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

Project Title: SDLC Compliance Automation for IEC 62443 Standards

Submitted by: Debanjona Bhattacharjya

CONTENT

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:

a) **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.

```
D:\>python
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

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

For installation of libraries, I used the pip command.

```
pip install bandit
```

```
pip install requests
```

```
pip install matplotlib
```

```
pip install pandas
```

b) Power BI Desktop:

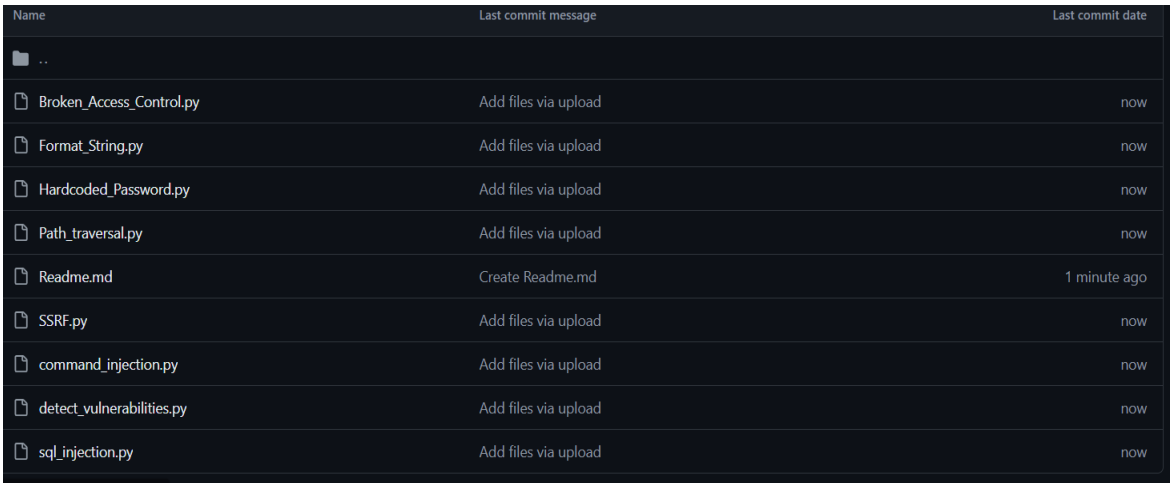
Power BI Desktop is used in this project to build models, and reports to visualize the CSV report created by the Python automation process (more in details in the next section). The Power BI Desktop is a free download and can be downloaded directly from the Microsoft Store in Windows.

c) 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.



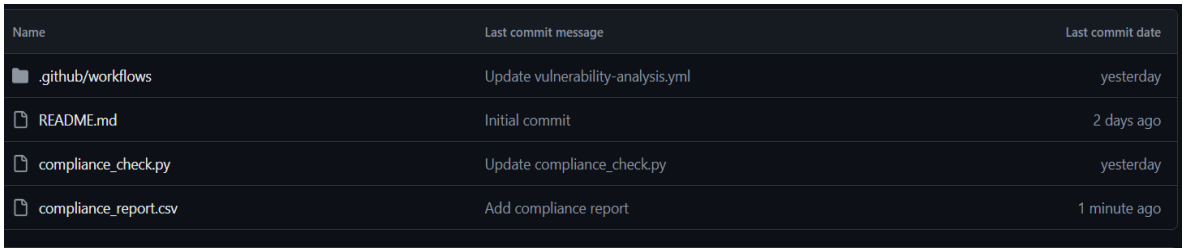
Name	Last commit message	Last commit date
..		
Broken_Access_Control.py	Add files via upload	now
Format_String.py	Add files via upload	now
Hardcoded_Password.py	Add files via upload	now
Path_traversal.py	Add files via upload	now
Readme.md	Create Readme.md	1 minute ago
SSRF.py	Add files via upload	now
command_injection.py	Add files via upload	now
detect_vulnerabilities.py	Add files via upload	now
sql_injection.py	Add files via upload	now

d) GitHub Actions:

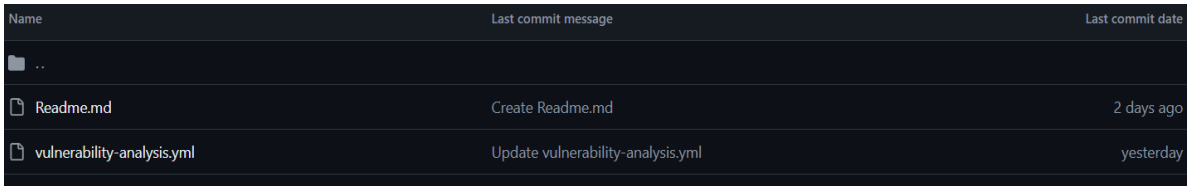
Additionally, I have developed a GitHub action to automate the process of testing the code samples present in the above-mentioned repository.

https://github.com/DEBANJANAB/Vulnerable_Code_Automation.git

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.



Name	Last commit message	Last commit date
.github/workflows	Update vulnerability-analysis.yml	yesterday
README.md	Initial commit	2 days ago
compliance_check.py	Update compliance_check.py	yesterday
compliance_report.csv	Add compliance report	1 minute ago



Name	Last commit message	Last commit date
..		
Readme.md	Create Readme.md	2 days ago
vulnerability-analysis.yml	Update vulnerability-analysis.yml	yesterday

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:

A) 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.

2. ***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**.**

3. ***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.

4. ***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.

5. ***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.

6. ***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.

7. ***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.

8. ***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.

9. ***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.

B) 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.

```
D:\>python compliance.py <GitHub-repo-URL>
```

In this case,

```
D:\>python compliance_check.py https://github.com/DEBANJANAB/Vulnerable-Code-Snippets
```

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:

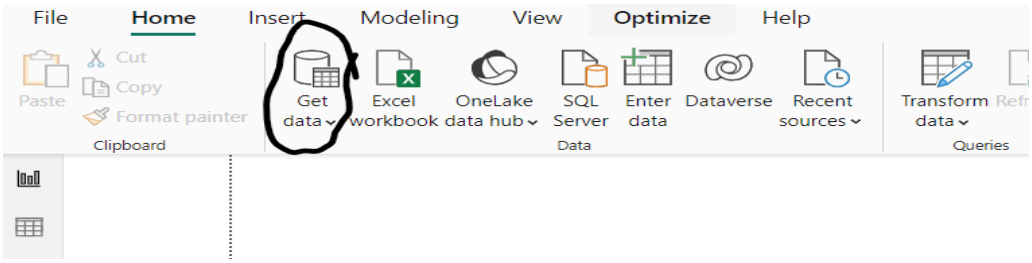
The "compliance_report.csv" file downloaded from the last section has the following structure.

	A	B	C	D	E
1	File	Line	Severity	Confidence	Issue
2	Broken_Access_Control.py	37	LOW	MEDIUM	Possible hardcoded password: 'letmein'
3	command_injection.py	5	HIGH	HIGH	Starting a process with a shell, possible injection detected, security issue.
4	Hardcoded_Password.py	15	LOW	MEDIUM	Possible hardcoded password: 'myPa55word'
5	Broken_Access_Control.py	55	MEDIUM	LOW	Possible SQL injection vector through string-based query construction.
6	Broken_Access_Control.py	69	MEDIUM	LOW	Possible SQL injection vector through string-based query construction.
7	Path_traversal.py	169	MEDIUM	MEDIUM	Possible binding to all interfaces.
8	sql_injection.py	23	MEDIUM	MEDIUM	Possible SQL injection vector through string-based query construction.
9	SSRF.py	21	MEDIUM	LOW	Requests call without timeout
10	SSRF.py	29	MEDIUM	LOW	Requests call without timeout
11					
12					
13					

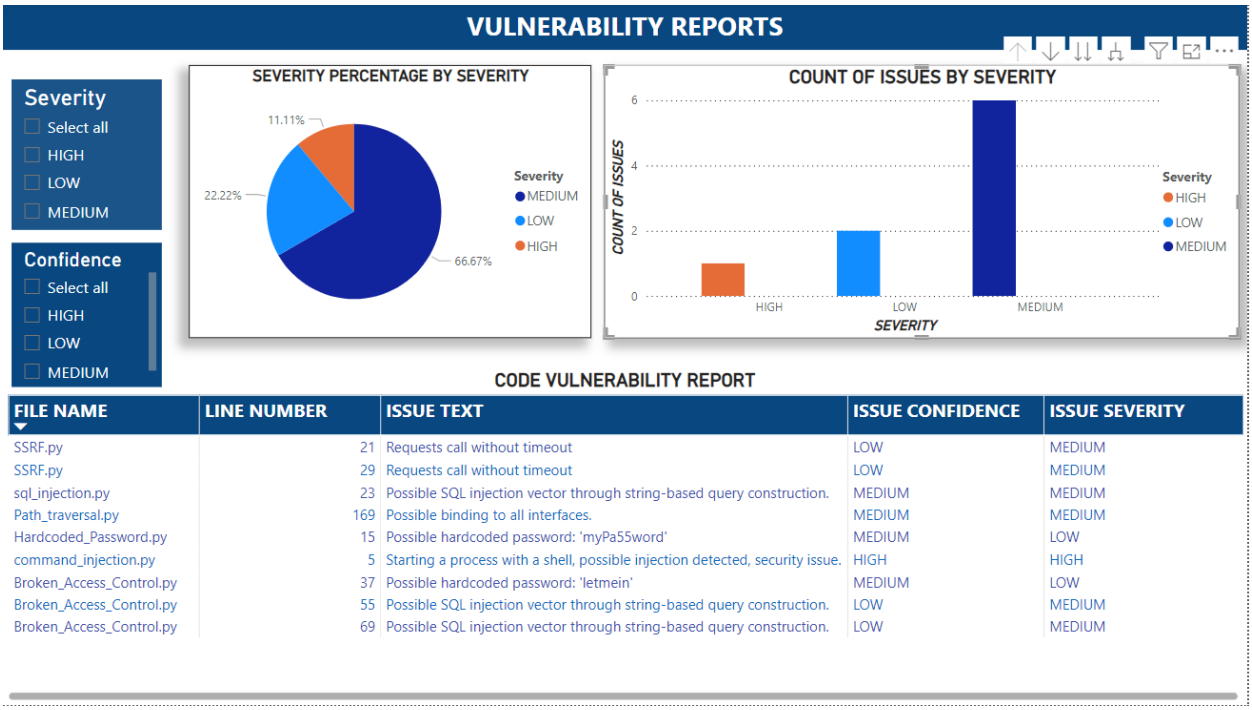
This contains the code filename, the line number where vulnerability is detected, the issue severity, the issue confidence and the description of the security issue.

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 using "Get data".



The "**SDLC_Compliance_Dashboard.pbix**" Power BI File has been designed with several interactive elements.



FILE NAME	LINE NUMBER	ISSUE TEXT	ISSUE CONFIDENCE	ISSUE SEVERITY
SSRF.py	21	Requests call without timeout	LOW	MEDIUM
SSRF.py	29	Requests call without timeout	LOW	MEDIUM
sql_injection.py	23	Possible SQL injection vector through string-based query construction.	MEDIUM	MEDIUM
Path_traversal.py	169	Possible binding to all interfaces.	MEDIUM	MEDIUM
Hardcoded_Password.py	15	Possible hardcoded password: 'myPa55word'	MEDIUM	LOW
command_injection.py	5	Starting a process with a shell, possible injection detected, security issue.	HIGH	HIGH
Broken_Access_Control.py	37	Possible hardcoded password: 'letmein'	MEDIUM	LOW
Broken_Access_Control.py	55	Possible SQL injection vector through string-based query construction.	LOW	MEDIUM
Broken_Access_Control.py	69	Possible SQL injection vector through string-based query construction.	LOW	MEDIUM

A) Dashboard Design:

➢ **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).

➢ **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)))
```

➢ **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.

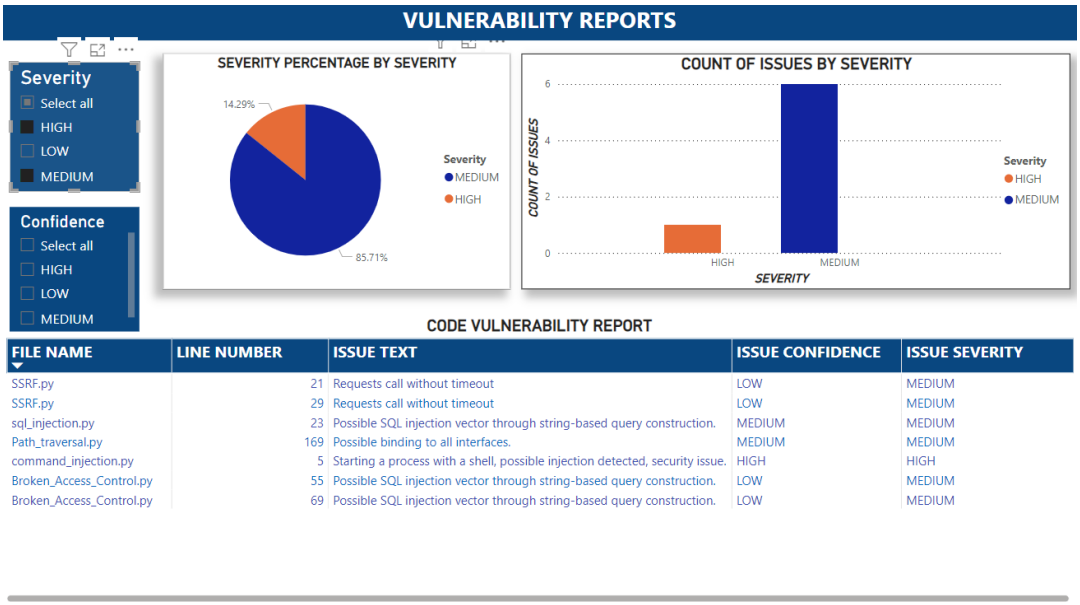
B) Usage Details:

This is a detailed guide to use the "SDLC_Compliance_Dashboard.pbix" for visualization and analysis.

➢ FILTER (LEFT SIDE):

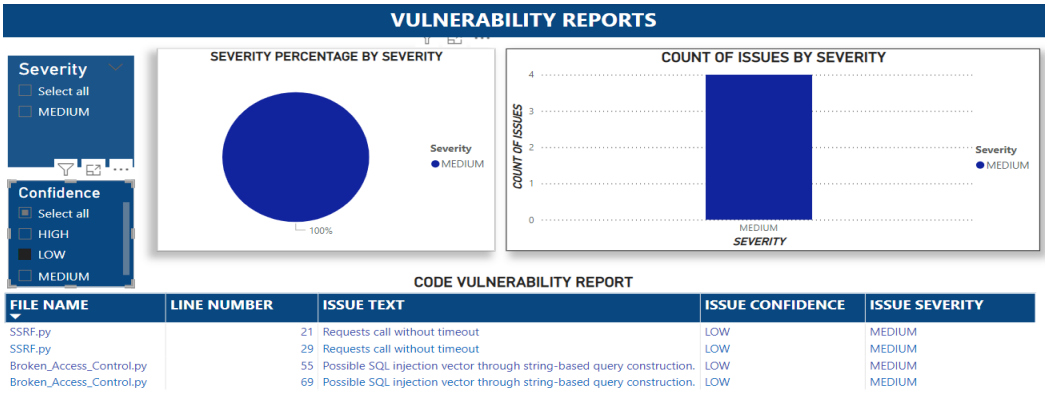
- **Severity Filter:**

User can check the boxes to select which severity level they want to include in the report. For example, in the below diagram, MEDIUM AND HIGH are selected which updates the visualizations and table report to show only high and medium issues.

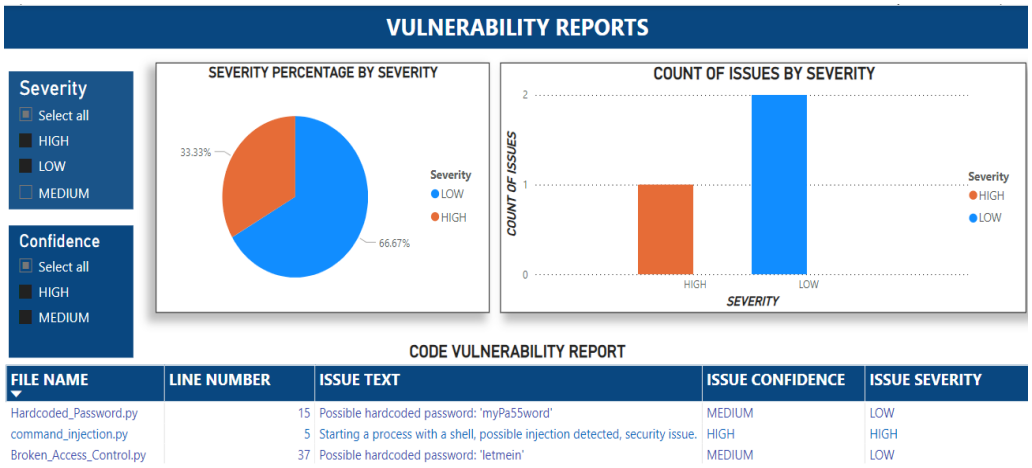


FILE NAME	LINE NUMBER	ISSUE TEXT	ISSUE CONFIDENCE	ISSUE SEVERITY
SSRF.py	21	Requests call without timeout	LOW	MEDIUM
SSRF.py	29	Requests call without timeout	LOW	MEDIUM
sql_injection.py	23	Possible SQL injection vector through string-based query construction.	MEDIUM	MEDIUM
Path_traversal.py	169	Possible binding to all interfaces.	MEDIUM	MEDIUM
command_injection.py	5	Starting a process with a shell, possible injection detected, security issue.	HIGH	HIGH
Broken_Access_Control.py	55	Possible SQL injection vector through string-based query construction.	LOW	MEDIUM
Broken_Access_Control.py	69	Possible SQL injection vector through string-based query construction.	LOW	MEDIUM

- **Confidence Filter:**
  With this slicer, the user can select the confidence levels they require to put in the report. For example, in this case, LOW is selected which updates the visualizations with only low confidence level issues.

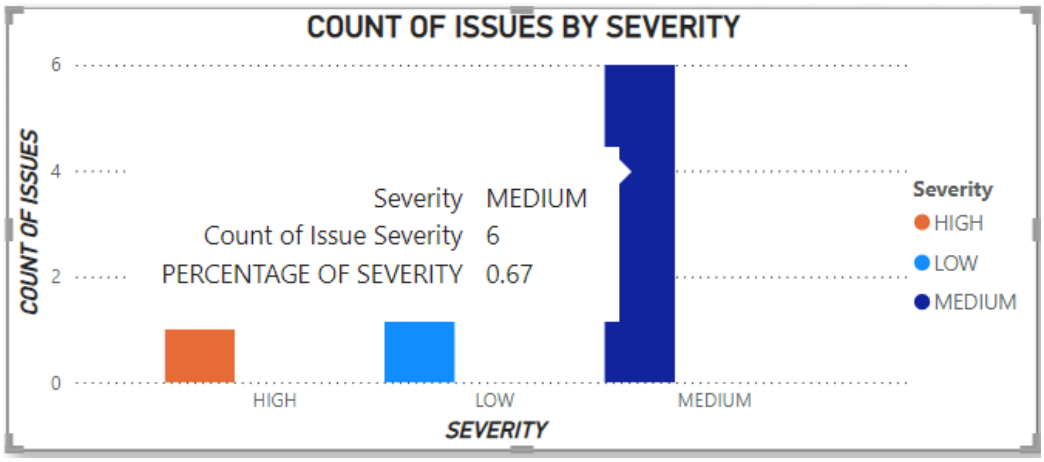


Additionally, one call also filter both Severity and Confidence slicers for different combinations of data.

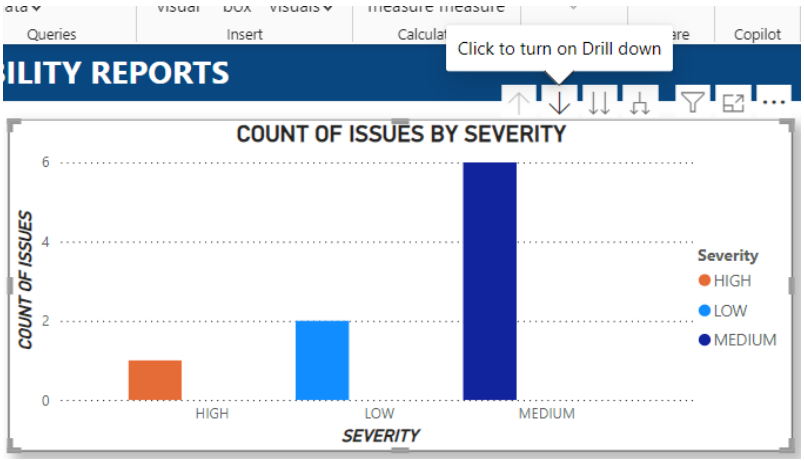


➢ Assessing Visualizations:

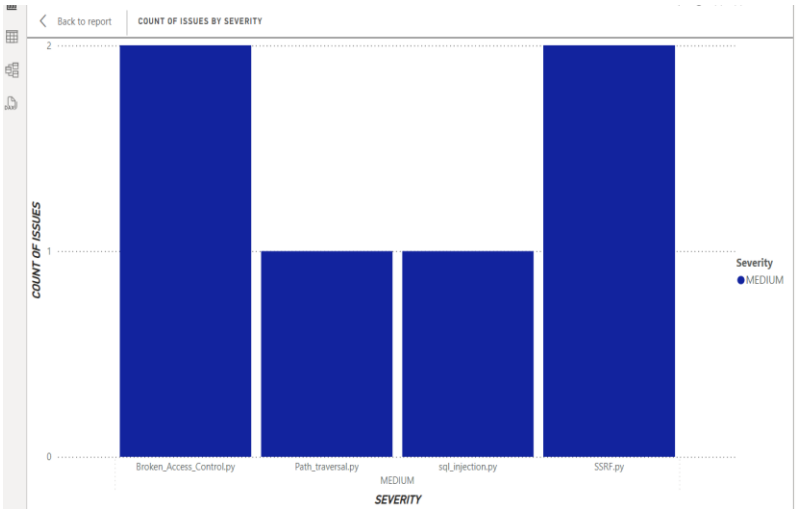
- **Bar Chart:**
  The bar chart shows the count of issue severity by the severity levels.
  If one hovers to any of the levels (HIGH, MEDIUM OR LOW), it will show details like the Severity, Count of Severity and Percentage of Severity.



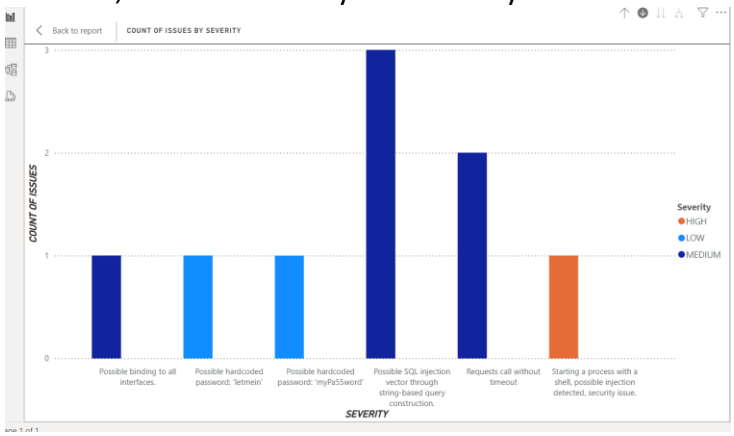
The **drill-down functionality** is also included in the bar chart. One needs to click on the drill-down icon in the top-right to start the process.



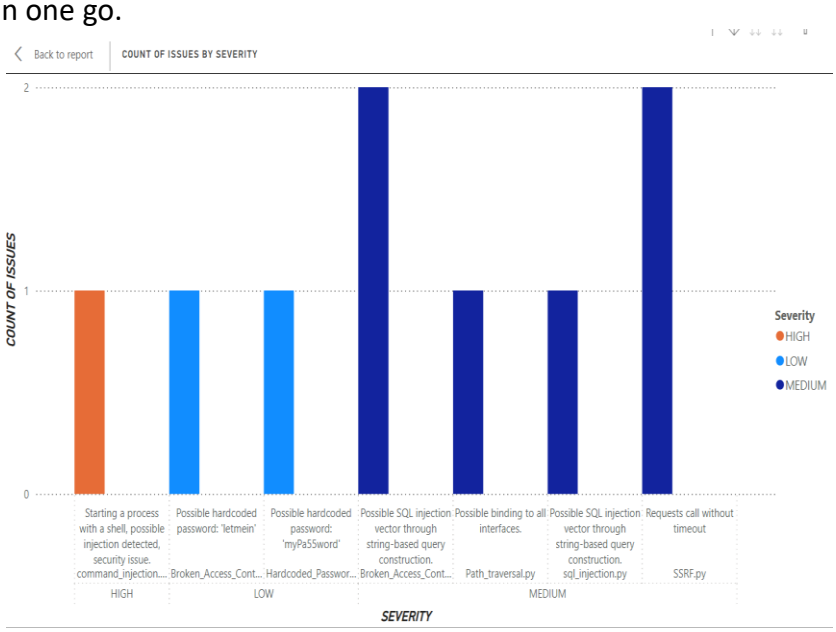
Once the drill-down mode is ON, one can click on any data point in the bar graph to show more details about the severity. For example, here I have clicked on MEDIUM which shows all the files containing medium severity issues along with the count of medium vulnerabilities.



There is the “Go to the next level in Hierarchy” option in the drill-down menu which will show the Issue text along with the Filename, count of Severity and Severity in details.

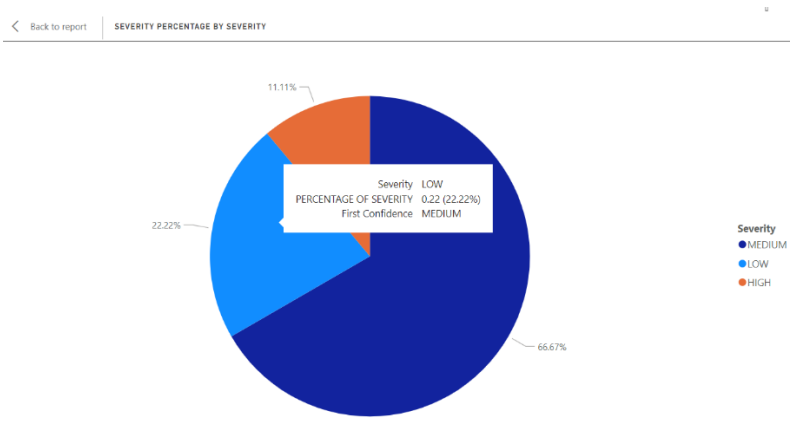


Additionally, “Expand all down one level in the Hierarchy” can be utilized to expand the entire hierarchy level in one go.



This drill-down functionality is particularly useful to prioritize issues or Filenames and to address the vulnerabilities accordingly.

- **Pie Chart:**
  The Pie Chart provides a quick glance at the percentage of issues by severity levels. One can hover over the sections to see the percentage and count of issues.



➢ Reading the Vulnerability Report Table:

The report table provides help for a detailed inspection of the data for specific details about each vulnerability. For instance, the **File Name** and the **Line Number** can be used to locate the issue in the codebase. Additionally, details about the issue in the **Issue text** column will make it easier for the developer to understand the issue and address it.

CODE VULNERABILITY REPORT

FILE NAME	LINE NUMBER	ISSUE TEXT	ISSUE CONFIDENCE	ISSUE SEVERITY
SSRF.py	21	Requests call without timeout	LOW	MEDIUM
SSRF.py	29	Requests call without timeout	LOW	MEDIUM
sql_injection.py	23	Possible SQL injection vector through string-based query construction.	MEDIUM	MEDIUM
Path_traversal.py	169	Possible binding to all interfaces.	MEDIUM	MEDIUM
Hardcoded_Password.py	15	Possible hardcoded password: 'myPa55word'	MEDIUM	LOW
command_injection.py	5	Starting a process with a shell, possible injection detected, security issue.	HIGH	HIGH
Broken_Access_Control.py	37	Possible hardcoded password: 'letmein'	MEDIUM	LOW
Broken_Access_Control.py	55	Possible SQL injection vector through string-based query construction.	LOW	MEDIUM
Broken_Access_Control.py	69	Possible SQL injection vector through string-based query construction.	LOW	MEDIUM

7. Testing and Results:

➢ Test Cases:

The test cases for this automation script is provided in the GitHub Repository.

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

The vulnerabilities include HardCoded Passwords, SQL Injection, Command Injection, Broken Access Control, Path Traversal and ServerSide Request Forgery Attack.

➢ Testing:

To test the script, the Python script “compliance_check.py” is executed in the same directory where it is stored. The Github Repository is provided as a command line argument.

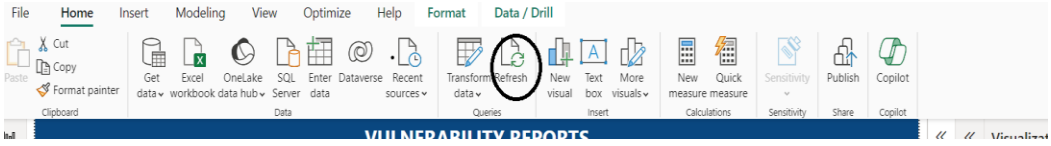
python compliance_check.py https://github.com/DEBANJANAB/Vulnerable-Code-Snippets

This generates the “compliance_report.csv” file in the same directory.

The CSV file is imported to the Power BI Desktop for visualization.

If some code files are added or deleted from the codebase repository or if the testing is to be done on another repository, the Python script needs to be executed again using the steps given above.

Additionally, the data in Power BI needs to be refreshed to reflect the latest compliance report.



➢ Testing Results:

On execution of the Python script, the “compliance_report.csv” is created. Here are the testing results:

File	Line	Severity	Confidence	Issue
Broken_Access_Control.py	37	LOW	MEDIUM	Possible hardcoded password: 'letmein'
command_injection.py	5	HIGH	HIGH	Starting a process with a shell, possible injection detected, security issue.
Hardcoded_Password.py	15	LOW	MEDIUM	Possible hardcoded password: 'myPa55word'
Broken_Access_Control.py	55	MEDIUM	LOW	Possible SQL injection vector through string-based query construction.
Broken_Access_Control.py	69	MEDIUM	LOW	Possible SQL injection vector through string-based query construction.
Path_traversal.py	169	MEDIUM	MEDIUM	Possible binding to all interfaces.
sql_injection.py	23	MEDIUM	MEDIUM	Possible SQL injection vector through string-based query construction.
SSRF.py	21	MEDIUM	LOW	Requests call without timeout
SSRF.py	29	MEDIUM	LOW	Requests call without timeout

There are 3 Testing scenarios which is verified by Manual Testing.

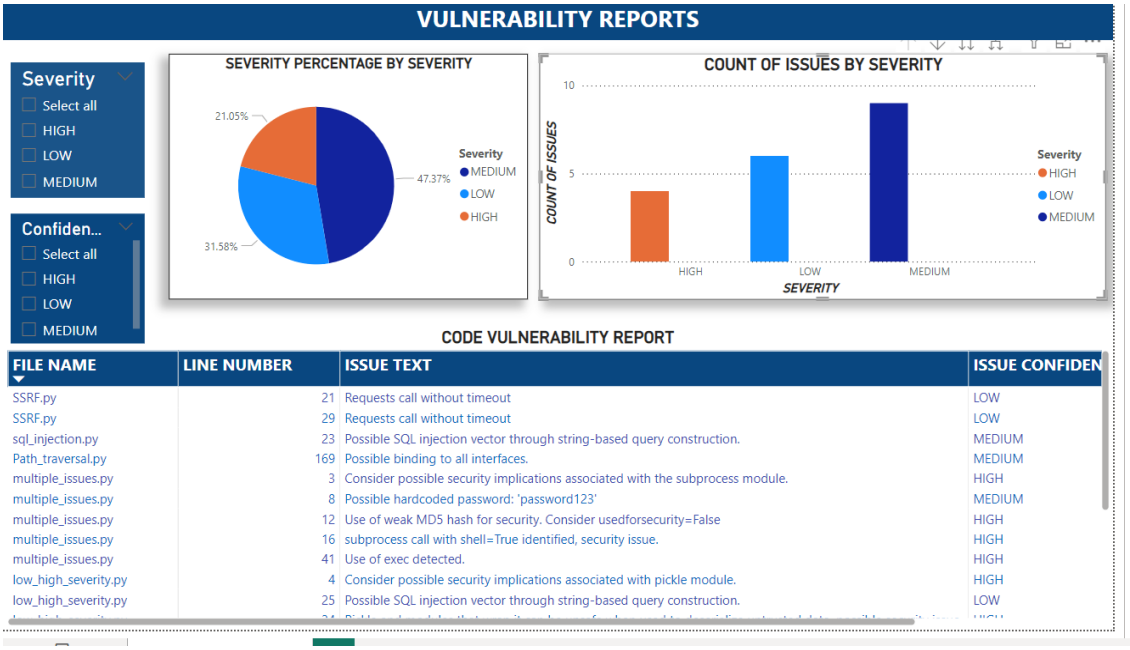
Sr. No.	Description	Expected result	Actual result
1	Run Bandit on Python scripts with SQL injection manually.	Bandit should report a possible SQL injection issue.	Worked as expected.
2	Check the file download functionality	File should be downloaded to the specific directory	Worked as expected
3	Successful Compliance Report Generation	CSV file should be formatted properly with issues	CSV file is generated with accurate data

To perform more testing, I have added new Python files with vulnerability in the repository.

I executed the Python script again and the results are as follows:

File	Line	Severity	Confidenc	Issue
multiple_is	12	HIGH	HIGH	Use of weak MD5 hash for security. Consider usedforsecurity=False
multiple_is	8	LOW	MEDIUM	Possible hardcoded password: 'password123'
multiple_is	41	MEDIUM	HIGH	Use of exec detected.
multiple_is	3	LOW	HIGH	Consider possible security implications associated with the subprocess module.
SSRF.py	29	MEDIUM	LOW	Requests call without timeout
Broken_Ac	37	LOW	MEDIUM	Possible hardcoded password: 'letmein'
Broken_Ac	69	MEDIUM	LOW	Possible SQL injection vector through string-based query construction.
command_	5	HIGH	HIGH	Starting a process with a shell, possible injection detected, security issue.
hardcoded	1	LOW	MEDIUM	Possible hardcoded password: '12345'
low_high_	4	LOW	HIGH	Consider possible security implications associated with pickle module.
low_high_	38	HIGH	HIGH	Starting a process with a shell, possible injection detected, security issue.
Broken_Ac	55	MEDIUM	LOW	Possible SQL injection vector through string-based query construction.
Hardcodec	15	LOW	MEDIUM	Possible hardcoded password: 'myPa55word'
low_high_	25	MEDIUM	LOW	Possible SQL injection vector through string-based query construction.
low_high_	34	MEDIUM	HIGH	Pickle and modules that wrap it can be unsafe when used to deserialize untrusted data, possible security issue.
Path_trave	169	MEDIUM	MEDIUM	Possible binding to all interfaces.
sql_injecti	23	MEDIUM	MEDIUM	Possible SQL injection vector through string-based query construction.
multiple_is	16	HIGH	HIGH	subprocess call with shell=True identified, security issue.
SSRF.py	21	MEDIUM	LOW	Requests call without timeout

On refreshing the data in Power BI Desktop, it represented the updated compliance dashboard.



8. Conclusion:

This mini project shows how one can automate the checking of SDLC compliance of their code by using IEC 62443 standards. Taking advantage of Python (for secure and seamless data collection and analysis) and Power BI (for interactive visualization and reporting), the project scales well providing an effective cybersecurity tool for industrial automation system. This automation increases the accuracy and consistency of compliance checking for industrial operations.

Future work could include extending the scope to more programming languages, using machine learning for a and including support to other compliance standards and thus building a security framework.