Secure Coding Review Task

I am choosing Python programming language for a web based application. I will review the code for security vulnerabilities and provide recommendations for secure coding practices.

I am using tools like static code analyzers (Bandit) for this task review.

app.py for Users_Todo_WebApp

reference: https://github.com/Birdo1221/Users-Todo-WebApp/tree/main

A To-Do List Application using the Flask framework. The application allows users to register, log in, create tasks, edit tasks, mark tasks as completed and delete tasks. The application was built for general usage and to help organize daily tasks effectively.

Code to Review

```
from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
import json
import os
import base64
import uuid
import time
from flask import make_response
from flask_bcrypt import Bcrypt, generate_password_hash




app = Flask(__name__)
app.secret_key = '(X)gi==A=~j0zX_`=@/XL"FPps\apO'  # Update with your secret key
bcrypt = Bcrypt(app)

TASKS_FOLDER = 'user_tasks'
```

```
if not os.path.exists(TASKS_FOLDER):
    os.makedirs(TASKS_FOLDER)

# Define security questions
SECURITY_QUESTIONS = [
    "What was the name of your first pet?",
    "In what city were you born?",
    "What is your favorite movie?",
    "What is your mother's maiden name?",
    "What is the name of your favorite teacher?",
    "What was the make and model of your first car?",
    "What is the name of your favorite childhood friend?",
    "What is the birthplace of your father?",
    "What is the title of your favorite book?",
    "In what year did you graduate from high school?"
]


# Load user data from users.json
def load_users():
    try:
        with open('users.json', 'r') as f:
            return json.load(f)
    except (FileNotFoundError, json.decoder.JSONDecodeError):
        return {}


# Save user data to users.json
def save_users(users):
    try:
        with open('users.json', 'w') as f:
            json.dump(users, f, indent=2)
    except Exception as e:
        print(f"Error saving users: {str(e)}")




# Function to verify user's identity by answering security question
def verify_user(username, password, security_answer):
    users = load_users()
    if username in users:
        user_data = users[username]
        if bcrypt.check_password_hash(user_data['password'], password) and
user_data['security_answer'] == security_answer:
            return True
    return False


@app.route('/')
def index():
    return render_template('index.html')


@app.route('/dashboard')
def dashboard():
    if 'username' in session:
        username = session['username']
        task_file_name = generate_task_file_name(username)
        task_file_path = os.path.join(TASKS_FOLDER, task_file_name)
        tasks = []

        if os.path.exists(task_file_path):
            with open(task_file_path, 'r') as f:
                tasks = json.load(f)

        return render_template('dashboard.html', tasks=tasks,
task=None,username=username)
        # Pass task=None if it's not available, and the username to
        # Render to the template file
    else:
        flash('You must log in to access the dashboard.')
        return redirect(url_for('login'))



@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        passwordconf = request.form['passwordconf']
        security_question = request.form['security_question']
        security_answer = request.form['security_answer']

        users = load_users()

        # Ensure users is loaded
        if not users:
            users = {}
        # Check if the username already exists
        if password != passwordconf:
            flash('Password and confirmation dont match.')
            return redirect(url_for('register'))

        if passwordconf != passwordconf:
            flash('Password and confirmation dont match.')
            return redirect(url_for('register'))

        if username in users:
            flash('Username already exists.')
            return redirect(url_for('register'))
        else:
            # Hash the password
            hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')

            # Save only necessary information
            users[username] = {
                'password': hashed_password,
                'security_question': security_question,
                'security_answer': security_answer
            }
            save_users(users)
            flash('Registration successful! Please login.')
            return redirect(url_for('login'))

    return render_template('register.html',
security_questions=SECURITY_QUESTIONS)
```

```
    return render_template('register.html',
security_questions=SECURITY_QUESTIONS)



@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        security_answer = request.form['security_answer']

        if verify_user(username, password, security_answer):
            session['username'] = username
            flash('Login successful!')
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid username, password, or security answer. Please try again.')

    return render_template('login.html')



@app.route('/logout')
def logout():
    session.pop('username', None)
    session.clear
    flash('You have been logged out.')
    return redirect(url_for('index'))



# Function to assign unique IDs to tasks
```

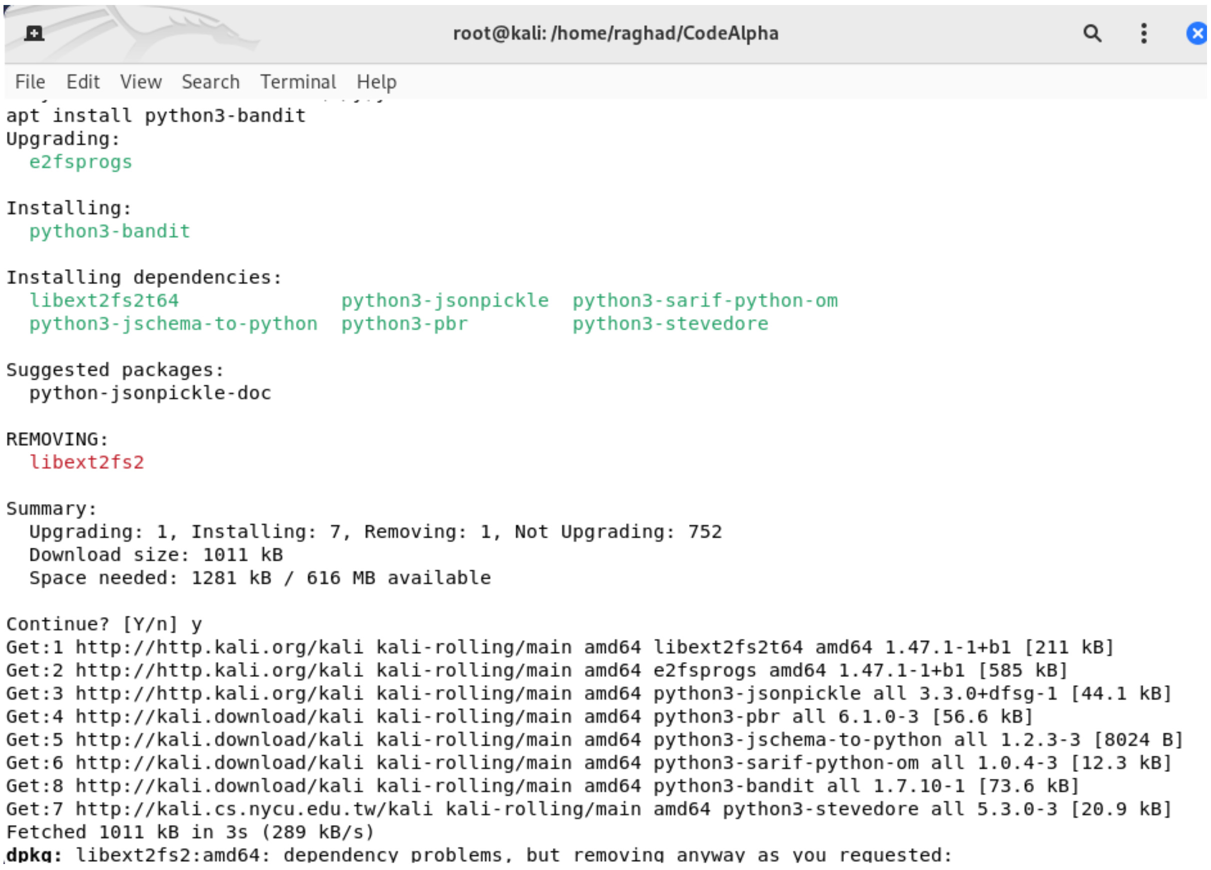
```
def assign_task_ids(tasks):
    for task in tasks:
        task['id'] = str(uuid.uuid4())

@app.route('/add_task', methods=['POST'])
def add_task():
    if 'username' in session:
        username = session['username']
        task_name = request.form['task_name']
        task_description = request.form['task_description']

        task_file_name = generate_task_file_name(username)
        task_file_path = os.path.join(TASKS_FOLDER, task_file_name)

        tasks = []

        if os.path.exists(task_file_path):
            with open(task_file_path, 'r') as f:
                tasks = json.load(f)

        new_task = {'id': str(uuid.uuid4()), 'name': task_name, 'description': 
task_description, 'completed': False}
        tasks.append(new_task)

        with open(task_file_path, 'w') as f:
            json.dump(tasks, f, indent=2)

        # Redirect to the dashboard after adding the task
        return redirect(url_for('dashboard'))
    else:
        # Return error if user is not logged in
        return jsonify({'error': 'User not logged in'}), 401


@app.route('/edit_task/<task_id>', methods=['POST'])
def edit_task(task_id):
    if 'username' in session:
        username = session['username']
        task_file_name = generate_task_file_name(username)
        task_file_path = os.path.join(TASKS_FOLDER, task_file_name)

        if os.path.exists(task_file_path):
            with open(task_file_path, 'r') as f:
                tasks = json.load(f)

            for task in tasks:
                if task['id'] == task_id:
                    # Update task details
                    task['description'] = request.form.get('description')
                    task['additional_description'] = request.form.get('additional_description')

            # Write the updated tasks back to the file
            with open(task_file_path, 'w') as f:
                json.dump(tasks, f, indent=2)

            return redirect(url_for('task_detail', task_id=task_id))
        else:
            return jsonify({'error': 'Task file not found'}), 404
    else:
        return jsonify({'error': 'User not logged in'}), 401
```

```
# app.py
@app.route('/dashboard/task/<task_id>')
def task_detail(task_id):
    if 'username' in session:
        # Retrieve the task details based on the task_id
        task = get_task_by_id(task_id)
        if task:
            return render_template('task_detail.html', task=task)
        else:
            flash('Task not found.')
            return redirect(url_for('dashboard'))
    else:
        return redirect(url_for('login'))



# This line should be indented to be part of the else block
@app.route('/delete_task', methods=['POST'])
def delete_task():
    if 'username' in session:
        username = session['username']
        task_id = request.json.get('taskId')

        task_file_name = generate_task_file_name(username)
        task_file_path = os.path.join(TASKS_FOLDER, task_file_name)

        if os.path.exists(task_file_path):
            with open(task_file_path, 'r') as f:
                tasks = json.load(f)
            # Filter out the task with the given taskId
            filtered_tasks = [task for task in tasks if task['id'] != task_id]

            # Save the updated tasks to the file
            with open(task_file_path, 'w') as f:
                json.dump(filtered_tasks, f, indent=2)

            return jsonify({'message': 'Task deleted successfully'}), 200
        else:
            return jsonify({'error': 'Task file not found'}), 404
    else:
        return jsonify({'error': 'User not logged in'}), 401

# Function to get task details by ID
def get_task_by_id(task_id):
    if 'username' in session:
        username = session['username']
        task_file_name = generate_task_file_name(username)
        task_file_path = os.path.join(TASKS_FOLDER, task_file_name)

        if os.path.exists(task_file_path):
            with open(task_file_path, 'r') as f:
                tasks = json.load(f)

            # Find the task with the given ID
            for task in tasks:
                if task['id'] == task_id:
                    return task
    return None




@app.route('/toggle_task', methods=['POST'])
def toggle_task():
    if 'username' in session:
        username = session['username']
        task_id = request.json.get('taskId')

        task_file_name = generate_task_file_name(username)
        task_file_path = os.path.join(TASKS_FOLDER, task_file_name)

        if os.path.exists(task_file_path):
            with open(task_file_path, 'r') as f:
                tasks = json.load(f)

            for task in tasks:
                if task['id'] == task_id:
                    task['completed'] = not task['completed']

            with open(task_file_path, 'w') as f:
                json.dump(tasks, f, indent=2)

            return jsonify({'message': 'Task toggled successfully'}), 200
        else:
            return jsonify({'error': 'Task file not found'}), 404
    else:
        return jsonify({'error': 'User not logged in'}), 401
```

```
# Helper function to generate unique task file name
def generate_task_file_name(username):
    encoded_username = base64.b64encode(username.encode()).decode()
    return f'tasks_{encoded_username}.json'


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

Installation of Bandit

- apt install python3-bandit



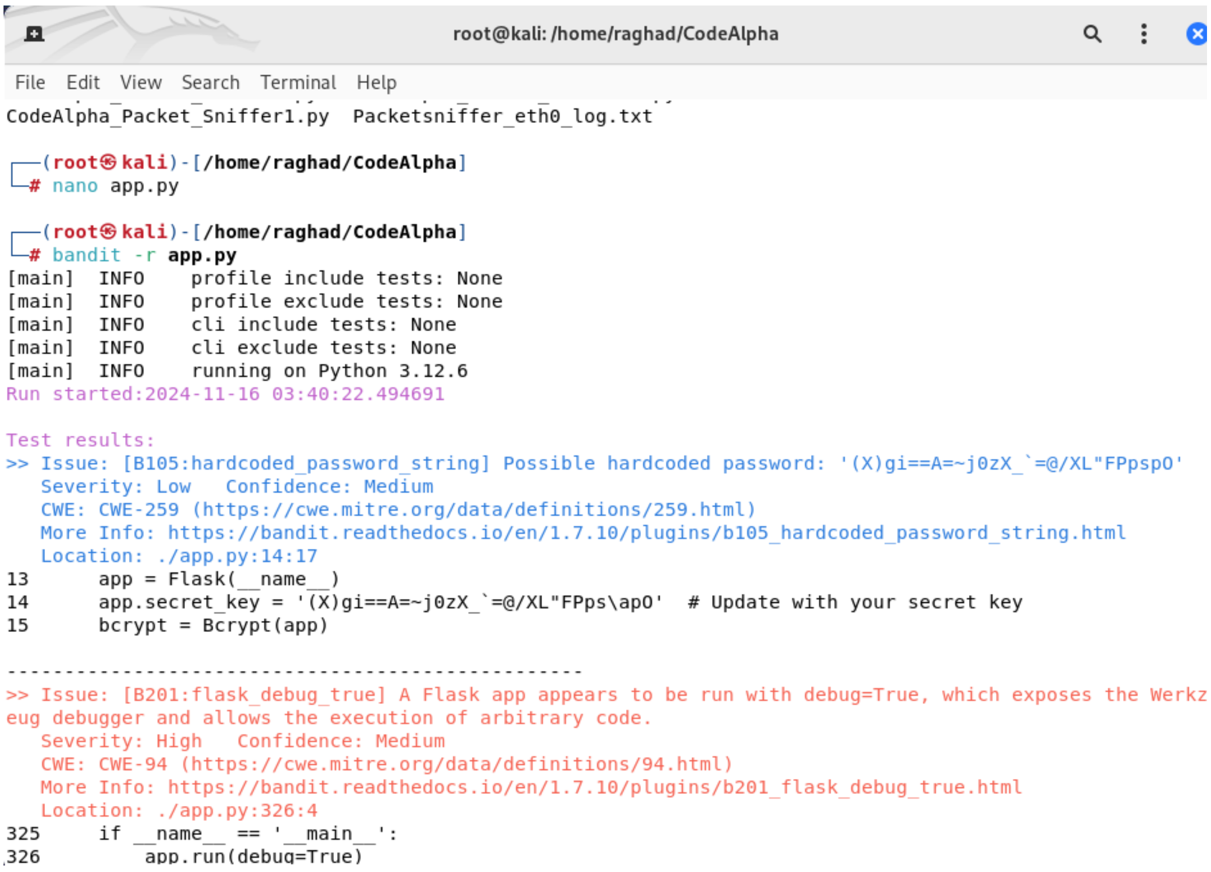
root@kali: /home/raghad/CodeAlpha
File Edit View Search Terminal Help

```
apt install python3-bandit
Upgrading:
  e2fsprogs

Installing:
  python3-bandit

Installing dependencies:
  libext2fs2t64            python3-jsonpickle  python3-sarif-python-om
  python3-jschema-to-python python3-pbr        python3-stevedore

Suggested packages:
  python-jsonpickle-doc

REMOVING:
  libext2fs2

Summary:
  Upgrading: 1, Installing: 7, Removing: 1, Not Upgrading: 752
  Download size: 1011 kB
  Space needed: 1281 kB / 616 MB available

Continue? [Y/n] y
Get:1 http://http.kali.org/kali kali-rolling/main amd64 libext2fs2t64 amd64 1.47.1-1+b1 [211 kB]
Get:2 http://http.kali.org/kali kali-rolling/main amd64 e2fsprogs amd64 1.47.1-1+b1 [585 kB]
Get:3 http://http.kali.org/kali kali-rolling/main amd64 python3-jsonpickle all 3.3.0+dfsg-1 [44.1 kB]
Get:4 http://kali.download/kali kali-rolling/main amd64 python3-pbr all 6.1.0-3 [56.6 kB]
Get:5 http://kali.download/kali kali-rolling/main amd64 python3-jschema-to-python all 1.2.3-3 [8024 B]
Get:6 http://kali.download/kali kali-rolling/main amd64 python3-sarif-python-om all 1.0.4-3 [12.3 kB]
Get:8 http://kali.download/kali kali-rolling/main amd64 python3-bandit all 1.7.10-1 [73.6 kB]
Get:7 http://kali.cs.nycu.edu.tw/kali kali-rolling/main amd64 python3-stevedore all 5.3.0-3 [20.9 kB]
Fetched 1011 kB in 3s (289 kB/s)
dpkg: libext2fs2:amd64: dependency problems, but removing anyway as you requested:
```

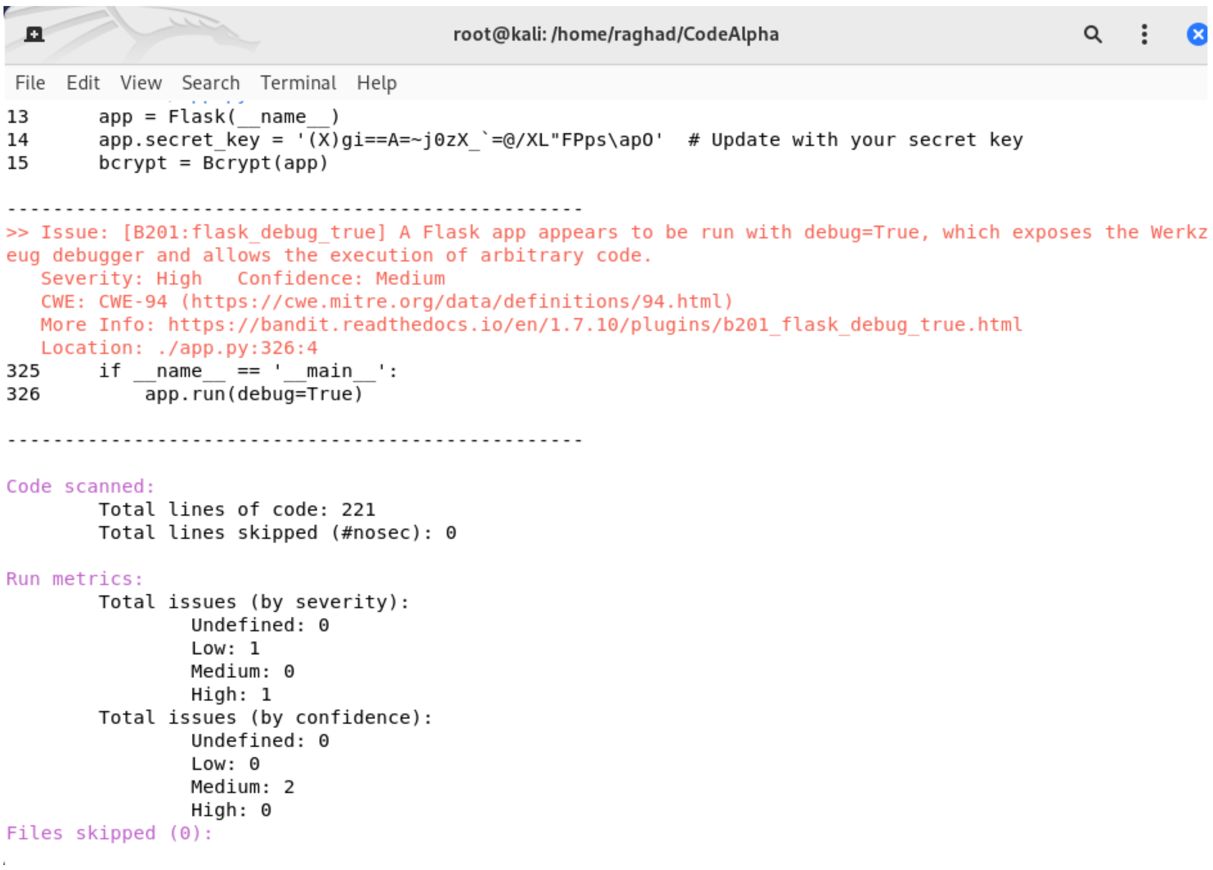
after saving the code in app.py

Run Analyzer (Bandit)

- bandit -r app.py


root@kali: /home/raghad/CodeAlpha
File Edit View Search Terminal Help

```
CodeAlpha_Packet_Sniffer1.py  Packetsniffer_eth0_log.txt

┌──(root㉿kali)-[/home/raghad/CodeAlpha]
└─# nano app.py

┌──(root㉿kali)-[/home/raghad/CodeAlpha]
└─# bandit -r app.py
[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 3.12.6
Run started:2024-11-16 03:40:22.494691

Test results:
>> Issue: [B105:hardcoded_password_string] Possible hardcoded password: '(X)gi==A=~j0zX_`=@/XL"FPpsp0'
   Severity: Low   Confidence: Medium
   CWE: CWE-259 (https://cwe.mitre.org/data/definitions/259.html)
   More Info: https://bandit.readthedocs.io/en/1.7.10/plugins/b105_hardcoded_password_string.html
   Location: ./app.py:14:17
13      app = Flask(__name__)
14      app.secret_key = '(X)gi==A=~j0zX_`=@/XL"FPps\ap0'  # Update with your secret key
15      bcrypt = Bcrypt(app)

--------------------------------------------------
>> Issue: [B201:flask_debug_true] A Flask app appears to be run with debug=True, which exposes the Werkz
eug 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/1.7.10/plugins/b201_flask_debug_true.html
   Location: ./app.py:326:4
325     if __name__ == '__main__':
326         app.run(debug=True)
```


root@kali: /home/raghad/CodeAlpha
File Edit View Search Terminal Help

```
13      app = Flask(__name__)
14      app.secret_key = '(X)gi==A=~j0zX_`=@/XL"FPps\apO'  # Update with your secret key
15      bcrypt = Bcrypt(app)

--------------------------------------------------
>> Issue: [B201:flask_debug_true] A Flask app appears to be run with debug=True, which exposes the Werkz
eug 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/1.7.10/plugins/b201_flask_debug_true.html
   Location: ./app.py:326:4
325     if __name__ == '__main__':
326         app.run(debug=True)

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

Code scanned:
        Total lines of code: 221
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 1
                Medium: 0
                High: 1
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 2
                High: 0
Files skipped (0):
```

The Bandit scan detected two main vulnerabilities;

1. the first one shows a “**possible**” hardcoded secret_key value that might lead to a security risk leading to public exposure or data breach
   - **Severity**: Low
   - **Confidence**: Medium
   - Line 14 (app.secret_key = '(X)gi==A=~j0zX_=@/XL"FPps\apO'`) //supposed to be replaced, so it might be just an example.
     However,
   - **Recommendations:** a secure code practice to prevent sensitive data exposure and to rotate keys without changing the original code.
     import os
     app.secret_key = os.getenv('FLASK_SECRET_KEY', 'default-fallback-key')

2. The second vulnerability is running a Flask app with debug=True enabling Werkzeug debugger., which possibly permits arbitrary code execution. This poses a serious concern, particularly in settings that involve production.
   - **Severity**: High
   - **Confidence**: Medium
   - Line 326 (`app.run(debug=True)`)
   - **Recommendations**: Make sure the debug flag is turned off for production and only turned on for development.
     if __name__ == '__main__':
        app.run(debug=os.getenv('FLASK_DEBUG', 'False') == 'True')