COSC310 - Assessment 4: TCP socket server/client v2 Security Report

By Chris Cody - ID:220209093

Unencrypted communication

Communication between the client and server was previously unencrypted. This meant that an attacker could intercept, read and potentially manipulate the messages sent between the client and server. This is a major security issue as the messages sent between the client and server contain sensitive information such as the user's password and the key/value pairs stored on the server. It creates potential for an attacker to implement a man-in-the-middle attack to steal the user's password and key/value pairs or manipulate data in transmission between server and client.

In the new version of this protocol, all communication between the client and server is encrypted using a TLS socket and open SSL. The server is authenticated using a server certificate and key file. For testing, a 2048-bit RSA private key and self-signed certificate (valid for 1 year) are supplied. This method of communication prevents an attacker from intercepting and reading the encrypted messages sent between the client and server.

For ease of testing, the client ignores the hostname in the certificate (so the server can run on any host) and will accept a self-signed certificate. The code can easily be modified to require a certificate signed by a trusted CA and to check the hostname in the certificate. This would be a good idea for a production system and would prevent an attacker from impersonating the server or intercepting client/server communication.

Server Authentication

The original protocol did not include a method for server authentication. The client connected to and trusted any server instance. In this version, the server can be authenticated using SSL. Purchasing an SSL certificate tied to a specific hostname would allow the client to authenticate the server, making it very difficult for an attacker to spoof the server without gaining access to the server SSL certificate and key. This would be a good idea for a production system and would prevent an attacker from impersonating the server and stealing the user's password/key/value pairs.

Client Authentication

The original protocol did not require any client authentication. The client sent an ID to the server and was instantly connected.

In the new version of the protocol, a client ID and password are are required for connection. The server confirms that the client id exists in the user database and that the password matches before a connection is established. Passwords are hashed and salted before storage/transmission (not in plaintext). The salt would usually be kept secret on the server, perhaps as an environment variable in memory rather than a hardcoded value or stored in the database. To keep this project simple, the salt and user database is hardcoded. This is a very bad idea for production code. Also, this implementation is limited to the two hardcoded users. New users can not be added, removed or passwords changed.

In a production situation, I would recommend use of an authentication package/library to minimise any risk of introducing security issues via the user authentication system. A library can also manage secure storage of the user database, adding new users, password changes, use of MFA etc... As these libraries can add significant additional complexity, I have not used them on in this project. The two hardcoded used are sufficient for demonstrating basic password authentication.

Corrupted or intercepted data

To original protocol had no error detection and the client had no way of detecting if a value returned after a GET command was changed, either accidentally during transmission or tampered with by a bad actor. In the second version of this protocol, the server returns the requested value along with a CRC. The client can compare the received CRC with the one it generates from the received value. If the checksums match, the client can be confident that the value returned by the server is the same as the value stored on the server. If the checksums do not match, the client can be confident that the value returned by the server is not the same as the value stored on the server. This helps to prevent an attacker from manipulating the value returned by the server.