Assurance

`Jammed` Alpha Release (1.0)

3.20.15

I. UserData

A. Description: This Class securely encrypts/decrypts the users data on the local machine, with a secret Key which remains only on client machines.

B. Testing: To test this class a program was made which using print statements to check functionality does the following operations.

1. creates and stores a Key to the local machine (enrollment)
2. takes an input which simulates a “userdata” and passes it through encoding
3. passes encoded data and IV used to a variable
4. loads the key from the machine.
5. uses the key and IV to decode the encrypted user data
6. displays decrypted data

C. Additional testing was done on the portion of the class that transforms a list of objects representing user data into a string representation of that data for storage in a database, and vice versa.

1. `stringToList`: This method was tested by creating both valid and invalid input strings by hand and running the function on them. Their validity in list form could be verified by simply printing out the values in the list.
2. `listToString`: This method was tested by manually creating a list of LoginInfo objects and calling the method on it, assuring us that the method would execute correctly on arbitrary cases.
3. Tests were also performed to ensure that the function `listToString`(`stringToList`) was an identity function on a string, ensuring that our method would always be able to properly parse its own output.

II. Request and Subclasses

A. Description: `Request` is the Superclass of all `Request` objects, which act as a message packet, for sending information between the client and server. Subclasses are divided by type of request; `login`, `log`, `userdata`, and `termination`. Each request has a corresponding response so that the issuer can establish a success or failure of their request.

B. Testing: This was done in place with actual transmission through the session between `Jelly` and `Jammed`. Since these classes do not have functions but primarily are holders for data, no real testing was necessary.

III. UserInterface

A. Description: This is the class that displays the (human-readable) user data and allows the user to make changes to it. For now, it simply prints to and reads from the command line; however, in order to facilitate the integration of a GUI in the future, it has been made into its own class.

B. Testing: As this is an interactive class, a test program was constructed that would allow us to interact with this class independently of other classes. We verified that the data was output in a correct and readable fashion and that changes input to the data were correctly saved and returned.

IV. Jammed

A. Description: This is the main class for the client application. It consists of a main method which prompts the user for login information, verifies this with the server, and requests/displays the user's data. If the user chooses to make changes to this data, it then uploads the new version of said data to the server for storage.

B. Testing: As this component is very difficult to unit test, testing for this was done by running it on a variety of inputs and user behaviors, and seeing how it behaved.

V. Jelly

A. Description: This is the main class for the server application. It consists of a main method which creates the server and database, and accepts a client connection. Once this connection is established, it starts a session that handles all relevant client requests.

B. Testing consisted of running it with `Jammed` and ensuring that basic functionality was present. Additionally, response type and data correctness was checked to ensure the server was sending back correct information, later confirmed again by checking after reception on the client side. This will be expanded upon to ensure input sanitation, user authentication and authorization, expanded logging, more

specific logging and error messages, and proper SSL certificate usage and authentication.

VI. Communication

A. Description: This class serves as a utility class for the server and client applications. It abstracts SSL communication using sockets to a constructor five methods: `send`, `receive`, `connect`, `accept`, and `close`. The constructor returns either a server or client socket, depending on the specified type. The other methods do as expected, with a special note of `connect` and `accept` being exclusive to client and server instances, respectively, of `Communications`.

B. Testing consisted of running `Jelly` and `Jammed`, debugging as needed. Since this is a wrapper class for Java's SSL libraries, testing focused more on correctness of implementation rather than correctness of functionality.

VII. DataBase (DB)

A. Description: This classes serves as a utility class for the server application. Its purpose is to abstract file management away from the server, and provide methods for initializing the file tree, reading and writing files, creating new users, searching for the existence of a user, and reading and writing user specific files. Because there is no instance information being maintained, and due to the utility nature of this class, it was decided that it would be a static class. This class was written by `mvp34`, and reviewed by the other members of the group.

B. Testing was performed on an individual method basis, a main test method in a junit test class was used to make method calls that would expected from the server, and the output was observed directly through the directory structure, and by comparing the data that was read with the data that was written to ensure that they were equivalent. Tests were run after any change or new feature was implemented.

1. `public static boolean initialize()`: This method was tested by calling it in the main method, and then checking the directory structure to ensure that the proper directories and files were initiated. This method was written under the assumption that the server would only ever call it once upon initial install, but was tested with multiple calls nonetheless. Upon multiple calls, the method simply returns true without any action, because it checks that the file structure already exists.
2. `public static boolean newUser(String uid)`: This method was tested by calling it in the main method with a sample input and then checking the file structure for the creation of the new user folder. Then it was tested by calling it again with same input to ensure no action

was taken because the method checks to ensure that no user already exists with user id input.

3. `public static boolean searchUser(String uid)`: This method was called with sample input to ensure that it returned true if the given uid had already been created, and false otherwise.
4. `public static boolean writeLog(String dataToLog)`: This method was tested by calling it with sample input and ensuring that it appended data to the log by viewing the log file after it had been written to.
5. `public static String readLog()`: This method was tested by calling it in the junit test framework and comparing it to the expected output.
6. `public static String readUserLog(String uid, DBFileTypes fileType)`: This method was tested by ensuring that it read the data that had been previously written to a specific user log, and returned null otherwise, such as a non existent user.
7. `public static boolean writeUserLog(String uid, DBFileTypes fileType, String fileData)`: This method was tested by writing sample data to a user log, and reading the data back to make sure that it was recorded correctly. The method was also tested against bad input to ensure it performed no action and returned false.
8. `public static byte[] readEncodedFile(String uid, DBFileTypes fileType)`: This method was tested by reading data that had been previously written, and ensuring that the information matched. It was tested to ensure that no action was performed if the user did not exist, and that the correct files were read from.
9. `public static boolean writeEncodedFile(String uid, DBFileTypes fileType, byte[] fileData)`: This method was tested by writing sample data to each of the file types and then reading it back to ensure it was written correctly. This method was tested on bad input, such as a user not existing.