ScrypKey Password Manager

Methodology:

Creation of vault: We take the master password as input from the user. The code generates a random salt and uses the Scrypt hashing algorithm to hash the user's input password together with the salt. The resulting hash is then stored in a secure database along with the salt.

Authenticating the master password: When the user wants to log in, we again ask the user to input the master password. The code retrieves the stored salt and uses it to hash the user's input password with the Scrypt algorithm. The resulting hash is compared to the stored hash, and if they match, the password is considered valid and the user is given access.

Options:

- Add password
- Search password
- Modify password
- Read all passwords
- Delete password
- Generate password
- Change master password

Benefits of our approach:

The Scrypt algorithm is designed to be slow and memory-intensive, which makes it difficult for attackers to perform brute-force attacks or dictionary attacks. This is because scrypt requires a large amount of memory to compute the hash function, which makes it much harder to perform attacks even with specialized hardware like ASICs or GPUs. As a result, the scrypt algorithm is considered to be more resistant to attacks than other popular algorithms like bcrypt or PBKDF2.

Another advantage of using Scrypt is that it allows for the use of larger salt values. This makes it more difficult for attackers to precompute hashes for commonly used passwords, as they would need to compute a separate hash for each possible salt value.

It’s obvious that since the passwords are not stored in plaintext, even if an attacker gains access to the database, they will not be able to retrieve the actual passwords without first cracking the hashes.

Our methodology provides protection against a variety of security threats, including brute-force attacks, dictionary attacks, and attacks that use precomputed hash tables or rainbow tables.

Usability:

- **Password generator:** User can now generate truly random and secure passwords of a desired length.
- **Password search:** User can search for passwords without knowing the exact website URL
- **Data scrubbing:** User activity won't be logged in the terminal output
- **Timeout:** After 90 seconds of idle, all sensitive info is deleted and user gets logged out
- **Auto Copy & Paste Logins:** Passwords are not displayed on the screen, instead are directly exported to the clipboard
- **Clear clipboard after pasting:** Once user have pasted your password to the required destination, it gets deleted from the clipboard
- **Hashing:** Uses Scrypt algorithm
- **Usability:** As the main flaw with current password managers was cluttered UI, we used a terminal based approach to keep it neat and tidy without any clutters.
- **Offline Storage:** No data is stored anywhere on the web or online databases, keeping it securely offline in an encrypted form.
- **No Third Party intervention:** from terminal based UI to offline database, there is no third party intervention like sketchy data hosts or infected browsers.
- **No Intrusive Dialog Boxes:** There are no dialog boxes used, all information is displayed over the terminal with minimal effort required from the user's point of view.

Usage:

- Download the repository
- Run command `$ pip install -r requirements`

- Run command $ python main.py
- You are good to go
- If running for the first time, you will be prompted with setting up the vault with a master password.
- After setup enter master password to login
- Welcome to password manager.

UserFlow:

The Final App design flow is: link