CS 495: Capstone Report 3
Tyler Allen
10/20/2014

The goals for this week were to implement user authentication and network encryption. In addition, the database schema created in the previous report was to be applied to the database. Adding default likes was not completed this week, as well as implementation of the login screen.

Android recommends that encryption be done via Hypertext Transfer Protocol Secure (HTTPS)/Secure Socket Layer (SSL) where possible[1]. SSL was chosen because our server base is not currently built to handle web HTTP(S) requests. This approach is ideal because support for SSL is built directly into the Java API and Android Framework, allowing us to conform to the standards in place and to create a more straightforward process of integrating encryption into our system. There is no downside to using exclusively SSL because HTTPS is simply HTTP with SSL included. HTTPS was actually tested first, and it was decided that implementing support for HTTP or integrating a library for that purpose was unnecessary; we have no need to support the HTTP GET/PULL requests because nearly every data transaction will require both the client and server to send and receive data.

Implementing SSL support introduced a number of problems. It was necessary to research the steps required to implement SSL support on both the client and server. While there is built-in support for SSL sockets in both Android and Java, the process of properly configuring SSL is more challenging. In order to test client-server communication, the SSL sockets were reverted to simple Transmission Control Protocol (TCP) sockets. It was not possible to create a connection from the client to the server in this manner, and a great deal of time was spent debugging it and confirming with the Server Administrator that the configuration was correct before it was realized that the router providing the server with internet access did not have the appropriate ports open. Afterwards, we were able to upgrade back to HTTPS, which was the original target. Shortly afterwards the client and server were refactored to use only SSL because, as previously mentioned, the HTTPS configuration was unnecessary.

SSL functionality requires SSL certificates. Certificates are necessary because, after the communication issues were resolved, SSL handshake errors were encountered. For the purposes of this application, a self-signed SSL certificate was sufficient. A certificate and keystore were created for both the client and server and appropriately loaded into the programs. Android and Java had different processes for properly loading the certificates. After resolving the issues in loading the certificates correctly, the SSL connection is now functioning correctly.

[1] "Security with HTTPS and SSL | Android Developers." 2013. 20 Oct. 2014 <https://developer.android.com/training/articles/security-ssl.html>

On the client, our UML from a previous report was deviated from slightly. Android supports a service known as a `SyncService` that utilizes `SyncAdapter`, which allow the client to synchronize with the server on a separate thread[2]. This required the integration of an additional service known as an `AccountService`, which contains logic for how to access data from `Accounts`. This led to the use of the `Accounts` class, which, in conjunction with the `AccountService`, will serve as our representation of user accounts. Utilizing these services, our application will be able to retrieve a User's stored data from Shared Preferences, once a user has created an account, and send that data to the `SyncAdapter` for transfer to the server for authentication. The `SyncService` and `SyncAdapter` using Android Framework `Intents`, and a series of XML and permissions configurations. A good amount of time was spent configuring the previously mentioned services and classes before they would operate appropriately; a slight mistake in an XML file is sufficient to cause the entire synchronization process to fail without any warning or debug information. The `SyncAdapter` approach, while arduous, provides us with a solid network interface that can be used modularly throughout the rest of the project, and provides us to trigger synchronization in a number of ways that may not have been feasible without it. The Android Framework lists the following times when a sync is easily usable using `SyncAdapter`: sync when server data changes, sync when content provider data changes, sync when network messages are received, sync periodically, and sync on demand[3]. This will provide us with several means to keep our application data up to date in the future.

During the course of this report an alternative way to handle user authentication was encountered. As opposed to sending a user's password over the network, a token authentication system could be used[4]. This seems potentially worthwhile, but requires more research from reliable sources. In the event this is to be implemented, the Java API supports hashing algorithms that may be useful in our token authentication[5].

Additionally, the Facebook Login API was researched as a possible login alternative[6]. This was mentioned previously in the initial proposal, and seems like it would be good alongside our own implementation. Unfortunately, they use a token authorization system and, as of now, it does not seem like it would be easily integratable into the current authentication system.

For the majority of the test process, the Server was set up to simply print the contents of messages received from clients due to the number of issues in connecting the client and server. This has been partially deprecated, and now follows the UML. Java Database Connectivity (JDBC) has been set up in the `Authenticator`, but Structured Query Language (SQL) statements have not been written

---

[2] "Creating a Sync Adapter | Android Developers." 2013. 20 Oct. 2014 <http://developer.android.com/training/sync-adapters/creating-sync-adapter.html>

[3] "Running a Sync Adapter | Android Developers." 2013. 20 Oct. 2014 <http://developer.android.com/training/sync-adapters/running-sync-adapter.html>

[4] "Token-based authentication - Securing the token - Security ..." 20 Oct. 2014 <http://security.stackexchange.com/questions/19676/token-based-authentication-securing-the-token>

[5] "MessageDigest (Java Platform SE 7) - Oracle Documentation." 2011. 20 Oct. 2014 <http://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html>

[6] "Facebook Login for Android - Facebook Developers." 2014. 20 Oct. 2014 <https://developers.facebook.com/docs/android/login-with-facebook/v2.0>

to check against the database as of yet, so any information given is treated as valid login information. At this point, the database schema was appropriately applied to the database, and, after test accounts are added to the database, we will be ready to approach testing user authentication.