168.43.150   TCP      371
20-09-17 06:06:42.2806042… 192.168.43.150   192.168.43.88    TCP      66
20-09-17 06:06:42.2812774… 192.168.43.150   192.168.43.88    TCP      1089
20-09-17 06:06:42.2820931… 192.168.43.88    192.168.43.150   TCP      66
20-09-17 06:06:42.2831231… 192.168.43.88    192.168.43.150   TCP      384
20-09-17 06:06:42.2900805… 192.168.43.150   192.168.43.88    TCP      292
20-09-17 06:06:42.2907215… 192.168.43.88    192.168.43.150   TCP      134
20-09-17 06:06:42.2908631… 192.168.43.150   192.168.43.88    TCP      373
20-09-17 06:06:42.2910797… 192.168.43.150   192.168.43.88    TCP      97
20-09-17 06:06:42.2916199… 192.168.43.88    192.168.43.150   TCP      66
20-09-17 06:06:42.2919936… 192.168.43.88    192.168.43.150   TCP      66
```

```
▼ Data (305 bytes)
    Data: 160301012c0100012803035132cb387353a955cf903682fb...
    [Length: 305]
```

```
0010  01 65 c9 0e 40 00 40 06  98 45 c0 a8 2b 58 c0 a8   .e..@.@. .E..+X..
0020  2b 96 92 cc 11 51 13 be  38 d9 9e fc e7 af 80 18   +....Q.. 8.......
0030  00 e5 6d 2f 00 00 01 01  08 0a 00 07 f1 d4 00 07   ..m/.... ........
0040  fc a5 16 03 01 01 2c 01  00 01 28 03 03 51 32 cb   ......,. ..(..Q2.
0050  38 73 53 a9 55 cf 90 36  82 fb 80 8b f2 6b 47 52   8sS.U..6 .....kGR
0060  8b 50 fa 96 60 f3 fc 8d  88 63 66 cf 4a 00 00 aa   .P..`... .cf.J...
0070  c0 30 c0 2c c0 28 c0 24  c0 14 c0 0a 00 a5 00 a3   .0.,.($. ........
0080  00 a1 00 9f 00 6b 00 6a  00 69 00 68 00 39 00 38   .....k.j .i.h.9.8
0090  00 37 00 36 00 88 00 87  00 86 00 85 c0 32 c0 2e   .7.6.... .....2..
00a0  c0 2a c0 26 c0 0f c0 05  00 9d 00 3d 00 35 00 84   .*.&.... ...=.5..
00b0  c0 2f c0 2b c0 27 c0 23  c0 13 c0 09 00 a4 00 a2   ./.+.'.# ........
00c0  00 a0 00 9e 00 67 00 40  00 3f 00 3e 00 33 00 32   .....g.@ .?.>.3.2
00d0  00 31 00 30 00 9a 00 99  00 98 00 97 00 45 00 44   .1.0.... .....E.D
00e0  00 43 00 42 c0 31 c0 2d  c0 29 c0 25 c0 0e c0 04   .C.B.1.- .).%....
00f0  00 9c 00 3c 00 2f 00 96  00 41 c0 11 c0 07 c0 0c   ...<./.. .A......
0100  c0 02 00 05 00 04 c0 12  c0 08 00 16 00 13 00 10   ........ ........
0110  00 0d c0 0d c0 03 00 0a  00 ff 01 00 00 55 00 0b   ........ .....U..
0120  00 04 03 00 01 02 00 0a  00 1c 00 1a 00 17 00 19   ........ ........
0130  00 1c 00 1b 00 18 00 1a  00 16 00 0e 00 0d 00 0b   ........ ........
0140  00 0c 00 09 00 0a 00 23  00 00 00 0d 00 20 00 1e   .......# ..... ..
0150  06 01 06 02 06 03 05 01  05 02 05 03 04 01 04 02   ........ ........
0160  04 03 03 01 03 02 03 03  02 01 02 02 02 03 00 0f   ........ ........
0170  00 01 01                                          ...
```

Wireshark 显示 TCP 包的 Data 数据的确已被加密。

Task 4: Authenticating the VPN Server

```
SSL_CTX_set_verify(ctx, SSL_VERIFY_PEER, NULL);
if(SSL_CTX_load_verify_locations(ctx,NULL, CA_DIR) < 1){
```

第一行代码用于告诉 TLS 是否进行证书验证。这里使用 SSL_VERIFY_PEER 选项来指明需要进行验证。第二行用于告诉 TLS 证书存储的文件夹位置。

```
X509_VERIFY_PARAM *vpm = SSL_get0_param(ssl);
X509_VERIFY_PARAM_set1_host(vpm, hostname, 0);
```

在 tlsclient.c 中上图代码用于主机名检查。

```
1   [ req ]
2   default_bits       = 4096
3   distinguished_name = req_distinguished_name
4
5   [ req_distinguished_name ]
6   countryName                 = Country Name (2 letter code)
7   countryName_default         = CN
8   stateOrProvinceName         = State or Province Name (full name)
9   stateOrProvinceName_default = JiangSu
10  localityName                = Locality Name (eg, city)
11  localityName_default        = NanJing
12  organizationName            = Organization Name (eg, company)
13  organizationName_default    = SEU
14  commonName                  = scj
15  commonName_max              = 64
16  commonName_default          = Ted CA Test
```

首先写好 ca.conf 配置文件。

```
[09/17/20]seed@VM:~/.../cert_test$ openssl genrsa -out cakey.pem 4096
Generating RSA private key, 4096 bit long modulus
.....................................................................++
.............................++
e is 65537 (0x10001)
```

生成 cakey.pem 密钥。

```
[09/17/20]seed@VM:~/.../cert_test$ openssl req  -new -sha256 -out cacsr.pem -key
 cakey.pem -config ca.conf
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [CN]:
State or Province Name (full name) [JiangSu]:
Locality Name (eg, city) [NanJing]:
Organization Name (eg, company) [SEU]:
scj [Ted CA Test]:
```

生成 ca 证书签发请求，得到 cacsr.pem。

```
[09/17/20]seed@VM:~/.../cert_test$ openssl x509 -req -days 365 -in cacsr.pem -si
gnkey cakey.pem -out cacrt.pem
Signature ok
subject=/C=CN/ST=JiangSu/L=NanJing/O=SEU/CN=Ted CA Test
Getting Private key
```

生成 ca 根证书，得到 ca.crt。

```
[09/17/20]seed@VM:~/.../cert_test$ openssl genrsa -out serverkey.pem 2048
Generating RSA private key, 2048 bit long modulus
.....................+++
.................................+++
e is 65537 (0x10001)
```

生成 Server 密钥。

```
[ req ]
default_bits       = 2048
distinguished_name = req_distinguished_name
req_extensions     = req_ext

[ req_distinguished_name ]
countryName                 = Country Name (2 letter code)
countryName_default         = CN
stateOrProvinceName         = State or Province Name (full name)
stateOrProvinceName_default = JiangSu
localityName                = Locality Name (eg, city)
localityName_default        = NanJing
organizationName            = Organization Name (eg, company)
organizationName_default    = SEU
commonName                  = scj
commonName_max              = 64
commonName_default          = www.scj2020.com

[ req_ext ]
subjectAltName = @alt_names

[alt_names]
DNS.1   = www.scj-go.com
DNS.2   = www.scj2020.com
IP      = 192.168.43.150
```

```
[09/17/20]seed@VM:~/.../cert_test$ openssl req -new -sha256 -out servercsr.pem -
key serverkey.pem -config server.conf
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [CN]:
State or Province Name (full name) [JiangSu]:
Locality Name (eg, city) [NanJing]:
Organization Name (eg, company) [SEU]:
scj [www.scj2020.com]:
```

同理写好 server.conf 配置文件然后运行上述命令。

```
[09/17/20]seed@VM:~/.../cert_test$ openssl x509 -req -days 3650 -CA cacrt.pem -C
Akey cakey.pem -CAcreateserial -in servercsr.pem -out servercrt.pem -extensions
req_ext -extfile server.conf
Signature ok
subject=/C=CN/ST=JiangSu/L=NanJing/O=SEU/CN=www.scj2020.com
Getting CA Private Key
[09/17/20]seed@VM:~/.../cert_test$ ls
ca.conf    cacrt.srl  cakey.pem    servercrt.pem  serverkey.pem
cacrt.pem  cacsr.pem  server.conf  servercsr.pem
```

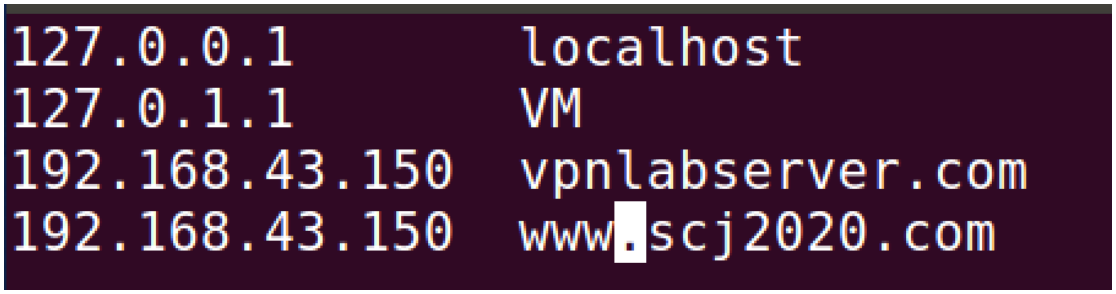
用 CA 证书生成终端用户证书，得到 server.crt。

```
[09/17/20]seed@VM:~/.../ca_client$ openssl x509 -in servercrt.pem -noout -subjec
t_hash
473bc94b
```

生成 hash。

```
[09/17/20]seed@VM:~/.../ca_client$ ln -s servercrt.pem 473bc94b.0
```

在主机 U 中输入上述命令，建立 server 证书的软链接。



在主机 U 的/etc/hosts 中添加上图第四行。

```
[09/17/20]seed@VM:~/.../tls$ sudo ./tlsclient www.scj2020.com 4433
3073877696:error:14094410:SSL routines:ssl3_read_bytes:sslv3 alert handshake fai
lure:s3_pkt.c:1487:SSL alert number 40
```

不幸的是，握手时出现了错误，没有找到错误原因。

Task 5 和 Task 6 目前没有能力完成 QAQ。