**Mini Project: Automating SDLC Compliance Check Using Python and Power BI**

**Project Title: SDLC Compliance Automation for IEC 62443 Standards**

**1.Introduction:**

IEC 62443 is an international series of standards that address cybersecurity for operational technology in automation and control systems.

* **Objective:** The purpose of this mini project is to automate the Software Development Life Cycle (SDLC) compliance check against IEC 62443 standards using Python for data collection and vulnerability analysis and Power BI for visualization and reporting. The objective is to streamline and enhance the process of ensuring that software development activities adhere to established standards, regulations, and best practices.
* **Scope:** This project focusses on automating the detection of secure coding practices and generating compliance reports through a Python script and a Power BI dashboard. This confines several key areas, ranging from the initial setup and integration to continuous monitoring and reporting.

**2. Components Required:**

* Python Script for Automation
* Power BI Dashboard

**3. Requirement Analysis:**

**Compliance Standards:**

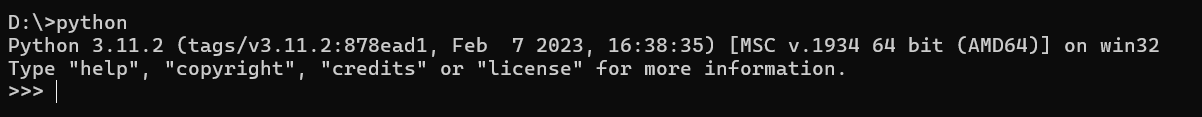
* Coding standards (IEC 62443)
* Security Requirements (OWASP Top 10)

**Tools and Platforms:**

* Git
* Python
* Visual Studio Code
* Power BI Desktop

**4. Setup Instructions:**

**1. Installation of Python and required libraries:** There are several ways to install Python. I used the command prompt to install the latest version of Python. In Command prompt, type python to check if it is already installed in the system or not. If not then when we press Enter, it will automatically launch and take us to the latest version of Python in store.



After this, I required some Python libraries for the project such as “bandit”, “pandas”, “matplotlib”, “requests”.

For installation of libraries, I used the pip command.









**2. Power BI Desktop:**

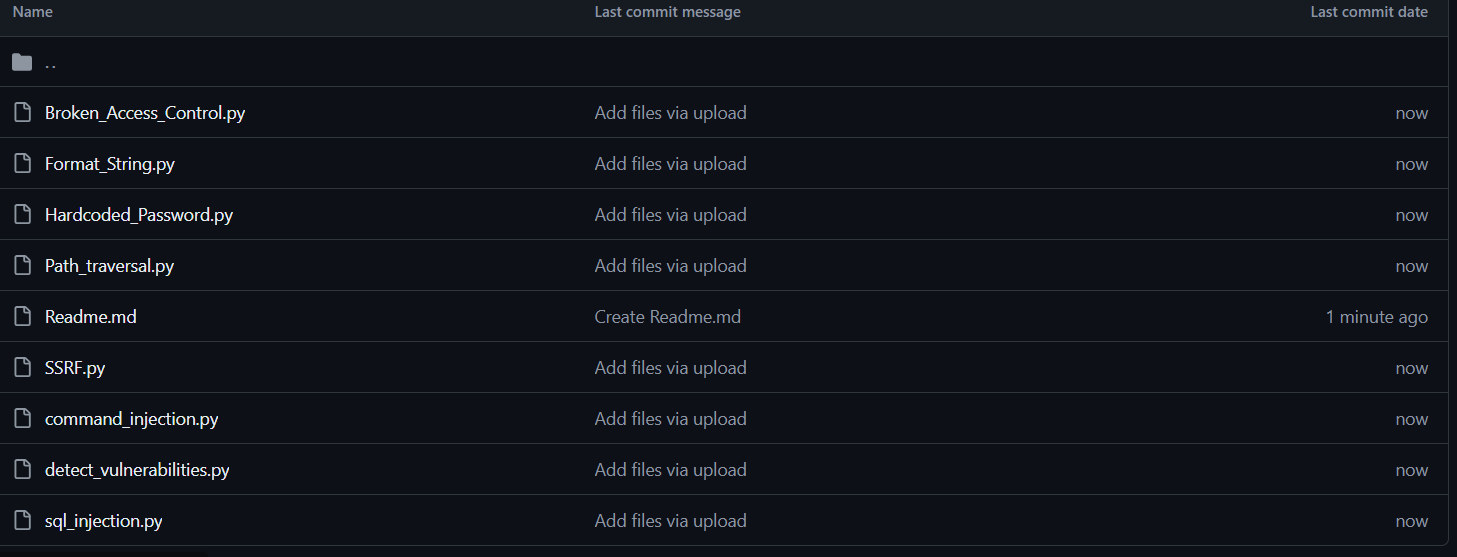
Power BI Desktop is used in this project to build models, and reports to visualize the CSV report created by the Python automation process (more in details in the next section). The Power BI Desktop is a free download and can be downloaded directly from the Microsoft Store in Windows.

**3. GitHub Repository Creation:**

To test the Python automation script, I created a GitHub repository to store a few sample code files that contain security vulnerabilities on purpose.

***https://github.com/DEBANJANAB/Vulnerable-Code-Snippets.git***

A folder named “Code Samples” contains the code files. This can be updated with new codes to test.

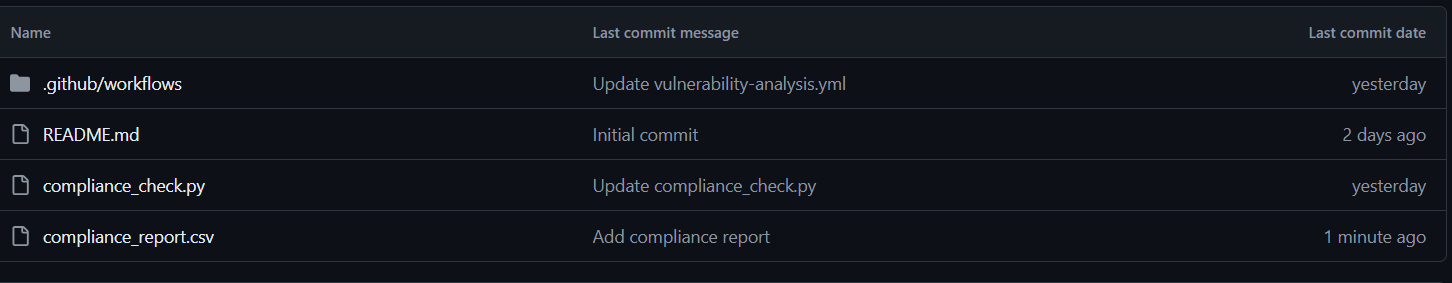


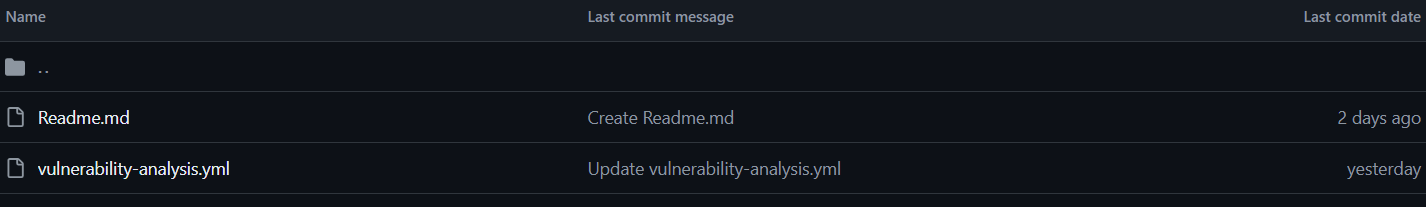
**4. GitHub Actions:**

Additionally, I have developed a GitHub action to automate the process of testing the code samples present in the above-mentioned repository.

[***https://github.com/DEBANJANAB/Vulnerable\_Code\_Automation.git***](https://github.com/DEBANJANAB/Vulnerable_Code_Automation.git)

Here, inside, the. github/workflows, I have added the “**vulnerability\_analysis.yml**” file to run the Python automation code (**compliance\_check.py**) and generate the “**compliance\_report.csv**” file.





**5. Python Script Explanation and Usage Guide:**

The python script “**compliance\_check.py**” automates the process of downloading Python files from a GitHub Repository to scan them for cyber vulnerabilities using Bandit library and saves the results in a CSV report.

A detailed explanation of the script goes as follows:

**Functions:**

1. **get\_file\_list\_recursive(url, file\_list = [])**

* Recursively fetches all file URLs from a GitHub repository directory.
* **Arguments**:

**url (str):** API-URL to the directory on GitHub repository.

**file\_list**: A list to store the file URLs

* **Returns**:

**list**: A list of file URLs in the repository directory.

1. **get\_first\_level\_files(url)**

* Gets the URLs of all files in the top level of a GitHub repository directory.
* **Arguments**:

**url (str):** The API URL of a directory in the GitHub repository.

* **Returns**:

**list:** A list of file URLs in the top level of the repository directory**.**

1. **download\_files(repo\_url, local\_dir)**

* Downloads Python files to local directory from a GitHub repository.
* **Arguments**:

**repo\_url (str):** The API URL of the GitHub repository.

**local\_dir (str):** The local directory to save the downloaded files.

1. **run\_bandit\_on\_file(b\_mgr, file\_path)**

* This function runs Bandit security analysis on a Python file.
* **Arguments:**

**b\_mgr (BanditManager):** The Bandit manager instance.

**file\_path (str):** Path to the Python file for Analysis.

* **Returns**:

**list**: A list of issues detected in the file.

1. **scan\_directory(directory)**

* Scans all Python files in a directory using Bandit for security issues.
* **Arguments:**

**directory (str):** The directory containing Python files to scan.

* **Returns:**

**list:** A list of issues found in the directory.

1. **format\_issue(issue)**

* This function formats a Bandit issue into a dictionary for CSV output.
* **Arguments:**

**issue (Issue):** A Bandit issue instance.

* **Returns:**

**dict:** A dictionary containing formatted issue information like Filename, Line Number, Severity, Confidence and Issue Text.

1. **save\_compliance\_report(issues, output\_file)**

* Saves the unique issues to a CSV file using pandas.
* **Arguments:**

**issues (list):** A list of Bandit issues.

**output\_file (str):** The path to the output CSV file.

1. **convert\_github\_url\_to\_api(url)**

* Converts a GitHub repository URL to the corresponding API URL.
* **Arguments:**

**url (str):** The GitHub repository URL.

* **Returns:**

**str:** The GitHub API URL.

* **Raises:**

**ValueError:** If the provided URL is not a valid GitHub URL.

1. **main(repo\_url, directory="temp")**

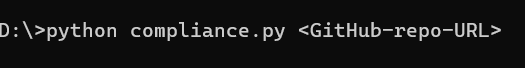
* Main function needed to download files, scan for vulnerabilities, and save the report.
* **Arguments:**

**repo\_url (str):** The GitHub repository API URL.

**directory (str):** The local directory to save the downloaded files.

**Usage Guide:**

Open a command prompt or terminal and navigate to the directory where the script is stored. Run the script with the GitHub repository URL as an argument.



In this case,

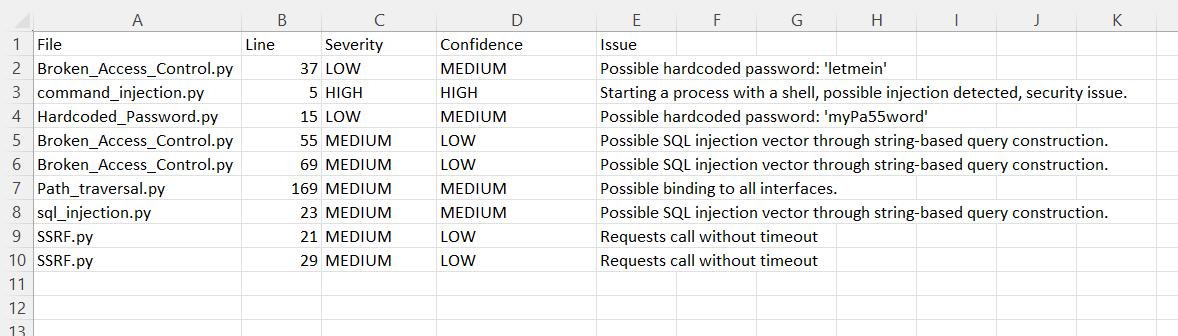


The script will convert the Github repository URL provided in the command line argument to its corresponding API URL using ‘**convert\_github\_url\_to\_api(url)’** function. It will then call the ‘**main’** function to:

* Download Python files from the repository.
* Scan the downloaded files for compliance issues using **Bandit.**
* Arrange and save the resulting report in a CSV file named “**compliance\_report.csv**” in the current directory.

**6. Power BI Dashboard Design and Usage Guide:**

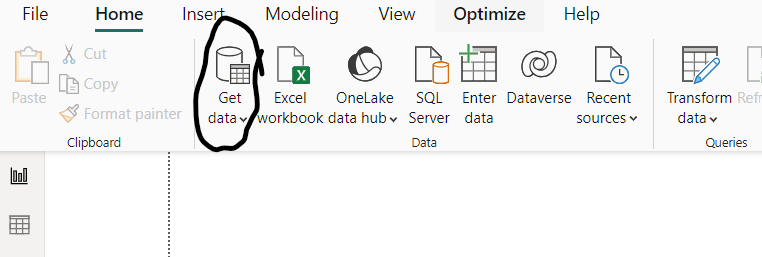
The “compliance\_report.csv” file downloaded from the last section has the following structure.



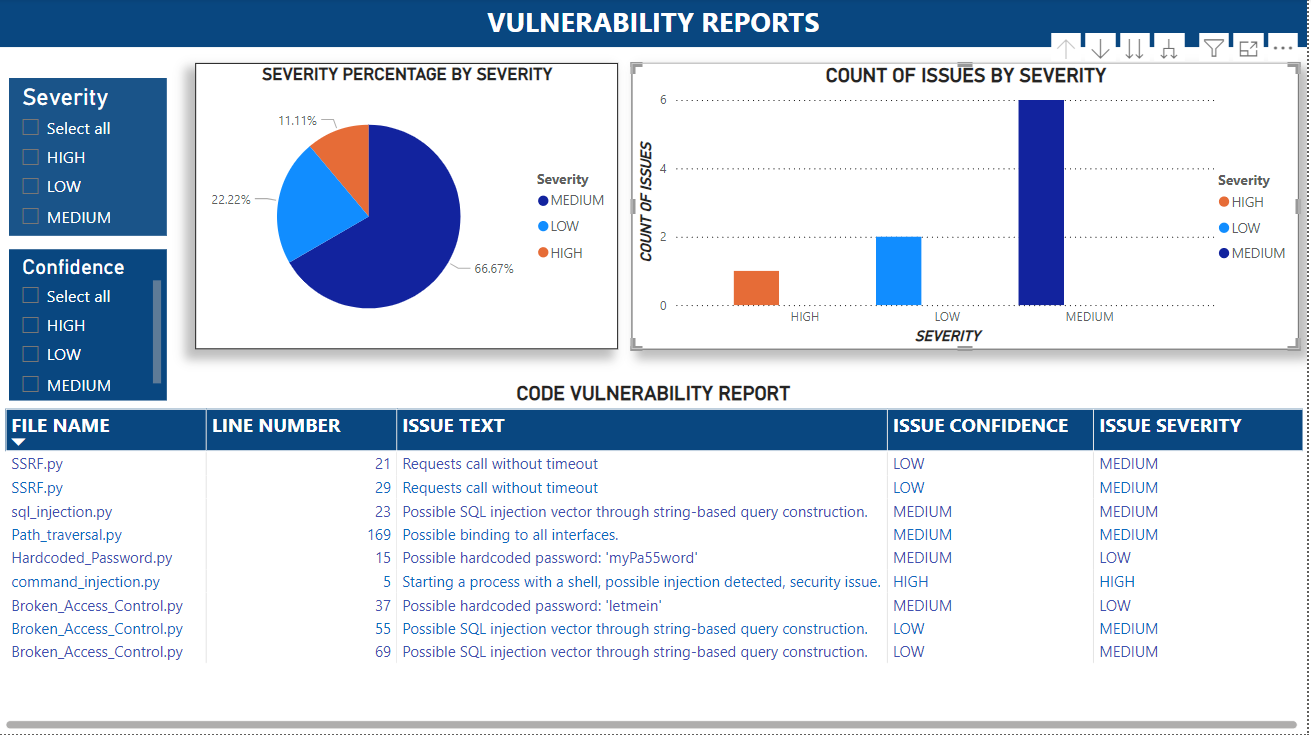
This contains the code filename, the line number where vulnerability is detected, the issue severity, the issue confidence and the description of the security issue.

This compliance data needs to be visualized for proper analysis and reporting of the vulnerabilities. **Power BI Dashboard** serves that purpose.

The “compliance\_report.csv” file has been imported into Power BI Desktop as a new data source.



The “SDLC\_Compliance\_Dashboard.pbix” Power BI File has been designed with several interactive elements.



The components of the dashboard are as follows:

1. **FILTERS (LEFT):**

* **Severity Filter:**

This slicer filters data based on their severity levels (HIGH, MEDIUM, LOW).

* **Confidence Filter:**

This slicer filters data based on the confidence levels of the issues (HIGH, MEDIUM, LOW).

1. **VISUALISATIONS (TOP):**

* **Bar Graph**:

A clustered column chart or bar graph is created displaying the count of vulnerabilities by each severity level. (High, Medium, Low).

The count of vulnerabilities was not present in the csv file where I had to utilize the DAX Query editor to create a new data column called “Count of Issues”.

The query is as follows:

COUNT(compliance\_report[Severity])

* **Pie Chart:**

A Pie Chart is created to show the percentage distribution of issues by their severity levels.

The percentage distribution of issues is calculated using the DAX Query Editor.

The query is : DIVIDE(COUNT(compliance\_report[Severity]),COUNTROWS(ALLNOBLANKROW(compliance\_report)))

1. **DETAIL TABLE (BOTTOM):**

This is a detailed table of the data collected from the CSV file.

The columns are as follows:

* **File Name**: Name of the Python file where vulnerability is detected.
* **Line Number**: The line number in that specific file with issue.
* **Issue Text**: A brief overview of the issue.
* **Issue Confidence**: Confidence level of the detected issue.
* **Issue Severity**: Severity level of the detected issue.

**USAGE DETAILS:**