Cyber Security Internship Task

Name : Raushan Raj

Email : roushan211003@gmail.com

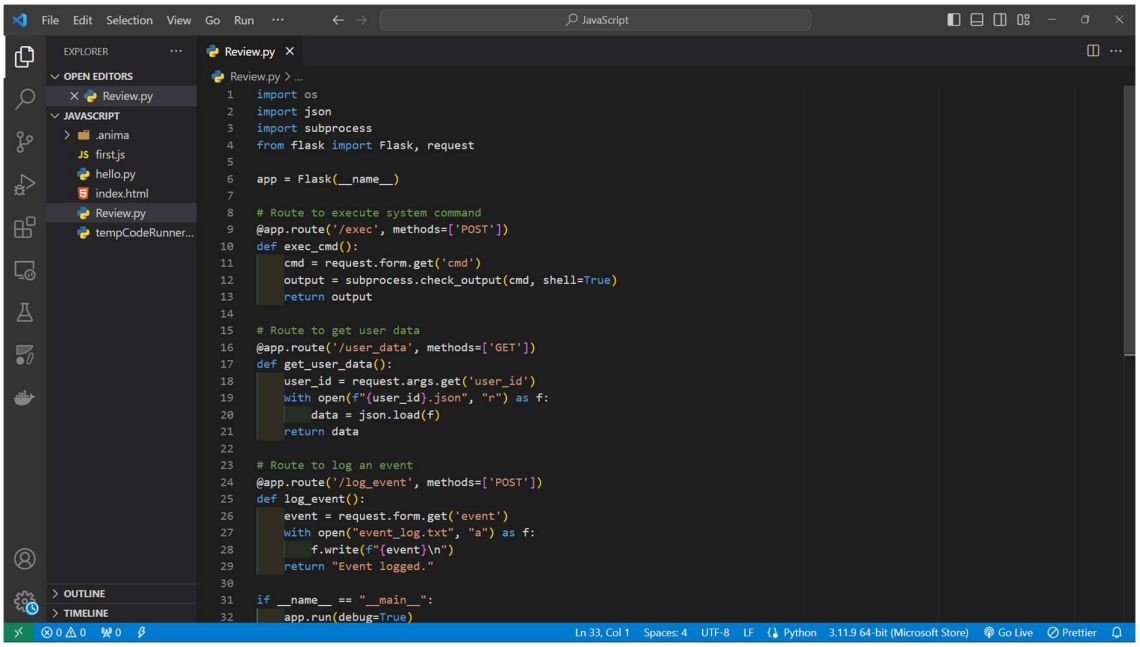
Student ID(Codealpha) : CA/S14120

Task Name :- Secure Coding Review

I'll use a python code to review it. I will be reviewing the code for vulnerabilities and provide recommendations for secure coding practices.

This review analyzes the Python code for a Flask application, focusing on potential security vulnerabilities.

Python Code :-



```python
import os
import json
import subprocess
from flask import Flask, request

app = Flask(__name__)

# Route to execute system command
@app.route('/exec', methods=['POST'])
def exec_cmd():
    cmd = request.form.get('cmd')
    output = subprocess.check_output(cmd, shell=True)
    return output

# Route to get user data
@app.route('/user_data', methods=['GET'])
def get_user_data():
    user_id = request.args.get('user_id')
    with open(f"{user_id}.json", "r") as f:
        data = json.load(f)
    return data

# Route to log an event
@app.route('/log_event', methods=['POST'])
def log_event():
    event = request.form.get('event')
    with open("event_log.txt", "a") as f:
        f.write(f"{event}\n")
    return "Event logged."

if __name__ == "__main__":
    app.run(debug=True)
```

Vulnerabilities and Recommendations

1. Command Injection (High Severity)

- **Vulnerable Code:**

```python
@app.route('/exec', methods=['POST'])
def exec_cmd():
  cmd = request.form.get('cmd')
  output = subprocess.check_output(cmd, shell=True)
  return output
```

- **Recommendation:** This code is highly vulnerable to command injection attacks. An attacker could craft a malicious request containing commands that could be executed on the server.
- **Fix:** Use subprocess.run with shell=False and separate arguments instead of building a single string command.

```
@app.route('/exec', methods=['POST'])
def exec_cmd():
  cmd = request.form.get('cmd').split()  # Split the command string into a list
  output = subprocess.run(cmd, capture_output=True, text=True, check=True).stdout
  return output
```

2. Insecure Direct Object References (Medium Severity)

- **Vulnerable Code:**

```
@app.route('/user_data', methods=['GET'])
def get_user_data():
  user_id = request.args.get('user_id')
  with open(f"{user_id}.json", "r") as f:
    data = json.load(f)
  return data
```

- **Recommendation:** This code directly constructs a filename based on user input. An attacker could potentially access unauthorized data by providing a crafted user ID.
- **Fix:** Validate and sanitize the user ID to ensure it represents a valid user before using it in filename construction. Consider a whitelist approach for allowed user IDs.

3. Sensitive Data Exposure (Medium Severity)

- **Vulnerable Code:** No encryption for user data or logging potentially sensitive events
- **Recommendation:** The code doesn't explicitly show handling of sensitive data. If user data or logged events contain sensitive information, implement appropriate security measures.
- **Solutions:**
  - Encrypt user data at rest and in transit (HTTPS).
  - Sanitize or anonymize logged events before storing them.

4. Flask Debug Mode (Medium Severity)

- **Vulnerability:** Running the application in debug mode (debug=True) exposes sensitive information in the development environment.
- **Recommendation:** Only run the application in debug mode during development. In production, use a production-ready configuration with debug mode disabled.

Additional Recommendations:

- **Dependency Management:** Use tools like ***pipenv*** to manage dependencies and ensure you're using the latest versions with security patches.
- **Regular Updates:** Keep your Python interpreter and libraries up-to-date to address security vulnerabilities.
- **Code Reviews:** Conduct regular code reviews to identify potential security issues.
- **Security Testing:** Perform security testing, such as vulnerability scanning and penetration testing, to identify and address weaknesses.