ENTERPRISE AND IT COMPUTING (ENITS)
OFFENBURG UNIVERSITY OF APPLIED SCIENCES

SOFTWARE SECURITY

WINTER SEMESTER 2022/2023

UNIT TESTING & STATIC CODE ANALYSIS

LECTURER : SCHAAD, ANDREAS

PROF. DR. PIL. M.SC

AUTHOR :

ARINN DANISH BIN ABDULLAH (191050-01)

TABLE OF CONTENT

UNIT TEST CASES

Project Name	Password Management System
Module Name	Login
Created By	Aniss Danish
Creation Date	01-10-2023
Reviewed By	
Review Date	MM-DD-YYYY

Test Scenario ID	Test Scenario Description	Test Case ID	Test Case Description	Test Steps	Pre-Conditions	Test Data	Post Conditions	Expected Result	Actual Result	Status
TS_PMS_001	Verify the login functionality of PMS Login page	TC_PMS_Login_001	Enter a valid username/ID and valid password	1. Enter valid username/ID 2. Enter valid password 3. Click on login button	Valid URL Test Data	username/ID: pms.User password: P@ssw0rd	User should be able to see the home page	Successful login		Pass
		TC_PMS_Login_002	Enter a valid username/ID and invalid password	1. Enter valid username/ID 2. Enter invalid password 3. Click on login button	Valid URL Test Data	username/ID: pms.User password: xxxxxxx	Error message " Invalid username/ID or password "	A pop-up message box to show an error "Invalid username/ID or password"		Fail
		TC_PMS_Login_003	Enter a invalid username/ID and valid password	1. Enter invalid username/ID 2. Enter valid password 3. Click on login button	Valid URL Test Data	username/ID: xxxxxxx password: P@ssw0rd	Error message " Invalid username/ID or password "	A pop-up message box to show an error "Invalid username/ID or password"		Fail
		TC_PMS_Login_004	Enter an invalid username/ID and invalid password	1. Enter invalid username/ID 2. Enter invalid password 3. Click on login button	Valid URL Test Data	username/ID: xxxxxxx password: xxxxxxx	Error message " Invalid username/ID or password "	A pop-up message box to show an error "Invalid username/ID or password"		Fail
TS_PMS_002	Verify the "Check Password" functionality of PMS	TC_PMS_CheckPassword_001	Choose checking method: username/ID and insert valid value	1. Choose checking method: username/ID 2. Enter valid value	Valid URL Test Data	username/ID: username value: pms.User	User should be able to see his/her password	Successful access		Pass
		TC_PMS_CheckPassword_002	Choose checking method: username/ID and insert invalid value	1. Choose checking method: username/ID 2. Enter invalid value	Valid URL Test Data	username/ID: username value: xxxxxxx	Error message " Invalid value"	A pop-up message box to show an error "Invalid value"		Fail

This is a unit testing template consisting of 6 different test cases divided into 2 test scenarios which are *Verify the Login Functionality of PMS Login Page (TS_PMS_001)* & *Verify the "Check Password" Functionality of PMS (TS_PMS_002).*

For *TS_PMS_001*, there are 4 different states that are high-likely to occur when a user or system admin trying to login into the system, which narrowed down to:

1. Enter a valid username/ID and password
2. Enter a valid username/ID and invalid password
3. Enter an invalid username/ID and valid password
4. Enter an invalid username/ID and invalid password

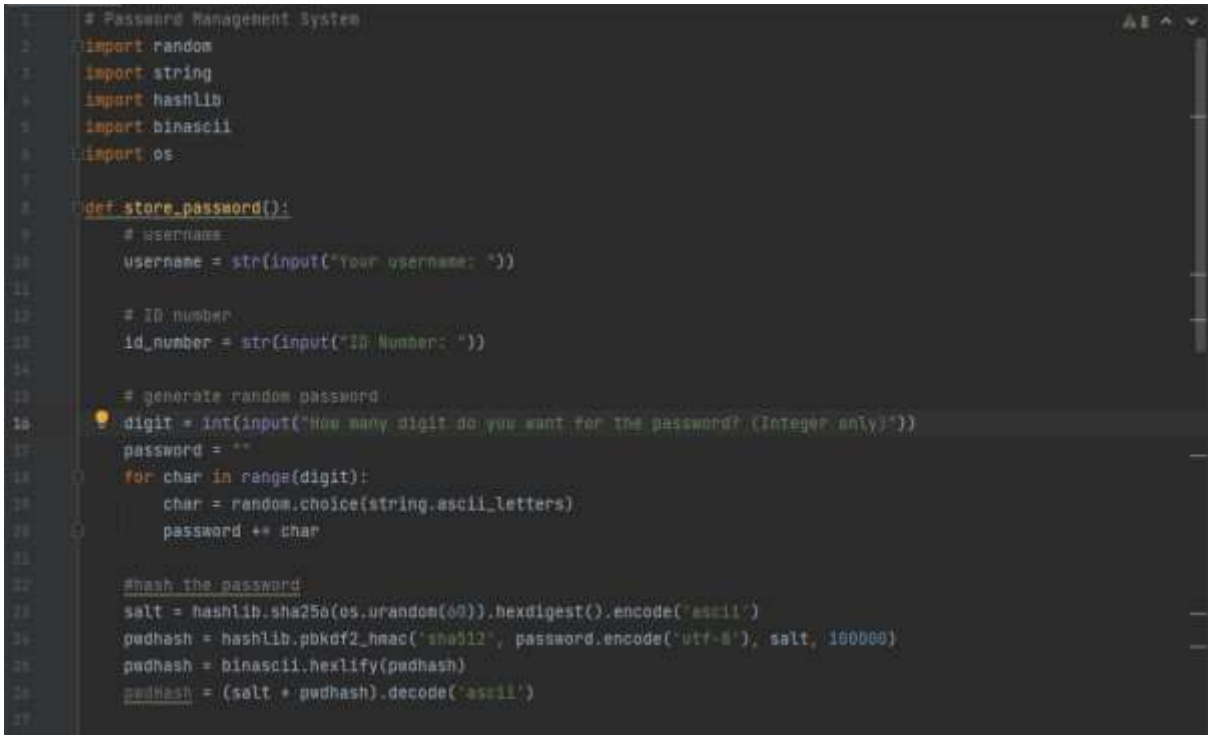
While for *TS_PMS_002*, there are only 2 different states that are to occur when a user or system admin prompt data (Username/ID and value) into the PMS, which are listed below:

1. Choose checking method: username/ID and insert valid value
2. Choose checking method: username/ID and insert invalid value

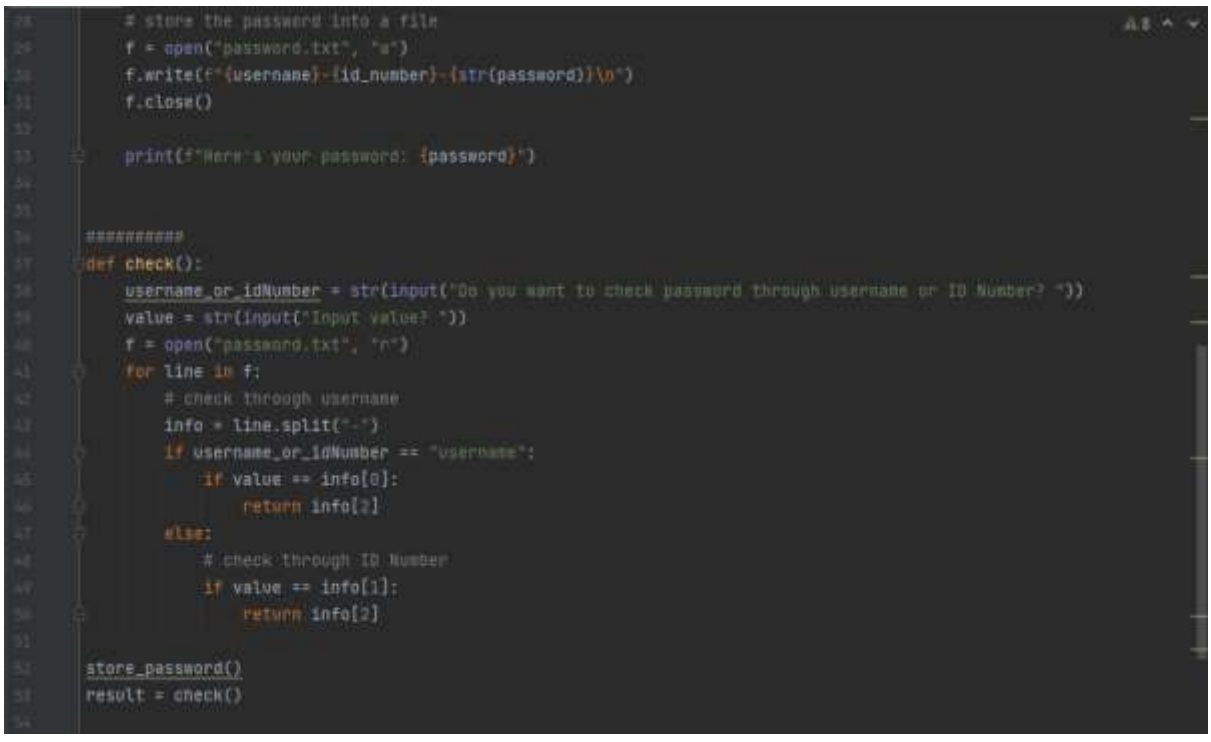
In each state the *steps taken* by user or system admin while logging in into the system are recorded as mentioned and *pre-conditions / post-conditions* are important to be jotted down as a requirement and result of either successful or unsuccessful attempt of logging in into the PMS.

Test Data is a bool example of input where the information affects the outcome or post-condition of the login action. In the other hand, *actual result / expected result* is taken note of as it will indicate how the PMS system behaves in real-time situation and testing situation. As a summary, a *status grade* of either Pass or Fail shows if the user or system Admin is able to login into the PMS or not.

STATIC CODE ANALYSIS



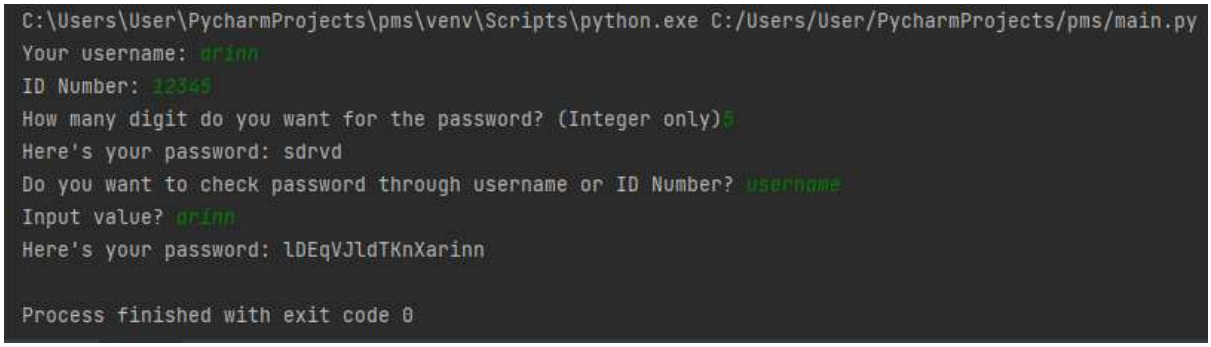
```python
# Password Management System
import random
import string
import hashlib
import binascii
import os

def store_password():
    # username
    username = str(input("Your username: "))

    # ID number
    id_number = str(input("ID Number: "))

    # generate random password
    digit = int(input("How many digit do you want for the password? (Integer only)"))
    password = ""
    for char in range(digit):
        char = random.choice(string.ascii_letters)
        password += char

    #hash the password
    salt = hashlib.sha256(os.urandom(60)).hexdigest().encode('ascii')
    pwdhash = hashlib.pbkdf2_hmac('sha512', password.encode('utf-8'), salt, 100000)
    pwdhash = binascii.hexlify(pwdhash)
    pwdhash = (salt + pwdhash).decode('ascii')
```



```python
    # store the password into a file
    f = open("password.txt", "a")
    f.write(f"{username}-{id_number}-{str(password)}\n")
    f.close()

    print(f"Here's your password: {password}")


#########
def check():
    username_or_idNumber = str(input("Do you want to check password through username or ID Number? "))
    value = str(input("Input value? "))
    f = open("password.txt", "r")
    for line in f:
        # check through username
        info = line.split("-")
        if username_or_idNumber == "username":
            if value == info[0]:
                return info[2]
        else:
            # check through ID Number
            if value == info[1]:
                return info[2]

store_password()
result = check()
```

```python
if result == None:
    print("No result found")
else:
    print(f"Here's your password: {result}")
```

PMS Python prototype using PyCharm



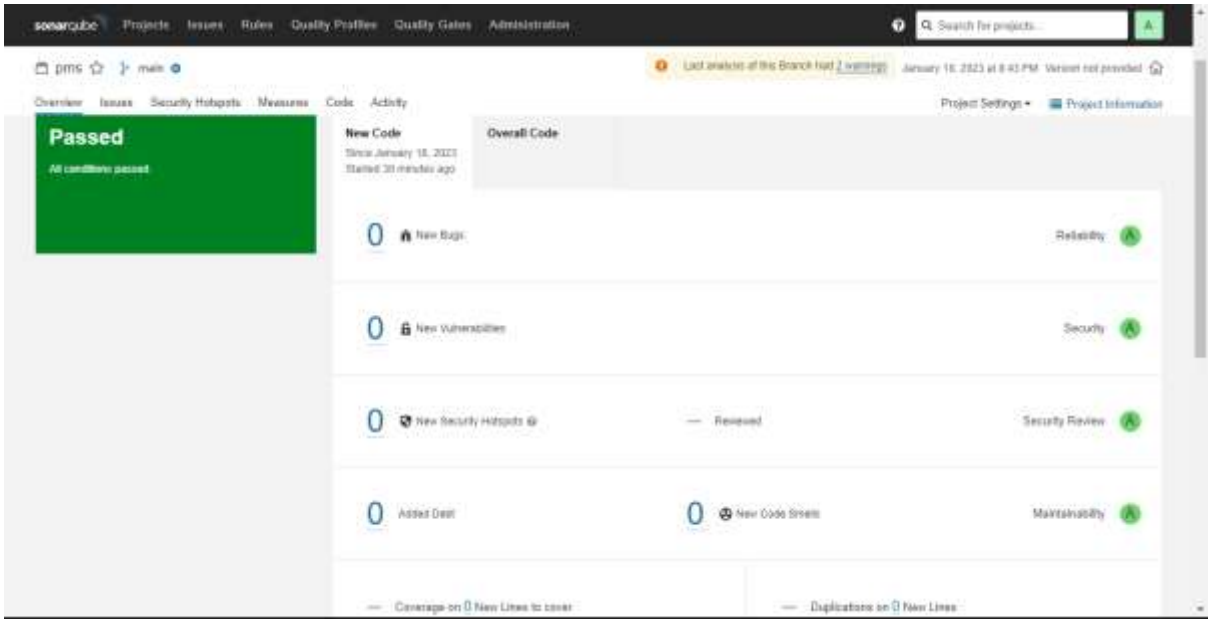
PMS Python codes output

The above codes written are only a hunch of the entire PMS system and this prototype focuses on 2 main functions which are:

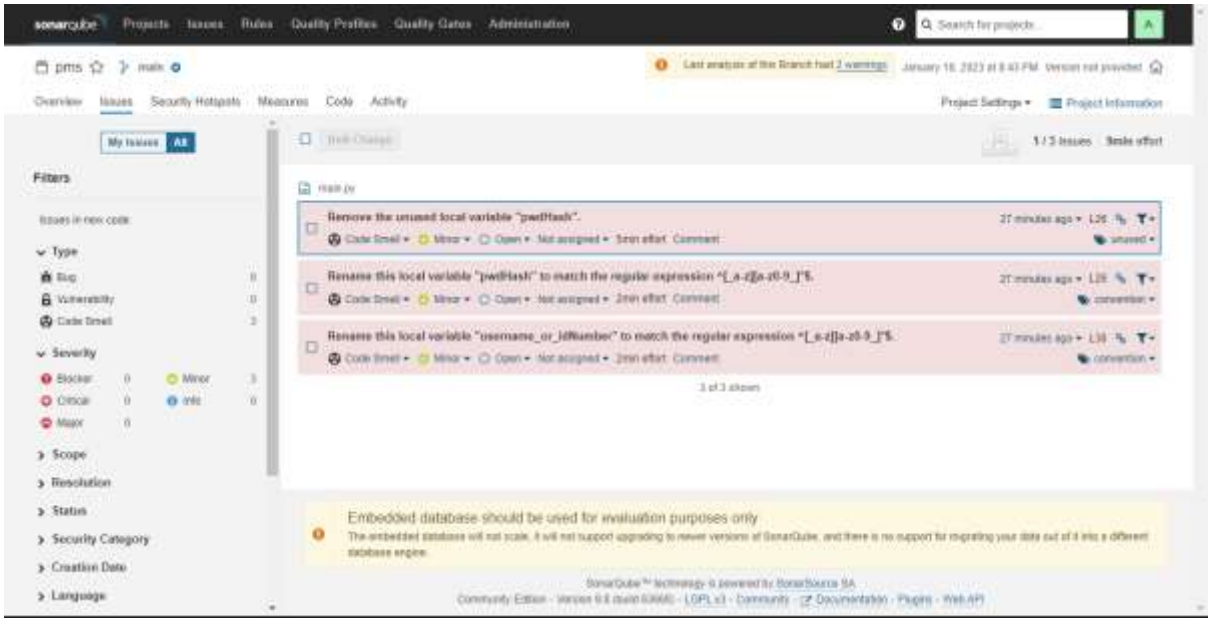
1. Generate and store password
2. Check password

First of all, important library dependencies are imported and next *store_password* function is defined where the system must ask for the username and ID number in the beginning and then prompt the user to input specific length of his/her random password. The random password will then be generated from a list of ASCII letters and next sent to the hash function before being stored in a .txt file. In the end, the password generated will be shown to the user.

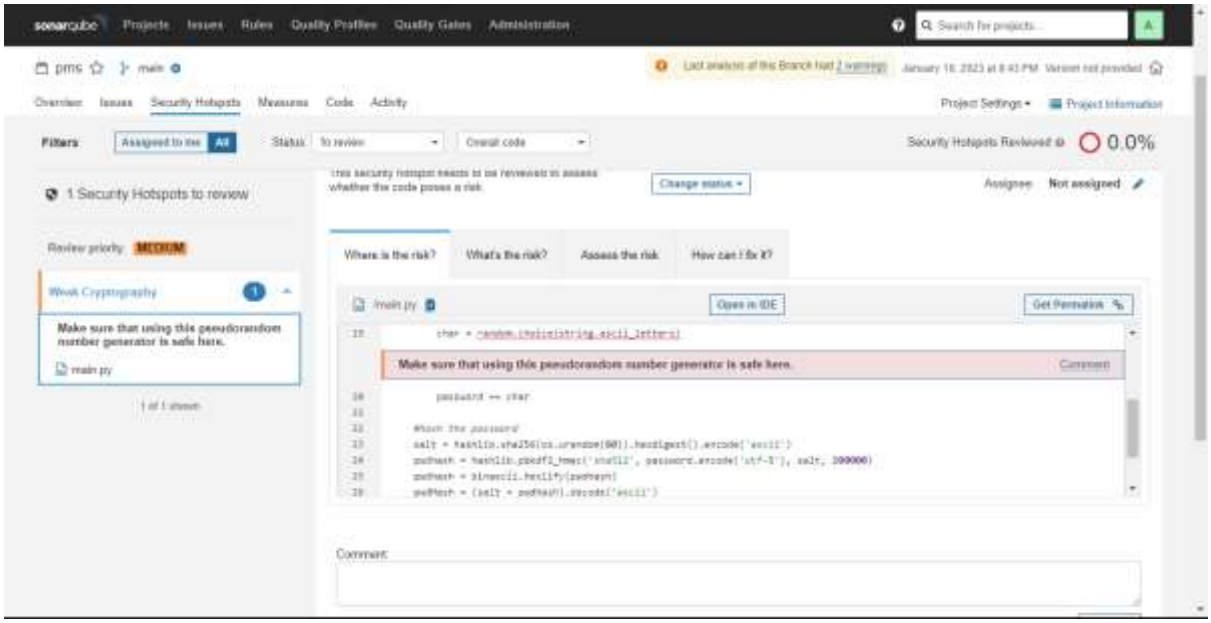
In the other hand, the *check* function prompt user to either input his/her username or ID and the value for password checking. If the username or ID value inserted exists in the storage file, the PMS will then show his/her password created earlier in the system. However, if the value does not exist in the database, the PMS will show “No result found” as a post condition statement.



Overview – SonarQube Analysis Report



Issues - SonarQube Analysis Report



Security Hotspots - SonarQube Analysis Report

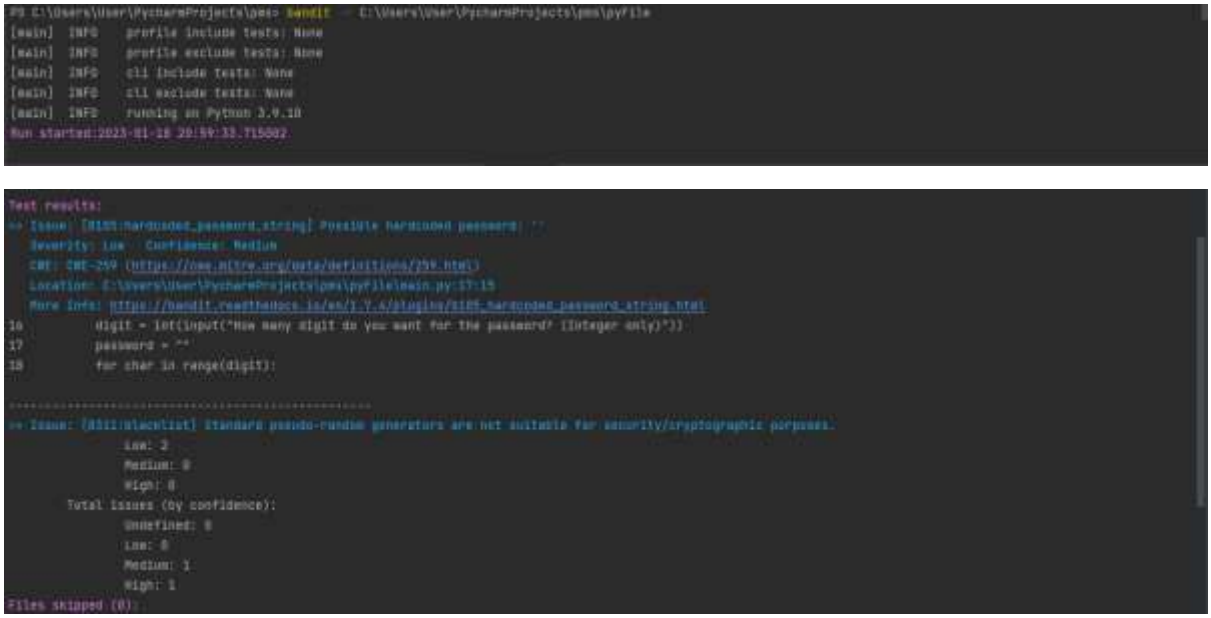
Here in SonarQube it shows that the successful or passed static code analysis of PMS prototype where the explanation will be divided into three parts whereas:

1. Overview
2. Issues
3. Security Hotspot

First of all, an *overview* of the static code analysis of PMS prototype itself indicated passed result where all conditions are successful in terms of code reliability, security, security review and maintainability. There were no bugs and vulnerabilities found inside the prototype codes.

However, there were some problems highlighted in the *issues* part where SonarQube recommends to remove the "pwdHash" local variable because it was unused after been created. The second recommendation is to rename the unused local variable and "Username_or_idNumber" to some variables that match its regular expression. By this, we can detect there was a typo for the code where "pwdHash" variable was supposed to be written with the small "h" – "pwdhash" etc. Even when these minor errors occur, the entire code can still be executed as usual.

Lastly, the *security hotspots* show what and where the risk might be in the PMS prototype. In this scenario, it shows the pseudorandom number generator was equipped with weak cryptography, allowing it to be exposed to threats. However, the good news is that SonarQube will then provide some recommendation and action to be taken for covering up the vulnerabilities.



Bandit Analysis Report

In the other hand, Bandit was also introduced as one of the effective tools to perform static code analysis on the PMS prototype. Unlike SonarQube, Bandit is only a Python software library that could easily downloaded and implemented in our Python codes, using the same virtual platform without external installation required. Only by inserting a simple line of code and pasting file directory path, Bandit will show the test result of our codes by displaying different level of *issues* (ranging from undefined, low, medium and high). In this scenario, the *issue* is that the prototype has a weak cryptography whereas pseudo-random generator was not suitable for security purpose as it easily been exposed to vulnerabilities. And by confidence of threats detection, there were 1 medium and 1 high level issues to be fix.

Although SonarQube and Bandit both perform great high analytical static code scanning, I would personally prefer SonarQube over Bandit as it provides better details of exposed risks plus tons of make-sense recommendation of actions to fix the vulnerabilities, even though the installation and implementation of SonarQube is far more complicated compared to Bandit.

REFERENCES

- https://www.vitoshacademy.com/hashing-passwords-in-python/
- https://replit.com/@boyuan12/SereneAncientCompilerbug#main.py
- https://youtu.be/aP3m2YFmroU
- https://youtu.be/HCoqJ1ArbUU
- https://www.youtube.com/watch?v=g0PrXoWKM2Y