Password Manager

Theodore Contes   A. Connor Waters

1 Introduction

Good passwords are a critical part of keeping information secure in the current day, but maintaining good passwords is becoming more difficult every day. Not only do you need long, complex passwords to defeat standard attacks, such as dictionary attacks or brute force methods, but you need multiple such passwords. Password reuse leaves oneself open to having multiple accounts and services vulnerable if any password is compromised. But the alternative is to memorize many passwords, and then memorize new ones when you need to change those passwords. For many, this is not a practical solution, and so they instead leave themselves open to attacks through weak passwords and password reuse.

This project aims to provide a solution to the problem, by allowing a user to memorize a single strong password, and then store the rest of their password information behind this password. The password manager will create and store passwords in encrypted files, allowing the user to access them but not others.

Password managers do have some weaknesses; inherently they rely on a single password to protect all of the other passwords, thus introducing a single point of failure. If the single password is compromised then all of the passwords and accounts stored within the manager are at risk. Alternatively, if the access to the password manager is lost, then the user has just lost access to all of their accounts. This can be mitigated through backups and storing the information and encryption keys on multiple computers, though this is not perfect. Finally, password managers cannot be used for all passwords, if only because they are stored on computers: to access the password manager you must first be able to access the computer it is on. Thus, while they greatly reduce the number of passwords needed, users are still required to know more than one strong password for good protection.

2 Related Work

The concept of a password manager is not new. There are many such programs offered, and they come with a vast array of useful features if you are willing to pay the price. As many of these services offer data backup, these costs are generally subscription based rather than one time. Interfaces to password managers range from simple terminal commands to browser extensions that automatically save and input passwords for web accounts without any prompting from the user. Our application uses a simple graphical user interface. The user is first prompted to input a master password or, if it is their first time using the software, prompted to create a master password, and this password is used to generate an encryption key to protect the password database.

3 Design

Our program is written in Python and makes use of well-tested free libraries for encryption and graphical interfacing. It uses AES-256 encryption, the key derived using the PBKDF2 hashing algorithm as specified in RFC 2898, both provided by the open-source PyCrypto module (`Crypto.Cipher.AES` and `Crypto.Protocol.KDF`, respectively). The interface is rendered via the Tkinter API, a wrapper around the Tk widget toolkit provided in the Python standard library.

The encryption key is generated by PBKDF2 using the master password and a randomly-generated salt. The salt is kept in a separate file, unencrypted because it does not leak any information about the key or the master password. When the master password is input, a decryption key is generated using it and the stored hash, and the database is decrypted with the resulting key and its first several bytes compared to a sentinel value to check for proper decryption. If the sentinel is found, the program continues, the correct password having been entered with extremely high probability; if it is not found, the password was not entered correctly, and the user is reprompted.

The graphical front-end and the password manager back-end are separate modules which talk to each other via a set of callback functions. The "application" is a Python script which runs the necessary steps for initialization, prompts the user for the password, and then releases control to the event-driven graphical code, which makes calls into the database as necessary for the operations the user requests.

4 Project Description

The program opens with a prompt. If this is the user's first time using the password manager, the prompt asks the user to set a master password to be used to encrypt and access the stored password database in the future; if the user has already set a master password, the prompt asks the user to input the password to access the database.

On decrypting the database, the app opens into a screen that displays all of the stored passwords, listed by their unique identifiers, and offers the option to add a new password, view, edit, or delete an existing password, or open an "Options" menu.

If the user chooses to add a new password, a dialog is opened that asks the user to input a unique name for the password (for example, "Facebook account" or "Bank PIN") and the password itself. The dialog also offers the option to generate a strong password based on a set of given parameters, including character set and length. Once a password has been input and a name has been given, the user hits “Confirm” and the password is added to the database.

If the user selects an existing password, they have the option to view, edit, or delete it. If they select “View”, a dialog opens that shows the identifier and password. If they select “Edit”, a similar dialog opens, showing the identifier and prompting the user to enter a new password. If they select “Delete”, a warning dialog opens, and the user can either confirm or cancel the permanent deletion of their password.

If the user opens the “Options” panel, they are presented with a few auxiliary functions: They can change the master password, generate a new encryption key using the same password, or permanently delete the password database.