SecureLogin

Two Way Authentication System Project for CSCE 3550.070

Project Owner:

Matthew Fredericksen

Professor:

Dr. Pradhumna Shrestha

University of North Texas

Department of Computer Science and Engineering

Date:

July 14, 2020

Contents

Figures

Introduction

Project Selection

I chose to complete the "Two Way Authentication System" project, one of the four options described in the *CSCE 3550/5550: Project Topics* document. I selected this project primarily because I was already familiar with sending SMS using Python, which was one of the requirements for this project.

Project Description

The purpose of this project is to create a secure login interface. By secure, I mean that it should be as hard as possible for an attacker to break into someone's account (this is called authentication) or for any third party to learn information about the user (this is called confidentiality).

The project's requirements are as follows:

1. The application must have a GUI.
2. The application must allow users to create an account.
    a. The user's account must consist of a username, password, and phone number.
    b. Account information may be stored on the user's computer.
    c. The user's password and phone number must not be stored in plaintext; the password must be hashed, and the phone number must be encrypted.
3. The application must allow users to log in.
    a. If the user's username and password are valid, they must be sent a text message containing a secret code.
    b. If the user correctly enters the secret code, then they have successfully logged in.
4. No further interface is necessary after a successful log in.

Practical Application

This project protects a user's information through cryptography, a process of transforming information into an unrecognizable form. When a user creates an account with this login app, their password is never stored; instead a complicated string of data (called a hash) is computed based on the original password, and this value is stored. This makes it virtually impossible for anyone viewing the stored data to determine what the original password was. A very similar approach (called AES encryption) is taken with the user's phone number, but that process is designed to be reversable so that the SMS verification code can be sent to the user.

By using these processes before storing a user's data, their private information becomes vastly more secure and safe from attackers who try to steal data. This kind of login interface is ideal for protecting your personal information.

Methodology

Languages and Libraries

The SecureLogin app is implemented using two languages: Python 3.7 and Kivy language (kvlang). Nearly all GUI design was implemented with kvlang, while business logic, functional aspects of the GUI, and two-factor authentication were implemented with Python.

Python was selected primarily for its ease-of-use and familiarity to the project owner. Additionally, Python offers many libraries for implementing security features, as shown below.

Kvlang was used as a core feature of the Kivy Python library, providing separation of concerns and simplifying the overall project structure by isolating GUI design.

Within Python, several libraries/modules were imported, including the following:

- cryptography: provides symmetric encryption.
- Kivy: provides GUI creation tools.
- passlib: provides password hashing algorithms.
- phonenumbers: provides phone number validation and formatting.
- secrets: provides secure string comparison and secure randomness.
- sqlite3: provides DBMS features for storing user account information.
- twilio: provides SMS capabilities.

More details about how these libraries were utilized will be provided in the following section.

Security Implementation

Security features were implemented both for stored user data and active user interactions with the application.

Stored User Data

Confidentiality vulnerabilities may arise from an attacker gaining access to the stored password and phone number information. To address this, the user's password is encrypted using the bcrypt hashing algorithm. Bcrypt was chosen for it's high computational cost and key-stretching properties, providing resistance to brute force attacks. The app utilizes a variation of bcrypt which first runs the password through HMAC-SHA2-256, eliminating the password-truncation effect of bcrypt, which otherwise limits effective password length to 72 bytes.

The user's phone number is encrypted using the high-level Fernet symmetric key encryption algorithm (which is built on AES). The Fernet key is generated from the user's password and a hash digest of the username, which are run through PBKDF2, a key-derivation function which again provides resistance to brute force attacks through key stretching.

All user data is validated before being inserted into the database. Even after validation, the application uses parameterization to guarantee protection from SQL injection attacks.

Active User Interactions

Confidentiality vulnerabilities may arise from a third party viewing the user's screen or from the user leaving the application unattended.

To address the former, the application takes the following measures:

- Passwords are masked so that no character of the user's password is visible on their screen at any time.
- When a user verifies their log-in attempt via SMS, only the last 4 digits of their phone number are made visible on the screen, so that no more information is shown than necessary.

To address the latter, the application takes the following measures:

- Password fields are wiped after a 30-second period of inactivity. If the user walks away from the application after typing their credentials, an attacker will not be able to sign in or even view their password length.
- All input fields are wiped when transitioning screens, so that previously entered data cannot be accessed later.

SMS Verification

The application implements two-factor authentication through SMS; in this way, it guarantees not only that the user *knows* the correct username and password, but also that they *have* the correct phone. Even if an attacker manages to steal a user's credentials, they will not be able to sign in without the user's phone.

The SMS verification aspect of the application also takes security measures. The verification code is generated using Python's secrets module, which provides access to the most secure source of randomness available to the OS. Additionally, the secrets module provides a `compare_digest` function which protects against string comparison timing attacks. This defense is probably overkill, however, since the verification code expires after 30 seconds or 3 failed attempts to enter it.

Results

Program Interaction

The GUI is divided into 3 main "screens" that the user can interact with: (1) the Login Screen, (2) the Create Account Screen, and (3) the Verification Screen.

The Login Screen

This is what you first see when opening the app. You are able to sign in with existing credentials, if you have already created an account, or you can click the "Don't have an account?" button to move to the Create Account Screen.

The application provides helpful popups when invalid user actions occur. For instance, if a user attempts to sign in with an incorrect username or password, they will see the alert shown in *figure 2*.

Additionally, a password that has been typed in will clear after 30 seconds of inactivity (*figure 3*).

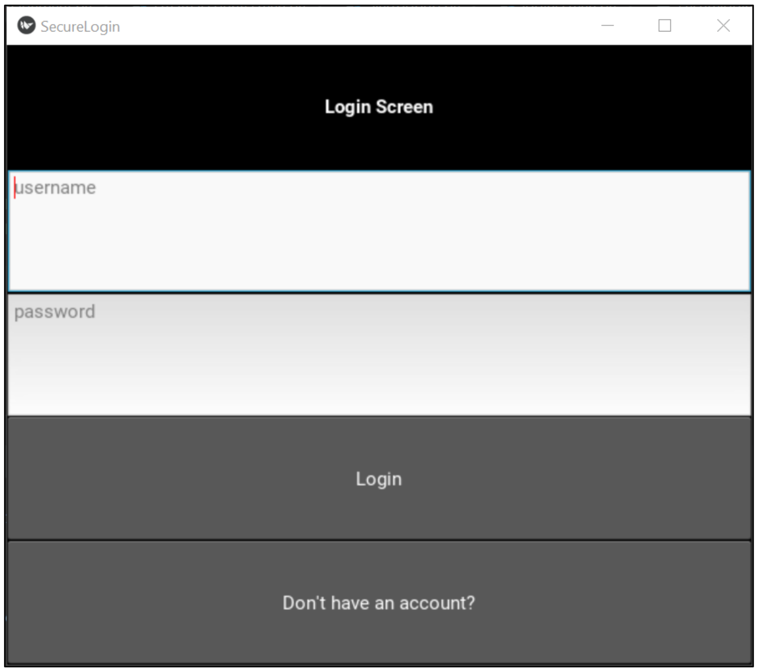
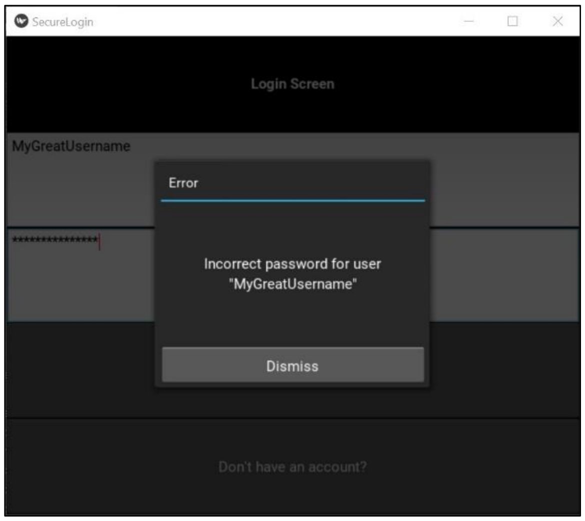
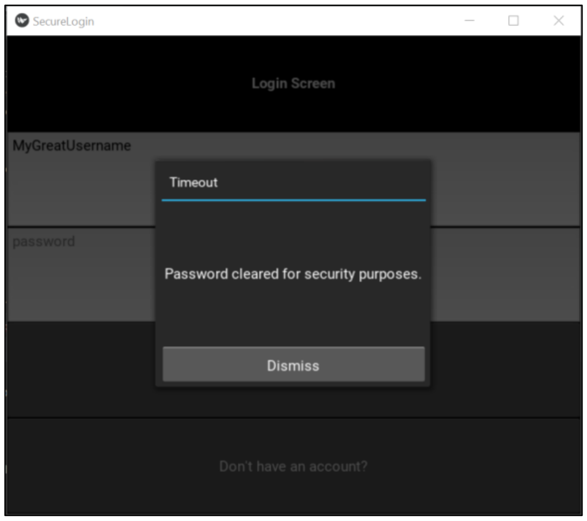


Figure 1: Login Screen



Incorrect password for user "MyGreatUsername"

Figure 2: Incorrect password alert



Password cleared for security purposes.

Figure 3: Password timeout alert

The Create Account Screen

Upon clicking the “Don’t have an account?” button on the Login Screen, the app transitions to the Create Account Screen. Here, you can type in information for creating a new account. As with the Login Screen, the app helpfully displays messages when it is unable to accept input (*figure 5*).

The app handles [tab] and [enter] to allow you to interact with the input fields intuitively.

A unique feature of this page is the automatic formatting of the phone number field as it is filled.

You will be redirected to the Login Screen upon successfully creating an account (*figure 6*).

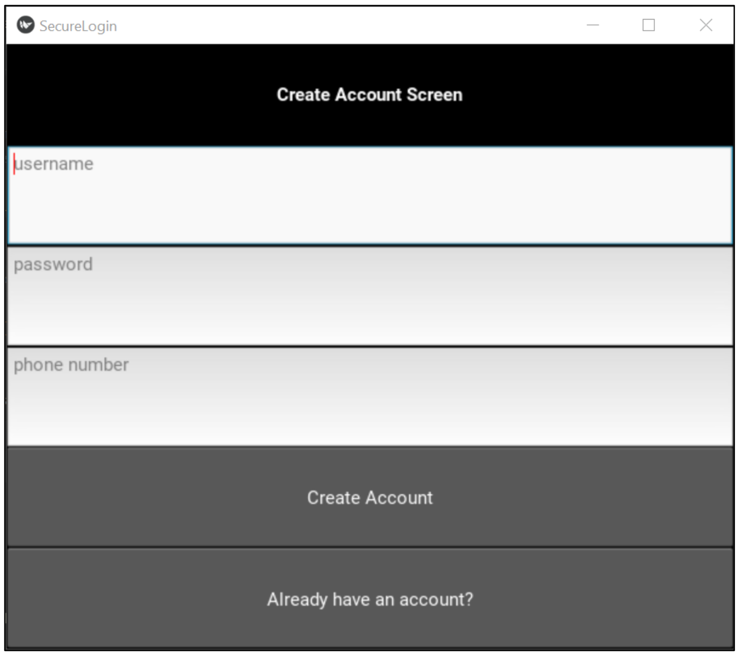
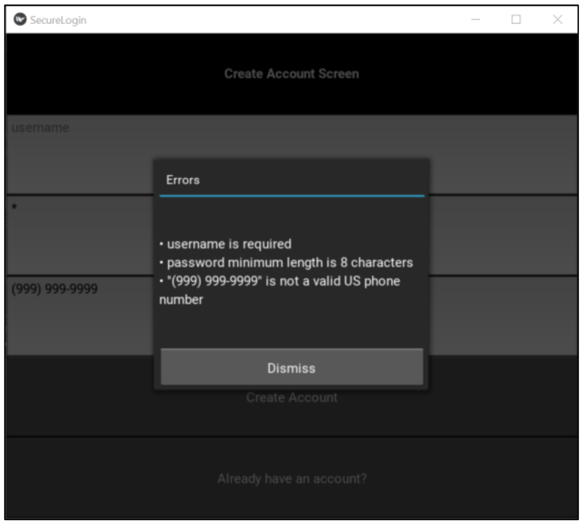
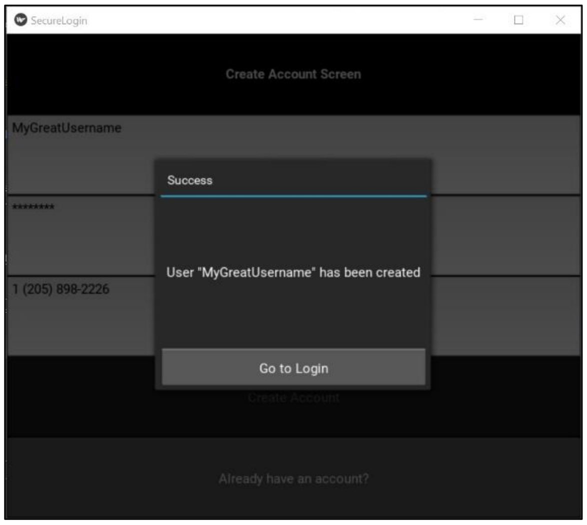


Figure 4: Create Account Screen



• username is required
• password minimum length is 8 characters
• "(999) 999-9999" is not a valid US phone number

Figure 5: Example errors for account creation



User "MyGreatUsername" has been created

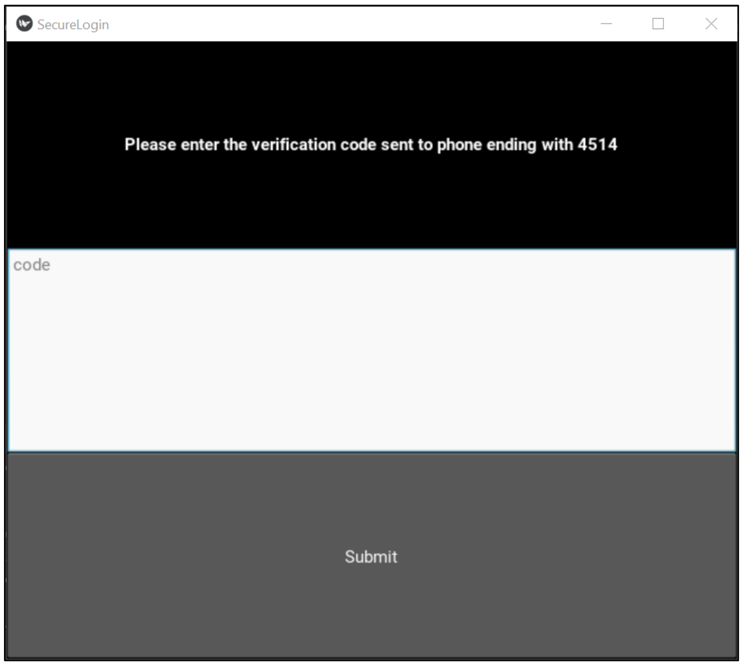
Figure 6: Successful account creation

The Verification Screen

This is the final screen you will interact with in this application. Upon entering a valid username and password, the phone number that was entered during account creation will be sent a randomly generated code that must be entered before you are fully logged in.

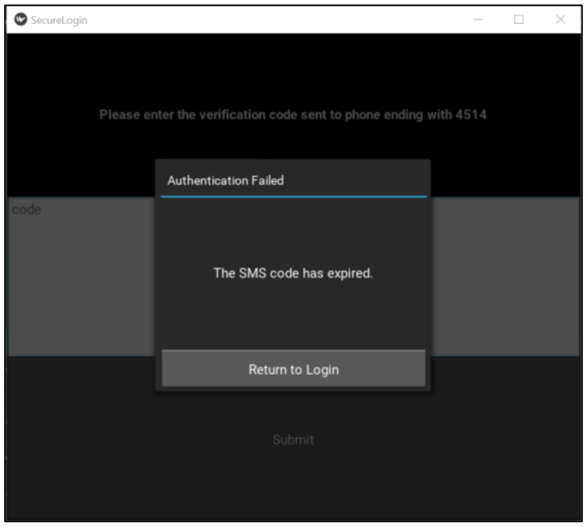
If the code is not entered within 30 seconds, or if it is entered incorrectly three times, the log in attempt will fail and you will be redirected to the Login Screen (*figure 8*).

Since this application only provides a secure login interface, once you have logged in your only option is to log back out (*figure 9*).



Please enter the verification code sent to phone ending with 4514

Figure 7: Verification Screen

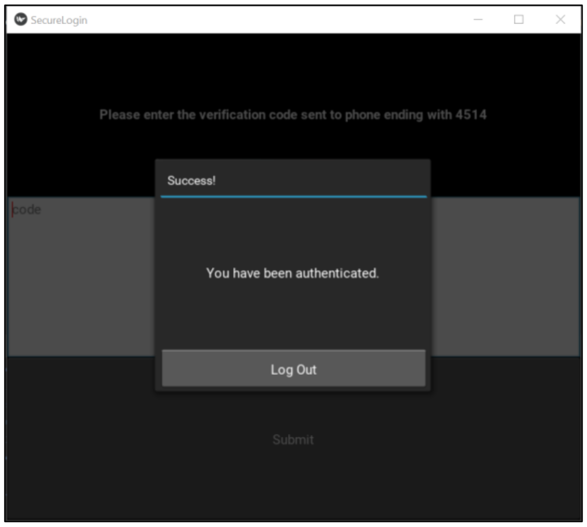


Please enter the verification code sent to phone ending with 4514

Authentication Failed

The SMS code has expired.

Figure 8: Code expiration alert



Please enter the verification code sent to phone ending with 4514

Success!

You have been authenticated.

Figure 9: Successful login

Video Demonstration

A video walkthrough of this application is available here on YouTube.

Conclusion

In this final section, I will briefly share my thoughts regarding this project.

I learned quite a bit as I researched how to best implement this project. As far as technical knowledge, I learned about various hash functions as I browsed passlib's options, I learned about key stretching, and I learned about the advantages of using hash algorithms that can easily increase their computational cost to stay relevant as processor technology advances. I was already aware of SQL injection attacks, but I learned how to use parameterization (not complicated at all).

I think I learned the most about Kivy, the GUI creation library. Probably upwards of 80% of the time I put into this project was devoted to learning Kivy and designing the functional aspects of the UI. I think the experience I gained with Kivy is the most valuable thing I will take away from this project.

As for non-technical knowledge, I learned that code can have some pretty obvious flaws that might not be noticed with only one pair of eyes. I was almost complete with the project before I noticed that I was allowing username checking to be case-sensitive. I also learned that writing reports is a lot less fun than writing code.

If I had more time, there are a few issues I would solve immediately. For instance, the application currently does not verify ownership of a phone number, so a user could spam someone's phone with verification codes if they wanted (note to the grader: please don't do this). After that, I would focus on making the GUI more aesthetically pleasing. It currently uses default sizes and styles for every widget. After that, I would probably set up a way to lock a user out of the database after a certain number of failed log in attempts, perhaps with a cooldown period.