Computer Networks Project: Encrypted File Transfer Application

Zekun Zhang

December 11th 2018

1 Introduction

In this project I designed and implemented an encrypted file transfer application. The goal of this application is to provide users with easy access to the core functionalities of a file transfer application without worrying about the safety of the whole file transfer process. The core functionalities of the application include downloading/uploading file from/to the server, deleting files on the server and listing current existing files on the server. The file transfer process is implemented by tcp sockets and the encryption/decryption part is enabled by SSL.

2 Environment Setup

My project requires the ssl module to wrap tcp socket and the openssl module to generate required files so we need to first install it using pip before running the application. Since we make use of SSL to provide security features, we need to provide SSL with necessary certfile and keyfile on the server side. Therefore we need to generate the mentioned two files using command: openssl req -new -newkey rsa:2048 -days 365 -nodes -x509 -keyout server.key -out server.crt (for the command part I referenced **here**) in the directory containing server.py.

3 Use Instruction

The use of the application is fairly simple. To start the application, forward to the directories and first start the server file by entering: python3 server.py server_port and then start the client file by entering python3 client.py client_port. The user is then asked to provide one of the following command options to the client side: dl file_name, ul file_name, del file_name, ls, terminate. To terminate the application just enter terminate and the server/client will close. Note that the user has to enter the exact command on the terminal since it is case sensitive. The failure to do so would receive a warning from the application.

4 Implementation Details

The implementation of the application is composed of two file: server.py and client.py, which is similar to our first assignment. The general structure is that the client and server first establish a connection and then the client side takes an input command from the user and sends a message/receives data based on the command type. The server side then parses the received message and sends back a message/data given the command type.

4.1 SSL

The security part of the application is implemented by the ssl module. We first create a regular tcp socket, like what we did in assignment 1, and then wrap it with a ssl context using ssl.wrap_socket() function. Here the wrapped tcp socket ensures secure data transfer between the server and the client and makes sure that the server/client is the one that we are trying to connect to. The encryption of the file and the handshakes are automatically taken care of by the sslsocket.send() method. Note that on the server side we need to provide the ssl.wrap_socket() function with the certfile and keyfile we set up earlier in the directory. The socket returned by sslsocket.accept() function is automatically wrapped by SSL so we do not need to wrap it again.

4.2 Download Files

To download a file from the server, the client side first send the utf-8 encoded request message to the server. The server would first try to find the requested file. If it can not be found, the server sends back a warning message to the client, otherwise the server will sends a message containing the size of the file (for getting the file size I referenced **here**). The client side would then act accordingly. If the client receives the file size, it will start to prepare for file receival. One thing to note here is that when opening a file in the server/client, we need to enable the 'b' option so that the file is transformed into binary data so the file transfer application will apply to all file types. Then we start the file download process by iterating through bytes of the file (for iterating through bytes in a file I referenced **here**). The process would stop if the received size matches the file size.

4.3 Upload Files

This part is just the reverse process of downloading files and only requires minor code modification. Here we will not dig into the details since it's pretty much the same as above.

4.4 Delete Files

To delete a file on the server, the client side first sends a request to the server. The server will first check whether the file exists on the directory. If the file does not exist, the server will send a warning message to the client, otherwise it will remove the file and send a successful message.

4.5 List Current Files

To list the files on the server, the client side first sends a request to the server. The server will get the current file list and put them into a string message. If there's currently no file in the directory it will send a warning message to the client. By the time the client receives the message, the client will inspect the content and if there's files in the message, it will split the message into a list and print out the file list.

5 References

For the command to generate SSL certfile and keyfile I referenced:
https://www.electricmonk.nl/log/2018/06/02/ssl-tls-client-certificate-verification-with-python-v3-4-sslcontext/

For getting the file size I referenced:
https://stackoverflow.com/questions/6591931/getting-file-size-in-python

For iterating through bytes in a file I referenced:
https://stackoverflow.com/questions/1035340/reading-binary-file-and-looping-over-each-byte