FILE INTEGRITY MONITORING TOOL CODE AUDIT

Check list and report

May 28, 2023

Perera l.m.c - IT21049972

Sri Lanka Institute of Information Technology



Contents

[Code Repository 2](#_Toc135306354)

[Technology Choices 2](#_Toc135306355)

[Coding best practices 4](#_Toc135306356)

[Testing for common security issues 5](#_Toc135306357)

[Safety tool analysis 8](#_Toc135306358)

[Bugs identified. 9](#_Toc135306359)

[Extras 9](#_Toc135306360)

# Code Repository

This project utilizes PyCharm's built-in support for version control, but it is configured to use GitHub as the dedicated version control system. You can find the GitHub repository for this project at the following link: GitHub Repository. - <https://github.com/IT21049972/file-monitoring-tool.git>

# Technology Choices

Main technology choices

* Python – Deciding to use python seemed like a sustainable option because the code can be modified according to latest technologies – Ai, machine learning.

Libraries used :

List of all technologies used can be obtained by

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/>)

|  |  |
| --- | --- |
| **Packages** | **Versions** |
| astroid | 2.15.2 |
| bandit | 1.7.5 |
| cachetools | 5.3.0 |
| certifi | 2022.12.7 |
| charset-normalizer | 3.1.0 |
| click | 8.1.3 |
| colorama | 0.4.6 |
| dill | 0.3.6 |
| dparse | 0.6.2 |
| et-xmlfile | 1.1.0 |
| exceptiongroup | 1.1.1 |
| gitdb | 4.0.10 |
| GitPython | 3.1.31 |
| httplib2 | 0.22.0 |
| idna | 3.4 |
| iniconfig | 2.0.0 |
| isort | 5.12.0 |
| lazy-object-proxy | 1.9.0 |
| markdown-it-py | 2.2.0 |
| mccabe | 0.7.0 |
| mdurl | 0.1.2 |
|  |  |
| penpyxl | 3.1.2 |
| packaging | 21.3 |
| pbr | 5.11.1 |
| Pillow | 9.5.0 |
| pip-check | 2.8.1 |
| platformdirs | 3.2.0 |
| pluggy | 1.0.0 |
| prompt-toolkit | 3.0.38 |
| protobuf | 4.23.0 |
| pyasn1 | 0.5.0 |
| pyasn1-modules | 0.3.0 |
| pyflakes | 3.0.1 |
| Pygments | 2.14.0 |
| pylint | 2.17.2 |
| pyparsing | 3.0.9 |
| pytest | 7.3.1 |
| PyYAML | 6 |
| requests | 2.28.2 |
| requests-oauthlib | 1.3.1 |
| rich | 13.3.3 |
| rsa | 4.9 |
| ruamel.yaml | 0.17.21 |
| ruamel.yaml.clib | 0.2.7 |
| safety | 2.3.5 |
| six | 1.16.0 |
| smmap | 5.0.0 |
| stevedore | 5.0.0 |
| terminaltables | 3.1.10 |
| toml | 0.10.2 |
| tomli | 2.0.1 |
| tomlkit | 0.11.7 |
| ttkbootstrap | 1.10.1 |
| typing\_extensions | 4.5.0 |
| uritemplate | 4.1.1 |
| urllib3 | 1.26.15 |
| watchdog | 3.0.0 |
| wcwidth | 0.2.6 |
| wrapt | 1.15.0 |

Majority of 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? PEP8 defines the most essential stylistic guidelines, ranging from naming conventions to indentation norms

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

During the initial test conducted by pylint, the code received a rating of 6.49 out of 10. The test revealed several common linting errors, including bad indentation, invalid variable names, and lines that exceeded the recommended length. The invalid-name error specifically occurred due to the use of camelCase method names, while pylint suggests following snake\_case naming conventions. To address these errors, you can use the command "#pylint: disable=error-name" after the line of code where the error is raised. This command instructs pylint to ignore the specific error and allows you to proceed with your preferred naming style.

\*Note – A well written python code is considered having atleast a score of 8/10

A screen shot of a computer screen

Description automatically generated with low confidence

After some adjustments was able to increase the pylint score uptu 7.45/10 which is considerably a good score for a python script

rating is affected by the readability there code segments like This **if x = = True** were replaced by

**if x is TRUE**

A screenshot of a computer program

Description automatically generated with medium confidence

# Testing for common security issues

The security of the code was assessed using the Bandit and Safety libraries. These tools can be installed by running the commands "pip install bandit" and "pip install safety" respectively. To perform the security tests, you can execute the tool followed by the file to be tested, for example, "bandit main.py".

During the security assessment, the following security issues were identified in the code:

[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 3.10.11

[node\_visitor] WARNING Unable to find qualified name for module: main.py

Run started:2023-05-18 06:20:24.704585

Test results:

No issues identified.

Code scanned:

Total lines of code: 159

Total lines skipped (#nosec): 0

Run metrics:

Total issues (by severity):

Undefined: 0

Low: 0

Medium: 0

High: 0

Total issues (by confidence):

Undefined: 0

Low: 0

Medium: 0

High: 0

Files skipped (0):

The test results indicate that no security issues were identified. It's important to note that these results are based on the code provided, which does not handle any sensitive data. Therefore, the absence of security issues suggests that the code is free from vulnerabilities that could potentially compromise the security of sensitive information.

Although there was a concern which should be paid attention to – This is regarding the storing of the email address for alerting purposes

* the email value was treated as a parameter and passed separately from the query. The ? placeholder in the query represents the parameter. By using parameterized queries, the database engine handles the proper escaping and sanitization of the user input, effectively preventing SQL injection attacks, hence no additional encryptions were carried out as would’ve done when handling sensitive information including passwords and many more

The bandit tool was also used to identify vulnerabilities in the alert.py file the results were as follows

>> Issue: [B105:hardcoded\_password\_string] Possible hardcoded password: 'test123'

Severity: Low Confidence: Medium

CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)

More Info: https://bandit.readthedocs.io/en/1.7.5/plugins/b105\_hardcoded\_password\_string.html

Location: alert.py:16:18

15 email\_receiver = receiver

16 email\_password = '\*\*\*\*\*'

17

--------------------------------------------------

Code scanned:

Total lines of code: 35

Total lines skipped (#nosec): 0

Run metrics:

Total issues (by severity):

Undefined: 0

Low: 1

Medium: 0

Total issues (by confidence):

Undefined: 0

Low: 0

Medium: 1

High: 0

Files skipped (0):

The test results raised a medium security issue this is due to hardcoding the password of the senders email on the python code directly. Can be observed by the highlighted section above

However to resolve this issue password was stored as an environment variable on the Operating system

os.environ.get("APP\_PW")

Special note – to send out emails a special feature names – ‘APP passwords’ which can be used to bypass the multi factor authentications for developing purposes however to send out emails in a more secure way on enterprise level subscriber service providers like sendblue, mailgun & twillio are been used.

## Safety tool analysis

Report

Safety is using pyup's free open-source vulnerability database. This data is 30 days old and limited.

-> c:\program files\windowsapps\pythonsoftwarefoundation.python.3.10\_3.10.3056.0\_x64\_\_qbz5n2kfra8p0\lib\site-packages

-> c:\users\user\appdata\local\packages\pythonsoftwarefoundation.python.3.10\_qbz5n2kfra8p0\localcache\local-packages\python310\site-packages

Using non-commercial database

Found and scanned 70 packages

Timestamp 2023-05-18 12:10:46

1 vulnerability found

0 vulnerabilities ignored

VULNERABILITIES FOUND

==================================================================================

-> Vulnerability found in setuptools version 65.5.0

Vulnerability ID: 52495

Affected spec: <65.5.1

ADVISORY: Python Packaging Authority (PyPA) setuptools before 65.5.1 allows remote attackers to cause a denial of service via HTML in a crafted package or custom PackageIndex page. There is a Regular Expression Denial of

Service (ReDoS) in package\_index.py.https://pyup.io/posts/pyup-discovers-redos-vulnerabilities-in-top-python-packages

* CVE-2022-40897

According to the safety tool as of right now this can only be mitigated by upgrading to an enterprise lisence,

The probability of this tool being affected by CVE-2022-40897 is very low because it does not interact with HTML attributes. This vulnerability typically affects systems or software that involve interactions with HTML attributes, although it is still important to regularly update and patch the tool to address any potential security vulnerabilities that may arise in the future.

# Bugs identified.

* Too many branches
* Too many statements
* Too many local variables
* Abusing the term Global on variables

# Extras

The code below is a test case designed to be executed using pytest. Its goal is to watch the path selection process and save hash values and file names to a text file. A temporary directory and several temporary files are created in order to run this test case. It is important to note that this test case ensures that all temporary files are deleted after execution, ensuring a clean testing environment. This test case was carried out with the objective of gaining additional knowledge about testing tools and methods when programming in Python. Also provided an opportunity to practice and learn about testing frameworks such as pytest, Unit test as well as receive hands-on experience evaluating code functionality and other behaviours in python

import os  
import tempfile  
from unittest.mock import patch  
from tkinter import filedialog  
from main import calculate\_file\_hash  
  
@patch('main.calculate\_file\_hash', return\_value='hash\_value')  
def test\_addPath(mock\_hash, tmpdir):  
 temp\_dir = tmpdir.mkdir("temp\_directory") # Create a temporary directory  
  
 # mock\_askdirectory = patch.object(filedialog, 'askdirectory', return\_value=str(temp\_dir)).start()  
 #mock\_askdirectory.return\_value = str(temp\_dir)  
  
 # Create temporary files inside the temporary directory  
 temp\_file1 = temp\_dir.join("file1.txt")  
 temp\_file2 = temp\_dir.join("file2.txt")  
 temp\_file1.write("Content of file1")  
 temp\_file2.write("")  
  
 # Call calculate\_file\_hash directly without calling addPath  
 hash\_value1 = calculate\_file\_hash(str(temp\_dir), "file1.txt")  
 hash\_value2 = calculate\_file\_hash(str(temp\_dir), "file2.txt")  
  
  
 # assert hash\_value1 == "0adb783b2345f60b24ef878e27e40fadd7cdbd51837ec0e5d98271ceb3e5bf3f71556968416a877f836b861490f0976443cf8625e65847511fb057a7db4cab5b"  
 assert hash\_value2 == "cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f63b931bd47417a81a538327af927da3e"  
  
 # Write file paths and hash values to a text file  
 baseline\_file = "Testbase.txt"  
 with open(baseline\_file, 'w') as f:  
 f.write(f"file1.txt|{hash\_value1}\n")  
 f.write(f"file2.txt|{hash\_value2}\n")  
  
 assert os.path.isfile(baseline\_file)

# Thoughts

A code audit is the most efficient way to improve the code and It helps with habit and practice identification and eradication, code optimization, and project stability and safety in general.

* Unnecessary commented lines should be removed