

Password Manager: Documentation

Description of the Application

Main idea of the application is simple: to provide a centralized platform for storing and managing passwords securely. Users can add, view, copy, modify and delete passwords as needed. The application supports multiple users, with each user's data encryption using strong, unique encryption methods. Additionally, users can assess the strength of their existing passwords when adding them and generate robust, random passwords when needed. To ensure security measures align with industry standards, the OWASP Top 10 list served as a comprehensive checklist for implementing security solutions within the program.

The program relies on several dependencies, all of which are detailed in the README file along with instruction for executing the application. These dependencies primarily encompass cryptographic solutions and the graphical user interface (GUI). To streamline installation, these dependencies can be installed using the 'pip install' command. It's important to note that if 'pip' is not already installed on the system, it must be installed first before proceeding with the dependency installation process.

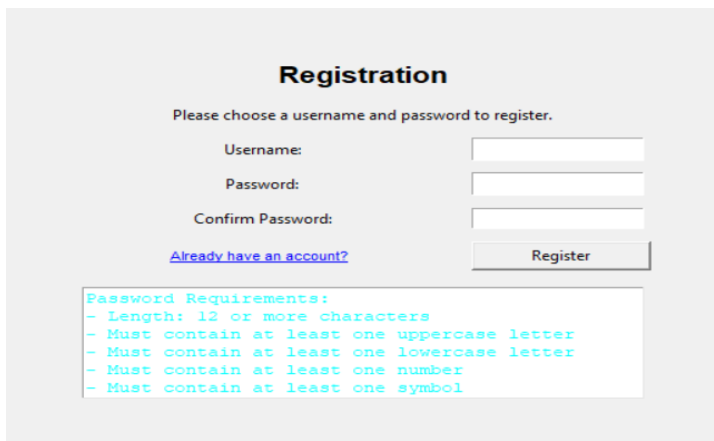
The application can be executed by running command 'Pyhon gui.py'. The login-window (Figure 1) should open.



Figure 1. Login View

In the login view, users input their credentials and proceed to log in. If an error occurs during the login process, such as an invalid user or incorrect password, a notification is displayed under the login button. This notification is highlighted

with red text to draw immediate attention to the issue, making it clear to the user that there was a problem with their login attempt. In the application, a new user can register by clicking the 'New user? Register' button, which opens the Register view (Figure 2).

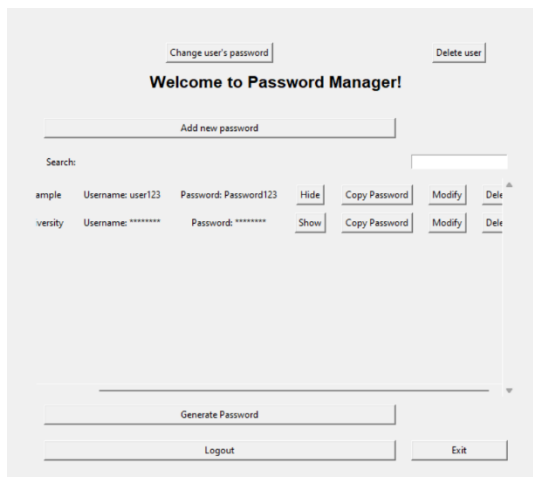


The screenshot shows a 'Registration' form with the title 'Registration' in bold. Below the title is the instruction 'Please choose a username and password to register.' There are three input fields labeled 'Username:', 'Password:', and 'Confirm Password:'. To the right of these fields is a 'Register' button. Below the input fields is a link that says 'Already have an account?'. At the bottom of the form is a box titled 'Password Requirements:' containing a list of rules: '- Length: 12 or more characters', '- Must contain at least one uppercase letter', '- Must contain at least one lowercase letter', '- Must contain at least one number', and '- Must contain at least one symbol'.

This view resembles the Login views, but it prompts the new user to provide their login information. Similar to the login view, error messages are displayed in red text to alert the user in case of any issue, and password requirements are listed under the Register button. After successful

Figure 2. Registration View

registration, the user is redirected to the login page, where they must log in separately using the credentials they just registered.



The screenshot shows the 'Main View' of the application. At the top, there are two buttons: 'Change user's password' and 'Delete user'. Below these is the title 'Welcome to Password Manager!'. There is a button 'Add new password' and a search bar labeled 'Search:'. Below the search bar is a table with two rows of password entries. The first row has columns for 'Username: user123', 'Password: Password123', 'Hide', 'Copy Password', 'Modify', and 'Delete'. The second row has columns for 'Username: *****', 'Password: *****', 'Show', 'Copy Password', 'Modify', and 'Delete'. At the bottom of the form are three buttons: 'Generate Password', 'Logout', and 'Exit'.

Figure 3. Main View

Once a user successfully logs in, they are directed to the Main window (Figure 3). This window serves as the central hub for all password-related actions. Users can interact with a list of passwords, each of which can be viewed, copied, modified, or deleted. The password many is scrollable both horizontally and vertically.

Clicking the “View” button reveals both username and password associated with a specific entry, and they can be hidden again with the “Hide” button for added security. Additionally, user have the ability to add new passwords, change their

own password, or delete their account entirely. Strong passwords can be generated on demand.

Furthermore, the application includes a search feature, allowing users to find specific passwords by entering the website name associated with each entry. Finally, users have the option to logout or exit the entire application, providing flexibility in managing their account and ensuring security.

Add new password information

Enter website's name, username and password.

Website's name:

Username for website:

Password:

Password score (scale 0-4): 1 Weak Password
Add another word or two. Uncommon words are better
.Capitalization doesn't help very much.

Figure 4. Add new password View

When the user clicks “Add new password”, a new view opens where they can input the website name, username and password details (Figure 4). To assess the strength of the password, users can click “Estimate password”, which provides a score ranging from 0 to 4, along with any additional suggestions if applicable. After entering the password, users can save it to the database by clicking “Save information” button. If any errors occur during

the information-saving process, they are promptly notified below the button, with the error message displayed in red text for clarity.

In the main view, passwords can be modified by clicking the modify button, which opens a new view similar to the “Add new password” view. In this modification view, users must first confirm their old password before entering and confirming the new password. Similar to “Add new password” view, users have the option to estimate the strength of the new password before saving it to the database. The process involves clicking “Estimate password”, which provides a score along with any relevant suggestions. Once the password is entered and all security checks are successfully passed, users can save the information to the database by clicking “Save information”. Any errors encountered during the password-saving process are handled similarly to other views, with error messages displayed in red text for user notification.

Change password

Enter old and new password.

Old password: ☐ Show Password

New password:

Confirm password: ☐ Show password

Password Requirements:
- Length: 12 or more characters
- Must contain at least one uppercase letter
- Must contain at least one lowercase letter
- Must contain at least one number
- Must contain at least one symbol

Figure 5. Change User Password View

User can change his/her own password by clicking ‘Change user password’ in main view. User need to confirm the old password and after that feed the new password as in password modification. There is no password estimation since there already exists strong password requirements which are informed to the user. Errors are informed similar way than earlier. After successful password change the user is logged out.

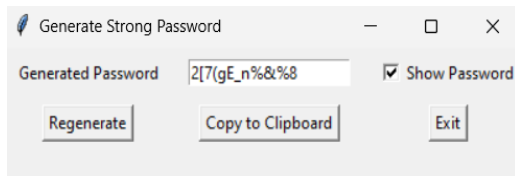


Figure 6. Generate Strong Password View

While deleting password or user account, the user is asked to verify it with popup window. 'Generate Password' button also opens a popup window where user can view the generated password, copy it to clipboard, regenerate it and close the window.

Structure of the Program

Dividing the program into five separate files serves to organize the code and facilitate the connection between different components. Each file has its own specific purpose within the program, contributing to modularity and ease of maintenance.

Authentication

In the 'authentication.py' file, the authentication logic, registration process, and creation of the SQL database are implemented. Each function within this file performs basic checks in the database, while input validation and encryption functionalities are handled in separate file.

Data Access

In the 'data_access.py' file, all database-related communication, apart from authentication, is handled. This file is responsible for various database operations, such as adding new passwords, fetching specific information from the database, modifying passwords (both from the password list and the user's login password), and deleting information from the database. It serves as the primary interface with the database, making connections, executing queries, and handling data retrieval and manipulation.

GUI

In the GUI file (gui.py), the graphical user interface of the application is constructed, with most functions dedicated to implementing various views or popup windows. While the primary focus is on GUI implementation, a few functions, not directly related to GUI elements, are also included in this file. These functions may handle certain functionalities that could not be efficiently implemented in the controller file, or where the implementation in the controller would become overly complex.

Controller (password_manager.py)

In the 'password_manager.py' file, the connection between the view side and the functionality side of the application is established. This file serves as the controller,

mediating communication between the graphical user interface and the underlying functionalities. Through 'password_manager.py', the GUI communicates with necessary functionalities to perform actions such as adding, retrieving, or deleting data from the database.

Security

In the 'security.py' file, security-related functionalities such as input validation, data encryption, password hashing and key derivation are implemented. This file is responsible for ensuring that the application adheres to security best practices by validating user inputs, encrypting sensitive data, and securely managing passwords. Password encryption and decryption, along with key derivation and salt implementation, are handled within 'security.py' to protect user passwords and sensitive information stored in the database.

Security

Program's security side was implemented and checked with OWASP Top 10 list. Next let's go through some security solutions related to the list.

Authentication

The program aligns with the OWASP Top 10 list, particularly addressing the issues of Broken Access Control and Identification and Authentication Failures through several measures. All users have access only to their own data, which they have personally added to the database. This strict access control mechanism ensures that users can only manipulate data they own, reducing the risk of unauthorized access or data breaches. The login password, also used as the master password, has stringent requirements to enhance authentication security. By enforcing strong password policies, the program mitigates the risk of identification and authentication failures due to weak or easily guessable passwords. The same access control mechanism is applied uniformly to every new user, ensuring consistency and predictability in user access privileges. This approach reduces the likelihood of misconfigurations or oversights leading to access control vulnerabilities. To mitigate brute force attacks and authentication failures, the program implements rate limiting for login attempts. After five consecutive failed login attempts, the user is required to wait for 30 seconds before attempting to log in again. Additionally, the timeout period increases by 10 seconds with each subsequent failed attempt. This strategy helps prevent automated attacks while still allowing legitimate users to access the system securely.

By addressing these aspects, the program strengthens authentication mechanisms, reduces the risk of access control vulnerabilities, and enhances overall security posture in accordance with OWASP guidelines.

Cryptography

The login passwords are securely stored in the database after undergoing hashing using the Python module `hashlib.sha256`, which implements the SHA-256 hashing algorithm. This algorithm transforms input into a fixed-size hash value (256 bits) that uniquely represents the original data. SHA-256 is chosen for its widespread use and recognized security in hashing.

Key derivation utilizes PBKDF2 (Password-Based Key Derivation Function 2), a method for deriving cryptographic keys from passwords. In this program, the key is derived from the login key (also known as the master key) and a randomly generated salt. The salt, produced using the `os.random-function`, generates cryptographically secure random bytes suitable for this purpose. The resulting derived key is then employed with the encryption algorithm for data encryption.

Encryption employs AES 256 in CBC (Cipher Block Chaining) mode, renowned for its robust encryption strength. This method ensures that brute force attacks are computationally infeasible due to the immense key space provided by AES 256. CBC mode introduces chaining, enhancing security by XORing each plaintext block with the previous ciphertext block before encryption. Additionally, the use of an Initialization Vector (IV) further fortifies the encryption process by adding randomness. This ensures that even if the same plaintext is encrypted multiple times with the same key, the resulting ciphertext will vary.

Each data securing algorithm and function is meticulously selected to optimize data security. These measures safeguard against cryptographic failures and fortify defences against brute force attacks and other security breaches, ensuring the highest level of data protection.

Input Validation

Validating user inputs before storing them in the database is crucial for maintaining data integrity and security. All inputs undergo rigorous validation to mitigate the risk of SQL injections and XSS (cross-site scripting) attacks.

Login credentials undergo further validation, ensuring that usernames and passwords meet specific size and content requirements. This is particularly stringent for the login password, which adheres to strict criteria to enhance security.

To prevent SQL injection attacks, parameterized queries are employed when fetching or adding information to the database. This method ensures that user-supplied input is treated as data rather than executable code, thereby minimizing the risk of malicious SQL injection attacks.

Design

From design to final implementation, security has been a top priority at every stage of development. The OWASP Top 10 list and other reputable security sources were consulted to ensure comprehensive security measures throughout the coding process. Authentication and encryption were foundational components designed and implemented first in this program, serving as cornerstones for all subsequent development.

Sensitive information stored in variables is promptly cleared when no longer needed, reducing exposure to potential threats. Additionally, every GUI functionality is meticulously checked for potential security vulnerabilities. All components and additional libraries utilized in this application are renowned for their security and have undergone thorough vetting to ensure trustworthiness and reliability. This dedication to security ensures the protection of user data and the overall integrity of the application.

Additional Security Components

The application includes two valuable features for users: a robust password generator and a password strength estimator.

The strong password generator empowers users to create randomized, highly secure passwords for various platforms, enhancing the security of their accounts. By utilizing this tool, users can confidently implement strong passwords across multiple accounts, bolstering their overall security posture. Estimates are implemented with zxcvbn which is a password strength estimation tool developed by Dropbox. It analyses passwords and provides an estimate of their strength based on factors such as length, complexity, and predictability. It produces lot of useful information but in this program only the score and possible suggestions are viewed to the user.

Meanwhile, the password strength estimator allows users to evaluate the strength of their existing passwords. It provides insights into the security level of passwords added or modified within the application, along with suggestions for strengthening weak passwords. This feature enables users to improve the security of their accounts by optimizing their password choices.

Both tools contribute to user security not only within the application but also across other platforms. By offering a convenient way to generate strong passwords and assess password strength, the application empowers users to safeguard their accounts effectively.

Testing

After completing the application, manual tests were conducted to identify and debug any errors. These tests covered authentication, cryptography, and input validation, which are crucial aspects of the program's security. Fortunately, the manual tests did not uncover any significant new issues, indicating that the application was robust in these key areas.

Due to my limited experience with test design, only a limited number of manual tests were performed. With increased skills and perhaps more time, additional tests, such as penetration testing, could have been conducted to further assess the security of the application. This could involve simulating security-breaking attacks to identify potential vulnerabilities and strengthen the overall security posture of the program.

Vulnerabilities and Further work

While the application's security features and encryption methods are robust, there are a few vulnerabilities that remain unresolved due to time constraints. One notable vulnerability is the storage of the master password value in a global variable after successful user login. Although this is necessary for decrypting and encrypting password data, it poses a security risk as the value is accessible within the application's memory. While the value is cleared upon user logout, storing it in a variable introduces a potential vulnerability.

Another deficiency is the lack of logging for suspicious behaviours, such as repeated injection attacks or failed login attempts. Time constraints prevented the implementation of logging functionality, which could help detect and mitigate security threats more effectively. Additionally, the application could benefit from enhanced authentication measures, such as two-factor authentication and CAPTCHA tests to prevent automated

attacks. Implementing a timeout for inactive user sessions would also bolster security by automatically logging out users after a period of inactivity.

Despite these vulnerabilities and deficiencies, the application addresses common security issues effectively and provides users with a user-friendly interface for managing their passwords. Ongoing improvements and updates could further enhance the application's security posture over time.