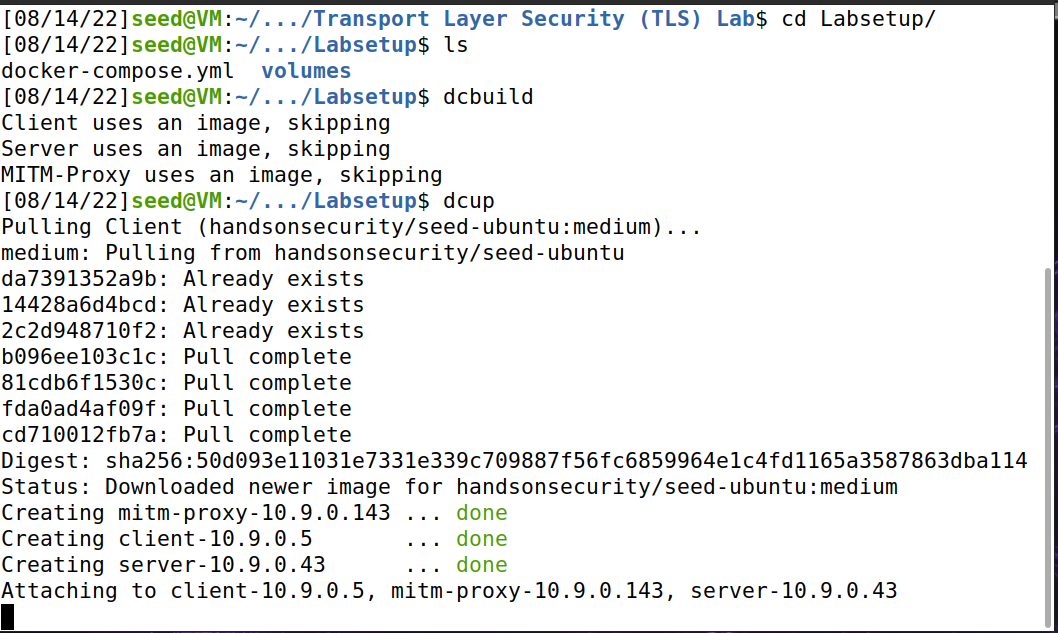
Lab3 TLS

实验环境

1. 进入实验目录，/home/seed/Desktop/Labs\_20.04/Cryptography/Transport Layer Security (TLS) Lab/Labsetup。
2. dcup命令开启容器。



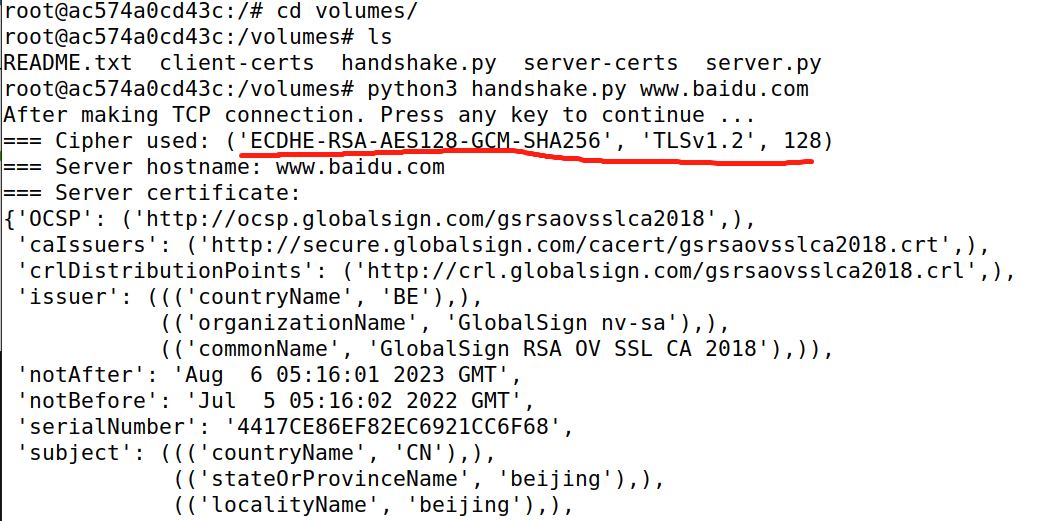
Dcup的过程中，pull镜像需要网络下载，网络有问题可能导致出错。

本次实验主要是编程实现TLS的客户端和服务器端程序，分别在客户端容器，服务器端容器完成。

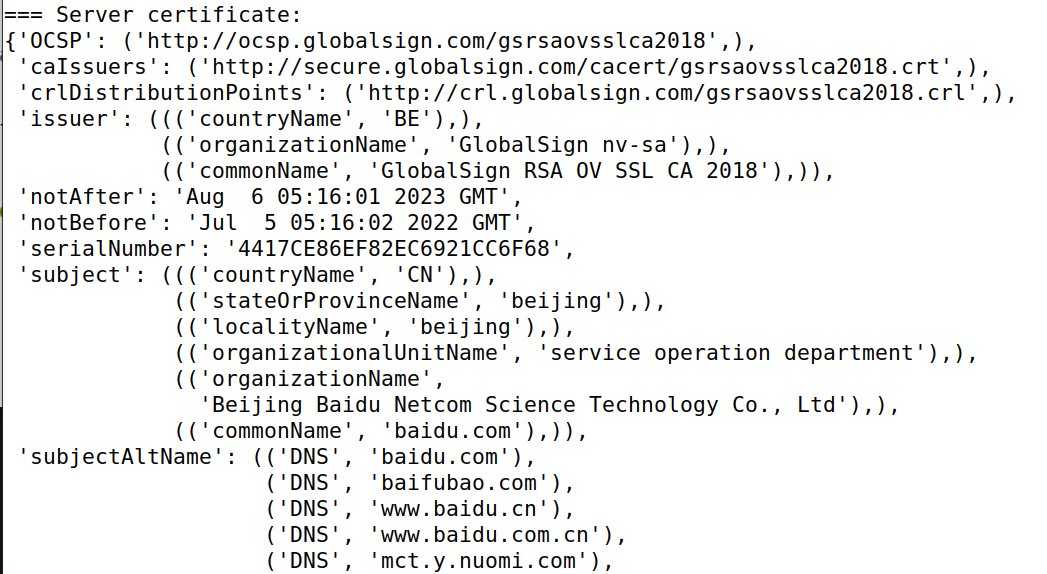
Task 1: TLS Client

Task 1A TLS handshake

**在client对应的容器中，运行handshake程序：**



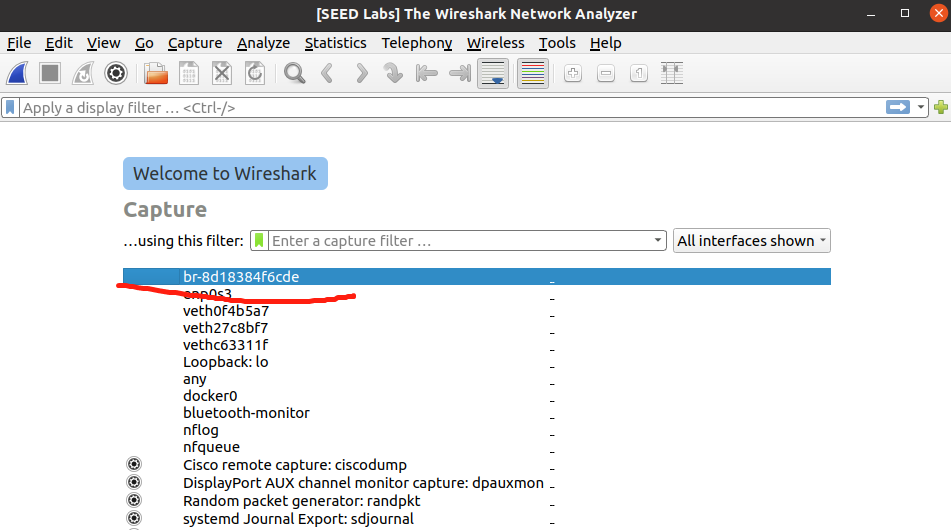
从上面的输出中可以看出server和client使用的加密算法。



输出中 Server certificate部分就是server的证书。

/etc/ssl/certs目录放置的是所有的信任CA证书，就类似于上个实验中浏览器中的CA证书一样。

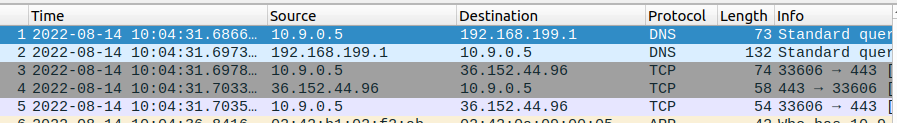
**开启wireshark查看握手过程**



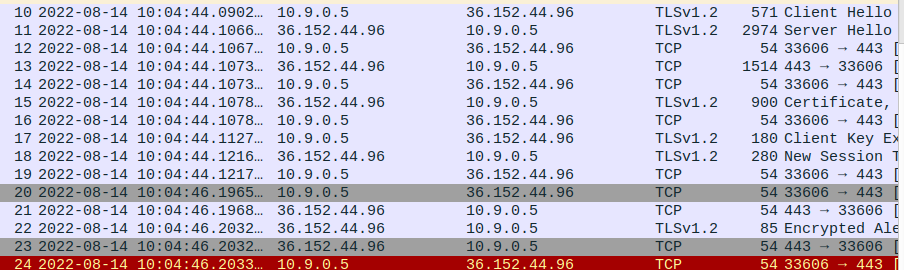
**选择抓取br…设备报文，然后重新运行Client容器的程序**

**可以看到wireshark的抓包情况。**

从下图中可以看到，在程序运行之后先建立了TCP连接，从3-5这三个包是用于建立TCP连接的，而这时候还没有TLS连接。



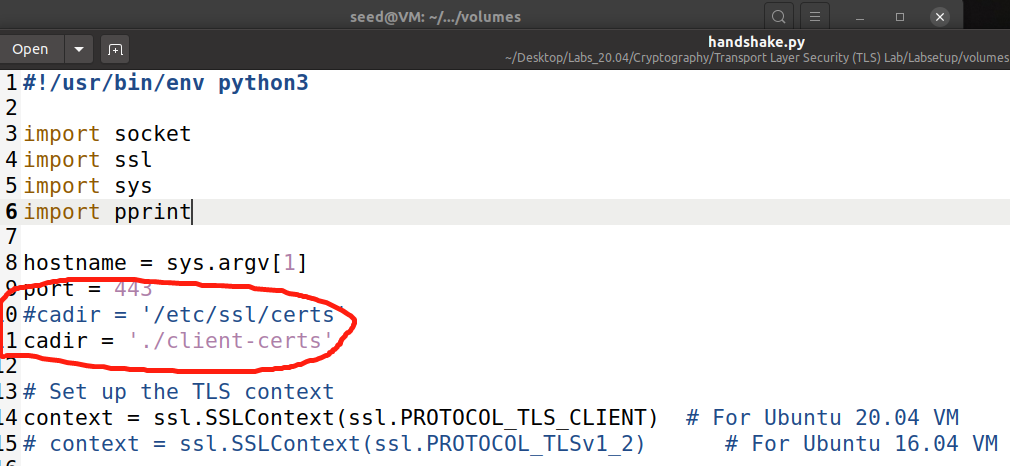
按一下任意键之后开始建立TLS连接。从第6个包开始建立TLS连接，到第15个包结束。



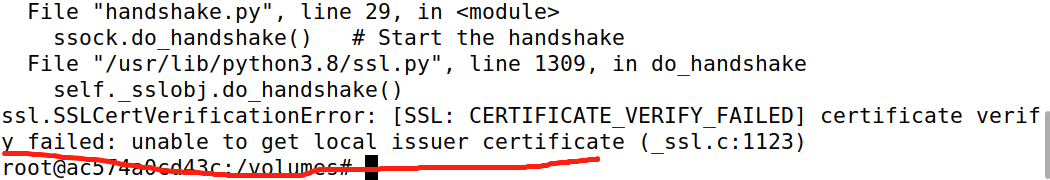
Task 1B CA’s Certificate

1. 这个实验我们建立自己的CA证书文件夹，来代替上面的/etc/ssl/certs文件夹。将handshake.py中的cadir修改一下，如下所示，

cadir=‘./client-certs’:

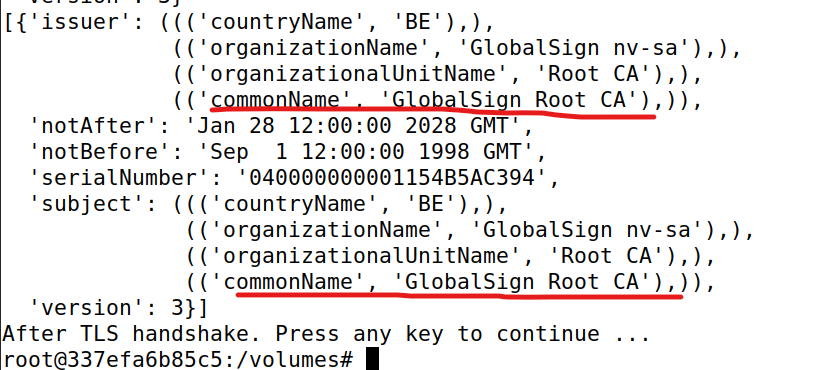


1. 再次运行handshake.py www.baidu.com然后观察结果，结果如下:

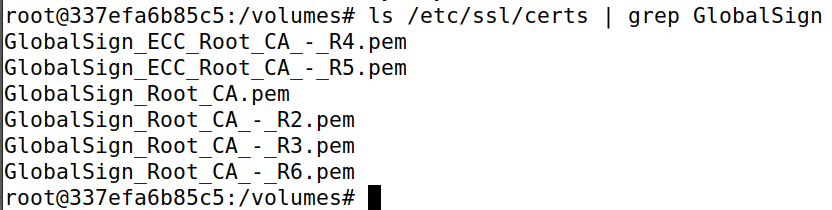


发现在建立TLS连接的时候出错了，无法使用本地的CA证书进行验证。

1. 我们需要把需要的CA证书放进client-certs文件夹中，需要哪个CA证书可以从最初执行handshake.py文件时的输出中获取，如下图所示:



找到划红线的部分就是根CA证书，然后再/etc/ssl/certs目录中找到对应的CA文件:



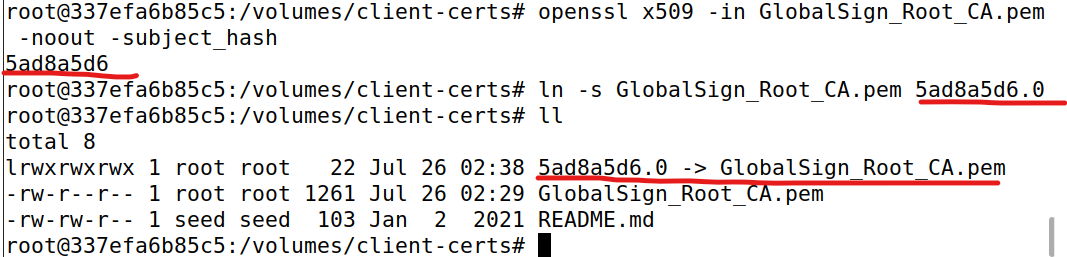
把GlobalSign\_Root\_CA.pem文件拷贝到clien-certs目录下

然后使用如下的openssl命令获取一个hash值

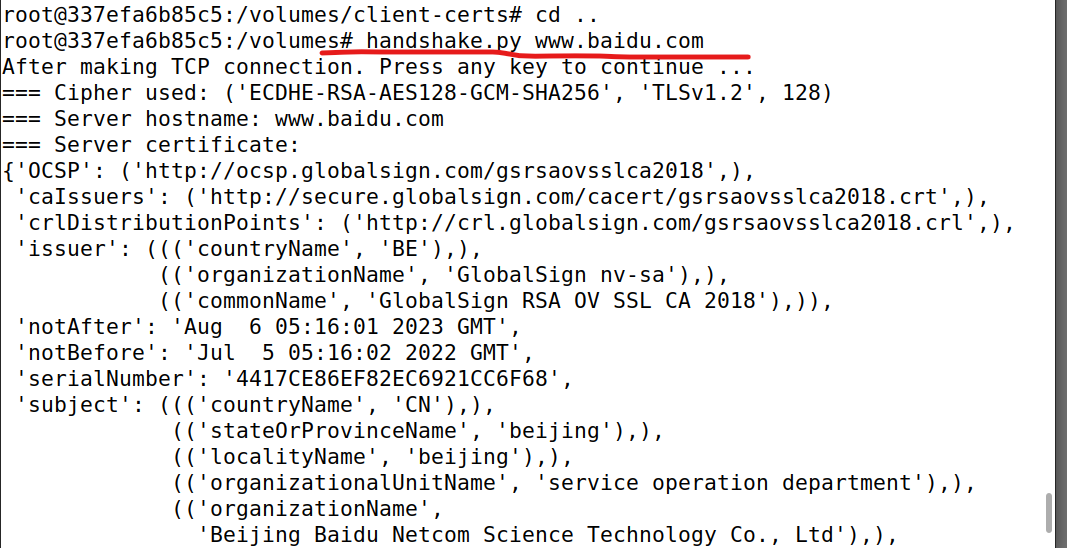
openssl x509 -in GlobalSign\_Root\_CA.pem -noout -subject\_hash

并创建一个名为该hash值的软链接。

结果如图所示:



最后再回到volumes目录中执行handshake.py [www.baidu.com](http://www.baidu.com)即可看到结果：

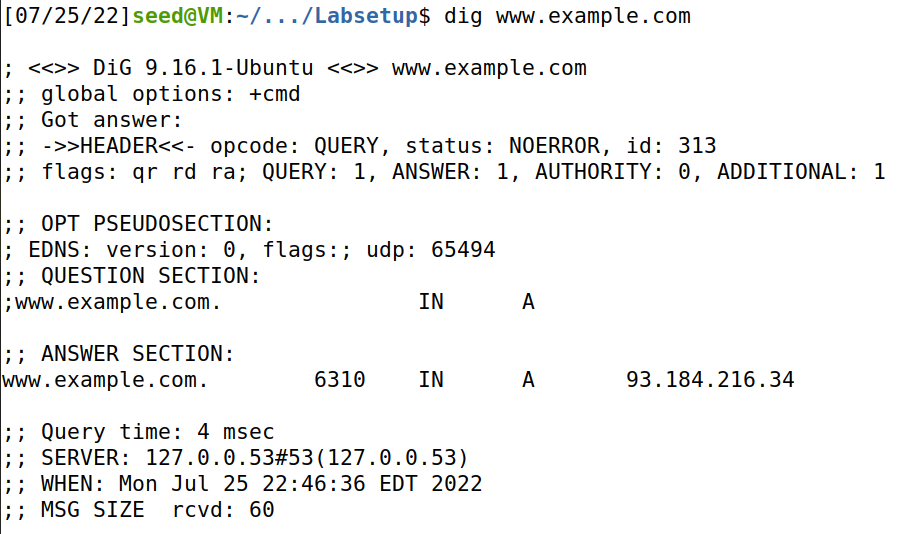


1. 请仿照上面的三步继续使用另外两个网站重复实验。

记得第一步运行前需要先修改handshake.py文件中的certs目录为/etc/ssl/certs目录。

Task 1C: Experiment with the hostname check

1. 在VM主机上执行dig命令获取[www.example.com的IP](http://www.example.com的IP)地址，如下所示:



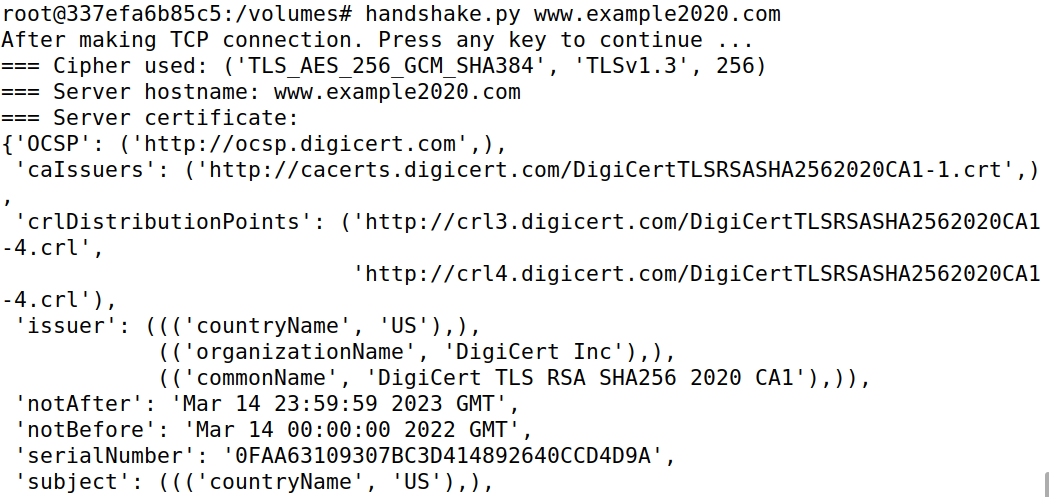
1. 修改容器的/etc/hosts文件，添加另一个域名



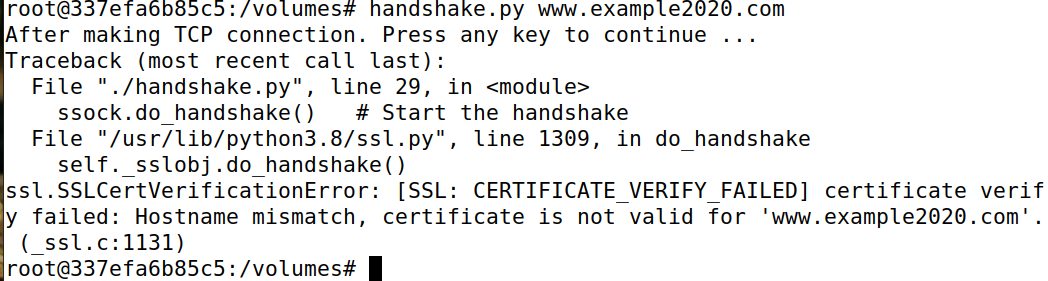
1. 将handshake.py中的context.check\_hostname选项分别设置为True和False，然后运行handshake.py [www.example2020.com](http://www.example2020.com)命令，然后查看结果如何。

**这里记得先把handshake.py中的cacerts目录修改为/etc/ssl/certs**

(1). 将其设置为True 之后的结果(结果正常)



(2). 将其设置为False的结果

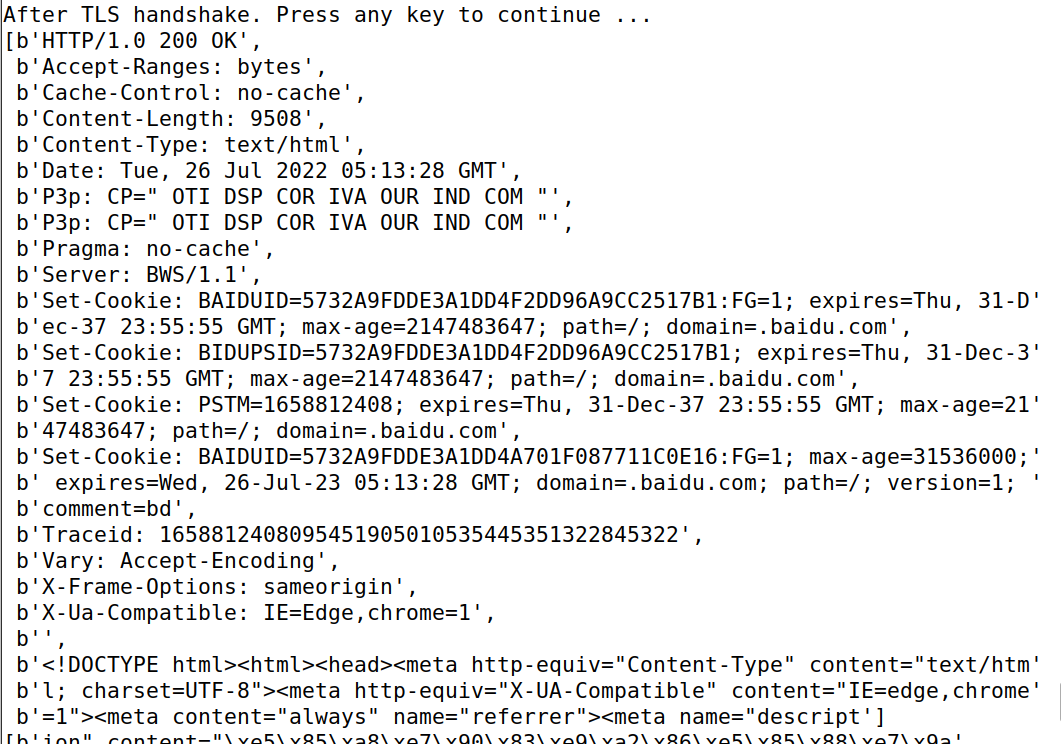


Task 1D: Sending and getting Data

发送HTTP请求，并接收数据。

1. 修改handshake.py文件内容，代码如下:
2. **import** socket
3. **import** ssl
4. **import** sys
5. **import** pprint
7. hostname = sys.argv[1]
8. port = 443
9. cadir = '/etc/ssl/certs'
10. #cadir = './client-certs'
12. # Set up the TLS context
13. context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_CLIENT)  # For Ubuntu 20.04 VM
14. # context = ssl.SSLContext(ssl.PROTOCOL\_TLSv1\_2)      # For Ubuntu 16.04 VM
16. context.load\_verify\_locations(capath=cadir)
17. context.verify\_mode = ssl.CERT\_REQUIRED
18. context.check\_hostname = True
20. # Create TCP connection
21. sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)
22. sock.connect((hostname, port))
23. input("After making TCP connection. Press any key to continue ...")
25. # Add the TLS
26. ssock = context.wrap\_socket(sock, server\_hostname=hostname,
27. do\_handshake\_on\_connect=False)
28. ssock.do\_handshake()   # Start the handshake
29. **print**("=== Cipher used: {}".format(ssock.cipher()))
30. **print**("=== Server hostname: {}".format(ssock.server\_hostname))
31. **print**("=== Server certificate:")
32. pprint.pprint(ssock.getpeercert())
33. pprint.pprint(context.get\_ca\_certs())
34. input("After TLS handshake. Press any key to continue ...")
35. # Send HTTP Request to Server
36. request = b"GET / HTTP/1.0\r\nHost: " + hostname.encode("utf-8") + b"\r\n\r\n"
37. ssock.sendall(request)
38. # Read HTTP Response from Server
39. response = ssock.recv(2048)
40. **while** response:
41. pprint.pprint(response.split(b"\r\n"))
42. response = ssock.recv(2048)
44. # Close the TLS Connection
45. ssock.shutdown(socket.SHUT\_RDWR)
46. ssock.close()

结果中多出了[www.baidu.com](http://www.baidu.com)主页中内容，请求结果如下：



1. 获取图片

没有成功

Task2: TLS Server

Task2a: Implement a simple TLS server

1. 首先我们的域名是姓名首字母+2022的组合，比如张三，那么域名就是[www.zs2022.com](http://www.zs2022.com)。这个域名对应的服务会在Server容器也就是地址10.9.0.43中开启，所以需要将DNS记录添加到Client容器中，即以下记录。



1. 按照PKI实验的步骤创建CA证书即ca.crt和ca.key，然后为自己的域名创建对应的server.key和server.crt。

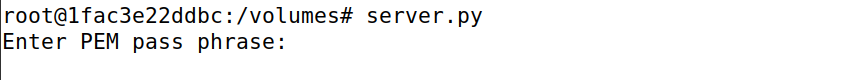
**可以直接使用PKI实验中的ca.crt和ca.key。**

**但是同一个CA为Server证书签名时的serial值(前面的例子是1000)不能相同(这个注意)。**

将生成的server.key和server.crt放在server-certs文件夹中。

Server.py中的server.key和server.crt位置和名称也要修改和自己生成的证书的名称和位置相同。

1. 在server容器启动服务，也就是运行server.py文件

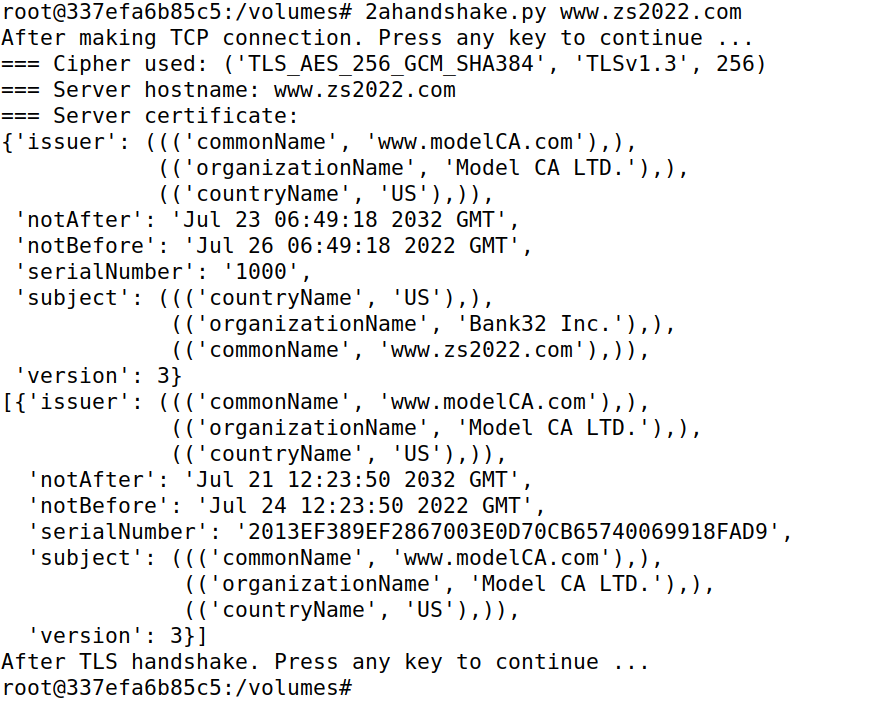


输入密码使输入dees。

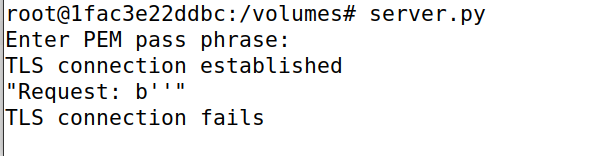
1. 将ca.crt放入client-certs中，并使用openssl命令获取hash之后在client-certs中建立软链接。

然后运行handshake.py文件，这里的handshake.py文件和最初的内容相同。

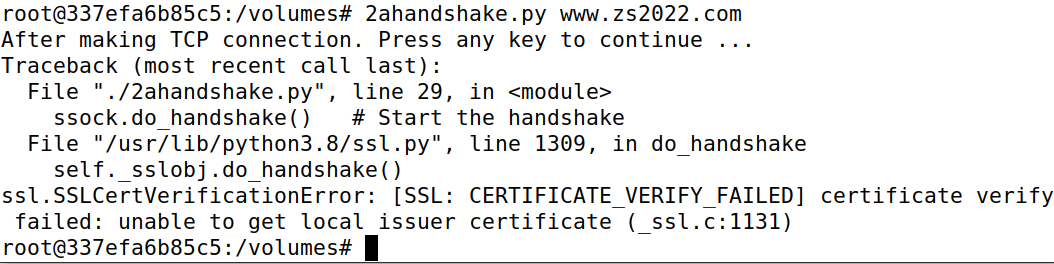
结果如下:



在Server容器中也有输出:



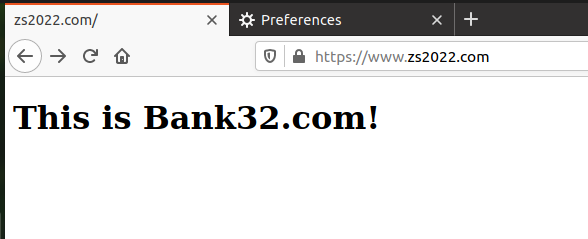
1. 然后将handshake.py中的certs目录修改为/etc/ssl/certs，再次运行测试，结果如下:



这里成功建立了TCP连接，但是无法建立TLS连接。

Task2b: Testing the server program using browsers

1. 在浏览器访问HTTPS需要将自己的CA证书加入浏览器的证书列表中，这在PKI实验中做过，这里也会同样的步骤。(如果已经加入过了，就不需要了)
2. 在浏览器访问你的域名，这里是[www.zs2022.com](http://www.zs2022.com)，结果如下：



Task 2c: Certificate with multiple names

该实验是在2b的基础上使多URL映射到同一个站点，这个其实在PKI实验中已经做过，但是当时使用的是命令行的方式添加参数生成CSR，这里是使用配置文件的方式生成CSR文件，初次之外没有其他区别。

1. 使用配置文件的方式添加别名，配置文件内容如下：
2. [ req ]
3. prompt = no
4. distinguished\_name = req\_distinguished\_name
5. req\_extensions = req\_ext
7. [ req\_distinguished\_name ]
8. C = US
9. ST = New York
10. L = Syracuse
11. O = XYZ LTD.
12. CN = www.zs2022.com
14. [ req\_ext ]
15. subjectAltName = @alt\_names
17. [alt\_names]
18. DNS.1 = www.zs2022.com
19. DNS.2 = www.zs2022.net
20. DNS.3 = \*.zs2022.com
21. 以配置文件的形式使用openssl req命令生成CSR文件

**openssl req -newkey rsa:2048 -config ./server\_openssl.cnf -batch \  
-sha256 -keyout server.key -out server.csr**

1. 之后使用openssl ca命令为证书签名。

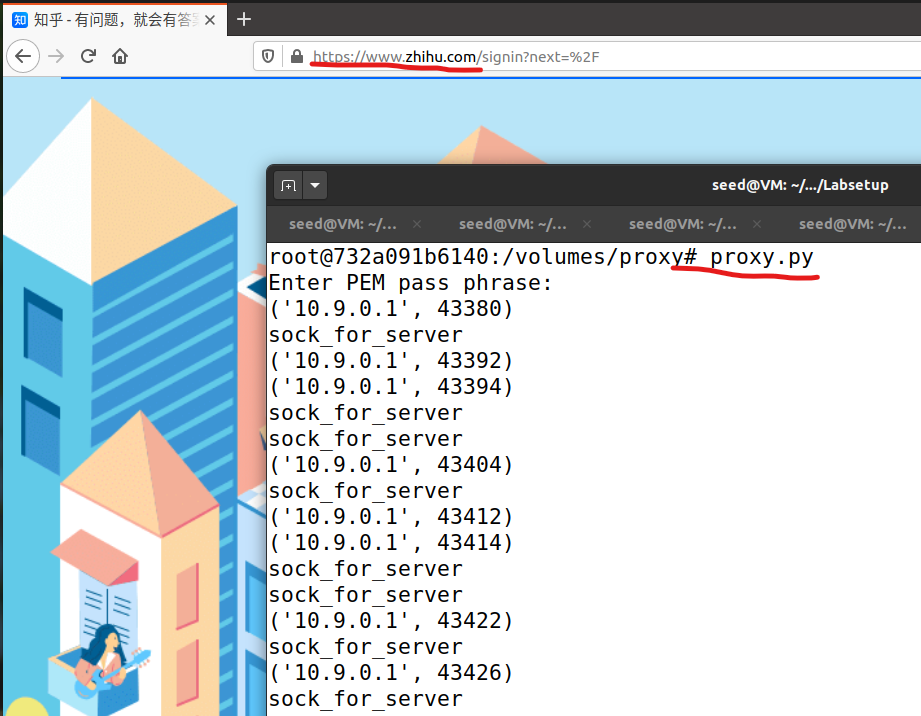
**openssl ca -md sha256 -days 3650 -config ./myopenssl.cnf -batch \  
-in server.csr -out server.crt \  
-cert ca.crt -keyfile ca.key**

1. 使用新生成的证书开启server服务。
2. 在Client容器访问你添加的别名，并截图。
3. 如果要在浏览器访问成功，需要吧别名也加入/etc/hosts中。

Task3: A Simple HTTPS Proxy

本实验使用proxy作为中间代理，既做Server又做Client，于真实的Server和Client进行交互。

1. 首选选择一个网站，例如[www.zhihu.com](http://www.zhihu.com)(可以选简单的，或者自己搭建简单的网站)
2. 使用CA对该网站签发证书(先CSR 然后是 CRT证书)。然后把证书zhihu.key和zhihu.crt(自己命名的)放在下面的proxy.py同目录下，以便proxy.py使用。
3. 在Proxy容器中编写以下程序:
4. #!/usr/bin/env python3
5. **import** threading
6. **import** ssl
7. **import** socket
9. cadir = "/etc/ssl/certs"
11. **def** process\_request(ssock\_for\_browser):
12. hostname = "www.zhihu.com"
13. # Make a connection to the real server
14. sock\_for\_server = socket.create\_connection((hostname, 443))
15. # Set up the TLS context
16. context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_CLIENT)
17. context.load\_verify\_locations(capath=cadir)
18. context.verify\_mode = ssl.CERT\_REQUIRED
19. context.check\_hostname = True
20. **print**("sock\_for\_server")
21. ssock\_for\_server = context.wrap\_socket(sock\_for\_server, server\_hostname=hostname, do\_handshake\_on\_connect=False)
22. ssock\_for\_server.do\_handshake()
24. request = ssock\_for\_browser.recv(2048)
25. **if** request:
26. # Forward request to server
27. ssock\_for\_server.sendall(request)
29. # Get response from server, and forward it to browser
30. response = ssock\_for\_server.recv(2048)
31. **while** response:
32. ssock\_for\_browser.sendall(response) # Forward to browser
33. response = ssock\_for\_server.recv(2048)
35. ssock\_for\_browser.shutdown(socket.SHUT\_RDWR)
36. ssock\_for\_browser.close()
38. SERVER\_CERT = "./zhihu.crt"
39. SERVER\_PRIVATE = "./zhihu.key"
40. context\_srv = ssl.SSLContext(ssl.PROTOCOL\_TLS\_SERVER)
41. context\_srv.load\_cert\_chain(SERVER\_CERT, SERVER\_PRIVATE)
42. sock\_listen = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM, 0)
43. sock\_listen.bind(("0.0.0.0", 443))
44. sock\_listen.listen(5)
46. **while** True:
47. sock\_for\_browser, fromaddr = sock\_listen.accept()
48. **print**(fromaddr)
49. ssock\_for\_browser = context\_srv.wrap\_socket(sock\_for\_browser, server\_side=True)
50. x = threading.Thread(target=process\_request, args=(ssock\_for\_browser,))
51. x.start()
52. 在VM虚拟机中将修改/etc/hosts,[www.zhihu.com](http://www.zhihu.com)的地址映射到10.9.0.143。
53. 在Proxy容器中修改/etc/resolv.cnf，将nameserver修改为8.8.8.8。
54. 在Proxy容器启动proxy.py之前在VM虚拟机的浏览器访问[www.zhihu.com](http://www.zhihu.com)，这时候应该是没有结果的。(Server Not Found)
55. 然后在Proxy容器中启动proxy.py。
56. 在宿主机容器的浏览器中输入<https://www.baidu.com然后看proxy>中是否有输出，有输出就表示经过了proxy程序。



可以看到浏览器不断于proxy建立连接。

1. 延伸(密码抓取)

这时候可以自己抓包对输入密码的包进行过滤，然后找到包含密码的包。

之后在网上查找如何对加密的数据进行解密(已经有私钥了)。

这样就可以做到使用proxy程序抓取密码的目的。

**完成实验后记得清理**

docker stop

docker rm

以免后续实验冲突。