## TIP:

In the VPN lab, I built 4 VMs, whose names are VPN\_Client1, VPN\_Client2, VPN\_Server, VPN\_InternalHost. And 2 networks: NAT network and Internal network. To make VMs easier to control, the 2 clients and the server are using host-only network as well, only for build SSH connections between them and the host windows machine.

As well, to verify VMs through SSH windows, I create files in Desktop/Downloads.

Details are as follows:

VPN\_Client1:

Description: The first client to use VPN.

IP address in NAT network:10.0.2.14

IP address in Host-only network: 192.168.56.102

Ls Dowloads: vpn ImClient1

VPN\_Client2:

Description: The second client to use VPN.

IP address in NAT network:10.0.2.16

IP address in Host-only network: 192.168.56.104

Ls Dowloads: vpn ImClient2

VPN\_Server:

Description: The VPN server.

IP address in NAT network:10.0.2.13

IP address in Internal network: 192.168.60.1

IP address in Host-only network: 192.168.56.101

Ls Dowloads: vpn ImServer

VPN\_InternalHost:

Description: The first client to use VPN.

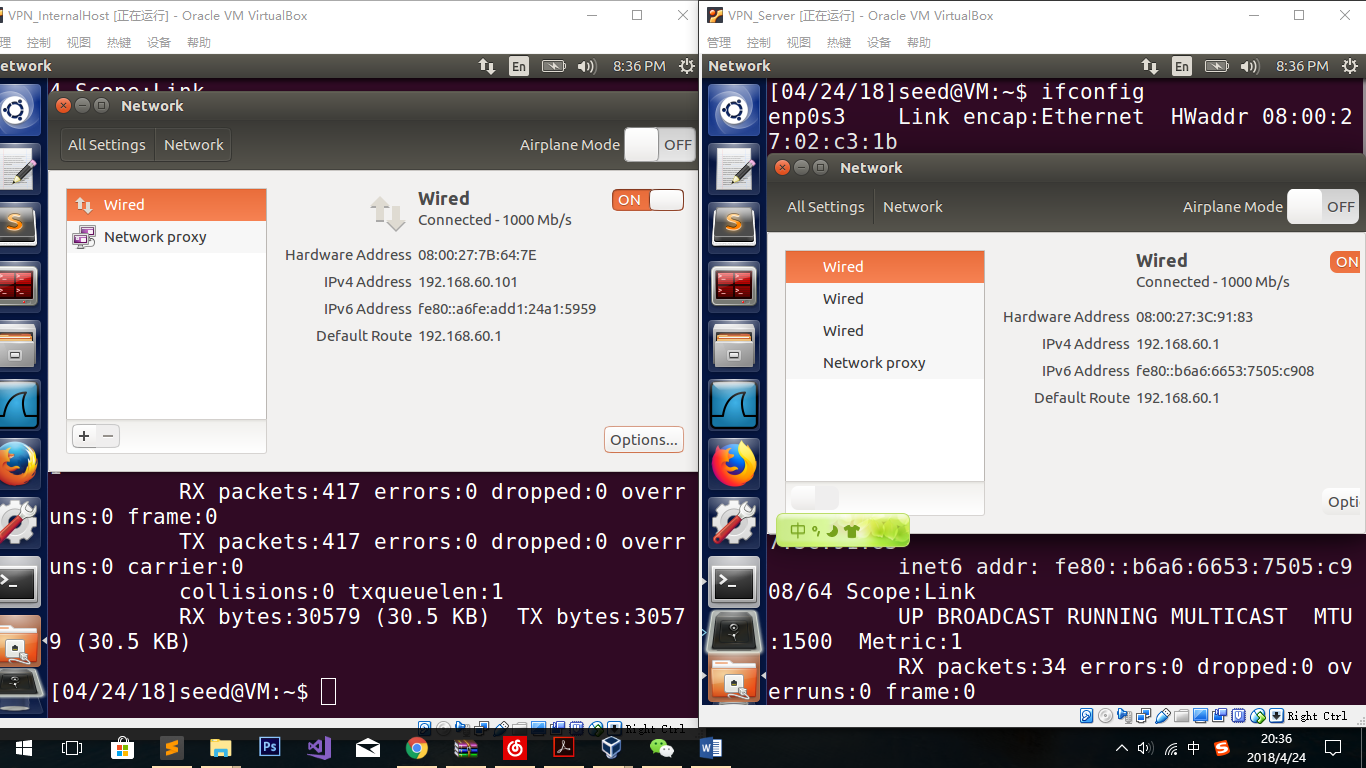
IP address in Internal network: 192.168.60.101

Ls Dowloads: imInternalHost

## Task 1: VM Setup

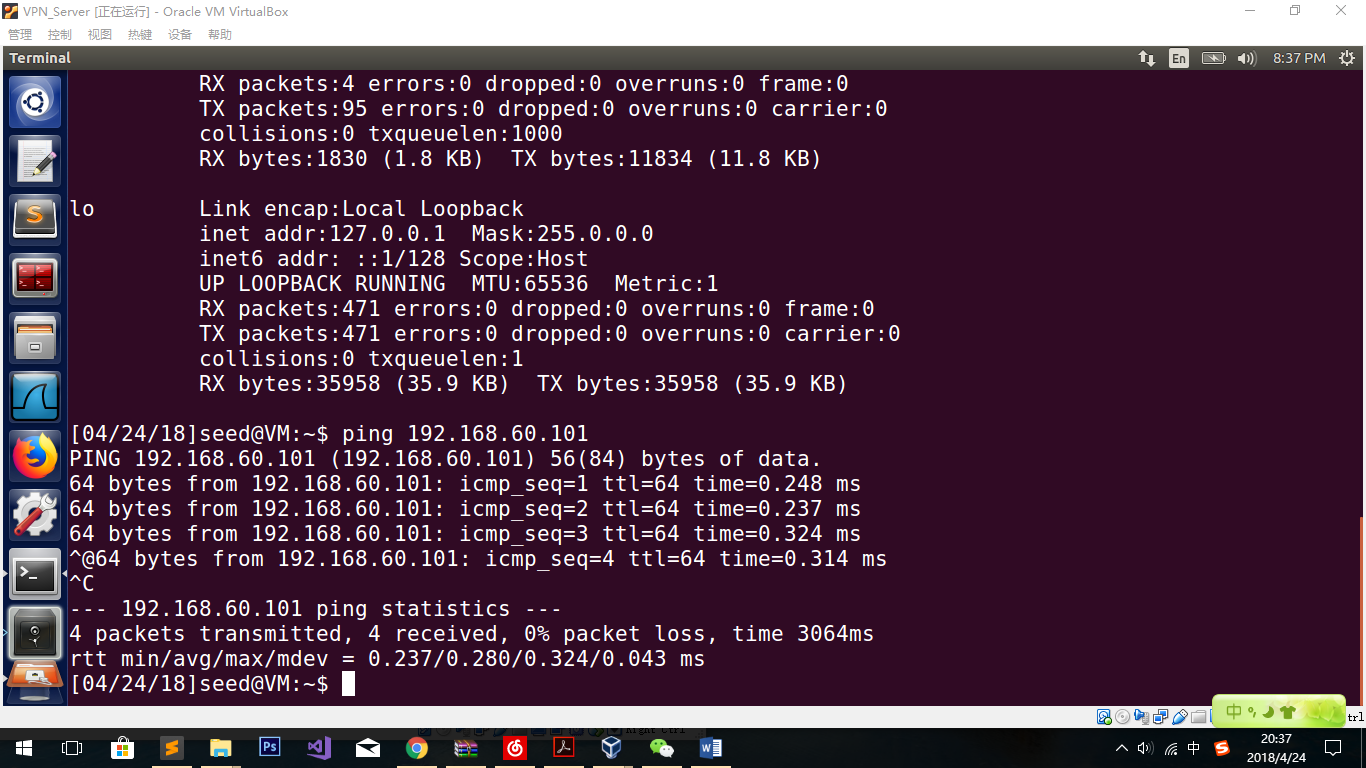
Create Nat network and Internal network in Virtual Box and set things up as the description says.

Set the ip addresses in the internal network:



Test it:

Ping 192.168.56.101 in VPN\_Server.



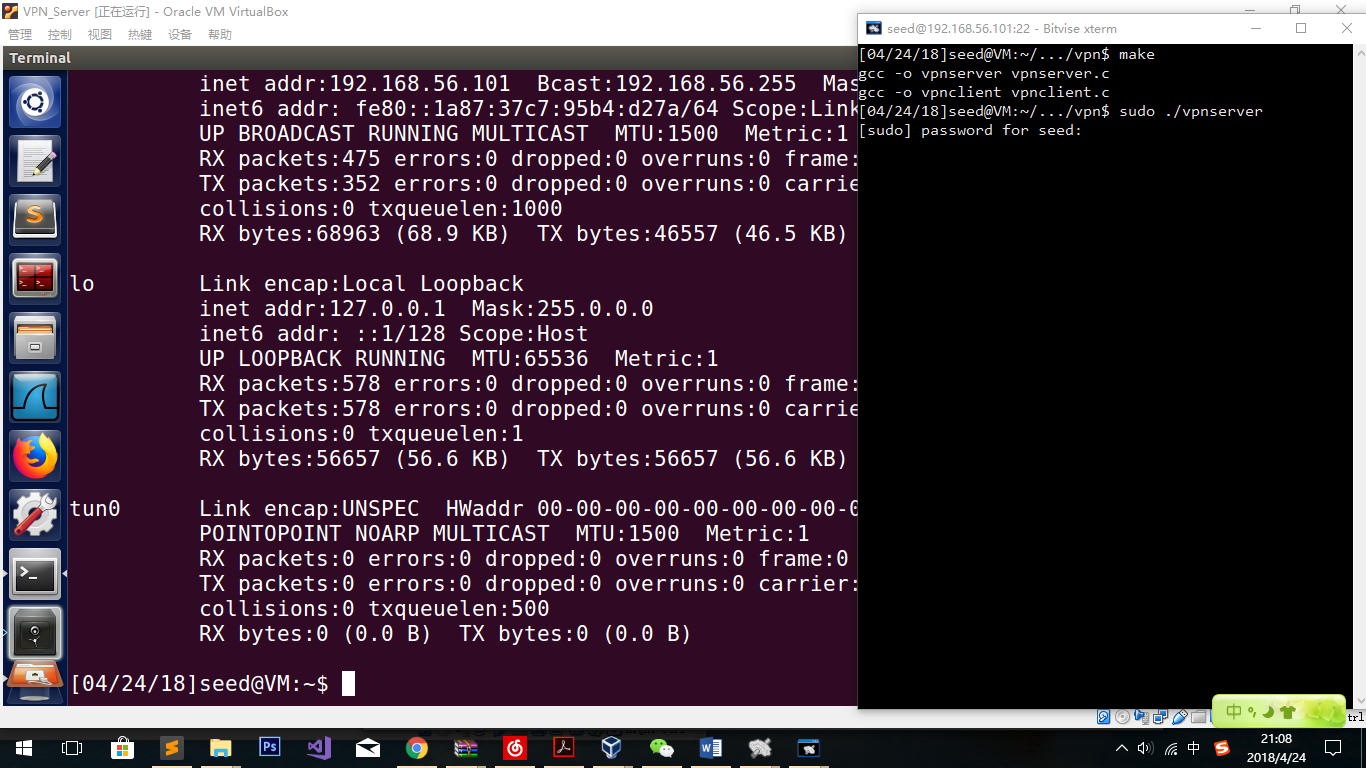
It shows that the configure is successfully done.

## Task 2: Creating a VPN Tunnel using TUN/TAP

### Step 1: Run VPN Server

*sudo ./vpnserver*

*ifconfig -a*



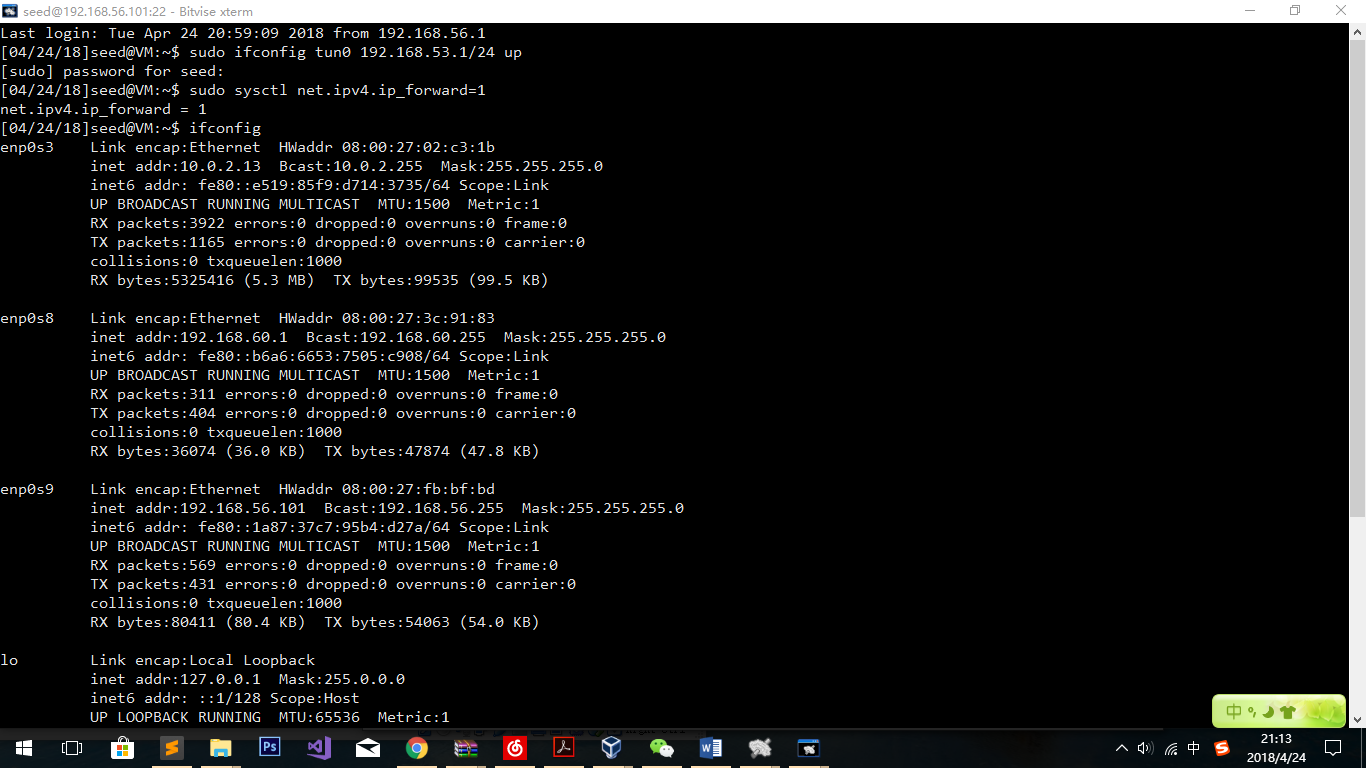
**Observation:**

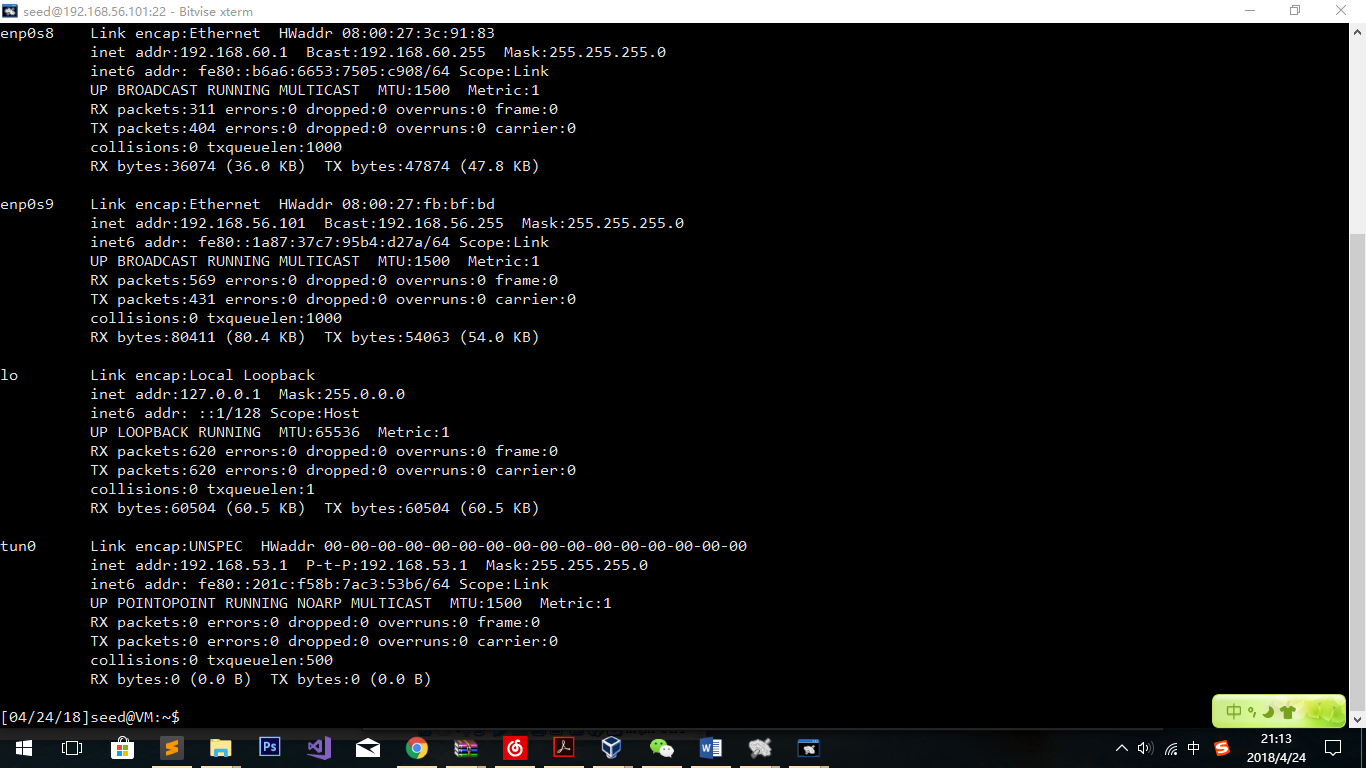
In the server’s machine, we can clearly see that the new net interface “tun0” appears, which means that the program is running correctly.

In another window:

sudo ifconfig tun0 192.168.53.1/24 up

sudo sysctl net.ipv4.ip\_forward=1



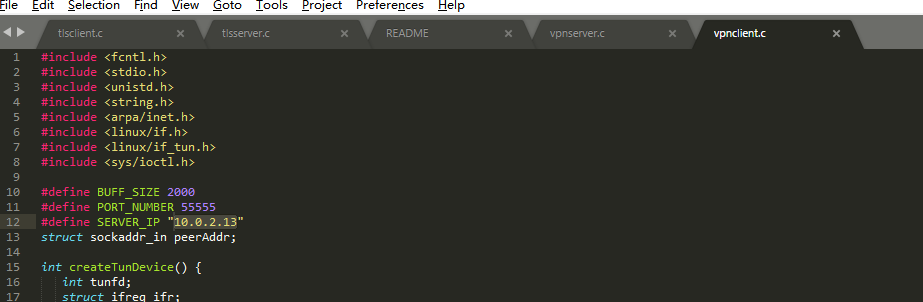


Observation:

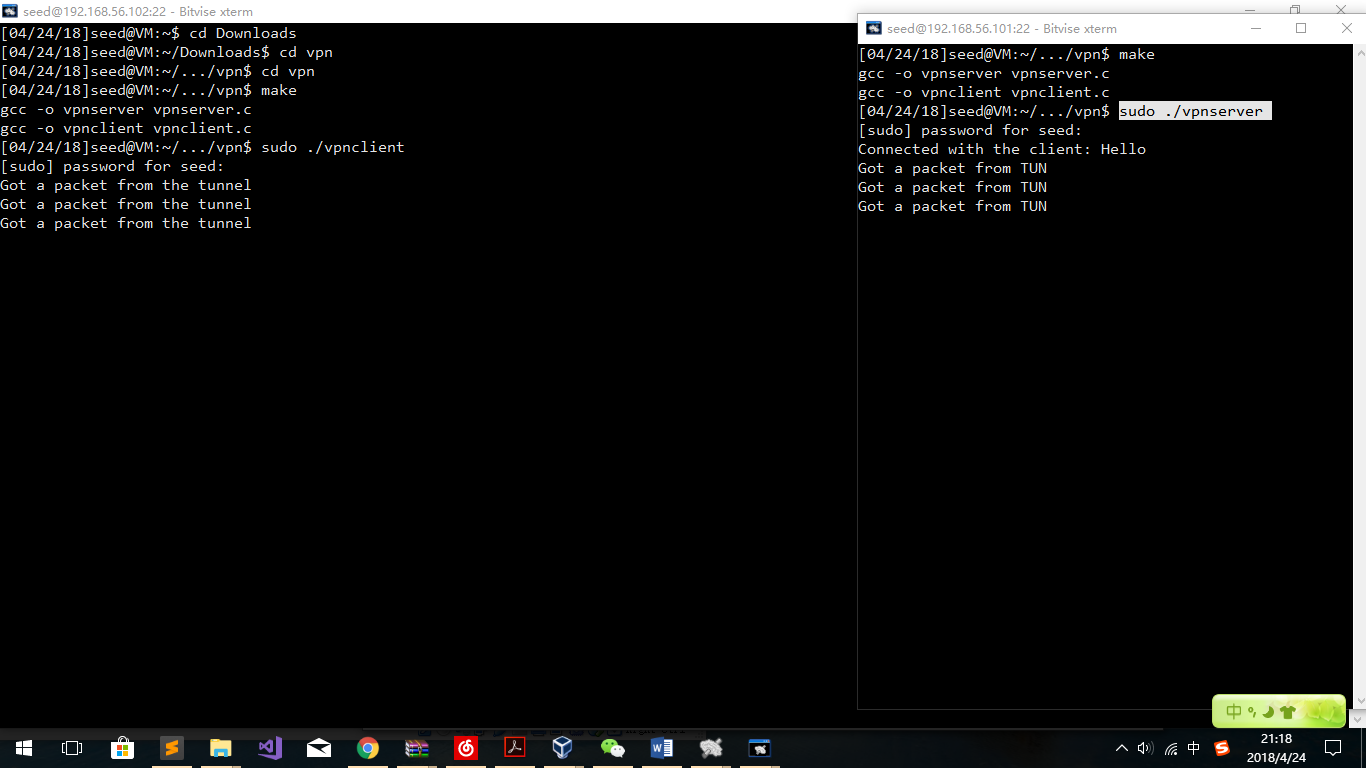
In the under the ifconfig command, we can see that the new net interface “tun0” has already had its new ip address.

### Step 2: Run VPN Client

Modify the file “vpnclient.c”, change the server ip to 10.0.2.13



sudo ./vpnclient

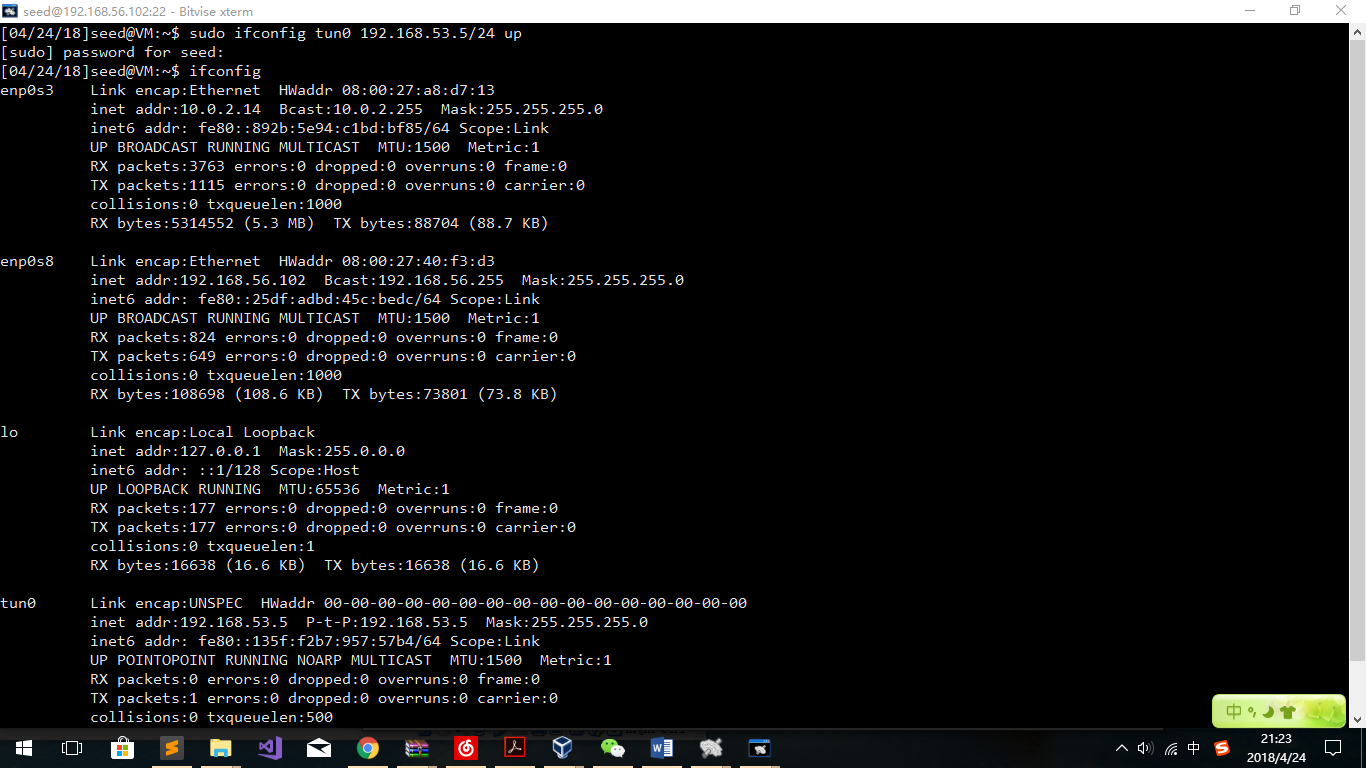


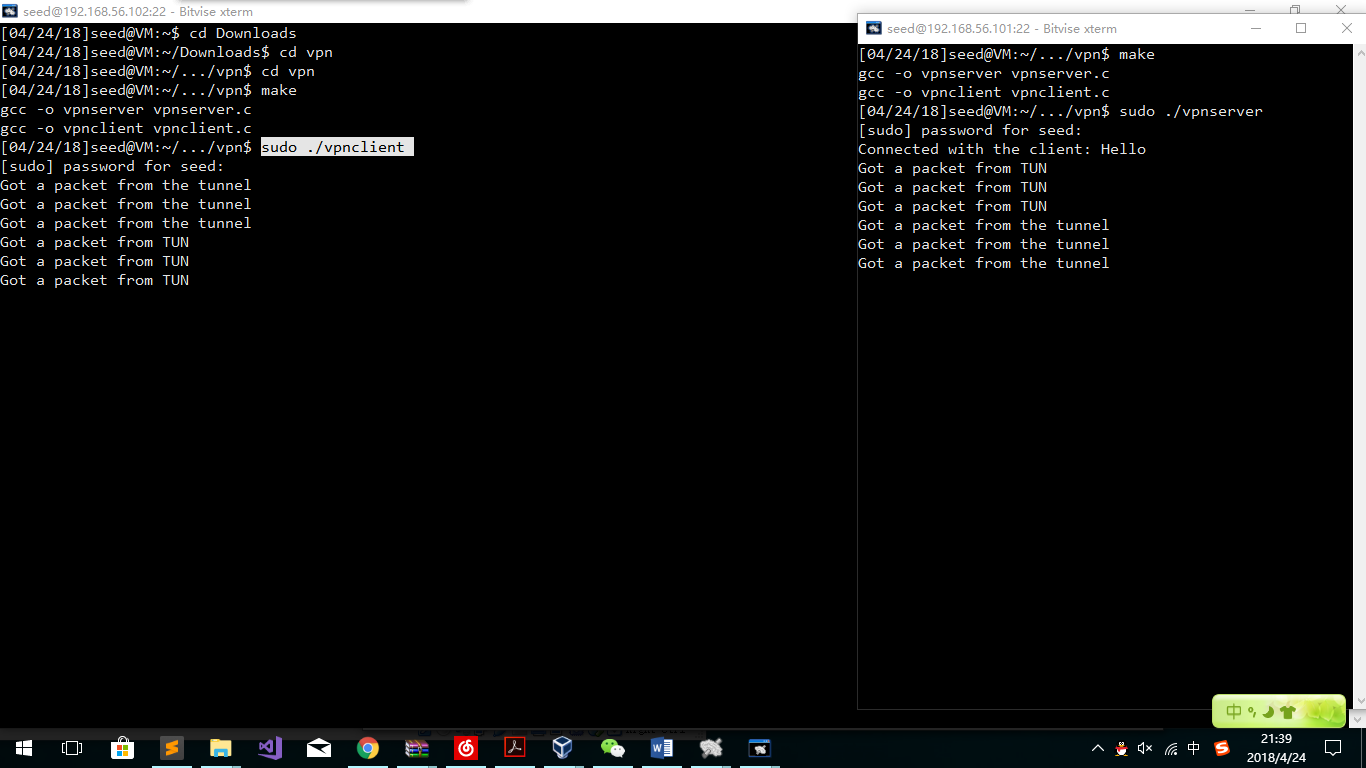
**Observation:**

After typing in the command, both the client window and the server window show that 3 get packets are got from each other, which means the tunnel is built successfully.

As well, we need to write an ip address for the new interface in the client’s “tun0”.

sudo ifconfig tun0 192.168.53.5/24 up





Observation:

Under the ifconfig, we can see the new interface with its own ip address. That means the interface is successfully set.

### Step 3: Set Up Routing on Client and Server VMs:

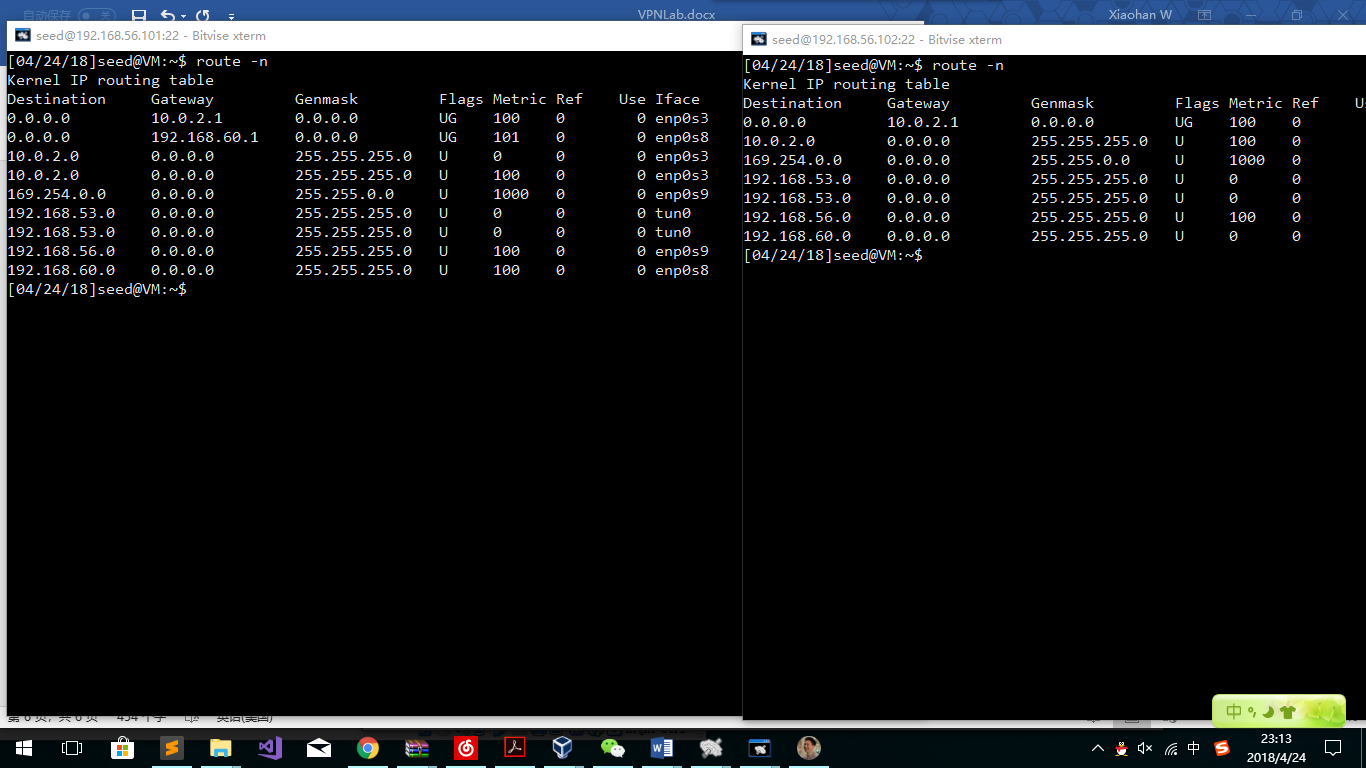
In the client:

sudo route add -net 192.168.53.0 netmask 255.255.255.0 dev tun0

sudo route add -net 192.168.60.0/24 tun0

In the server:

sudo route add -net 192.168.53.0 netmask 255.255.255.0 dev tun0



Observation:

Using the route -n command, we can clearly see that the routes are added into their route tables successfully.

### Step 4: Set Up Routing on Host V

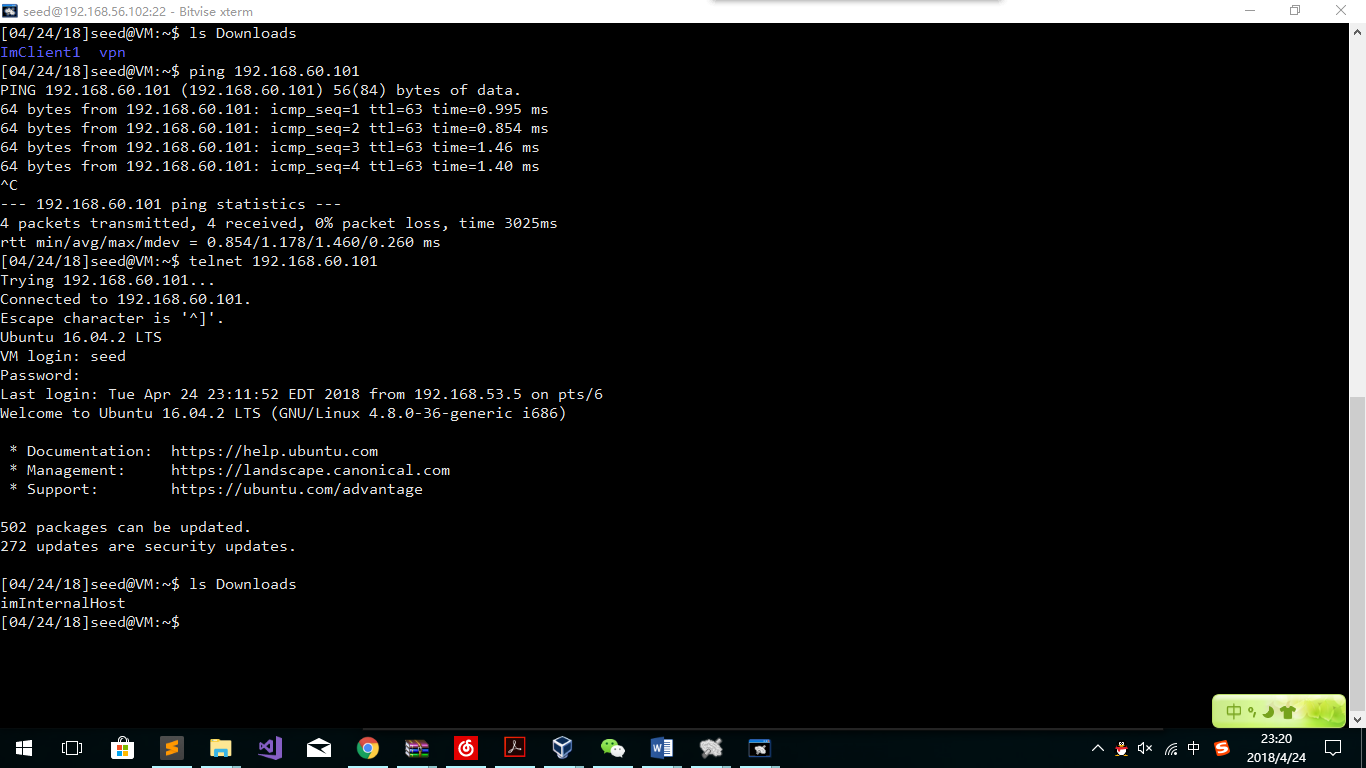
It is necessary that we want to send packets from Host V to Host U. Those packets sent from U to V has the source ip of 192.168.53.5. As a result, we need to route those packets to the server.

sudo route add -net 192.168.53.0/24 gw 192.168.60.1

### Step 5: Test the VPN Tunnel:

To verify which one is the one we are telnetting to, we will use the files in Downloads listed in Tip Section.

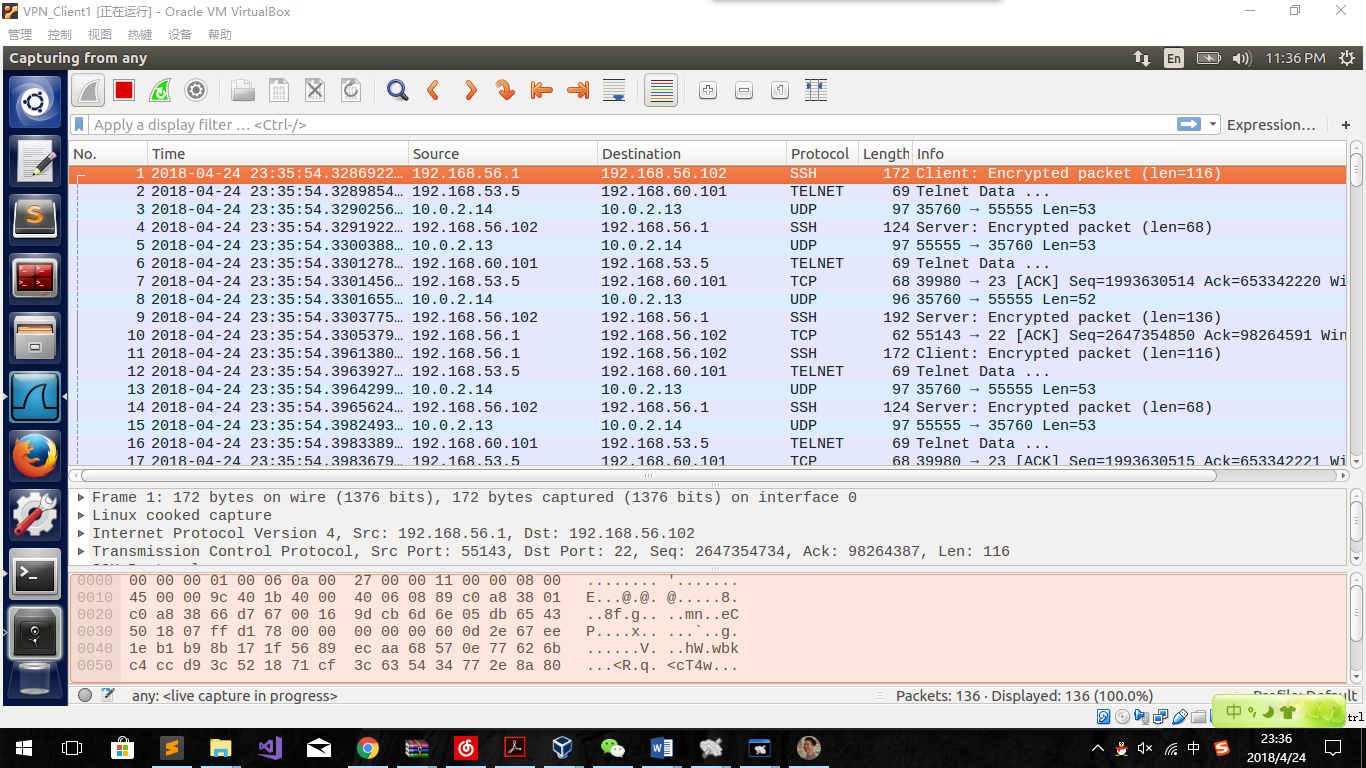
Ping and Telnet:



Observation:

We can clearly see that after the configuration, we can ping the InternalHost machine successfully. As well, the telnet works successfully.

WireShark: (Using the first 17 packets as an example)

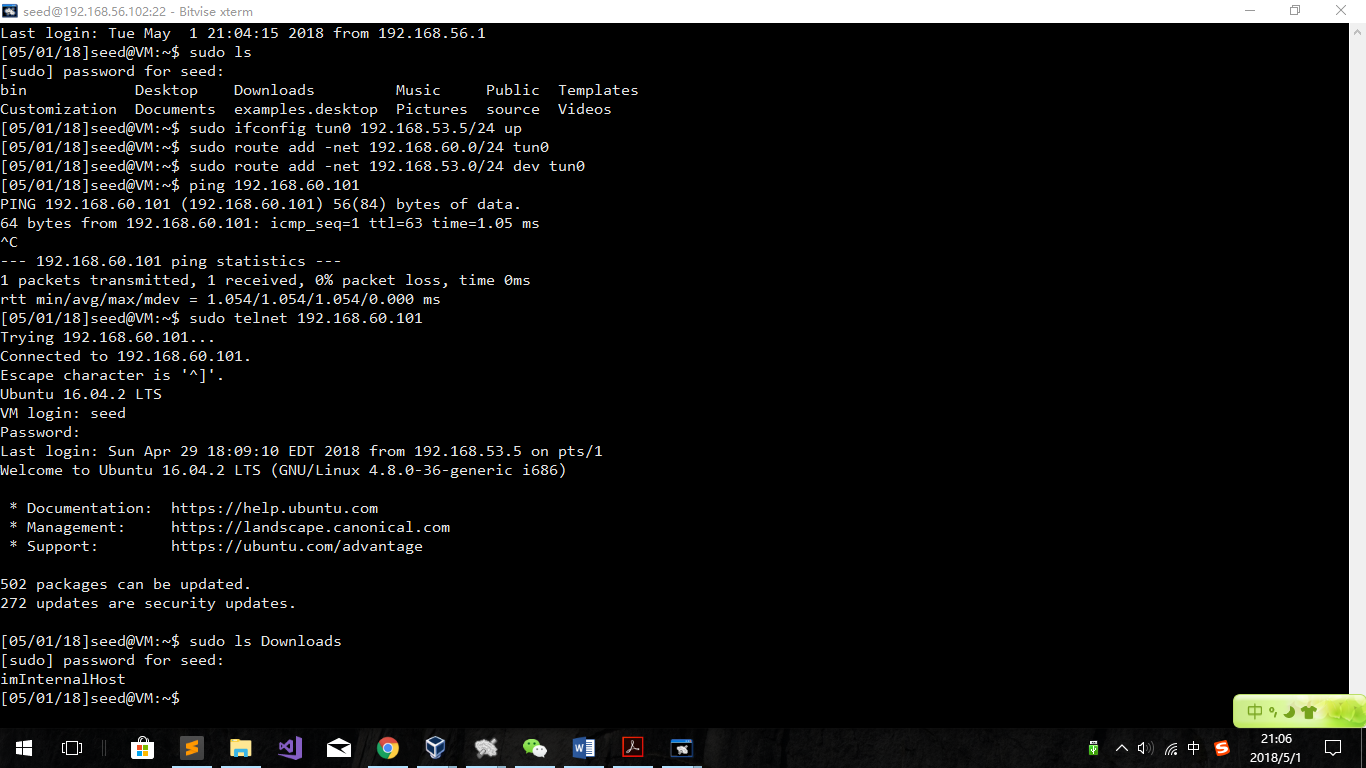


Tunnel traffic should have destination or source as 192.168.56.1 or 192.168.56.5, because those virtual address are only used in “tun0”. As well, the source and destination should be the client and the server. So packets in tunnel traffic are No: 1,2,4,6,7,9,10,11,12,14,16,17. Including packets keeping the telnet connection between the client and the server, and some packets sending from Host V as No.6,16, etc.

Others are not using “tun0”. For No.3,5,8,13,15 are UDP packets between the client and the server through NAT network. Those UDP packets have the data part as the packets being used when Host U and Host V are communicating with each other.

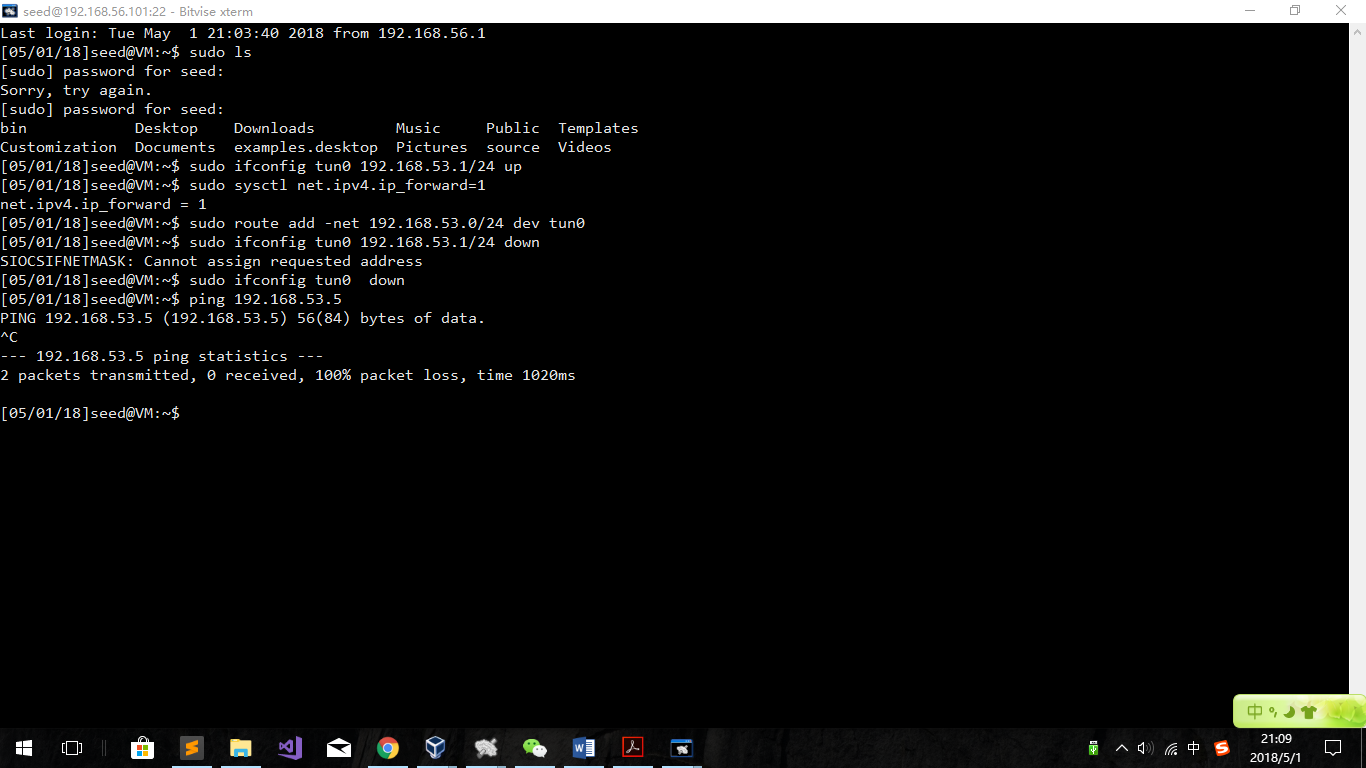
### Step 6: Tunnel-Breaking Test.

Firstly, telnet to the internal host.



Try the “ls Downloads” command to show that I am now telnetting the internal host.

To break the tunnel, we can easily down the tun device of the server. As a result, the tunnel will be broken.

sudo ifconfig tun0 down

Observation:

Since we cannot ping 192.168.53.5 again, the tunnel is definitely broken.

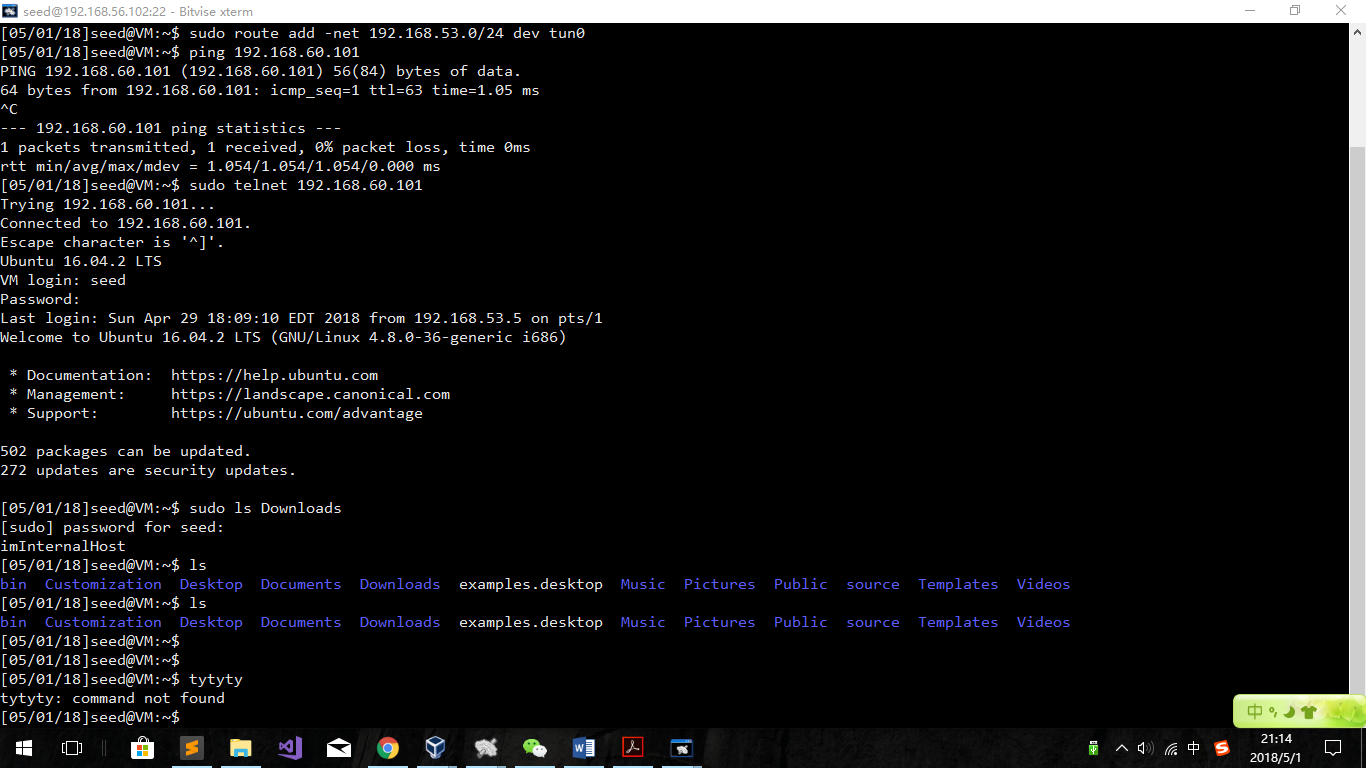
Come back to the client machine, finding that the window seems to be locked. As a result, we type in ls. And rebuild the tunnel.

sudo ifconfig tun0 192.168.53.1/24 up

sudo sysctl net.ipv4.ip\_forward=1

sudo route add -net 192.168.53.0/24 dev tun0

After a while, the window become unlocked:



Explanation:

After the tunnel is broken, since the user’s client is not broken, the client will keep sending request of telnet using the tun0 interface. However, the tun0 interface can not send any packets using the tunnel again. So it turns out that tunnel’s client side is resending the data again and again. After the tunnel being rebuilt and the tun0 interface can send packets to 192.168.53.1 again. That is why after the rebuild the result shows in the window.

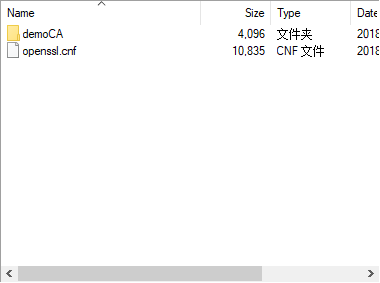
## Task 3: Encrypting the Tunnel

### Becoming a Certificate Authority

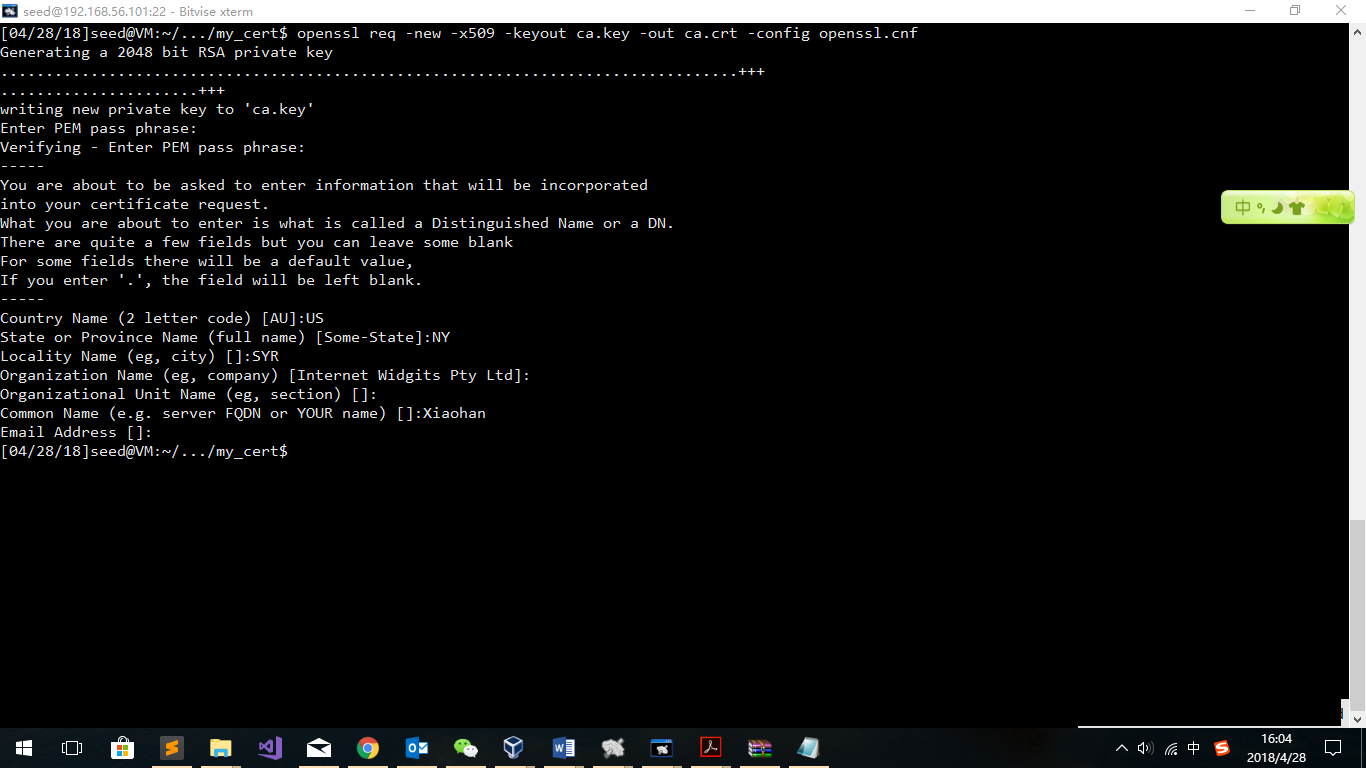
Since we need to create our own certification file, we need to use x509 commands to build our own pem files.

sudo cp /usr/lib/ssl/openssl.cnf /home/seed/Downloads/tls/my\_cert

Create folders as follows:

openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf

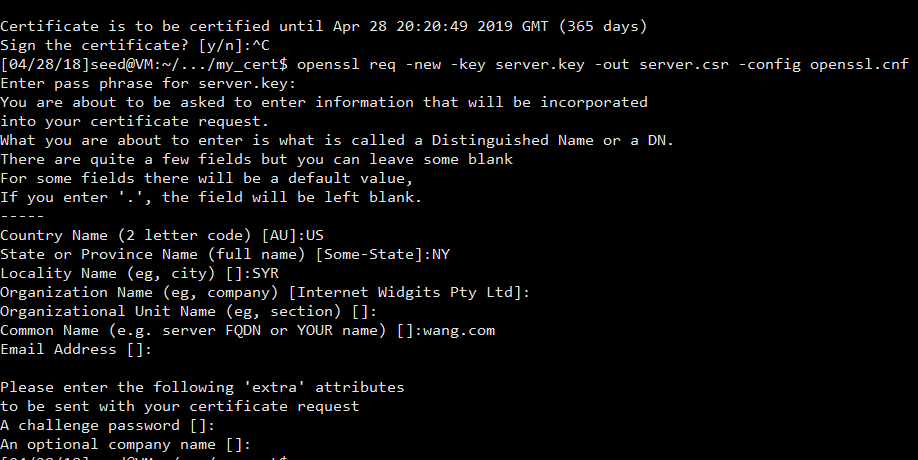


### Creating a Certificate for wang.com

openssl genrsa -aes128 -out server.key 1024

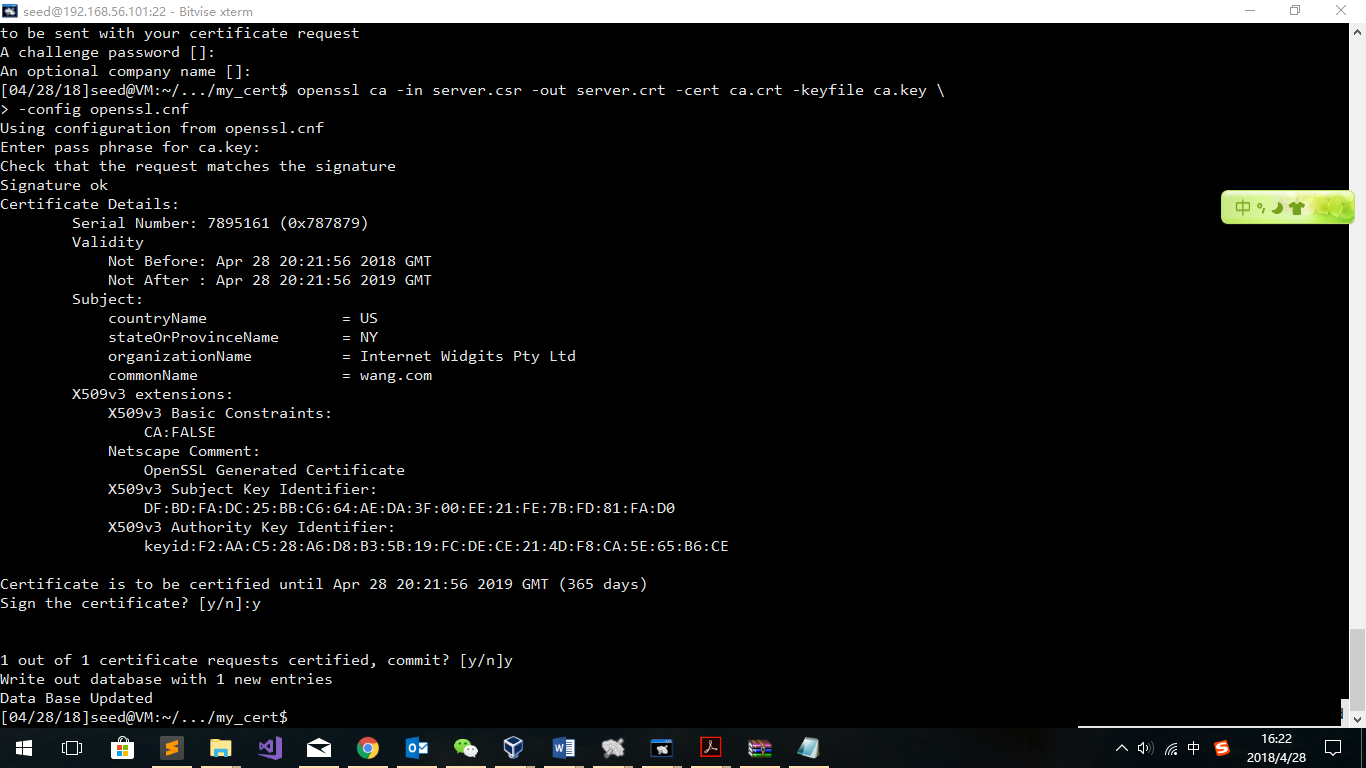
openssl rsa -in server.key -text

openssl req -new -key server.key -out server.csr -config openssl.cnf



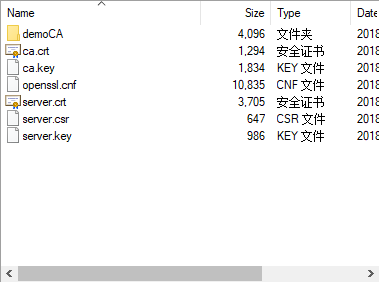
openssl ca -in server.csr -out server.crt -cert ca.crt -keyfile ca.key \

-config openssl.cnf



Sign complete.

File dictionary is now like this:



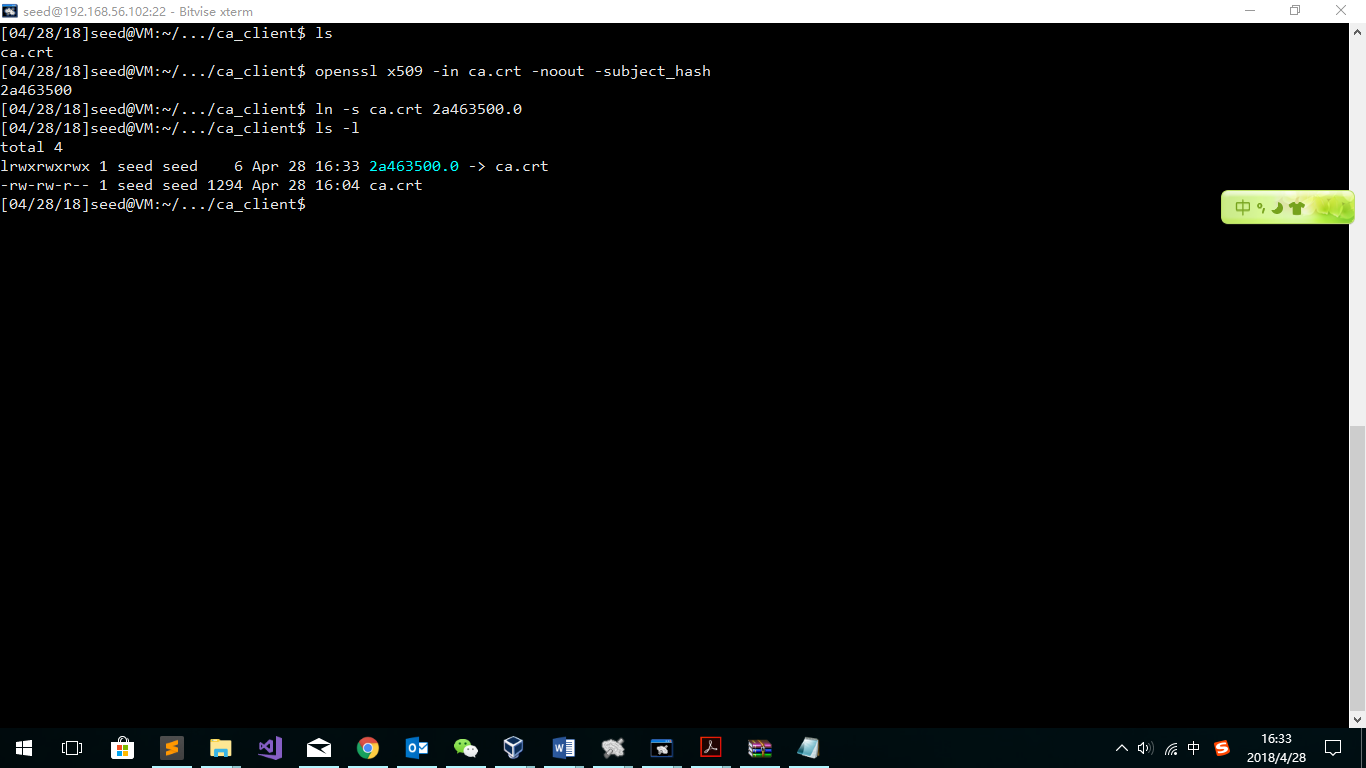
### Settings in the Client

Copy the file ca.crt to the client and use openssl to find the hash function to rename the file.

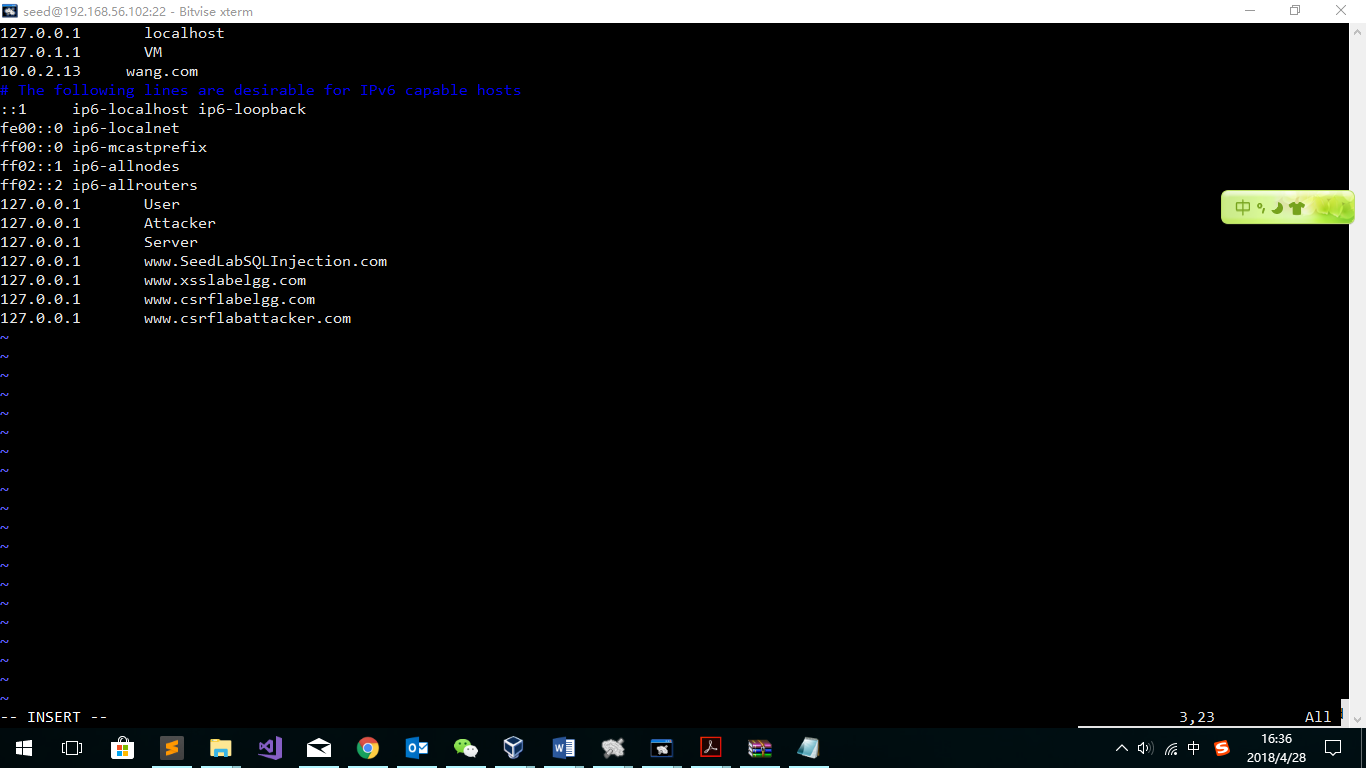
openssl x509 -in ca.crt -noout -subject\_hash

ln -s ca.crt 2a463500.0

ls -l

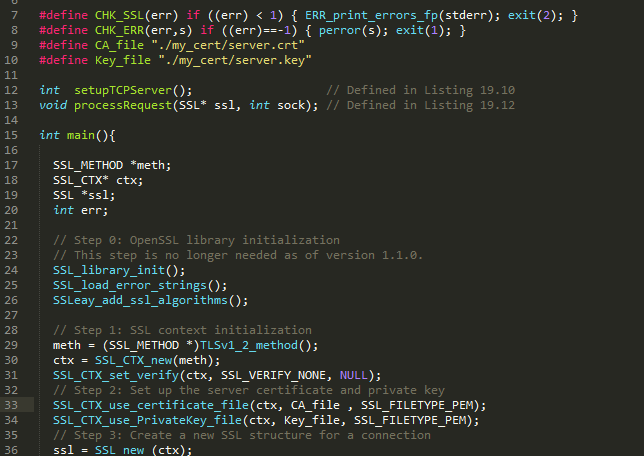


Modify /etc/hosts



### Settings on the server

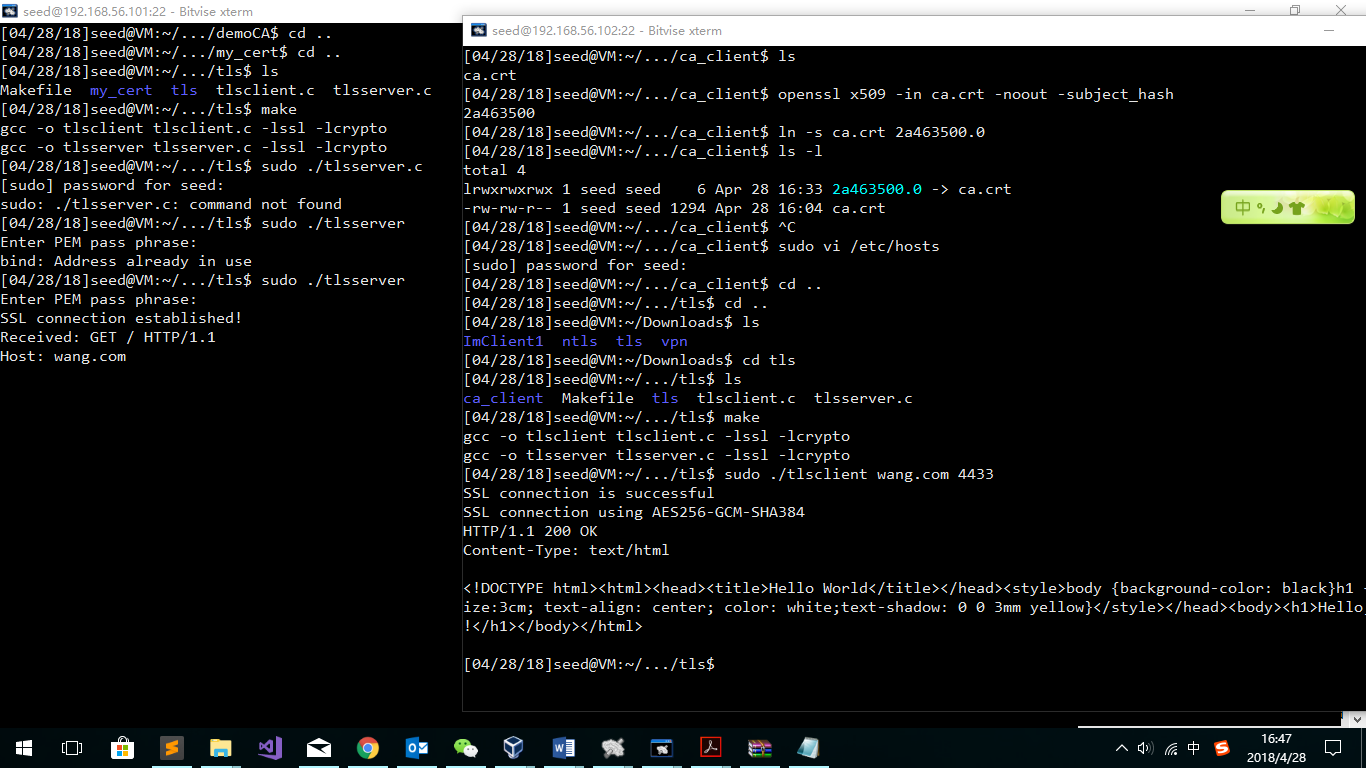
The commands of signature is done on the server’s machine. As a result, what we need to do is to modify the codes to find the right key file and the certification file.



### Test the codes

Server: sudo ./tlsserver

Client: sudo ./tlsclient wang.com 4433



**Observation:**

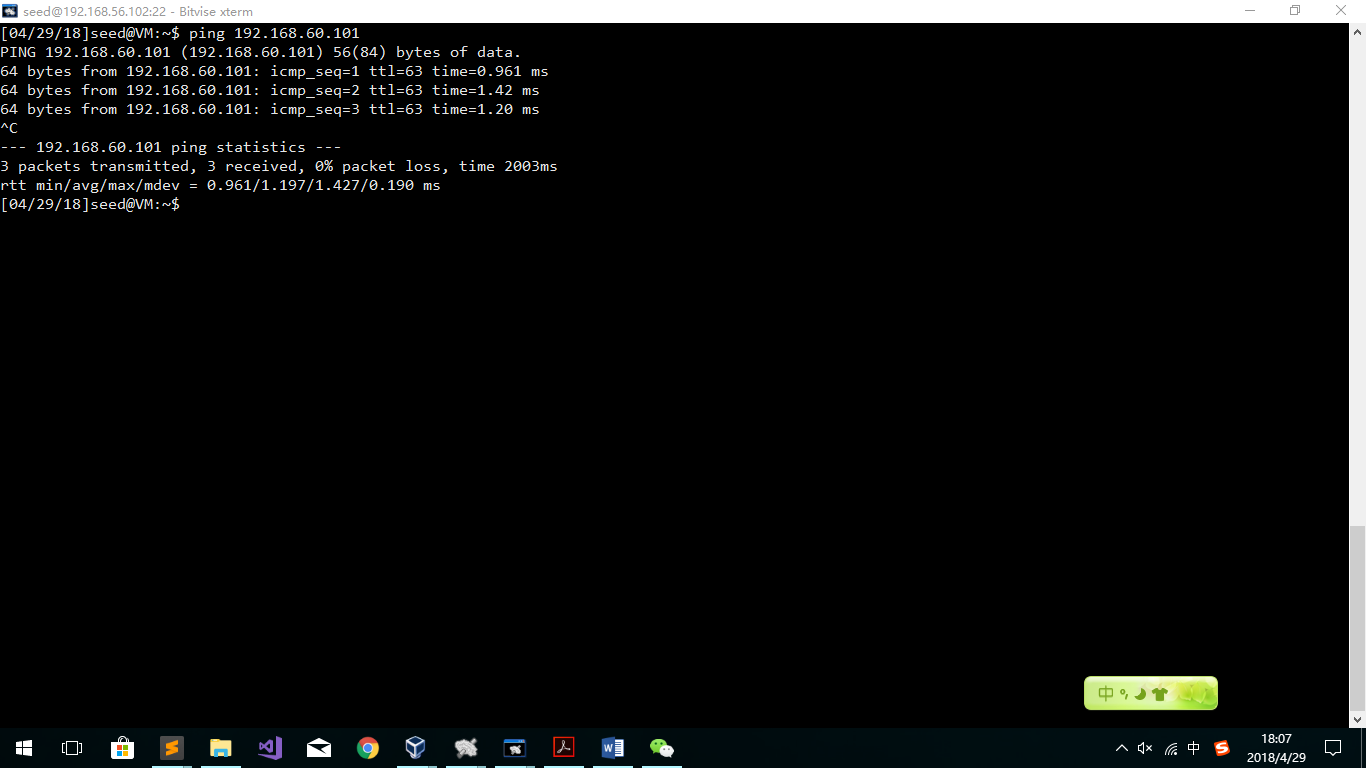
Obviously, the TLS coonection is successfully set.

Until now, I have had my own certification files and a good- working tsl connection. What I need to do is to replace the udp part with tls in the vpn codes.

### Modify VPN codes:

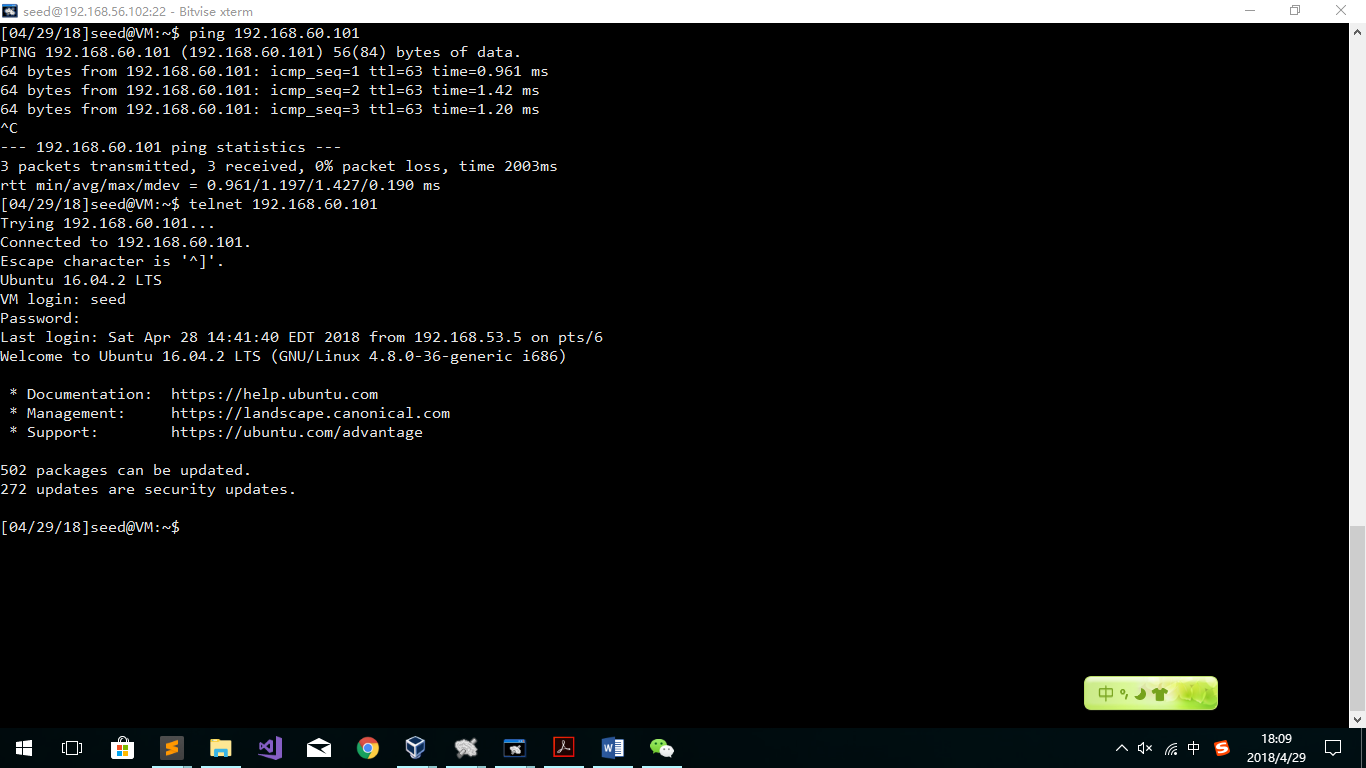
Combine the tls codes and vpn codes to build the connection in a secure tunnel. Codes are at the end of this lab report.

After running the codes and doing configurations and routes, we can test the codes as follows:



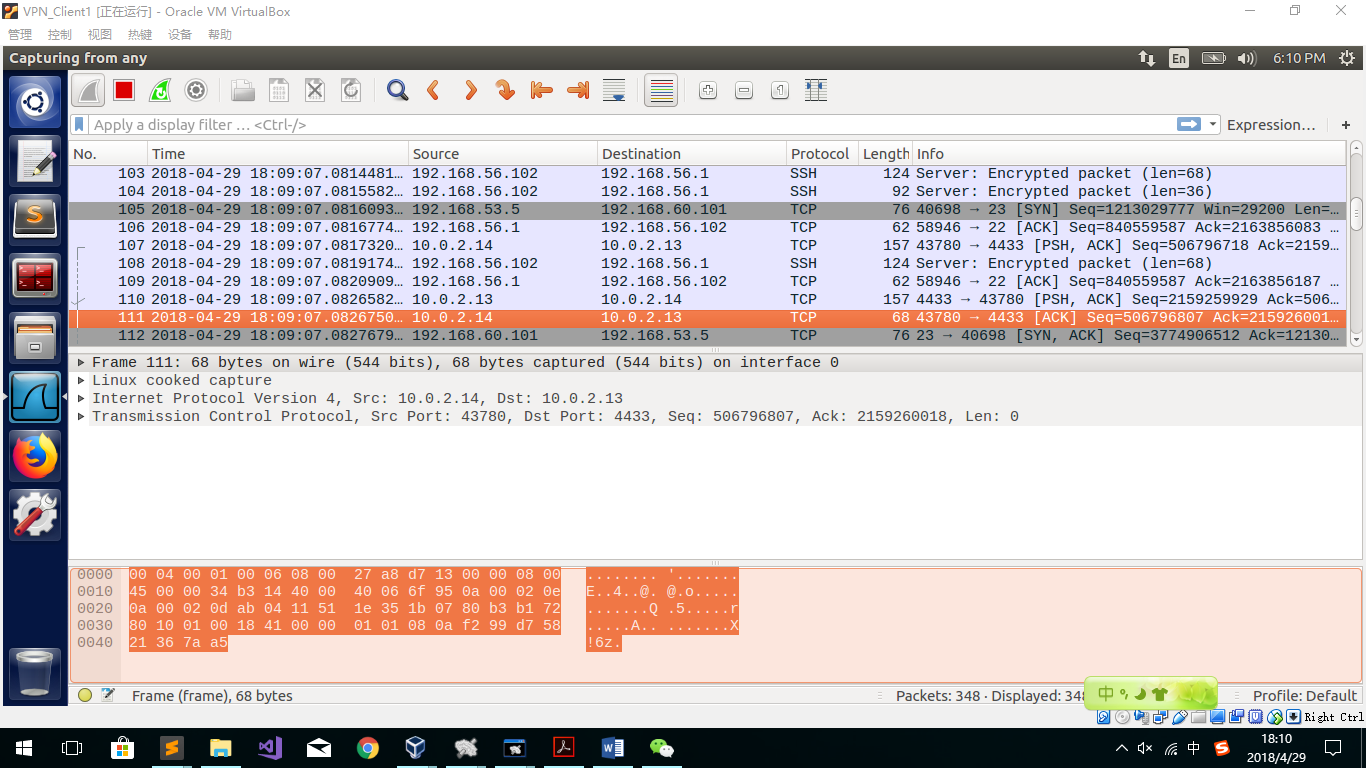
Obviously, vpn is still making effects.

Let us do telnet and watch those packets via wireshark:



It seems that the telnet is working good.

Wireshark shot screen:

as we can see in the screen, there are no UDP packets between the client(10.0.2.14) and the server (10.0.2.13). Instead, we can see those packets have become TCP packets. As we know, telnetting is using plaintext to transform username and passwords. But now, we cannot see information in an encrypted TCP packet. As a result, we can see that the tunnel is well encrypted.

## Task 4: Authenticating the VPN Server

### Questions

#### verifying that the server certificate is valid,

SSL\_CTX\_set\_verify(ctx, SSL\_VERIFY\_PEER, verify\_callback);

Having this sentence, in the SSL\_new(ctx) command, the system will verify the host for its certification. If the last parameter is null, it will use a default callback function.

#### verifying that the server is the owner of the certificate

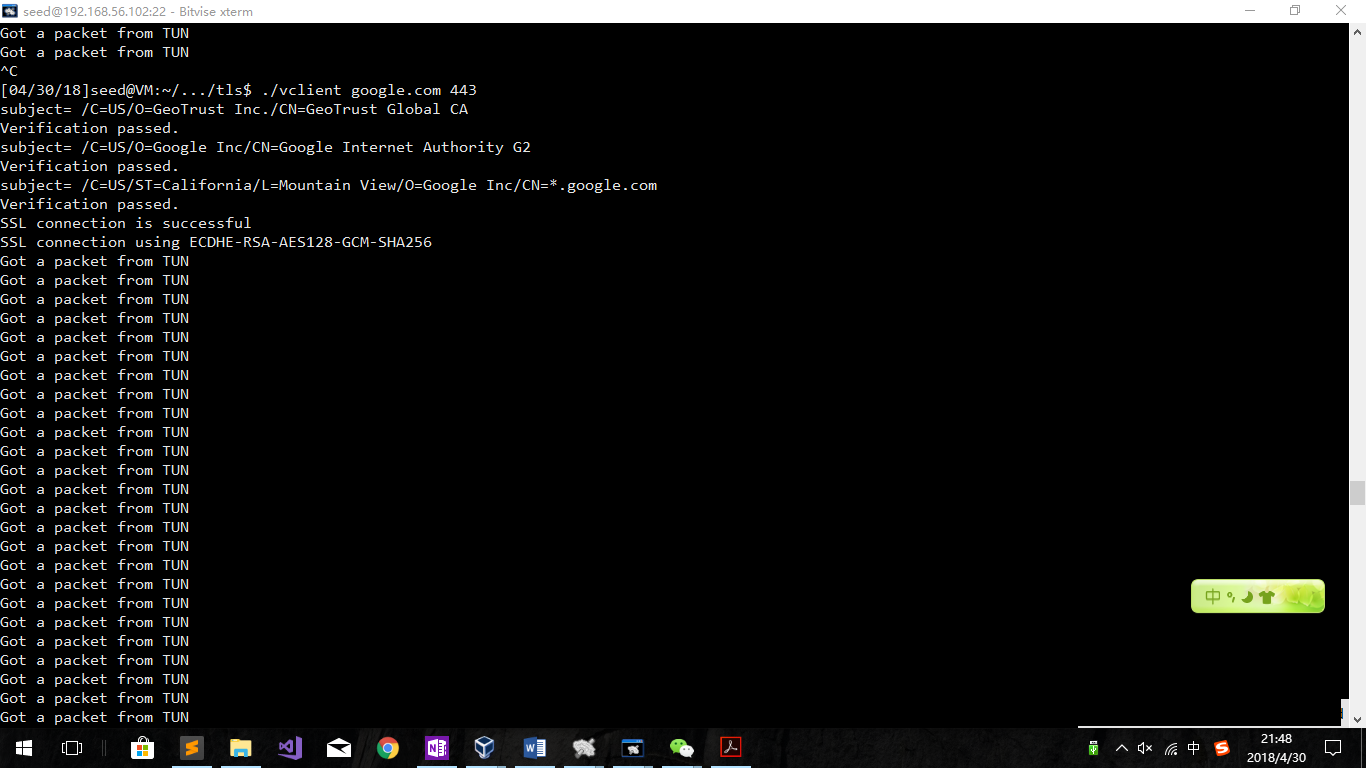
SSL\_CTX\_set\_verify(ctx, SSL\_VERIFY\_PEER, verify\_callback);

Having this sentence, in the SSL\_new(ctx) command, the system will verify whether the server is the owner of the certificate. If the last parameter is null, it will use a default callback function.

#### verifying that the server is the intended server

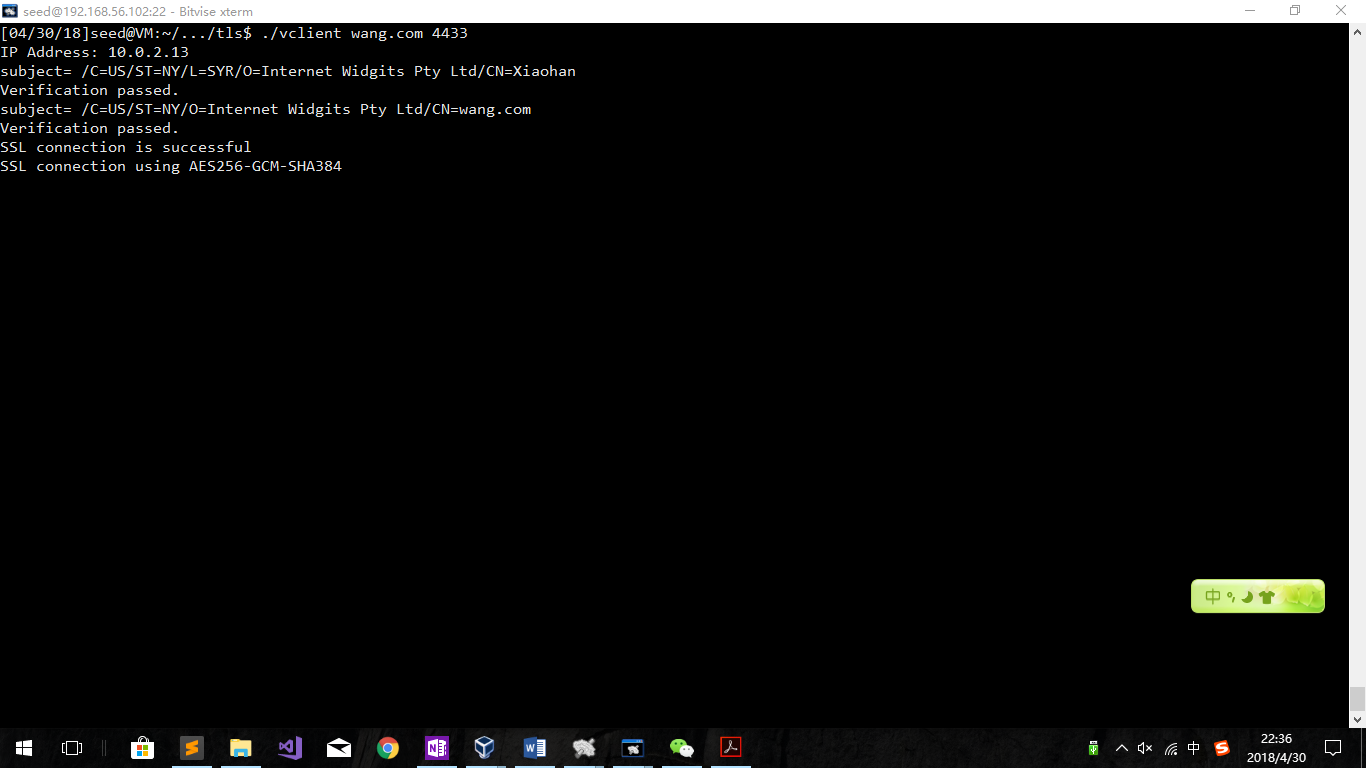
X509\_VERIFY\_PARAM\_set1\_host(vpm, hostname, 0);

\*Without which, the result is like:

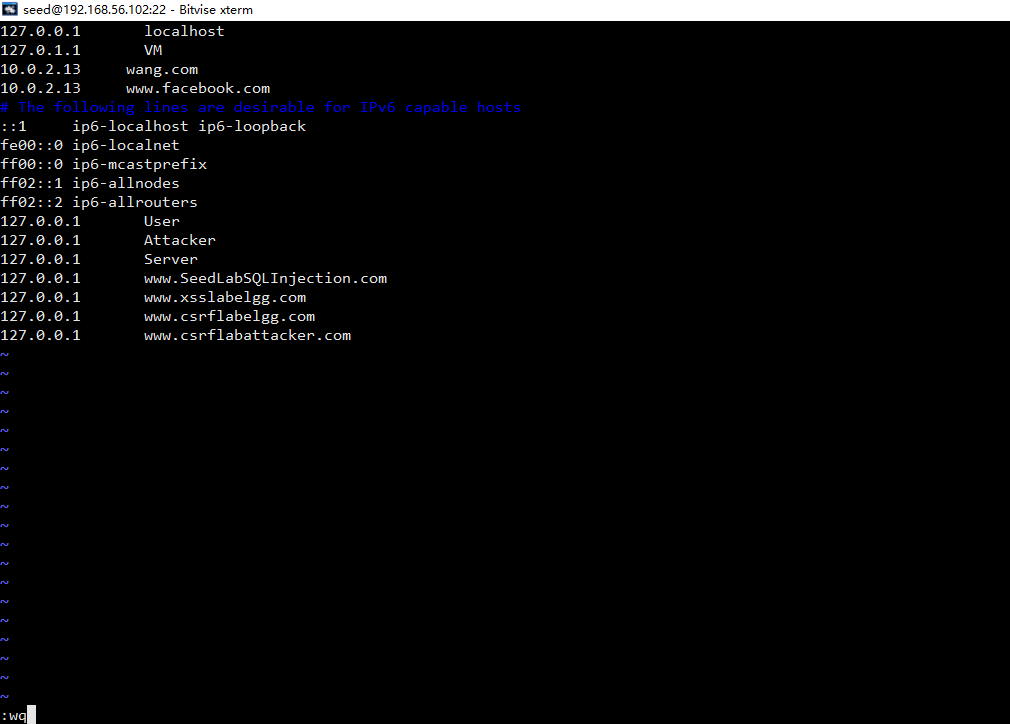


### Tests:

The successful one: sudo ./vclient wang 4433

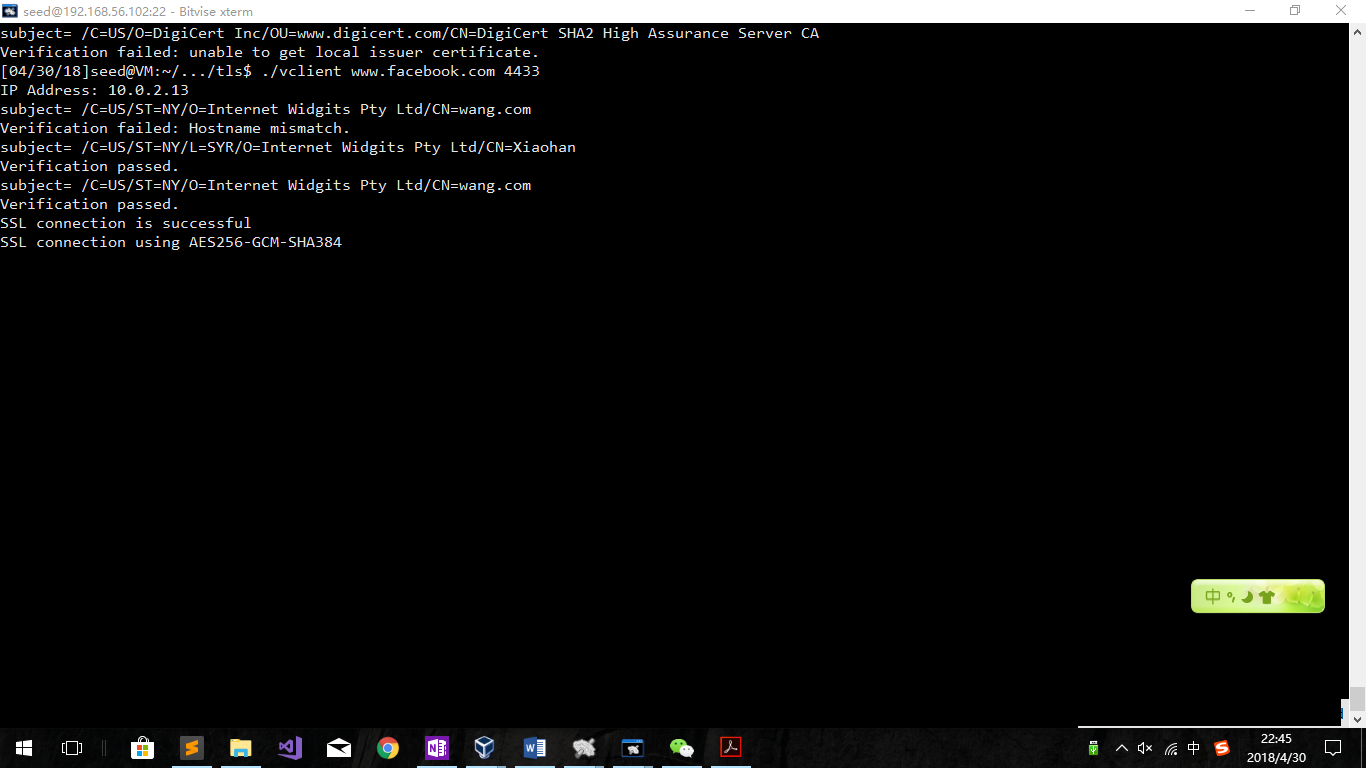


The failure one:



Modify the hosts file to redirect [www.facebook.com](http://www.facebook.com)

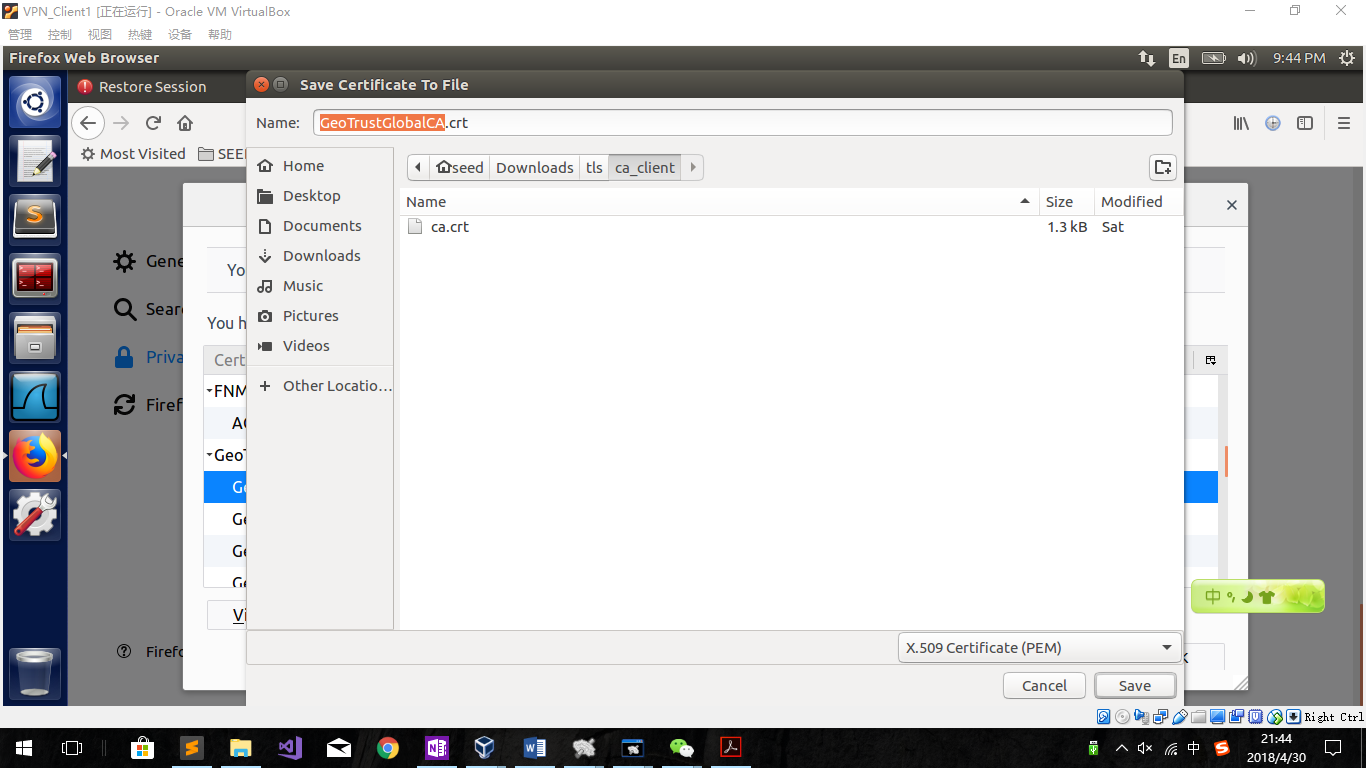
And then do ./vclient [www.facebook.com](http://www.facebook.com) 4433



Hostname mismatch! Which means we have failed using other hostname.

### Google Configure and test:

Export the google’s cert into ca\_client folder.

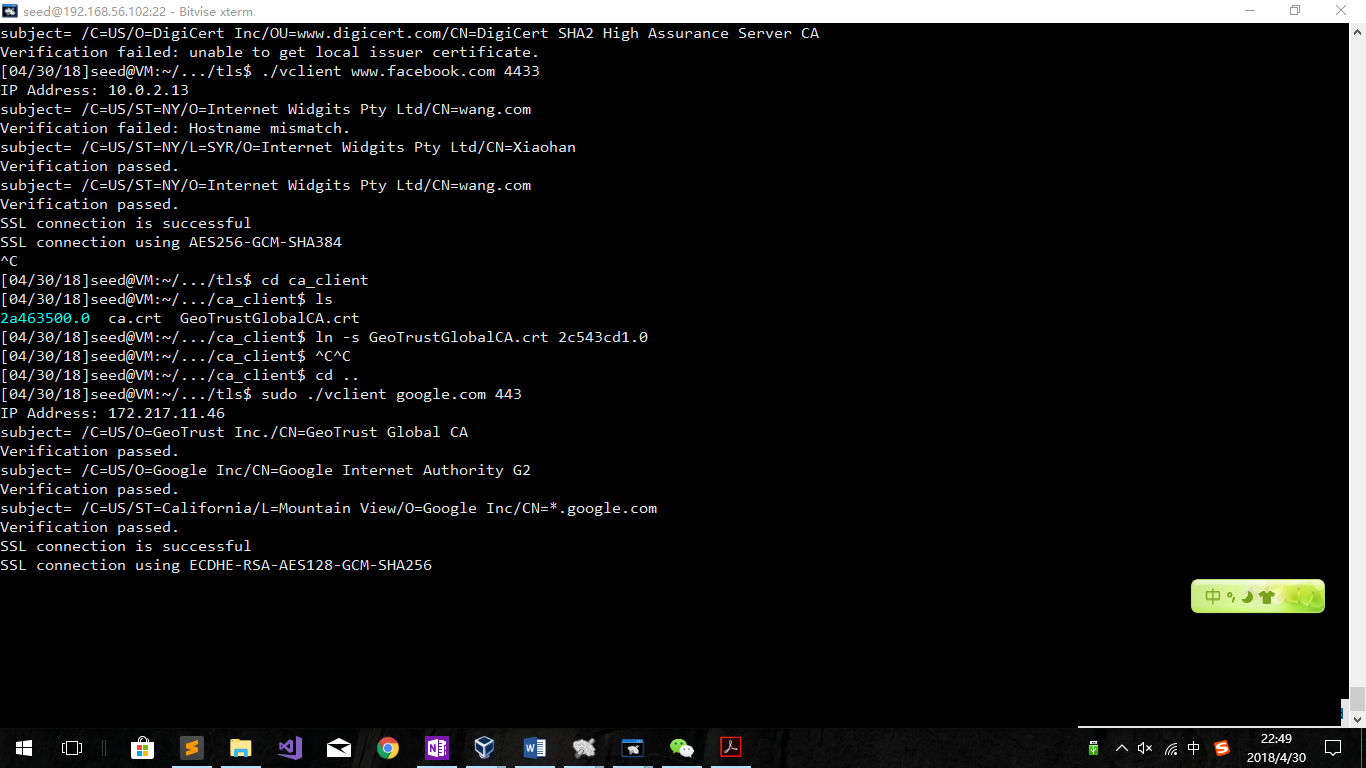


**openssl x509 -in GeoTrustGlobalCA.crt -noout -subject\_hash**

**2c543cd1**

ln -s GeoTrustGlobalCA.crt **2c543cd1.0**

**Try ./vclient google.com 443**

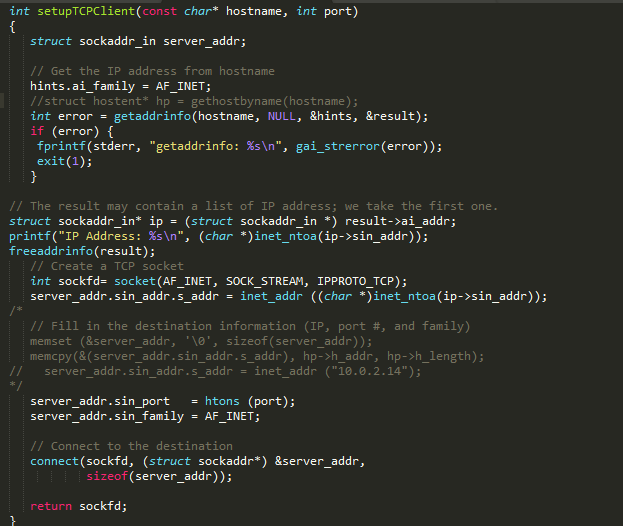


Observation:

After we have the ca files in the ca\_client folder, we can use our program to connect google.com.

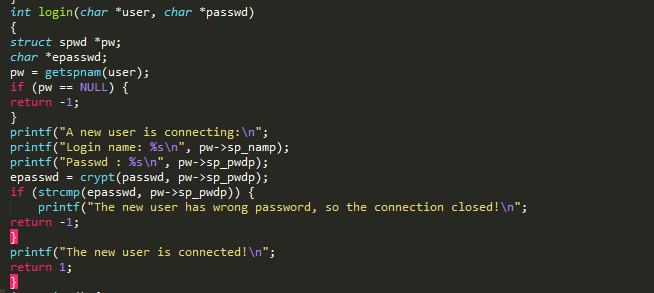
### Addition about getaddrinfo:

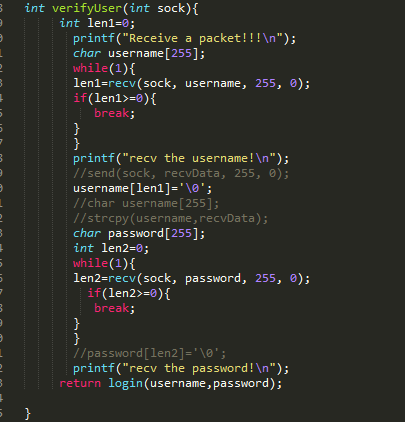
I have modified codes to use getaddrinfo command instead of gethostbyname command. Codes part screens as follows:

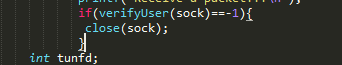


## Task 5: Authenticating the VPN Client

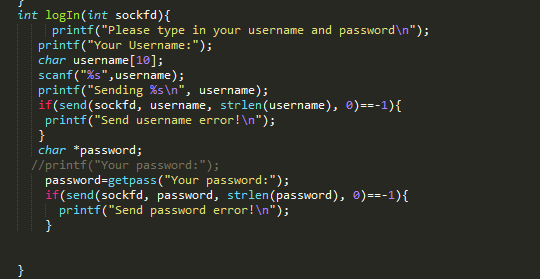
In the server:

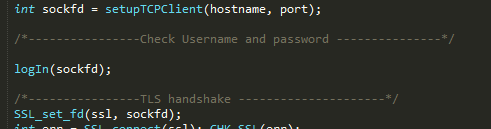




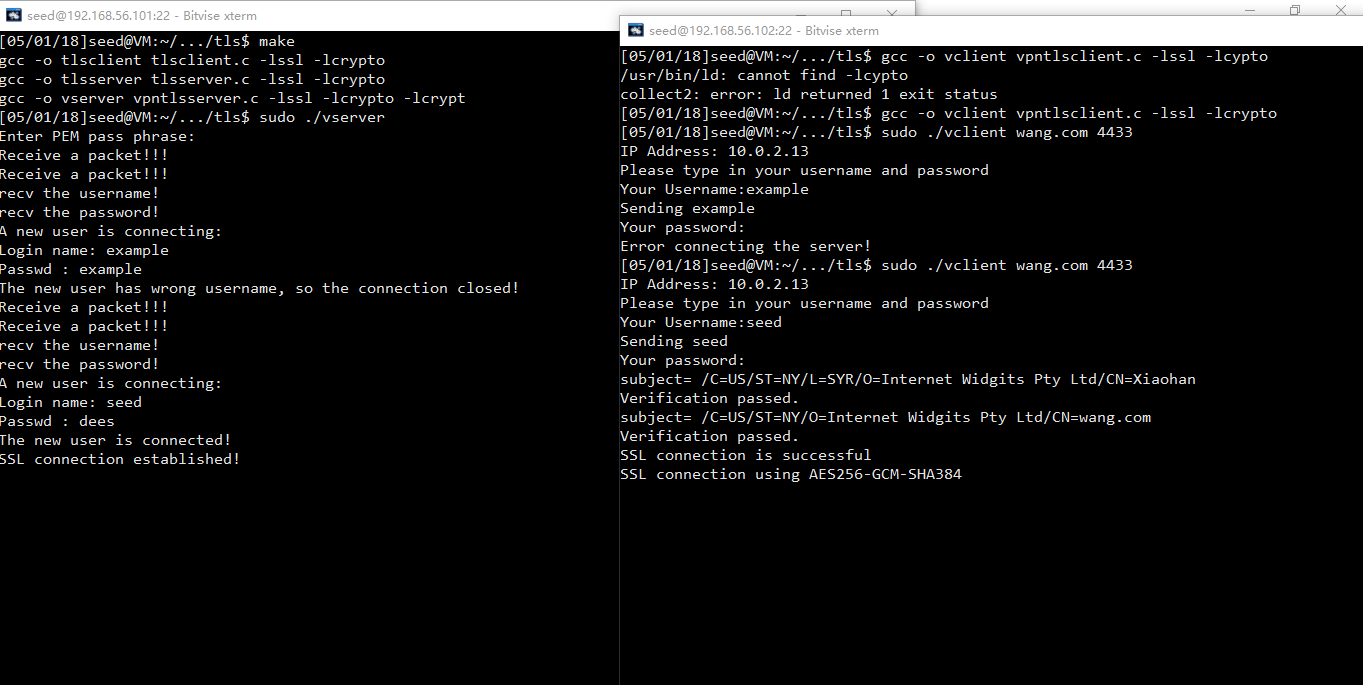


In the client:





Test it:



**Observation:**

Firstly, we use “example” as username as well as password. As a result, the server print that the username and password don’t match, so the server closed the socket and the TLS connection is not established.

Then, we use “seed” and “dees” as the username and password. After checking the shadow file, the server finds this client passed the verification. As a result, the server does not close the socket and the TLS connection is established after that.

## Task 6: Supporting Multiple Clients

To have the multiple clients, we can’t let the server’s every child process listen&reading from the tun0. As a result, the

if (FD\_ISSET(tunfd, &readFDSet)) tunSelected(tunfd, ssl);

can be a part of the parent process.

We need to create more than 1 process. That is why we need to add a for loop.

Using the pipe function, we can let the parent process read the packets from tun0 and send them. As a result, the child process should do things like this:

That is all what I can think&do. I did not implement the multiple client function.

## Codes:

### Codes of Client:

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <arpa/inet.h>

#include <linux/if.h>

#include <linux/if\_tun.h>

#include <sys/ioctl.h>

#include <openssl/ssl.h>

#include <openssl/err.h>

#include <netdb.h>

#define BUFF\_SIZE 2000

#define PORT\_NUMBER 55555

#define SERVER\_IP "10.0.2.13"

#define CHK\_SSL(err) if ((err) < 1) { ERR\_print\_errors\_fp(stderr);printf("Error connecting the server!\n"); exit(2); }

#define CA\_DIR "ca\_client"

struct sockaddr\_in peerAddr;

struct addrinfo hints, \*result;

int verify\_callback(int preverify\_ok, X509\_STORE\_CTX \*x509\_ctx)

{

char buf[300];

X509\* cert = X509\_STORE\_CTX\_get\_current\_cert(x509\_ctx);

X509\_NAME\_oneline(X509\_get\_subject\_name(cert), buf, 300);

printf("subject= %s\n", buf);

if (preverify\_ok == 1) {

printf("Verification passed.\n");

} else {

int err = X509\_STORE\_CTX\_get\_error(x509\_ctx);

printf("Verification failed: %s.\n",

X509\_verify\_cert\_error\_string(err));

}

}

SSL\* setupTLSClient(const char\* hostname)

{

// Step 0: OpenSSL library initialization

// This step is no longer needed as of version 1.1.0.

SSL\_library\_init();

SSL\_load\_error\_strings();

SSLeay\_add\_ssl\_algorithms();

SSL\_METHOD \*meth;

SSL\_CTX\* ctx;

SSL\* ssl;

meth = (SSL\_METHOD \*)TLSv1\_2\_method();

ctx = SSL\_CTX\_new(meth);

SSL\_CTX\_set\_verify(ctx, SSL\_VERIFY\_PEER, verify\_callback);

if(SSL\_CTX\_load\_verify\_locations(ctx,NULL, CA\_DIR) < 1){

printf("Error setting the verify locations. \n");

exit(0);

}

ssl = SSL\_new (ctx);

X509\_VERIFY\_PARAM \*vpm = SSL\_get0\_param(ssl);

X509\_VERIFY\_PARAM\_set1\_host(vpm, hostname, 0);

return ssl;

}

int setupTCPClient(const char\* hostname, int port)

{

struct sockaddr\_in server\_addr;

// Get the IP address from hostname

hints.ai\_family = AF\_INET;

//struct hostent\* hp = gethostbyname(hostname);

int error = getaddrinfo(hostname, NULL, &hints, &result);

if (error) {

fprintf(stderr, "getaddrinfo: %s\n", gai\_strerror(error));

exit(1);

}

// The result may contain a list of IP address; we take the first one.

struct sockaddr\_in\* ip = (struct sockaddr\_in \*) result->ai\_addr;

printf("IP Address: %s\n", (char \*)inet\_ntoa(ip->sin\_addr));

freeaddrinfo(result);

// Create a TCP socket

int sockfd= socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

server\_addr.sin\_addr.s\_addr = inet\_addr ((char \*)inet\_ntoa(ip->sin\_addr));

/\*

// Fill in the destination information (IP, port #, and family)

memset (&server\_addr, '\0', sizeof(server\_addr));

memcpy(&(server\_addr.sin\_addr.s\_addr), hp->h\_addr, hp->h\_length);

// server\_addr.sin\_addr.s\_addr = inet\_addr ("10.0.2.14");

\*/

server\_addr.sin\_port = htons (port);

server\_addr.sin\_family = AF\_INET;

// Connect to the destination

connect(sockfd, (struct sockaddr\*) &server\_addr,

sizeof(server\_addr));

return sockfd;

}

int createTunDevice() {

int tunfd;

struct ifreq ifr;

memset(&ifr, 0, sizeof(ifr));

ifr.ifr\_flags = IFF\_TUN | IFF\_NO\_PI;

tunfd = open("/dev/net/tun", O\_RDWR);

ioctl(tunfd, TUNSETIFF, &ifr);

return tunfd;

}

void tunSelected(int tunfd, SSL \*ssl){

int len;

char buff[BUFF\_SIZE];

printf("Got a packet from TUN\n");

bzero(buff, BUFF\_SIZE);

read(tunfd, buff, sizeof(buff));

SSL\_write(ssl, buff, sizeof(buff));

}

void socketSelected (int tunfd, SSL \*ssl){

int len;

char buff[BUFF\_SIZE];

printf("Got a packet from the tunnel\n");

bzero(buff, BUFF\_SIZE);

SSL\_read(ssl, buff, sizeof(buff));

write(tunfd, buff, sizeof(buff));

}

void logIn(SSL\* ssl){

printf("Please type in your username and password\n");

printf("Your Username:");

char username[BUFF\_SIZE];

scanf("%s",username);

SSL\_write(ssl, username,BUFF\_SIZE);

char \*password;

//printf("Your password:");

password=getpass("Your password:");

SSL\_write(ssl, password, BUFF\_SIZE);

}

int main (int argc, char \* argv[]) {

char \*hostname = "wang.com";

int port = 4433;

int tunfd = createTunDevice();

if (argc > 1) hostname = argv[1];

if (argc > 2) port = atoi(argv[2]);

/\*----------------TLS initialization ----------------\*/

SSL \*ssl = setupTLSClient(hostname);

/\*----------------Create a TCP connection ---------------\*/

int sockfd = setupTCPClient(hostname, port);

/\*----------------Check Username and password ---------------\*/

SSL\_set\_fd(ssl, sockfd);

/\*----------------TLS handshake ---------------------\*/

int err = SSL\_connect(ssl); CHK\_SSL(err);

printf("SSL connection is successful\n");

printf ("SSL connection using %s\n", SSL\_get\_cipher(ssl));

logIn(ssl);

char test[BUFF\_SIZE];

SSL\_read(ssl, test, sizeof(test));

if(test[0]=='n'){

printf("You put in the wrong username/password! The program will be terminated！\n");

exit(0);

}

printf("Receive the confirm. The VPN Started!!\n");

//SSL\_read(ssl,test,BUFF\_SIZE);

/\*----------------Send/Receive data --------------------\*/

//sprintf(sendBuf, "GET / HTTP/1.1\nHost: %s\n\n", hostname);

while (1) {

fd\_set readFDSet;

FD\_ZERO(&readFDSet);

FD\_SET(sockfd, &readFDSet);

FD\_SET(tunfd, &readFDSet);

select(FD\_SETSIZE, &readFDSet, NULL, NULL, NULL);

if (FD\_ISSET(tunfd, &readFDSet)) tunSelected(tunfd, ssl);

if (FD\_ISSET(sockfd, &readFDSet)) socketSelected(tunfd, ssl);

}

}

### Codes of server:

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <arpa/inet.h>

#include <linux/if.h>

#include <linux/if\_tun.h>

#include <sys/ioctl.h>

#include <openssl/ssl.h>

#include <openssl/err.h>

#include <netdb.h>

#include <shadow.h>

#include <crypt.h>

#define PORT\_NUMBER 55555

#define BUFF\_SIZE 2000

#define CHK\_SSL(err) if ((err) < 1) { ERR\_print\_errors\_fp(stderr); exit(2); }

#define CHK\_ERR(err,s) if ((err)==-1) { perror(s); exit(1); }

#define CA\_file "./my\_cert/server.crt"

#define Key\_file "./my\_cert/server.key"

int setupTCPServer()

{

struct sockaddr\_in sa\_server;

int listen\_sock;

listen\_sock= socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

CHK\_ERR(listen\_sock, "socket");

memset (&sa\_server, '\0', sizeof(sa\_server));

sa\_server.sin\_family = AF\_INET;

sa\_server.sin\_addr.s\_addr = INADDR\_ANY;

sa\_server.sin\_port = htons (4433);

int err = bind(listen\_sock, (struct sockaddr\*)&sa\_server, sizeof(sa\_server));

CHK\_ERR(err, "bind");

err = listen(listen\_sock, 5);

CHK\_ERR(err, "listen");

return listen\_sock;

}

int createTunDevice() {

int tunfd;

struct ifreq ifr;

memset(&ifr, 0, sizeof(ifr));

ifr.ifr\_flags = IFF\_TUN | IFF\_NO\_PI;

tunfd = open("/dev/net/tun", O\_RDWR);

ioctl(tunfd, TUNSETIFF, &ifr);

return tunfd;

}

int tunSelected(int tunfd, SSL \*ssl){

char buff[BUFF\_SIZE];

printf("Got a packet from TUN\n");

bzero(buff, BUFF\_SIZE);

read(tunfd, buff, sizeof(buff));

SSL\_write(ssl, buff, sizeof(buff));

return 1;

}

int socketSelected (int tunfd, SSL \*sockfd){

char buff[BUFF\_SIZE];

printf("Got a packet from the tunnel\n");

bzero(buff, BUFF\_SIZE);

SSL\_read(sockfd, buff, sizeof(buff));

write(tunfd, buff, sizeof(buff));

return 1;

}

int login(char \*user, char \*passwd)

{

printf("A new user is connecting:\n");

printf("Login name: %s\n", user);

printf("Passwd : %s\n", passwd);

struct spwd \*pw;

char \*epasswd;

pw = getspnam(user);

if (pw == NULL) {

printf("The new user has wrong username, so the connection closed!\n");

return -1;

}

epasswd = crypt(passwd, pw->sp\_pwdp);

if (strcmp(epasswd, pw->sp\_pwdp)) {

printf("The new user has wrong password, so the connection closed!\n");

return -1;

}

printf("The new user is connected!\n");

return 1;

}

int verifyUser(SSL\* ssl){

int len1=0;

printf("Receive a packet!!!\n");

char username[BUFF\_SIZE];

len1=SSL\_read(ssl, username, BUFF\_SIZE);

printf("recv the username!\n");

//send(sock, recvData, 255, 0);

username[len1]='\0';

//char username[255];

//strcpy(username,recvData);

char password[BUFF\_SIZE];

SSL\_read(ssl, password, BUFF\_SIZE);

//password[len2]='\0';

printf("recv the password!\n");

int m=login(username,password);

char sendBuf[200];

if(m==1){

sprintf(sendBuf, "hi");

}

else{

sprintf(sendBuf,"no");

}

SSL\_write(ssl,sendBuf,strlen(sendBuf));

return m;

}

int main () {

SSL\_METHOD \*meth;

SSL\_CTX\* ctx;

SSL \*ssl;

int err;

// Step 0: OpenSSL library initialization

// This step is no longer needed as of version 1.1.0.

SSL\_library\_init();

SSL\_load\_error\_strings();

SSLeay\_add\_ssl\_algorithms();

// Step 1: SSL context initialization

meth = (SSL\_METHOD \*)TLSv1\_2\_method();

ctx = SSL\_CTX\_new(meth);

SSL\_CTX\_set\_verify(ctx, SSL\_VERIFY\_NONE, NULL);

// Step 2: Set up the server certificate and private key

SSL\_CTX\_use\_certificate\_file(ctx, CA\_file , SSL\_FILETYPE\_PEM);

SSL\_CTX\_use\_PrivateKey\_file(ctx, Key\_file, SSL\_FILETYPE\_PEM);

// Step 3: Create a new SSL structure for a connection

ssl = SSL\_new (ctx);

int tunfd;

tunfd = createTunDevice();

//SSL\* ssl=setupTLSServer();

struct sockaddr\_in sa\_client;

size\_t client\_len;

int listen\_sock = setupTCPServer();

while(1){

int sock = accept(listen\_sock, (struct sockaddr\*)&sa\_client, &client\_len);

if (fork() == 0) { // The child process

SSL\_set\_fd (ssl, sock);

close (listen\_sock);

printf("Receive a packet!!!\n");

int err = SSL\_accept (ssl);

CHK\_SSL(err);

printf ("SSL connection established!\n");

if(verifyUser(ssl)<0){

close(sock);

exit(2);

}

//vertigy

while(1){

fd\_set readFDSet;

FD\_ZERO(&readFDSet);

FD\_SET(sock, &readFDSet);

FD\_SET(tunfd, &readFDSet);

select(FD\_SETSIZE, &readFDSet, NULL, NULL, NULL);

if (FD\_ISSET(tunfd, &readFDSet)) tunSelected(tunfd, ssl);

if (FD\_ISSET(sock, &readFDSet)) socketSelected(tunfd, ssl);

}

close(sock);

return 0;

} else { // The parent process

close(sock);

}

}

}

## Instructions:

Server configure:

sudo ifconfig tun0 192.168.53.1/24 up

sudo sysctl net.ipv4.ip\_forward=1

sudo route add -net 192.168.53.0/24 dev tun0

Client1 configure:

sudo ifconfig tun0 192.168.53.5/24 up

sudo route add -net 192.168.60.0/24 tun0

sudo route add -net 192.168.53.0/24 dev tun0

Client2 configure:

sudo ifconfig tun0 192.168.53.7/24 up

sudo route add -net 192.168.60.0/24 tun0

sudo route add -net 192.168.53.0/24 dev tun0

Make:

gcc -o vserver vpntlsserver.c -lssl -lcrypto

gcc -o vclient vpntlsclient.c -lssl -lcrypto -lcrypt

## Security of the System:

Firstly, for the safety of client, the system uses a CA certificate to find if the CA is valid, if the one I am connecting to own this CA and if the hostname is right. In the real world, if a server is signed by a root-CA and we can ensure that the server is not fake, then most likely the server can be trusted.

Then, for the safety of the server, we have login certificate on the server side. If it is on the server side, the client’s code may be unpacked to jump this step. The valid user information is stored in the server’s shadow file. Only when the client passes the verification can the tunnel connection be settled.

Thirdly, packet transportation between client and server is using TLS structure, which means that every packet is encrypted before sending to the server. If someone is sniffing in the middle, he will get nothing useful.