# ‘Passman’ - A Secure Password Manager

Ewan Taylor, Liam Townsley, Owen Walker, Jack Barnett & Ryan van Ee

CMP315: Secure Software Engineering

**BSc Cybersecurity Year 3**

2023/24

.

# Abstract [Liam T]

This report documents the design and development of a password manager following Secure Software Engineering principles throughout the Software Development Lifecycle (SDLC). The project aimed to build a password manager to be resilient against a range of attack vectors. The development methodology followed was influenced strongly by OWASP's secure coding guidelines and incorporated AES-256 encryption and continuous testing.

This project was completed by developing a password manager which consisted of a Windows desktop application with a backend consisting of MongoDB and a REST API. Continuous security testing and an agile approach were used to prioritise the workload so that progress could be made where possible and vulnerabilities were patched as a matter of urgency. The main challenge encountered in the development process centred around testing, primarily due to the lack of engagement from team members, which restricted the deployment of broader testing protocols.

Suggestions for future improvements include modifying the tool into a browser extension to provide a wider scope of accessibility and compatibility. Overall, the project addresses the importance of incorporating security from the early stages of software development, particularly after the initial stable version of the project is available. As security is an area that is relevant to every aspect of the development lifecycle, there is a continuous justification to ensure that securing coding practices are always prioritised.

.

# Contents



.

.

# 1 INTRODUCTION

## 1.1 BACKGROUND [EWAN]

With a broad array of people's personal and working lives now dependent upon passwords to access online services through smartphones, banks, and computers, there is an ever-growing threat attackers pose to user's information. Despite predictions of their demise in 2004 (Kotadia, 2004), passwords remain the most common form of security measure, with many services specifying requirements for a suitable password (Davis *et al.*, 2022).

Effective password management is crucial for cybersecurity, especially considering the increasing level of threats mentioned by Kurtz, where he remarks how adversaries are using "more subtle and effective methods such as credential phishing, password spraying" to gain credentials (Kurtz, 2024). Furthermore, the rapid pace at which attacks can be conducted when an attacker has compromised credentials is displayed in Figure 1, highlights the significant harm that can be caused before users can take preventative measures. Maclean and Ophoff (2018) present another factor relevant to the importance of effective password management, stating that user passwords are "exceptionally lacking" with users reusing passwords and struggling to recall their passwords.

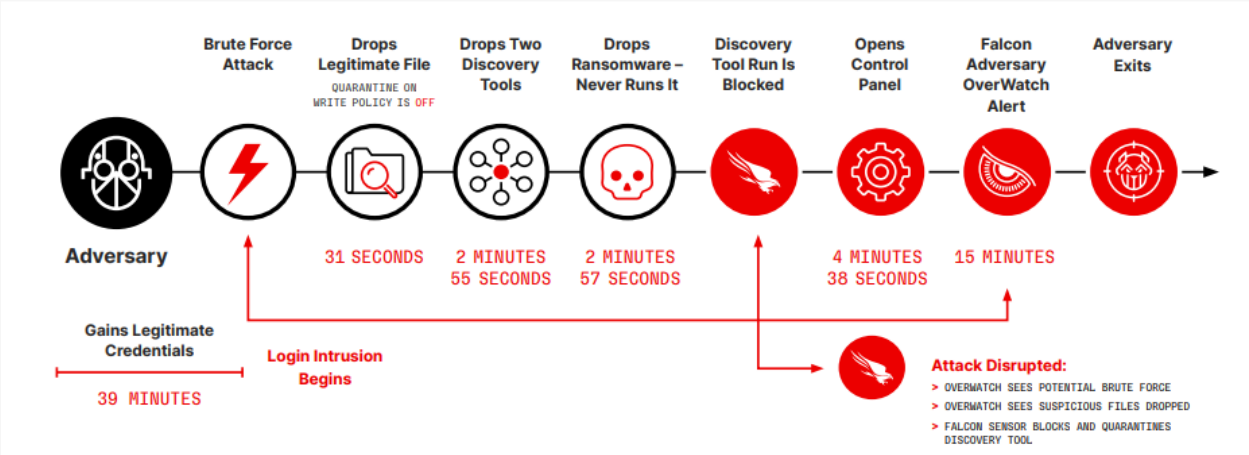


**Figure 1:** Real-world hands-on-attack timeline (Kurtz 2024, p. 11)

The traditional approach to password management, characterised by memorisation or simplistic storage methods, has proven to be inadequate in the face of evolving cyber threats. Password managers provide a solution to this, as they software applications that offer a method that generates complex passwords whilst encrypting and storing passwords for multiple accounts (Padalia *et al.*, 2023). As human entropy is the predominant cause of weak, guessable, and reused passwords, password managers assist in eradicating the vulnerable habits users can display. Considering that the number of user accounts and passwords is growing exponentially (Maclean and Ophoff, 2018 p. 6), passwords will continue to be a key linchpin of every user's security.

A password manager is not a perfect solution, as some users find them challenging to use and it can only be a secure method of storing passwords if the application itself is secure and resistant to cyber-attacks. Moreover, despite the user only being required to remember one 'master' password, should this be breached by a cybercriminal they will obtain access to all your accounts (NCSC, 2018). A password manager is ultimately as secure as the platform it is built on and how the user accesses it.

This highlights the importance of ensuring software is developed with secure software principles. Mead et al. (2016, p.28) share this opinion, remarking that defective software can be exploited and that it is unacceptable to fix and find fails "after the product has been delivered. The global impact of insecure code has highlighted the critical importance of software resilience and security for businesses and organisations, through recent notable incidents, including the SolarWinds attack and the Log4j vulnerability (Lopez, 2023).

A solution to patching insecure code after deployment is to develop software with secure engineering principles from the beginning of the Software Development Lifecycle. This is reinforced by Khan *et al.*, (2021, p.19157), who state "incorporating software protection is now a primary need for secure software development". This involves regular automated and manual testing, code reviews, pair programming and considering the security flaws and implications for the architecture and programming languages to strengthen a software's resilience to attacks. The benefits of designing software in this manner influenced the aims of this project outlined in the next section.

## 1.2 AIMS [EWAN]

The project brief focused on designing a security-oriented project that required the utilisation of the appropriate Secure Software Engineering practices throughout the Software Development Lifecycle.

To satisfy this brief, the following aims were established:

- Develop a password manager with appropriate security mitigations and defence-in-depth, to be resilient against a range of attack vectors.
- Plan and implement secure software engineering tools to ensure a secure password manager is developed.
- Create a report documenting the technical work undertaken throughout the Software Development Lifecycle.

# 2 METHODOLOGY

## 2.1 DEVELOPMENT METHODOLOGY [OWEN]

During the development of the application, an initial priority was to follow an agreed methodology to build a secure application. The OWASP guidance on secure coding practices (OWASP, 2023) shaped our methodology and allowed for the splitting up of the project into eight phases, to ensure that software development is as secure and efficient as possible:

- Initiation
    - The scope and the objectives of the project are identified, and the security requirements and any constraints are identified in this stage.
- Planning
    - Create a comprehensive plan for the entire software development process and includes doing a risk assessment, threat modelling and designing a secure architecture.
- Requirements
    - Any functional and non-functional requirements are identified, and any security requirements are documented.
- Design
    - In this phase security controls and mechanisms are incorporated into the design and architecture of the software, this can include security measures such as encryption, authentication, access controls and secure communication protocols.
- Implementation
    - In this phase you use the plans and the design that were created in the earlier stages to implement your software with a focus on securing the implementation of the security controls identified in the earlier stages.
- Verification
    - Steps such as code reviews, static analysis and security testing are some of the steps that can be taken to verify that any security controls implemented into the software are free from vulnerabilities.
- Release
    - Prior to the release of the program a final code review is conducted to confirm that all security matters have been met this can involve penetration testing and vulnerability scanning.
- Operation and maintenance
    - In this final stage the developers perform maintenance such as, ongoing monitoring, patching, and updates to address emerging security threats and vulnerabilities.

By ensuring that security is of the utmost importance throughout the entire development process it makes sure that security is built into each crevasse of the program to make sure that it is all as secure as possible reducing the risk of any breaches and protecting users' sensitive data ensuring that the client is protected.

## 2.2 SECURE SOFTWARE PRACTICES [OWEN]

Fundamental to the development of a secure application is the assurance that those with coding responsibilities adhere to best practices. This is important to help ensure the security of the application and consistency across all developers, best practices taken from Vermeer, B. and Gee, T. (2020) were used, and listed in Table 1.

**Table 1:** Eight Security Code Review Best Practices – Snyk (Vermeer and Gee, 2020)

| |
|---|
| Sanitize and validate all inputs. |
| Never store credentials as code/config |
| Enforce the least privilege principle. |
| Enforce secure authentication. |
| Test for new vulnerabilities in your OS app dependencies. |
| Handle sensitive data with care. |
| Protect against well-known attacks. |
| Statically test your source code |

Throughout the project these best practices assisted in the development of a secure application.

## 2.3 TOOLS USED [OWEN]

The development of a secure application was reliant upon a range of tools that were accessible to the entire group. For the development of the password manager tools to support front-end development using C# and backend development using Python the following tools were selected to aid the secure software development process.

- Visual Studio Code
- Visual Studio
- GitHub
- MongoDB
- NodeJS
    - bcrypt – used for encryption.
- Python
    - Flask
- C#
- Snyk

## 2.4 ROLES AND RESPONSIBILITIES

| Role | Responsibilities | Name |
|---|---|---|
| Project manager | Oversees project schedules, outputs, and dialogue with stakeholders. Produces and updates project records for clear and open access to project details and progress. | Ewan Taylor |
| Front-end developer | Crafts and refines the user interface with C#, ensuring it's user-friendly, secure, and visually appealing. | Owen Walker & Liam Townsley |
| Back-end developer | Constructs the application's server-side framework, centering on server, logic, and database operations using Python. Guarantees safe data management and streamlined server-side functionalities. | Liam Townsley |
| Security & Quality Assurance Engineer (Front end) | Perform a range of testing on the project's front-end interface layer, focusing on verifying the functionality of GUI components. Collaborate closely with Front-End Developer to ensure user interactions align with expected behaviours | Jack Barnet |
| Security & Quality Assurance Engineer (back end) | Implements security measures and appropriate testing to identify faults. Confirms the application functions as desired and is resilient to attacks. | Jack Barnet |
| Database & Encryption Specialist | Manages database architecture with an emphasis on data security through encryption. Works in tandem with the back-end developer to protect and secure user information. | Owen Walker & Liam Townsley |

# 3 RESULTS

## 3.1 RESULTS OVERVIEW

The development of a project where user's sensitive information and credentials are being handled emphasised the critical need for security features that were robust and compliant with data protection laws.

The project aimed to provide users with a reliable and secure tool for generating, storing and accessing their credentials whenever requested. Security was one of the primary concerns throughout the design and implementation of the entire project. However, during our evaluation, a small number of vulnerabilities were identified. This process was carried out and fixed by our security tester who would relay information to the developers through code reviews and raising issues through GitHub.

## 3.2 IMPLEMENTED SECURITY FEATURES [OWEN]

To ensure users' confidence when using the password manger, a range of security features have been implemented. As its core the project has carefully and correctly used encryption, coupled with secure data handling.

### 3.2.1 AES-256 Encryption

All users' passwords and sensitive notes are encrypted using the AES-256 encryption. AES-256 formally known as Advanced Encryption Standard with a 256-bit key, is a common encryption algorithm it works by breaking the data into 128-bit blocks and expanding into round keys which then does 14 separate rounds of encryption which involves operations which as "Sub Bytes" and "Mix Columns". This is currently the most secure form of encryption possible.

The reason AES encryption was selected for our password manager is because of its reversibility, which allows for quick data retrieval. Unlike hashing as seen in password storage, where the process is one-way and irreversible. AES encryption allows for decryption of data when required. Hashing is computationally expensive to reverse, this makes our choice of AES more practical for this project and secure and efficient for data storage.

### 3.2.2 Secure Data Movement

The data movement between the desktop application, REST API, and MongoDB was designed with a focus on security and reliability. During testing no vulnerabilities were found in this critical component of the system.

The process begins with the desktop application securely working alongside and communicating with the rest API using the C# programming language, this helps ensuring the data sent between the desktop application and API remains confidential and secure.

Our API, developed in Python by our developers, works along with MongoDB for data storage. MongoDB was chosen for its secure management and robust features; it helps provide a reliable platform which is one of the main parts of the project. By securely centralising data management with the API, this enhanced the security of data transactions.

This approach provides confidence for our users and stakeholders. Giving them the belief and confidence that our data is handled with care and correctly to follow data protection laws.

## 3.3 JWS AUTHENTICATION [RYAN]

Within our application, JSON Web Signature (JWS) authentication was utilised which is the standard for securely signing data. This will allow the user to continue their session within the application, so they don't have to sign in every time they close the password manager app.

### 3.3.1 Signing

When Using JWS authentication the issuer creates a JSON object which will contain data, which will be transmitted securely. The data itself will store the user's data and the permissions they possess within their account. The issuer creates a digital signature of the JSON object though the use of a cryptographic algorithm.

### 3.3.2 JWT & Structure

The signed data is structured as a JSON Web Token (JWT), which consists of three parts: the header, the payload, and the signature.

- The header contains metadata about the token, such as the type of token and the cryptographic algorithm used.

- The payload contains the actual data being transmitted, often referred to as claims. These claims can include information about the user and any additional metadata.

- The signature is a cryptographic algorithm of the header and payload, signed using a secret key known only to the sender.

### 3.3.3 Transmission

Once the JWT is created, it will be transmitted over the network, within the request body. This will be done through HTTPS.

### 3.3.4 Verification

On the receiving end, the verifier receives the JWT. They then parse the JWT to extract the header, payload, and signature.

The verifier recalculates the signature using the same algorithm and secret key used by the issuer. If the recalculated signature matches the signature in the JWT, it means that the data has not been tampered with and originated from the expected sender.

After verifying the signature, the verifier can examine the payload to extract the information contained within it. This information will be used for authentication and authorisation.

### 3.3.5 Implementation

Our coders implemented the JWS authentication by using JWTs, which is handled within our API script located in "api/src/services/jwt_utils.py". The implementation of RS256 was agreed as a team, which is an asymmetric cryptographic algorithm. It was chosen as it offers a high level of security when compared to other algorithms. Within our token's payload it contains the user's ID and session ID. Then the token is created and encoded with the payload, private key and the cryptographic algorithm used (RS256) inside it. The last step is to decode the token using the public key and then validate the user. It is important to note that the token will not always be valid and will be rejected once it has expired.

## 3.4 System Architecture Overview [Ryan]

The overall architecture of the system is comprised of three processes, each serving a distinct function to ensure the application's functionality and security. They go as follows: Client Interface, Application Layer (API) and Database Layer.

The Client Interface (Windows Desktop App) is responsible for the front end of the application, it was programmed using C# in Visual Studio (VS) with the VS Designer influential in the development of the client interface.

The API plays a core part to the integrity of the project as it is responsible for the backend of the application. Moreover, the API manages all the data that the client interacts with facilitates a connection with the Database and ensures that the Desktop Application can request and send the required data.

The Database Layer is responsible for storing the data that the system may require, this layer is inaccessible from the Desktop Application and can only be accessed through the API. The Database Layer stores and communicates Master Account User Details (within the 'users' collection), Saved Passwords (within the 'passwords' collection), and Saved Notes (within the 'notes' collection) with the API.

### 3.4.1 System Architecture Diagram

Primarily, the client interface will send requests to the API, which is responsible for establishing a secure connection to the database and ensuring that all data transmitted and stored is encrypted. The API then checks if the client possesses a valid session token, the absence of which will result in the user being prompted to log in. Without a valid session token, the authentication process involves verifying user credentials and issuing a new session token upon successful login.

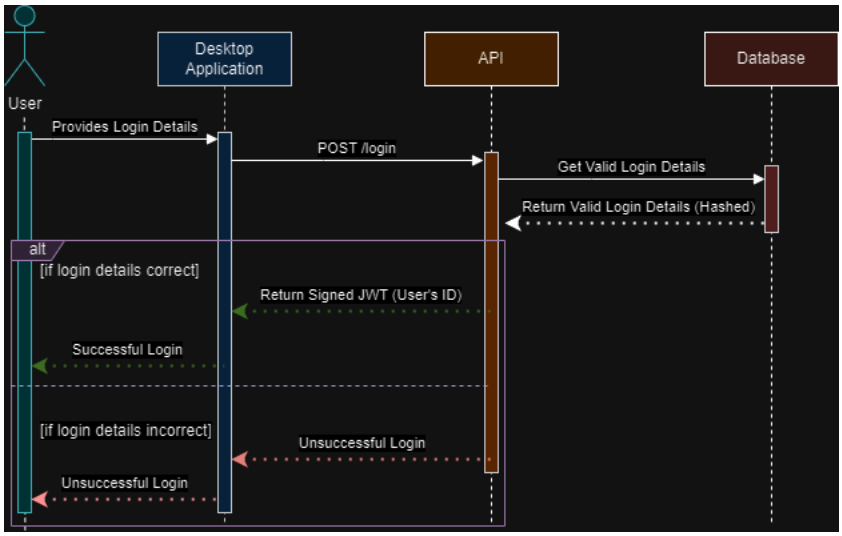


**Figure 2:** Sequence Diagram for user logging in

Upon successful user authentication, the API handles execution of database queries sent from the Client Application. This includes retrieving or storing data such as passwords and notes. All data retrieved is displayed on the user's interface, and any data stored is encrypted using AES 256 encryption. This encryption process is managed by the API, ensuring robust security for sensitive information.

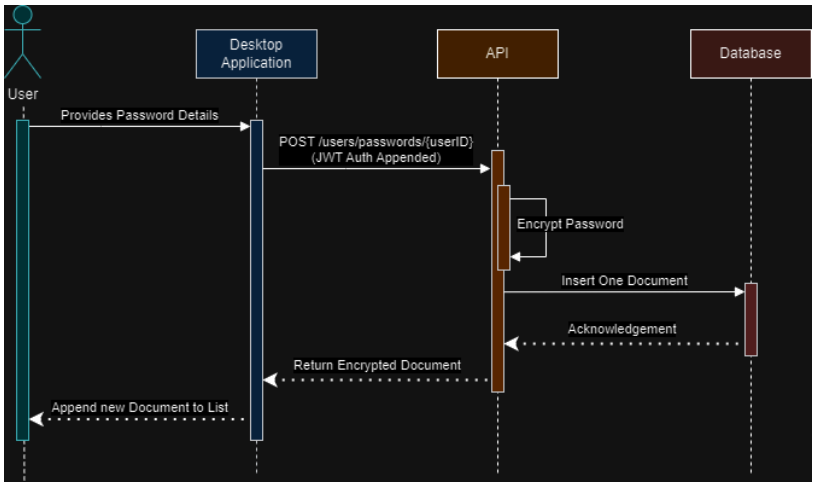


**Figure 3:** Sequence Diagram for user saving a password.

### 3.4.2 Class Diagram

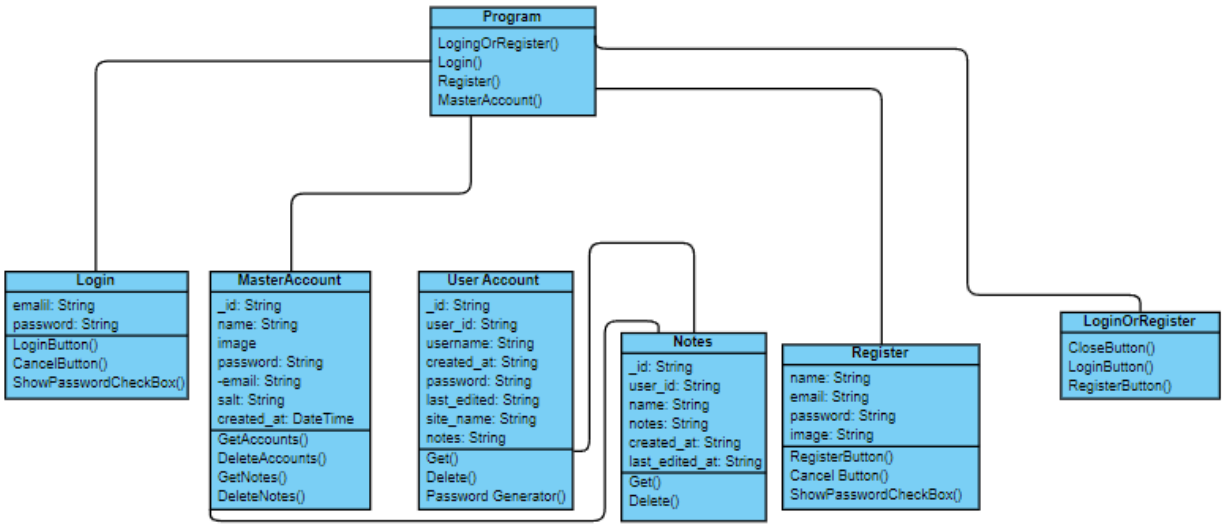


**Figure 4** Class Diagram

Figure 4 displays the relationships between all the classes within the Windows Desktop Application along with all the attributes and operations it contains and handles.

## 3.5 Security Vulnerabilities Identified [Jack]

Security testing is paramount in any application where sensitive information is handled, particularly for a password manager. Manual and automated testing was employed to assess how robust the project is, as vulnerabilities can affect a project and have significant risks to users' data and overall security. For automated testing, Snyk was utilised with two minor vulnerabilities found in each part of the project, which were remedied.

### 3.5.1 API/requirements.txt Vulnerabilities:

The 'requirements.txt' file in the project's API was found to be vulnerable to the following issues:

CWE-696

CVE-2023-29483

CVSS 5.9

SNYK-PYTHON-DNSPYTHON-6241713

To fix these vulnerabilities the following changes were changed to update the dependencies to change the version to a newer version of the dependencies which were in requirements.txt.

### 3.5.2 project/requirements.txt Vulnerabilities:

The ‘requirements.txt’ file in the project’s project section was found to be vulnerable to the following issues:

CWE-696

CVE-2023-29483

CVSS 5.9

SNYK-PYTHON-DNSPYTHON-6241713

To fix these vulnerabilities the following changes were made to update the dependencies to change the version to a newer version of the dependencies which were in requirements.txt.

### 3.5.3 Minor vulnerabilities that were fixed

#### *3.5.3.1 Writing saved in plain text*

Initially the project stored certain information in plain text but to mitigate this the AES-256 encryption was implemented to ensure that even if the database is compromised the stored information remains secure and unreadable without a decryption key.

#### *3.5.3.2 Implementation of Authentication*

Authentication mechanisms were implemented in the API to ensure secure access to endpoints. This was implemented as it was found there was no authentication, so industry standard authentication protocols were added. The implementation of API tokens was created so that users couldn't impersonate others and obtain access to accounts.

#### *3.5.3.3 Deletion of accounts*

While testing it was discovered that the system allows users to delete their accounts without properly invalidating existing accounts. This left it so that users could delete their accounts but remained still logged in and access their passwords. This is a significant security risk as unauthorized access to sensitive data could occur even after the user intends to terminate the account. Proper session management upon account deletion was implemented.

# 4 DISCUSSION

## 4.1 ACHIEVED RESULTS [OWEN]

The project successfully achieved its primary aims of designing and implementing a secure password manager, adhering to the principles of Secure Software Engineering throughout the Software Development Lifecycle. By implementing AES-256 encryption to ensure secure data transmission, the application provides a high level of security that is resilient against a range of attack vectors. The encryption strategy was particularly chosen for its efficiency and reversibility, enabling quick data retrieval, while the secure handling and movement of data via a REST API and MongoDB ensured that the application remained secure and reliable.

## 4.2 CRITICAL ANALYSIS OF THE DEVELOPMENT PROCESS [EWAN]

The secure software development was a fluid process throughout the project, with security issues being prioritised throughout. The use of an agile methodology also created a fluid environment, where areas where progress could be made were prioritised, to facilitate the building of a stable version of the password manager as quickly as possible. This was an effective approach, as only once a stable version of the password manager was available could a full suite of testing be applied. Furthermore, the collaborative efforts of the front and back-end developers allowed for paired programming to occur, minimising errors that can occur during the implementation of the project. Paired Programming, coupled with rigorous testing and evaluation, highlighted a commitment to maintaining security integrity and compliance with data protection laws.

Despite the identification of a few vulnerabilities, these were quickly addressed, substantially enhancing the system's security framework. The adoption of an agile methodology played a crucial role in this process, as it allowed for the flexible addition of tasks to the backlog and the ability to pivot priorities quickly, ensuring that security-related concerns were addressed promptly and effectively. This agile approach and the team's quick response to discovered issues underscored the effectiveness of the secure software development practices employed, ensuring the project met its security goals.

Challenges faced during the development process included not fully applying the range of testing desired. Further and more rigorous testing could have been employed if all members of the group had fully performed their roles and participated consistently throughout the development of the process. This greater involvement could have ensured a more comprehensive evaluation and refinement of the application.

## 4.3 Suggestions for Future Enhancements [Ewan]

Future improvements for the password manager would centre around expanding the methods of access by adapting the tool to function as a browser extension. This enhancement would not only make the tool easier to access directly through the browser, eliminating the need for users to open a separate application, but it would also be especially beneficial in environments where installing software on desktops is restricted. Additionally, making the tool available as a browser extension would increase its compatibility across multiple operating systems without requiring many pre-installed components, thus broadening its usability and reach.

## 4.4 Conclusion [Owen]

The development and implementation of the password manager project were carried out with security as one of the main goals. Testing discovered a small number of vulnerabilities, pointing towards overall code that has been maintained and completed at a high standard. The high code strength is reaffirmed through the ability to quickly remediate vulnerabilities that were discovered.

The project demonstrates a commitment to industry standards and security practices evidently by effectively using AES-256 encryption, secure data handling, and a rapid response to identified vulnerabilities. The introduction of authentication and improved session management further strengthened the security to ensure the users of the password manager had confidence in the product.

# REFERENCES


D. K. Davis, M. M. Chowdhury and N. Rifat (2022) *Password Security: What Are We Doing Wrong?* pp. 562.

M. Kotadia, "Gates predicts death of the password," Security, 2004. [Online]. Available: https://www.cnet.com/news/privacy/gates-predicts-death-of-the-password/ . [Accessed 06 April 2024]

G. Kurtz, (2024) Crowdstrike, 2024 Global Threat Report. Available at: https://go.crowdstrike.com/rs/281-OBQ-266/images/Report2020CrowdStrikeGlobalThreatReport.pdf (Accessed: 05 April 2024).

J. Lopez, (2023) Call for views on software resilience and security for businesses and organisations, gov.uk. Available at: https://www.gov.uk/government/publications/call-for-views-on-software-resilience-and-security-for-businesses-and-organisations/call-for-views-on-software-resilience-and-security-for-businesses-and-organisations (Accessed: 08 April 2024).

H. Padalia, H. Patel, A. Deshmukh, M. Patil, A. Kumar and N. Kumar Nrip (2023) *A Study on Password Manager: Users' Perspective.* pp. 72.

N. R. Mead, G. Seshagiri and J. Howar (2016) *Meeting Industry Needs for Secure Software Development.* pp. 28.

NCSC (2018) Password manager buyers guide, Password administration for system owners. Available at: https://www.ncsc.gov.uk/collection/passwords/password-manager-buyers-guide (Accessed: 06 April 2024).

R. A. Khan, S. U. Khan, H. U. Khan and M. Ilyas (2021) 'Systematic Mapping Study on Security Approaches in Secure Software Engineering', *IEEE Access,* 9, pp. 19139-19160 Available at: 10.1109/ACCESS.2021.3052311.

R. Maclean and J. Ophoff (2018) *Determining Key Factors that Lead to the Adoption of Password Managers.* pp. 1.

Vermeer, B. and Gee, T. (2020) Secure code review, Snyk. Available at: https://snyk.io/blog/secure-code-review/ (Accessed: 30 March 2024).

OWASP (2023) *Secure coding practices*, *OWASP Secure Coding Practices - Quick Reference Guide | Secure Coding Practices | OWASP Foundation*. Available at: https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/stable-en/01-introduction/05-introduction (Accessed: 15 April 2024).