Design Document
Jammed Final Release (3.0)
5.5.15

I. Confidentiality

- We ensure confidentiality in both network transmission of data and in data stored on the server's machine. All information sent between the client and the server is sent over a connection secured by Java's implementation of SSL, meaning that data sent over the network will be secure.

  In order to ensure the confidentiality of the data while it is stored on the server, we store it in an encrypted state. We only ever store the keys to this file on the client's local machine; they are never transmitted over the network. Therefore, if the client's machine is secure, the data stored on the server is also secure. Additionally, the keys themselves on the client machine are encrypted with the user's password, so only someone who knows the password can decrypt the client's data.

  The data is encrypted using the AES block cipher with CBC block cipher mode and PKCS5 padding. The key length used is 256 bits. This is stored locally on the client machine in a protected state with an IV file. It is protected via the user's login password which is hashed and salted then used to decrypt the key and load it into the program.

  Additionally, we generate a new IV every time the data is re-encrypted. We store this IV on the server with the encrypted user data. The IV length is 64 bits long.

II. Integrity

- Our system is concerned with the preservation of integrity across the network, as it is assumed that both the server and client machines are free from corruption. Therefore, in order to be assured that integrity was maintained, we relied on Java's existing SSL implementation and the MAC-then-encrypt policies underlying it.

III. Audit

- Our system performing auditing by writing to logs whenever the state of the server changes. There are two sets of logs that are maintained, the first is a general server log that only the server admin has access to and records all state changes on the server. The second set of logs are logs for each specific user that record state changes specific to that user.

  A state change is defined by any action that alters the information on the server. These include receiving and sending data and login requests, user data changes, and account level changes such as creation or deletion.

IV. Authentication

- Authentication can be broken down into two separate parts. The first is enrollment of a new user, and the second is login of an existing user. To for a new user *u* to enroll into the system, they enter an enrollment request, and then enter their desired user name *un* and master password *mp*. The information is then sent to the server, which checks to ensure that no user with username *un* already exists on the server and that the password entered is not null or an empty string. If that condition is met, then the server generates a new secure random salt of 16 bytes, and hashes the password with that salt with twenty thousand iterations and a key length of five hundred twelve bits using PBKDF2WithHmacSHA1. The bytes of the salt and the resulting hash are then stored in a file within the newly created user folder. At this point enrollment is complete in terms of authentication.

- For user *u* to then login, they enter their username and password associated with that username. The server performs a lookup for that username, if it exists, it reads the password file to get the salt and the hashed password. The server then hashes the presented password using the same stored salt and hash method as described above, and then compares the resulting hash to the stored hash. If the two hashes are equal, then the user is successfully authenticated and the server grants them access to only the data stored within their user folder. If any of these steps fail then the user is not authenticated and must try again. Additionally, the system ensures that only one instance of a connection can exist per user, using a synchronized set of logged in usernames.

V. Authorization

- Our system implements a very simple access control policy where users are able to view files which only exist in their user folder. Because all contact between the server database and the user is mediated by the server code, this was quite simple to implement.

We create a different folder for every user, named for their username, which is required by the server to be strictly alphanumeric and unique. When a user authenticates with some username, the server will only ever read files from that user’s folder in that session. It does this by calling all appropriate database functions with the correct username as a parameter, which are guaranteed to only read from that username’s file. These functions also write to the server log, but do not read from it.