CS 165 Lab
Spring 2020

The starter code given to you contains a simple client and server that communicate over a socket in plaintext. Obviously, messages sent in plaintext are not secure and can be read by a third party. Our goal in this lab is to modify both the client and the server in order to make them communicate securely using TLS.

1 Running the plaintext client and server

1. Download the code from iLearn.
2. cd TLSCache/
3. Run 'source ./scripts/setup.sh'
4. cd build/
5. Run './src/server 9999'
6. Run './src/client 127.0.0.1 9999'

Note: If you are running the above commands on bolt, you may get an error since the port number 9999 may not be free. Try substituting 9999 with another random port number > 10000.

2 Using TLS

libtls is a relatively new C library specifically designed to make setting up TLS connections over sockets easy. It is distributed as a part of the LibreSSL suite of cryptographic tools. The starter code given to you already contains the latest version of libtls that you can start using immediately. However, we must first create the public and private keys for the client and the server, and obtain a digital certificate for the server.

2.1 Certificate Authorities and chains of trust

A Certificate Authority (CA) is a trusted third party that can issue digital certificates. A digital certificate is used to determine that a public key belongs to the named subject of the certificate. There are two kinds of certificates in use: *root certificates*, that come preinstalled in most browsers and operating systems, and *intermediate certificates*, which are signed by the CA using the root certificate. Nearly all certificates issued to servers on the internet are signed using intermediate certificates, and not the root certificate. Write a short note on why this design is used for issuing certificates. What are the possible security issues that could arise if all server certificates were signed by the root certificate instead? How does using intermediate certificates mitigate these issues?

2.2 Creating root, intermediate and server certificates

1. cd TLSCache/certificates
2. Run ‘make’

2.3 Adding TLS to client and server

Your task in this section is to modify the client and server code to enable them to communicate over TLS. The official documentation for libtls can be found here. You will need the following subset of functions and structs :

1. *tls_init()*: To initialize the internal libtls structs.
2. *struct tls_config*: To set a CA root certificate file.
3. *struct tls_ctx*: To store the libtls context returned by *tls_server()* or *tls_client()*.
4. *tls_config_new()*: To create an empty tls_config struct.
5. *tls_config_set_ca_file()*: To set the CA root certificate file, *root.pem*.
6. *tls_config_set_cert_file()*: To set the server’s certificate file, *server.crt*.
7. *tls_config_set_key_file()*: To set the server’s public and private keys in file *server.key*.
8. *tls_accept_socket()*: Analogous to UNIX accept().
9. *tls_client()*: Creates a tls_ctx struct that can be connected to socket.
10. *tls_configure()*: Applies the configuration in tls_config to tls_ctx.
11. *tls_connect_socket()*: Attaches the tls_ctx to the socket.
12. *tls_handshake()*: Performs the TLS handshake.
    May return TLS_WANT_POLLIN or TLS_WANT_POLLOUT.
13. *tls_read() and tls_write()*: Similar to libc’s read() and write(), securely read and write to and from the socket. May return TLS_WANT_POLLIN or TLS_WANT_POLLOUT.
14. *tls_close()*: To close and tear down the TLS connection.