Secure Coding Review Report: Flask Web App

1. Application Overview

App Name: Simple Notes Manager

Stack: Python (Flask) + SQLite

Functionality: Login, view notes, add new note

2. Sample Code (Before Review)

```
from flask import Flask, request, render_template, redirect
import sqlite3

app = Flask(__name__)

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

    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
    cursor.execute(f"SELECT * FROM users WHERE username='{username}' AND password='{password}'")
    user = cursor.fetchone()
    conn.close()

    if user:
        return redirect('/dashboard')
    else:
        return "Invalid credentials"

@app.route('/add_note', methods=['POST'])
def add_note():
    title = request.form['title']
```

```
    content = request.form['content']

    conn = sqlite3.connect('notes.db')
    cursor = conn.cursor()
    cursor.execute(f"INSERT INTO notes (title, content) VALUES ('{title}', '{content}')")
    conn.commit()
    conn.close()

    return redirect('/dashboard')
```

3. Vulnerability Analysis

- SQL Injection: vulnerable due to string formatting in SQL
- No Input Validation: no sanitation of form data
- Plaintext Passwords: stored and compared directly
- No Session Handling: missing authentication/session tracking
- XSS: unsanitized note content
- No CSRF Protection: no tokens to prevent CSRF

4. Static Analysis Tool: Bandit

Command:

```
    bandit app.py
```

Sample Output:

[HIGH] Possible SQL injection via string-based query

5. Remediation (Secure Code Examples)

- Use parameterized queries:

```
    cursor.execute("SELECT password FROM users WHERE username=?", (username,))
```

- Use password hashing:

  from werkzeug.security import check_password_hash

  check_password_hash(hash, password)

- Use sessions:

  session['user'] = username

- Add CSRF tokens with Flask-WTF
- Validate inputs with `.strip()` and length checks

6. Secure Coding Best Practices

1. Use parameterized queries
2. Hash passwords securely
3. Use CSRF tokens
4. Sanitize inputs and outputs
5. Implement proper session management
6. Avoid hardcoded secrets
7. Use static code analysis tools
8. Avoid exposing sensitive error logs

7. Findings Summary

Issue: SQL Injection | Severity: High | Fix: Use parameterized queries

Issue: No Password Hashing | Severity: High | Fix: Use password hashing

Issue: No Session | Severity: Medium | Fix: Use Flask session

Issue: XSS | Severity: High | Fix: Sanitize input/output

Issue: No CSRF | Severity: Medium | Fix: Add CSRF tokens

Issue: No Input Validation | Severity: Medium | Fix: Add input checks