华中科技大学

“计算机网络安全”实验报告

题目： VPN 实验

院　　系　网络空间安全学院

专业班级　信息安全 1801

姓　　名　杨傲

学　　号　U201814834

日　　期　2021 年　5　月

评分项	实验报告评分（40%）	检查单分数（60%）	综合得分	教师签名
得分				

实验报告评分标准

评分项目	分值	评分标准	得分
实验原理	20	18-20 系统流程清晰，报文处理过程描述清楚； 15-17 系统流程比较清晰，报文处理过程描述比较清楚； 12 以下 描述简单	
实验步骤	30	25-30 实验步骤描述详细、清楚、完整，前后关系清晰； 18-24 实验步骤描述比较清楚，关键步骤都进行了描述 18 分以下，实验步骤描述比较简单或不完整	
结果验证与分析	20	16-20，任务完成，针对任务点的测试，对结果有分析 10-15，针对任务点的测试截图，没分析 10 分以下，测试很简单，没有覆盖任务点	
心得体会	10	8-10 有自己的真实体会 4-7 真实体会套话 3 分以下，没有写什么体会	
格式规范	10	图、表的说明，行间距、缩进、目录等，一种不规范扣 1 分	
实验思考	10	思考题的回答，以及其它的简介	
总 分			

目　　录

实验三 VPN 实验

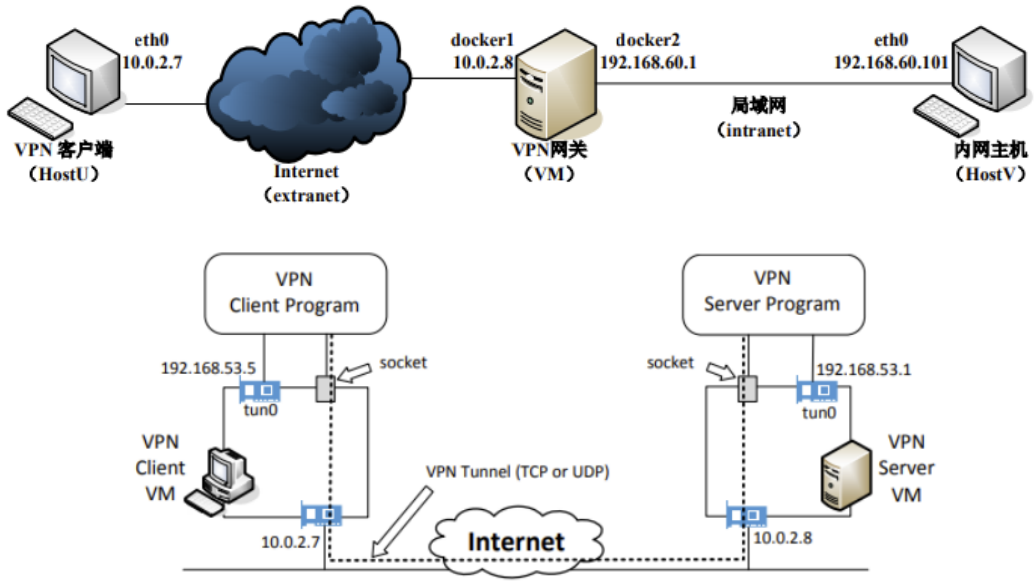
1 实验目的

虚拟专用网络（VPN）用于创建计算机通信的专用的通信域，或为专用网络到不安全的网络（如 Internet）的安全扩展。VPN 是一种被广泛使用的安全技术。 在 IPSec 或 TLS/SSL（传输层安全性/安全套接字层）上构建 VPN 是两种根本不同的方法。本实验中，我们重点关注基于 TLS/SSL 的 VPN。这种类型的 VPN 通常被称为 TLS/SSL VPN。本实验的学习目标是让学生掌握 VPN 的网络和安全技术。为实现这一目标，要求学生实现简单的 TLS/SSL VPN。虽然这个 VPN 很简单，但它包含了 VPN 的所有基本元素。TLS/SSL VPN 的设计和实现体现了许多安全原则，包括以下内容：

- 虚拟专用网络
- TUN/TAP 和 IP 隧道
- 路由
- 公钥加密，PKI 和 X.509 证书
- TLS/SSL 编程
- 身份认证

2 实验环境

VMware Workstation 虚拟机。
Ubuntu 16.04 操作系统（SEEDUbuntu16.04）。
openssl 软件，该软件包含头文件，库函数和命令。该软件包已经安装在上述 VM 镜像中



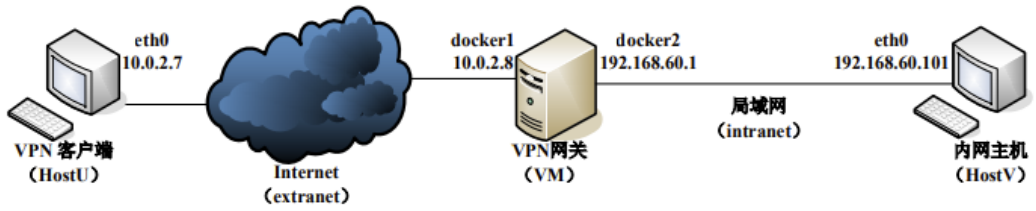
3 实验内容

本次实验需要为 Linux 操作系统实现一个简单的 VPN。
包括以下几个方面：
·网络环境搭建
·建立 VPN 隧道
·加密隧道
·身份认证
·多客户端支持

4 实验步骤及结果分析

4.1 配置环境

在计算机（客户端）和网关之间创建 VPN 隧道，允许计算机通过网 关安全地访问专用网络。我们使用虚拟机本身作为 VPN 服务器网关（VM），并 创建两个容器分别作为 VPN 客户端（HostU）和专用网络中的主机（HostV）。



在本实验中将这两台机器直接连接到同一 docker 网络“extranet”，模拟 Internet。第三台机器 HostV 是内部局域网的计算机。Internet 主机 HostU 上的用户希望通过 VPN 隧道与内部局域网的主机 HostV 通信。使用 docker 网络“intranet”将 HostV 与 VPN 服务器网关的内网口连接，模拟内部局域网。在这种设置中，HostV 不能直接从 Internet 访问，即不能直接从 HostU 访问。

使用以下命令在 vm 创建 extranet

sudo docker network create --subnet=10.0.2.0/24 --gateway=10.0.2.8 --opt "com.docker.network.bridge.name"="docker1" extranet

```
root@VM:/home/seed# sudo docker network create --subnet=10.0.2.0/24 --gateway=10
.0.2.8 --opt "com.docker.network.bridge.name"="docker1" extranet
fe908311e9f7f5d233930a28e9af23d6739f5eab3fcaabb7438a0c923581cb85
```

使用以下命令在 vm 创建 intranet

sudo docker network create --subnet=192.168.60.0/24 --gateway=192.168.60.1 --opt "com.docker.network.bridge.name"="docker2" intranet

```
root@VM:/home/seed# sudo docker network create --subnet=192.168.60.0/24 --gatewa
y=192.168.60.1 --opt "com.docker.network.bridge.name"="docker2" intranet
fc7207ea5ebb379c86504bf839a3255ef788e994d99326ae4cfea6a6c84c731e
```

此时查看 docker network ls 可以看到两个新添加的网络

```
root@VM:/home/seed# docker network ls
NETWORK ID          NAME                DRIVER
bcb53dfc0179        bridge              bridge
43d4114b55dd        none                null
000a7002a6fb        host                host
fe908311e9f7        extranet            bridge
fc7207ea5ebb        intranet            bridge
1c0ffbe05a76        mynetwork           bridge
```

新开一个终端创建以及运行容器 HostU

sudo docker run -it --name=HostU --hostname=HostU --net=extranet --ip=10.0.2.7 --privileged "seedubuntu" /bin/bash

```
root@VM:/home/seed# sudo docker run -it --name=HostU --hostname=HostU --net=extr
anet --ip=10.0.2.7 --privileged "seedubuntu" /bin/bash
root@HostU:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:0a:00:02:07
          inet addr:10.0.2.7  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::42:aff:fe00:207/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:48 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:6224 (6.2 KB)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@HostU:/#
```

新开一个终端创建以及运行容器 HostV

sudo docker run -it --name=HostV --hostname=HostV --net=intranet --ip=192.168.60.101 --privileged "seedubuntu" /bin/bash

```
root@VM:/home/seed# sudo docker run -it --name=HostV --hostname=HostV --net=intr
anet --ip=192.168.60.101 --privileged "seedubuntu" /bin/bash
root@HostV:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:c0:a8:3c:65
          inet addr:192.168.60.101  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::42:c0ff:fea8:3c65/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:44 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:5870 (5.8 KB)  TX bytes:578 (578.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@HostV:/#
```

在容器 HostU 和 HostV 内分别删除掉默认路由

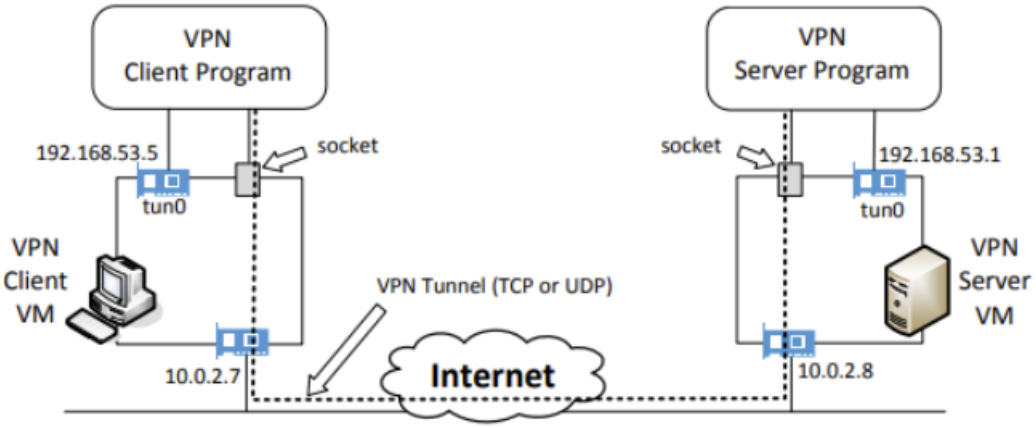
route del default

4.2 使用 TUN/TAP 创建一个主机到主机的隧道

TAP 模拟以太网设备，处理的是以太网帧等二层数据包；TUN 模拟网络层设备，处理的是 IP 等三层数据包。用 TAP/TUN 创建虚拟网络接口。

操作系统通过 TUN/TAP 网络接口将数据包传送到用户空间程序。另一方面，程序通过 TUN/TAP 网络接口发送的数据包被注入操作系统网络栈；在操作系统看来，数据包是通过虚拟网络接口的外部源进来的。当程序从 TUN/TAP 接口读取数据时，计算机发送到此接口的 IP 数据包将被传送给程序；另一方面，程序发送到接口的 IP 数据包将被传送到计算机中，就好像这些数据包通过这个虚拟网络接口从外部来的一样。程序可以使用标准的 read()和 write()系统调用来接收或发送数据包到虚拟接口。

vpnclient 和 vpnserver 程序是 VPN 隧道的两端，在提供的程序中使用 UDP 协议通过套接字相互通信。VPN 客户端和服务器程序通过 TUN 接口连接到主机系统做两件事：（1）从主机系统获取 IP 数据包，因此数据包可以通过隧道发送；（2）从隧道获取 IP 数据包，然后将其转发到主机系统，主机系统将数据包转发到其最终目的地



在 VM 运行提供的 VPN 服务器程序 vpnserver，首先 make

```
root@VM:/home/seed/myvpn# ls
Makefile  vpnclient.c  vpnserver.c
root@VM:/home/seed/myvpn# make
gcc -o vpnserver vpnserver.c
gcc -o vpnclient vpnclient.c
root@VM:/home/seed/myvpn#
```

之后启动

```
root@VM:/home/seed/myvpn# sudo ./vpnserver
root@VM:/home/seed/myvpn# Setup TUN interface success!
```

在另一个 vm 终端界面执行以下命令

sudo ifconfig tun0 192.168.53.1/24 up

sudo sysctl net.ipv4.ip_forward=1

sudo iptables -F

```
root@VM:/home/seed# sudo ifconfig tun0 192.168.53.1/24 up
root@VM:/home/seed# sudo sysctl net.ipv4.ip_forward=1
net.ipv4.ip_forward = 1
root@VM:/home/seed# sudo iptables -F
root@VM:/home/seed#
```

通过 ifconfig -a 查看虚拟 TUN 网络接口：

```
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
-00
          inet addr:192.168.53.1  P-t-P:192.168.53.1  Mask:255.255.255.0
          inet6 addr: fe80::e46b:7398:6b13:be16/64 Scope:Link
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

运行 VPN 客户端

在容器 HostU 上运行 VPN 客户端程序。首先在 VM 上将编译好的 VPN 客户端拷贝到容器 HostU 中。sudo docker cp vpnclient HostU:/vpnclient

```
root@VM:/home/seed/myvpn# sudo docker cp vpnclient HostU:/vpnclient
root@VM:/home/seed/myvpn#
```

在 HostU 中运行以下命令

./vpnclient 10.0.2.8

之后再次打开一个 HostU 终端执行

sudo ifconfig tun0 192.168.53.5/24 up

```
root@HostU:/# ./vpnclient 10.0.2.8
root@HostU:/# Setup TUN interface success!
Connect to server 10.0.2.8: Hello
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from the tunnel
Got a packet from TUN
Got a packet from TUN
Got a packet from TUN
```

```
root@HostU:/# sudo ifconfig tun0 192.168.53.5/24 up
root@HostU:/#
```

隧道建立起来。在使用隧道之前，需要在 HostU 和 VPN 服务器上设置路由，以指示通过隧道的预期流量。在 HostU 上，需要将所有进入专用网络（192.168.60.0/24）的数据包定向到 tun0 接口，从该接口可以通过 VPN 隧道转发数据包。使用 route 命令添加路由条目，使用此条命令

route add -net 192.168.60.0/24 tun0

```
root@HostU:/# route add -net 192.168.60.0/24 tun0
root@HostU:/#
```

在 HostV 上设置路由，使用以下命令：

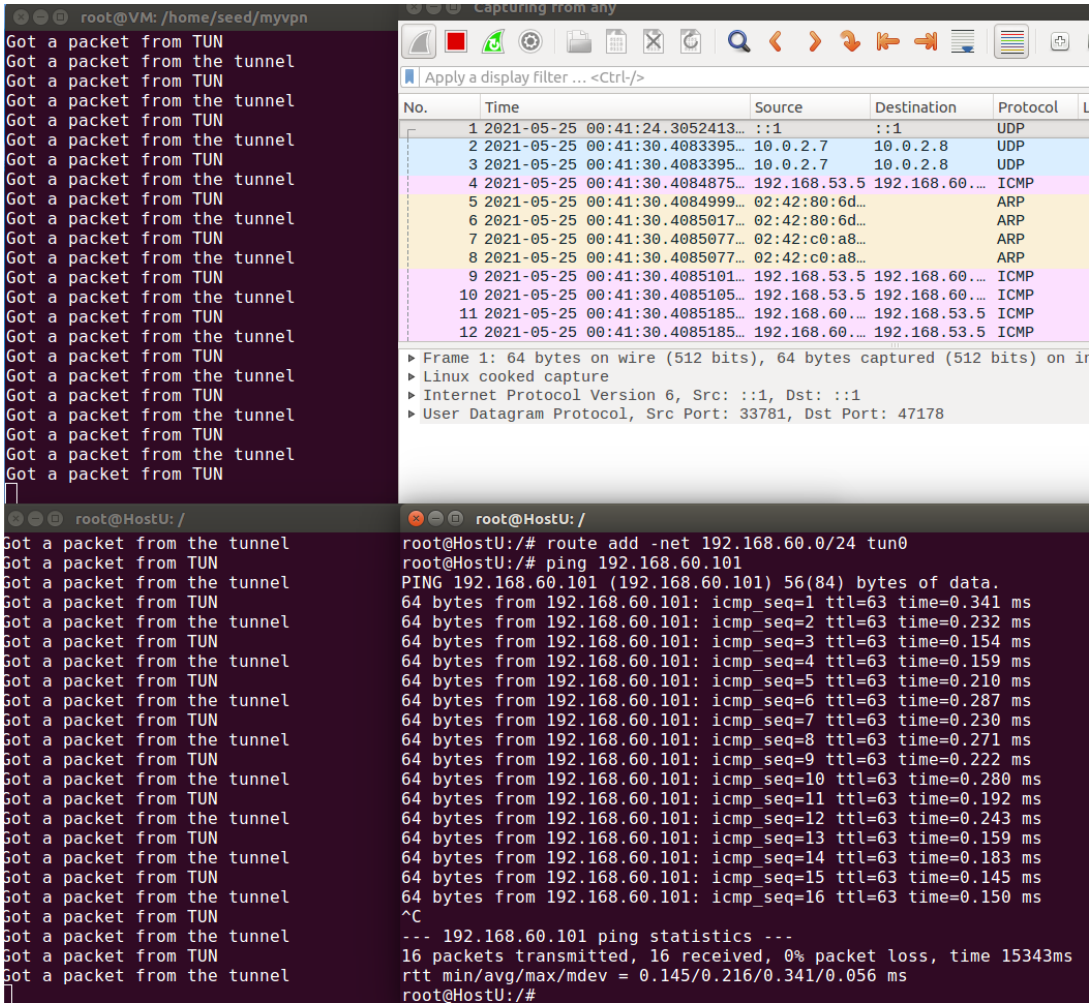
route add -net 192.168.53.0/24 gw 192.168.60.1

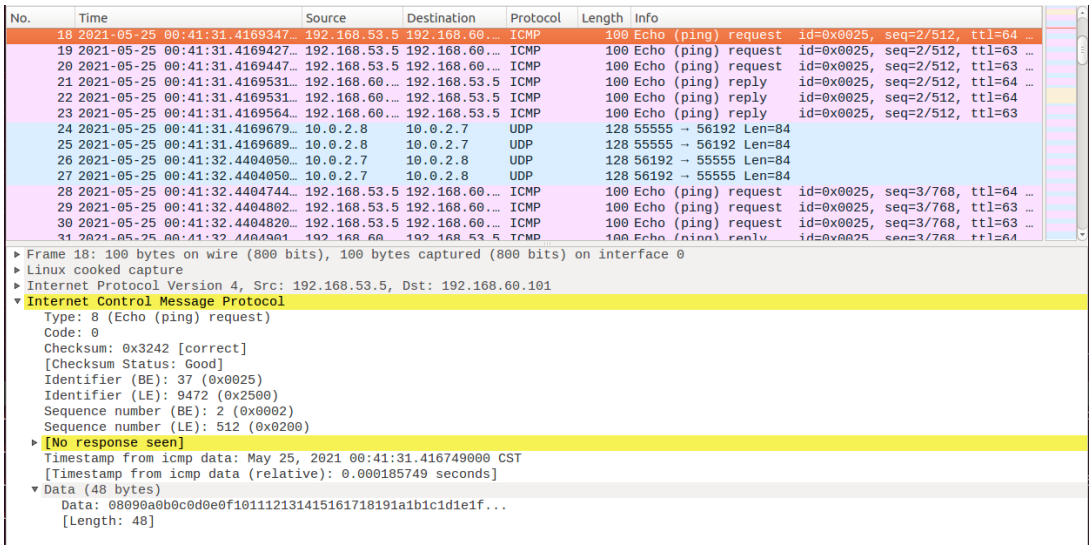
当 HostV 回复从 HostU 发送的数据包时，它将数据包路由到 VPN 服务器虚拟机，从那里，它可以被送入 VPN 隧道到另一端

```
root@HostV:/# route add -net 192.168.53.0/24 gw 192.168.60.1
root@HostV:/#
```

测试 VPN 隧道。从 HostU 访问 HostV。使用 ping 和 telnet 进行测试，期间 wireshark 捕获 HostU 上所有接口上流量，确定哪些数据包是隧道流量的一部分，哪些不是：

ping：粉色是隧道中 client 到 server 以及 server 返回的流量即原始报文，蓝色是 Host U 到网关以及网关到 HostU 即隧道封装报文。





telnet 测试：

先使用 service openbsd-inetd start 打开 telnet 服务，之后进行 telnet 和抓包

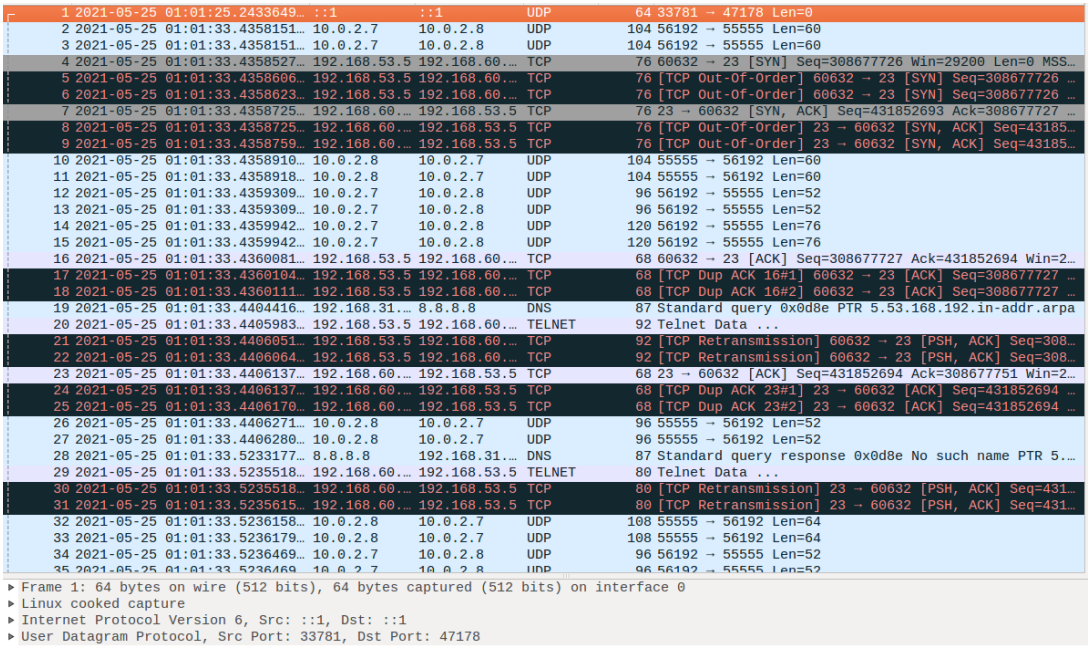
```
root@HostU:/# service openbsd-inetd start
 * Starting internet superserver inetd                              [ OK ]
```

```
root@HostV:/# service openbsd-inetd start
 * Starting internet superserver inetd                                [ OK ]
root@HostV:/# netstat -a|grep telnet
tcp        0      0 *:telnet                *:*                     LISTEN
root@HostV:/#
```

进行 telnet

```
root@HostU:/# telnet 192.168.60.101
Trying 192.168.60.101...
Connected to 192.168.60.101.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
HostV login: seed
Password:
Last login: Tue Jul 21 11:14:47 CST 2020 from 192.168.2.104 on pts/17
sh: 1: cannot create /run/motd.dynamic.new: Directory nonexistent
[05/25/21]seed@HostV:~$
telnet> quit
Connection closed.
root@HostU:/#
```

wireshark 中可以看到蓝色为 HostU 到网关以及网关到 HostU 流量，TCP 流量为 client 到 server 以及 server 到 client 的流量。



4.3 加密隧道

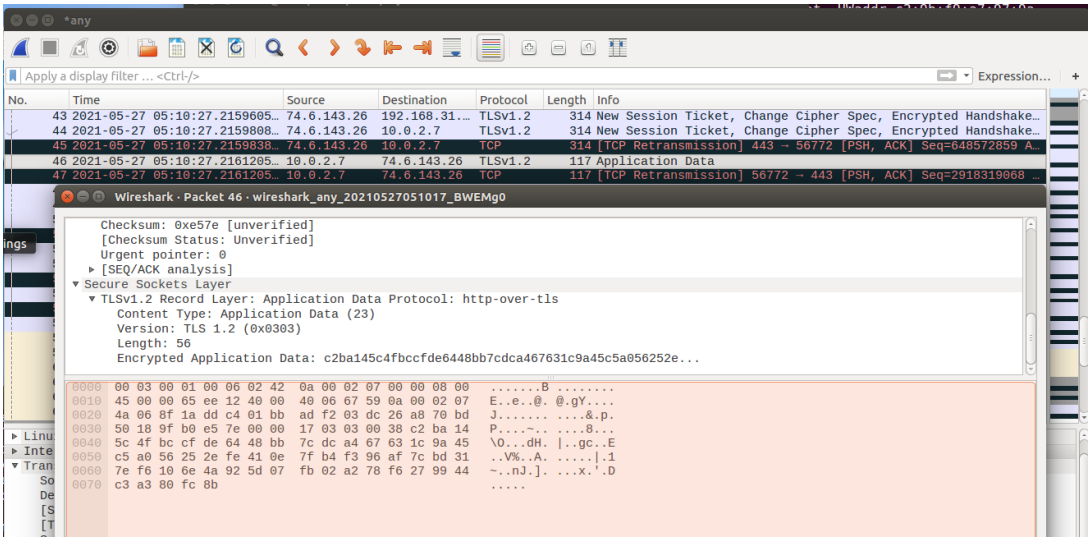
TLS 通常建立在 TCP 之上。首先需要使用 TCP 通道替换示例代码中的 UDP 通道，然后在隧道的两端之间建立 TLS 会话。TLS 客户端和服务器程序为 tlsclient 和 tlsserver。需要使用 Wireshark 捕获 VPN 隧道内的流量，并显示流量已加密。示例口令 123456。

使用示例 TLS 程序进行尝试：

```
root@HostU:/# ./tlsclient 10.0.2.8 4433
Enter PEM pass phrase:
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>bod
y {background-color: black}h1 {font-size:3cm; text-align: center; colo
r: white;text-shadow: 0 0 3mm yellow}</style></head><body><h1>Hello, w
orld!</h1></body></html>

root@HostU:/#
```

```
root@VM: /home/seed/mytls
[05/27/21]seed@VM:~/mytls$ su
Password:
root@VM:/home/seed/mytls# ./tlsserver
listen_sock = 3
sock = 4
SSL_accept return 1
SSL connection established!
Received: GET / HTTP/1.1
Host: 10.0.2.8
```

wireshark 抓包发现 application data 为加密的，无法看到原本的 get 请求。



4.4 身份认证

VPN 客户端必须对 VPN 服务器进行身份认证，确保服务器不是假冒的服务器。VPN 服务器必须认证客户端（即用户），确保用户具有访问专用网络的权限。

认证服务器的典型方法是使用公钥证书。VPN 服务器需要首先从CA 获取公钥证书。当客户端连接到 VPN 服务器时，服务器将使用证书来证明它是客户端预期的服务器。只需要正确配置 TLS 会话，openssl 就可以自动进行身份验证。

为 CA 生成自签名证书：**后面的 common name 均为 YA 即我的名字缩写用来表示个人所作。**

```
root@VM: /home/seed/zheng
root@VM:/home/seed/zheng# openssl req -new -x509 -keyout ya.key -out ya.crt -con
fig openssl.cnf
Generating a 2048 bit RSA private key
.....................+++
.............................................+++
writing new private key to 'ya.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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) [AU]:CN
State or Province Name (full name) [Some-State]:Hubei
Locality Name (eg, city) []:Wuhan
Organization Name (eg, company) [Internet Widgits Pty Ltd]:HUST
Organizational Unit Name (eg, section) []:YA
Common Name (e.g. server FQDN or YOUR name) []:YA
Email Address []:719001405@qq.com
root@VM:/home/seed/zheng#
```

创建一对公钥和私钥

```
root@VM:/home/seed/zheng# openssl genrsa -des3 -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.........++++++
.....................................................++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
root@VM:/home/seed/zheng#
```

生成证书签名请求 CSR

```
root@VM: /home/seed/zheng
Verifying - Enter pass phrase for server.key:
root@VM:/home/seed/zheng# openssl req -new -key server.key -out server.csr -conf
ig openssl.cnf
Enter pass phrase for server.key:
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) [AU]:CN
State or Province Name (full name) [Some-State]:Hubei
Locality Name (eg, city) []:Wuhan
Organization Name (eg, company) [Internet Widgits Pty Ltd]:HUST
Organizational Unit Name (eg, section) []:YA
Common Name (e.g. server FQDN or YOUR name) []:YA
Email Address []:719001405@qq.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:654321
An optional company name []:HUST
root@VM:/home/seed/zheng#
```

客户端生成 RSA 密钥对和 CSR

```
root@VM:/home/seed/zheng# openssl genrsa -des3 -out client.key 1024
Generating RSA private key, 1024 bit long modulus
..++++++
.............................++++++
e is 65537 (0x10001)
Enter pass phrase for client.key:
Verifying - Enter pass phrase for client.key:
root@VM:/home/seed/zheng#
```

```
root@VM: /home/seed/zheng
Verifying - Enter pass phrase for client.key:
root@VM:/home/seed/zheng# openssl req -new -key client.key -out client.csr -conf
ig openssl.cnf
Enter pass phrase for client.key:
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) [AU]:CN
State or Province Name (full name) [Some-State]:Hubei
Locality Name (eg, city) []:Wuhan
Organization Name (eg, company) [Internet Widgits Pty Ltd]:HUST
Organizational Unit Name (eg, section) []:YA
Common Name (e.g. server FQDN or YOUR name) []:YA
Email Address []:719001405@qq.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:654321
An optional company name []:HUST
root@VM:/home/seed/zheng#
```

CSR 文件使用可信 CA 进行生成证书

```
root@VM: /home/seed/zheng
            stateOrProvinceName       = Hubei
            organizationName          = HUST
            organizationalUnitName    = YA
            commonName                = YA
            emailAddress              = 719001405@qq.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                99:15:19:40:0C:95:D3:68:67:42:83:C2:D2:B8:9D:BE:CD:8F:65:F4
            X509v3 Authority Key Identifier:
                keyid:56:BE:66:93:3F:05:70:C4:9B:9D:9A:99:AB:BE:82:F2:A2:1D:9B:A
7

Certificate is to be certified until May 26 23:51:22 2022 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
root@VM:/home/seed/zheng#
```

```
root@VM: /home/seed/zheng
            stateOrProvinceName       = Hubei
            organizationName          = HUST
            organizationalUnitName    = YA
            commonName                = YA
            emailAddress              = 719001405@qq.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                AE:D5:E5:38:3F:57:78:F9:4F:9E:2B:E2:68:44:8B:2C:52:0B:B8:71
            X509v3 Authority Key Identifier:
                keyid:56:BE:66:93:3F:05:70:C4:9B:9D:9A:99:AB:BE:82:F2:A2:1D:9B:A
7

Certificate is to be certified until May 27 00:02:45 2022 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
root@VM:/home/seed/zheng#
```

使用自己生成的证书所在文件夹放到例程文件夹中，修改例程中的路径，将文件重新编译并放到 HostU 中，再次进行测试：

```
root@VM:/home/seed/mytls# sudo docker cp zheng HostU:/zheng
root@VM:/home/seed/mytls# sudo docker cp tlsclient HostU:/tlsclient
root@VM:/home/seed/mytls# sudo docker cp tlsserver HostU:/tlsserver
root@VM:/home/seed/mytls#
```

```
root@VM:/home/seed/mytls# ./tlsserver
Enter PEM pass phrase:
listen_sock = 3
sock = 4
SSL_accept return 1
SSL connection established!
Received: GET / HTTP/1.1
Host: 10.0.2.8


```

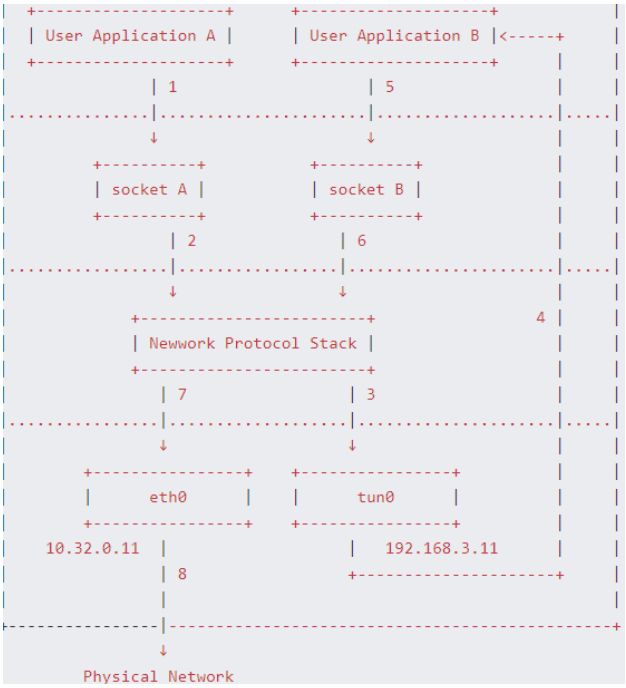
```
root@HostU:/# ./tlsclient 10.0.2.8 4433
Enter PEM pass phrase:
SSL connection is successful
SSL connection using AES256-GCM-SHA384
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html><html><head><title>Hello World</title></head><style>bod
y {background-color: black}h1 {font-size:3cm; text-align: center; colo
r: white;text-shadow: 0 0 3mm yellow}</style></head><body><h1>Hello, w
orld!</h1></body></html>

root@HostU:/#
```

4.5 编写自己的程序 YAVPN：

在之前 TLS 示例程序基础上，结合 VPN 示例程序，融合之后添加身份认证功能以及多客户端实现。

进行 SSL 握手前的环境准备，之后进行 SSL 握手，从而建立起 SSL 连接。先由客户端发送 client hello 再有 server 回应 server hello，之后进行验证 server 公钥证书，server 密钥交换，至此 server hello 完成，之后进行验证 client 公钥证书，client 密钥交换，之后可以进行 application data 的加密传输，在 wireshark 中查看 application data 此时已经是加密的密文。

server 端的 tun 设置 ip 为 192.168.53.1，以 sprintf 到 cmd 之后使用 system()函数调用的形式进行设置，对于每一个客户端，在运行初期会提示输入希望设置的 tun 的 ip，又由于 tun 的 ip 只有最后一位不同，为了方便使用时设置，这里只用输入最后一位，记录到一个数组，之后通过 cmd 和 system()的相同方式进行设置。



一个网络设备就像一个管道 pipe，从其中任意一端收到的数据将从另一端发送出去。例如 eth0 的两端分别是协议栈和物理网络，从物理网络收到的数据会转发给协议栈，而应用程序从协议栈发过来的数据则会通过物理网络发送出去。对于虚拟网络设备 tun 来说虚拟设备和物理设备几乎一样，能配置 ip 以及将网络发来的数据转发给协议栈，协议栈发来的数据交给网络发送出去。即 tun 设备一端是协议栈，另一端取决于虚拟网络设备的设置。这里将 tun 的另一端设为一个用户程序，协议栈发给 tun 的数据包能被应用程序读取并且应用程序能直接向 tun 写数据。

server 的主进程进行监听，子进程进行 ssl 连接，login 等实现。使用 createTUNDevice()中，ifreq 用来包含一个接口名字和具体内容，ioctl 作为设备控制接口函数，用于设备输入输出，控制设备 io 通道，TUNSETIFF 参数表示调用函数进行注册。ProcessRequest()进行将 ssl 中读到的数据写入 tun 设备发出去。若读到长度为 0 则断开 SSL。使用 ListenPipe()打开 pipe 文件（将每个 tun 的 pipe 文件保存在 server 端文件夹下，并且以 ip 为名字，如果发现某一个 ip 的文件已经存在则再申请此 ip 会提示冲突），将从 pipe 文件读到的写入 ssl。结构体 Tdata 记录了每个客户端中的线程中缩需要的信息如 pipe 文件一个 char*指针，以及一个 SSL*指针。其他函数可以参考样例进行修改。代码见附件。

测试：

打开 server

```
[05/31/21]seed@VM:~/.../mymini$ su
Password:
root@VM:/home/seed/mini2/mymini# ./yaserver
Enter PEM pass phrase:
net.ipv4.ip_forward = 1
```

第一个 client 连接上 server 之后在另一个相同 HostU 中使用 ping 命令：

```
root@HostU:/mymini# ./yaclient 10.0.2.8 4433
Input the username:seed
Input the password:dees
Input the tun ip(only the last like 5):5
Enter PEM pass phrase:
SSL connection is successful
SSL connection using AES256-GCM-SHA384
```

此中用户名和密码会有提示输入，之后由于每次为 tun 输入一个完整的地址进行设置比较麻烦，加上地址的前几位其实都是一样的，这里只需要输入最后一位即可。最后需要输入证书口令这里由于使用的自己生成的证书，口令为了方便均设置为 654321。

此时 server 显示：

```
root@VM:/home/seed/mini2/mymini# ./yaserver
Enter PEM pass phrase:
net.ipv4.ip_forward = 1
SSL connection established!
[Login username: seed]
[Password      : $6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/s
fYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/]
[Login successful!]
```

另一个 HostU 进行 ping

```
root@HostU:/# 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=0.375 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=0.319 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=0.256 ms
64 bytes from 192.168.60.101: icmp_seq=4 ttl=63 time=0.267 ms
^C
--- 192.168.60.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3071ms
rtt min/avg/max/mdev = 0.256/0.304/0.375/0.048 ms
root@HostU:/#
```

此时原 HostU 中为：

```
root@HostU:/mymini# ./yaclient 10.0.2.8 4433
Input the username:seed
Input the password:dees
Input the tun ip(only the last like 5):5
Enter PEM pass phrase:
SSL connection is successful
SSL connection using AES256-GCM-SHA384
Received(TUN) len = 84
Receive SSL: 84
Received(TUN) len = 84
Receive SSL: 84
Received(TUN) len = 84
Receive SSL: 84
Received(TUN) len = 84
Receive SSL: 84
```

ping 后 server 中为：

```
root@VM:/home/seed/mini2/mymini# ./yaserver
Enter PEM pass phrase:
net.ipv4.ip_forward = 1
SSL connection established!
[Login username: seed]
[Password      : $6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/s
fYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/]
[Login successful!]
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.5
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.5
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.5
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.5
```

可见成功 ping 通，在这里满足了认证、登录等功能。

再创建另一个 HostU2 来执行多客户端测试：HostU2 的 ip 设置为 10.0.2.11

```
root@HostU2:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:0a:00:02:0b
          inet addr:10.0.2.11  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::42:aff:fe00:20b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:22 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3278 (3.2 KB)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@HostU2:/#
```

删除默认路由 route del default
在这里运行客户端，成功建立连接观察 HostU、HostU2、server 终端的表现以及 wireshark 抓包查看是否加密：
此时使用 192.168.53.7，在 HostU 中使用的是 192.168.53.5
HostU2 中执行 client 后：

```
root@HostU2:/mymini# ./yaclient 10.0.2.8 4433
Input the username:seed
Input the password:dees
Input the tun ip(only the last like 5):7
Enter PEM pass phrase:
SSL connection is successful
SSL connection using AES256-GCM-SHA384
```

此时的 server 显示：

```
Received(len = 84 ) ---- ip = 192.168.53.5
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.5
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.5
SSL connection established!
[Login username: seed]
[Password      : $6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/s
fYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/]
[Login successful!]
```

此时再开一个 HostU2 进行 ping
另一个 HostU2 的 client 显示：

```
root@HostU2:/# 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=0.314 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=0.211 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=0.205 ms
64 bytes from 192.168.60.101: icmp_seq=4 ttl=63 time=0.212 ms
^C
--- 192.168.60.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3055ms
rtt min/avg/max/mdev = 0.205/0.235/0.314/0.047 ms
root@HostU2:/#
```

此时 server 显示：

```
Received(len = 84 ) ---- ip = 192.168.53.5
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.5
SSL connection established!
[Login username: seed]
[Password      : $6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/s
fYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/]
[Login successful!]
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.7
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.7
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.7
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.7
```

原 HostU2 窗口显示：

```
root@HostU2:/mymini# ./yaclient 10.0.2.8 4433
Input the username:seed
Input the password:dees
Input the tun ip(only the last like 5):7
Enter PEM pass phrase:
SSL connection is successful
SSL connection using AES256-GCM-SHA384
Received(TUN) len = 84
Receive SSL: 84
Received(TUN) len = 84
Receive SSL: 84
Received(TUN) len = 84
Receive SSL: 84
Received(TUN) len = 84
Receive SSL: 84
```

两个客户端同时 ping 成功，没有发生一个会挤掉另一个的情况，wireshark 抓包稳定，在检查时老师已经检查了此点。

断开 HostU2 的连接，再次测试 HostU 能否 ping 通，以表现 HostU2 中的客户端不会对 HostU 的客户端产生影响：
断开 HostU2 的连接，在 HostU2 测试 ping 无法 ping 通：
此时的 HostU2

```
Receive SSL: 84
Received(TUN) len = 84
Receive SSL: 84
Received(TUN) len = 84
Receive SSL: 84
^C
root@HostU2:/mymini#
```

此时 server

```
Received(len = 84 ) ---- ip = 192.168.53.7
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.7
Received SSL: 0
SSL shutdown.
[Close sock]
[Return 0]
```

此时另一个 HostU2 ping 测试：

```
root@HostU2:/# ping 192.168.60.101
connect: Network is unreachable
root@HostU2:/#
```

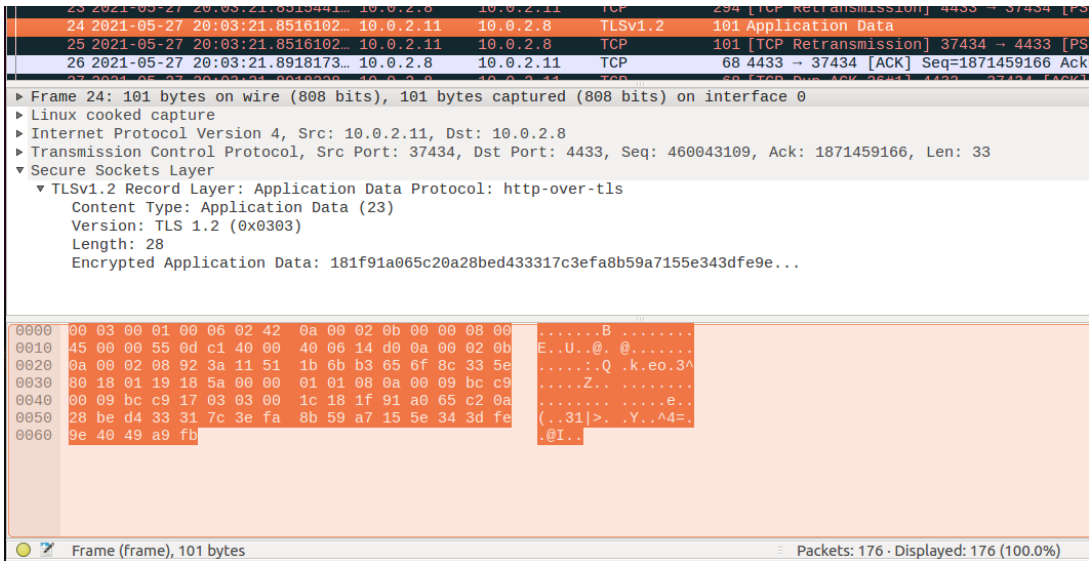
此时 HostU 中保持连接，ping 测试：

```
root@HostU:/# 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=0.473 ms
64 bytes from 192.168.60.101: icmp_seq=2 ttl=63 time=0.209 ms
64 bytes from 192.168.60.101: icmp_seq=3 ttl=63 time=0.217 ms
^C
--- 192.168.60.101 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2054ms
rtt min/avg/max/mdev = 0.209/0.299/0.473/0.124 ms
root@HostU:/#
```

此时 server 看到：

```
Received(len = 84 ) ---- ip = 192.168.53.7
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.7
Received SSL: 0
SSL shutdown.
[Close sock]
[Return 0]
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.5
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.5
Received SSL: 84
Received(len = 84 ) ---- ip = 192.168.53.5
```

仍然可以 ping 通可见一个客户端不会影响另一个客户端，实现了多客户端。
并且 wireshark 抓包观察可以看到数据是加密的：



说明完成了加密工作。

在启动 server 时需要进行证书口令验证，在启动 client 时需要证书口令验证并且需要输入在 server 端存在的用户 seed 以及密钥 dees，登陆时 client 和 server 会显示 login name 以及 passwd 并且在成功登录后会显示 login successful！ ：

```
Enter PEM pass phrase:
SSL connection is successful
SSL connection using AES256-GCM-SHA384
Received(TUN) len = 84
Receive SSL: 84
Received(TUN) len = 84
```

```
root@HostU:/mymini# ./yaclient 10.0.2.8 4433
Input the username:seed
Input the password:dees
```

```
[Login username: seed]
[Password      : $6$wDRrWCQz$IsBXp9.9wz9SGrF.nbihpoN5w.zQx02sht4cTY8qI7YKh00wN/s
fYvDeCAcEo2QYzCfpZoaEVJ8sbCT7hkxXY/]
[Login successful!]
```

可见身份认证成功完成。

综上，各方面任务均已完成，并且使用的是自己生成的证书，完成实验。

5 实验思考

tunnel 断开测试：

在 HostU 上，telnet 到 HostV。在保持 telnet 连接存活的同时，断开 VPN 隧道。然后我们在 telnet 窗口中输入内容，并报告观察到的内容。然后我们重新连接 VPN 隧道。需要注意的是，重新连接 VPN 隧道，需要将 vpnserver 和 vpnclient 都退出后再重复操作，请思考原因是什么。正确重连后，telnet 连接会发生什么？ 会被断开还是继续？请描述并解释你的观察结果。

在保持 telnet 存活的同时断开 VPN 会发现，在 telnet 窗口进行输入，会发现 telnet 窗口无法输入，卡死在断开时的样子。在重新连接 VPN 之后，telnet 窗口中的之前的输入会立即回显，并且可以重新正常工作。说明断开后信息无法正常递交，暂时放在设备中或者不断重试，而在重新连接后信息仍然可以继续按照原来的路径递交。

心得体会与建议

1 心得体会

在平常的生活中因为自己经常使用 VPN，而且之前网络安全也学到了 VPN，自以为对 VPN 有着一定的了解，可是这次实验真的让我的一些认知发生了很大的变化，学习到了很多之前没有仔细了解过的知识。

第一个印象很深的地方就是证书，之前不管是密码学还是其他课上，对于证书始终停留在概念上的认识，这次自己按照指导书以及搜索的资料学习并且进行实践，对证书的生成、验证这些方面收获了很多新的理解。

第二个学到很多的是 TUN 设备的使用，目前认识到 TUN 能够：在一个应用将信息通过 socket 发给协议栈后，协议栈另一端连着 eth 或者 tun，eth 另一边为物理网络，tun 另一边则是连着一个用户层的应用，协议栈根据包的目的地址以及路由判断要交给 tun，则 tun 将数据包发给另一端的其他进程，进程收到数据包经过处理又通过 socket 发出去，socket 交给协议栈，协议栈判断交给 eth，之后 eth 就把包从物理网络发出去，这是我理解的网络层设备 eth 和 tun 的工作流程。tun 能将协议栈中的数据包转发给用户空间的应用程序，使这些程序能够处理数据包。

这次实验过程遇到了很多困难，但都能够通过咨询同学和老师一一解答，而且在老师发的 FAQ 文档中也找到了两三个疑问的解答，感到收获良多。

2 建议

建议老师可以在 VPN 实验的演示中添加一项，即将老师希望我们最终要完成的程序展示一遍最后要达成什么样子，当然这个示例的程序不要提供给同学只是进行展示，这个程序可以是老师自己编写的也可以是往届的优秀的程序，以此告诉同学们最后要能够实现完成哪些功能的展示。这样要求更加清晰明了，而不像 PPT 或者指导书上那样仅仅写了一行：要求最终结合在一起。这样一句话会带来很多歧义，结合在一起的理解我一开始就理解成了，运行一个程序程序中可以选择多个功能比如 1 是纯建立隧道，2 是加密的隧道，3 是实现了身份认证的版本等等，这样子的话会让人在理解题意的过程中浪费掉一些不必要的时间精力。以上即是我的建议，谢谢老师。