Escalona, J.M.

Group 10

Case Study 1 - Authentication

Security Issue	**Java File**	**Vulnerabilities**	**Solution**
No input validation on login input fields	Login.java SQLite.java	Logging in does not verify login input fields and will let the user proceed to the main screen of the application regardless of the content. Unvalidated inputs also make the application susceptible to SQL injections.	Login.java: (**IMPLEMENTED**) Verify that both the username AND password fields are not empty and comply with character validity will be based on the rules on minimum credential requirements upon registering which are: 1. Lowercase letters: a-z 2. Numbers: 0-9 3. Symbols (Username): _-. 4. Symbols (Password): ~`!@#$%^&*()_-+={[}]|\:;"'<,>.?/
No user verification	Login.java SQLite.java	Users are not necessarily verified to exist on the database before proceeding to checking authenticity. *This is not strictly for security purposes but rather includes performance ones.	Login.java: (**IMPLEMENTED**) Add user verification which will dictate whether the user authenticity will proceed or not. Assumes login input verification returns true. SQLite.java: (**IMPLEMENTED**) Implement function that would only return a Boolean value to indicate whether user exists or not instead of the already implemented SQLite.getUsers() which returns all users including plaintext passwords of users.
Password visible	Login.java Register.java	Password visible on screen.	Login.java & Register.java: Change password fields and confirm password fields from JTextField to JPasswordField.
Passwords not hashed	SQLite.java User.java (via the DB)	Passwords are stored and accessed in the DB in plaintext.	SQLite.java: (**IMPLEMENTED**) Implement hashing at a controller level for both

		Prone to brute force or dictionary attacks should a known user be found.	password verification (read) and password saving (write). User.java: (**IMPLEMENTED**) Change values of 'password' to hash equivalent in database. *DB was rewritten such that all passwords in plaintext are hashed.
No user authenticity verification (i.e., username & password match)	Login.java	User can enter application's main screen without proper credentials. Login logging is also recommended to be implemented should login attempts be successful or not.	Login.java: (**IMPLEMENTED**) Implement controls that would verify the username and password before proceeding to the main screen should it be successful. Logging (both successful and not) will also be added for audit purposes.
No invalid login pop-ups/notifiers	Login.java	Should a login-related error occur, the user should be notified so in a manner that is ambiguous or generic.	Login.java: (**IMPLEMENTED**) Implement necessary login error pop-ups that is generic should an invalid log-in condition occur.
No lock-out mechanism	Login.java SQLite.java	Repetitive unsuccessful login attempts should not be allowed and does not stop user from logging in after certain retries at a certain time. Prone to brute force attacks. Note the ff.: 1. By this point, a known username must have been known by the malicious actor already. 2. Unlocking the account is recommended to be done with user (administrator) intervention. The administrator must also restore the user accounts original	Login.java: (**IMPLEMENTED**) 1. Implement login lockout measure should successive missed logins are attempted. 2. Should the user be locked, login attempt won't proceed (assumes that username is known) whether the entered password is correct or not. 3. Successive login attempts limited to 8 tries. Once it reaches the limit, the account will be locked immediately. SQLite.java: (**IMPLEMENTED**) 1. Add a function that will set a given (valid) user as locked (Role code 1 and locked 1). 2. Add a function that will set a given (valid user as unlocked (Defaults to role code 2 and locked 0).

		role number as unlocking an account reverts the account to a client role.	
Exposed and incorrectly placed function	Register.java Frame.java	Users (programmers) may accidentally call the function which could unwantedly alter contents of the database as it is set as a public rather than a private one. It is also written on a parent class which other View classes could inherit and use even if it is not related to the registration feature of the program.	Register.java & Frame.java: (**IMPLEMENTED**) 1. Remove Frame.registerAction() 2. Transfer it to Register.java
No input validation on register input fields	Register.java	Inputs for user account credentials do not require minimum credential requirements such as minimum length (especially for passwords) and character composition. Action of register was also found to be lost and accessible (public) on another class file Frame.java	Register.java: (**IMPLEMENTED**) 1. Add input validations for each user field such that it meets the following: a. Uppercase letters (Username only): A-Z b. Lowercase letters: a-z c. Numbers: 0-9 d. Symbols (Username): _-. e. Symbols (Password): ~`!@#$%^&*()_-+={[}]|\:;"'<,>.?/ 2. Add username availability checking.