**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**](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()**

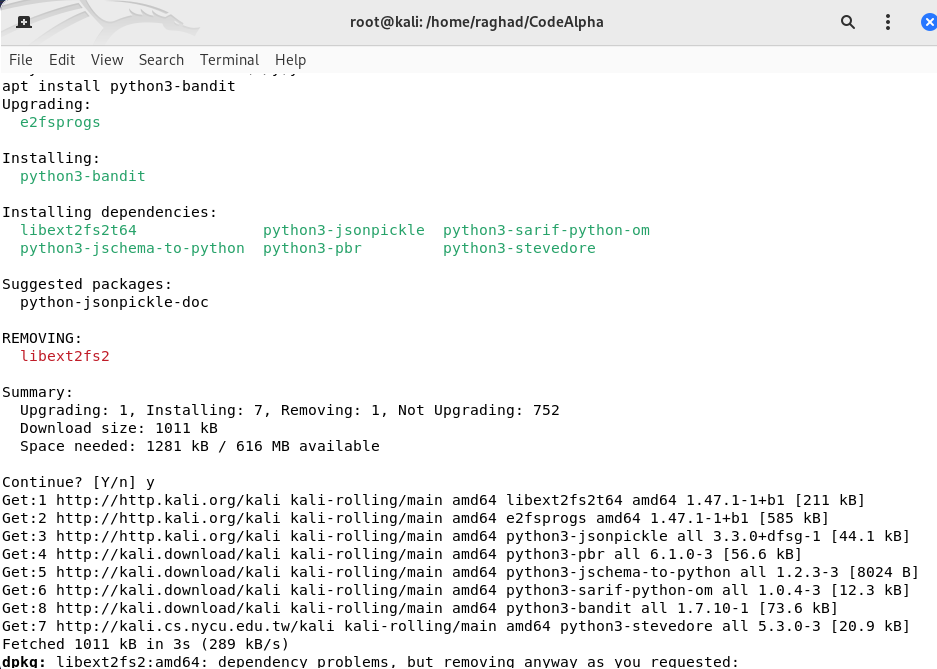
**return f'tasks\_{encoded\_username}.json'**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(debug=True)**

**Installation of Bandit**

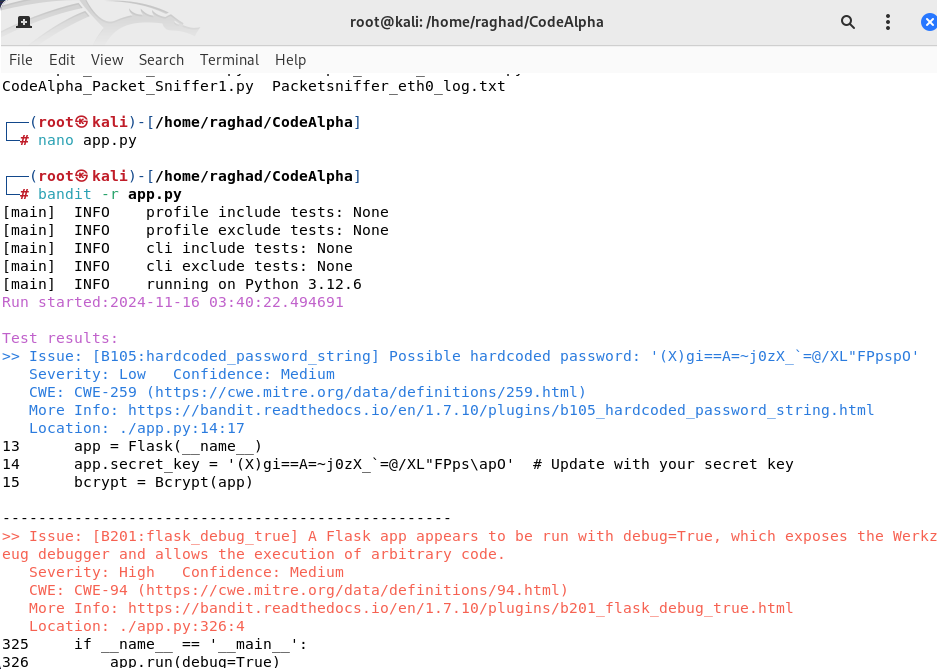
* apt install python3-bandit

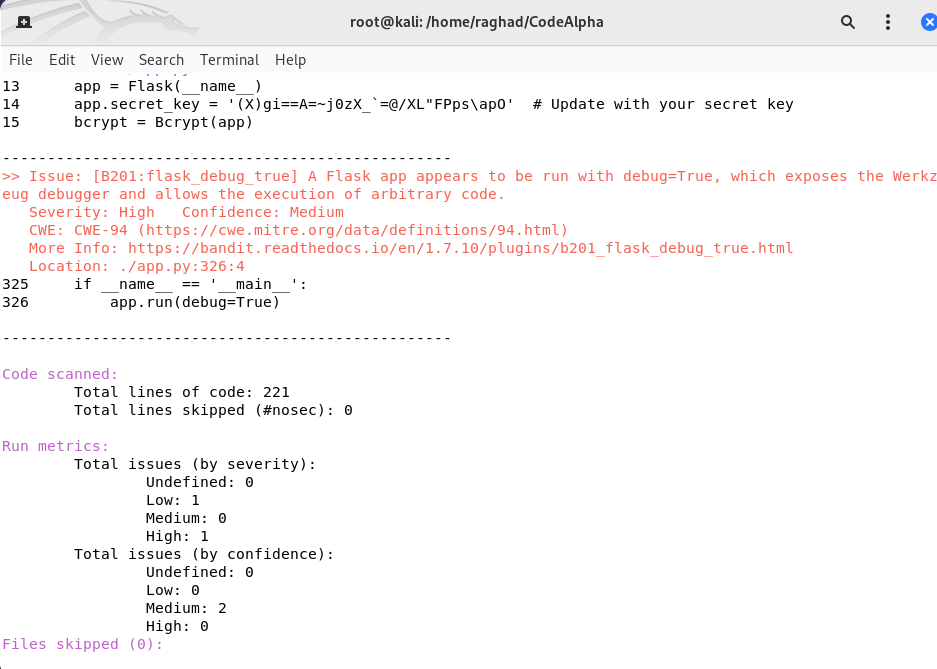
****

after saving the code in app.py

**Run Analyzer (Bandit)**

* bandit -r app.py

****

****

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')**

1. 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')**