- **Title:**

Secure Login System, UN Secure Login System Simulation

- **Author:**

Eng. NAJD FARIS A ALEID

Description: This project aims to develop a secure login system using password hashing and database storage. Users can register and login with their credentials securely. The project demonstrates the importance of password security and user authentication.

Code: [APP-SECURE - Replit https://replit.com/@NAJDALEID/APP-SECURE]

Code: [app-not-secure - Replit https://replit.com/@NAJDALEID/app-not-secure]

[ALEID], [NAJD]. "Secure Login System." " UN Secure Login System.", [https://github.com/NAJDAL/CYBER-PROJECT-APP-SECURE-NOT], [June 13 2023].

- **Purpose:**

The purpose of the project is to develop a secure login system using a combination of password hashing and database storage. The system allows users to register and login with their credentials securely.

- **Authentication:**

The project focuses on user authentication, which is a fundamental aspect of secure systems. By implementing a login system, the project aims to verify the identity of users before granting them access to protected resources.

- **Password Hashing:**

The project uses a strong password hashing mechanism to protect user passwords. Originally, the bcrypt library was utilized for this purpose. However, in the unsecure version, plain text passwords are stored for learning purposes, which is not recommended in real-world scenarios.

- **Database Storage:**

The project utilizes SQLite, a lightweight and embedded relational database management system. It demonstrates how to create a user table in the database to store user information securely.

- **User Registration:**

The project includes a user registration process where users can create an account by providing a unique username and password. It performs validation checks to ensure that duplicate usernames are not allowed.

- **User Login:**

Once registered, users can log in to the system by providing their username and password. The system verifies the provided credentials against the stored information in the database to authenticate the user.

- **Menu-driven Interface:**

The project implements a menu-driven interface to interact with the login system. Users can choose between registration, login, or quitting the system. Invalid choices are handled with appropriate error messages.

- **Security Considerations:**

The secure version of the project incorporates industry-standard security practices, such as password hashing and salt generation using bcrypt. It aims to protect user passwords from potential attackers by making it difficult to reverse-engineer the original password from the stored hashed value.

- **Educational Value:**

The unsecure version of the project has been provided for educational purposes, allowing learners to understand the potential vulnerabilities of storing passwords in plain text. It serves as a lesson in the importance of secure password storage and the risks associated with weak security practices.

By working on this project, learners can gain practical experience in implementing secure login systems, understand the significance of password hashing, and explore various security considerations in user authentication.

The code you provided implements a simple secure login system using the bcrypt library for password hashing and SQLite for storing user information. Here's a breakdown of how the code works:

1. The **SecureLoginSystem** class is defined, which handles user registration, login, and database operations.
2. In the **__init__** method, the class establishes a connection to the SQLite database and creates a table called "users" if it doesn't exist.
3. The **create_user_table** method executes an SQL statement to create the "users" table with columns for user ID, username, and hashed password.
4. The **register_user** method prompts the user to enter a username and password. It checks if the username already exists in the database, and if not, it hashes the password using bcrypt and inserts the username and hashed password into the "users" table.
5. The **hash_password** method takes a password as input, generates a salt using bcrypt's **gensalt** function, and hashes the password using **hashpw**. The hashed password is then returned.
6. The **login_user** method prompts the user to enter their username and password. It fetches the hashed password from the database based on the provided username, and if the user exists and the provided password matches the hashed password, it returns a success message. Otherwise, it returns an error message.
7. The **check_password** method takes a password and a hashed password as input and uses bcrypt's **checkpw** function to verify if the password matches the hashed password.
8. The **close_connection** method closes the database connection.
9. The **run** method is the main control flow of the program. It displays a menu for the user to register, login, or quit. Based on the user's choice, it calls the corresponding method and displays the result.
10. Finally, the program creates an instance of the **SecureLoginSystem** class and calls the **run** method to start the login system.

To use this code, you need to have the **bcrypt** library installed.

You can install it using pip: **pip install bcrypt**. Additionally, the code assumes that you have a file named **users.db** in the same directory where the code is executed to store the SQLite database.

Unsecure version:

Removed the use of bcrypt library for password hashing.

Removed the hash_password method responsible for generating salt and hashing the password.

Stored the passwords in plain text format in the database.

Removed the check_password method that verifies the password against the stored hashed password.

Please note that storing passwords in plain text is highly discouraged in real-world scenarios due to security vulnerabilities. This unsecure version is for educational purposes to understand the importance of proper password hashing and security practices.