PYTHON Password manager Code audit

Check list and report



May 24, 2023

IT21049972

# **Code repository**

Uses the built in PyCharm support version control system, however to this project version control system is set up on GitHub - [IT21049972/pwManager (github.com)](https://github.com/IT21049972/pwManager)

# Technology choices

*Main technology choices*

* **Python** – Not making any special statements python seemed to be a very good option for this project.
* **MySQL** – At database level mysql works perfect with python.
* **Tkinter library –** This is a python library used to create moden graphical user interface for applications
* Apart from these there are some other libraries used to create this project

Are all applications libraries up to date?

pip freeze > requirements.txt

pip-check -r requirements.txt

each package used to build this application is listed in the requirements file and compared alongside with --> website (<https://pypi.org/>)

|  |  |  |
| --- | --- | --- |
| **Libraries** | **Version** | **LTS** |
| bandit | 1.7.5 |  |
| certifi | 2022.12.7 |  |
| charset-normalizer | 3.1.0 |  |
| click | 8.1.3 |  |
| colorama | 0.4.6 |  |
| dparse | 0.6.2 |  |
| gitdb | 4.0.10 |  |
| GitPython | 3.1.31 |  |
| markdown-it-py | 2.2.0 |  |
| mdurl | 0.1.2 |  |
| pbr | 5.11.1 |  |
| Pygments | 2.14.0 | 2.15.0 |
| pyparsing | 3.0.9 |  |
| requests | 2.28.2 |  |
| rich | 13.3.3 | 13.3.4 |
| ruamel.yaml | 0.17.21 |  |
| ruamel.yaml.clib | 0.2.7 |  |
| safety | 2.3.5 |  |
| smmap | 5.0.0 |  |
| stevedore | 5.0.0 |  |
| toml | 0.10.2 |  |
| urllib3 | 1.26.15 |  |

* **Majority of the libraries are run in the latest versions**

# **Coding best practices**

Is the code compliant with the PEP 8 style guide and the PEP 257 docstring conventions ?

To test whether the code is compliant with the PEP8 style guide the **pylint** tool was used.

According to the initial test the code was rated 5.35 out of 10 later it was increased to 7.59 some of the common linting errors were bad indentation, invalid-name and line too long. Invalid-name error was raised as a result of using camelCase method but the pylint tool recommends snake-case naming styles. These types of errors can be ignored by inserting this command “#pylint: disable=error-name” after the line of code which the error points to for example



Snapshots of the test results initial and final

Graphical user interface, text

Description automatically generated

Managed to increase the rating of the code to 7.68 by ignoring the snake case naming convention suggestions and also by adding necessary indentations etc. and also the rating is affected by the readability there code segments like This **if result == True** were replaced by

**if result is TRUE**

**this 🡪** password\_letters = []  
 for \_ in range (random.randint(8, 10)):  
 letter = random.choice(letters)

was replaced by this

password\_letters = [choice(letters) for \_ in range(randint(8, 10))]

\*Note – A good rating on this scale is considered when code analysis scores at least 9 out of 10



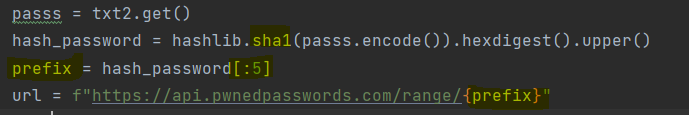
# Testing for common security issues

The bandit and safety libraries were used to test this code for security vulnerabilities similarly these tools can be installed by entering **“pip install bandit” */* “pip install safety ”** and to run the tests enter the tool name and the file to be tested along with it – “*bandit main.py”*

*Security issues are as follows;*

* **Issue: [B105:hardcoded\_password\_string] Possible hardcoded password: 'INSERT INTO masterpassword(password, recoveryKey) VALUES(?, ?) ' -** can be ignored since the password is encrypted and stored on the database
* **Issue: [B311:blacklist] Standard pseudo-random generators are not suitable for security/cryptographic purposes. –** This security vulnerability was shown a couple of times as a result of **CWE-330: Use of Insufficiently Random Values.** This was also ignored because to generate the password a mixture of upper case letters, lowercase letters , digits and special characters were used. To provide more context there are 70 character used to randomly generate a password and also a password should atleast be 12 character long therefore there are atleast 70^12(7012) number of combinations
* **Issue: [B324:hashlib] Use of weak SHA1 hash for security. Consider usedforsecurity=False**

**Severity: High Confidence: High – SHA1 is used to check whether the entered password have been associated with a previous data breach, which is risen as a conflict in the haveibeenpwned.com API to check password**

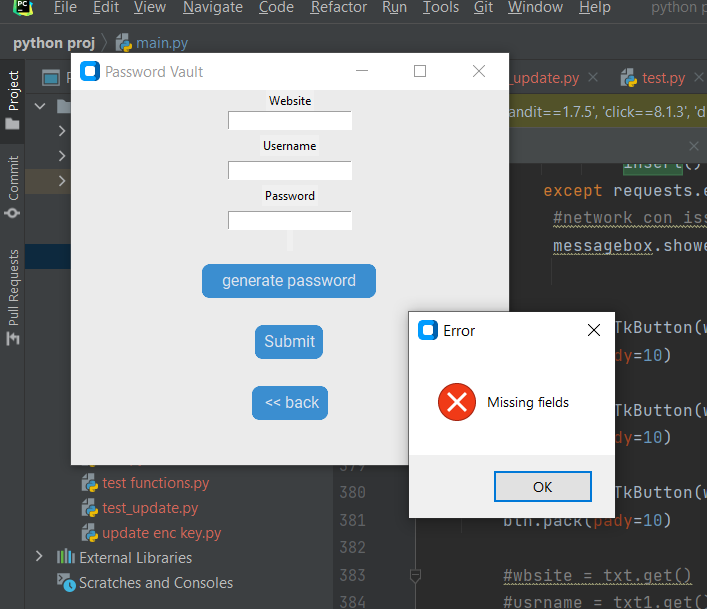


The have I been pwned website uses a technique called K anonymity to check if a password is associated with a data breach –

The entered password is first hashed using the SHA1 algorithm and the first five characters of the hashed password which is called a prefix is sent to the server, and the server returns a list of hashed password which matches he prefix, afterwards the users computer carries out the comparison to check the hashed password with the received list of passwords – in this scenario the users security must be well established

According to the OWASP framework this application can be subjected to SQL injections although necessary countermeasures are applied by encrypting each input data not allowing to escape characters.

**Input validations –** Does not accept empty fields, when a user tries to add an empty field or update a field with empty values a “missing fields ” error is raised

 Graphical user interface, text, application, chat or text message

Description automatically generated

# bugs identified.

have identified various bugs for now some of them are.

* Does not allow users to login after entering the password incorrectly – in this scenario th user is should close the application and reopen it this is due to conflict with PBKDF2 instance

This was fixed by creating a new instance every time a user enter the password incorrectly (in tho login Screen function)

**kdf = PBKDF2HMAC(  
 algorithm=hashes.SHA256,  
 length=32,  
 salt=salt,  
 iterations=100000,  
 backend=backend  
)**