Project Report: Secure Code Review

NAVNEET BIJALWAN

Executive Summary

The Secure Code Review project aims to enhance the security posture of web applications developed using Python and the Flask framework. With the increasing number of cyber threats targeting web applications, it is imperative to adopt secure coding practices to mitigate risks and vulnerabilities. This report outlines the objectives, methodologies, and findings from conducting a secure code review process on a sample Flask application. Utilizing tools such as Bandit and PyLint, alongside manual code review techniques, we have identified several security vulnerabilities and provided recommendations for best practices in secure coding. The findings emphasize the importance of continuous improvement in coding practices to adapt to evolving security threats.

Project Objectives

The primary objectives of the Secure Code Review project are as follows:

- **Identify Security Vulnerabilities:** Conduct a thorough examination of a Python Flask application to identify security vulnerabilities using both automated tools and manual review techniques.

- **Utilize Static Code Analysis Tools:** Employ tools like Bandit and PyLint to automatically identify security flaws and code quality issues within the application.

- **Implement Best Practices:** Recommend secure coding practices based on identified vulnerabilities to improve the overall security of the application.

- **Foster Continuous Improvement:** Emphasize the need for ongoing research and development in secure coding practices to keep pace with emerging security threats**.**

Project Scope

The scope of this project includes:

- A comprehensive examination of the secure code review process, detailing its stages and methodologies.
- Conducting code reviews in a **Kali Linux** environment, which is well-suited for security testing and analysis.
- The development of a sample web application using **Python** and Flask for the purpose of review.
- The use of static code analysis tools like **Bandit and Pylint** to identify security vulnerabilities in the Python code.
- Manual code review tools like **Flask** and Pair Programming to uncover vulnerabilities that automated tools may miss.
- An overview of additional tools and technologies that support secure code review, including PyLint for code quality checks and platforms like GitHub for collaboration.
- A discussion of best practices for effective implementation, focusing on input validation, output encoding, and secure error handling.
- The inclusion of data visualizations to illustrate vulnerability frequency, trends, and severity classifications.
- Recommendations for future research and development in secure coding practices.

Key Deliverables

- **Code Review Process Documentation:** A detailed description of the code review process, including automated and manual techniques.

- **Vulnerability Assessment Report:** A list of identified vulnerabilities, complete with severity ratings, confidence levels, CWE classifications, code locations, and recommendations for remediation.

- **Best Practices Guide:** A compilation of secure coding practices tailored to mitigate the identified vulnerabilities.

- **Visual Data Representations**: Charts and graphs illustrating the frequency and severity of vulnerabilities found during the review process.

- **Conclusion and Future Recommendations:** A summary of findings and suggestions for future improvements in secure coding practices.

1. Introduction

In today's digital landscape, the security of web applications is paramount. With an increase in data breaches and cyberattacks, it is crucial for developers to adopt secure coding practices to protect sensitive information and maintain user trust. This project focuses on conducting a secure code review for a web application built using Python and the Flask framework. The primary purpose of this project is to identify security vulnerabilities and provide actionable recommendations to improve the security of the application.

Python is a popular programming language known for its simplicity and versatility, making it an ideal choice for web development. Flask, a lightweight web framework for Python, allows developers to build web applications quickly and efficiently. However, like any technology, Flask applications can be susceptible to security vulnerabilities if not developed with best practices in mind.

This report will detail the methodologies employed during the secure code review process, the vulnerabilities identified, and recommendations for secure coding practices. By integrating automated tools and manual review techniques, the project aims to provide a comprehensive assessment of the application's security posture.

2. Code Review Process for a Python Flask Application

The code review process for a Python Flask application involves multiple stages, combining automated tools and manual review techniques to ensure a thorough examination of the codebase.

Automated Code Review:

- **Static Code Analysis with Bandit:** Bandit is a security-focused static analysis tool that scans Python code for security vulnerabilities. It identifies issues such as hardcoded passwords, insecure use of functions, and potential injection vulnerabilities.

The tool generates a report highlighting the vulnerabilities found, categorized by severity.

- **Code Quality Checks with PyLint:** PyLint is a static code analysis tool that checks code quality issues, such as coding standards violations, potential bugs, and maintainability concerns. While its primary focus is not on security, it can help identify areas of the code that may be prone to vulnerabilities.

➢ <u>Manual Code Review:</u>

In addition to automated tools, manual code reviews are essential for identifying vulnerabilities that may not be detected by static analysis. The manual review process includes:

- **Reviewing Code Logic:** Assessing the logic of the application to identify potential flaws or insecure coding practices.

- **Checking Input Validation:** Ensuring that all user inputs are properly validated and sanitized to prevent injection attacks.

- **Analyzing Error Handling:** Reviewing how the application handles errors to avoid exposing sensitive information.

➢ <u>Documentation and Reporting:</u>

After completing the code review, a comprehensive report is generated, summarizing the findings and providing recommendations for remediation.

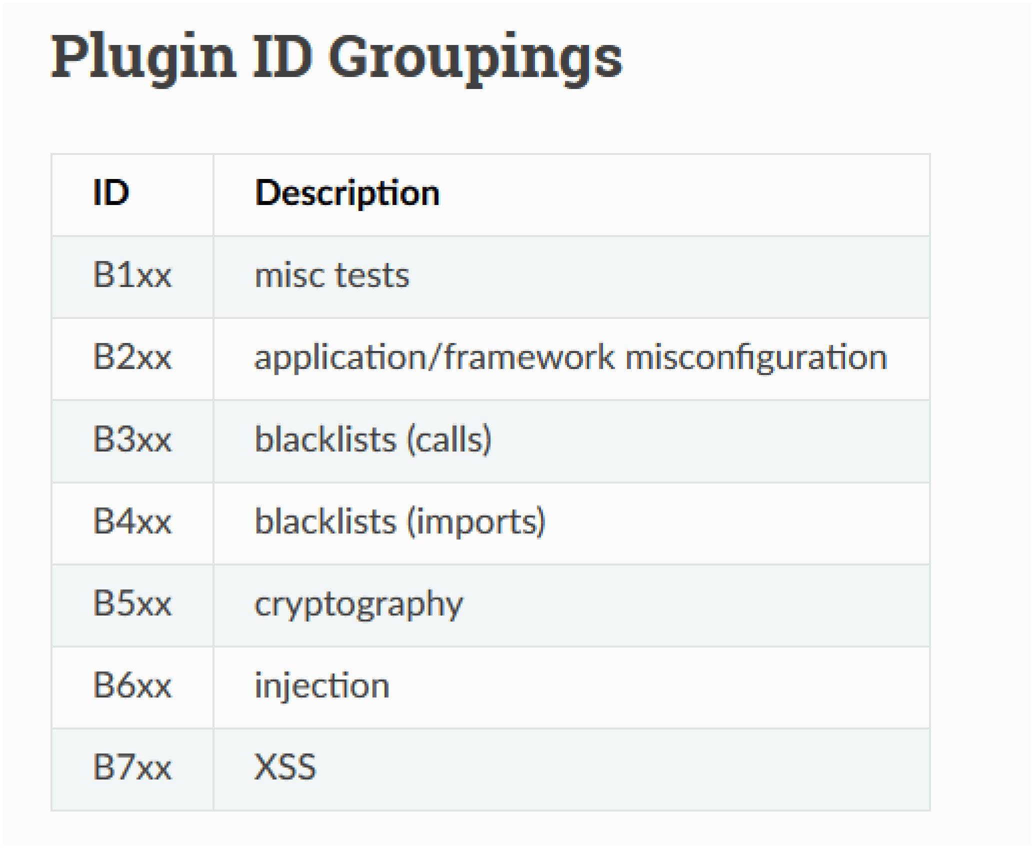
Tools:

➢ Bandit

Bandit is an open-source security tool used to detect security vulnerabilities in Python projects. It analyzes potential security risks in your Python code and alerts you against these vulnerabilities.
Bandit is used to identify and detect security vulnerabilities in Python code.

<u>What does is look for</u>

1. Bandit aims to identify passwords or other sensitive information hardcoded in the code
2. It identifies usages that may pose SQL or OS injection risks.
3. It checks for known security vulnerabilities in external libraries used by your project.
4. Some Python functions or modules can introduce security risks when misused. Bandit attempts to detect such situations.



Plugin ID Groupings

ID	Description
B1xx	misc tests
B2xx	application/framework misconfiguration
B3xx	blacklists (calls)
B4xx	blacklists (imports)
B5xx	cryptography
B6xx	injection
B7xx	XSS

➢ Pylint

Pylint serves as a static analysis tool designed to evaluate the quality and compliance of code written in the Python programming language.

Purpose and Functions

1. **Code Quality Assessment:** Pylint evaluates the quality of Python coding and checks its adherence to generally accepted Python coding standards. This helps improve the readability, maintainability, and overall quality of the code.

2. **Error and Warning Detection:** Pylint identifies potential error points and coding mistakes, helping to prevent errors that may occur at runtime.

3. **Style Guide Compliance:** Pylint performs compliance checks with the style guides, including PEP 8 and other guidelines defined by the Python community. This ensures that Python code adheres to a consistent style.

What It Checks:

1. It verifies compliance with relevant style guides.
2. It evaluates the complexity and clarity of functions or classes, considering factors such as nested loops within a function.
3. It checks for error situations like undefined variables or functions.
4. It assesses the extent of code testing and verifies test coverage.

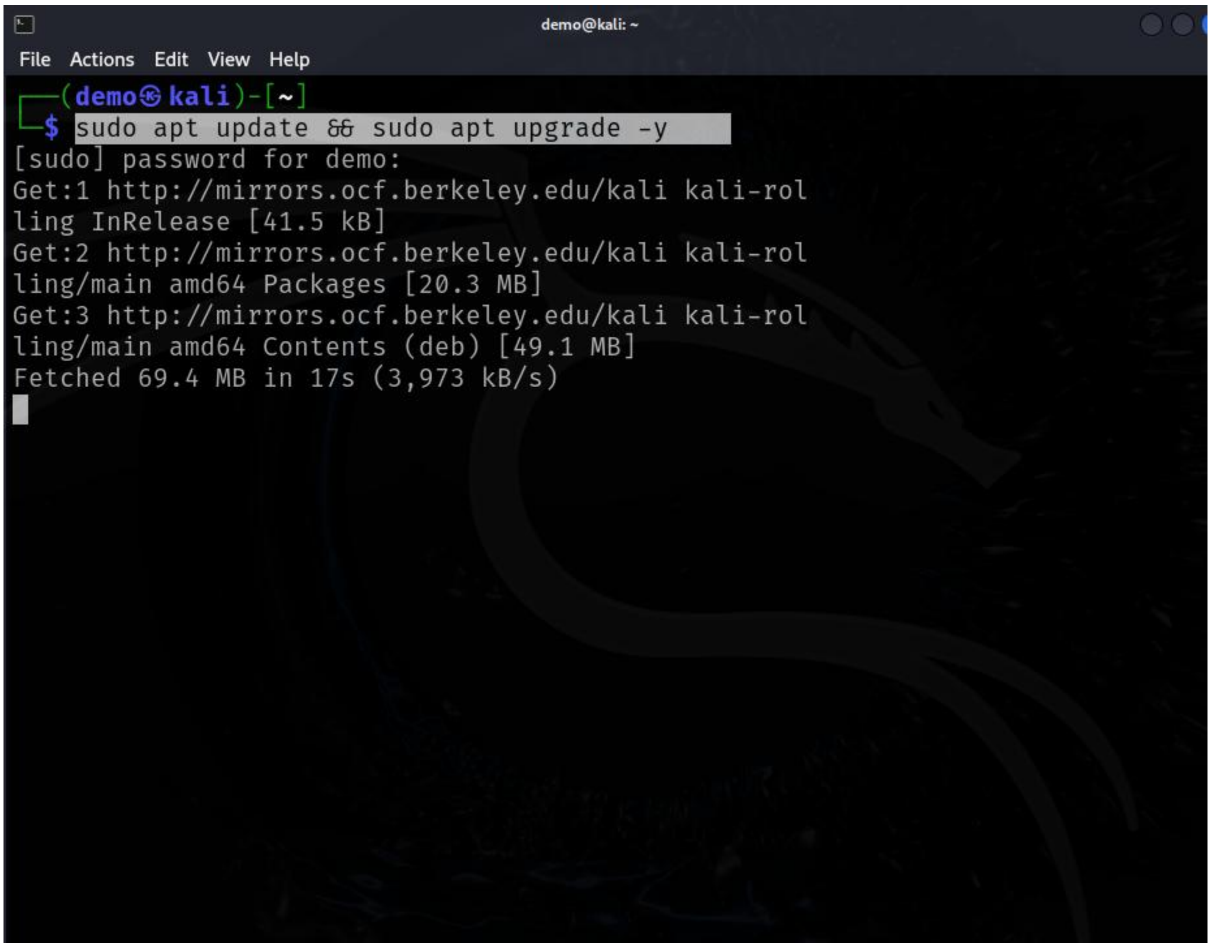
3. Identified Security Vulnerabilities

➢ Setting Up Your Environment

Step 1: First, let's ensure your environment is ready for development.

I. Update Your System: Open a terminal in Kali and run the following commands to update your system:

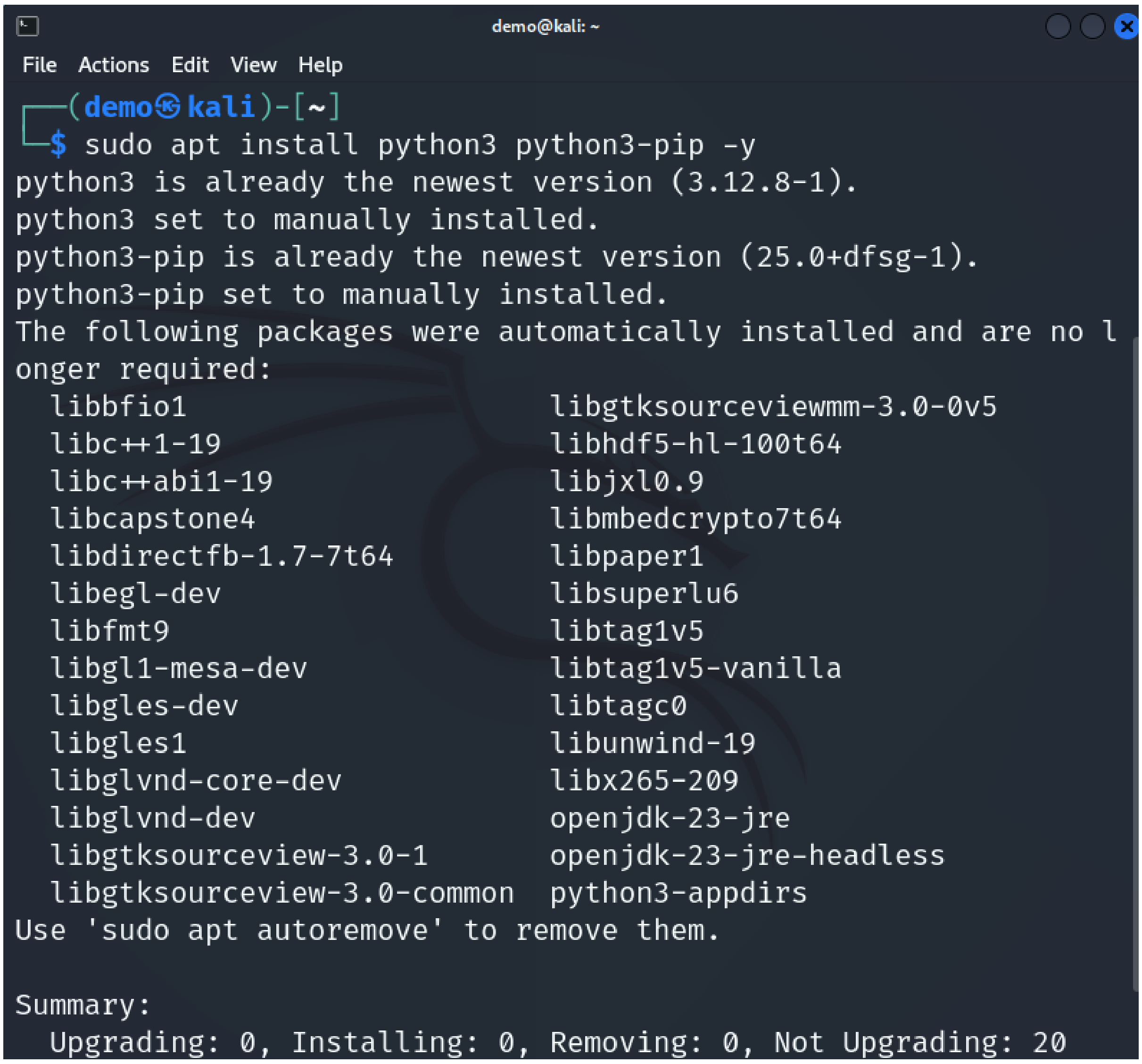
sudo apt update
sudo apt upgrade -y



```
┌──(demo㉿kali)-[~]
└─$ sudo apt update && sudo apt upgrade -y
[sudo] password for demo:
Get:1 http://mirrors.ocf.berkeley.edu/kali kali-rol
ling InRelease [41.5 kB]
Get:2 http://mirrors.ocf.berkeley.edu/kali kali-rol
ling/main amd64 Packages [20.3 MB]
Get:3 http://mirrors.ocf.berkeley.edu/kali kali-rol
ling/main amd64 Contents (deb) [49.1 MB]
Fetched 69.4 MB in 17s (3,973 kB/s)
```

II. Install Python and Pip: Make sure Python and pip (Python package manager) are installed:

sudo apt install python3 python3-pip -y



```
┌──(demo㉿kali)-[~]
└─$ sudo apt install python3 python3-pip -y
python3 is already the newest version (3.12.8-1).
python3 set to manually installed.
python3-pip is already the newest version (25.0+dfsg-1).
python3-pip set to manually installed.
The following packages were automatically installed and are no l
onger required:
  libbfio1                        libgtksourceviewmm-3.0-0v5
  libc++1-19                      libhdf5-hl-100t64
  libc++abi1-19                   libjxl0.9
  libcapstone4                    libmbedcrypto7t64
  libdirectfb-1.7-7t64            libpaper1
  libegl-dev                      libsuperlu6
  libfmt9                         libtag1v5
  libgl1-mesa-dev                 libtag1v5-vanilla
  libgles-dev                     libtagc0
  libgles1                        libunwind-19
  libglvnd-core-dev               libx265-209
  libglvnd-dev                    openjdk-23-jre
  libgtksourceview-3.0-1          openjdk-23-jre-headless
  libgtksourceview-3.0-common     python3-appdirs
Use 'sudo apt autoremove' to remove them.

Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 20
```

III. Install Virtualenv: It's a good practice to use a virtual environment to manage dependencies:

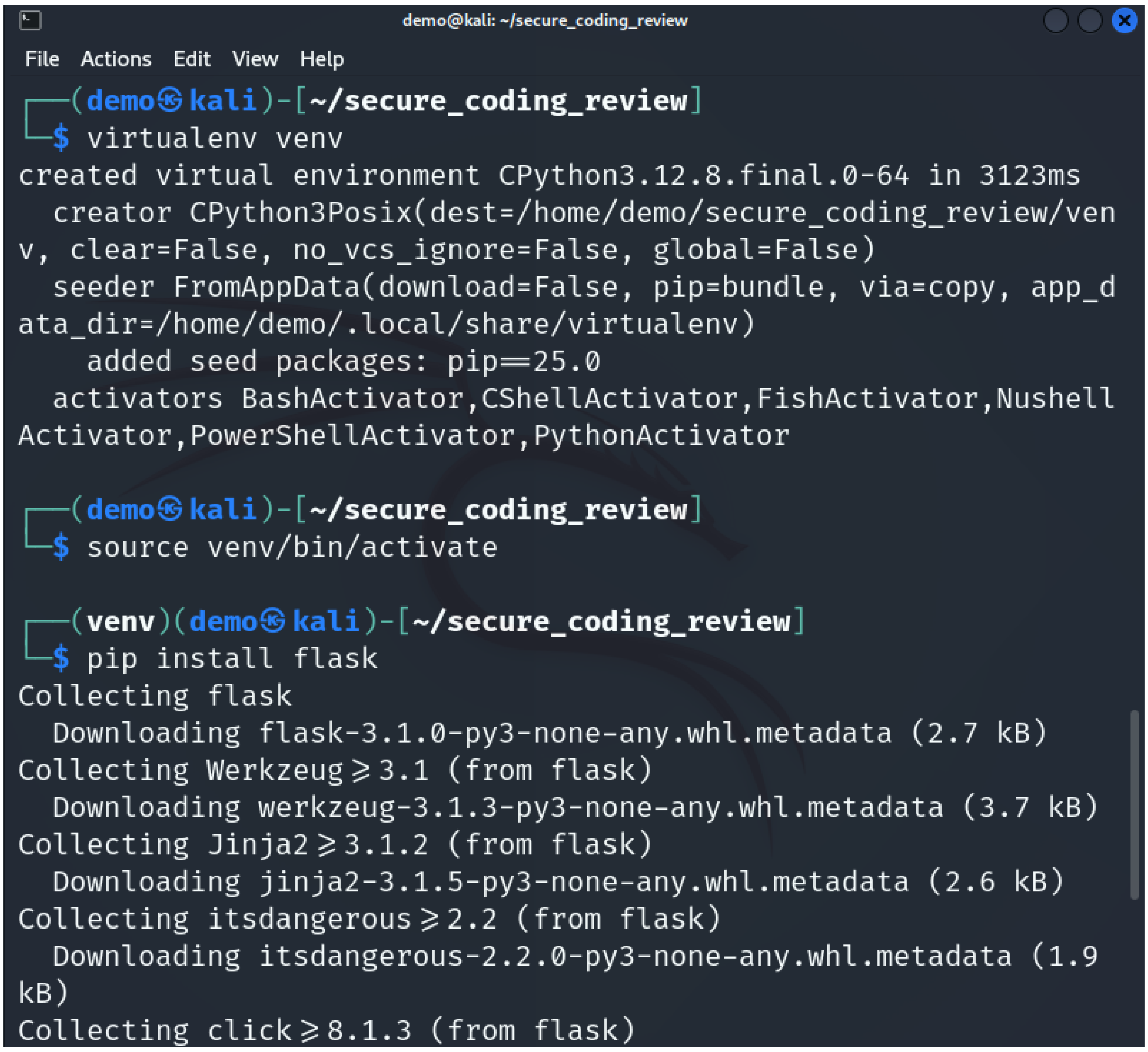
sudo pip3 install virtualenv

Step 2: Creating a Simple Flask Application.

I. Set Up a Virtual Environment: Create and activate a virtual environment:

mkdir secure_coding_review

cd secure_coding_review
virtualenv venv
source venv/bin/activate


demo@kali: ~/secure_coding_review
File Actions Edit View Help

```
┌──(demo㉿kali)-[~/secure_coding_review]
└─$ virtualenv venv
created virtual environment CPython3.12.8.final.0-64 in 3123ms
  creator CPython3Posix(dest=/home/demo/secure_coding_review/ven
v, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, via=copy, app_d
ata_dir=/home/demo/.local/share/virtualenv)
    added seed packages: pip==25.0
  activators BashActivator,CShellActivator,FishActivator,Nushell
Activator,PowerShellActivator,PythonActivator

┌──(demo㉿kali)-[~/secure_coding_review]
└─$ source venv/bin/activate

┌──(venv)(demo㉿kali)-[~/secure_coding_review]
└─$ pip install flask
Collecting flask
  Downloading flask-3.1.0-py3-none-any.whl.metadata (2.7 kB)
Collecting Werkzeug>=3.1 (from flask)
  Downloading werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Collecting Jinja2>=3.1.2 (from flask)
  Downloading jinja2-3.1.5-py3-none-any.whl.metadata (2.6 kB)
Collecting itsdangerous>=2.2 (from flask)
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9
kB)
Collecting click>=8.1.3 (from flask)
```

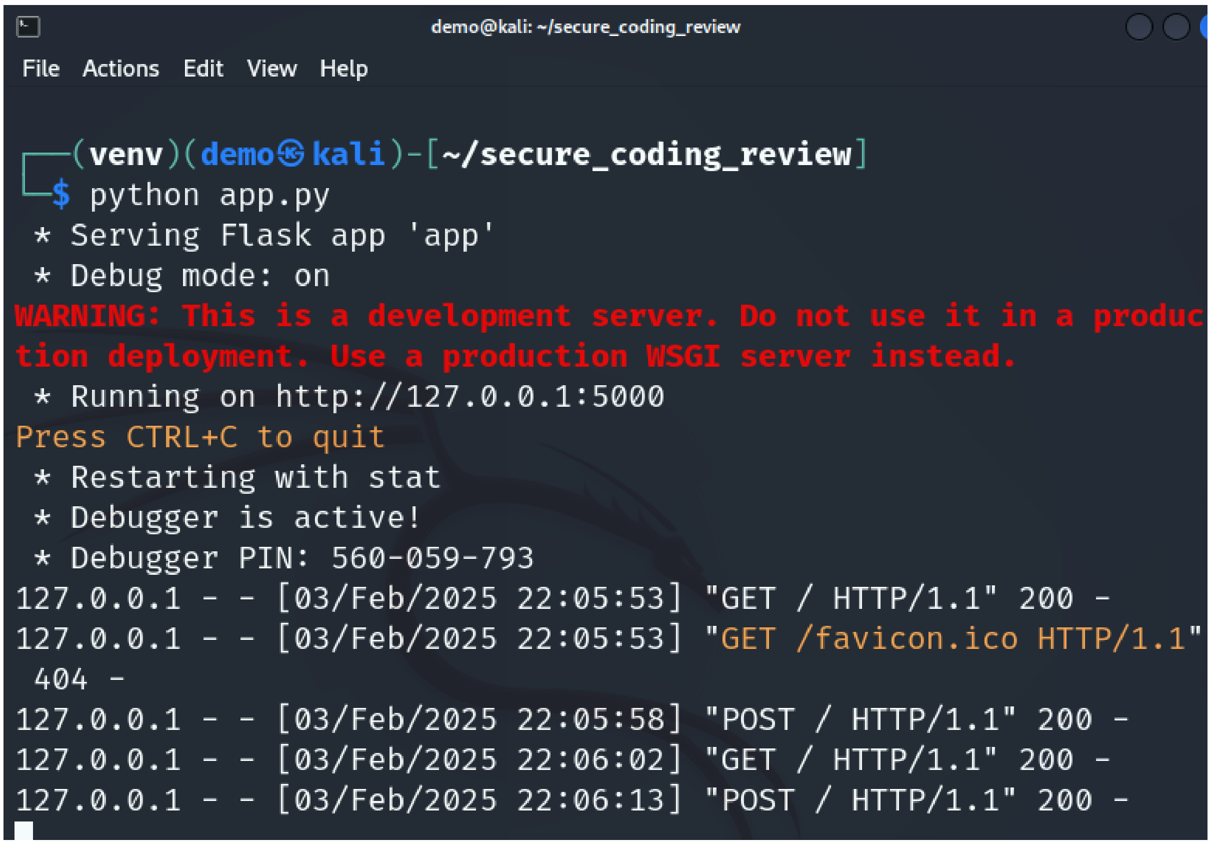
II. Install Flask: Install Flask in your virtual environment:

pip install Flask

III. Create a Simple Flask App: Create a file named app.py and write the code

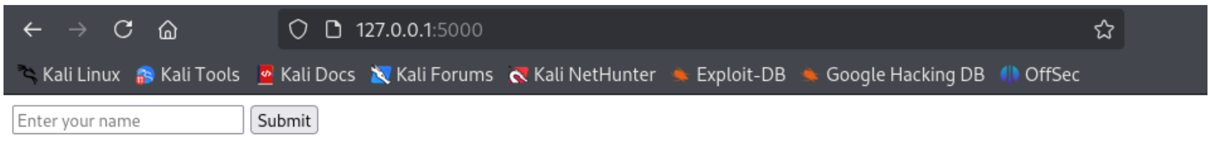
IV. Run the Flask App: Start your Flask application:

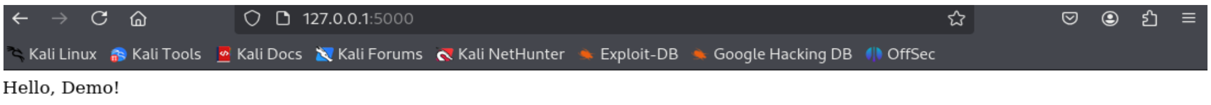
python app.pp


demo@kali: ~/secure_coding_review
File Actions Edit View Help

```
┌──(venv)(demo㉿kali)-[~/secure_coding_review]
└─$ python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 560-059-793
127.0.0.1 - - [03/Feb/2025 22:05:53] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [03/Feb/2025 22:05:53] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [03/Feb/2025 22:05:58] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [03/Feb/2025 22:06:02] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [03/Feb/2025 22:06:13] "POST / HTTP/1.1" 200 -
```

V. Open a browser and go to http://127.0.0.1:5000 to see your application in action.





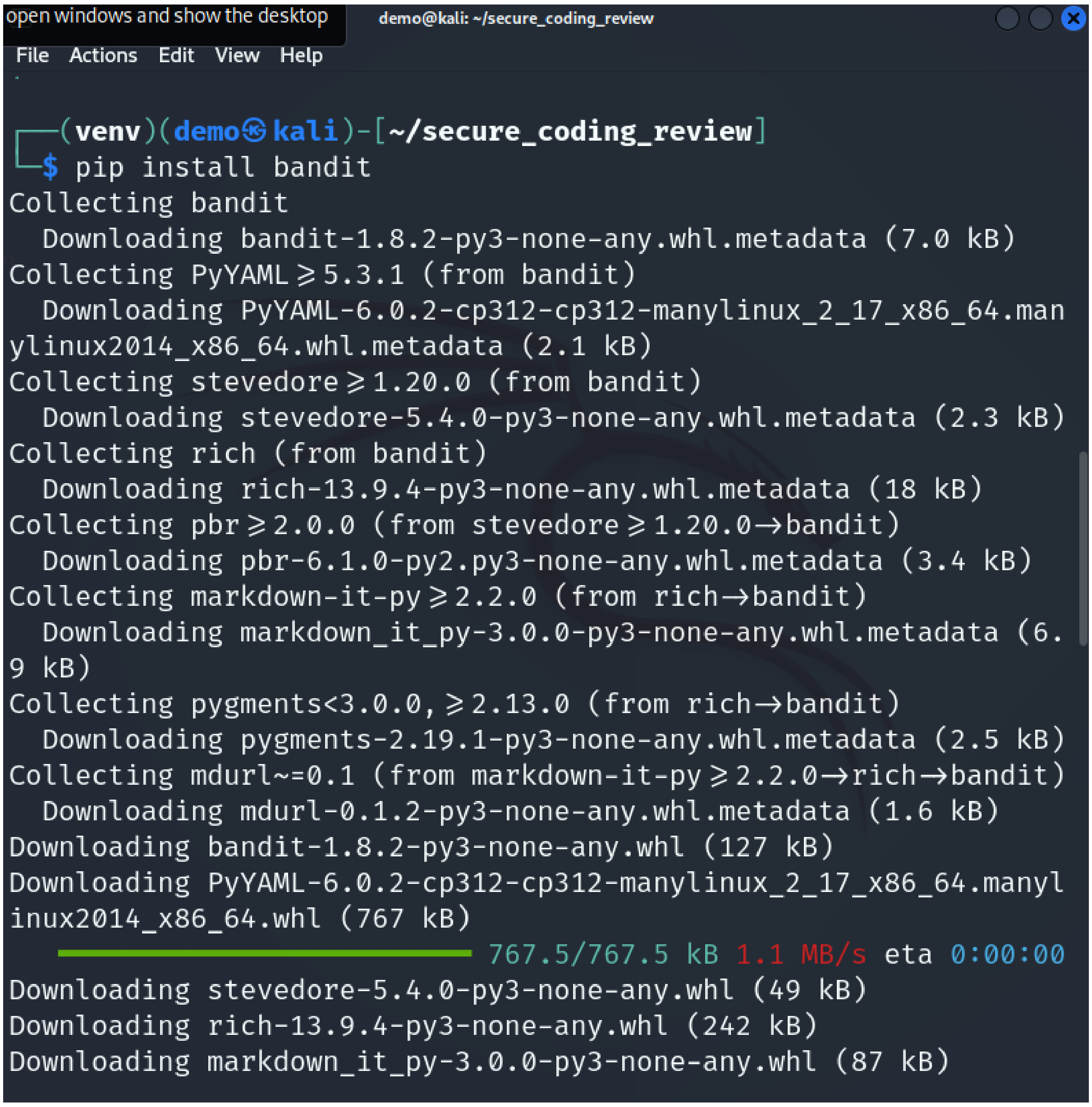
Hello, Demo!

➢ **Using Static Code Analyzers**

Use tools to analyze your code:

1. Install Bandit: Install Bandit to find common security issues in Python code:

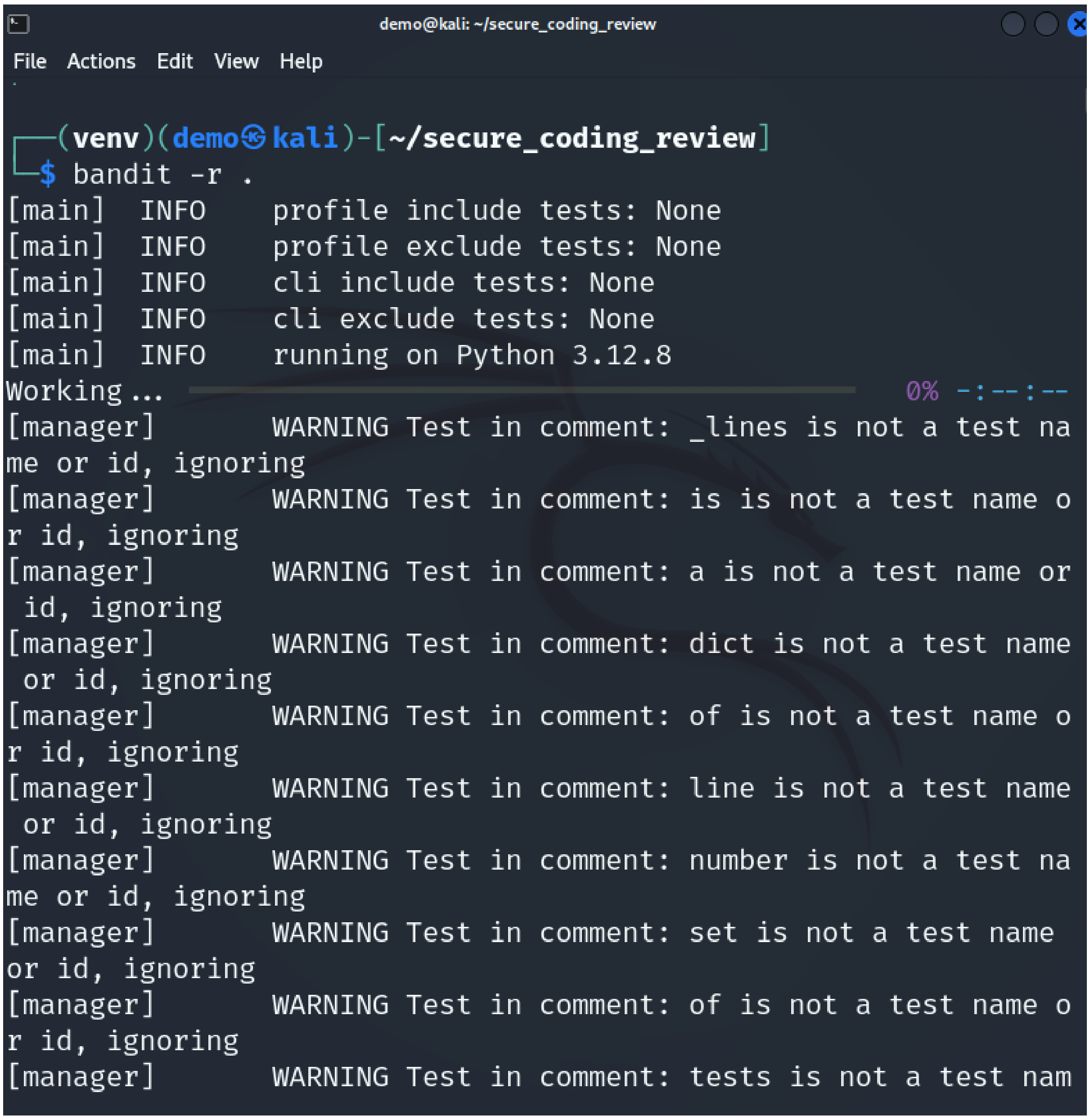
pip install bandit


open windows and show the desktop
demo@kali: ~/secure_coding_review
File Actions Edit View Help

```
┌──(venv)(demo㉿kali)-[~/secure_coding_review]
└─$ pip install bandit
Collecting bandit
  Downloading bandit-1.8.2-py3-none-any.whl.metadata (7.0 kB)
Collecting PyYAML>=5.3.1 (from bandit)
  Downloading PyYAML-6.0.2-cp312-cp312-manylinux_2_17_x86_64.man
ylinux2014_x86_64.whl.metadata (2.1 kB)
Collecting stevedore>=1.20.0 (from bandit)
  Downloading stevedore-5.4.0-py3-none-any.whl.metadata (2.3 kB)
Collecting rich (from bandit)
  Downloading rich-13.9.4-py3-none-any.whl.metadata (18 kB)
Collecting pbr>=2.0.0 (from stevedore>=1.20.0->bandit)
  Downloading pbr-6.1.0-py2.py3-none-any.whl.metadata (3.4 kB)
Collecting markdown-it-py>=2.2.0 (from rich->bandit)
  Downloading markdown_it_py-3.0.0-py3-none-any.whl.metadata (6.
9 kB)
Collecting pygments<3.0.0,>=2.13.0 (from rich->bandit)
  Downloading pygments-2.19.1-py3-none-any.whl.metadata (2.5 kB)
Collecting mdurl~=0.1 (from markdown-it-py>=2.2.0->rich->bandit)
  Downloading mdurl-0.1.2-py3-none-any.whl.metadata (1.6 kB)
Downloading bandit-1.8.2-py3-none-any.whl (127 kB)
Downloading PyYAML-6.0.2-cp312-cp312-manylinux_2_17_x86_64.manyl
inux2014_x86_64.whl (767 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 767.5/767.5 kB 1.1 MB/s eta 0:00:00
Downloading stevedore-5.4.0-py3-none-any.whl (49 kB)
Downloading rich-13.9.4-py3-none-any.whl (242 kB)
Downloading markdown_it_py-3.0.0-py3-none-any.whl (87 kB)
```

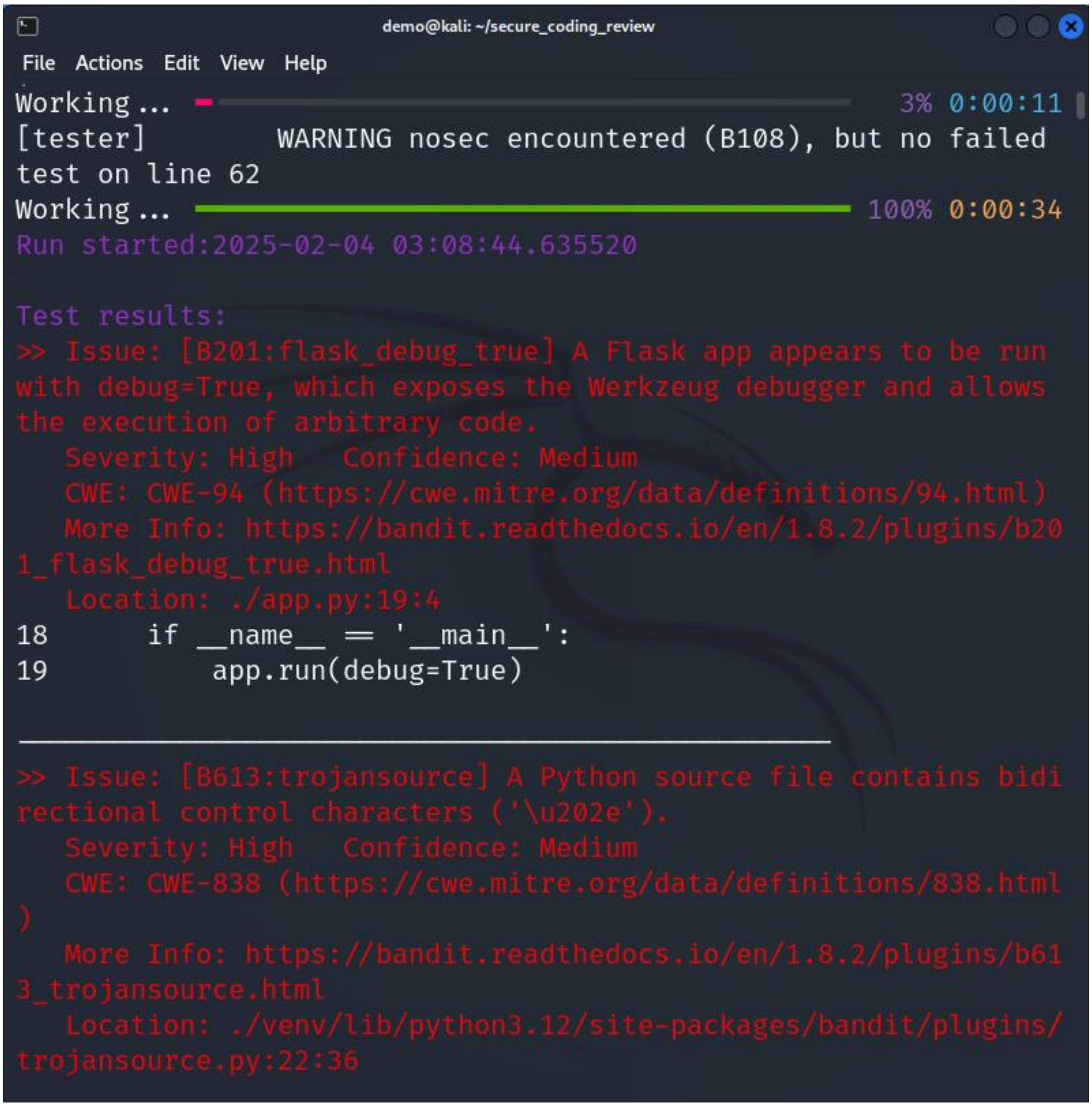
2. Run Bandit: Run Bandit on your Flask app:

bandit -r .


demo@kali: ~/secure_coding_review
File Actions Edit View Help

```
┌──(venv)(demo㉿kali)-[~/secure_coding_review]
└─$ bandit -r .
[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 3.12.8
Working...                                        0% -:--:--
[manager]       WARNING Test in comment: _lines is not a test na
me or id, ignoring
[manager]       WARNING Test in comment: is is not a test name o
r id, ignoring
[manager]       WARNING Test in comment: a is not a test name or
 id, ignoring
[manager]       WARNING Test in comment: dict is not a test name
 or id, ignoring
[manager]       WARNING Test in comment: of is not a test name o
r id, ignoring
[manager]       WARNING Test in comment: line is not a test name
 or id, ignoring
[manager]       WARNING Test in comment: number is not a test na
me or id, ignoring
[manager]       WARNING Test in comment: set is not a test name
or id, ignoring
[manager]       WARNING Test in comment: of is not a test name o
r id, ignoring
[manager]       WARNING Test in comment: tests is not a test nam
```

Findings:

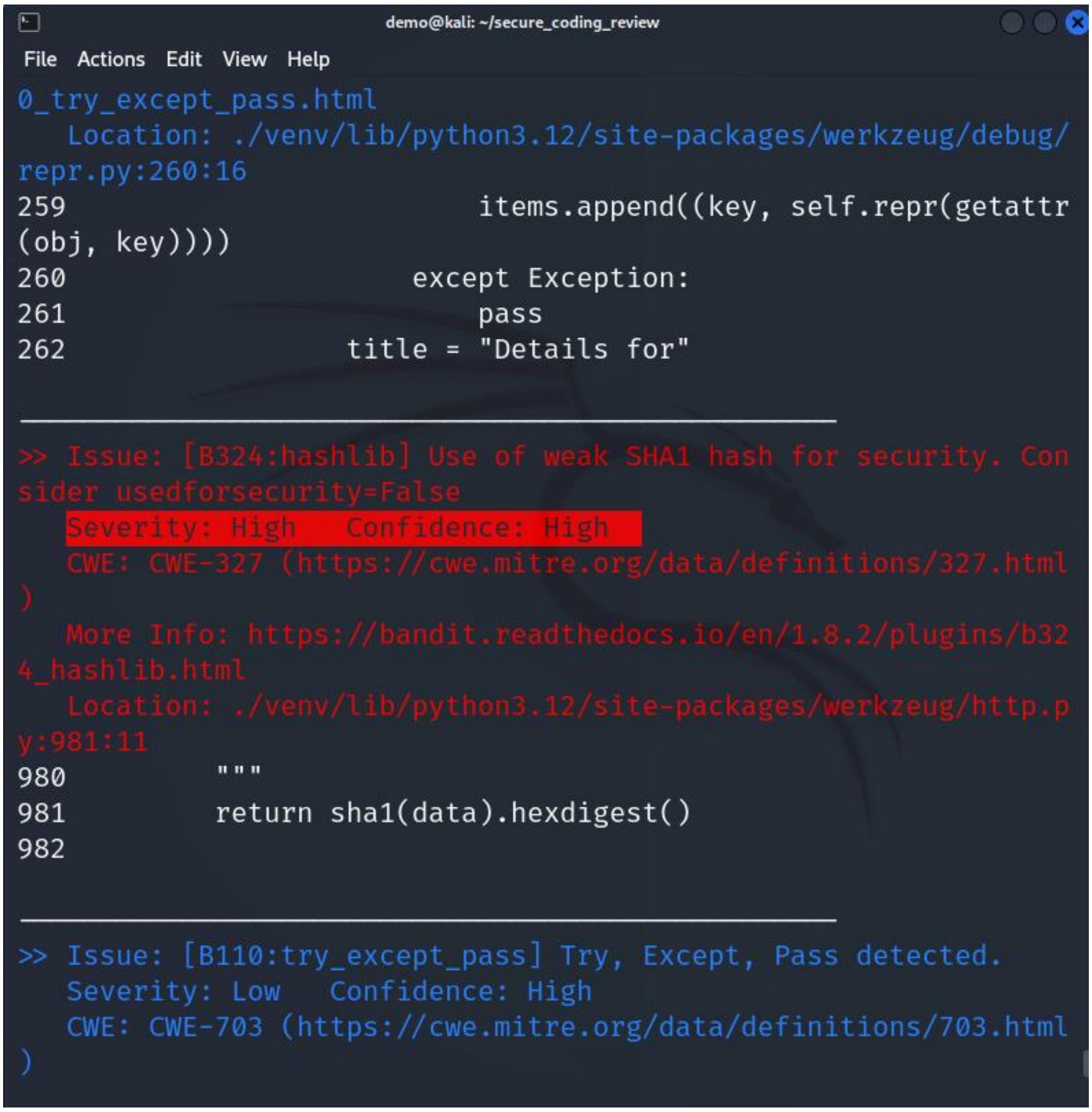

demo@kali: ~/secure_coding_review
File Actions Edit View Help

```
Working ... ━                                            3% 0:00:11
[tester]        WARNING nosec encountered (B108), but no failed
test on line 62
Working ... ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 0:00:34
Run started:2025-02-04 03:08:44.635520

Test results:
>> Issue: [B201:flask_debug_true] A Flask app appears to be run
with debug=True, which exposes the Werkzeug debugger and allows
the execution of arbitrary code.
   Severity: High   Confidence: Medium
   CWE: CWE-94 (https://cwe.mitre.org/data/definitions/94.html)
   More Info: https://bandit.readthedocs.io/en/1.8.2/plugins/b20
1_flask_debug_true.html
   Location: ./app.py:19:4
18      if __name__ == '__main__':
19          app.run(debug=True)

--------------------------------------------------
>> Issue: [B613:trojansource] A Python source file contains bidi
rectional control characters ('\u202e').
   Severity: High   Confidence: Medium
   CWE: CWE-838 (https://cwe.mitre.org/data/definitions/838.html
)
   More Info: https://bandit.readthedocs.io/en/1.8.2/plugins/b61
3_trojansource.html
   Location: ./venv/lib/python3.12/site-packages/bandit/plugins/
trojansource.py:22:36
```

Bandit Finding | Issue, Severity and CWE

Flask Debug Mode Enabled (B201: flask_debug_true)

- **Severity:** High
- **Confidence:** Medium
- **Description:** The Flask application is configured to run with debug=True. This exposes the Werkzeug debugger, enabling arbitrary code execution if vulnerability is exploited. This is **critically dangerous** in a production environment and must be addressed immediately.
- **Location:** /app.py:19:4 (Line 19, character 4 of app.py)
- **Recommendation:** Disable debug mode in production by setting debug=False in the app.run() call. Debug mode should *only* be used during development. Example of corrected code:

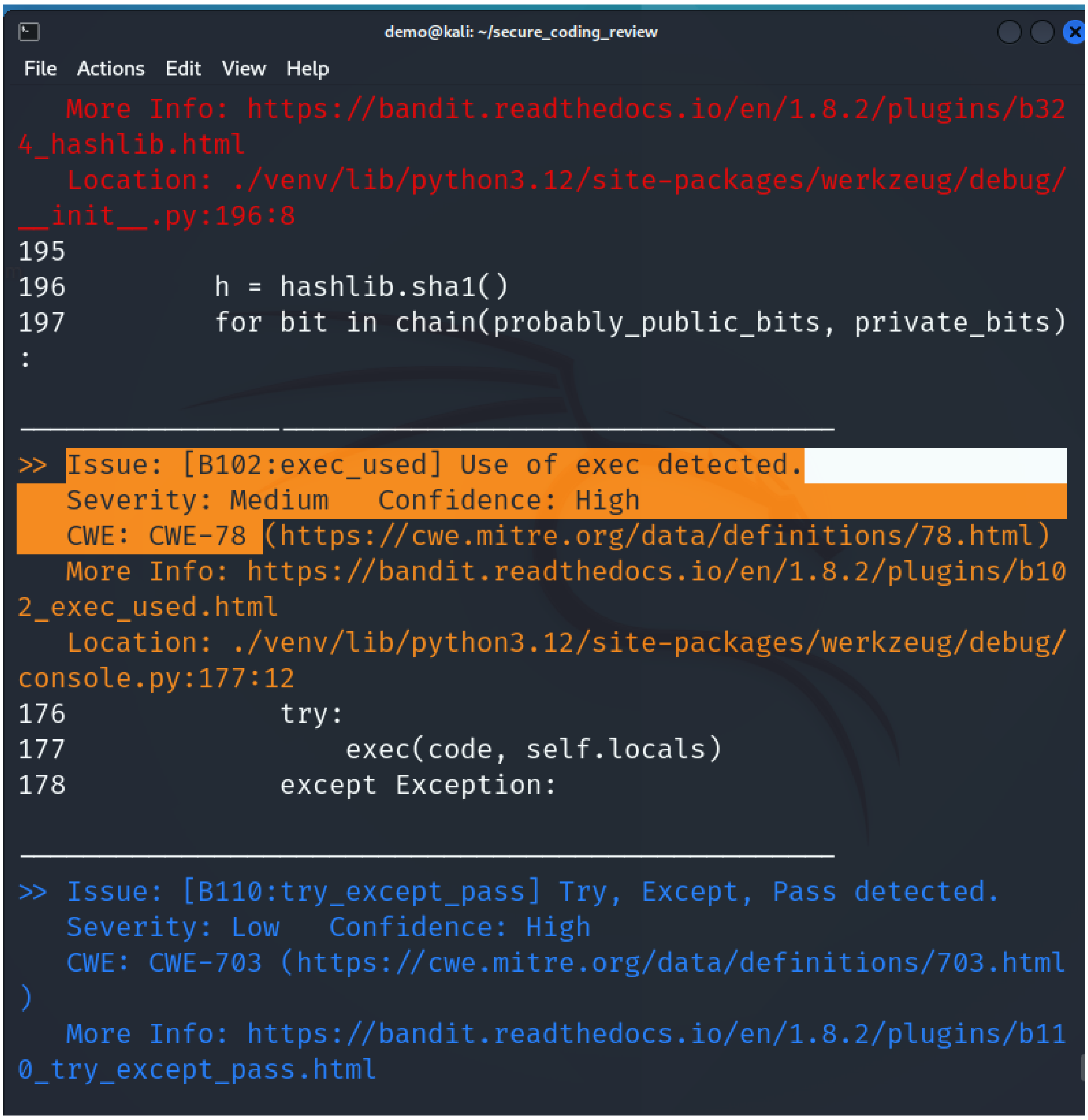

demo@kali: ~/secure_coding_review
File Actions Edit View Help

```
0_try_except_pass.html
   Location: ./venv/lib/python3.12/site-packages/werkzeug/debug/
repr.py:260:16
259                         items.append((key, self.repr(getattr
(obj, key))))
260                     except Exception:
261                         pass
262                 title = "Details for"

--------------------------------------------------
>> Issue: [B324:hashlib] Use of weak SHA1 hash for security. Con
sider usedforsecurity=False
   Severity: High   Confidence: High
   CWE: CWE-327 (https://cwe.mitre.org/data/definitions/327.html
)
   More Info: https://bandit.readthedocs.io/en/1.8.2/plugins/b32
4_hashlib.html
   Location: ./venv/lib/python3.12/site-packages/werkzeug/http.p
y:981:11
980         """
981         return sha1(data).hexdigest()
982

--------------------------------------------------
>> Issue: [B110:try_except_pass] Try, Except, Pass detected.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html
)
```

Bandit | Issue, Severity and CWE

Use of Weak SHA1 Hash (B324: hashlib)

- **Severity:** High
- **Confidence:** High
- **Description:** The code utilizes the SHA1 hashing algorithm for security purposes. SHA1 is cryptographically broken and should **not** be used for protecting sensitive data. Its weakness makes it vulnerable to collision attacks, where different inputs can produce the same hash value.
- **Location**: /venv/lib/python3.12/site-packages/werkzeug/http.py:981:11
- **Recommendation**: Replace SHA1 with a stronger hashing algorithm such as SHA256, SHA384, or SHA512. If the use case is genuinely not security-sensitive (e.g., generating non-unique identifiers), the warning can be suppressed using

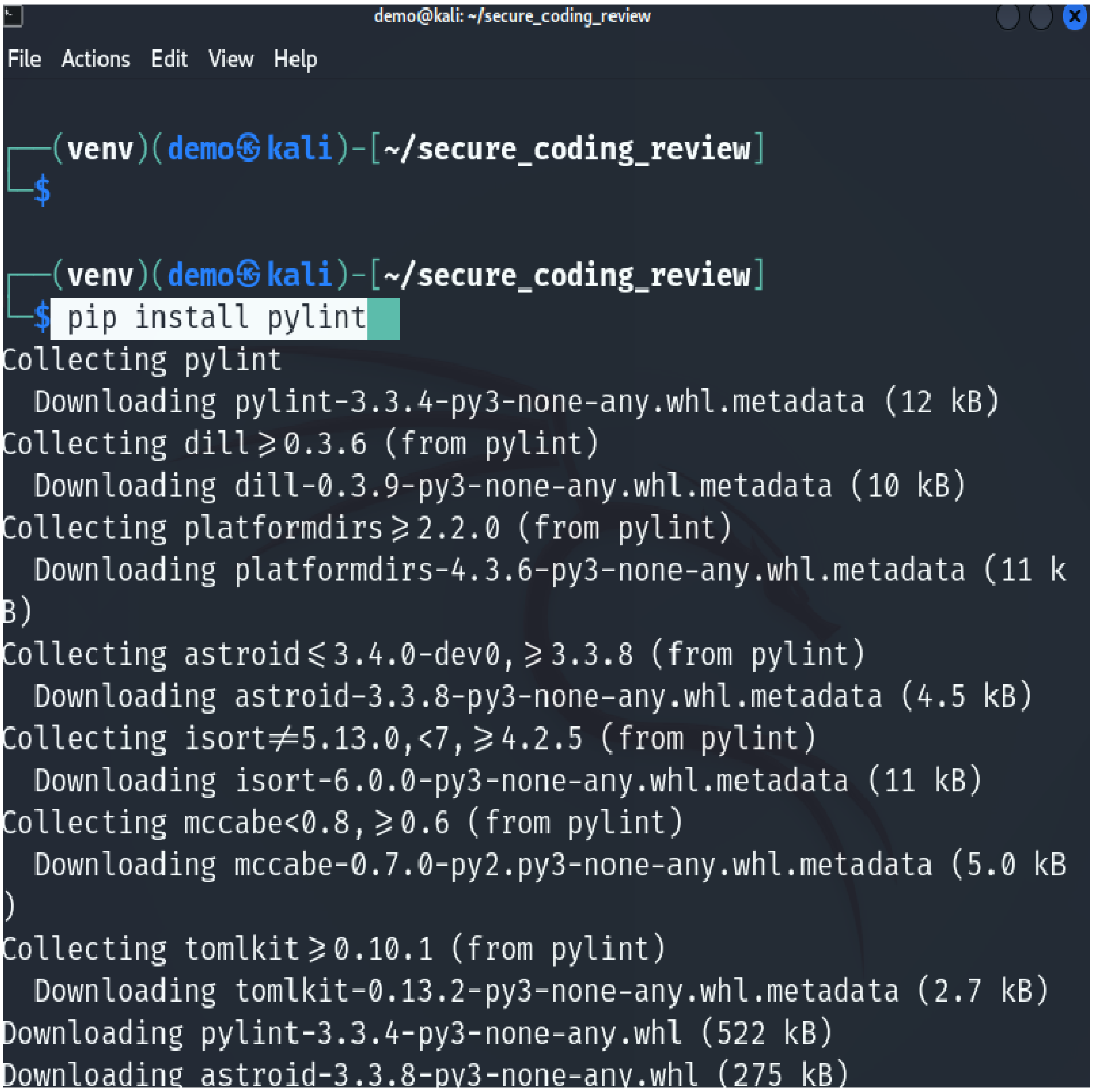

demo@kali: ~/secure_coding_review
File Actions Edit View Help

```
   More Info: https://bandit.readthedocs.io/en/1.8.2/plugins/b32
4_hashlib.html
   Location: ./venv/lib/python3.12/site-packages/werkzeug/debug/
__init__.py:196:8
195
196         h = hashlib.sha1()
197         for bit in chain(probably_public_bits, private_bits)
:

--------------------------------------------------
>> Issue: [B102:exec_used] Use of exec detected.
   Severity: Medium   Confidence: High
   CWE: CWE-78 (https://cwe.mitre.org/data/definitions/78.html)
   More Info: https://bandit.readthedocs.io/en/1.8.2/plugins/b10
2_exec_used.html
   Location: ./venv/lib/python3.12/site-packages/werkzeug/debug/
console.py:177:12
176             try:
177                 exec(code, self.locals)
178             except Exception:

--------------------------------------------------
>> Issue: [B110:try_except_pass] Try, Except, Pass detected.
   Severity: Low   Confidence: High
   CWE: CWE-703 (https://cwe.mitre.org/data/definitions/703.html
)
   More Info: https://bandit.readthedocs.io/en/1.8.2/plugins/b11
0_try_except_pass.html
```

Bandit Report

➢ **Install PyLint:** Install PyLint to check for coding errors:

pip install pylint



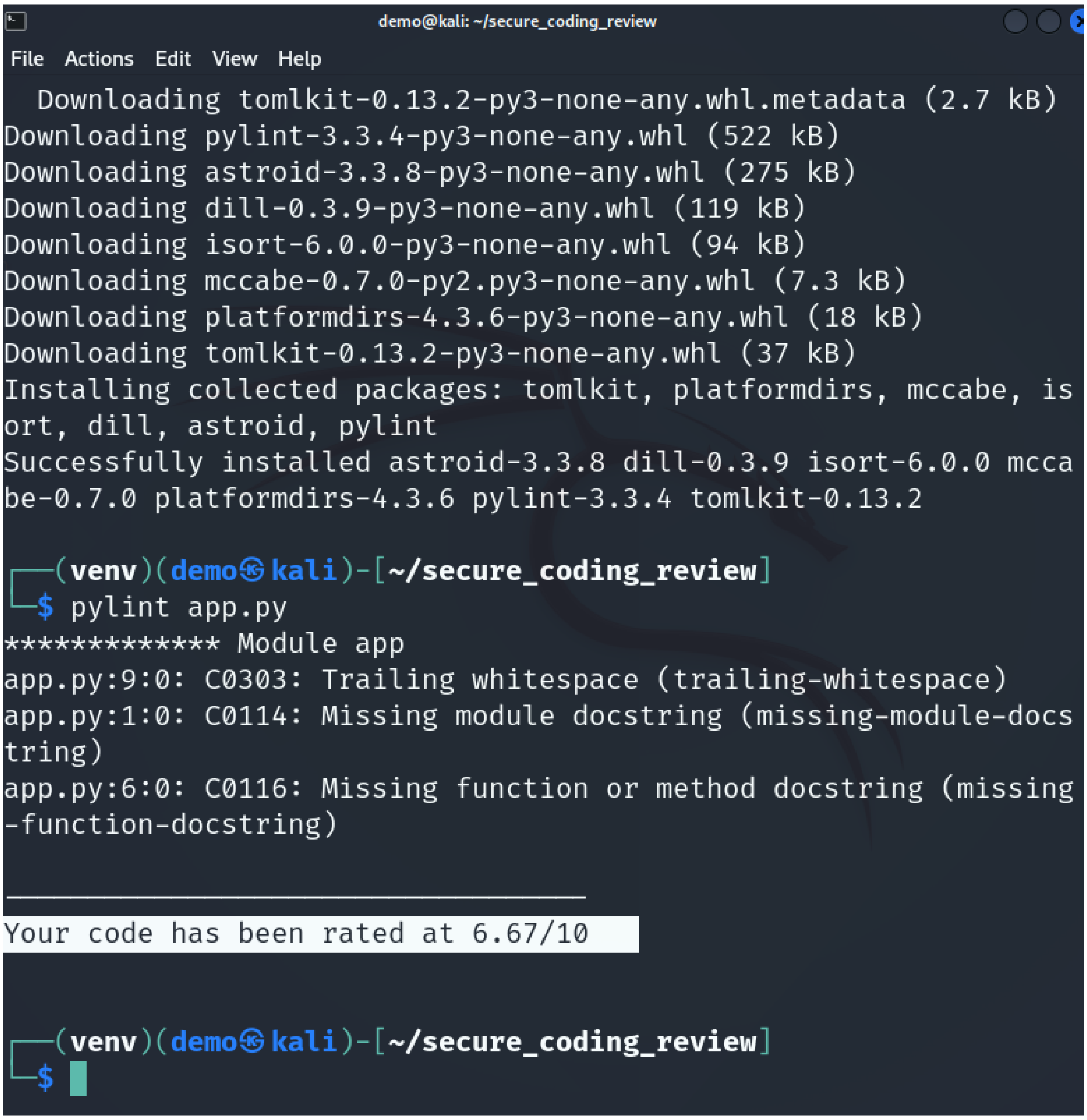
demo@kali: ~/secure_coding_review

```
┌──(venv)(demo㉿kali)-[~/secure_coding_review]
└─$ 

┌──(venv)(demo㉿kali)-[~/secure_coding_review]
└─$ pip install pylint
Collecting pylint
  Downloading pylint-3.3.4-py3-none-any.whl.metadata (12 kB)
Collecting dill≥0.3.6 (from pylint)
  Downloading dill-0.3.9-py3-none-any.whl.metadata (10 kB)
Collecting platformdirs≥2.2.0 (from pylint)
  Downloading platformdirs-4.3.6-py3-none-any.whl.metadata (11 k
B)
Collecting astroid≤3.4.0-dev0,≥3.3.8 (from pylint)
  Downloading astroid-3.3.8-py3-none-any.whl.metadata (4.5 kB)
Collecting isort≠5.13.0,<7,≥4.2.5 (from pylint)
  Downloading isort-6.0.0-py3-none-any.whl.metadata (11 kB)
Collecting mccabe<0.8,≥0.6 (from pylint)
  Downloading mccabe-0.7.0-py2.py3-none-any.whl.metadata (5.0 kB
)
Collecting tomlkit≥0.10.1 (from pylint)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Downloading pylint-3.3.4-py3-none-any.whl (522 kB)
Downloading astroid-3.3.8-py3-none-any.whl (275 kB)
```

Installing Pylint

- **Run PyLint:** Run PyLint on your Flask app:

pylint app.py


demo@kali: ~/secure_coding_review
File Actions Edit View Help

```
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Downloading pylint-3.3.4-py3-none-any.whl (522 kB)
Downloading astroid-3.3.8-py3-none-any.whl (275 kB)
Downloading dill-0.3.9-py3-none-any.whl (119 kB)
Downloading isort-6.0.0-py3-none-any.whl (94 kB)
Downloading mccabe-0.7.0-py2.py3-none-any.whl (7.3 kB)
Downloading platformdirs-4.3.6-py3-none-any.whl (18 kB)
Downloading tomlkit-0.13.2-py3-none-any.whl (37 kB)
Installing collected packages: tomlkit, platformdirs, mccabe, is
ort, dill, astroid, pylint
Successfully installed astroid-3.3.8 dill-0.3.9 isort-6.0.0 mcca
be-0.7.0 platformdirs-4.3.6 pylint-3.3.4 tomlkit-0.13.2

┌──(venv)(demo㉿kali)-[~/secure_coding_review]
└─$ pylint app.py
************* Module app
app.py:9:0: C0303: Trailing whitespace (trailing-whitespace)
app.py:1:0: C0114: Missing module docstring (missing-module-docs
tring)
app.py:6:0: C0116: Missing function or method docstring (missing
-function-docstring)

-----------------------------------
Your code has been rated at 6.67/10


┌──(venv)(demo㉿kali)-[~/secure_coding_review]
└─$ 
```

Pylint Score

Step 3: **Reviewing the Code for Security Vulnerabilities**

1. SQL Injection: Not applicable here as we're not using databases yet.
2. Cross-Site Scripting (XSS): Check the /greet route for XSS vulnerabilities.
3. Cross-Site Request Forgery (CSRF): Check for CSRF protection.
4. Authentication and Authorization: Not implemented in this basic app.

4. Recommendations for Secure Coding Practices

To mitigate the identified vulnerabilities, the following secure coding practices are recommended:

- **Input Validation:** Always validate and sanitize user inputs. Use libraries such as WTForms or Flaskto handle form validation effectively.

- **Output Encoding:** Implement output encoding for all user-generated content to prevent XSS attacks. Use Flask's built-in escape functions to ensure that dynamic content is safely rendered.

- **Secure Libraries:** Utilize well-maintained libraries and frameworks that are known for their security features. Regularly update dependencies to patch known vulnerabilities.

- **Error Handling:** Avoid exposing sensitive information in error messages. Use generic error messages for end-users while logging detailed error information securely for developers.

- **Environment Configuration:** Store sensitive information such as API keys and database credentials in environment variables instead of hardcoding them in the source code.

Conduct regular code reviews and security audits to identify and remediate vulnerabilities proactively.

5. Conclusion

The Secure Code Review project has successfully identified several security vulnerabilities within the Python Flask application. By employing a combination of automated tools and manual review techniques, we were able to assess the security posture of the application comprehensively. The findings highlight critical areas for improvement, emphasizing the need for secure coding practices in web development.

Moving forward, it is essential to foster a culture of security awareness among developers and to incorporate secure coding practices into the development lifecycle. Continuous improvement and adaptation to emerging security threats will be vital in maintaining the integrity and security of web applications.

Future research should focus on developing more robust tools and methodologies for secure code reviews, ensuring that developers are equipped to tackle evolving security challenges effectively.

In conclusion, secure coding is not just the best practice but a necessity in today's threat landscape. By implementing the recommendations outlined in this report, developers can significantly reduce the risk of vulnerabilities and enhance the overall security of their applications.

References:

1. **Pylint**: https://pypi.org/project/pylint/

2. **Pylint Official Website**: https://www.pylint.org/

3. **Bandit**: https://bandit.readthedocs.io/en/latest/index.html, https://github.com/PyCQA/bandit

4. **Flask**: https://flask.palletsprojects.com/en/stable/

5. **Aqua Security**: https://www.aquasec.com/cloud-native-academy/devsecops/secure-code-review/

6. **Black Duck:** https://www.blackduck.com/glossary/what-is-code-review.html

7. **SpectralOps:** https://spectralops.io/blog/performing-a-secure-code-review/

THANK YOU