Final Lab
Virtual Private Network (VPN) Lab

Author：57119108 吴桐，57119111 唐翠霜

Date：2022.9.1

Lab Tasks

Task 1: VM Setup

在本次实验中，我们将搭建一个 Linux 系统下的 VPN。我们需要在客户端和网关之间构建一个 VPN 隧道，允许主机能够通过网关安全地访问专有网络。我们至少需要三台主机：VPN 客户端（主机 U），VPN 服务器（网关），专有网络内的主机 V。网络拓扑如图 1 所示。

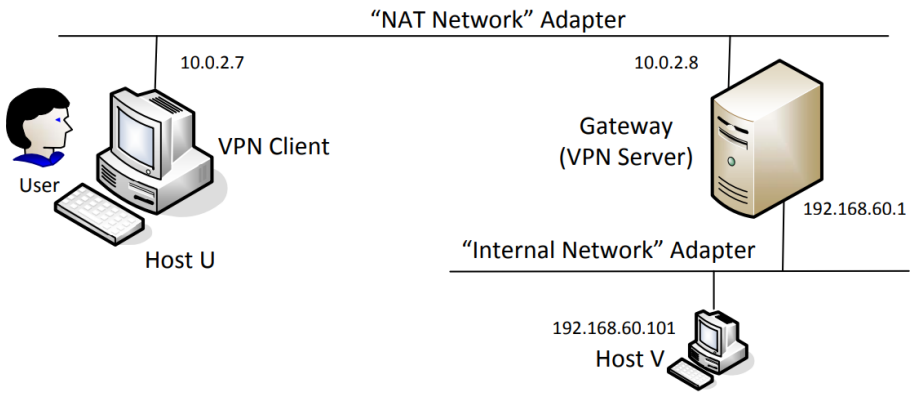

图 1

我们这里使用容器来搭建实验环境，具体设置见 docker-compose.yml。在搭建好环境后，我们可以利用 ping 测试一下网络环境。从主机 V 上 ping 网关，结果如图 2 所示。

```
root@83455ebc0426:/# 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=0.105 ms
64 bytes from 192.168.60.1: icmp_seq=2 ttl=64 time=0.125 ms
64 bytes from 192.168.60.1: icmp_seq=3 ttl=64 time=0.112 ms
```

图 2

从主机 U 上 ping 网关，结果如图 3 所示。

```
root@ed8df028d5eb:/# 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=0.162 ms
64 bytes from 10.0.2.8: icmp_seq=2 ttl=64 time=0.129 ms
64 bytes from 10.0.2.8: icmp_seq=3 ttl=64 time=0.077 ms
```

图 3

从主机 U 上 ping 主机 V，结果如图 4 所示。

```
root@ed8df028d5eb:/# ping 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 56(84) bytes of data.
^C
--- 192.168.60.101 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2031ms
```

图 4

从 VPN 服务器上 ping 主机 V，结果如图 5 所示。

```
root@ffaece68ed2a:/# 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=0.088 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=64 time=0.065 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=64 time=0.149 ms
```

图 5

从 VPN 服务器上 ping 主机 U，结果如图 6 所示。

```
root@ffaece68ed2a:/# ping 10.0.2.7
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=64 time=0.122 ms
64 bytes from 10.0.2.7: icmp_seq=2 ttl=64 time=0.143 ms
64 bytes from 10.0.2.7: icmp_seq=3 ttl=64 time=0.131 ms
```

图 6

Task 2: Creating a VPN Tunnel using TUN/TAP

TUN 和 TAP 是虚拟网络内核驱动程序，能够实现软件支持的网络设备。TAP 模拟以太网设备，使用第二层数据包进行操作；TUN 模拟网络层设备，使用第三层数据包进行操作。使用 TUN/TAP 技术，我们可以创建一个虚拟网络接口。一方面，内核通过该接口发送的数据包会被传递给用户程序；另一方面，用户程序写入该接口的数据会被当作网络数据包进入操作系统内核。

在此任务中，客户端代码为 vpn_client_task2.py，服务器代码为 vpn_server_task2.py。首先在服务器上运行服务程序 vpn_server_task2.py，之后在主机 U 运行客户端程序 vpn_client_task2.py。由于我们在配置文件中，已经规定了主机 U 和主机 V 上的路由信息：主机 U 上发往 192.168.60.0/24 网段的数据包会被转发到 VPN 隧道端口 cocot0 上，主机 V 上发出的数据包会被转发到网关 10.0.2.8（VPN 服务器）上，因此我们不再需要配置路由信息。

在主机 U 上 ping 主机 V，结果如图 7 所示。同时，客户端和服务器也产生了相应的转发信息，如图 8~9 所示。

```
root@ed8df028d5eb:/# 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=4.79 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=7.02 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=7.21 ms
```

图 7

```
root@ed8df028d5eb:/volumes# vpn_client.py
Interface Name: cocot0
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
```

图 8

```
root@ffaece68ed2a:/volumes# vpn_server.py
Interface Name: cocot0
From socket <==: 192.168.53.99 --> 192.168.60.101
From tun ==>: 192.168.60.101 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.101
From tun ==>: 192.168.60.101 --> 192.168.53.99
From socket <==: 192.168.53.99 --> 192.168.60.101
From tun ==>: 192.168.60.101 --> 192.168.53.99
```

图 9

我们可以使用 WireShark 嗅探网络上的数据包，结果如图 10 所示。可以看到，ping 命令大致可以分为四个过程：主机 U（客户端）向服务器发送请求→服务器将请求转发主机 V→主机 V 向服务器发送回复→服务器将回复返回给主机 U。

其中数据包 14 是 ping 命令产生的，由于路由的配置，ICMP 包被转发到 TUN 接口，这就是为什么这个数据包的 IP 源地址使用的是 TUN 接口的 IP 地址 192.168.53.5。隧道应用程序得到 ICMP 包后，把它传入隧道，即把它放入 UDP 包中发送给 VPN 服务器（10.0.2.8）。因为这个 UDP 包是从 VPN 客户端的正常接口发出的，所以它的源 IP 地址为 10.0.2.7，也就是 VPN 客户端的地址。

从 VPN 服务器返回的 UDP 包中封装了来自 192.168.60.101 的 ICMP 响应。VPN 客户端的隧道应用程序得到这个数据包后，把封装的 ICMP 包取出，通过 TUN 接口传递给内核，这就变成了数据包 15。这时，计算机意识到目标 IP 地址 192.168.53.5 是它自己，因此把 ICMP 响应传递给 ping 程序。

No.	Time	Source	Destination	Protocol	Length	Info
13	2022-08-26 23:27:43.968671569	192.168.53.5	192.168.60.101	ICMP	100	Echo (ping) request id=0x001d,
14	2022-08-26 23:27:43.968742556	192.168.53.5	192.168.60.101	ICMP	100	Echo (ping) request id=0x001d,
15	2022-08-26 23:27:43.968767341	192.168.60.101	192.168.53.5	ICMP	100	Echo (ping) reply id=0x001d,
16	2022-08-26 23:27:43.968771885	192.168.60.101	192.168.53.5	ICMP	100	Echo (ping) reply id=0x001d,
17	2022-08-26 23:27:43.969514280	10.0.2.8	10.0.2.7	UDP	128	9090 → 9090 Len=84
18	2022-08-26 23:27:43.969523764	10.0.2.8	10.0.2.7	UDP	128	9090 → 9090 Len=84
19	2022-08-26 23:27:44.969087940	10.0.2.7	10.0.2.8	UDP	128	9090 → 9090 Len=84
20	2022-08-26 23:27:44.969123710	10.0.2.7	10.0.2.8	UDP	128	9090 → 9090 Len=84
21	2022-08-26 23:27:44.970553132	192.168.53.5	192.168.60.101	ICMP	100	Echo (ping) request id=0x001d,
22	2022-08-26 23:27:44.970577256	192.168.53.5	192.168.60.101	ICMP	100	Echo (ping) request id=0x001d,
23	2022-08-26 23:27:44.970608291	192.168.60.101	192.168.53.5	ICMP	100	Echo (ping) reply id=0x001d,
24	2022-08-26 23:27:44.970617047	192.168.60.101	192.168.53.5	ICMP	100	Echo (ping) reply id=0x001d,
25	2022-08-26 23:27:44.971964228	10.0.2.8	10.0.2.7	UDP	128	9090 → 9090 Len=84
26	2022-08-26 23:27:44.971989682	10.0.2.8	10.0.2.7	UDP	128	9090 → 9090 Len=84
27	2022-08-26 23:27:45.970761595	10.0.2.7	10.0.2.8	UDP	128	9090 → 9090 Len=84
28	2022-08-26 23:27:45.970795274	10.0.2.7	10.0.2.8	UDP	128	9090 → 9090 Len=84

图 10

在主机 U 上使用 telnet 指令连接主机 V，结果如图 11 所示。可见此时主机 U 可以成功连接到主机 V，并可以在得到的主机 V 的终端上任意输入指令。

```
root@ed8df028d5eb:/# telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
83455ebc0426 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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@83455ebc0426:~$ pwd
/home/seed
```

图 11

此时，终止客户端程序，尝试继续在终端上输入指令，可以发现终端没有任何反应。重新启动客户端程序 vpn_client_task2.py，之前输入的指令出现在终端并正常执行（图 12）。之所以出现这样的现象，是因为 VPN 的建立对于上层应用来说是透明的，断开连接后，应用不会立刻终止，输入的内容会保存在缓冲区内，当连接重新建立后，缓冲区内的内容会被立即发送。

```
seed@83455ebc0426:~$ pwd
/home/seed
seed@83455ebc0426:~$ cd ..
seed@83455ebc0426:/home$ ls
seed
seed@83455ebc0426:/home$ pwd
/home
seed@83455ebc0426:/home$ cd ..
seed@83455ebc0426:/$ ls
bin   dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt  root  sbin  sys  usr
seed@83455ebc0426:/$
seed@83455ebc0426:/$ pwd
/
```

图 12

Task 3: Encrypting the Tunnel

在此任务中，我们需要使用 TLS 技术对 VPN 隧道进行保护，保证通信的保密性和完整性。保密性可以通过对隧道内的内容加密来实现，完整性可以使用消息认证码（MAC）实现。以上两个安全目标都可以使用 TLS 技术实现。

TLS 是建立在 TCP 协议上的，因此我们需要在客户端与服务器之间建立一个 TCP 连

接，在此之上附加 TLS 功能。

我们先解决建立 TCP 连接的问题。客户端代码为 vpn_client_task3.py，服务器代码为 vpn_server_task3.py。这里我们简单解释一下代码的逻辑和关键部分。

在客户端，我们通过以下代码建立 TCP 连接，隧道的数据通过 sock 对象进行收发。

```
# Create TCP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
SERVER_IP, SERVER_PORT = '10.0.2.8', 4433
sock.connect((SERVER_IP, SERVER_PORT))
```

图 13

在服务器，我们通过以下代码建立 TCP 连接，绑定相应的端口后持续监听是否有进程试图连接该端口。

```
# Create TCP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

IP_A = '10.0.2.8'
PORT = 4433
sock.bind((IP_A, PORT))
sock.listen(5)
```

图 14

服务器使用 select 函数监听 inputs 列表中的对象。如果收到了来自 sock 对象的数据，则表明有客户端连接到监听端口，我们将此连接 con 保存下来，并加入到 inputs 列表中，作为新的监听对象。如果收到了来自 con 的数据，则表明客户端与服务器之间开始正常传输数据包，通信的代码逻辑与 Task 2 中相同。

```
inputs = [sock,tun]
outputs = sock

while True:
    ready, _, _ = select.select(inputs, [], [])

    for fd in ready:
        if fd is sock:
            con, addr = sock.accept()
            con.setblocking(0)
            inputs.append(con)
            outputs = con
        elif fd is tun:
            packet = os.read(tun, 2048)
            pkt = IP(packet)
            print("From tun ==>: {} --> {}".format(pkt.src, pkt.dst))
            outputs.send(packet)
        else:
            data = fd.recv(2048)
            if data != b'':
                pkt = IP(data)
                print("From socket <==: {} --> {}".format(pkt.src, pkt.dst))
                os.write(tun, bytes(pkt))
            else:
                print("Closing {}".format(addr))
                inputs.remove(fd)
                fd.close()
```

图 15

首先在服务器上运行服务程序 vpn_server_task3.py，之后在主机 U 运行客户端程序

vpn_client_task3.py。在主机 U 上 ping 主机 V，结果如图 16 所示。同时，客户端和服务器也产生了相应的转发信息，如图 17~18 所示。

```
root@7c687bae7d2f:/# 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=3.54 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=4.81 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=5.85 ms
```

图 16

```
root@7c687bae7d2f:/volumes# vpn_client_task3.py
Interface Name: cocot0
From tun ==>: 192.168.53.5 --> 192.168.60.101
From socket <==: 192.168.60.101 --> 192.168.53.5
From tun ==>: 192.168.53.5 --> 192.168.60.101
From socket <==: 192.168.60.101 --> 192.168.53.5
From tun ==>: 192.168.53.5 --> 192.168.60.101
From socket <==: 192.168.60.101 --> 192.168.53.5
```

图 17

```
root@49109ce9f8fa:/volumes# vpn_server_task3.py
Interface Name: cocot0
From socket <==: 192.168.53.5 --> 192.168.60.101
From tun ==>: 192.168.60.101 --> 192.168.53.5
From socket <==: 192.168.53.5 --> 192.168.60.101
From tun ==>: 192.168.60.101 --> 192.168.53.5
From socket <==: 192.168.53.5 --> 192.168.60.101
From tun ==>: 192.168.60.101 --> 192.168.53.5
```

图 18

至此，我们成功建立了 TCP 连接并实现了 VPN 通信，TLS 的加密功能我们将在 Task 4 中与服务器验证功能一起实现。

Task 4: Authenticating the VPN Server

在此任务中，我们将进行服务器的身份验证。服务器身份验证有三个重要步骤：

（1）验证服务器证书是否有效；

（2）验证服务器是证书的所有者；

（3）验证服务器是否是预期的服务器（例如，如果用户希望访问 example.com，我们需要确保服务器确实是 example.com，而不是其他站点）。

需要注意的是，我们的 VPN 程序应该能够与不同机器上的 VPN 服务器通信，因此我们需要从命令行键入主机名与端口。我们需要根据主机名查找服务器的 IP 地址，并验证服务器的证书。

为了便于测试，我们在此任务中暂时将 VPN 服务器的 IP 地址（10.0.2.8）硬编码到程序中。用户自定义主机名与端口的问题将在 Task 5 中解决。

在进行服务器身份验证时，我们需要颁发服务器证书的 CA 证书。我们在./ca-client 文件夹中放置可信 CA 证书，即我们在 PKI 实验中创建的 ModelCA 自签发证书。如图 19~20 所示，我们还可以加入其他 CA 证书，以便在连接其他 VPN 服务器时能够完成服务器验证。

```
[08/26/22]seed@VM:~/.../ca-client$ sudo openssl x509 -in GeoTrust_Global_CA.pem -noout -subject_hash
2c543cd1
[08/26/22]seed@VM:~/.../ca-client$ sudo ln -s GeoTrust_Global_CA.pem 2c543cd1.0
```

图 19

```
07ec5d08.0 -> ca.crt
2c543cd1.0 -> GeoTrust_Global_CA.pem
3513523f.0 -> DigiCert_Global_Root_CA.pem
5ad8a5d6.0 -> GlobalSign_Root_CA.pem
```

图 20

之后，我们使用 ModelCA 自签发证书对服务器颁布证书，服务器的主机名为 www.cocotvpn.com，相应的配置信息如图 21 所示，颁布证书的命令如图 22 所示。

```
1 [ req ]
2 prompt = no
3 distinguished_name = req_distinguished_name
4 req_extensions = req_ext
5
6 [ req_distinguished_name ]
7 C = CN
8 ST = Jiangsu
9 L = Nanjing
10 O = CocoT LTD.
11 CN = www.cocotvpn.com
12
13 [ req_ext ]
14 subjectAltName = @alt_names
15
16 [alt_names]
17 DNS.1 = www.cocotvpn.com
18 DNS.2 = www.cocotvpn.net
19 DNS.3 = *.cocotvpn.com
```

图 21

```
root@49109ce9f8fa:/volumes# mkdir -p ./demoCA/newcerts
root@49109ce9f8fa:/volumes# touch ./demoCA/index.txt
root@49109ce9f8fa:/volumes# touch ./demoCA/serial
root@49109ce9f8fa:/volumes# echo '1000' > ./demoCA/serial
root@49109ce9f8fa:/volumes# openssl req -newkey rsa:2048 -config ./server_openssl.cnf -batch
 -sha256 -keyout server.key -out server.csr
Generating a RSA private key
..+++++
.............+++++
writing new private key to 'server.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
root@49109ce9f8fa:/volumes# openssl ca -md sha256 -days 3560 -config ./myCA_openssl.cnf -pol
icy policy_anything -batch -in server.csr -out server.crt -cert ca.crt -keyfile ca.key
Using configuration from ./myCA_openssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4096 (0x1000)
        Validity
            Not Before: Aug 25 12:36:02 2022 GMT
            Not After : May 24 12:36:02 2032 GMT
        Subject:
            countryName               = CN
            stateOrProvinceName       = Jiangsu
            localityName              = Nanjing
            organizationName          = CocoT LTD.
            commonName                = www.cocotvpn.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                6B:18:F4:6F:C3:C2:AB:A5:96:08:E9:ED:8D:FB:B3:2A:69:E7:A9:4E
            X509v3 Authority Key Identifier:
                keyid:CC:0D:1D:BD:BC:7C:A1:25:0A:F5:C1:78:F5:1C:25:7E:45:1F:61:A9

            X509v3 Subject Alternative Name:
                DNS:www.cocotvpn.com, DNS:www.cocotvpn.net, DNS:*.cocotvpn.com
Certificate is to be certified until May 24 12:36:02 2032 GMT (3560 days)

Write out database with 1 new entries
Data Base Updated
```

图 22

我们的客户端代码为 vpn_tls_client.py，服务器代码为 vpn_tls_server.py。首先在服务器上运行服务程序 vpn_tls_server.py，之后在主机 U 运行客户端程序 vpn_tls_client.py。注意，我们在运行客户端程序时，需要输入主机名，用于对证书的所有者进行认证。在主机 U 上 ping 主机 V，结果如图 23 所示。同时，客户端和服务器也产生了相应的转发信息，如图 24~25 所示。使用 WireShark 抓取的 TLS 加密流量信息如图 26 所示。

```
root@3d35108ebe9b:/# 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=2.49 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=5.08 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=3.09 ms
64 bytes from 192.168.60.101: icmp_seq=4 ttl=63 time=6.52 ms
```

图 23

```
root@3d35108ebe9b:/volumes# vpn_client_task3.py www.cocotvpn.com
Interface Name: cocot0
TLS handshake Finish.
From tun ==>: 192.168.53.5 --> 192.168.60.101
From socket <==: 192.168.60.101 --> 192.168.53.5
From tun ==>: 192.168.53.5 --> 192.168.60.101
From socket <==: 192.168.60.101 --> 192.168.53.5
From tun ==>: 192.168.53.5 --> 192.168.60.101
From socket <==: 192.168.60.101 --> 192.168.53.5
```

图 24

```
root@31983c5566e4:/volumes# vpn_server_task3.py
Interface Name: cocot0
Enter PEM pass phrase:
From socket <==: 192.168.53.5 --> 192.168.60.101
From tun ==>: 192.168.60.101 --> 192.168.53.5
From socket <==: 192.168.53.5 --> 192.168.60.101
From tun ==>: 192.168.60.101 --> 192.168.53.5
From socket <==: 192.168.53.5 --> 192.168.60.101
From tun ==>: 192.168.60.101 --> 192.168.53.5
```

图 25

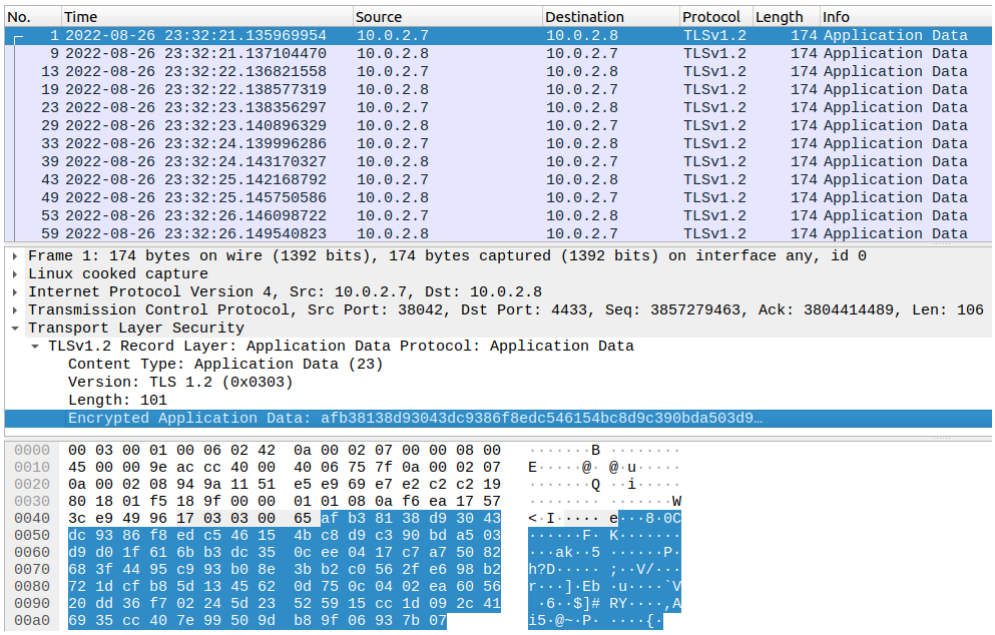

图 26

在主机 U 上使用 telnet 指令连接主机 V，结果如图 27 所示。可见此时主机 U 可以成功连接到主机 V，并可以在得到的主机 V 的终端上任意输入指令。

```
root@3d35108ebe9b:/# telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
438c04be4ac5 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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

图 27

如果我们启动客户端时输入的服务器并不是预期的服务器，程序在验证服务器证书阶段就会失败。如图 28 所示，在 docker-compose.yml 中，我们将 www.cocot2022.com 映射到了服务器的 IP 地址上。但是我们在服务器上提供的证书是颁发给 www.cocotvpn.com 的，如果在客户端输入了 www.cocot2022.com，这个不匹配就会导致程序终止并报错（图 29）。

```
extra_hosts:
        - "www.cocotvpn.com:10.0.2.8"
        - "www.cocot2022.com:10.0.2.8"
```

图 28

```
root@baef1b684948:/volumes# vpn_tls_client.py www.cocot2022.com 4433
Username:seed
Password:
Login Succeed!
Interface Name: cocot0
10.0.2.8
Traceback (most recent call last):
  File "./vpn_tls_client.py", line 85, in <module>
    ssock.do_handshake()   # Start the handshake
  File "/usr/lib/python3.8/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed:
 Hostname mismatch, certificate is not valid for 'www.cocot2022.com'. (_ssl.c:1123)
```

图 29

Task 5: Authenticating the VPN Client

在此任务中，我们将实现对用户的身份认证。访问专用网络内的计算机是授权用户的特权，我们需要在服务器上验证用户身份，确保请求连接的用户是在 VPN 服务器上拥有有效帐户的用户。我们将使用标准密码身份验证来验证用户。当用户与 VPN 服务器建立 VPN 隧道时，服务器会要求用户提供用户名和密码。服务器将检查其影子文件（/etc/shadow），如果找到匹配的记录，则认证成功并建立 VPN 隧道；如果没有匹配的记录，服务器将断开与用户的连接。

我们的客户端代码为 vpn_tls_client_login.py，服务器代码为 vpn_tls_server_login.py。在客户端代码中，我们修复了 Task 4 中遗留的问题，在启动客户端时要求用户输入主机名和端口号，在客户端使用 socket.gethostbyname()得到主机名对应的 IP 地址。

首先在服务器上运行服务程序 vpn_tls_server_login.py，之后在主机 U 运行客户端程序 vpn_tls_client_login.py。注意，我们在运行客户端程序时，需要输入主机名与端口号。

如果我们在客户端输入了有效的用户名及密码，服务器和用户端均会提示我们登陆成功（图 30），之后就可以使用 VPN 实现主机 U 和主机 V 之间通信。如果我们在客户端提供了错误的用户信息，服务器会返回错误原因，并中断与客户端的连接（图 31）。

```
=== Login
Username: seed
Password:
Login Succeed!
```

图 30

```
=== Login
Username: cocot
Password:
Login Fail: user 'cocot' not found.
```

图 31

成功登录后，在主机 U 上 ping 主机 V，结果如图 32 所示。可见此时主机 U 与主机 V 之间能够正常通信。

```
root@cbe58bd3c65f:/# 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=3.72 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=5.71 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=5.94 ms
```

图 32

成功登陆后，在主机 U 上使用 telnet 指令连接主机 V，结果如图 33 所示。可见此时主机 U 可以成功连接到主机 V，并可以在得到的主机 V 的终端上任意输入指令。

```
root@cbe58bd3c65f:/# telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
57593de1ddbf 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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@57593de1ddbf:~$ pwd
/home/seed
seed@57593de1ddbf:~$ cd ..
seed@57593de1ddbf:/home$ ls
seed
```

图 33

我们也可以尝试连接其他服务器，图 34 展示了客户端连接 www.baidu.com 的结果。

```
root@cdf21ebb07eb:/volumes# vpn_tls_client_login.py www.baidu.com 443
=== Get Server IP: 36.152.44.96
=== Interface Name: cocot0
=== Cipher used: ('ECDHE-RSA-AES128-GCM-SHA256', 'TLSv1.2', 128)
=== Server hostname: www.baidu.com
=== Server certificate:
{'OCSP': ('http://ocsp.globalsign.com/gsrsaovsslca2018',),
 'caIssuers': ('http://secure.globalsign.com/cacert/gsrsaovsslca2018.crt',),
 'crlDistributionPoints': ('http://crl.globalsign.com/gsrsaovsslca2018.crl',),
 'issuer': ((('countryName', 'BE'),),
            (('organizationName', 'GlobalSign nv-sa'),),
            (('commonName', 'GlobalSign RSA OV SSL CA 2018'),)),
 'notAfter': 'Aug  6 05:16:01 2023 GMT',
 'notBefore': 'Jul  5 05:16:02 2022 GMT',
 'serialNumber': '4417CE86EF82EC6921CC6F68',
 'subject': ((('countryName', 'CN'),),
             (('stateOrProvinceName', 'beijing'),),
             (('localityName', 'beijing'),),
             (('organizationalUnitName', 'service operation department'),),
             (('organizationName',
               'Beijing Baidu Netcom Science Technology Co., Ltd'),),
             (('commonName', 'baidu.com'),)),
 'subjectAltName': (('DNS', 'baidu.com'),
                    ('DNS', 'baifubao.com'),
                    ('DNS', 'www.baidu.cn'),
                    ('DNS', 'www.baidu.com.cn'),
                    ('DNS', 'mct.y.nuomi.com'),
                    ('DNS', 'apollo.auto'),
                    ('DNS', 'dwz.cn'),
                    ('DNS', '*.baidu.com'),
                    ('DNS', '*.baifubao.com'),
                    ('DNS', '*.baidustatic.com'),
                    ('DNS', '*.bdstatic.com'),
                    ('DNS', '*.bdimg.com'),
                    ('DNS', '*.hao123.com'),
                    ('DNS', '*.nuomi.com'),
                    ('DNS', '*.chuanke.com'),
                    ('DNS', '*.trustgo.com'),
                    ('DNS', '*.bce.baidu.com'),
                    ('DNS', '*.eyun.baidu.com'),
                    ('DNS', '*.map.baidu.com'),
                    ('DNS', '*.mbd.baidu.com'),
                    ('DNS', '*.im.baidu.com'),
                    ('DNS', '*.baiducontent.com'),
                    ('DNS', '*.dlnel.com'),
                    ('DNS', '*.dlnel.org'),
                    ('DNS', '*.dueros.baidu.com'),
                    ('DNS', '*.su.baidu.com'),
                    ('DNS', '*.91.com'),
                    ('DNS', '*.hao123.baidu.com'),
                    ('DNS', '*.apollo.auto'),
                    ('DNS', '*.xueshu.baidu.com'),
                    ('DNS', '*.bj.baidubce.com'),
                    ('DNS', '*.gz.baidubce.com'),
                    ('DNS', '*.smartapps.cn'),
                    ('DNS', '*.bdtjrcv.com'),
                    ('DNS', '*.hao222.com'),
                    ('DNS', '*.haokan.com'),
                    ('DNS', '*.pae.baidu.com'),
                    ('DNS', '*.vd.bdstatic.com'),
                    ('DNS', '*.cloud.baidu.com'),
                    ('DNS', 'click.hm.baidu.com'),
                    ('DNS', 'log.hm.baidu.com'),
                    ('DNS', 'cm.pos.baidu.com'),
                    ('DNS', 'wn.pos.baidu.com'),
                    ('DNS', 'update.pan.baidu.com')),
 'version': 3}
[{'issuer': ((('countryName', 'BE'),),
             (('organizationName', 'GlobalSign nv-sa'),),
             (('organizationalUnitName', 'Root CA'),),
             (('commonName', 'GlobalSign Root CA'),))),
  'notAfter': 'Jan 28 12:00:00 2028 GMT',
  'notBefore': 'Sep  1 12:00:00 1998 GMT',
  'serialNumber': '040000000001154B5AC394',
  'subject': ((('countryName', 'BE'),),
              (('organizationName', 'GlobalSign nv-sa'),),
              (('organizationalUnitName', 'Root CA'),),
              (('commonName', 'GlobalSign Root CA'),))),
  'version': 3}]
=== TLS handshake Finish.
```

图 34

如图 35 所示，我们尝试从主机 U 向主机 V 发送超长数据包。

```
root@cdf21ebb07eb:/# ping -s 3001 192.168.60.101
PING 192.168.60.101 (192.168.60.101) 3001(3029) bytes of data.
3009 bytes from 192.168.60.101: icmp_seq=1 ttl=63 time=10.7 ms
3009 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=12.7 ms
3009 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=16.7 ms
```

图 35

客户端和服务器产生了相应的转发信息，如图 36~37 所示。使用 WireShark 嗅探数据包，结果如图 38 所示。可以看到数据被分片传输，在目标主机上重组为完整的 ICMP 报文后递交到 ping 程序。

```
=== Login
Username: seed
Password:
Login Succeed!
From tun ==>: 192.168.53.5 --> 192.168.60.101
From tun ==>: 192.168.53.5 --> 192.168.60.101
From tun ==>: 192.168.53.5 --> 192.168.60.101
From socket <==: 192.168.60.101 --> 192.168.53.5
From socket <==: 192.168.60.101 --> 192.168.53.5
From socket <==: 192.168.60.101 --> 192.168.53.5
```

图 36

```
root@f90718ec075c:/volumes# vpn_tls_server_login.py
Interface Name: cocot0
Enter PEM pass phrase:
Login Succeed!
From socket <==: 192.168.53.5 --> 192.168.60.101
From socket <==: 192.168.53.5 --> 192.168.60.101
From socket <==: 192.168.53.5 --> 192.168.60.101
From tun ==>: 192.168.60.101 --> 192.168.53.5
From tun ==>: 192.168.60.101 --> 192.168.53.5
From tun ==>: 192.168.60.101 --> 192.168.53.5
```

图 37

No.	Time	Source	Destination	Protocol	Length	Info
56	2022-08-27 04:04:32.156439779	192.168.53.5	192.168.60.101	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=0, ID=8702) [Reassembled in #58]
57	2022-08-27 04:04:32.156440611	192.168.53.5	192.168.60.101	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=8702) [Reassembled in #58]
58	2022-08-27 04:04:32.156441322	192.168.53.5	192.168.60.101	ICMP	85	Echo (ping) request id=0x0041, seq=1/256, ttl=63 (no response found!)
59	2022-08-27 04:04:32.156466082	192.168.53.5	192.168.60.101	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=0, ID=8702) [Reassembled in #61]
60	2022-08-27 04:04:32.156467671	192.168.53.5	192.168.60.101	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=8702) [Reassembled in #61]
61	2022-08-27 04:04:32.156468793	192.168.53.5	192.168.60.101	ICMP	85	Echo (ping) request id=0x0041, seq=1/256, ttl=63 (reply in 64)
62	2022-08-27 04:04:32.156496739	192.168.60.101	192.168.53.5	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=0, ID=6c31) [Reassembled in #64]
63	2022-08-27 04:04:32.156497652	192.168.60.101	192.168.53.5	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6c31) [Reassembled in #64]
64	2022-08-27 04:04:32.156498377	192.168.60.101	192.168.53.5	ICMP	85	Echo (ping) reply id=0x0041, seq=1/256, ttl=64 (request in 61)
65	2022-08-27 04:04:32.156508368	192.168.60.101	192.168.53.5	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=0, ID=6c31) [Reassembled in #67]
66	2022-08-27 04:04:32.156509640	192.168.60.101	192.168.53.5	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6c31) [Reassembled in #67]
67	2022-08-27 04:04:32.156510715	192.168.60.101	192.168.53.5	ICMP	85	Echo (ping) reply id=0x0041, seq=1/256, ttl=64

图 38

接下来，我们简单解释一下代码的关键部分。

如图 39 所示，在客户端程序中，行 40 创建了一个新的 SSL 上下文，传输协议定义为 TLS；行 42 加载 CA 证书，capath 定义了证书的目录；行 43 定义了认证模式，该模式需要从套接字连接的另一端获得证书，如果未提供证书或证书验证失败，将引发 SSL 错误；行 44 用于验证主机名是否与证书相匹配。行 51 返回一个 SSL 套接字，绑定相应的上下文信息和证书等。其中，参数 server_hostname 指定连接的服务器主机名；参数 do_handshake_on_connect 指定在执行 socket.connect()后是否自动执行 SSL 握手，若值为 False 则需要显示调用 do_handshake()函数完成握手。

客户端将在握手阶段实现服务器证书的验证，如果验证失败，客户端程序就会立刻终止并报错。

```
# cadir = '/etc/ssl/certs'
cadir = './ca-client'
# Set up the TLS context
context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)  # For Ubuntu 20.04 VM

context.load_verify_locations(capath=cadir)
context.verify_mode = ssl.CERT_REQUIRED  # load CA CRT
context.check_hostname = True  # Step 3

# Create TCP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((SERVER_IP, SERVER_PORT))

# Add the TLS
ssock = context.wrap_socket(sock, server_hostname=hostname,
                            do_handshake_on_connect=False)
ssock.do_handshake()   # Start the handshake
```

图 39

如图 40 所示，客户端将用户名和密码通过安全隧道传输给 VPN 服务器。

```
# Login
print("=== Login")
username = input("Username: ")
password = getpass.getpass()
ssock.send(username.encode())
ssock.send(password.encode())
```

图 40

如图 41 所示，服务器在收到客户端提供的用户名和密码后，调用 login 函数利用影子文件/etc/shadow 实现用户的身份认证。

```
def login(username, password):
    try:
        enc_pwd = spwd.getspnam(username)[1]
        if enc_pwd in ["NP", "!", "", None]:
            return "user '%s' has no password set" % username
        if enc_pwd in ["LK", "*"]:
            return "account is locked"
        if enc_pwd == "!!":
            return "password has expired"
        # Encryption happens here, the hash is stripped from the
        # enc_pwd and the algorithm id and salt are used to encrypt
        # the password.
        if crypt.crypt(password, enc_pwd) == enc_pwd:
            return True
        else:
            return "incorrect password"
    except KeyError:
        return "user '%s' not found" % username
    return "unknown error"
```

图 41

Summary

本次实验还是有一定难度的，需要在充分理解 VPN，TCP，TLS，PKI 等技术的基础上融会贯通。在实验过程中，我们也遇到了不少困难，例如在加入 TLS 功能后，主机 U 与主机 V 无法实现通信，后来才明白是因为 TLS 的版本问题导致的接收数据阻塞，有两个解决方法：一是降低使用的 TLS 版本，二是在服务器端的代码中将参数 context.num_tickets 的值设为 0。在老师和同学们的帮助下，我们顺利完成了本次实验。