Login System

Problem Statement - Build a login system for a web application that and ensure that user passwords are stored securely.

Objectives

A. Store user passwords in a way that makes it difficult for an attacker who gains access to the database to determine users' actual passwords.

B. Allow users to login by verifying their passwords without storing their passwords in plain text.

C. Provides an additional layer of security to prevent brute force attack.

Implementation

A Login System is an essential part of most applications, wherein the identity of a user is verified before they are allowed to access the services provided by the app.

Using C++, we have implemented a very primitive login system in the command line, which allows users to perform signup and login, and exit the application.

User passwords, during signup, are randomly salted and subsequently hashed before being stored in the database, which makes it extremely difficult for an attacker to reverse the hash or brute force the encryption module.

During login too, passwords input by the user are stored for an extremely short while in the memory, as they are immediately hashed after input and then are matched against the pre-encrypted values in the database.

Our code also implements a rate-limiter on the password entry module, which ensures brute force attacks are impeded by restricting input after a certain number of inputs.

Program Flow

When we run the code, this is how it works:

A. A menu screen is displayed, that shows enumerated options "signup", "login", and "exit". Input is taken in the form of natural numbers.

B. If the user enters '1', the program goes to the signup menu, and the user is prompted to enter a new user id. This input is checked against the database to ensure the provided user id doesn't already exist, and if it does the user is prompted to try again. Then the control moves back to the menu screen.

When a valid user id is entered, the user is prompted to provide a password for their account, which is then salted, hashed, and pushed alongwith the user id and other relevant data to the User database. A success message is displayed after this and the control goes back to the menu screen.

C. When the user enters '2', the program goes to the login menu, and the user is prompted to enter their username. This input is checked against the database to ensure the user exists, and if the program is not able to find said user a message informing the user of such is displayed, and the control goes back to the menu.

If the user id exists, the program prompts a password entry, and the user is given 5 chances to input their password. If the user fails to input their password correctly in given number of attempts, an error message is displayed and password entry is restricted for 5 seconds.

D. Provided the user inputs the correct password, control goes into the user menu where the options to change password, username, and to logout are displayed.

Input is taken in the same way as the main menu;

```
 1. If the user chooses to change their password, a password entry field is displayed and the input taken is modified
in the same way as it is during signup, and changes pushed to the database.

2. If the user chooses to modify their username, an input field is displayed, and the input is checked against the
database to ensure the new username doesn't already exist. Then the changes are pushed to the database.

3. Choosing the logout option moves the control flow to the main menu.
```

E. If the user enters '3', the program exits and any changes made to the database are saved permanently (this is a txt file which stores the username alongwith the hashed passwords and salt).

Algorithm

Algorithms utilised by our program include the Mersenne Twister PRNG, Salting, and CPP Stl Hashing.

The MTPRNG is used to initialise the salting method, which creates a random salt for every password, which is then hashed with the cpp standard library method.

Complexity Analysis

The PRNG and the hashing method from the STL are initialized in constant time.

Salting takes time O(n) in the length of the password, as the salt is generated in a for loop from a predefined charset.

Hashing in the stl is linear as well.

Login and signup operations are O(n) in numbers of existing users.

Overall, the time complexity of the program is O(n), and the space complexity is linear as well considering we store an extra salt and hash with the user data.

Features

- Database Encryption
- Rate Limiter
- Memory Security
- Data Storage

Drawbacks

- No admin mode, thus, no way to access user data during runtime.
- Users are searched linearly, which would throttle the speed of the program as the number of users increase.
- No way to remove users (follows from the program not having an admin mode).

Authors

- @AnarchistHoneybun Raj Rajeshwar Singh Bisen , 2210110917 , rs135
- @dtele Dhruv Sharma , 2210110259 , ds332
- @Arhamjain1 Arham Jain , 2210110924 , aj919
- @Xhadou Pratyush Jain , 2210110970 , pj825