Password Management System (PMS)

TEJAS MURALIDHAR, Offenburg University of Applied Sciences, Germany

The project's goal is to provide users with a secure and convenient method for creating and managing passwords. Password generation, RESTful password access, and strong security measures are among the key functionalities. The report addresses in detail the system architecture and secure software development practices used in PMS development to ensure the security and reliability of the system.

Additional Key Words and Phrases: Software Security, Threat Modelling, Secure Design, Password Management System, Unit Testing, Static Analysis

ACM Reference Format:
Tejas Muralidhar. 2024. Password Management System (PMS). 1, 1 (February 2024), 20 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Author's address: Tejas Muralidhar, tmuralid@hs-offenburg.de, Offenburg University of Applied Sciences, Badstraße 24, Offenburg, Baden-Württemberg, Germany, 77652.

Contents

1 INTRODUCTION

In our discussion of software security, we discussed the importance of taking a "shift-left" approach and incorporating security concerns early in the software development process. A good case study for the successful use of this approach is to develop a password management system.

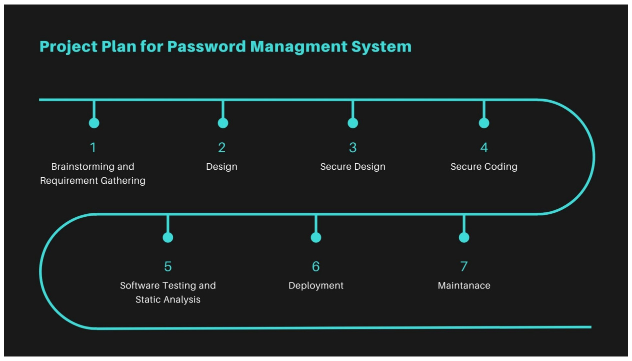
A secure password management system performs few essentials. Specifically, it must have the ability to generate passwords in accordance with pre-defined guidelines(policies), which may change in the future to accommodate new security risks and industry standards. This policy can specify requirements for passwords complexity, length, characters, upper and lower cases etc.

A system must be designed to store this data safely so as to prevent any unauthorized access towards the sensitive credentials and tampering with them. It should also provide strong encryption mechanisms and access controls for securing data used in password verification processes. Assertively, by making calls outside its own environment, such a system can check if passwords have been hacked or has compromised to prompt it to act on it through proactively initiating service calls using resources like the Have I Been Pwned (HIPB) database.

The range of the password management system is inclusive of the whole password managing cycle within an organization's framework. This includes generating, keeping track and checking on passwords together with establishing accountabilities for creation of new passwords or changing old ones. The above service for instance collaborates with external services such as Have I Been Pwned (HIBP) API which detects when someone has hacked one's email address, thus leading to a better security for password authentication process. Separate interfaces are available for administrators who maintain user accounts and system settings, while users themselves can set their own passwords or update profile information. Moreover, there are some security features in the system like administrative task authentication and password encryption among others.

2 THE SECURE SOFTWARE DEVELOPMENT LIFECYCLE

The Secure Software Development Lifecycle (SSDLC) is crucial for Password Management Systems (PMS) due to its importance in protecting user credentials, preventing data breaches, adhering to compliance standards, mitigating risks for legacy systems, fostering user trust and confidence, ensuring resilience against cyber threats, reducing operational risks, and encouraging continuous monitoring and improvement.


Project Plan for Password Managment System
1
Brainstorming and Requirement Gathering
2
Design
3
Secure Design
4
Secure Coding
5
Software Testing and Static Analysis
6
Deployment
7
Maintanace

A PMS handles sensitive facts, such as consumer names, and passwords. By integrating security measures during the development process, the probability of vulnerabilities which can lead to security breaches can be significantly reduced. Effective and reliable protection is essential for latency systems, as they can be vulnerable to risks. Thus, a robust PMS is necessary to counter these threats and seamlessly incorporate into the system. By implementing secure development methods, users can trust that their passwords and personal information are safeguarded. A strong Secure Software Development Lifecycle (SSDLC) model is crucial in detecting and resolving potential vulnerabilities during the development process. This not only minimizes operational risks linked to security breaches, but also encourages developers to actively confront evolving threats and bolster the system's security stance. Overall, the Secure SDLC is useful for establishing and sustaining a robust Password Management System that efficiently safeguards user credentials, mitigates security threats, complies with regulations, and security assurance amongst users.

3 SYSTEM DESIGN FOR PASSWORD MANAGEMENT SYSTEM

Having a well-crafted system design is of utmost importance when it comes to a Password Management System (PMS). This is because it has a direct influence on the system's effectiveness, level of security, and usability. The design essentially determines the system's capabilities, such as user interaction and password management. A carefully thought-out system design guarantees secure authentication methods, encryption protocols, and access controls, thus ensuring the safety of sensitive information. Moreover, it plays a critical role in making the system scalable, so it can easily handle an increased number of users and passwords without compromising its performance. The crucial role of system design extends beyond just meeting organizational requirements – it also guarantees reliability, adherence to regulatory standards. By honing database architecture, improving query speed, and reducing network delays, system design ensures that a PMS not only meets expectations but also safeguards user credentials.

3.1 Assumptions

The Secure Software Development project for the Password Management System has several key assumptions. These include a password policy defined in a JSON file, which defines acceptable password requirements. The passwords are generated using Python's secrets module for random number generation and password strength checks are performed using the "Have I Been Pwned" API to verify if a password has been exposed in data breaches.

An authentication mechanism is assumed for admin access, relying on an 'Admin-Token' header. A MySQL database is used for storing user credentials, and error-handling mechanisms are necessary for various scenarios. An admin panel is provided with functionalities like adding/deleting users, fetching statistics, managing system settings, and updating profiles.

Secure communication is assumed to be HTTPS, as it is generally a best practice in secure software development. Logging is necessary for debugging and auditing. Request validation is performed to ensure required parameters are present and have the expected format. Authorization checks are required before performing sensitive operations, such as generating passwords for multiple users or accessing admin functionalities. Resource isolation is ensured to prevent unauthorized access. MySQL database credentials are correctly configured and secured.

These assumptions highlight key aspects of the secure software development process for a password management system, including password security, authentication, authorization, data storage, error handling, and logging.

3.2 High-level overview of Password Management System

The system supports two types of users: admins and regular users. Admins have access to an admin panel for managing users, and system settings(like policy changes, password strength checks), while regular users have access to a user panel for password generation and password strength checks. The security measures include password generation following a defined policy, checking passwords against a service like HIBP API for known data breaches, and protecting admin actions with authentication tokens, and user actions with user authentication mechanisms. User data, including usernames and passwords, is stored in a MySQL database. The system connects to external services like the Have I Been Pwned (HIBP) API to check passwords and store user data in a MySQL database. Users and admins interact via HTTP requests to various endpoints. Passwords are generated based on policies and stored securely in the database.

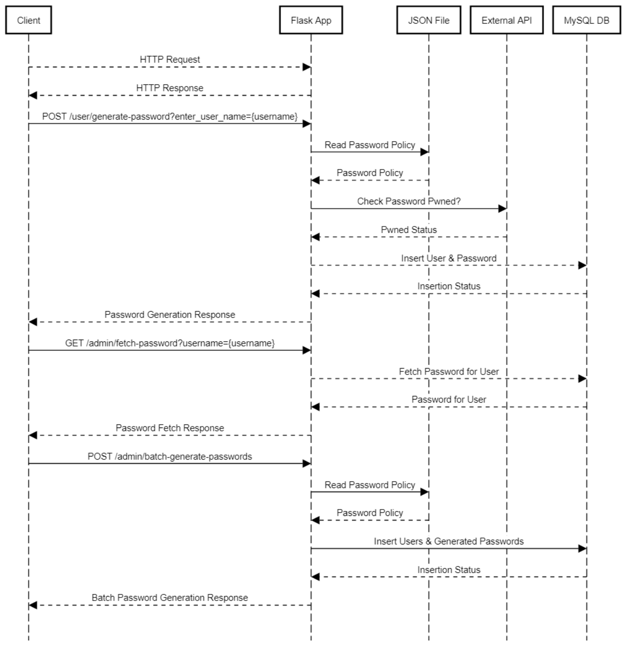


Fig. 1. Sequence Diagram of Password Management System

The Flask application is a web-based system that manages user data. It is a client-side system that uses HTTP requests to perform various tasks, such as generating passwords, fetching passwords, and performing batch operations. The application routes these requests to the appropriate endpoint based on the provided URL. The application reads the password policy from a JSON file to ensure it adheres to the defined policy. If the password is not compromised, it is inserted into the MySQL database using an SQL query. If an administrator requests a specific user's password, the application queries the MySQL database to retrieve the password associated with the provided username. If a batch password is generated, the application reads the password policy from the JSON file and generates passwords for multiple users. These generated passwords are then inserted into the MySQL database. The Flask application then sends an HTTP response back to the client with the result of the requested operation. This sequence diagram illustrates the flow of messages and interactions between the client, the Flask application, the MySQL database, the external HIBP API, and file operations during various actions.

3.3 Requirement Table

A Requirements Table is a structured approach to describing and organizing software requirements. It provides an overview of both functional and non-functional requirements, making it an effective tool for guiding the development team and communicating with stakeholders. The below table gives information about the functional and non-functional requirements of PMS. Functional requirements outline the precise features and functionalities expected from a software system, specifying its intended functionalities and it specifies the system's intended actions. In a Requirements Table, these functional requirements are commonly structured in a manner that enhances clarity and comprehension. Non-functional requirements define criteria for evaluating the way a system operates rather than specific behaviours. They address issues like performance, security, usability, and reliability.

Table 1. Password Management System Requirements

Category	Requirements
Functional Requirements	
User Management	Users should be able to register with a username and password. Registered users should be able to log in with their credentials. Users should be able to generate passwords based on predefined policies. Users should be able to fetch their stored passwords.
Admin Management	Admins should have privileges to perform administrative tasks. Admins should be able to authenticate using admin tokens. Admins should be able to perform operations such as adding/deleting users and fetching user passwords.
Password Generation	Passwords should be generated based on configurable policies. Policies should include parameters such as length, character types (uppercase, lowercase, digits, special characters), and any additional requirements.
Password Security	Passwords should be stored securely using strong hashing algorithms and salting. Communication between clients and the server should be encrypted using HTTPS. Authentication mechanisms should be robust and resistant to common attacks like brute force and replay attacks.
Database Interaction	The system should interact with a MySQL database for storing user information and passwords. CRUD operations should be supported for user and password management.
Non-Functional Requirements	
Security	Passwords should be stored securely using strong hashing algorithms and salting. Communication between clients and the server should be encrypted using HTTPS.
Performance	The system should be able to handle concurrent user requests efficiently. Database interactions should be optimized for performance, including efficient querying and indexing strategies.
Scalability	The system architecture should be scalable to accommodate increasing numbers of users and passwords. Database scalability should be considered to handle growing data volumes.
Reliability	The system should be highly available and resilient to failures. Error handling mechanisms should be in place to gracefully handle exceptions and provide meaningful error messages to users.
Logging and Monitoring	The system should log relevant events and actions for auditing, troubleshooting, and monitoring purposes. Logs should include information such as user actions, authentication events, database interactions, and system errors.
Integration Requirements	
Third-Party Services	Integration with a third-party service (e.g., Have I Been Pwned API) for checking if passwords are compromised. Integration with logging and monitoring tools for centralized log management and analysis.

3.4 Use Case Diagram

The figure represents use case diagram for a password management system. In this system, the users can generate password using dynamically defined password attributes. Also, the user can check if the password has been compromised to the internet before. The password management system can store the user id, password, also it should generate batch passwords supported in CSV format. The system should also be capable to update/change in policy of the password.

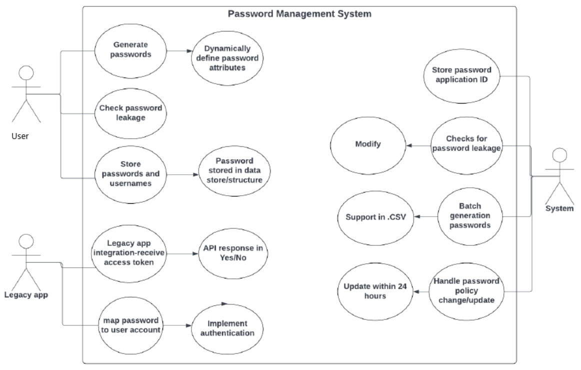

Password Management System
User
Generate passwords
Dynamically define password attributes
Check password leakage
Store passwords and usernames
Password stored in data store/structure
Legacy app
Legacy app integration-receive access token
API response in Yes/No
map password to user account
Implement authentication
Store password application ID
Modify
Checks for password leakage
Support in .CSV
Batch generation passwords
Update within 24 hours
Handle password policy change/update
System

Fig. 2. Use Case Diagram for PMS

4 SECURE DESIGN FOR PASSWORD MANAGEMENT SYSTEM(PMS)

Secure Design is a crucial aspect of the Secure Software Development Lifecycle (SDLC) by integrating security best practices from the beginning. It involves early integration of security considerations, threat modeling, secure architecture, secure libraries and components, and secure design documentation. Benefits include reduced vulnerabilities, improved code quality, faster development, cost savings, and enhanced compliance. The SDLC phases include requirements, design, development, testing, deployment, and maintenance. Secure design ensures proper configuration, access control, and timely application of security patches and updates, reducing the need for rework and vulnerabilities.

4.1 Threat Modeling for Password Management System

STRIDE is a threat modelling framework used in the field of computer security as a tool for the identification of various security risks involved in software development. It was developed by Microsoft engineers Loren Kohnfelder and Praerit Garg in 1999 [1]. STRIDE provides a mnemonic for security threats in six categories:

- Spoofing of user identity
- Tampering
- Repudiation
- Information disclosure (privacy breach or data leak)
- Denial of service (DOS)
- Elevation of privilege

STRIDE framework can be used during the software development process to identify threats, risk assessment, and implement risk mitigation strategies. Threat modelling with STRIDE is a continuous procedure as the software develops. Because new system features, updates, and changes may introduce new vulnerabilities, it is critical to reassess threats and security measures on a regular basis.
Threat modelling helps to understand system security threats and vulnerabilities, and how those threats potentially impact users and organizations, and to determine the most cost-effective security solutions to mitigate attacks [2] Using the STRIDE model during development allows organizations to create safer software by recognising and fixing potential threats early in the process, reducing the probability and impact of security breaches or vulnerabilities in developed systems.
The C4 Model is used to adjust abstraction layers for PMS software. The layers(C1, C2 and C3) are designed to understand the architecture of the software and to get an insight into possible threats occurring in the software system effectively.

4.2 C1 Level

The C1 level provides a high-level overview of the entire architecture. This level helps to understand the scope of the system without providing much technical details.

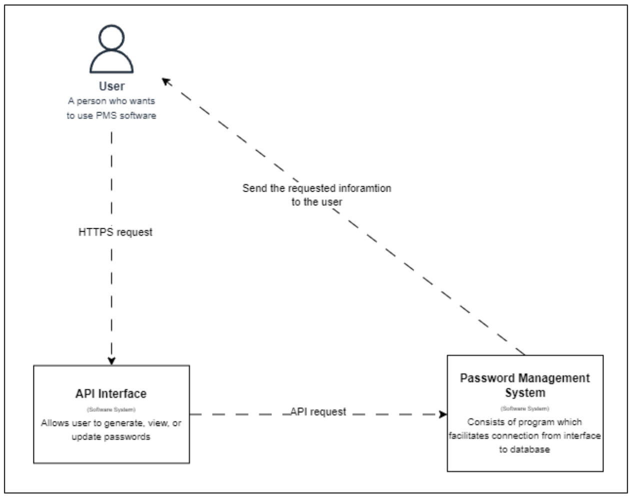

User
A person who wants to use PMS software
Send the requested inforamtion to the user
HTTPS request
API Interface
(Software System)
Allows user to generate, view, or update passwords
API request
Password Management System
(Software System)
Consists of program which facilitates connection from interface to database

Fig. 3. Fig (a) C1 Level Data Flow Diagram of PMS

The figure shows the C1 Level architecture of PMS software. It highlights the major components involved in the system and the high-level communication between each component.

4.3 C2 Level

This layer gives information about all the major blocks involved in each container. It explains the functional details of how different components involved in the system communicate with each other.

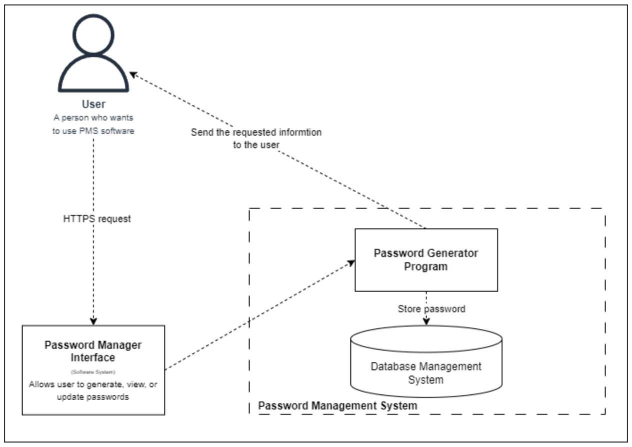

User
A person who wants to use PMS software
Send the requested informtion to the user
HTTPS request
Password Generator Program
Store password
Database Management System
Password Manager Interface
(Software System)
Allows user to generate, view, or update passwords
Password Management System

Fig. 4. C2 Level Data Flow Diagram of PMS

The figure shows the C2 Level architecture of PMS software. It highlights the major components involved specifically in PMS like password generator programs and database management systems.

4.4 C3 Layer

This layer breaks down each container into more detailed pieces. This layer helps in understanding the code functionality occurring at each compartment. It represents modules, classes, services, APIs etc.

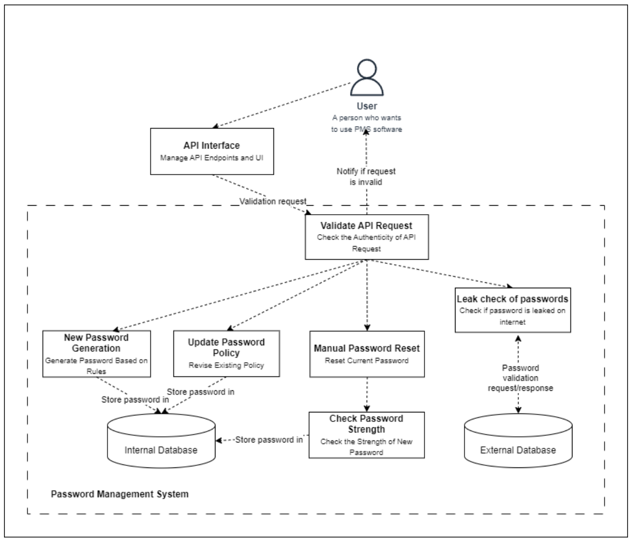

User
A person who wants to use PMS software
API Interface
Manage API Endpoints and UI
Notify if request is invalid
Validation request
Validate API Request
Check the Authenticity of API Request
Leak check of passwords
Check if password is leaked on internet
New Password Generation
Generate Password Based on Rules
Update Password Policy
Revise Existing Policy
Manual Password Reset
Reset Current Password
Password validation request/response
Store password in
Store password in
Check Password Strength
Check the Strength of New Password
Store password in
Internal Database
External Database
Password Management System

Fig. 5. C3 Level Data Flow Diagram of PMS

The figure shows the C3 Level architecture of PMS software. It gives much detailed information about the Components identified in the C2 layer.

4.5 Application of STRIDE on PMS Architecture

Below are the points which highlight the application of STRIDE on PMS architecture:

Spoofing: Login and authentication mechanisms are susceptible to spoofing attacks at the user end. Attackers might attempt to impersonate legitimate users to gain unauthorized access to passwords.
Tampering: Unauthorized modification of stored passwords can lead to data loss or unauthorized access at the database level.
Repudiation: Lack of proper logging within the PMS can lead to repudiation. Users or attackers might manipulate the system without leaving traces.
Information Disclosure: Weak password policies, insecure communication channels, or inadequate access controls can disclose the confidentiality of the system.
Denial of Service: The APIs may show unusual behaviour and may become irresponsive to admins/legit users of the system.
Elevation of Privilege: Insufficient access controls, weak authentication mechanisms, or vulnerabilities in the code. The attackers may gain unauthorized access to sensitive password data.

4.6 Threat Modelling Tools

Iriusrisk

IriusRisk is a tool that helps teams prioritize risks based on severity, providing an easy-to-use user interface and a compiled report of threats and countermeasures. It also enables tracking of risk mitigation measures. However, it requires regular updates and maintenance for new features and requires more effort to redesign the model for existing systems.

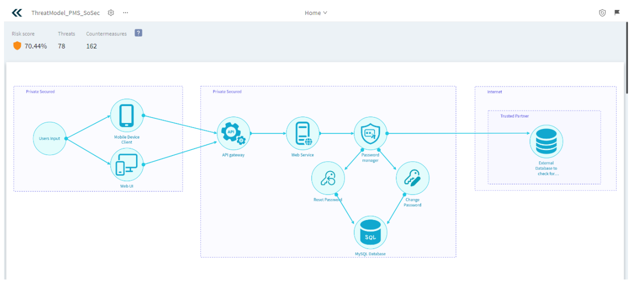


Fig. 6. Threat Modeling Diagram of PMS in IriusRisk

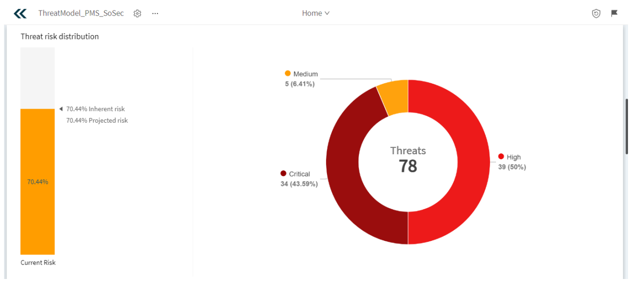
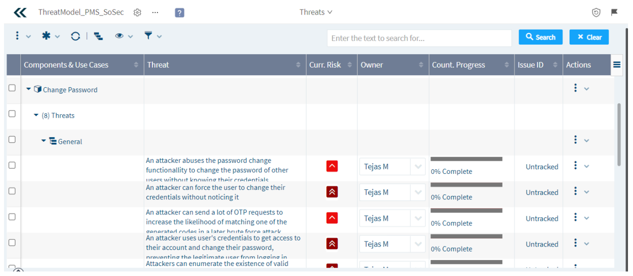


Fig. 7. Threat Risk Distribution of PMS in IriusRisk



Components & Use Cases	Threat	Curr. Risk	Owner	Count. Progress	Issue ID	Actions
Change Password						
(8) Threats						
General						
	An attacker abuses the password change functionallity to change the password of other users without knowing their credentials		Tejas M	0% Complete	Untracked	
	An attacker can force the user to change their credentials without noticing it		Tejas M	0% Complete	Untracked	
	An attacker can send a lot of OTP requests to increase the likelihood of matching one of the generated codes in a later brute force attack		Tejas M	0% Complete	Untracked	
	An attacker uses user's credentials to get access to their account and change their password, preventing the legitimate user from logging in		Tejas M	0% Complete	Untracked	
	Attackers can enumerate the existence of valid					

Fig. 8. Possible Threats Present in PMS

After building a threat model for PMS software, it was found that the model has a risk score of 70.44 percent and 78 threats. Below are some of the recommendations which can incorporated to increase the security of the system:

- To avoid attackers from exploiting vulnerabilities in the service and gain access to data, or to services for which they are not authorized, the users need to ensure that they are updating the passwords as per the latest password policy.
- An attacker can bring a service down by exhausting either the network or the service itself. This kind of attack can be prevented from taking place by rate-limiting ICMP traffic.
- An unauthorized party might access/breach the personal data stored in the database. personal data like username and password must be hashed so they cannot be recovered easily.

Microsoft Threat Modeling Tool
Microsoft Threat Modeller offers automated analysis for identifying potential threats and vulnerabilities in software designs, offering a structured approach. However, it is complex for beginners or those unfamiliar with threat modelling concepts and may become less effective or time-consuming due to limitations in handling complex architectures or customization. STRIDE is a framework that allows for a detailed examination of security vulnerabilities at a component level, understanding their impact on the entire system, and providing easy application. It has good documentation for reference. However, its scope is limited and requires knowledge of system design, making it suitable for software development. The number of threats can grow rapidly as a system increases in complexity.

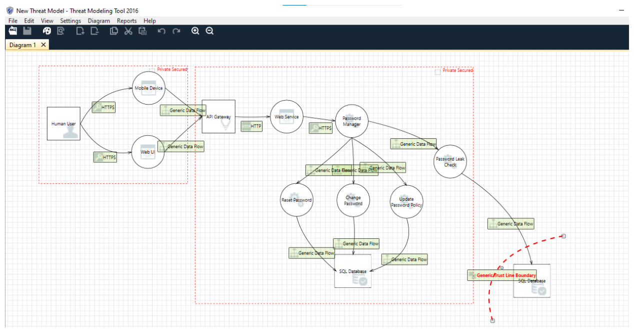
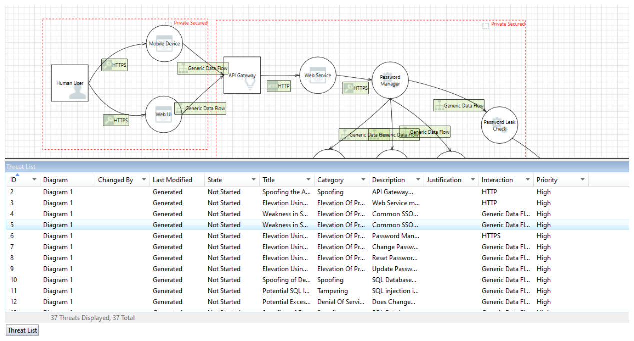


Fig. 9. Threat Modeling Diagram of PMS in Microsoft Threat Modeling Tool



ID	Diagram	Changed By	Last Modified	State	Title	Category	Description	Justification	Interaction	Priority
2	Diagram 1		Generated	Not Started	Spoofing the A...	Spoofing	API Gateway...		HTTP	High
3	Diagram 1		Generated	Not Started	Elevation Usin...	Elevation Of Pr...	Web Service m...		HTTP	High
4	Diagram 1		Generated	Not Started	Weakness in S...	Elevation Of Pr...	Common SSO...		Generic Data Fl...	High
5	Diagram 1		Generated	Not Started	Weakness in S...	Elevation Of Pr...	Common SSO...		Generic Data Fl...	High
6	Diagram 1		Generated	Not Started	Elevation Usin...	Elevation Of Pr...	Password Man...		HTTPS	High
7	Diagram 1		Generated	Not Started	Elevation Usin...	Elevation Of Pr...	Change Passw...		Generic Data Fl...	High
8	Diagram 1		Generated	Not Started	Elevation Usin...	Elevation Of Pr...	Reset Passwor...		Generic Data Fl...	High
9	Diagram 1		Generated	Not Started	Elevation Usin...	Elevation Of Pr...	Update Passw...		Generic Data Fl...	High
10	Diagram 1		Generated	Not Started	Spoofing of De...	Spoofing	SQL Database...		Generic Data Fl...	High
11	Diagram 1		Generated	Not Started	Potential SQL I...	Tampering	SQL injection i...		Generic Data Fl...	High
12	Diagram 1		Generated	Not Started	Potential Exces...	Denial Of Servi...	Does Change...		Generic Data Fl...	High

37 Threats Displayed, 37 Total

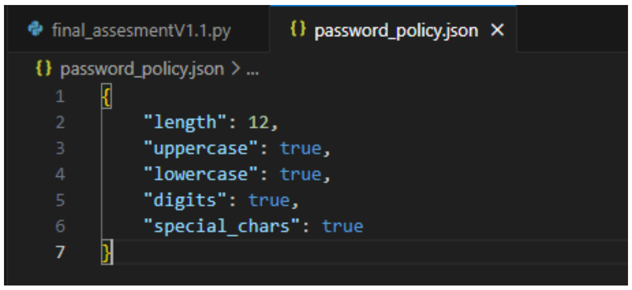
Fig. 10. Possible Threats in PMS

5 PROTOTYPICAL IMPLEMENTATION IN PYTHON AND RUST

The provided code is a Flask application written in Python for PMS, designed to serve as a password manager system. It includes endpoints for both admin and user panels to manage passwords, generate, fetch and manage passwords. The application interacts with a MySQL database to store and retrieve user information securely.

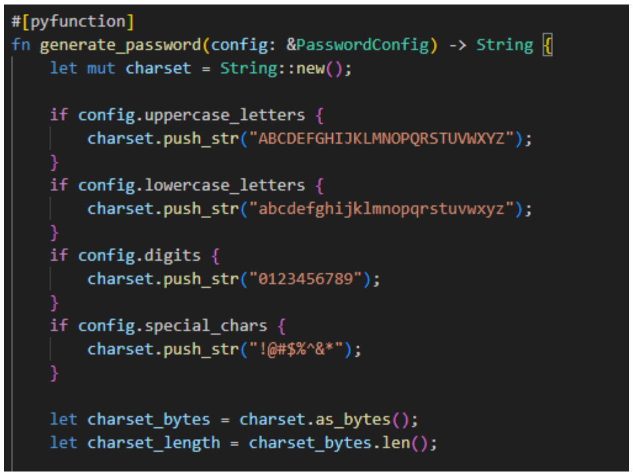
5.1 Setup and Dependencies:

The application requires Flask, MySQL Connector, and Requests libraries to be installed. It also assumes the availability of a MySQL database with appropriate credentials. Additionally, a password policy JSON file is needed to enforce password policies.



```json
{
    "length": 12,
    "uppercase": true,
    "lowercase": true,
    "digits": true,
    "special_chars": true
}
```

Fig. 11. Password Policy Defined in JSON File

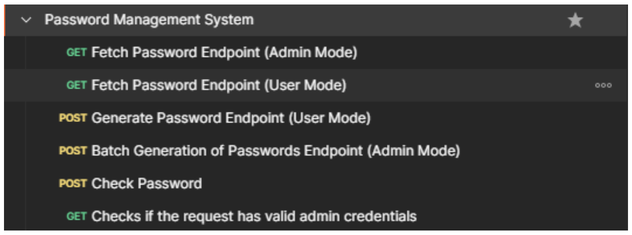


```rust
#[pyfunction]
fn generate_password(config: &PasswordConfig) -> String {
    let mut charset = String::new();

    if config.uppercase_letters {
        charset.push_str("ABCDEFGHIJKLMNOPQRSTUVWXYZ");
    }
    if config.lowercase_letters {
        charset.push_str("abcdefghijklmnopqrstuvwxyz");
    }
    if config.digits {
        charset.push_str("0123456789");
    }
    if config.special_chars {
        charset.push_str("!@#$%^&*");
    }

    let charset_bytes = charset.as_bytes();
    let charset_length = charset_bytes.len();
```

Fig. 12. Pre defined policy is linked to password generation function in RUST

5.2 Endpoints and Functions:

Admin Panel (/admin): Allows administrators to perform CRUD operations on users and access system statistics.
User Panel (/user): Allows users to generate passwords, check passwords, update profiles, etc.
Check Password Endpoint (/check-password): Validates whether a provided password has been compromised.
Generate Password Endpoint (/user/generate-password): Generates a password for a user based on specified password policies.
Fetch Password Endpoint (Admin and User mode): Retrieves the password for a specific user.
Admin Batch Generate Passwords Endpoint (/admin/batch-generate-passwords): Allows admins to generate passwords for multiple users in a batch.

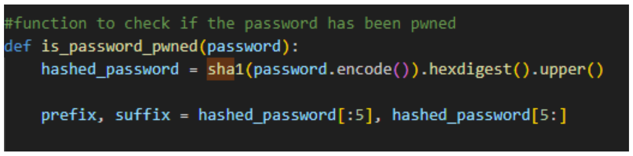


Password Management System
GET Fetch Password Endpoint (Admin Mode)
GET Fetch Password Endpoint (User Mode)
POST Generate Password Endpoint (User Mode)
POST Batch Generation of Passwords Endpoint (Admin Mode)
POST Check Password
GET Checks if the request has valid admin credentials

Fig. 13. List of all endpoints associated with PMS

5.3 Security Measures

Passwords are hashed using SHA-1 before being checked against a database of compromised passwords. Password policies are enforced to ensure generated passwords meet certain criteria (length, complexity). Admin authentication is required to access certain endpoints, implemented using admin tokens.

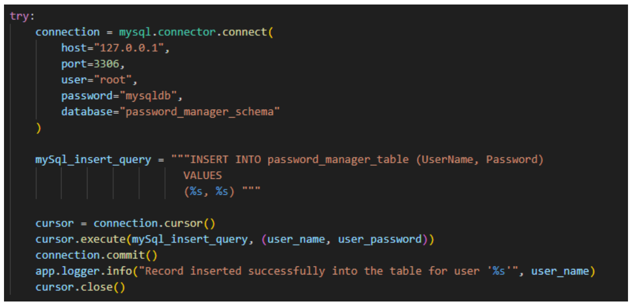


```python
#function to check if the password has been pwned
def is_password_pwned(password):
    hashed_password = sha1(password.encode()).hexdigest().upper()

    prefix, suffix = hashed_password[:5], hashed_password[5:]
```

Fig. 14. Password encrypted with SHA-1

5.4 Database and Interactions

The application interacts with a MySQL database to store user information (usernames and passwords). It uses SQL queries to insert, retrieve, and manipulate data in the database.

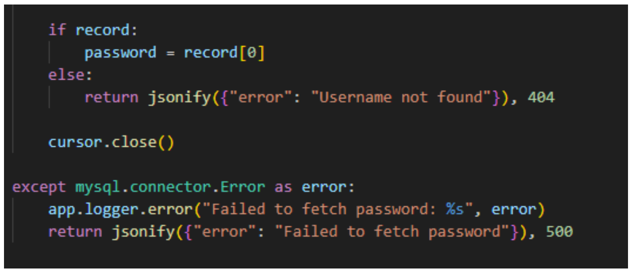


```python
try:
    connection = mysql.connector.connect(
        host="127.0.0.1",
        port=3306,
        user="root",
        password="mysqldb",
        database="password_manager_schema"
    )

    mySql_insert_query = """INSERT INTO password_manager_table (UserName, Password)
                            VALUES
                            (%s, %s) """

    cursor = connection.cursor()
    cursor.execute(mySql_insert_query, (user_name, user_password))
    connection.commit()
    app.logger.info("Record inserted successfully into the table for user '%s'", user_name)
    cursor.close()
```

Fig. 15. Database Operation

5.5 Error Handling

The application handles errors gracefully, returning appropriate HTTP status codes and error messages. It logs errors, warnings, and other events using Flask's built-in logging functionality.



```python
    if record:
        password = record[0]
    else:
        return jsonify({"error": "Username not found"}), 404

    cursor.close()

except mysql.connector.Error as error:
    app.logger.error("Failed to fetch password: %s", error)
    return jsonify({"error": "Failed to fetch password"}), 500
```

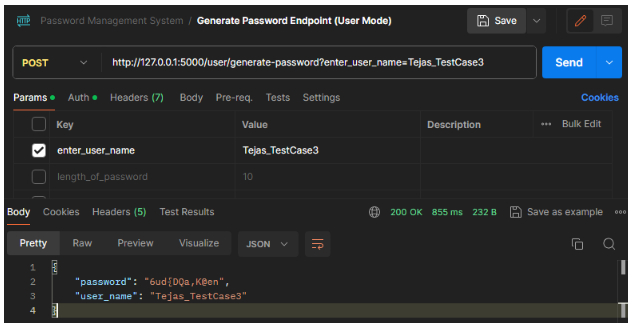
Fig. 16. Error handling mechanism in PMS - An Example

6 UNIT TESTING

Unit testing is a crucial practice in software development, ensuring the correct functioning of individual components. It involves isolating and testing the smallest building blocks of software, such as functions or modules, to verify their correctness before integration. This early detection of bugs aids in bug fixation, improves code quality, facilitates faster development, and provides confidence in future maintenance efforts. Unit testing offers several benefits, including reduced costs, improved software quality, continuous feedback, and team collaboration. It reduces time and money spent fixing bugs, ensures fewer crashes and unexpected behavior, and facilitates smooth handoffs between developers.

6.1 Test Case 1: Verify that the endpoint generates a password for a user following the password policy.

This test validates that the application can generate passwords according to the specified password policy. It ensures that users can receive strong and secure passwords that adhere to the predefined criteria. This is essential for maintaining the overall security posture of the application and its users.
Execution Step: Send a POST request to /user/generate-password endpoint with a custom username.
Expected Output: Response status code should be 200. Response JSON should contain the generated password and custom username.



POST http://127.0.0.1:5000/user/generate-password?enter_user_name=Tejas_TestCase3

	Key	Value	Description
✓	enter_user_name	Tejas_TestCase3	
	length_of_password	10	

200 OK 855 ms 232 B

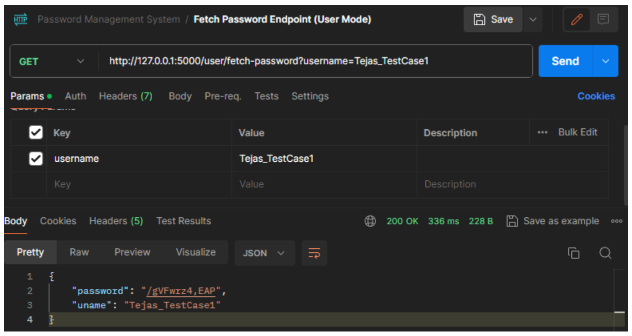
```
{
    "password": "6ud{DQa,K@en",
    "user_name": "Tejas_TestCase3"
}
```

Fig. 17. Send a POST request to /user/generate-password endpoint with a custom username.

Status: Pass

6.2 Test Case 2: Verify that the endpoint fetches the password for a specific user in user mode.

This test validates that the application can generate passwords according to the specified password policy. It ensures that users can receive strong and secure passwords that adhere to the predefined criteria.
Execution Step: Send a GET request to /user/fetch-password endpoint with a valid username.
Expected Output: For a valid username: Response status code should be 200. Response JSON should contain the correct username and password. For an invalid username: Response status code should be 404.



GET http://127.0.0.1:5000/user/fetch-password?username=Tejas_TestCase1

✓	Key	Value	Description
✓	username	Tejas_TestCase1	
	Key	Value	Description

200 OK 336 ms 228 B

```
{
    "password": "/gVFwrz4,EAP",
    "uname": "Tejas_TestCase1"
}
```

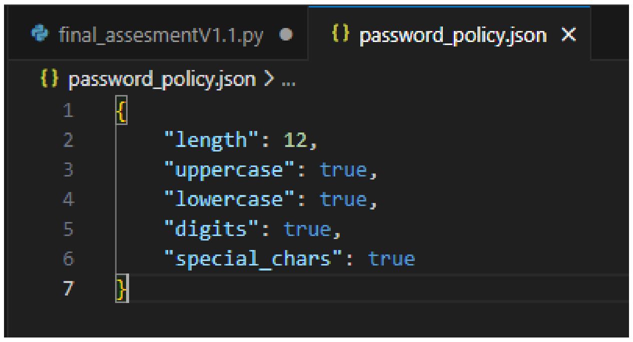
Fig. 18. Send a GET request to /user/fetch-password endpoint with a valid user name

Status: Pass

6.3 Test Case 3: Verify that generated passwords adhere to the password policy.

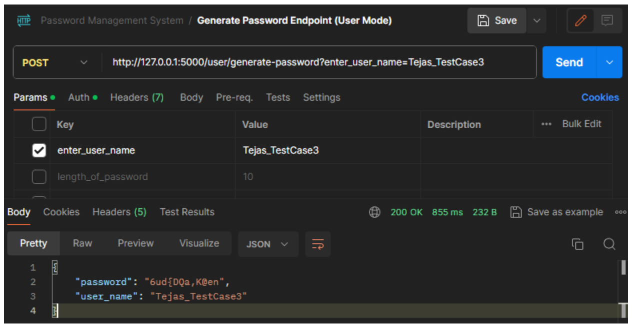
This test ensures that the passwords generated by the application adhere to the specified password policy. It's crucial to validate that generated passwords meet the required criteria, such as length and complexity, to ensure the security of user accounts. Weak passwords can lead to security vulnerabilities and compromise user accounts.

Execution Step: Generate password with different combinations of password policy settings (length, uppercase, lowercase, digits, special characters).



```
{
    "length": 12,
    "uppercase": true,
    "lowercase": true,
    "digits": true,
    "special_chars": true
}
```

Fig. 19. Policy defined in JSON file



POST http://127.0.0.1:5000/user/generate-password?enter_user_name=Tejas_TestCase3

Key	Value	Description
✓ enter_user_name	Tejas_TestCase3	
length_of_password	10	

200 OK 855 ms 232 B

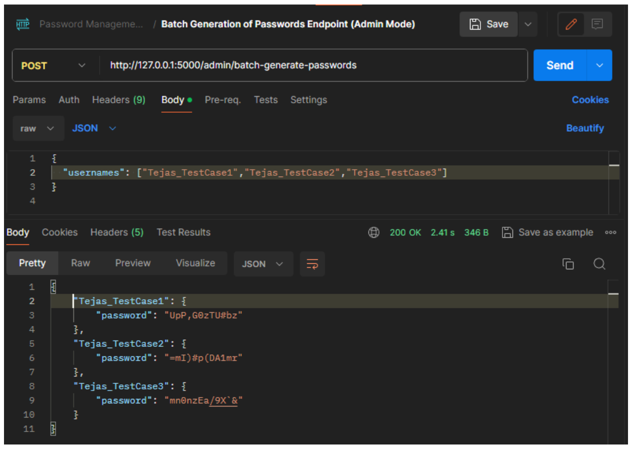
```
{
    "password": "6ud{DQa,K@en",
    "user_name": "Tejas_TestCase3"
}
```

Fig. 20. Password generated as per the policy

Expected Output: Generated passwords should meet the specified criteria based on the password policy settings.
Status: Pass

6.4 Test Case 4: Verify the ability to generate passwords for multiple users(Batch Generation of Passwords)

This test case ensures that the batch password generation endpoint works correctly and provides generated passwords for multiple users according to the specified policy. It helps in efficiently managing user accounts and ensuring password security across the system.
Execution Step: Send a POST request to /admin/batch-generate-passwords endpoint with a list of usernames. Verify the response for each username in the list.
Expected Output: If password generation is successful: Response status code: 200 Response JSON contains generated passwords.


Password Manageme... / Batch Generation of Passwords Endpoint (Admin Mode)

POST http://127.0.0.1:5000/admin/batch-generate-passwords

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

raw JSON Beautify

```
{
  "usernames": ["Tejas_TestCase1","Tejas_TestCase2","Tejas_TestCase3"]
}
```

Body Cookies Headers (5) Test Results — 200 OK 2.41 s 346 B Save as example

Pretty Raw Preview Visualize JSON

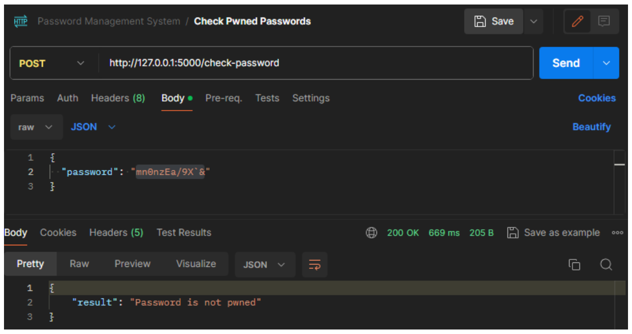
```
{
    "Tejas_TestCase1": {
        "password": "UpP,G0zTU#bz"
    },
    "Tejas_TestCase2": {
        "password": "=mI)#p(DA1mr"
    },
    "Tejas_TestCase3": {
        "password": "mn0nzEa/9X`&"
    }
}
```

Fig. 21. Batch generation of passwords

Status: Pass

6.5 Test Case 5: Verify that the endpoint correctly checks if a password is known to be pwned.

This test ensures that the password-checking mechanism works correctly. It's crucial to verify that the application correctly identifies whether a password is known to be compromised or not. This helps in ensuring the security of user accounts and preventing the use of weak or compromised passwords.


Password Management System / Check Pwned Passwords

POST http://127.0.0.1:5000/check-password

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

raw JSON Beautify

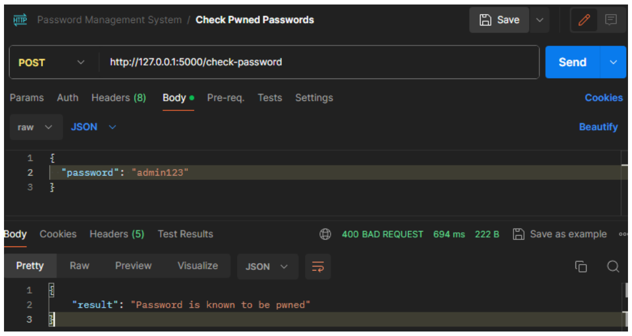
```
{
  "password": "mn0nzEa/9X`&"
}
```

Body Cookies Headers (5) Test Results — 200 OK 669 ms 205 B Save as example

Pretty Raw Preview Visualize JSON

```
{
    "result": "Password is not pwned"
}
```

Fig. 22. Password not pwned


Password Management System / Check Pwned Passwords

POST http://127.0.0.1:5000/check-password

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

raw JSON Beautify

```
{
  "password": "admin123"
}
```

Body Cookies Headers (5) Test Results — 400 BAD REQUEST 694 ms 222 B Save as example

Pretty Raw Preview Visualize JSON

```
{
    "result": "Password is known to be pwned"
}
```

Fig. 23. Password known to be pwned

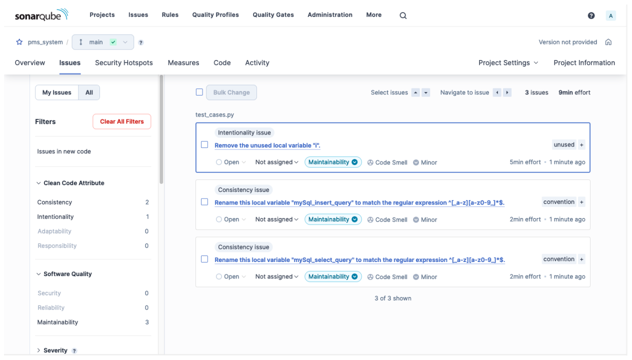
Execution Step: Send a POST request to /check-password endpoint with a password known to be pwned and not known to be pwned.

Expected Output: For a pwned password: Response status code should be 400. Response JSON should contain the message indicating that the password is known to be pwned. For a non-pwned password: Response status code should be 200. Response JSON should contain the message indicating that the password is not pwned.
Status: Pass

7 STATIC CODE ANALYSIS FOR PMS

7.1 Sonar

Static code analysis with SonarQube is the process of examining code without executing it. SonarQube allows us to inspect the quality of code. Regarding the code for the password management system, a code analysis has been conducted to identify security vulnerabilities.



Issues in new code

Clean Code Attribute

Consistency	2
Intentionality	1
Adaptability	0
Responsibility	0

Software Quality

Security	0
Reliability	0
Maintainability	3

Severity

3 issues 9min effort

test_cases.py

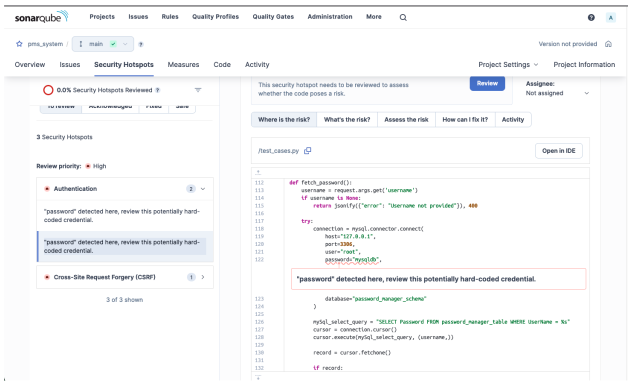
Intentionality issue
Remove the unused local variable "i". unused
Open Not assigned Maintainability Code Smell Minor 5min effort · 1 minute ago

Consistency issue
Rename this local variable "mySql_insert_query" to match the regular expression ^[_a-z][a-z0-9_]*$. convention
Open Not assigned Maintainability Code Smell Minor 2min effort · 1 minute ago

Consistency issue
Rename this local variable "mySql_select_query" to match the regular expression ^[_a-z][a-z0-9_]*$. convention
Open Not assigned Maintainability Code Smell Minor 2min effort · 1 minute ago

3 of 3 shown

Fig. 24. The above figure provides an overall depiction of the vulnerabilities present in the Python program for the password management system. The report highlights some of the maintainability fixes needed in the program.



0.0% Security Hotspots Reviewed

3 Security Hotspots

Review priority: High

Authentication 2

"password" detected here, review this potentially hard-coded credential.

"password" detected here, review this potentially hard-coded credential.

Cross-Site Request Forgery (CSRF) 1

3 of 3 shown

This security hotspot needs to be reviewed to assess whether the code poses a risk.

Assignee: Not assigned

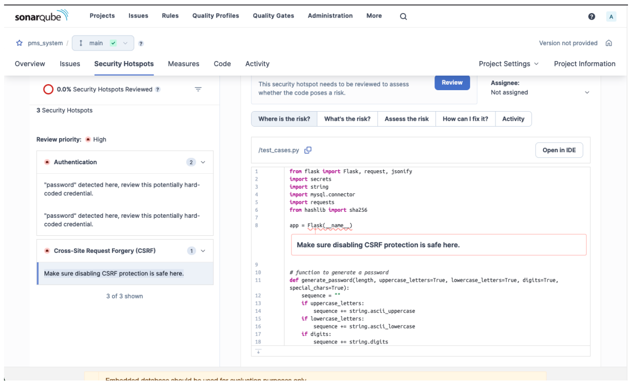
/test_cases.py

```
def fetch_password():
    username = request.args.get('username')
    if username is None:
        return jsonify({"error": "Username not provided"}), 400

    try:
        connection = mysql.connector.connect(
            host="127.0.0.1",
            port=3306,
            user="root",
            password="mysqldb",
```

"password" detected here, review this potentially hard-coded credential.

```
            database="password_manager_schema"
        )

        mySql_select_query = "SELECT Password FROM password_manager_table WHERE UserName = %s"
        cursor = connection.cursor()
        cursor.execute(mySql_select_query, (username,))

        record = cursor.fetchone()

        if record:
```

Fig. 25. Since the username and password to access the MySQL database are hardcoded, Sonar is suggesting to review the code.



0.0% Security Hotspots Reviewed

3 Security Hotspots

Review priority: High

Authentication 2

"password" detected here, review this potentially hard-coded credential.

"password" detected here, review this potentially hard-coded credential.

Cross-Site Request Forgery (CSRF) 1

Make sure disabling CSRF protection is safe here.

3 of 3 shown

This security hotspot needs to be reviewed to assess whether the code poses a risk.

Assignee: Not assigned

/test_cases.py

```
from flask import Flask, request, jsonify
import secrets
import string
import mysql.connector
import requests
from hashlib import sha256

app = Flask(__name__)
```

Make sure disabling CSRF protection is safe here.

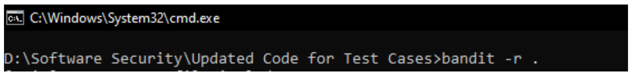
```

# function to generate a password
def generate_password(length, uppercase_letters=True, lowercase_letters=True, digits=True,
special_chars=True):
    sequence = ""
    if uppercase_letters:
        sequence += string.ascii_uppercase
    if lowercase_letters:
        sequence += string.ascii_lowercase
    if digits:
        sequence += string.digits
```

Embedded database should be used for evaluation purposes only

Fig. 26. The statement suggests that we should be cautious when considering disabling this CSRF protection in the Flask application. CSRF protection is a security feature, and disabling it should only be done after implementing an alternate approach.

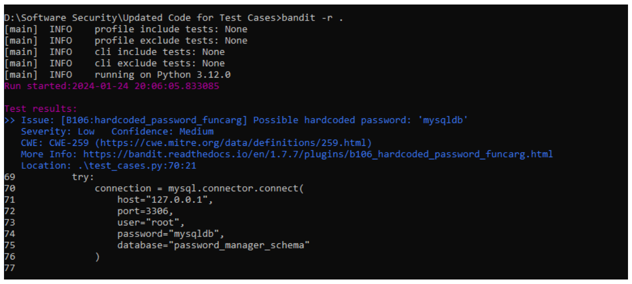
7.2 Bandit

Bandit is a static code analysis tool designed to find security issues in Python code. It specifically focuses on identifying security vulnerabilities and also the issues that can lead to security vulnerabilities in Python applications.



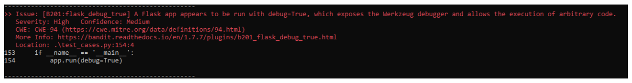
```
C:\Windows\System32\cmd.exe
D:\Software Security\Updated Code for Test Cases>bandit -r .
```

Fig. 27. The above command recursively analyzes all Python files in the current directory and its subdirectories for security vulnerabilities.



```
D:\Software Security\Updated Code for Test Cases>bandit -r .
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.12.0
Run started:2024-01-24 20:06:05.833085

Test results:
>> Issue: [B106:hardcoded_password_funcarg] Possible hardcoded password: 'mysqldb'
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
   More Info: https://bandit.readthedocs.io/en/1.7.7/plugins/b106_hardcoded_password_funcarg.html
   Location: .\test_cases.py:70:21
69          try:
70              connection = mysql.connector.connect(
71                  host="127.0.0.1",
72                  port=3306,
73                  user="root",
74                  password="mysqldb",
75                  database="password_manager_schema"
76              )
77
```

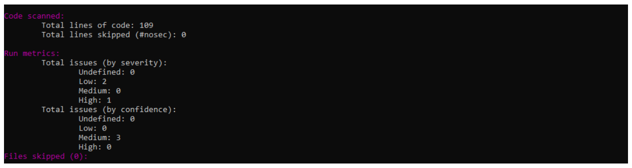
Fig. 28. Issue with low severity



```
--------------------------------------------------
>> Issue: [B201:flask_debug_true] A Flask app appears to be run with debug=True, which exposes the Werkzeug debugger and allows the execution of arbitrary code.
   Severity: High   Confidence: Medium
   CWE: CWE-94 (https://cwe.mitre.org/data/definitions/94.html)
   More Info: https://bandit.readthedocs.io/en/1.7.7/plugins/b201_flask_debug_true.html
   Location: .\test_cases.py:154:4
153     if __name__ == '__main__':
154         app.run(debug=True)

--------------------------------------------------
```

Fig. 29. Issue with high severity

As we can see in the above fig 2.2a and 2.2b, each issue will have an associated code (ex. B106), a severity level (High, Medium, Low), a confidence level, and the location in the code where the issue was found.

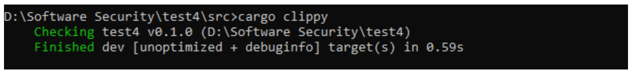


```
Code scanned:
        Total lines of code: 109
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 2
                Medium: 0
                High: 1
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 3
                High: 0
Files skipped (0):
```

Fig. 30. Bandit provides a report with security issues it found in the code. It categorizes issues by severity (Low, Medium, High).

7.3 Analyzing RUST code with Clippy

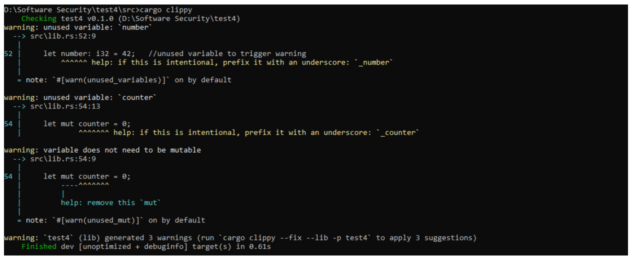
Clippy is a linting tool for Rust code, designed to catch common mistakes and improve code quality by enforcing best practices.



```
D:\Software Security\test4\src>cargo clippy
    Checking test4 v0.1.0 (D:\Software Security\test4)
    Finished dev [unoptimized + debuginfo] target(s) in 0.59s
```

Fig. 31. The initial code written for the password management system passed Clippy's analysis without any issues or warnings.

To verify the behaviour of the tool, I have manually introduced some errors to see if Clippy can catch them or not.



```
D:\Software Security\test4\src>cargo clippy
    Checking test4 v0.1.0 (D:\Software Security\test4)
warning: unused variable: `number`
  --> src\lib.rs:52:9
   |
52 |     let number: i32 = 42;   //unused variable to trigger warning
   |         ^^^^^^ help: if this is intentional, prefix it with an underscore: `_number`
   |
   = note: `#[warn(unused_variables)]` on by default

warning: unused variable: `counter`
  --> src\lib.rs:54:13
   |
54 |     let mut counter = 0;
   |             ^^^^^^^ help: if this is intentional, prefix it with an underscore: `_counter`

warning: variable does not need to be mutable
  --> src\lib.rs:54:9
   |
54 |     let mut counter = 0;
   |         ----^^^^^^^
   |         |
   |         help: remove this `mut`
   |
   = note: `#[warn(unused_mut)]` on by default

warning: `test4` (lib) generated 3 warnings (run `cargo clippy --fix --lib -p test4` to apply 3 suggestions)
    Finished dev [unoptimized + debuginfo] target(s) in 0.61s
```

Fig. 32. Clippy has issued two warnings, highlighting unused variables and identifying a mutable variable issue.

8 IMPACT ANALYSIS WRT TO DESIGN, IMPLEMENTATION AND TESTING

Password policy changes, such as length requirements, character types, or complexity rules, require updating the generatepassword() function and the readpasswordpolicy() function. This has a moderate impact, as it requires updating password generation logic and ensuring consistency. Password-pwned checks, such as moving from HIBP API to a different service, require updating the ispasswordpwned() function. Database schema changes require updating SQL queries in functions like fetchpasswordadminmode() and fetchpasswordusermode(). This has a moderate impact, as it involves changes in database interactions and may require thorough testing to ensure data integrity. Admin authentication changes, such as switching from token-based to session-based authentication, require modification of the isvalidadmin() function. Logging changes may require adjustments in logging statements throughout the code, with a low to moderate impact depending on the extent of changes. Testing is crucial for maintaining the security and functionality of the application, with a high impact due to the importance of comprehensive testing.

9 SECURE STORAGE OF PASSWORDS AND SUPPORT PASSWORD VALIDATION WITH THIRD PARTY APPLICATIONS

Keeping passwords safe is essential for security. Passwords are securely hashed as opposed to being stored in plain text. By giving each password a random value, salting prevents identical hashes and provides an additional layer of security. Additional care should be taken to avoid brute-force attacks to further mitigate risks. Passwords should be transmitted securely over networks using HTTPS, which guards against man-in-the-middle and eavesdropping attacks.

The system provides an API endpoint for password validation, allowing third-party applications to request a user's username and password. The system retrieves the hashed password and salt associated with the username from the database, uses the same algorithm and salt to hash the password, and compares it with the stored hash. If they match, the password is considered valid. The system also responds to the third-party application and sends a response over HTTPS. To prevent brute-force attacks, the service implements rate-limiting mechanisms.

10 CONCLUSION

The Password Management System is a robust and efficient project that includes essential functionalities like generating passwords, checking password strength against breaches, securely storing passwords, and providing endpoints for user and admin operations. It incorporates security measures like password hashing, secure data transmission over HTTPS, and password strength checking using the Have I Been Pwned API. The system is modular, with clearly defined functions and endpoints, making it easy to understand, maintain, and extend. Error handling mechanisms, such as logging and HTTP error responses, enhance the system's reliability. The system design allows for scalability by separating user and admin functionalities, implementing database interactions efficiently, and employing secure authentication mechanisms.

11 FUTURE WORK AND ENHANCEMENTS

PMS will be further enhanced with the below features:

- Expand authentication options to support secure methods like OAuth 2.0, JWT-based, or multi-factor authentication.
- Implement session management features for persistent user sessions.
- Introduce caching mechanisms to improve performance and reduce database load. Optimize database queries, indexing strategies, and data retrieval mechanisms to minimize latency and improve system responsiveness.
- All password validation attempts will be logged for auditing and security analysis purposes, including username, IP address and timestamp, so that at the time of auditing, all activities can be traced.

12 REFERENCES

[1] https://en.wikipedia.org/wiki/STRIDE(security)
[2]A. Shostack, Threat modeling: Designing for security. John Wiley and Sons, 2014.
[3] Khan, Rafiullah and Mclaughlin, Kieran and Laverty, David and Sezer, Sakir. (2017). STRIDE-based threat modeling for cyber-physical systems. 1-6. 10.1109/ISGTEurope.2017.8260283.
[4] Abomhara, Mohamed and Køien, Geir and Gerdes, Martin. (2015). A STRIDE-Based Threat Model for Telehealth Systems.