**KeyGuardian Defense Report**

**A logo of a shield with a lock

Description automatically generated**

**Project Name:** KeyGuardian

**Date:** 04/09/2024

**Designed and developed by:** Logan Ferguson

**Major:** Cybersecurity

**Degree:** Bachelor of Science in Cybersecurity

**Academic Advisor:** Dr. Sean Hayes

Table of Contents

[Statement of Purpose: 4](#_Toc161649815)

[Research and Background: 4](#_Toc161649816)

[Project Language(s), Software, Hardware: 5](#_Toc161649817)

[Project Requirements: 5](#_Toc161649818)

[KeyGuardian 5](#_Toc161649819)

[Functional: 6](#_Toc161649820)

[Look and Feel: 10](#_Toc161649821)

[Usability: 12](#_Toc161649822)

[Performance: 15](#_Toc161649823)

[Maintainability and Support: 19](#_Toc161649824)

[Security: 20](#_Toc161649825)

[Cultural: 21](#_Toc161649826)

[Project Implementation Description and Explanation: 22](#_Toc161649827)

[Test Plan: 23](#_Toc161649828)

[Introduction: 23](#_Toc161649829)

[Test Items: 24](#_Toc161649830)

[Features to be Tested: 24](#_Toc161649831)

[Approach: 24](#_Toc161649832)

[Item Pass/Fail Criteria: 25](#_Toc161649833)

[KeyGuardian Test Cases: 25](#_Toc161649834)

[Suspension Criteria and Resumption Requirements: 33](#_Toc161649835)

[Test Environment: 34](#_Toc161649836)

[Schedule: 34](#_Toc161649837)

[Responsibilities: 35](#_Toc161649838)

[Risks: 35](#_Toc161649839)

[Assumptions and Dependencies: 35](#_Toc161649840)

[Crowd-Source Testing Script: 36](#_Toc161649841)

[Test Results: 38](#_Toc161649842)

[Challenges Overcome: 38](#_Toc161649843)

[Future Enhancements: 39](#_Toc161649844)

[Defense Presentation Slides: 39](#_Toc161649845)

**Statement of Purpose:**

The purpose of this project is to provide a solution to the problem of poor password management and to educate users on password best practices and security. Many users do not take the appropriate steps to secure their online accounts, even with simply using a more complex password, but with KeyGuardian, it will be easy for users to create secure passwords and check currently used passwords for security. Everyone should be able to create and store secure passwords for each online account they have and also be able to test their current passwords and evaluate their security. However, most people do not put much effort into creating secure passwords, but rather rely on easy-to-remember phrases that they do not have to write down, such as “Password123” or “qwerty”. Most are not well-educated on the importance of having a secure password for every account and do not realize that simple passwords like those can quickly be guessed or cracked and the user’s sensitive data can be stolen. Out of 56 million breached passwords in 2022, in over 110,000 of those cases “123456” was used. The second and third most used passwords were “12345” and “password” (Black, 2022). As a result, these are the passwords that threat actors will guess first, meaning that users should never use one of these simple phrases or string of numbers.

With a program that can generate secure passwords, store them, and check the security of current passwords, users can more easily safeguard their accounts without having to come up with a password and determine if it is safe to use. A user’s passwords provide the first line of defense to an online account, so if that password is long and complex, it makes it nearly impossible to guess or crack with brute force software. KeyGuardian will aid in protecting a user from hackers and make the internet a safer, more secure place.

Source: https://cybernews.com/security/weakest-passwords-2022/

# **Research and Background:**

Prior to formulating my problem statement, one of the huge issues I noticed in the field of cybersecurity was the vast amount of default credentials used and extremely simple passwords such as “12345” or “Password123”. This is the leading cause of account breaches and can allow threat actors to easily gain elevated privileges and even traverse organizations and networks to deploy malware and often hold data for ransom. I decided to create a solution that could help others learn about the dangers of weak passwords and to provide a way for them to easily generate secure passwords to use for their online accounts. I researched more into data breaches and came across a Cybernews article that mentioned a majority of passwords breached in 2022 were between 8 and 10 characters (Black, 2022). An 8 character password is no longer considered secure due to advanced password cracking algorithms. Therefore, I believe the new standard should be at least 12 characters. When it comes to passwords, complexity and length matter greatly and can be the difference between an account being breached or not. KeyGuardian is a solution that can help with these prominent issues in cybersecurity.

# **Project Language(s), Software, Hardware:**

KeyGuardian was developed in python and utilizes some built-in libraries to accomplish each of my goals and requirements. I coded this in Visual Studio Code on my MacBook, but the project can run on any system that has python installed. KeyGuardian does not require any specific hardware to run and is fully software based. The Python libraries I used are tkinter, customtkinter, bcrypt, cryptography, fernet, hashes, passlib, password\_strength, pyperclip, and pillow. These were all used to accomplish different tasks, some of which are listed in my requirements below.

# **Project Requirements:**

Below are project requirements for KeyGuardian:

# **KeyGuardian**

KeyGuardian is a password generator, checker, and manager of passwords which are stored securely on a local system. This software allows the user to choose the length of a password and to generate a secure password using upper and lower case letters, numbers, and symbols. KeyGuardian will check any current passwords the user may have to see if it is secure and whether or not it has been exposed in a previous data breach. The security of passwords will be checked and rated on a scale of 1 to 5, and a friendly message will prompt a description of the security scale. Lastly, the user will be able to store passwords in a local database that will be encrypted with a master key, which is generated by KeyGuardian. This will be the only password that users will need to keep in a safe place to unlock the rest of their passwords. The purpose of this software is not to just create, check, and store passwords securely, but to educate users on how to better protect their online accounts.

**Description**

This document will list and describe the Requirements for the KeyGuardian software and will represent a complete description of this project. Each of the requirements will include a requirement number and type, along with a description, rationale, fit criterion, priority, and the dependencies. The types of requirements that will be included are functional, look and feel, usability, performance, maintainability and support, security, and cultural. The priority scale is 1 to 10, with 1 being not a priority at all and 10 being of upmost importance for this software.

**Requirements**

## **Functional:**

Requirement ID: 01

Description:

The software must prompt the user to create a master password on its first execution.

Rationale:

This will allow the user to lock their stored passwords with one master password.

Fit Criterion:

Upon running for the first time, the software prompts the user to create a master password before moving to the next window.

Priority: 10

Dependencies: None

Requirement ID: 02

Description:

The user can choose between three options on the main window.

Rationale:

The software has three major options that the user can choose from.

Fit Criterion:

The three options are viewable via buttons in the main window.

Priority: 9

Dependencies: 09

Requirement ID: 03

Description:

The software allows the user to generate a secure password and prints it in the window.

Rationale:

This partially fulfils the purpose of this password generator.

Fit Criterion:

User can successfully generate and view the password to the meet desired security policies.

Priority: 9

Dependencies: 26

Requirement ID: 04

Description:

The software allows the user to input a password to check it for security.

Rationale:

Users can secure their online accounts by ensuring that their passwords are safe and secure.

Fit Criterion:

Software takes a string as input and tests it for security.

Priority: 9

Dependencies: 26

Requirement ID: 05

Description:

The software will rate each checked password on a security scale of 1 to 5.

Rationale:

The user must be able to visually see the scale of security for their passwords.

Fit Criterion:

A security scale will print to the window and show the user the results of the assessment.

Priority: 7

Dependencies: 26

Requirement ID: 06

Description:

Each password checked by the software will be cross referenced with a breached passwords list.

Rationale:

The user must be able to identify whether or not their password has been contained in a data breach in the past.

Fit Criterion:

The password is used to scan through the breached word list to search for a matching word and notifies the user accordingly.

Priority: 8

Dependencies: None

Requirement ID: 07

Description:

The software allows the user to securely store passwords for any online account.

Rationale:

This allows the user to keep their passwords organized and stored securely in one place.

Fit Criterion:

The users’ passwords are stored in a local file and the user can view them at any time.

Priority: 10

Dependencies: None

Requirement ID: 08

Description:

The software allows the user to add/remove a password at any time and the local file is updated accordingly.

Rationale:

The user can move all their passwords over to this software and remove them as necessary.

Fit Criterion:

The user can add and remove any passwords from the local file.

Priority: 9

Dependencies: 07

## **Look and Feel:**

Requirement ID: 09

Description:

This software will be equipped with a full graphical user interface (GUI).

Rationale:

Allows for the ease of usability for the user.

Fit Criterion:

All functions are inserted into one user interface allowing the user to navigate the

features with ease.

Priority: 10

Dependencies: None

Requirement ID: 10

Description:

The appearance of the GUI will automatically match the system chosen theme.

Rationale:

The user will not have to manually edit the appearance of the interface.

Fit Criterion:

The GUI will be either in light, dark, or auto mode depending on what the user has selected on their host computer.

Priority: 5

Dependencies: 09

Requirement ID: 11

Description:

This software will have separate windows for each option selected by the user.

Rationale:

This keeps the software organized and easy for the user to interact with.

Fit Criterion:

A new window is displayed for each option the user selects in the main window.

Priority: 7

Dependencies: 09, 02

## **Usability:**

Requirement ID: 12

Description:

This software will be multi-platform, meaning that it will be available on Windows, MacOS, and Linux based systems.

Rationale:

This makes it easy for the user to use this software on any system listed above.

Fit Criterion:

The software is executable on any version of Windows, MacOS, or any distribution of Linux.

Priority: 8

Dependencies: None

Requirement ID: 13

Description:

Each user can create their own personal database of passwords saved locally on their system.

Rationale:

The user will no longer need to write passwords down and wonder if they are secure or

not.

Fit Criterion:

The user can add and remove passwords to be stored in the local file.

Priority: 8

Dependencies: 07, 08

Requirement ID: 14

Description:

User friendly messages will be printed to the screen throughout the software that educate the user on best practices for password security.

Rationale:

Allows the user to learn about how to keep their online accounts secure.

Fit Criterion:

Periodic messages print informing the user on what makes a certain password secure or unsecure.

Priority: 5

Dependencies: None

Requirement ID: 15

Description:

This software is easy to use for anyone and guides the user through each step.

Rationale:

Anyone can use the software because of its simple yet functional design.

Fit Criterion:

Every option and window are clearly labeled on the interface and guides the user through each selection.

Priority: 7

Dependencies: 09, 11

Requirement ID: 16

Description:

Anyone is free to install this software from the GitHub repository and use it on their system.

Rationale:

The purpose of this software is to be usable to anyone and help them store their

passwords and learn about password security.

Fit Criterion:

The software can be downloaded from the repository and run on any system with the correct packages installed.

Priority: 9

Dependencies: None

## **Performance:**

Requirement ID: 17

Description:

The software should not have any latency noticeable to the user.

Rationale:

Allows the user to quickly generate, check, and store their passwords locally.

Fit Criterion:

The software is not connected to the internet in any way and is completely local to each system allowing for little to no latency.

Priority: 8

Dependencies: None

Requirement ID: 18

Description:

The software will scan through the previously breached passwords list as efficiently as possible.

Rationale:

The software must scan through millions of passwords in order to accurately determine if

the password has been exposed.

Fit Criterion:

Latency exists but will not be overly conspicuous to the user.

Priority: 7

Dependencies: 06, 17

Requirement ID: 19

Description:

The software will be designed to be as precise as possible when generating passwords, checking their security, and storing them efficiently.

Rationale:

This will ensure the software is reliable and a useful password manager for everyone.

Fit Criterion:

The software will validate a generated password to certify that it meets the security requirements, ensures the whole breached wordlist is scanned, and makes sure that a password the user wishes to store does not already exist in their password database.

Priority: 8

Dependencies: 26, 06

Requirement ID: 20

Description:

The software will check any user input for validity and inform the user of any incorrect input.

Rationale:

Allows the software to be robust and to handle any erroneous input.

Fit Criterion:

All input fields that require it will be validated to ensure no syntax errors are present.

Priority: 7

Dependencies: None

Requirement ID: 21

Description:

The amount of passwords stored will not have a max capacity and is only dependent on the storage of the local system.

Rationale:

Every password the user has for online accounts can be stored in this software.

Fit Criterion:

The software will allow the user to store as many passwords as they wish.

Priority: 7

Dependencies: 07

Requirement ID: 22

Description:

The software is designed in such a way that new features can be added with ease.

Rationale:

To allow the software to have extensibility and longevity.

Fit Criterion:

New features can be added to the software.

Priority: 8

Dependencies: None

## **Maintainability and Support:**

Requirement ID: 23

Description:

Any updates or bug fixes for the software will be added to the GitHub repository.

Rationale:

Allows the software to be properly maintained.

Fit Criterion:

Updates for the software and to the breached wordlist are available through the repository.

Priority: 5

Dependencies: None

## **Security:**

Requirement ID: 24

Description:

The software can only be accessed with the user created master password.

Rationale:

Allows for only the owner of the stored passwords to view them.

Fit Criterion:

The software is unlocked and the passwords are accessible by the user upon correct entry of the password.

Priority: 10

Dependencies: 01

Requirement ID: 25

Description:

The database file storing the user’s passwords will be encrypted and will only be decrypted when the software is unlocked.

Rationale:

Maintains the privacy and integrity of the software.

Fit Criterion:

The file is encrypted when the software is not running and is only decrypted when the user is signed in.

Priority: 10

Dependencies: 01, 24

Requirement ID: 26

Description:

The password complexity requirements will include a minimum length of 8 characters, at least 2 upper case letters, 2 numbers, 2 special characters, and at least 50 bits of entropy.

Rationale:

The software will use this policy to generate secure passwords and to check user

passwords for the level of security.

Fit Criterion:

A generated password will not be given to the user unless it meets these requirements and each password checked by the user will be rated using the security scale based on this policy.

Priority: 9

Dependencies: None

## **Cultural:**

Requirement ID: 27

Description:

The goal of this software is to focus on the quality of the product to reduce faults and increase user satisfaction.

Rationale:

Allowing the software to be reliable and enjoyed by the users.

Fit Criterion:

The software is refined and the updates and user support are readily available.

Priority: 8

Dependencies: 23

# **Project Implementation Description and Explanation:**

In order to implement my solution of KeyGuardian I used Python along with some open-source libraries to best accomplish my goals and meet the requirements that I have. I used tkinter and customtkinter to design a sufficient user interface and for encryption I used bcrypt, fernet, hashes, and cryptography. By default, Fernet uses 128-bit encryption keys, which is no longer considered secure, so to solve this I derived an encryption key from the user’s master password using SHA-256 through bcrypt and PBKDF2. After formatting the key correctly, I sent it to fernet and used it for encryption to make the password database have industry standard security. I was able to use pieces from all of these libraries and combine them to create KeyGuardian and make it as robust as possible.

For the password generator, I created an algorithm that randomly generates a string of characters, including letters, special symbols, and numbers, based on the length that the user chooses. KeyGuardian can generate passwords from 8 to 35 characters. Part of this algorithm checks the password for the bits of entropy which determines the randomness of the password. Each password is regenerated until it meets the password entropy requirements which is at least 85 bits, making the passwords secure and resilient to brute force attacks.

The breach checker tool built into KeyGuardian uses previously well-known breached password lists including the infamous rockyou data breach among others. The password will also be rated on a scale for security that has 5 levels. Level 1 being the weakest and 5 being the strongest password meeting all the password policies. It is important for users to understand that this breach checker does not have every breached password included so it does not always mean that a password has never been breached before.

The passwords are stored in a local database file that is fully encrypted. Users can add, remove, search for accounts, and display all their passwords and account usernames. All of these actions are implemented using SQLite in Python to modify and search through the database. As soon as a user signs in, the database file is decrypted and can be modified and when they logout, the file is fully encrypted again.

I found that Python along with tkinter has all the features that I require for implementation of my software. There were some challenges throughout the development of KeyGuardian, however the software is able to meet all of my requirements and go even further than that and will allow me to easily add more usability features in the future. The security considerations had the highest priority for me due to this software being a password manager and the great sensitivity of that information. Since security is of the upmost importance, I focused on the encryption algorithms and the password login functions. I used some of the most secure industry standard libraries including bcrypt and PBKDF2 to ensure the password database is protected. I also utilized predefined SQL queries and input validation to eliminate the possibility of command injection for the login page and the storage database.

I designed KeyGuardian purposely with the idea that it can easily be scalable in the future and extended to cover more features and offer more options to users. The performance of the software is something that I will continually be improving on while considering that some of the algorithms used, namely encryption, will naturally take the most time to execute. My goal is to keep KeyGuardian quick but also robust and secure.

I plan to continue maintaining KeyGuardian and to improve it in the future and make sure that my GitHub repository is up to date and has new releases or versions often. Any support or issues found by users can be reported to me through my email that is in my repository. Additionally, anyone is free to download my software and test it for security and usability, offering their opinions and feedback through my Google forms survey link.

# **Test Plan:**

Below is the test plan for KeyGuardian:

**Test Plan Identifier:**

I will organize each of my test cases by labeling each with “Case-01”, “Case-02”, etc. This will ensure that each case is organized and can easily be found or referenced in the document.

## **Introduction:**

This is my formal test plan document for KeyGuardian and will detail all the cases I plan to test my software with. This plan will include system-level testing and unit testing. All the features of this software will be tested for accuracy and efficiency and I am going to develop a survey for testers to provide some feedback for KeyGuardian, known as crowdsourced testing.

The goal of this test plan is to collect the necessary data from testers and ensure that any bugs are corrected and the overall software product can be improved. KeyGuardian is extendable software so that it can eventually allow the user to accomplish even more tasks with one central tool. It is important to test a product so that features are added and/or improved. My goal for KeyGuardian is to provide users a secure and reliable way to store and generate passwords without having to remember or write down all their passwords. This cannot be accomplished without robust and comprehensive testing.

A main constraint is that I am the sole developer of this project and therefore bugs may take more time to fix than if I had a development team. This software is also an early version and is subject to many changes in the future. Any libraries that are used are free and open source because I have chosen to not use paid software or libraries. Another constraint is the amount of risk for KeyGuardian. Since this is a password manager tool, there is a great risk involved due to the sensitivity of the data stored. However, the security will be tested extensively.

**References:**

Link for the project repository: <https://github.com/logon02/KeyGuardian>

Proposal document: [Proposal.md](https://github.com/logon02/KeyGuardian/blob/master/docs/Proposal.md)

Requirements document: [KeyGuardianRequirements.docx](https://github.com/logon02/KeyGuardian/blob/master/docs/Key%20Guardian%20Requirements%20-%20Ferguson.docx)

## **Test Items:**

* KeyGuardian v1.0

## **Features to be Tested:**

* Logging in
* Creating user: requirement ID #01
* Characters slider
* Generating passwords: requirement ID #03
* Password breach checker: requirement ID #04
* Security scale rating: requirement ID #05
* Storing passwords: requirement ID #07
* Add/remove passwords: requirement ID #08
* Search for password
* GUI matches system theme: requirement ID #10
* Works on any OS: requirement ID #12
* Logout button
* Input validation: requirement ID #20
* Access only with master password: requirement ID #24
* Encryption/decryption for passwords: requirement ID #25
* Password policy: requirement ID #26

**Features Not to be Tested:**

* User friendly messages: requirement ID #14
  + Reasoning: These messages are not part of the overall functionality of the software and are low priority.
* No max capacity for passwords database: requirement ID #21
  + Reasoning: There is no need to test this because the database file will always only be limited by the host system’s storage capacity

## **Approach:**

My approach to testing this software and its features is to utilize a master test plan which will test all levels of KeyGuardian and ensure that all the functionality included in the requirements document is complete. A majority of the testing will be manual, involving ad hoc testing and some black-box testing with use cases. I will also conduct some functional and usability testing by allowing some selected testers to use the software and provide their feedback and opinions on the software product. This kind of testing is better known as crowdsourced testing. I also plan to run some security testing mainly due to the sensitivity of any data that may be stored by KeyGuardian.

## **Item Pass/Fail Criteria:**

These are the primary questions I will use to determine whether or not a test is successful:

* + Is the feature/product functional?
  + Does it perform as expected or required?
  + Are there any errors?
  + Was the testing sufficient?
  + Is it user-friendly?
  + Are there any risks or concerns?

If there is something that still needs to be addressed after asking these questions, then the test case is a fail. The software must be corrected and the feature tested again asking the same questions. As long as there are no further issues and the questions have been addressed appropriately, then the test will be considered a pass.

### **KeyGuardian Test Cases:**

Case-01:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Creation of user and password | Check response on creating user | Username: testUser1  Password: Knc815&j(e | User Creation successful |

Case-02:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Login Functionality | Check response on leaving login fields blank | n/a | Gives error and login unsuccessful |

Case-03:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Login Functionality | Check response on incorrect login credentials | Username: John Doe  Password: password123  Username: Jane\_Doe  Password: NB\*95e}2h | Gives error and login unsuccessful |

Case-04:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Login Functionality | Check response on entering correct credentials | Username: testUser1  Password: Knc815&j(e | Login Successful |

Case-05:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Characters Slider | Check response to dragging slider | Number of characters:  8, 13, 20, 32 | The slider moves and displays the number of characters chosen |

Case-06:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Password Generation | Check response to choosing characters and generating password | Number of characters:  10, 15, 21, 35 | Generated password displays and is the number of characters chosen |

Case-07:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Copy Button | Check response when clicking the copy button | n/a | The button copies the current password to the clipboard |

Case-08:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Breach Checker | Check response when a password is searched | password123  N745grtIj@M  ilikepie  oMj|F"C28S$]  tkF$V?2QT;%8:,ZCyxb | Displays if the password has been leaked previously |

Case-09:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Security Scale | Check response when a password is entered | password123  N745grtIj@M  ilikepie  oMj|F"C28S$]  tkF$V?2QT;%8:,ZCyxb | Password is rated on a color-coded scale |

Case-10:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Adding Passwords | Check response when a password is added to the database | Website: Google  Username: jackSims  Password: u&8Q-phY2a  Website: Github  Username: codingMaster  Password: H$2f3i}VU- | Password is stored to the local database |

Case-11:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Removing Passwords | Check response when a password is removed | Remove:  Website: Google  Username: jackSims  Password: u&8Q-phY2a | Password is removed from the database |

Case-12:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Searching for Password | Check response when a password is searched | Search for: Github, Canva | Password is displayed when searched for |

Case-13:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test GUI Theme | Check response when system theme is light/dark mode | n/a | GUI matches system chosen theme |

Case-14:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test KeyGuardian on Windows, Linux, and MacOS | Check response when run on different OS | n/a | Software runs on each OS tested |

Case-15:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Logout Button | Check response when logout button is clicked | n/a | User is logged out and confirmation message displays |

Case-16:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Input Validation for All Fields | Check response when unexpected data is entered |  | Software catches any errors and continues when input is corrected |

Case-17:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Master Password | Check response for incorrect password and master password | Incorrect password: KeyGuardian123@nh | Access is granted only through master password |

Case-18:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Database Encryption | Check response when logged out for first time | n/a | Database is fully encrypted when the software is not logged in |

Case-19:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Database Decryption | Check response when user is logged in | n/a | Database is decrypted and passwords can be viewed |

Case-20:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Password Policy Minimum Length | Check required length of passwords | n/a | Minimum length is at least 8 characters |

Case-21:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Password Policy Uppercase Letters | Check required amount of uppercase letters | n/a | Minimum amount of uppercase letters is 2 |

Case-22:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Password Policy Numbers | Check required amount of numbers | n/a | Minimum amount of numbers is 2 |

Case-23:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Password Policy Special Characters | Check required amount of special characters | n/a | Minimum amount of special characters is 2 |

Case-24:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Test Case** | **Test Data** | **Expected Result** |
| Test Password Policy Bits of Entropy | Check desired bits of entropy | n/a | Desired amount of entropy is 85 bits |

## **Suspension Criteria and Resumption Requirements:**

There are no suspension criteria for the testing of this software due to continuous testing and ad hoc tests. Furthermore, participants in crowd-source testing are at any time completely free to stop testing and withdraw from the project.

**Test Deliverables:**

* KeyGuardian Test Plan
* Testing Report

## **Test Environment:**

The testing environment will be on my laptop with no network connection required. The purpose of the software is to be completely local to a system, so no internet will ever be required to use KeyGuardian. I am going to provide my laptop to each tester and allow them to use the software for a period of time and to provide their feedback.

**Estimate:**

There are no monetary costs involved for the development or testing of the project and every library/tool I am using is free and open source.

## **Schedule:**

Testing milestones:

* + Test Plan Completed: 11/20/2023
  + Ad hoc tests: 11/27/2023
  + Initial Soft Tests: 01/08 – 01/22/2024
  + Black-box Testing: 01/22/2024
  + Crowdsourcing Tests: 01/29 – 02/12/2024
  + Tester Feedback: 02/19
  + Bug Fixes Applied: NLT 03/11/2024
  + Functionality or Feature Updates: 04/08/2024
  + Final Test Report: 03/18/2024

## **Responsibilities:**

Developer & Project Manager (Myself):

* + - As the project manager, I am responsible for planning and organizing the testing of the project
    - I am responsible for all documentation
    - As the sole developer, I am also responsible for designing and implementing any software updates or features

## **Risks:**

* Myself as the sole developer and tester
* Changes in requirements or priority
* Changes in project scope
* Sensitivity of password management
* Collection of testing data
* Testing environment configuration

## **Assumptions and Dependencies:**

Software assumptions:

* I will have 8-12 testers
* Each tester will provide feedback
* The scope and project requirements will remain stable
* The testing environment (my laptop) will be available to all testers
* Each tester will create their own master password
* Encryption will provide data security

Software dependencies:

* Requirements documentation
* Complete software testing
* Fernet encryption library
* Customtkinter library for GUI
* PasswordPolicy library
* Sqlite3 library for database connections
* Pyperclip library
* Bcrypt for hashing and salting

## 

## **Crowd-Source Testing Script:**

**Introduction:**

Welcome! I am Logan Ferguson, a Cybersecurity student at Charleston Southern University, and I am running some tests for my senior project. I have developed a password manager and generator called KeyGuardian and am thrilled to offer you the chance to participate in testing my software!

**Overview:**

KeyGuardian is a password generator, checker, and manager of passwords which are stored securely on a local system. This software allows the user to choose the length of a password and to generate a secure password using upper and lower case letters, numbers, and symbols. KeyGuardian will check any current passwords the user may have to see if it is secure and whether or not it has been exposed in a previous data breach. The security of passwords will be checked and rated on a scale, and a friendly message will prompt a description of the security rating. Lastly, the user will be able to store passwords in a local database that will be encrypted with a master key, which is generated by KeyGuardian. This will be the only password that users will need to keep in a safe place to unlock the rest of their passwords. The purpose of this software is not to just create, check, and store passwords securely, but to educate users on how to better protect their online accounts.

**Testing Purpose:**

You may be thinking why I am asking for your help. Well, the reason is because my project has entered the critical stage of testing that requires user feedback. I need your help to test my software’s functionality, security, and its overall user-friendliness. Your feedback and opinions are important!

**Testing Scenario:**

Imagine you are trying to create a new Google account and you have just entered an email address and the site is now asking for a password. Instead of coming up with a password and remembering it or even writing it down, you have just installed a new password manager called KeyGuardian and you want to try it out. Create your KeyGuardian master password and use the software to generate a secure password for you new Google account.

You now can store this password and username and you only need one master password to access them! KeyGuardian can be used to generate and store passwords for every online account. It can also check your current or even new passwords you wish to use and determine if they have been exposed in a previous data leak.

Try generating a password and check if it has been breached. Come up with a random password that you think has been exposed before and try it. Generate and add any accounts you want to the database! Try searching for a password then try removing one. Also, try logging out and back in again. It would be helpful if you could test every feature you can to ensure my software is fully functional.

**Tester Feedback:**

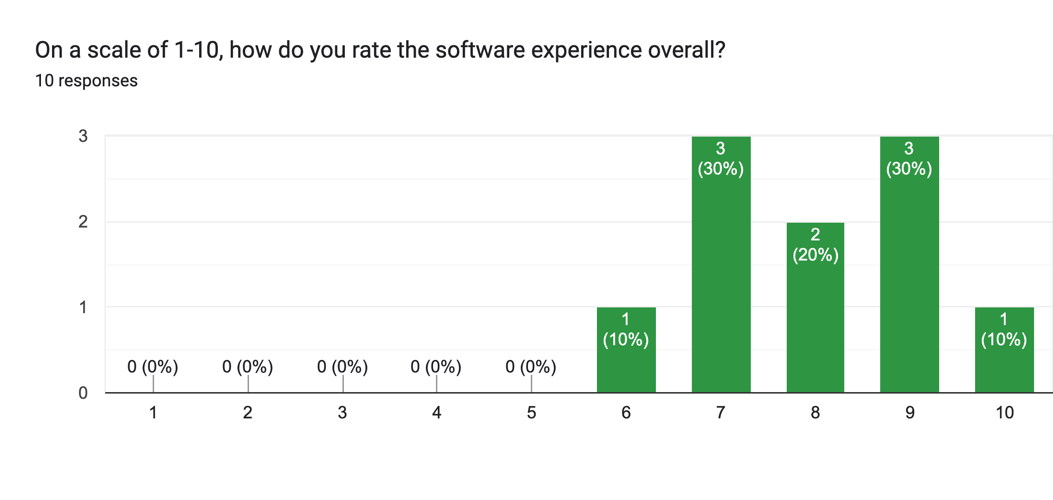
If you have any questions, feel free to let me know at any time during testing! To provide your feedback, please use the link below and answer the questions.

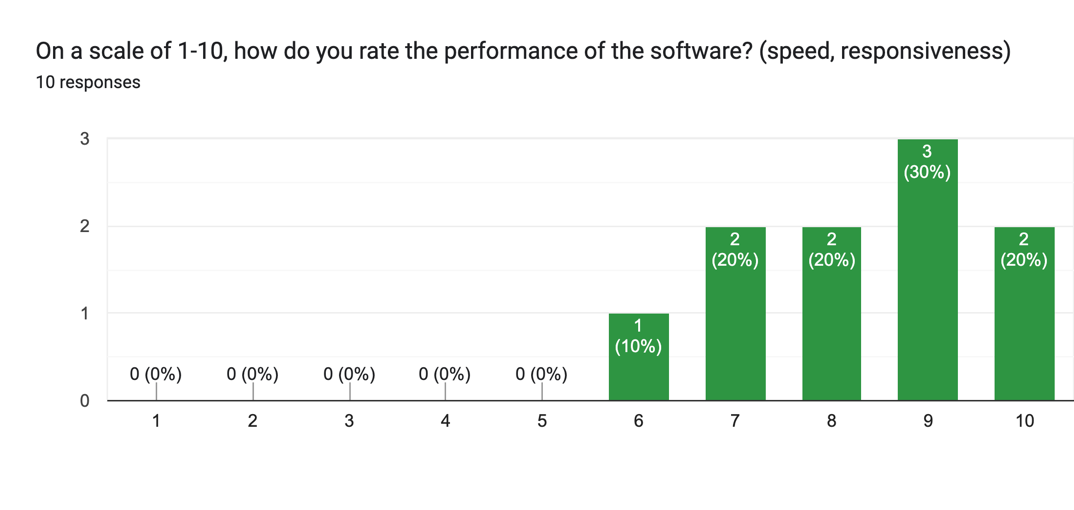
Feedback form: <https://forms.gle/d6vBi57NoqwHBSB67>

Thank you so much for participating in this test and contributing with your feedback!

# **Test Results:**

The testing process went smooth and I received some great feedback that helped me fix a few errors and get more ideas for future enhancements. I was able to correct a few smaller errors but there seemed to be no major flaws with KeyGuardian. The most common feedback from the testers was that it should have more help options to the users and show what each action does, such as a tooltip when hovering over the buttons. Some also said that the GUI could be rearranged to make it sleeker and the options clearer. There was also occasional lag when logging out. These are all things that I will take into consideration as I continue to develop KeyGuardian. I plan to continue the development and update it often, allowing for new releases on my GitHub repository. Below is some of that data from my testing feedback survey:





# **Challenges Overcome:**

* Learning customtkinter

For this implementation, I used an open-source library for python called customtkinter to design the User Interface. The learning curve for this library was immense and it was difficult to apply the vision that I had for a GUI using this tool. However, once I figured out how to use frames to pack the interface items together, I was able to create the interface that met my requirements and was satisfactory.

* Database issues

An issue I had toward the beginning was an interface glitch that caused the screen to flash whenever the passwords were displayed from the database. It took me some time to fix this issue, but once I tweaked the SQL queries and changed the way it reads from the database by using a temporary database file, it worked and the GUI no longer flashed when displaying the passwords. When displaying all the passwords from the database, it would sometimes show extra characters and whitespace that were not part of the data stored. I solved this by cleaning up the data and stripping extra characters or whitespace before it is printed to the screen.

* MacOS update

About halfway through the development of KeyGuardian, my laptop updated from MacOS 13 to MacOS 14. This immediately caused issues and customtkinter stopped working on my MacBook. To solve this problem, I used a VM and loaded MacOS 13 onto it and used this as my new development environment. This worked very well and I continued testing and developing KeyGuardian this way.

* Encryption using password as key

Another challenge I overcame was implementing secure encryption by deriving and encryption key from the user’s password. This was the most difficult aspect of KeyGuardian and also one of the most important. Encryption is vital to a password manager and is what makes it a usable platform to store passwords. I was able to use bcrypt, PBKDF2, and sha256 which are industry standard algorithms and libraries. I used these along with generated salt to derive an encryption key for the passwords database from the user’s password.

* Bcrypt and hashes library issues not installing

During the testing phase, I tried to install KeyGuardian on another machine for testing and ran into some issues with the packages not installing properly. I narrowed this problem down to the hashes and bcrypt libraries and I had to install older versions of these for KeyGuardian to work properly again.

# **Future Enhancements:**

* Create an app
* Add more user accounts
* Profile page to delete or add accounts or change password
* Help options
* Tooltips when hovering over a button
* More usability for displaying and editing passwords
* More efficient way to check for breached passwords
* Updated breached lists
* Password change reminders
* Check for duplicate passwords
* Interface customization (theme, scale, etc.)

# **Defense Presentation Slides:**

Presentation link:

<https://github.com/logon02/KeyGuardian/blob/master/docs/DefensePresentation.pptx>