FOUNDATIONS OF INFORMATION SECURITY

CSD365
Group 8:
Kaavya Jain (1910110185)
Prakriti Agrawal (1910110290)
Rimjhim Singh (1910110317)
Ramya Karna (1910110308)

ANALYSIS AND IMPLEMENTATION OF PASSWORD MANAGERS

22nd April 2022

MOTIVATION

As the world becomes more and more digital, users may find themselves integrating almost every aspect of their life into online services such as online banking, google services, etc. Assuming that the user is following security practices and has a different password for each service, this leads to one user having many passwords - too many to remember, hence we decided to combine all the passwords into a Password Manager under a strong master password.

PROBLEM DEFINITION

Analysis of different Password Managers and Implementation of one of our own Password Manager "PassMan" having majority of the features we found in existing programs.

BACKGROUND

A password manager is a program that helps us store, generate and manage passwords on our local desktop or online usage. These passwords are usually generated and stored in what is commonly referred to as a password vault, that might take place using an encrypted database, to enhance security.

Often, users tend to reuse their same old passwords for multiple websites due to laziness and convenience. However, this makes you susceptible to cybersecurity issues that have the ability to affect work and personal files alike.

Password managers that are being commonly used these days are for personal use as well as for workplace security.

There have been some obvious security issues that have arisen in the past, and we must understand whether these have been addressed by information security experts or not.

Autofill is a common feature of password managers, where the user saves their password for a particular website, and the password manager itself fills the saved credentials without us having to prompt for the particular manager. This greatly simplifies the process of login and authorisation.

PREVIOUS WORK

There are several types of password managers, as discussed, that perform different functions and cater to different organizations. These can broadly be classified into a few categories: Desktop managers, Online managers, and Portable managers.

In each of these approaches, the strong passwords are typically protected using a master password; at the time of recalling a specific password, the user simply types in her master password. If a user is mobile and uses multiple terminals for authentication (e.g., her desktop at home and her laptop in the office), a desktop manager would not offer any portability to the user. We, therefore, do not consider desktop managers to be of much benefit on their own. The online and portable managers have their own pros and cons. An online manager, although portable, requires the user to trust the third-party service provider(s). Since user's passwords would typically be encrypted using her master password and then stored on remote server(s), they might be vulnerable to offline dictionary attacks.Imagine if all users were to use a remote manager, the passwords corresponding to all of them might be susceptible to an adversarial break-in at the end of the server(s). Moreover, often proprietary, a remote manager might not offer the users any transparency in outsourcing their sensitive information and how this information has been protected.

A portable manager can possibly be more trusted since it can be locally managed by the user on her own trusted portable device. However, all existing phone managers typ-ically involve displaying a (long and random) password on the portable device, which the user is simply asked to copy onto the terminal. Typing in a long and random ASCIIpassword might have poor usability. USB managers do not have this drawback, but they may not offer a desired level of portability and accessibility to a modern user

According to a study that was carried out by analyzing the autofill feature of ten different password managers for desktop autofill functionality, it was observed that the credentials were vulnerable to a network injection attack. This was if the password manager autofill passwords without requiring user interaction. [Silver et al]

Next, Stock and Johns further confirmed the findings of Silver et al., observing logic and authorisation errors, with vulnerabilities to CSRF/XSS attacks.

In the case of mobile autofill services, these were proved to be not as strong, as Aonzo et al. showed that there exist malicious apps that could trick the popular Android autofill services. This could be done by suggesting to the user that they should autofill their credentials for a genuine application into the malicious application. Some of the most popular password managers, like Keeper, LastPass and Dashlane were all found to be vulnerable to the mapping based attacks that occurred because of the gap in mapping strategy found between the application and the online domains utilized by the respective password managers.

However, Google Smart Lock was demonstrated to not be susceptible to any of the above attacks. According to research by Timothy Sean Oesch, it was observed that the iOS Password Autofill framework enforces a secure app-to-domain credential mapping for all password managers, but the iOS app extensions tend to take a quite insecure approach to the app-to-domain mapping.

HISTORY OF PASSWORD MANAGERS

The first successful implementation of an effective password manager which was open to the public was KeePass, developed by Dominik Reichl, with an initial launch in 2003. KeePass is a simple password manager that works on Windows, Mac OS X, Linux, and Android, as well as unofficial Android, iOS, and Blackberry imports. KeePass uses AES-256 bit encryption and ChaCha20-256 bit encryption in its most recent versions.

Following a mixed response to KeePass's launch, it would be a few years before password managers were commercially successful; LastPass, Dashlane, and Roboform are notable examples, with LastPass being the most popular, with a reported 7 million users as of 2015. By the 2010s, it appears that a password manager craze had taken hold. Many developers have begun focusing their time on password managers, and in order to increase their popularity, they have made their work open-source, allowing the general public to use their product and view their code, as well as allowing the technical community to shape a password manager to their liking by providing feedback on what can be added to or improved, or by exposing vulnerabilities.

EXISTING PASSWORD MANAGERS

During the research for our project we looked into some open-source password managers which have been briefly discussed below.

A. Passbolt

Passbolt is an open-source password manager initially developed by Kevin Muller, Diego Lendoiro, Remy Bertot, and Cedric Alfonsi, with later work of Passbolt being supported by the GitHub community. Passbolt is currently only available in a browser. It is only compatible with Firefox and Google Chrome. This is supposedly owing to the fact that Passbolt is still in early development. Passbolt was created in JavaScript, PHP, and Shell, and its encryption standard is currently OpenPGP. Passwords are encrypted in Passbolt, and the client's database can also be secured for added security. User names, on the other hand, are not encrypted and are saved in plain text. The usage of a faulty pseudo-random number generator is one of Passbolt's existing issues. You can't update your master password or utilize multi-factor authentication right now.
If the option for email notification is enabled, you can email yourself a copy of your encrypted passwords. Passbolt also uses a color security token to avoid phishing. Another present weakness reported by the Passbolt team is the situation in which the client and server both trust all keys; while they acknowledge that this is a flaw, they want to solve it in the future.

Reported Security Flaws:
Found early in the surveillance of Passbolt, it was discovered by Wigginton et al that the use of the PHPseclib has the potential to default to the use of ECB encryption. While this isn't a direct flawin Passbolt itself, it is still a flaw to consider. Reported Problems by Passbolt include server integrity problems, DDoSattacks, server information leaks, key revocation, the potential of authentication cookies

being stolen if SSL is broken, and the potential to mimic server keys. It should be noted that Passbolt currently only uses MySQL servers, which havehad reported problems that were recently fixed by Golunski. Given that Passbolt is only in alpha and developed by a single team rather than a company, they do not have the resources to perform a full-stretched security audit; most vulnerabilities have been found by the developers themselves or by the GitHub community. The team's use of cryptographic functions (by use of OpenPGP) has been reviewed by security audit team Cure53. The team was able to find several vulnerabilities in the OpenPGP library but we shall omit the details.

B. Encryptr

Encryptr is an open-source password manager created by Tommy Williams and later purchased by SpiderOak, a business that specializes in developing services with no-knowledge foundations. Encryptr is a JavaScript, HTML, CSS, JSON, and XML-based cross-platform password manager, e-wallet, and note-keeper. The Crypton framework, designed by SpiderOak, was used to create its encryption standard. Crypton is a JavaScript-based open-source framework with the primary objective of storing data on a server without the server ever knowing what is stored. PostgreSQL, Redis, Node.js, and Docker are all used in Crypton's backend. The use of Galois/Counter Mode for encryption and decryption is assumed in AES-256.

ElGamal encryption and ECDSA (Elliptic Curve Digital Signature Algorithm) are used for signature verification, elliptic curve cryptography is used for key generation, and these ciphers can be replaced for others if the user wishes. Crypton's strength is the safety of user data and data sharing, as well as the direct belief in end-to-end encryption, which makes consumers feel safer from attackers and the organization hosting such a service. Crypton additionally employs SRP (Secure Remote Password) authentication, which is said to restrict data compromise by preventing brute-force attacks on AES keys. The capabilities of peer graph analysis and container access frequency analysis are two of Crypton's flaws. The former statement means that because user names are maintained in plain text, database records can be analyzed to establish relationships between users and undertake intelligence gathering, potentially opening up more attack routes.

Reported security flaws:

Only two official security audits were performed and published onCrypton. The main issues reported by Leviathan SecurityGroup include:

1) An account's public key is not verified against the encrypted private key. This could result in a user encrypting something that cannot be decrypted.
2) The public signing key is not verified against the private signing key.
3) A container by the name of containerNameHMacKeyis not verified before decryption, so the server could replace it with a different known container and encrypta new symmetric key to the user's public key

A look into the report by Least Authority gave us some more insight into the security of Crypton which includes:

1. Server information forgery: attackers with access to theserver can overwrite and forge data on a user’s account.
2. Guessable private keys: an attacker with server accesscan grab copies of cipher-text and read the plain-text.
3. An attacker with server access can disclose the en-cryption key, essentially making all container contents available to them.Having read both reports we noticed plenty of the attacks included DDoSing. While not the largest security threat to worry about, it can still cost companies quite a bit of money and reputation. The reports failed to include auditing web-based attacks such as XSS attacks (Cross-Site Scripting),CSRF attacks (Cross-Site Request Forgery), Man-In-the-Browser attacks, and SQL injections.

C. Padloc

Padloc is a simple, open-source password manager created by Martin Kleinschrodt utilizing the Electron and Polymer frameworks. Like Passbolt and Encryptr, it was constructed entirely in JavaScript, HTML, and CSS, with the majority of the code being publicly available. Padlock is cross-platform, supporting Windows, MacOS, Android, iOS, and, in the future, Linux. Padlock, like Passbolt and Encryptr, uses a copy/paste capability to make utilizing one's passwords faster. As previously stated, this results in significant security issues. One of Padlock's standout features is that if no user activity is detected for one minute, the software will immediately log you out of your vault. This feature can be set to a maximum of ten minutes or deactivated entirely if the user so desires. Padlock has its own password generator, however it has a flaw in that it considers 7-character passwords containing at least one uppercase letter, lowercase letter, and special character to be particularly strong. These generated passwords are not considered secure by the 2017 NIST criteria.

Reported security flaws:

Surprisingly, the creator ofPadlock has a repository for penetration testing of his own application. Furthermore, the penetration team Cure53 washired to do even more extensive testing. The reports detail some of the following vulnerabilities:

1. Tap-jacking
2. Exposed authentication tokens during API requests,leading to Man in the Middle attacks
3. Permanent DoS attack on mobile devices: an attacker with server access can increase the number of iterations,essentially making the CPU do a job it cannot do, and ultimately having to make the user reset Padlock if they want to use the application on their phone again.However, resetting Padlock will delete all information stored
4. DoS email attacks

Review of the three password managers:

Overall, Passbolt was not a very user-friendly password manager and we question the integrity of the product. They have no reported security audits on their product except a reference to the audit of

OpenPGP, and theylack many key features other password managers have, which brings into question why they would even release an alpha version of their product. Additionally, their demo page is quite shoddy and the intended use of the password manager seems to often lead attackers to use cunning phishing attacks and DDoS attacks. The design also increases the risk of attack on the main administrator, since they are at the core of how Passbolt should be used by a company. It should also be noted there is no enforcement of a strong master password, which in itself is a big security risk.

Encryptr was the most minimal of the three open-source password managers reviewed by far; it was incredibly simple, it could be used on almost all platforms and did not require the use of an email, yet you could still retrieve the same data from other devices. One Critique of Encryptr is that after some further investigation it was discovered some code is still obfuscated. It is also noted that there is no strict enforcement of strong passwords and generated passwords have a default length of 12 characters.

Padlock lived up to its name of being a minimalist password manager that got the job done and we were quite pleased with the initial security audits that were reported on the application. Overall it is easy to use and the ability to use a custom server was a nice addition.However, we did not like the minimum security standards of generated passwords, nor was there any strict enforcement of strong master passwords.

Our password manager: PassMan

Github repository : PassMan

Abstract

PassMan is a simple PASSword MANager (PassMan) based on the success of commercial password managers in terms of security principles/protocols, technical design, and features. The graphical user interface (GUI) of PassMan has three unauthenticated and four authenticated operations. A user can create an account, sign in to an account, and sign out of an account using unauthenticated actions. Authenticated actions are built on the CRUD (create, read, update, delete) persistent storage paradigm and allow a user

to add, get, update, or delete a password. These operations work together to allow authenticated users to interact with a local MySQL database that stores data that has been sanitized and encrypted.

Introduction

Why do we need a Password Manager? As the world becomes more and more digital, users may find themselves integrating almost every aspect of their life into online services. Assuming that the user is following security practices and has a different password for each service, this leads to one user having many passwords - too many to remember.

This introduces the question: how and where should a user store their various passwords? This is where password management steps in.
A password manager is a platform that allows users to consolidate their passwords and securely store them. Password managers usually consist of an interface that allows users to add, delete, remove, and change their passwords, which are then encrypted and sent to a database, which are all protected through a master password set by the user.

Well-supported password managers often have a plethora of features and user-experience improvements like a copy-to-clipboard button, random password generator, multi-factor authentication, network-hosted databases, browser extensions, and form autofill.

In this project we plan to implement a graphical user interface (GUI) that offers three unauthenticated actions and four authenticated actions. **Unauthenticated actions** allow a user to create an account, sign in to an account, and sign out of an account. **Authenticated actions** are based on the persistent storage model of CRUD (create, read, update, delete) and allow a user to add, get, update, or delete a password. Together, these actions allow users to interface with a local database that stores data that has gone through some sort of encryption.

Encryption is a key detail in password management, as it is a procedure that obfuscates information from curious eyes. Encryption takes plaintext, such as the password "helloworld123," and encodes it as ciphertext. The **ciphertext** is a representation of plaintext that obscures all useful information from it, rendering it useless to anyone who gains access to it.

PassMan uses encryption when storing your passwords, ensuring that an attacker who gains access to them will have no sensitive information of yours. In addition to passwords, PassMan encrypts other sensitive information, such as derivation keys. Finally, this encrypted data needs to be stored somewhere and somehow. PassMan leverages a locally hosted MySQL database to store, retrieve, update, and delete all of this information. MySQL is a relational database management system, which means that the information is stored in tables that have relationships with one another.

Motivation and Current Systems

Apart from being central to daily life for many people, passwords are also often the main, if not only, point of authentication for many applications, services, and systems. However, many users do not take their password security seriously, often using short, easy to guess passwords that are often reused for many different sites and applications. Coupled with the fact that malicious actors have also evolved alongside this digital landscape, creating robust and secure passwords and password protection/management protocols have become increasingly important. And while multi-factor authentication is becoming increasingly common, it still does not make recycling passwords or using slight variations a safe venture. Many users often do not feel the repercussions of this until being on the tail-end of a security breach or an attack like the COMB (Compilation of Many Breaches) data breach that was leaked on a hacking forum, which is a well-organized and searchable database containing 3.2 billion unique (plaintext) email-password pairs for services such as Netflix, LinkedIn, and Bitcoin (COMB, 2021).

In order to maintain long, hard-to-guess passwords that are different for every service, therefore ensuring a high level of security, many users turn to password managers so they do not have to go through the hassle of remembering all of their passwords. There are many password managers on the market, all of which do the same central goal of maintaining your passwords but all are unique in how they operate and what they offer. For example, take two popular password managers: the Google Chrome password manager extension, and LastPass. Both of which are extensions that either come loaded into the Google Chrome web browser or that can be downloaded into the Google Chrome web browser. Both automate many of the steps that arise in the process of making and storing passwords, such as asking users if they want to save their password for a particular service, inputting their saved password into the login field, and even generating “suggested passwords” that are often long passwords comprised of numbers and letters with no distinguishable patterns or structure. Both also provide a user interface locked behind a master password, where the user can see and manage all of the services and passwords they have saved.

This knowledge, as well as our personal experience from using different password managers, led to us asking: can we build one ourselves? Our curiosity and desire to implement the numerous security concepts we studied throughout the semester and create some sort of digital tool motivated us to create PassMan.

PassMan

PassMan is a simple, open-source password manager that is modeled after the successes of commercial password managers, such as their security principles/protocols, engineering design, and features offered.

PassMan has the following actions for the users:

1. **Unauthenticated actions**
    1. Create Account
    2. Sign in to account
    3. Sign out of account
2. **Authenticated actions** (based on CRUD)

1. Add password
2. Get password
3. Update password
4. Delete password

GUI

To minimize time on non-essential features like the GUI, parsing inputs, and packaging, our group decided to leverage Gooey. Gooey is a Python library that creates a GUI for Python scripts with a few lines of code or less. Gooey attaches to our main function with a one-line decorator and some additional configuration options like a description, default window size, and disabling the restart button.

In the unabridged version of PassMan's code, we see that Gooey's ease of use comes with a cost. It does not scale well with complexity. Even with only seven distinct actions, PassMan quickly becomes a jumbled mess of conditionals that brings iterative development to a grinding halt. Nonetheless, since it automatically generates a working GUI with minimal configuration, Gooey remains an excellent option for prototyping and proofs-of-concept. However, it is not robust or powerful enough to develop highly scalable, user-friendly, and secure desktop applications.

Encryption details -

We have used cryptography and fernet python libraries to implement the encryption.

PassMan works by first generating a master key, also called a Key Encryption Key (KEK) from the user's master password. The user must remember this master password, it is not the responsibility of PassMan to store it. The key derived from this is then used in a key derivation function to derive the user table key - the key that will be used to encrypt all information stored in the user table. The user table stores a user's KEK, user table key, and a validator used for authentication. The validator is the user_table_key encrypted on itself. Once the user has been created, i.e they have generated a master key and user table key, they can begin adding service/password pairs to PassMan.The service, username, password, and encryption key are all stored in the user's service_table, where the password and key are each stored as encrypted information. The password is encrypted with the key, and the key is encrypted with the KEK.

Key generation in generate_master_key uses a constant time key derivation function to derive a key from the user-created master_password. This uses the SHA256 hashing algorithm to generate the key. Generate_user_table_key uses key-based key derivation, which derives a cryptographically secure key from another key. The master password must first be encoded into URL-safe base64 bytes before it can be used to derive the key.

Encryption in add_service took the naive approach offered by Fernet. It generates a key using the Fernet generate_key() function, which returns a cryptographically secure key, which is then used to encrypt the password for the new service. The KEK encrypts the generated key before it is sent to the database, where all of this information stays at rest

Authentication details

Our authentication checks the user-entered master password against a validator, and a user is authenticated if the function properly decrypts this validator.

First, the user must navigate to the login page, and enter their master password. This password is then passed to generate_master_key. If the user entered password is correct, generate_master_key should return the KEK we used to generate our user_table_key. We retrieve the user_table_key stored in the user table. We previously encrypted this key with the KEK, so if we have the correct KEK then user_table_key should decrypt correctly. Finally, we decrypt the validator with the user_table_key. The validator is the user_table_key encrypted on itself, so if we have the correct decrypted user_table_key, then the decrypted validator should be equal to the user_table_key. Thus, we have authentication.

Database details -

PassMan uses a MySQL database to store and manage passwords and sensitive information. In order to integrate MySQL into Python, we used the mysql-connector-python Python library to access and send information to and from our database. Upon first use, PassMan users are prompted to create a name and password for their database, which is then the one used for their instance of PassMan on their local computer. From there, users receive their own user table, which stores their username, encrypted user-table-key, encrypted key-encryption-key, encrypted validator, and encrypted service-key.

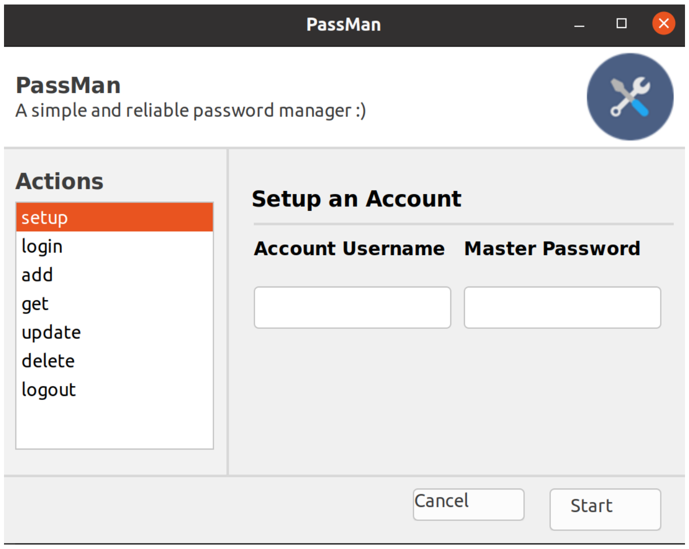
After the initial user_table is created users, once authenticated and after entering a master password, can begin to use the password manager like any other, adding, removing, and changing passwords. All of these services and passwords are stored in a services table, which holds their username, the name of the service, the encrypted password, and the encrypted encryption key. All of these functions require either retrieving or sending information to the user_table or the services table, which is done using the Python connector and sending queries to the tables. In order to send information, it became crucial that we convert our byte-values into strings to ensure that they are being stored properly. To retrieve information, it was also crucial to ensure that we not only converted our string back to bytes but converted it properly- this would ensure encryption was working properly.

When creating these predetermined queries that are needed for each function, our main concern was the threat of SQL injection attacks. In order to protect against these, we use prepared statements in Python to ensure our inputs are sanitized before being sent to the database. These work by taking in the parameters as the type we designate them to be (in this case strings) so that malicious inputs will no longer read as SQL statements but as string literals, ensuring a level of safety against injection attacks.
However, a security issue presented by our database schema is that all of our primary keys are of a predetermined length. This means that if someone inputs a username or service that is over 100 characters long, PassMan will crash and the user will not be able to either access their manager or add/delete/change

services. We would have liked to make these fields TEXT as opposed to VARCHAR(100) to combat this, but MySQL does not allow primary keys of undetermined lengths.

Screenshots

1. Setting up an account

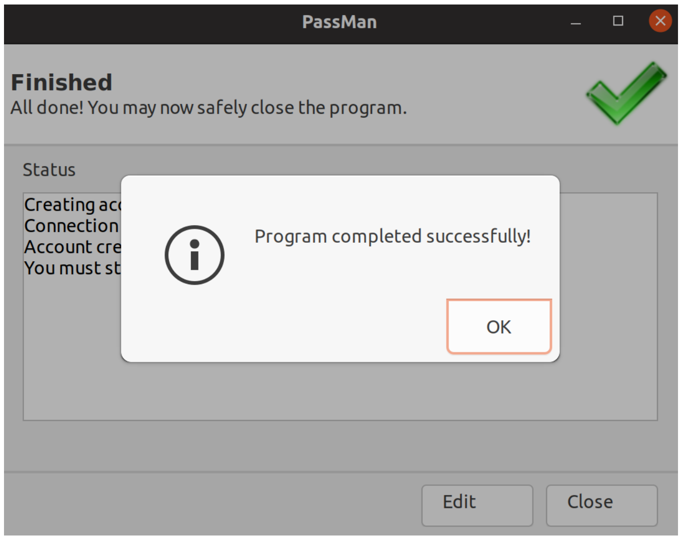


PassMan
A simple and reliable password manager :)

Setup an Account

Account Username Master Password

2. Successful account creation message:

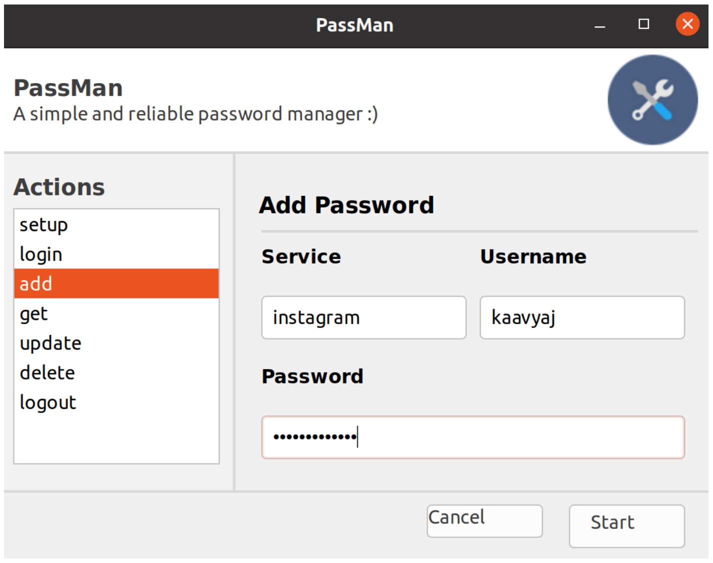


Finished
All done! You may now safely close the program.

Creating acc
Connection
Account cre
You must st

Program completed successfully!

3. Add a password for a service:

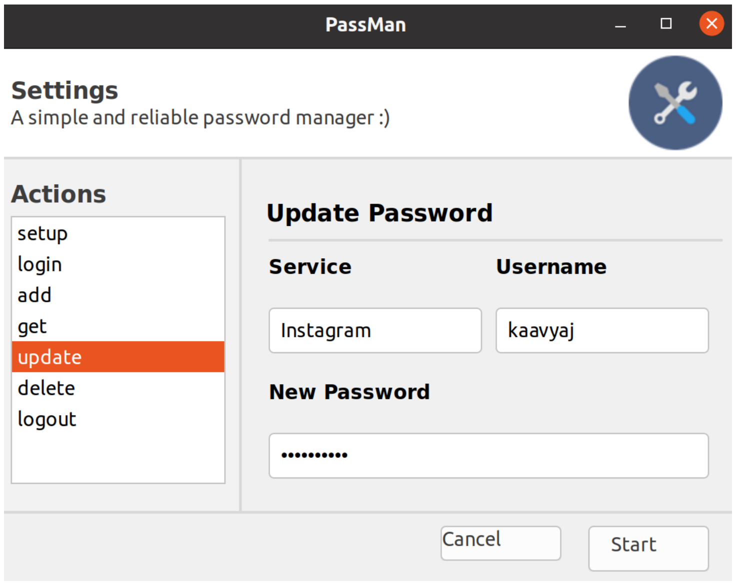


Service: instagram

Username: kaavyaj

Password: ••••••••••••

4. Update a saved password:



Service: Instagram

Username: kaavyaj

New Password: ••••••••••

CONCLUSION:

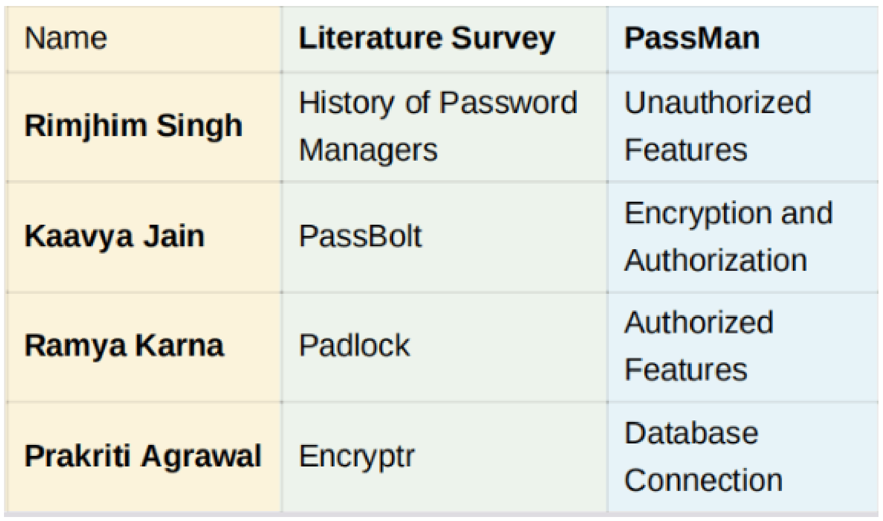
Firstly, the password manager would be open source to ensure that anyone who uses it can know what it is doing to protect their privacy. While a long password may be annoying to the user, the master password must be strong. If the master password is weak to brute force attacks, it brings down the security of the whole password manager. We would also suggest to add an auto-fill function to the password manager. This would prevent clipboard and keylogger attacks if implemented correctly. We would also lock the user's vault after a specified period of inactivity set by the user (no longer than two hours).

We would try to make every step as automated as possible to increase usability. This would include setting up a custom server, since all of the password managers we have tested had a very difficult setup process.

REFERENCES:

- Luevanos, Carlos & Elizarraras, John & Hirschi, Khai & Yeh, Jyh-haw. (2017). Analysis on the Security and Use of Password Managers. 17-24. 10.1109/PDCAT.2017.00013.
- https://cybernews.com/best-password-managers/how-do-password-managers-work/
- https://cryptography.io/en/latest/
- https://www.annalsofrscb.ro/index.php/journal/article/view/9761
- https://dev.to/anishde12020/cryptography-with-python-using-fernet-40o3

INDIVIDUAL CONTRIBUTION OF TEAM MEMBERS:



Name	**Literature Survey**	**PassMan**
Rimjhim Singh	History of Password Managers	Unauthorized Features
Kaavya Jain	PassBolt	Encryption and Authorization
Ramya Karna	Padlock	Authorized Features
Prakriti Agrawal	Encryptr	Database Connection