Lab Assignment 6

(Due: **Nov 6/7/8/2**)

In this lab assignment, you will create a simple TLS-protected TCP server. It consists of two parts. In part I, you will create server's public-key and its certificate that is signed by an authority (created by you). In part II, you will modify a simple normal TCP server and make it TLS-protected. The submission requirements will be given at the end of this file.

Part I (create server public-key and its certificate)

In TLS, the server must have a RSA public key. To guarantee this public key is owned by this server, it must be certified by an authority. In reality, this is done by some special company such as VeriSign. In our experiment, we ourselves will play the role of an authority. This authority will generate its own public/private key and certificate (which is a self-signed signature), just as what a real root CA has done. It then will generate the certificate for the TLS server. The task can be done by following the following procedure.

1. Copy **/usr/lib/ssl/openssl.cnf** to your current working directory and make the following change to this file:

   "policy = policy_match" to "policy = policy_anything"

/*this allows the CA to generate certificate for more potential users. */

2. Create a new directory **demoCA** in the current directory. Then, do the following.

   - Create new directories **certs**, **crl** and **newcerts** in **demoCA** and empty files **index.txt** and **serial**. Put a single serial number (.e.g., 1000) in the file **serial**.

     (do the following steps outside demoCA)

   - Generate a self-signed certificate for our certificate authority (CA).
     $openssl req -new -x509 -keyout demo_ca.key -out demo_ca.crt -config openssl.cnf -days 365
     /* **demo_ca.key** has private RSA key for CA; **demo_ca.crt** is its self-signed certificate with 365 days validity. */
   - Create a certificate for our test TLS server, signed by our authority's key **demo_ca.key**.
     1. Generate a RSA private key for TLS server.
        $ openssl genrsa -aes128 -out Test.key 2048
        /*To view the file, **$openssl rsa -in Test.key -noout -text** */
     2. Generate a certificate signing request:
        $ openssl req -new -key Test.key -out Test.csr -config openssl.cnf
        /* this generates a certificate request so that CA can sign a cert for TLS server: **common name** uses the container name (e.g., **client1-10.9.0.5**). */
     3. Generate the certificate for TLS server:
        $ openssl ca -in Test.csr -out Test.crt -cert demo_ca.crt -keyfile demo_ca.key -config openssl.cnf
        /*Test.crt is the certificate for TLS server */

/*the following steps operated in **volumes** might need **sudo; volumes** is the folder created by your docker-compose (using the **newly** provided yml file) ***/**

4. Copy your certificate **Test.crt** and **Test.key** to a folder **certS** in the shared folder **volumes** (your server program such as **server.py** will send Test.crt to client and use Test.key to decrypt or sign a document).
5. Copy *demo_ca.crt* to folder (such as **certC**) in the shared folder **volumes**. Later your client program (such as client.py) will use this *demo_ca.crt* to verify if the server certificate Test.crt is indeed owned by your server VM. However, your client VM can not *directly* locate demo_ca.crt. You need to create a symbolic link that links to demo_ca.crt. Your client VM can use this symbolic link to find demo_ca.crt in certs.

   (a) openssl x509 -in demo_ca.crt -noout -subject_hash
   8f838d2e /*this is the hash value; your case might be different*/

   (b) ln -s demo_ca.crt 8f838d2e.0 /***warning**: do not forget **.0** */

Part II. (TLS Client and Server Communication)

In this part, you need to create a TLS server and TLS client with some functions. Toward this, you are encouraged to run the provided TLS server and client to get familiar with how TLS client and server can be connected.

- **Step 1**. Make sure that Part I has been done.
- **Step 2**. Use the provided **client.py** and **server.py**. Modify the certificate directory **cadir** in server.py to make sure it is the directory **certS** for the server certificate and server private key directory (in **Part I**). Also modify the certificate directory **cadir** in client.py to make sure that it is the directory **certC** of the CA's certificate demoCA.crt.
- **Step 3.** Run .$sudo python3 server.py on the server VM (should be the same as in **Test.crt**)
- **Step 4**. Run client.py with server container name (e.g., client1-10.0.2.5):

  $ python3 client.py client1-10.9.0.5

  Then, if you receive the response from server, then you are done

Task. You are required to modify the client.py and **server.py** satisfying the following.

1) Client can interactively communicate with the server. The client takes the input from user and send to server; when the server receives the client message, it reverses it and sends back (e.g., **hello** will become **olleh**). The client then prints to the screen and waits for the next user input.
2) Server should support multi-threads. That is, it can communicate with several clients.

Submission requirements:

(a) your client and server code.

(b) **two** clients and server screen shots that shows TLS establishment and messages.

(c) screen shot for the server certificate content

/*using **openssl x509 -in your_server_cert -text -noout** */