Assignment 2

Assigned: 1/11/2022
Due: 7/11/2022

In this assignment you are going to implement a secure server-client communication using OpenSSL in C. The purpose of this assignment is to provide you the opportunity to get familiar with the internals and implementations of client-server communication, SSL and OpenSSL.

Basic Theory

What is SSL?

An SSL (Secure Sockets Layer) is the standard security protocol used to establish an encrypted connection between a server and a client. After establishing the connection SSL/TLS ensures that the data transmitted between server and client are secured and intact.

SSL is designed to exchange sensitive data over the network using some secure algorithms and prevent another program that wants to access the private data from the network connection. SSL uses **asymmetric encryption algorithms** to secure the transmission of data. These algorithms use the pair of keys (**public and private**). The **public key** is **freely available** and known for anybody. The **private key** is only known by the **server or the client**.In SSL data encrypted by the **public key can only decrypt by the private key** and the data encrypted by the **private key can only decrypt by the public key**.

In the SSL communication, the client starts the connection from the first hello (SSL) message. This hello message starts the negotiation and performs the handshaking between server and client. After completing the handshaking if everything is fine then generate a secured key for the current connection. The server and client have used this secret key in data exchanging.

SSL handshake flow

The SSL handshake is an authentication process. In which server and client authenticate to each other using a certificate. This certificate is generated by the user himself with the help of **OpenSSL commands.**

Steps of Handshaking between client and server:

1. In the beginning of the communication, SSL/TLS client sends a “client_hello” message to the server. This message contains all the cryptographic information which is supported by the client, like the highest protocol version of SSL/TLS, encryption algorithm lists, data compression method, resume session identifier, and randomly generated data (which will be used in symmetric key generation).
2. The SSL/TLS server responds with a “server_hello” message to provide all the necessary information to establish a connection like protocol version used, data compression algorithms and encryption method selected, assigned session id and random data (which will be used in symmetric key generation).
3. The server sends a certificate to the client and also inserts a request message for the client certificate because the server requires the client certificate for the mutual authentication.
4. The SSL or TLS client verifies the server’s digital certificate.
5. If the SSL or TLS server sent a “client certificate request”, the client sends a random byte string encrypted with the client’s private key, together with the client’s digital certificate, or a “no digital certificate alert”. This alert is only a warning, but with some implementations, the handshake fails if client authentication is mandatory.
6. The SSL or TLS client sends the randomly generated data that enables both the client and the server to compute the secret key to be used for encrypting subsequent message data. The randomly generated data itself is encrypted with the server’s public key.
7. The SSL or TLS server verifies the client’s certificate.
8. The SSL or TLS client sends the server a “finished” message, which is encrypted with the secret key, indicating that the client part of the handshake is complete.
9. The SSL or TLS server sends the client a “finished” message, which is encrypted with the secret key, indicating that the server part of the handshake is complete.
10. For the duration of the SSL or TLS session, the server and client can now exchange messages that are symmetrically encrypted with the shared secret key.

Secure Server-Client Program using OpenSSL in C

In this assignment we will create a secure connection between client and server using the TLS1.2 protocol. In this communication, the client sends an XML request to the server which contains the username and password. The server verifies the XML request, if it is valid then it sends a proper XML response to the client or gives a message of Invalid Request.

Assignment Steps:

- Install the OpenSSL library, for the ubuntu.
- Generate your own certificate with the command below and explain every argument of this command:
    - **openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mycert.pem -out mycert.pem**
- Open and fill the client.c in order to establish a communication with a server.
- Open and fill the server.c in order to listen to requests from a client program.

Tool Specifications

1. Compile the Server: **gcc -Wall -o server server.c -L/usr/lib -lssl -lcrypto**
2. Compile the Client : **gcc -Wall -o client slient.c -L/usr/lib -lssl -lcrypto**
3. First run the server. Example: **sudo ./server 8082**
    a. Why should you use the sudo command?
    b. What is the number 8082?
4. Then run the client. Example: **./client 127.0.0.1 8082**
    a. What is 127.0.0.1?
    b. What is 8082?
5. If the client sends a **valid** request then server gives a proper response like then one below:
    a. Client Request:

```
<Body>
        <User>Sousi</UserName>
        <Password>123</Password>
</Body>
```

    b. Server Response:

```
<Body>
        <Name>sousi.com</Name>
```

<year>1.5</year>
<BlogType>Embedede and c c++</BlogType>
<Author>John Johny</Author>
</Body>

6. If the client sends a **invalid** request to the server, then the server responds to an **“Invalid message”.**
    a. Client Request:
    <Body>
    <UserName>Sousi</UserName>
    <Password>12345</Password>
    </Body>/
    b. Server Response:
    “Invalid Message”

Useful Links

OpenSSL Library: https://www.openssl.org/

Important Notes

1. You need to submit the client.c and server.c that are giving the answers mentioned in Steps: (5) and (6) in the Tool Specification Section, including a README file and a Makefile. The README file should briefly describe your tool. You should place all these files in a folder named <AM>_assign2 and then compress it as a .zip file. For example, if your login is 2022123456 the folder should be named 2022123456_assign2 you should commit 2022123456_assign2.zip.
2. **Google** your questions first.
3. Use the tab “Συζήτηση” in courses for questions.
4. Do not copy-paste code from online examples, we will know ;)