A. INTRODUCTION

1. Team Name
    - Ternary Crew
2. Team Member List
    - i. Christian Siador
    - ii. Karl Penuliar
    - iii. Kevin Quilantang
3. Application Title, Description and Functional Requirements Specification

    The application that we're developing is called PassCollector. This application is a simple password manager that allows multiple users to store and access their own passwords. As this application is desktop-based, all passwords are stored and encrypted locally on the user's machine.

    PassCollector should allow the user to be able to create and access passwords using the provided graphical user interface. The user interface has to be simple enough in design so that the user can easily use the application, while at the same time feature rich and secure.

4. Type of program e.g. Browser Extension, Web Application, ..etc
    - i. Desktop Application - this application is intended to run locally on the user's own machine.
5. Development Tools
    - i. Python (development language)
    - ii. Pycharm (IDE)
    - iii. Command Line

B. REQUIREMENTS

1. Security and Privacy Requirements:

   - Ensuring that the user authentication method is secure is a must. Specifically, the user's login information (master password) should be kept private and stored safely. The risk of exposing a user's login information to other people is substantial, as their login information could be used to access other important passwords stored in their profile.

   - The multiple passwords that are stored and associated with a user profile should be kept safe as well.

   - In consideration of the aforementioned points, PassCollector should be able to encrypt sensitive information like the master password and user data. Having some form of encryption mitigates security risks like data breaches.

   ○ In order to track security flaws that can potentially arise while developing our PassCollector, our development team will need to collaborate on a Github repository. This repository will contain a master branch, along with other supporting branches that each of us will be assigned to work on. Each supporting branch will represent a certain issue we have to work on or address. These issues can be created and kept track of within the "Issues" tab of our repository. Once these issues have been resolved, members can update the "Issues" tab and label them as "Done." Additionally, any pushed updates made to the supporting branches will be merged to a test branch, where our team can test for any bugs or security flaws before merging into the master branch, where the final version of our application will be.

Overall, adhering to certain, secure coding standards throughout the development of our application will ensure that the chances of security vulnerabilities, errors, or other factors which could possibly compromise our application and the sensitive data it stores are mitigated. In relation to PassCollector, adhering to secure coding standards means implementing previously mentioned features such as input validation, authentication and password management, cryptographic practices, and data protection.

2. Quality Gates (or Bug Bars):

There are four levels of privacy within the privacy bug bar - critical, important, moderate, and low. In our program, we'll discuss the End-User Scenarios and the Enterprise Administration Scenarios with different sets of privacy levels. In the End-User Scenario at the critical level where it may create a liability for the developers, one of them is the lack of notices and consent. The transfer of sensitive personal identifiable information such as email, name, and password from the user's system without prominent notices and opt-in consent in the UI prior to transfer may occur. Another critical level is the lack of child protection, the program does not collect age and/or verified with age restrictions towards any children for usage of this program. The last critical level is the lack of data protection as user's information data will be stored and collected in the database without any authentication for users to access and change its information data. Within the important level, there will be data minimization where a third party api will be used for the program. At the moderate level, lack of user control may occur where user's data is collected and stored locally as hidden metadata. It is accessible by others or transmitted if files or folders are shared.

3. Risk Assessment Plan for Security and Privacy:

To assess the security and privacy of PassCollector, the following questions may be asked:

- Are the passwords stored and encrypted properly?
- Are the passwords available only to the authorized user?

The main task of this program is to store multiple passwords from multiple sites that the user may use as their login credentials. The main security issue is ensuring that these passwords are protected at all costs, in which encrypting helps to do, in the case that the user's PassCollector gets hacked, encryption is another form of security that makes the stored passwords nearly impossible to identify. To access their list of stored passwords, the user will need to login to their PassCollector account to retrieve them, meaning each account will have its own set of passwords available only to the authorized user of that account.

Parts of our program that we may need to utilize a threat model for would possibly be encryption of the stored passwords, as this is another layer of security that ensures the necessary means of keeping our users secure.

C. DESIGN

1. Design Requirements:

In order to make sure that PassCollector meets the specified functional and security requirements, the following will be implemented:

a. User Profile

   i. Before being able to use PassCollector, the user will be required to create a username and master password

b. User Authentication

   i. Everytime the user opens PassCollector, they will be prompted to enter their master password, which will allow them to access the database and see their stored usernames and passwords for other sites/software (login credentials).

   ii. Check if the user profile is valid: if the user entered the wrong username or master password, return an error message.

c. Ability to Modify Information

   i. The user should be allowed to easily add new passwords, or delete and modify existing passwords in the database

   ii. The user can change their own username and/or master password

   iii. Performing these former actions may require the user to enter their master password for verification

d. Encryption

   i. All passwords created and submitted by the user are encrypted. This means that passwords in the database are not stored as plain text, and potential attackers will not be able to easily read or use them.

2. Attack Surface Analysis and Reduction:

There will be three levels of privilege in this application. The first level is the application is available to all who downloaded this application. Sign in for this application is not required where it is free to download on GitHub.

The second privilege is creating an account that requires a username and password to login. Email will not be collected where this application is only on the client side. Each user login will have their own individual information and data.

The last privilege is the user interface that contains other various passwords stored by each individual user login that is stored to the local computer. The UI will have features that allow access to create password information as well as changing the database entries.

This program includes:

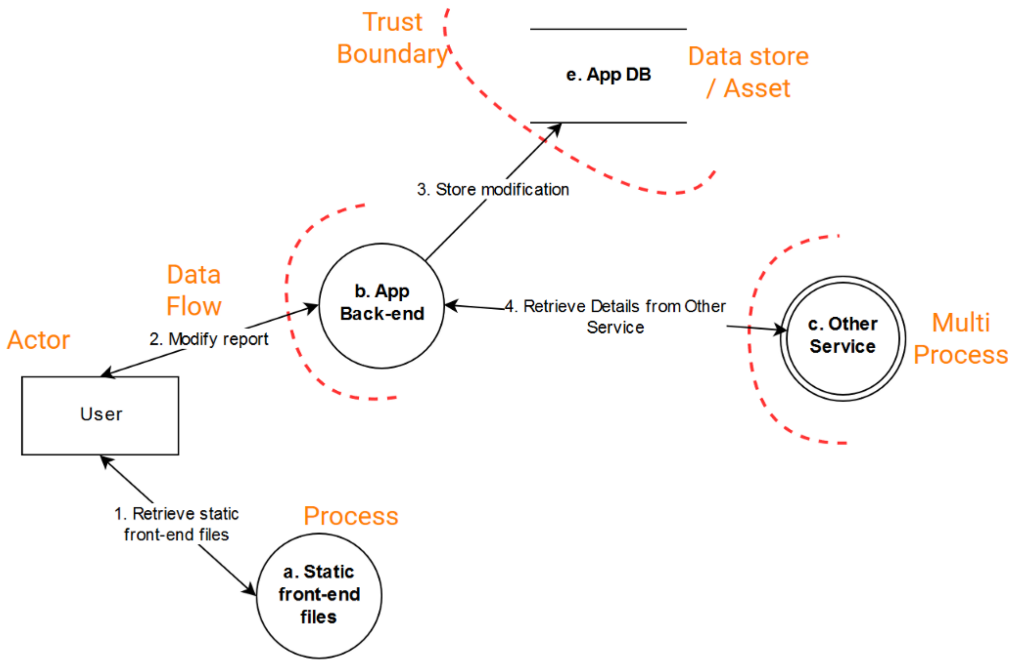
- User Interface
- APIs
- Files
- Databases
- Local Storage
- Run-Time arguments

The most common desktop application security vulnerabilities are:

- Using Components with Known Vulnerabilities

    - Usage of 3rd party vendors such as APIs may have vulnerabilities for any potential malicious intents.
- Sensitive Data Exposure
    - Data can be found on the client's local hard drive, meaning that it is exposed to any attackers with malicious intent.

3. <u>Threat Modeling:</u>



This flow diagram is a representation of PassCollector and shows how the different components of the program are connected to each other. The user modifies the app through storing their password credentials, and this data gets encrypted and stored into the App database of the authorized users' PC. The app will be connected to the services and sites that the stored passwords are for to allow the user to fill in the password automatically when visiting a certain site.

IMPLEMENTATION

1. Approved Tools

The following is a comprehensive list of tools that will be used for the project.

Tool/Framework	**Version**	**Additional Comments**
Cryptography	37.0.2	https://pypi.org/project/cryptography/ Cryptography is a Python library that aids in the encryption and decryption of data. It gives Python programmers cryptographic recipes.
CustomTkinter	4.3.0	https://github.com/TomSchimansky/CustomTkinter CustomTkinter is a Python GUI package that provides fresh, modern, and fully customisable widgets based on Tkinter. They are generated and used in the same way as other Tkinter widgets, and they can also be combined with regular Tkinter elements.
Git	2.36.1	https://github.com/git-guides/install-git Git is a distributed, open-source version control system (VCS) that lets you save code, track revision history, merge code changes, and rollback to a previous version of the code when necessary.
GitHub Desktop	3.0.1	https://desktop.github.com/ GitHub Desktop allows you to push to, pull from, and clone remote repositories, as well as use collaboration capabilities like crediting commits and generating pull

Tool/Framework	Version	Additional Comments
		requests.
Hashlib	20081119	https://docs.python.org/3/library/hashlib.html The Python hashlib module provides an easy way to hash messages. This includes a number of methods for hashing any raw message into an encrypted format. Additionally, FIPS secure hash algorithms such as SHA1, SHA224, SHA256, SHA384, SHA512, and RSA MD5 are available with hashlib.
Pillow	9.1.1	https://pillow.readthedocs.io/en/stable/ Pillow is a PIL (Python Image Library) that allows you to open, manipulate, and save images. This library supports a wide range of file formats, has a fast internal representation, and can perform pretty complex image processing.
PyCharm	2022.1.1	https://www.jetbrains.com/pycharm/ PyCharm is a JetBrains hybrid platform that serves as a Python IDE. It's a popular tool for creating Python applications.
SQLite	3.38.5	https://docs.python.org/3/library/sqlite3.html SQLite is a file-based SQL database that is self-contained. SQLite is included with Python and can be used in any Python application without the need for additional software.

Tool/Framework	**Version**	**Additional Comments**
Tkinter	8.6.12	https://docs.python.org/3/library/tkinter.html Tkinter is a common graphical user interface (GUI) package. Tkinter is Python's default GUI module and the most widely used method for GUI programming in Python.
Visual Studio Code	1.68	https://code.visualstudio.com/ Microsoft's Visual Studio Code (often known as VS Code) is a free open source text editor available for Windows, Linux, and macOS. Although VS Code is regarded as a relatively lightweight editor, it has a number of useful capabilities, like debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git.

2. Deprecated/Unsafe Functions :
    - Cryptography
        - MD-5()
            - A hash algorithm that has practical known collision attacks.
            - Alternatives:
                - Use SHA256() which is known to be one of the secured hashing functions.
        - SHA-1()

            - A hash algorithm that has practical known collision attacks.
            - Alternatives:
                - Use SHA256() which is known to be one of the secured hashing functions.
- Hashlib
    - Salted Hashing
        - Salted Hashing with other cryptographic hash, such as SHA-256 or BLAKE2, are not suitable for hashing passwords.
        - Example Function:
            - salt1 = os.urandom(blake2b.salt_size)
        - Alternatives:
            - Don’t use salted hashing by randomizing hashes that were created by other types of cryptographic hashes.
            - Use a cryptographic hashes function such as blake2b() instead.

3. <u>Static Analysis:</u>

For our project, the static analysis tool that we will be using is called Pyright, which was developed by Microsoft and is included as part of the Pylance extension for Visual Studio Code. Utilizing this tool will ensure that our development team follows general coding standards and also gains an understanding of the structure of our code. These general coding standards include language-specific rules regarding how to name functions and variables, as well as how to format source code. In similarity to other static analysis tools available, Pyright is useful in automating the process of checking for coding issues such as programming errors, syntax violations, and security vulnerabilities. Such issues are then highlighted to the developer, along with detailed suggestions on how to possibly resolve them.

After running the code through Pyright, the static code analyzer will determine whether or not the code follows the set of rules. It's probable that Pyright will signal false positives, therefore it remains important for our team to go through and dismiss any that come up. Once false positives have been eliminated, we can proceed to correct any obvious errors, usually starting with the most serious. Once the coding concerns have been rectified, the code can be executed for testing.

Even with the limited time spent with Pyright so far, it seems faster and more accurate compared to commonly used Python static analysis tools such as mypy, which would apparently sometimes generate unusual false positives and rarely catch errors. Additionally, Pyright is extremely adaptable in ways that work well with real-world Python programs. Specifically, Pyright, like other tools, can be set up per-project using a JSON-formatted configuration file in the project's directory. Individual routes can be ignored (errors and warnings are silenced) or excluded (never checked) in the config file, and the options are quite complex.

VERIFICATION & IMPLEMENTATION

1. Dynamic Analysis

Code profilers (also known as dynamic analysis tools) are generally useful for code optimization, as they examine how much memory, CPU, or other resources each software component or routine uses. During the development process, our team has researched which Python profilers are available and ultimately decided to use Scalene. Scalene is a high-performance CPU, GPU, and memory profiler developed by Emery Berger, a computer science professor at the University of Massachusetts Amherst. Professor Berger provides detailed documentation about Scalene's features in his Python Package Index page, in addition to instructions on how it can be installed. Scalene is available as a pip package for Mac OS X, Linux, and Windows (albeit with limitations such as lack of GPU profiling).

As Scalene is a sampling profiler, it profiles code by analyzing a running application on a regular basis to check what assembly instructions are currently being executed (the program counter) and which routines call the current function (the call stack). If a routine is too large, this indicates to our team that it is a potential performance bottleneck which we will need to address and optimize for better application performance.

Thus far, the benefits of using Scalene while developing our application is that it works relatively fast and efficiently because it just looks at the frequency of routine calls, which essentially means that it doesn't create any extra overhead or interfere with the application's performance while it's running. Scalene also does not change the source code in any way, so there's no risk of corruption or data loss. Feature-wise, Scalene provides our team many customization options such as the ability to toggle profiling on and off, and targeted profiling for specific lines, files, or functions. Being able to obtain information about which functions or lines of code exactly affect runtime has been particularly convenient. However, the most useful and powerful feature that we recently discovered is that Scalene also has a web-based GUI. Scalene will launch a web browser with an interactive user interface after it profiles our application and provide a visual representation of the amount of computer resources that are being used during runtime.

Overall, there have been no apparent difficulties while using Scalene besides the fact that it offers many powerful features beyond what has been previously mentioned that seem excessive given the rather non-complex design of our application. Nevertheless, throughout our continuous development Scalene has been a valuable tool for dynamic code analysis, and we intend to keep on using it accordingly when we further modify our code.

2. Attack Surface Review

Going over the developer tools that our team used, there were no changes, updates, or patches that were made so far. There were no new vulnerabilities reported as well. However, some of these tools that we used are still in development. Therefore, the tools we use for our project that are still being supported by other developers will be prone to updates, changes, and perhaps any future vulnerabilities during and after the development of our team's app.

VERIFICATION

1. Fuzz Testing

- Testing Master Password Incorrectly

The first attempt to break our app was to test the login functionality. We decided to test inserting the wrong password 20 times to see if the program will break or allow access to the password vault. After inserting the wrong password 20 times, the result came to a success in preventing the attacker from accessings the user's password vault. However, we noticed that after the second wrong password, there was an error that prevented the users from logging in with the right password after the 2nd or more wrong attempt. To prevent this from happening, we'll review our code and find the bug to fix this to prevent this outcome from happening.

- Password Reset/Recovery Key

  The implementation of giving the user a recovery key to allow a password reset did not go as planned. With the implementation of this feature, the database would break and not allow the user to add any entries. Although the feature of creating a random key that the user could copy to reset their master password worked properly, once the user attempts to reset their password, entering a new master password would cause the database to crash, and the only fix to this issue without removing it was using the same master password that the user originally created, which defeats the purpose of a password reset. In the end we decided we had to remove this feature to deter our program from crashing.

- Testing the Encryption of the Database

  To protect our master password, we used a SHA256 hashing function with salt. Additionally, information stored within the database such as the website, username, and password, are all encoded in the Base64 format. In order to verify that our database is secure, we first used a website called SQLite Viewer, where we can upload our database and view the contents of the sqlite file. Upon initial inspection of the contents of the file, it seems that our master password has been properly hashed. Proceeding to look at the collection of data within our database, we confirmed that the data was encoded properly as well.

  One discrepancy that we noticed was that if the user made their master password too simple, with no variation between each character, then it was easy to figure out what the master password was. We tested this by making a simple master password that contained all lowercase letters, then inputted that master password into an online SHA256 encrypt/decrypt tool. After running the online tool, it was able to decrypt the master password easily in less than a second. Consequently, we tried making a more complicated master password with some variations (such as the use of lowercase and uppercase letters, symbols, and random numbers) to see if the tool could also decrypt it. When we entered our complicated password and ran the tool, it turned out that the tool ran exponentially longer and seemed to have much more difficulty trying to decrypt our complicated password in comparison to our simple one. From our results overall, it seems that in order to alleviate such an issue we would have to require the user to enter a strong master password upon initial setup if they want to use the application.

2. Static Analysis Review

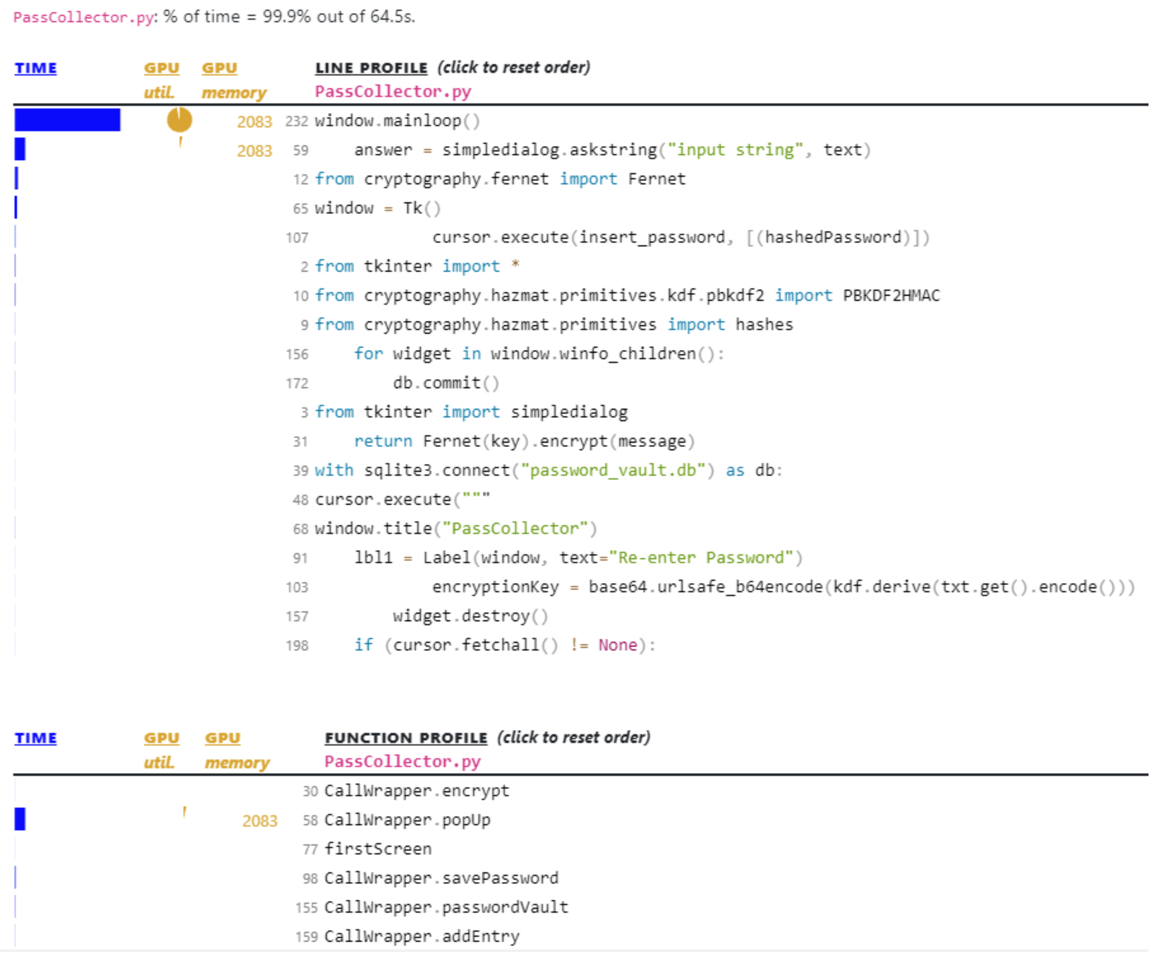
Since the start of our implementation of our project, Pyright has been a very efficient static analysis tool. Pyright's coding standards as well as its automatic error checking has made writing the code a lot more smoother than if we were not using it. The only downside to this static analysis tool as well as with a number of other tools is its tendency to detect false positive errors, in which we would need to manually get rid of. Overall, Pyright is still a fast and accurate tool compared to other Python static analysis tools.

3. Dynamic Review

After using Scalene for an extensive amount of time for dynamic code analysis, our team can affirm that it has been fairly successful in assessing overall application performance, as well as in providing in-depth information regarding the computer resources that have been particularly utilized during application runtime.

Of the many features that Scalene offered which were previously mentioned in the Dynamic Analysis section of our report, we unsurprisingly happened to use Scalene's web-based GUI feature the most after profiling our application. This interactive web GUI helped make the development process easier for our team, as it allowed us to visually see if there were any specific issues or flaws within our code that were overlooked during static analysis.

Fortunately, Scalene did not seem to find any significant errors in our code during application runtime, and the data it returned indicated to us that each of our functions work as intended and used the appropriate amount of resources. The screenshot below is an example of the visual data that is displayed after Scalene finishes profiling:



```
PassCollector.py: % of time = 99.9% out of 64.5s.

TIME    GPU    GPU          LINE PROFILE (click to reset order)
        util.  memory       PassCollector.py
                    2083 232 window.mainloop()
                    2083  59     answer = simpledialog.askstring("input string", text)
                          12 from cryptography.fernet import Fernet
                          65 window = Tk()
                         107             cursor.execute(insert_password, [(hashedPassword)])
                           2 from tkinter import *
                          10 from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
                           9 from cryptography.hazmat.primitives import hashes
                         156     for widget in window.winfo_children():
                         172         db.commit()
                           3 from tkinter import simpledialog
                          31     return Fernet(key).encrypt(message)
                          39 with sqlite3.connect("password_vault.db") as db:
                          48 cursor.execute("""
                          68 window.title("PassCollector")
                          91     lbl1 = Label(window, text="Re-enter Password")
                         103             encryptionKey = base64.urlsafe_b64encode(kdf.derive(txt.get().encode()))
                         157         widget.destroy()
                         198     if (cursor.fetchall() != None):

TIME    GPU    GPU          FUNCTION PROFILE (click to reset order)
        util.  memory       PassCollector.py
                          30 CallWrapper.encrypt
                    2083  58 CallWrapper.popUp
                          77 firstScreen
                          98 CallWrapper.savePassword
                         155 CallWrapper.passwordVault
                         159 CallWrapper.addEntry
```

From this data, it is apparent that there were certain functions which notably utilized the GPU the most and required more GPU resources compared to others, such as in line 232 of our code, where window.mainloop() was executing, and the simpledialog.askstring() function in line 59. Another function that we defined with the function name “popUp” also utilized a noticeable amount of GPU resources. It can be inferred that the reason why all of these aforementioned functions have substantial GPU utilization is because they are responsible for creating the graphical elements that are central to making our application interactive. Considering that our application has a graphical user interface as opposed to a text-based one, the resulting data made sense to us and was what we rather expected.

In summary, there were no complications while using Scalene to perform dynamic analysis. The ease of use and convenience that the tool provides enabled us to thoroughly debug our application and ensure that there were no lines of code that could have overall hindered its responsiveness and performance. Additionally, using a static analysis tool like Pyright in conjunction with Scalene has proven beneficial in improving the quality of our code and especially relieving our team from tedious, manual code analysis. Having both of these code analysis tools automate tasks that would have otherwise taken a surmountable amount of time for us to do meant that we were able to better shift our focus towards GUI adjustments and implementing the core features of our application.

RELEASE, ARCHIVE REPORT, USERS GUIDE

1. Incident Response Plan

Kevin - Escalation Manager:

The escalation manager is in charge of dealing with customer complaints or issues. They decide the prioritization of multiple customer concerns by ranking them based on the severity of the issue and ensure they are addressed by the right representatives.

Christian - Legal Representative:

Legal representatives will deal with the litigation in an investigation. They will need to maintain confidentiality, work with participating attorneys and clients and anticipate other legal risks or issues.

Karl - Public Relations Representative:

The public relations rep is in charge of shaping the image of our program through social media and advertising. They will create and maintain a positive image for our team and its individuals.

Karl - Security Engineer:

The Security Engineer will keep the security of our program up to date. It is their responsibility to implement new security features and respond to any security incidents that clients may have experienced.

For any inquiries or questions, please contact: PassCollector@gmail.com

Incident Response Procedure:

Purpose

The purpose of an Incident Response Procedure is to lay out a systematic process that can be utilized to resolve privacy escalation efficiently. This process will manage the internal and external communicating parties and determine a root cause of the incident to help improve company policies.

What is a Privacy Escalation

A privacy escalation is an internal process to communicate the details of a privacy-related incident. The following types of incidents may warrant a privacy escalation:

- Data breaches or theft
- Failure to meet communicated privacy standards and commitments
- Privacy-related lawsuits
- Privacy-related regulatory inquiries
- Contact from media outlets or a privacy advocacy organization regarding a privacy incident

Privacy Escalation Team

The core of a privacy escalation team will include an escalation manager, as well as a legal representative and a public relations representative. The roles of each job have been previously detailed above.

Submitting a Privacy Escalation Request

The privacy core team will have a group or managed email account that any employee can contact regarding a potential privacy escalation.

Procedure

An incident shall begin once an email notification of the issue has been received. It is the escalation manager's job to rank the priority of the issue to determine if it should be instantly investigated and resolved or whether more information is required to send the issue to the right representative. The escalation manager is also responsible for working with the reporting party and other contacts to determine:

- The source of the issue
- The impact the issue may have
- The validity of the incident
- A summary of known facts on the issue
- Timeline expectations
- Employees who know about the situation/incident, product or service.

After reviewing the information, the escalation manager will then ascribe the representatives responsible for resolving the current incident. The escalation manager may assign portions of the workload to different representatives if need be to ensure all aspects of the incident are resolved. The resolutions made shall be assessed by a privacy escalation core team in cooperation with the reporting party and participating contacts. Resolutions should include some or all of the following:

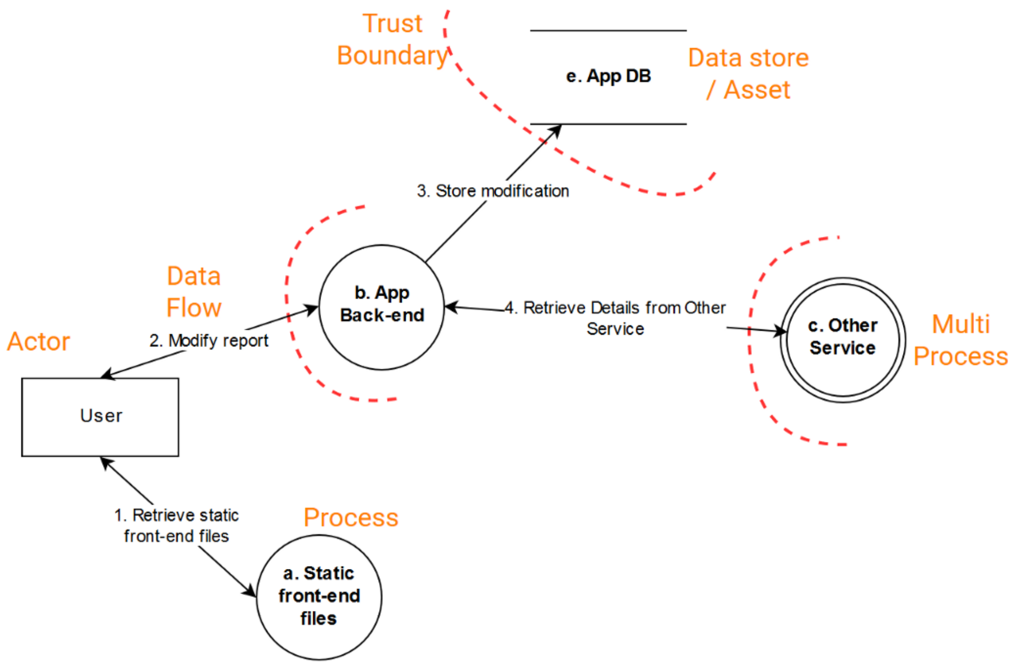
- Internal incident Management

- Communications and training
- Human Resources actions, in the case of a misuse of data
- External communications, such as:
    - Online help articles
    - Public relations outreach
    - Breach of contract
    - Documentation updates
    - Short-term and long-term product or service changes

2. <u>Final Security Review</u>

Prior to releasing our application, we performed a final review on our code. The following section contains our team's concluding thoughts regarding the threat model we have created, the quality gates we defined, and the static and dynamic analysis tools we used throughout application development.

- Threat Modeling



After revisiting our original data flow diagram and assessing how data flows between the user and the application, we have determined that there have not been any significant modifications to our threat model. Furthermore, the threats we identified have been satisfactorily handled.

- User Access

  There were various security issues to consider, such as how to authenticate the user and grant them the ability to modify the application and store their password credentials. These issues have been mitigated by implementing secure hash algorithms with salt to protect the user's master password. While master passwords that have been hashed are generally near impossible to decrypt, our team nevertheless recommends any user who uses our application to create strong master passwords.

- Application Back-end/Database

  Even though the user will primarily be interacting with our application using the provided graphical user interface (front-end), we still had to ensure that the back-end, the layer responsible for the application's core functionality and handling of user data, communicated well with the graphical user interface and all the other components of our application such as the database. To that end, our team performed a series of fuzzy tests to emulate how the user would possibly use our application. After conducting these tests and noting the vulnerabilities we found, we addressed each vulnerability accordingly and discussed workarounds or changes to the design of our application.

- Static and Dynamic Analysis

    Pyright remains a valuable static analysis tool that has been integral to debugging our code and highlighting any coding mistakes our team made throughout application development. Since our application utilizes various Python modules that are either built-in to the language or custom, Pyright has been particularly helpful in pointing out any unresolved references to missing modules, which was an indication that our team did not install a certain module correctly. Another specific case where Pyright was useful was in helping us manage variables. Considering that we decided to design our application with many GUI elements, we effectively had to define a lot of variables for those elements and keep track of each one. Occasionally, when there were any undefined variables in our code, or variables that we tried to use before assigning a value to it, Pyright successfully warned us about them.

    Similarly, Scalene has also been a convenient tool for us to use for dynamic analysis. While many coding errors were found and resolved early in the development life cycle thanks to Pyright, Scalene was still prominently used to detect any vulnerabilities in our code during application runtime which may not have been initially caught by Pyright. Despite Scalene not having found any particular vulnerabilities in our code to date, it still proved to be useful as a tool for evaluating the performance of our application and verifying its functionality.

- Quality Gates

    Throughout the development process, our team has fulfilled the privacy and security requirements that were established by each quality gate. Specifically, this means that:

- We have explicitly noted in our wiki page that our application does not automatically collect a user's personal information. Rather, it is up to the user how much information they want to store while using our application.
- Given that our team gains no specific benefit from collecting age data, we do not require the user to enter their age prior to using our application.
- Consequently, in addition to our application not collecting age data from our users, it also does not collect and store any hidden metadata from them.
- Our application does not disclose any sensitive information to third parties, nor does it have any third-party integration.
- Data encryption has been implemented for the user's master password and for sensitive user data stored within the database (website name, username, passwords).
- Only the user has knowledge of their own master password, and thus no other outside individual or party has access to the user's data.
- Even if the database file has been successfully transferred to another system, potential hackers will not be able to easily decipher the data stored in the file unless they have knowledge of the hash for the master password and the hashing algorithm used for said hash.

● FSR Grading

After careful deliberation, our team has reached the conclusion that we suitably achieved the security milestones that we defined for ourselves. This conclusion was mainly made on the basis of how secure the user's master password and stored data are. By performing various fuzz tests, we have proven that the hashing algorithm (SHA-256) and encryption format (Base64) used were sufficient in protecting the application database during most usage scenarios. Should any hacker attempt to steal the database file and decrypt it, the stored data will remain inaccessible to the hacker if they do not specifically know how the master password was hashed.

Besides the issue of securing the master password and stored user data, any other potential threats cited in our threat model were mitigated. Additionally, in utilizing the code analysis tools Pyright and Scalene, our team was able to mitigate vulnerabilities in our code and perform code cleanup before we released our application. With all that being said, we finally completed development of PassCollector and have decided to give it a "Passed FSR" grade.

3. Certified Release & Archive Report

- https://github.com/Ternary-Crew/PassCollector/releases
- The features using PassCollector have a vault that holds passwords that've been inserted by the users. The passwords are salted hash, therefore, making the password harder to crack.
- Technical Portion -
    - PassCollector is a password manager application that is based on the user's desktop. It allows users to store their password in a protected vault that stores data of their website, username, and password.

To install PassCollector, please visit the download page and download the zip file.

Requirements:

- Python 3.9 or Later.

- Python's pip installer

Dependencies:

- sqlite3
- hashlib
- tkinter
- functiontools
- trutle
- base64
- os
- cryptography
- customtkinter
- pillow

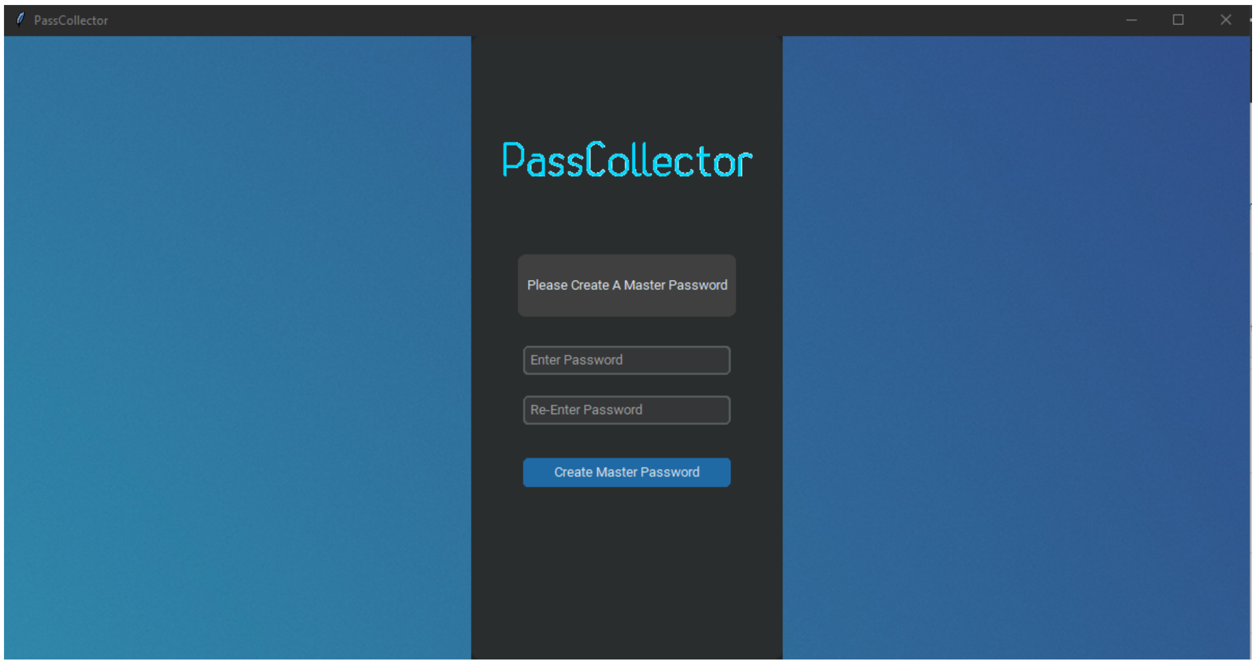
Program Setup

1. If Python 3.9 and/or pip installer haven't been downloaded, please install it.
2. Install each of the ten dependencies using pip.

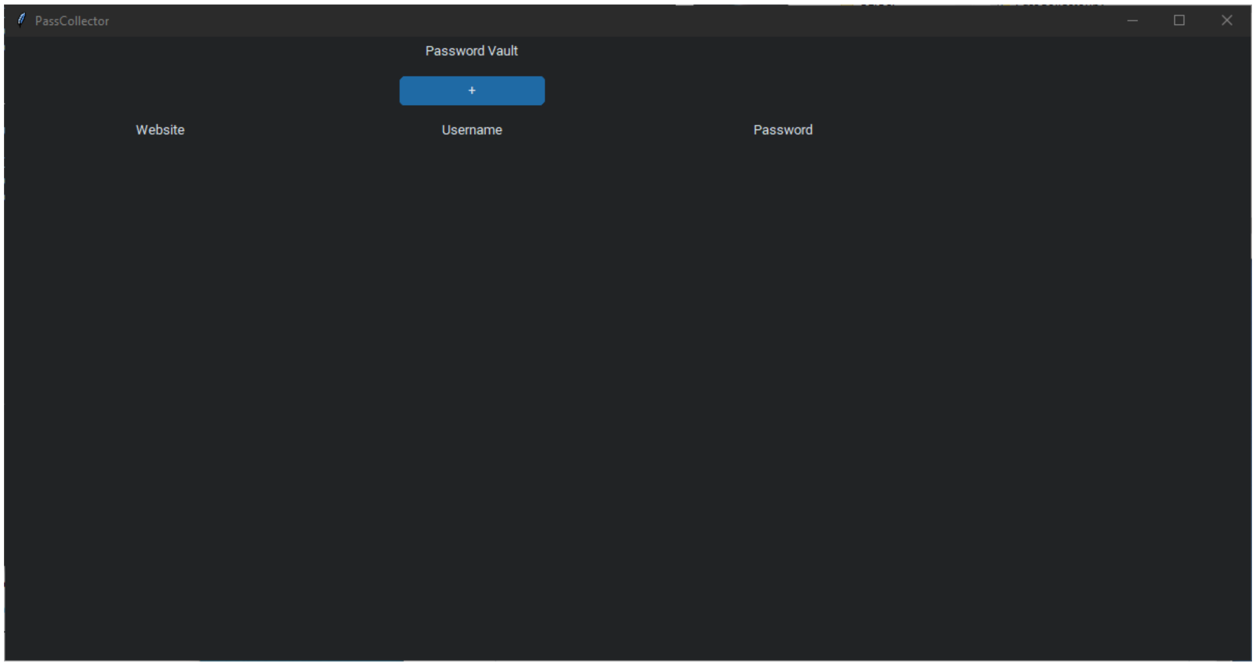
   Example: pip install sqlite3
3. Download the PassCollector from the download page.
4. Open PassCollector.py

How To Use PassCollector

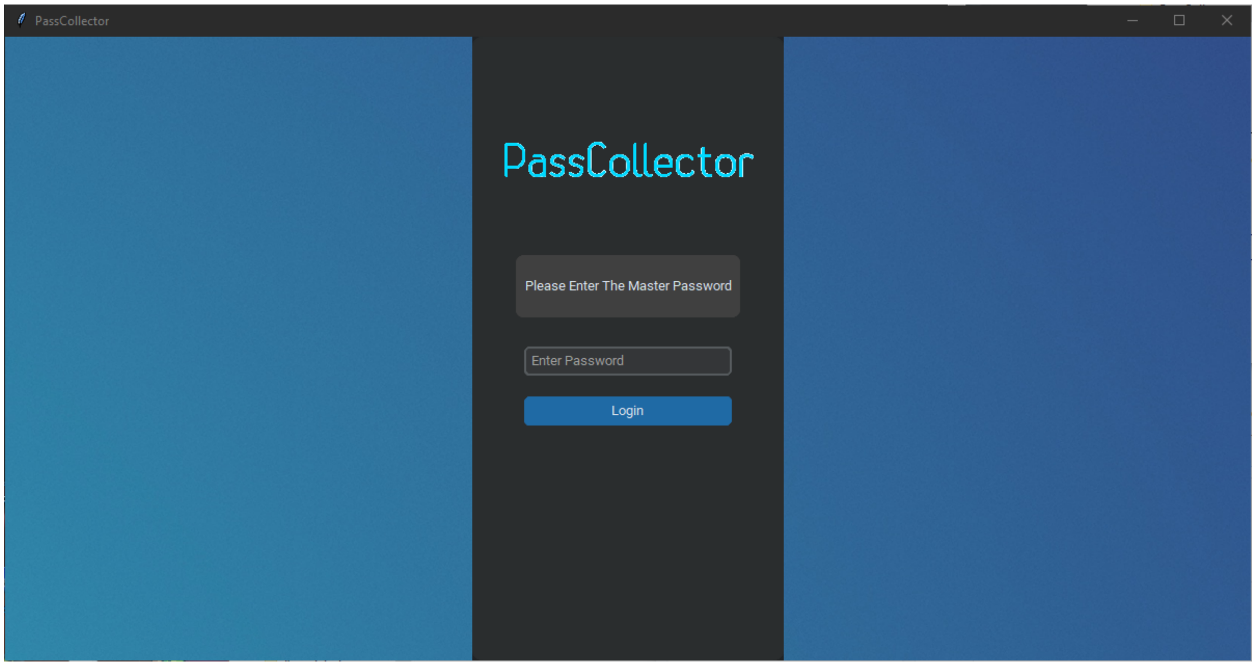
First, double click Passcollector.py You'll need to create a master password.



Once you created your master password, then you'll be able to enter the password vault.



After creating a master password, you'll have to login using the same password you created.



Security Notes

1. Deleting password_vault.db will lose all the data within that vault. If deleted, users must re-create a new empty vault.
2. Inserting a total of 2 or more passwords would result in preventing users from getting in despite inserting a correct password after 2 or more wrong passwords. If this happens then the users are forced to quit the application and open the program again.

Version

PassCollector 1.0 is the most current version of our application and is also the release version that users will be able to try. As we continue developing PassCollector and find vulnerabilities along the way, patches and new features may be implemented in the future.

Future Development

In future developments, our plan is to allow users to create multiple accounts for each individual vault. We’ll also work on the application’s design to a modern design.

Features

As of version 1.0, PassCollector includes multiple features like:

- An easy-to-navigate user interface
- The ability to store and modify account information for other services (website name, username, password)
- Encryption for the user’s master password and stored data
- Small installation files