

Information Security

Transport Layer Security



Ambreen Kanwal
University of Sahiwal

Transport Layer Security (TLS) Lab

Task1: TLS Client:

Task 1.A: TLS handshake

Use the following code, in order to print out the various algorithms used by TLS, you can add the `pprint.pprint(ssock.cipher())` statement on the basis of the experimental manual code:

```
#!/usr/bin/python3

import socket, ssl, sys, pprint

hostname = sys.argv[1]

port = 443

cadir = '/etc/ssl/certs'

# Set up the TLS context

context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)    # For Ubuntu 16.04 VM

context.load_verify_locations(capath=cadir)

context.verify_mode = ssl.CERT_REQUIRED

context.check_hostname = True


# Create TCP connection

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.connect((hostname, port))

input("After making TCP connection. Press any key to continue ...")

# Add the TLS
```

```

ssock = context.wrap_socket(sock, server_hostname=hostname,
                             do_handshake_on_connect=False)

ssock.do_handshake() # Start the handshake

print("=== Cipher used: {}".format(ssock.cipher()))

print("=== Server hostname: {}".format(ssock.server_hostname))

print("=== Server certificate:")

pprint.pprint(ssock.getpeercert())

pprint.pprint(context.get_ca_certs())

input("After TLS handshake. Press any key to continue ...")

# Close the TLS Connection

ssock.shutdown(socket.SHUT_RDWR)

ssock.close()

```

TLS requires four steps to provide secure communication for sockets.

- The first step is to create a TLS context object. The object saves our preference settings for certificate authentication and encryption algorithm selection. The context part of the code.
- The second step is to establish a TCP handshake protocol, make a TCP connection, and the code sock part.
- The third step is to call the context object created in the first step and use the wrap_socket() method on it, which means that the OpenSSL library is responsible for controlling our TCP connection. Then exchange the necessary handshake information with the communicating party and establish an encrypted link. The ssock = context.wrap_socket... part of the code

- The last step is to use the `ssl_sock` object returned by the `wrap_socket()` call for all subsequent communications. Subsequent communication needs to use a format similar to `ssock.method name`.

Question

1. print out

```
[12/24/20]seed@VM:~/lab_Tls$ ./handshake.py www.baidu.com
After making TCP connection. Press any key to continue ...
('ECDHE-RSA-AES128-GCM-SHA256', 'TLSv1/SSLv3', 128)
None
After handshake. Press any key to continue ...
[12/24/20]seed@VM:~/lab_Tls$
```

https://blog.csdn.net/C_Ronaldo__

That is ECDHE-RSA-AES128-GCM-SHA256, the meaning of each parameter is as follows:

Protocol:

Transport Layer Security (TLS)

Key Exchange:

Elliptic Curve Diffie-Hellman Ephemeral (ECDHE)

Authentication:

Rivest Shamir Adleman algorithm (RSA)

Encryption:

Advanced Encryption Standard with 128bit key in Galois/Counter mode (AES 128 GCM)

Hash:

Secure Hash Algorithm 256 (SHA256)

Included in RFC:

RFC 5289

Machine-readable:

application/json

https://blog.csdn.net/C_Ronaldo__

That is, use the AES128-GCM algorithm for encrypted communication

2.

```
[12/24/20]seed@VM:~/lab Tls$ ./handshake.py www.baidu.com
After making TCP connection. Press any key to continue ...
{'OCSP': ('http://ocsp2.globalsign.com/gsignorganizationvalsha2g2',),
 'caIssuers': ('http://secure.globalsign.com/cacert/gsignorganizationvalsha2g2r1.crt',),
 'crlDistributionPoints': ('http://crl.globalsign.com/gsignorganizationvalsha2g2.crl',),
 'issuer': (((('countryName', 'BE'),),
              (('organizationName', 'GlobalSign nv-sa'),),
              (('commonName',
                'GlobalSign Organization Validation CA - SHA256 - G2'),)),
 'notAfter': 'Jul 26 05:31:02 2021 GMT',
 'notBefore': 'Apr 2 07:04:58 2020 GMT',
 'serialNumber': '725878366E9F56E81D418848',
 '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'),
```

3. Store many certificate files:

```
RSA_Security_2048_v3.pem
Secure_Global_CA.pem
SecureSign_RootCA11.pem
SecureTrust_CA.pem
Security_Communication_EV_RootCA1.pem
Security_Communication_RootCA2.pem
Security_Communication_Root_CA.pem
Sonera_Class_1_Root_CA.pem
Sonera_Class_2_Root_CA.pem
ssl-cert-snakeoil.pem
Staat_der_Nederlanden_EV_Root_CA.pem
Staat_der_Nederlanden_Root_CA_-_G2.pem
Staat_der_Nederlanden_Root_CA_-_G3.pem
Staat_der_Nederlanden_Root_CA.pem
Starfield_Class_2_CA.pem
Starfield_Root_Certificate_Authority_-_G2.pem
Starfield_Services_Root_Certificate_Authority_-_G2.pem
StartCom_Certification_Authority_2.pem
StartCom_Certification_Authority_G2.pem
StartCom_Certification_Authority.pem
5-TRUST_Authentication_and_Encryption_Root_CA_2005_BN.pem
```

These files are root CA files, which are used to verify the validity of the certificate sent back by the server when establishing a TLS connection.

4. Use wireshark to capture relevant information about the TLS connection with www.baidu.com, get a series of data packets, and analyze them one by one:

- **TCP handshake:**

No.	Time	Source	Destination	Protocol	Length	Info
1	2020-12-24 00:10:19.595626883	10.0.2.4	192.168.254.245	DNS	73	Standard query 0x9224 A www.baidu.com
2	2020-12-24 00:10:19.599789283	RealtekU12:35:00	Broadcast	ARP	60	Who has 10.0.2.4? Tell 10.0.2.1
3	2020-12-24 00:10:19.599721713	PcsCompu_e9:cf:f4	RealtekU12:35:00	ARP	42	10.0.2.4 is at 00:00:27:e9:cf:f4
4	2020-12-24 00:10:19.600051014	192.168.254.245	10.0.2.4	DNS	132	Standard query response 0x9224 A www.baidu.com CNAME www.a.shifen.com A 180.101.49.11
5	2020-12-24 00:10:19.600259269	10.0.2.4	180.101.49.11	TCP	74	50998 → 443 [SYN] Seq=1119800647 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1119800648
6	2020-12-24 00:10:19.633754961	180.101.49.11	10.0.2.4	TCP	60	443 → 50998 [SYN, ACK] Seq=39877 Ack=1119800648 Win=32768 Len=0 MSS=1460
7	2020-12-24 00:10:19.633936243	10.0.2.4	180.101.49.11	TCP	54	50998 → 443 [ACK] Seq=1119800648 Ack=39878 Win=29200 Len=0

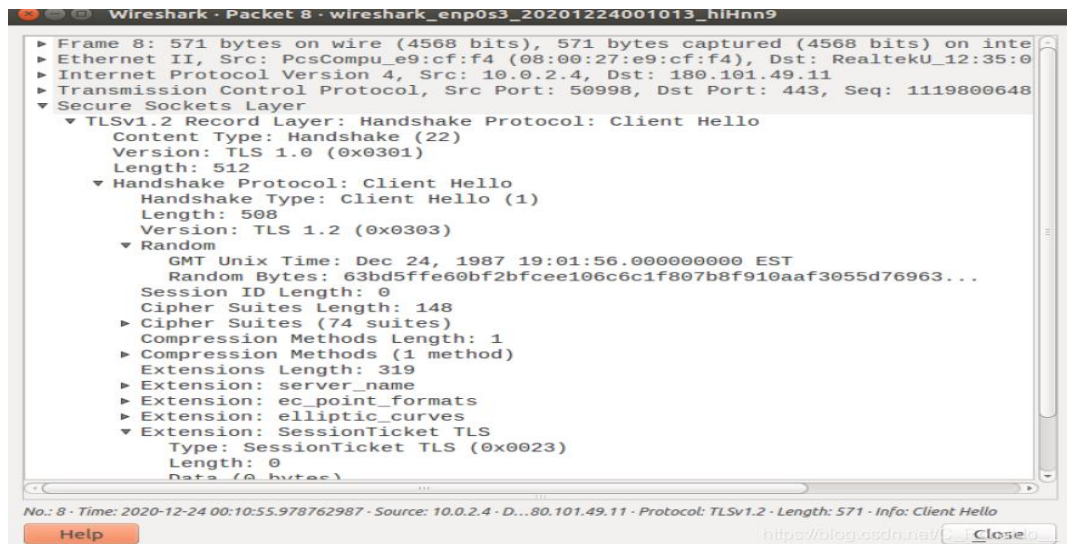
The TCP three-way handshake protocol, it can be seen that the first TCP handshake is intuitive. TLS is based on the transport layer, so it should be based on the TCP connection. Of course, there are also based on UDP, called DTLS. .

- **TLS handshake**

No.	Time	Source	Destination	Protocol	Length	Info
1	2020-12-24 00:10:19.595626883	10.0.2.4	192.168.254.245	DNS	73	Standard query 0x9224 A www.baidu.com
2	2020-12-24 00:10:19.599789283	RealtekU12:35:00	Broadcast	ARP	60	Who has 10.0.2.4? Tell 10.0.2.1
3	2020-12-24 00:10:19.599721713	PcsCompu_e9:cf:f4	RealtekU12:35:00	ARP	42	10.0.2.4 is at 00:00:27:e9:cf:f4
4	2020-12-24 00:10:19.600051014	192.168.254.245	10.0.2.4	DNS	132	Standard query response 0x9224 A www.baidu.com CNAME www.a.shifen.com A 180.101.49.11
5	2020-12-24 00:10:19.600259269	10.0.2.4	180.101.49.11	TCP	74	50998 → 443 [SYN] Seq=1119800647 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1119800648
6	2020-12-24 00:10:19.633754961	180.101.49.11	10.0.2.4	TCP	60	443 → 50998 [SYN, ACK] Seq=39877 Ack=1119800648 Win=32768 Len=0 MSS=1460
7	2020-12-24 00:10:19.633936243	10.0.2.4	180.101.49.11	TCP	54	50998 → 443 [ACK] Seq=1119800648 Ack=39878 Win=29200 Len=0
8	2020-12-24 00:10:55.978762987	10.0.2.4	180.101.49.11	TLSv1.2	571	Client Hello
9	2020-12-24 00:10:56.004798695	180.101.49.11	10.0.2.4	TCP	60	443 → 50998 [ACK] Seq=39878 Ack=1119801165 Win=32251 Len=0
10	2020-12-24 00:10:56.034141896	180.101.49.11	10.0.2.4	TLSv1.2	2974	Server Hello
11	2020-12-24 00:10:56.034182071	10.0.2.4	180.101.49.11	TCP	54	50998 → 443 [ACK] Seq=1119801165 Ack=42798 Win=35640 Len=0
12	2020-12-24 00:10:56.034443116	180.101.49.11	10.0.2.4	TLSv1.2	1300	Certificate, Server Key Exchange, Server Hello Done
13	2020-12-24 00:10:56.034468345	10.0.2.4	180.101.49.11	TCP	54	50998 → 443 [ACK] Seq=1119801165 Ack=44044 Win=37960 Len=0
14	2020-12-24 00:10:56.037544387	10.0.2.4	180.101.49.11	TLSv1.2	180	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
15	2020-12-24 00:10:56.070520571	180.101.49.11	10.0.2.4	TLSv1.2	280	New Session Ticket, Change Cipher Spec, Hello Request, Hello Request
16	2020-12-24 00:10:56.116577109	10.0.2.4	180.101.49.11	TCP	54	50998 → 443 [ACK] Seq=1119801291 Ack=44270 Win=40880 Len=0

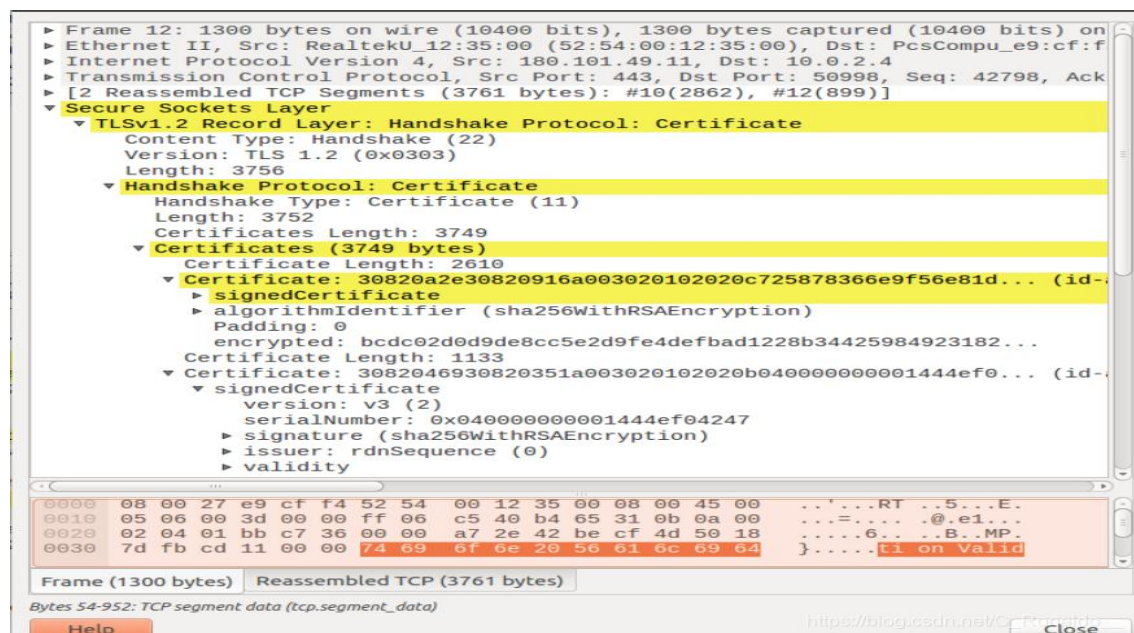
The TLS packets in the picture are all TLS handshake related data packets. The following analysis is one by one:

- The client first sends out Client Hello related data, including all cipher suites supported by the client, client random numbers (used as Nonce), uh... a lot of information
- The server replies to the client Server Hello related data:

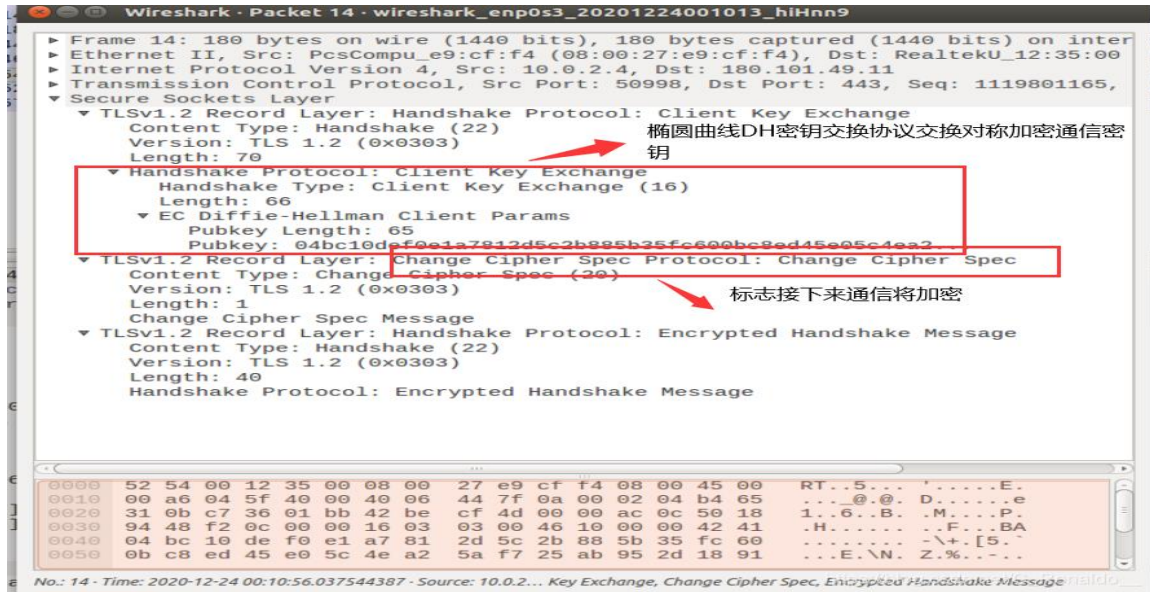




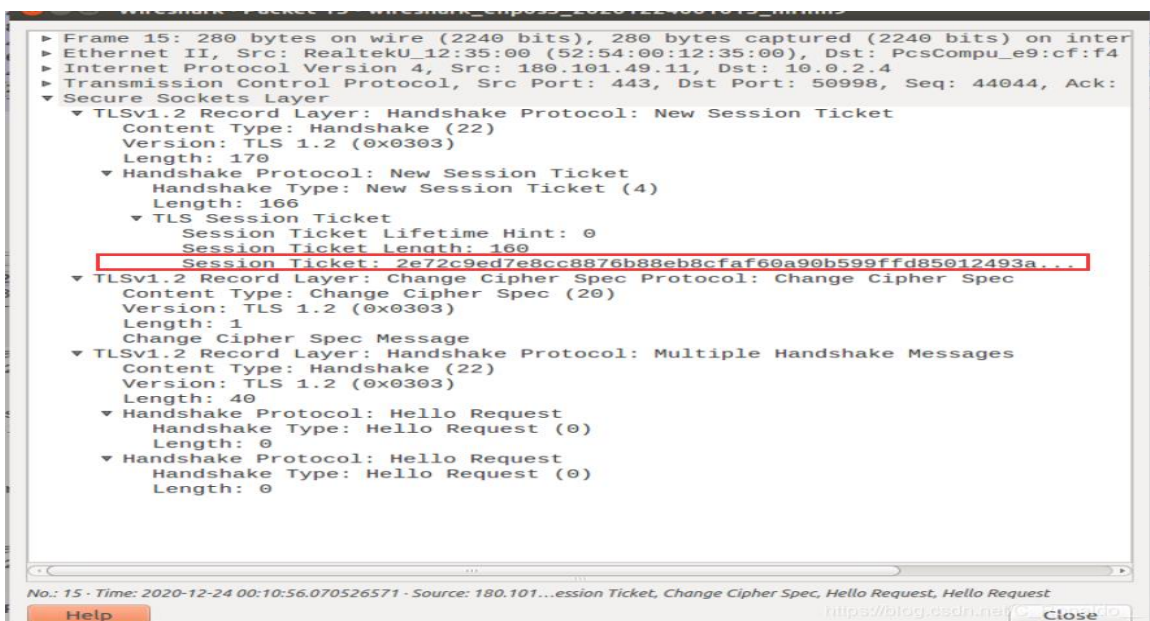
And the certificate (maybe the amount of certificate data is too large, it will be transmitted separately from the above):



- Subsequently, the client verifies the validity of the certificate. If it is valid, the client sends a Key Exchange Client Key Exchange and a Change Cipher Spec (change cipher specification message) to the server, and the client handshake ends:



- The server replies to Change Cipher Spec (change cipher specification message) and a New Session value, and the handshake ends. Sometimes these two will be sent separately, that is, they may not be sent together:



Task 1.b: CA's Certificate

This experiment took me a long time... I took a big detour...

First write the correct experimental ideas and process:

According to the experimental requirements, two URLs need to be selected for testing:

Destination URL: www.baidu.com:

1. In the above task, we print out the information of the certificate sent back by the server and find the issuer information in the certificate information:

```
      ('DNS', '*.hao222.com'),
      ('DNS', '*.haokan.com'),
      ('DNS', '*.pae.baidu.com'),
      ('DNS', '*.vd.bdstatic.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}]
After TLS handshake. Press any key to continue ...
```

https://blog.csdn.net/C_Ronaldo_

2. Go to the `/etc/ssl/certs` directory to find the `.pem` file that matches the `organizationalUnitName` field or `commonName` field of the certificate sent back by the server. As for when to choose which field, I don't know what is needed when testing `www.baidu.com`. The `commonName` field of the file corresponds to the `GlobalSign_Root_CA.pem` file in the `/etc/ssl/certs` directory, and in the subsequent `www.jd.com` test, we found that the file matching the `organizationalUnitName` field information is valid... As for the specific matching mechanism, I don't know... On this point, I took a lot of detours. Finally...

- Copy the GlobalSign_Root_CA.pem file in the /etc/ssl/certs directory to the newly created ./certs folder in the experiment directory, and use the corresponding command to generate the hash value of the file and perform the soft link operation to obtain the following effect:

```
[12/24/20]seed@VM:~/lab_Tls$ cd certs/
[12/24/20]seed@VM:~/.../certs$ ls -l
total 16
lrwxrwxrwx 1 seed seed 27 Dec 24 06:48 062cdee6.0 -> GlobalSign_Root_CA_-_R3
em
lrwxrwxrwx 1 seed seed 6 Dec 24 08:15 28d58a62.0 -> ca.crt
lrwxrwxrwx 1 seed seed 22 Dec 24 04:16 5ad8a5d6.0 -> GlobalSign_Root_CA.crt
-rw-r--r-- 1 seed seed 1415 Dec 24 08:14 ca.crt
-rw-r--r-- 1 root root 1261 Dec 24 04:08 GlobalSign_Root_CA.crt
-rw-r--r-- 1 root root 1261 Dec 24 04:02 GlobalSign_Root_CA.pem
-rw-r--r-- 1 seed seed 1229 Dec 24 06:46 GlobalSign_Root_CA_-_R3.pem
```

- Subsequently, modify the cadir='./certs' in the previous code to test and find that the results are consistent, so task1a is the same, the results can print out the corresponding information of the certificate, and the connection is successful.

Destination URL: www.jd.com

We originally planned to test www.alibaba.com, but since the verification certificate corresponding to the certificate of this website is the same as the .pem file of the www.baidu.com file, we changed a website with a different certificate to test and use Jingdong.

Direct test, because ./cert is not configured with the corresponding certificate, I get the error message:

```
[12/24/20]seed@VM:~/lab_Tls$ ./task1.py www.jd.com
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "./task1.py", line 25, in <module>
    ssock.do_handshake() # Start the handshake
  File "/usr/lib/python3.5/ssl.py", line 988, in do_handshake
    self._sslobj.do_handshake()
  File "/usr/lib/python3.5/ssl.py", line 633, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (ssl.c
:645)
```

At this point, modify cadir='./certs' in the code as the host storage certificate directory: cadir='/etc/ssl/certs' to get the corresponding certificate issuer information:

```
(('DNS', 'jd.com')),  
'version': 3}  
[{'issuer': (((('organizationalUnitName', 'GlobalSign Root CA - R3'),),  
                (('organizationName', 'GlobalSign'),),  
                (('commonName', 'GlobalSign'),))),  
  'notAfter': 'Mar 18 10:00:00 2029 GMT',  
  'notBefore': 'Mar 18 10:00:00 2009 GMT',  
  'serialNumber': '04000000000121585308A2',  
  'subject': (((('organizationalUnitName', 'GlobalSign Root CA - R3'),),  
                 (('organizationName', 'GlobalSign'),),  
                 (('commonName', 'GlobalSign'),))),  
  'version': 3}]  
After TLS handshake. Press any key to continue ...
```

https://blog.csdn.net/C_Ronaldo

Find the corresponding .pem file in the etc/ssl/certs directory, move it to the ./cert file, perform hash calculation, soft link processing, and then change back to cadir='./certs' to get the correct return result .connection succeeded.

Summary of problem-oriented thinking

Question 1:

According to the experimental requirements, we need to find the corresponding certificate file in the `/etc/ssl/certs` directory to verify the certificate returned by the server, but how to find the correct `.pem` file?

In response to this problem, the first thing that comes to mind is to look for the .pem field information in the server certificate printed by the code. However, the .pem format field did not appear in the certificate field, so I wondered: Is there any function or method in the ssl library that can print out the name of the .pem file that needs to be matched, so a search...Look at the ssl source code, Although I didn't understand much, I still didn't find anything usable. After spending a long time, forget it, and don't find it, I tried to find a few names and the subject field in the certificate in the etc/ssl/certs directory. I found several certificate files with the best match, but the GlobalSign_Root_CA.pem file looks the best match. I tried it, hey. . . Okay

- Maybe there is no such method. . . The certificate file name directly matches the hash value.
.. I split...

Question 2:

In the experiment manual, all the .crt files are manipulated, and the .pem files in the etc/ssl/certs directory are all .pem files, this...can it be successful? What is the difference between the two files?

According to the soft link, two files with the same file name and different suffixes were found. These two files were originally linked to each other. You can find them using the `ls -l` command in the `etc/ssl/certs` directory and move them to the experiment folder.

Use the `diff` command to compare and find that the file content is the same:

```
-----END CERTIFICATE-----  
[12/24/20]seed@VM:~/.../certs$ diff -r GlobalSign_Root_CA.pem GlobalSign_Root_CA  
.pem
```

Access to information: `.pem` is a file in a specific format encoded by BASE64. There are fixed strings at the beginning and end of the file, such as `-BEGIN-...` and `.crt` files are common in UNIX systems, and most of them are files in PEM format.

So the two are basically equal,

The hash value generated by using the two file names is also the same, and both can be verified by the certificate

...Bingo!

Task 1.c

Test the `www.sdu.edu.cn` website, and perform the following sub-case tests after configuring `/etc/hosts`:

False: No hostname verification is performed, and the wrong certificate can be verified successfully.

```
      ('DNS', 'www.jzb.sdu.edu.cn'),  
      ('DNS', 'www.zsdw.sdu.edu.cn'),  
      ('DNS', 'www.nc.sdu.edu.cn'),  
      ('DNS', 'www.icm.sdu.edu.cn'),  
      ('DNS', 'www.skicm.sdu.edu.cn'))),  
  'version': 3}  
[{'issuer': (((('countryName', 'US'),),  
               (('organizationName', 'DigiCert Inc'),),  
               (('organizationalUnitName', 'www.digicert.com'),),  
               (('commonName', 'DigiCert Global Root CA'),)),  
  'notAfter': 'Nov 10 00:00:00 2031 GMT',  
  'notBefore': 'Nov 10 00:00:00 2006 GMT',  
  'serialNumber': '083BE056904246B1A1756AC95991C74A',  
  'subject': (((('countryName', 'US'),),  
                (('organizationName', 'DigiCert Inc'),),  
                (('organizationalUnitName', 'www.digicert.com'),),  
                (('commonName', 'DigiCert Global Root CA'),)),  
  'version': 3}]  
After handshake. Press any key to continue ...
```

https://blog.csdn.net/C_Ronaldo_

True: Perform host name verification, and the wrong certificate cannot be verified successfully.

```
[12/24/20]seed@VM:~/lab_Tls$ ./handshake.py www.sdu2020.edu.cn
After making TCP connection. Press any key to continue ...
Traceback (most recent call last):
  File "./handshake.py", line 20, in <module>
    ssock.do_handshake() # Start the handshake
  File "/usr/lib/python3.5/ssl.py", line 988, in do_handshake
    self._sslobj.do_handshake()
  File "/usr/lib/python3.5/ssl.py", line 638, in do_handshake
    match_hostname(self.getpeercert(), self.server_hostname)
  File "/usr/lib/python3.5/ssl.py", line 297, in match_hostname
    % (hostname, ' '.join(map(repr, dnsnames))))
ssl.CertificateError: hostname 'www.sdu2020.edu.cn' doesn't match either of 'www.sdu.edu.cn', 'www.en.sdu.edu.cn', 'www.vie
w.sdu.edu.cn', 'www.sdrj.sdu.edu.cn', 'www.media.sdu.edu.cn', 'www.culture.sdu.edu.cn', 'www.tzgh.sdu.edu.cn', 'www.xxgk.sdu
u.edu.cn', 'www.qdxq.sdu.edu.cn', 'www.glyxb.sdu.edu.cn', 'www.bkix.sdu.edu.cn', 'www.grad.sdu.edu.cn', 'www.istudy.sdu.edu
.cn', 'www.bkzs.sdu.edu.cn', 'www.yz.sdu.edu.cn', 'www.job.sdu.edu.cn', 'www.swpx.sdu.edu.cn', 'www.rsrc.sdu.edu.cn', 'www.
rsrczp.sdu.edu.cn', 'www.ipos.sdu.edu.cn', 'www.fwsd.sdu.edu.cn', 'www.dcd.sdu.edu.cn', 'www.lib.sdu.edu.cn', 'www.ygb.sdu.e
du.cn', 'www.youth.sdu.edu.cn', 'www.xljk.sdu.edu.cn', 'www.rvsk.sdu.edu.cn', 'www.zzb.sdu.edu.cn', 'www.rd.sdu.edu.cn', 'w
ww.sc.sdu.edu.cn', 'www.cs.sdu.edu.cn', 'www.flc.sdu.edu.cn', 'www.art.sdu.edu.cn', 'www.chem.sdu.edu.cn', 'www.phy.sdu.edu
.cn', 'www.sps.sdu.edu.cn', 'www.glxy.sdu.edu.cn', 'www.econ.sdu.edu.cn', 'www.hj.sdu.edu.cn', 'www.cie.sdu.edu.cn', 'www.m
ath.sdu.edu.cn', 'www.lit.sdu.edu.cn', 'www.history.sdu.edu.cn', 'www.jc.sdu.edu.cn', 'www.sph.sdu.edu.cn', 'www.dent.sdu.e
du.cn', 'www.pharm.sdu.edu.cn', 'www.bms.sdu.edu.cn', 'www.medicine.sdu.edu.cn', 'www.nursing.sdu.edu.cn', 'www.mech.sdu.ed
u.cn', 'www.ee.sdu.edu.cn', 'www.tjst.sdu.edu.cn', 'www.epe.sdu.edu.cn', 'www.qtrns.sdu.edu.cn', 'www.tyb.sdu.edu.cn', 'w
ww.law.sdu.edu.cn', 'www.rgdyjy.sdu.edu.cn', 'www.huanke.sdu.edu.cn', 'www.lhp.sdu.edu.cn', 'www.imst.sdu.edu.cn', 'www.jg
dw.sdu.edu.cn', 'www.jjsh.sdu.edu.cn', 'www.sdtz.sdu.edu.cn', 'www.dbxb.sdu.edu.cn', 'www.dcx.sdu.edu.cn', 'www.dangxiao.s
du.edu.cn', 'www.xcb.sdu.edu.cn', 'www.stu.sdu.edu.cn', 'www.jgb.sdu.edu.cn', 'www.gonghui.sdu.edu.cn', 'www.women.sdu.edu
.cn', 'www.tsxt.sdu.edu.cn', 'www.ttc.sdu.edu.cn', 'www.sp.sdu.edu.cn', 'www.jmrh.sdu.edu.cn', 'www.cwc.sdu.edu.cn', 'www.sh
j.sdu.edu.cn', 'www.zcb.sdu.edu.cn', 'www.gac.sdu.edu.cn', 'www.jj.sdu.edu.cn', 'www.hqc.sdu.edu.cn', 'www.ltxc.sdu.edu.cn',
'www.journal.sdu.edu.cn', 'www.xlzx.sdu.edu.cn', 'www.cbs.sdu.edu.cn', 'www.xyy.sdu.edu.cn', 'www.archives.sdu.edu.cn', '
support.qd.sdu.edu.cn', 'www.onlineqd.sdu.edu.cn', 'www.onlineglyx.sdu.edu.cn', 'www.shcm.sdu.edu.cn', 'www.online.sdu.edu
.cn', 'www.wlly.sdu.edu.cn', 'www.ise.sdu.edu.cn', 'www.jzb.sdu.edu.cn', 'www.zsdw.sdu.edu.cn', 'www.nc.sdu.edu.cn', 'www.ic
m.sdu.edu.cn', 'www.sklcm.sdu.edu.cn'
[12/24/20]seed@VM:~/lab_Tls$
```

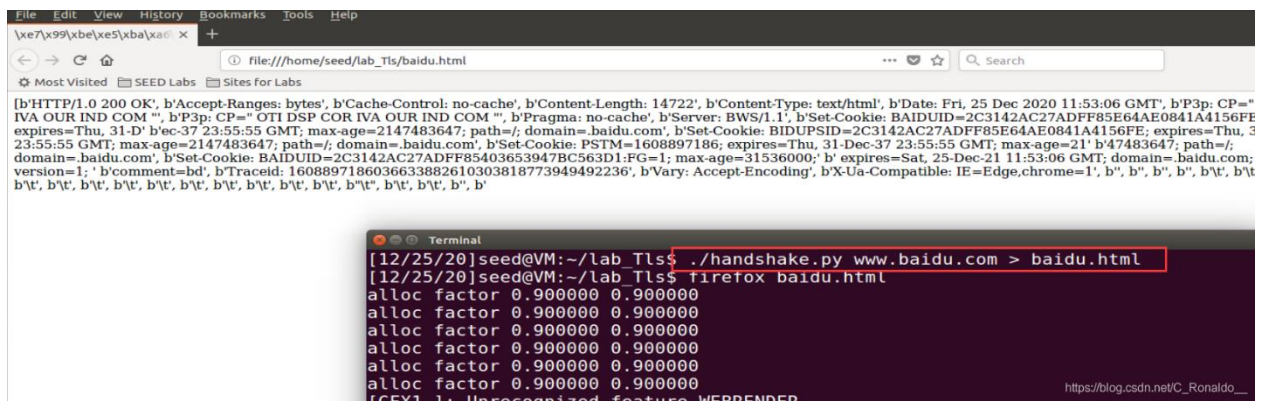
主机名与证书可以支持的域名不相同,所以出错

Question:

If hostname verification is not performed, then the attacker can use the legitimate certificate of his forged website and send it back to the user for verification. Since the hostname verification is ignored, the user will not notice it. Achieve the website's purpose of deceiving users.

task 1.d: Sending and getting Data

(1). After adding the manual code to the previous code, run it and get the following results:



The screenshot shows a web browser window displaying the raw HTTP response from baidu.com. The response includes headers like 'HTTP/1.0 200 OK', 'Content-Type: text/html', and various cookies. Below the browser window, a terminal window shows the execution of the script: `./handshake.py www.baidu.com > baidu.html`. The terminal output shows the script successfully establishing a connection and saving the response to a file named `baidu.html`.

Return the response data to the www.baidu.com website to the baidu.html file and open it with a browser.

(2). Modify the HTTP request line code to make the program return an image file data,...

Will not modify the http request,

I have used it before learning crawlers,

Tried to modify,

But always reporting errors...

I'll make up later...

Make up later...

Make up...

repair...

Task2: TLS Server

task 2.a: Implement a simple TLS server

This experiment needs to use some files generated in the seedlab Crypto_PKI experiment. In that experiment, we set CN to be the server certificate of SEEDPKILab.com and related files. In this experiment, we will use these files as the certificate of the server program. And send it back to the client program.

Modify the server.py code in the experimental manual as follows:

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

```
import socket, ssl, pprint
```

```
html = """
```

```
HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n
```

```
<!DOCTYPE html><html><body><h1>This is SEEDPKILab.com!</h1></body></html>
```

```
"""
```

```
SERVER_CERT = '../lab_pki/server.crt'#seedlab Crypto_PKI The certificate file of  
SEEDPKILab.com obtained in the experimentSERVER_PRIVATE =  
 '../lab_pki/server.key'#seedlab Crypto_PKIThe private key file of SEEDPKILab.com obtained in  
the experiment
```

```
# context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER) # For Ubuntu 20.04 VM
```

```
context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)    # For Ubuntu 16.04 VM
```

```
context.load_cert_chain(SERVER_CERT, SERVER_PRIVATE)
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM, 0)
```

```
sock.bind(('0.0.0.0', 4433))
```

```
sock.listen(5)
```

```
while True:
```

```
    newsock, fromaddr = sock.accept()
```

```
    try :
```

```
        ssock = context.wrap_socket(newsock, server_side=True)
```

```
        print("TLS connection established")
```

```
        data = ssock.recv(1024)          # Read data over TLS
```

```
        pprint.pprint("Request: {}".format(data))
```

```
        ssock.sendall(html.encode('utf-8')) # Send data over TLS
```

```
        ssock.shutdown(socket.SHUT_RDWR)  # Close the TLS connection
```

```
        ssock.close()
```

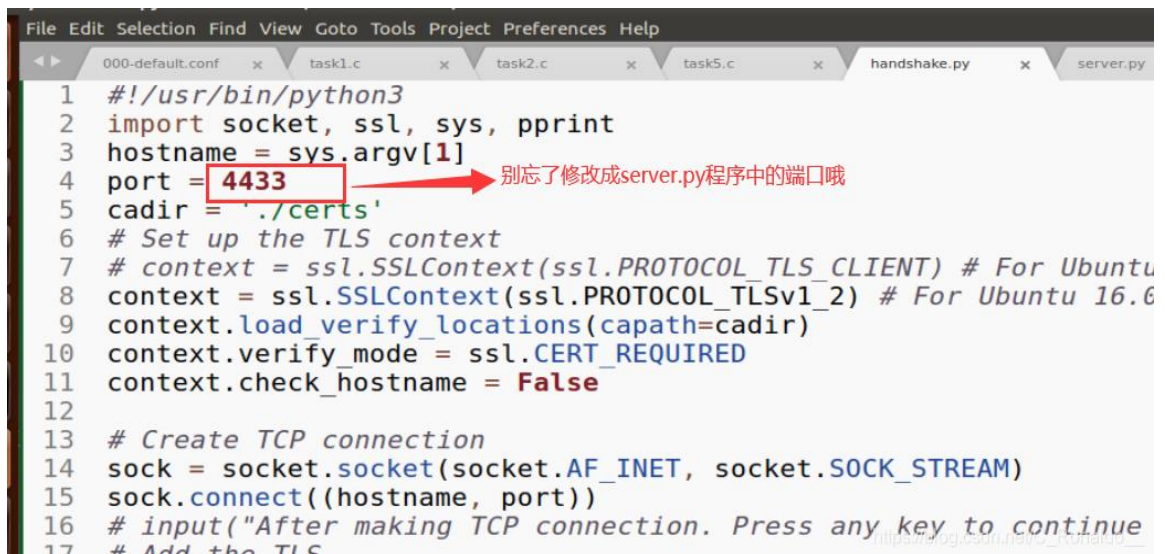
except Exception:

```
print("TLS connection fails")
```

```
continue
```

There are a few things to note:

- In the server.py code, we use `sock.bind(('0.0.0.0', 4433))` to bind the listening port 4433, so we need to make some modifications based on the client code in task1, that is, modify the port number (Otherwise the client program and server program will not run successfully):



```
File Edit Selection Find View Goto Tools Project Preferences Help
000-default.conf x task1.c x task2.c x task5.c x handshake.py x server.py
1  #!/usr/bin/python3
2  import socket, ssl, sys, pprint
3  hostname = sys.argv[1]
4  port = 4433
5  cadir = './certs'
6  # Set up the TLS context
7  # context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT) # For Ubuntu
8  context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2) # For Ubuntu 16.6
9  context.load_verify_locations(capath=cadir)
10 context.verify_mode = ssl.CERT_REQUIRED
11 context.check_hostname = False
12
13 # Create TCP connection
14 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
15 sock.connect((hostname, port))
16 # input("After making TCP connection. Press any key to continue")
17 # Add the TLS
```

- We need to use SEEDPKILab.com for the host name parameter when running the code on the client, so we need to configure the static IP in the local `/etc/hosts` file and add the `127.0.0.1 SEEDPKILab.com` entry so that the client will access it locally. .
- In the server.py program, the `sock.bind(('0.0.0.0', 4433))` code sentence, the IP address of the binding code is 0.0.0.0, which means that for any IP address on the machine, server.py will accept , And then reply to the data, this is very important: it corresponds to the role of the above configuration IP, so that the server and the client program can connect.

- Two Terminals need to be opened, one to run the server.py program and the other to run the client program. The client gets the following results:

```
socket.gaierror: [Errno 2] Name or service not known
[12/24/20]seed@VM:~/lab_Tls$ ./handshake.py SEEDPKILab2020.com
[b'\nHTTP/1.1 200 OK',
b'Content-Type: text/html',
b'',
b'\n<!DOCTYPE html><html><body><h1>This is SEEDPKILab.com!</h1></body></html>\n']
[12/24/20]seed@VM:~/lab_Tls$
```

Linux statements used in the experiment

In the process of debugging the server.py program, we may encounter the following situations:

```
[12/25/20]seed@VM:~/lab_Tls$ ./server.py
Enter PEM pass phrase:
Traceback (most recent call last):
  File "./server.py", line 19, in <module>
    sock.bind(('0.0.0.0', 4433))
OSError: [Errno 98] Address already in use
[12/25/20]seed@VM:~/lab_Tls$
```

This is a port occupancy problem. Although the server.py program is closed at this time, 0.0.0.0:4433 is still in the listening state. You can use the following statement to close it for subsequent debugging.

netstat -tunlp

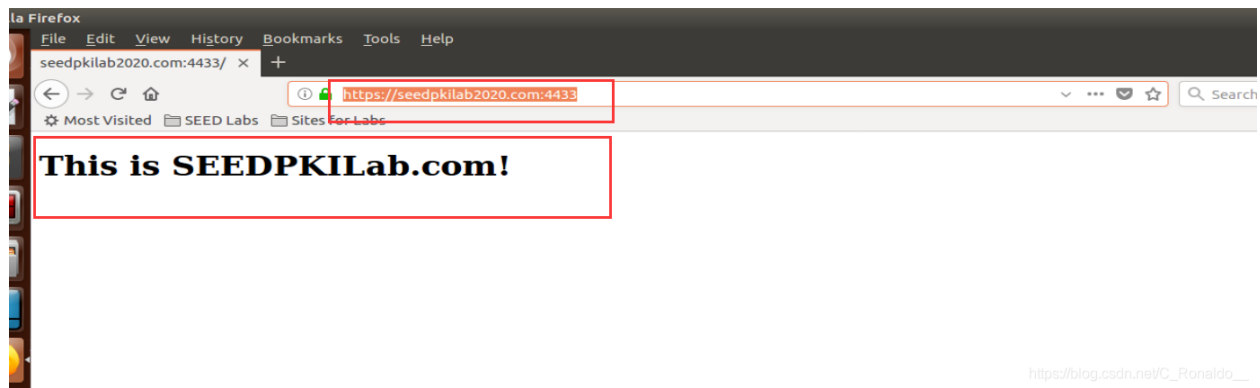
This is a combined command, -t, -u, -n, -l, -p. It can print out linux network status information, tcp, udp port, IP address and other information, check the 0.0.0.0 in Local Address: 4433 status, if it is running, after getting the corresponding PID, use the following command to kill:

kill -9 PID (fill in the PID obtained above)

-9 means forcibly kill the background process.

Task 2.b. Testing the server program using browsers

Since in the Crypto_PKI experiment, the root CA certificate created by yourself has been installed in the browser, you can directly test it. After running the server.py program, you will get the following results in the browser:



Task 2.c. Certificate with multiple names

Configure the server.openssl.cnf file as required, and then use the root CA file ca.crt to sign the certificate with multiple host names according to the experimental manual command, output the server_alt.crt file and the server_alt.key file, and then use these two The file is added to the certification path in the server.py program for testing.

One thing to note:

For the req_distinguished_name field information configuration in the server.openssl.cnf file configuration, it is intuitively felt that you can specify it at will, as long as you add the hostname you need in the CN field at the end, but the result is always full of surprises. After randomly filling in an ST field information, the following error message will appear in the command to generate a certificate file:

```
[12/24/20]seed@VM:~/lab_pki$ openssl ca -md sha256 -days 3650 -config ./myopenssl.cnf -batch -in server_alt.csr -out server_alt.crt -cert ca.crt -keyfile ca.key
Using configuration from ./myopenssl.cnf
Enter pass phrase for ca.key:
Check that the request matches the signature
Signature ok
The stateOrProvinceName field needed to be the same in the
CA certificate (shandong) and the request (Qingdao)
```

??WHY

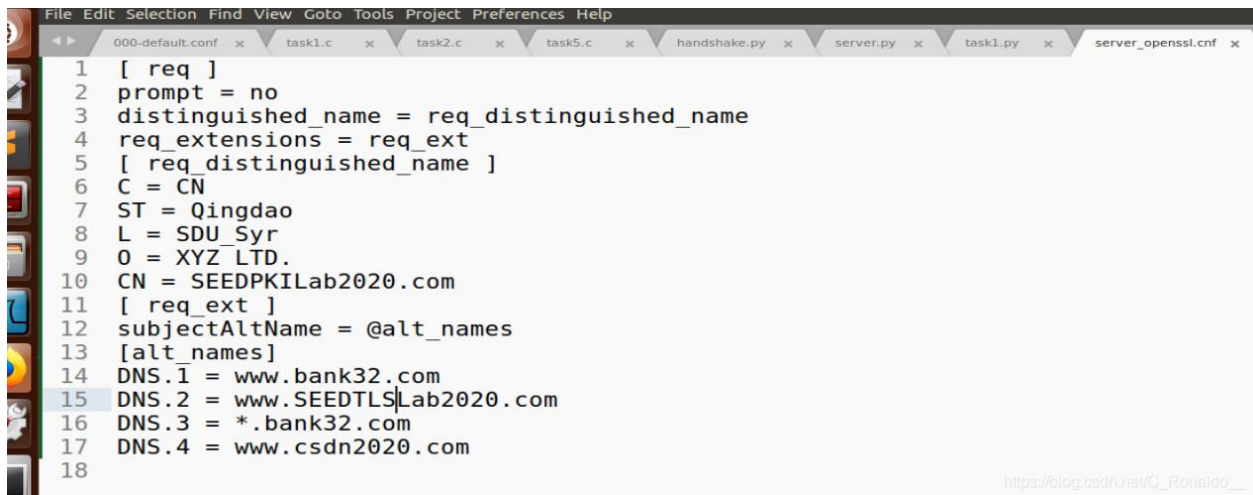
https://blog.csdn.net/C_Ronaldo_

According to the prompt, it needs to be the same as the ST field information in the CA file, set to shandong, why?

Google it and saw a blog written. For a social-oriented CA issuing organization, the country and province in front of the certificate issued by it can be set at will, but when issuing a certificate with

a private CA, it needs to be filled in with the server's certificate. The same, otherwise the signing is unsuccessful, and the reason is not said, remember...

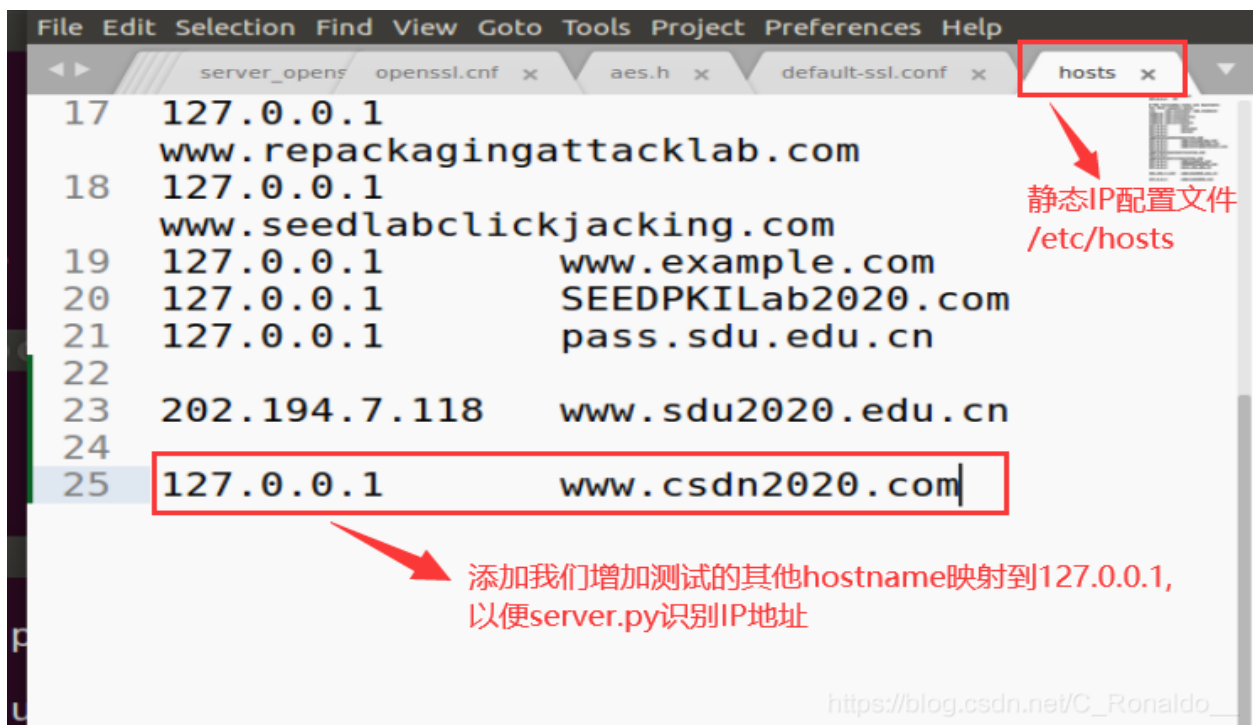
Finally, configure server.openssl.cnf as follows:



```
1 [ req ]
2 prompt = no
3 distinguished_name = req_distinguished_name
4 req_extensions = req_ext
5 [ req_distinguished_name ]
6 C = CN
7 ST = Qingdao
8 L = SDU_Syr
9 O = XYZ LTD.
10 CN = SEEDPKILab2020.com
11 [ req_ext ]
12 subjectAltName = @alt_names
13 [alt_names]
14 DNS.1 = www.bank32.com
15 DNS.2 = www.SEEDTLSLab2020.com
16 DNS.3 = *.bank32.com
17 DNS.4 = www.csdn2020.com
18
```

https://blog.csdn.net/C_Ronaldo_

Run the server.py program in one Terminal, and run the client program in task1 in the other Terminal, and use the host name www.csdn2020.com in the alt_name field to access (because it is tested on this machine, it is necessary to configure a static IP in the etc/hosts file , Add 127.0.0.1 www.csdn2020.com entry:



```
17 127.0.0.1
   www.repackagingattacklab.com
18 127.0.0.1
   www.seedlabclickjacking.com
19 127.0.0.1      www.example.com
20 127.0.0.1      SEEDPKILab2020.com
21 127.0.0.1      pass.sdu.edu.cn
22
23 202.194.7.118  www.sdu2020.edu.cn
24
25 127.0.0.1      www.csdn2020.com
```

静态IP配置文件 /etc/hosts

添加我们增加测试的其他hostname映射到127.0.0.1, 以便server.py识别IP地址

https://blog.csdn.net/C_Ronaldo_

After testing), the following results are obtained:

```
[12/24/20]seed@VM:~/lab_Tls$ ./handshake.py www.csdn2020.com
[b'\nHTTP/1.1 200 OK',
b'Content-type: text/html',
b'',
b'\n<!DOCTYPE html><html><body><h1>This is SEEDPKILab.com!</h1></body></html>\n']
[12/24/20]seed@VM:~/lab_Tls$
```

https://blog.csdn.net/C_Ronaldo__

It can also be tested in the browser. Since we have imported the private ca.crt certificate in the browser, and the server certificate we generated is issued by this ca.crt, it will pass the verification:



Of course, we can also test on other hosts, just need to configure a static IP in the etc/hosts file to make an IP modification.

Task3. A Simple HTTPS Proxy

Encountered a technical problem...

The part of the code that needs to be supplemented is reported

I have too little knowledge of python network ssl programming

Too busy recently

I have to review at the end of the term

In the future, I have time to learn ssl programming systematically

Let's add this part of the experimental report.