TASK 3: Secure Coding Review

Name: Mahi Patel
Date: December 28, 2025

1. Application chosen

- Language: Python
- Framework: Flask
- Purpose: Simple user login system using SQLite

2. Vulnerabilities identified (manual review + Bandit)

- Critical: SQL Injection (CWE-89)
  → Caused by f-string concatenation of user input into SQL query
  → Bandit found: B608 (hardcoded_sql_expressions)
- High: Plaintext password storage
- Medium: Debug mode enabled
- Others: No input validation, potential for brute-force / CSRF

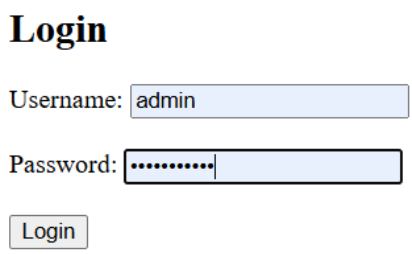
CODE:

sniffer.py app.py

app.py > login

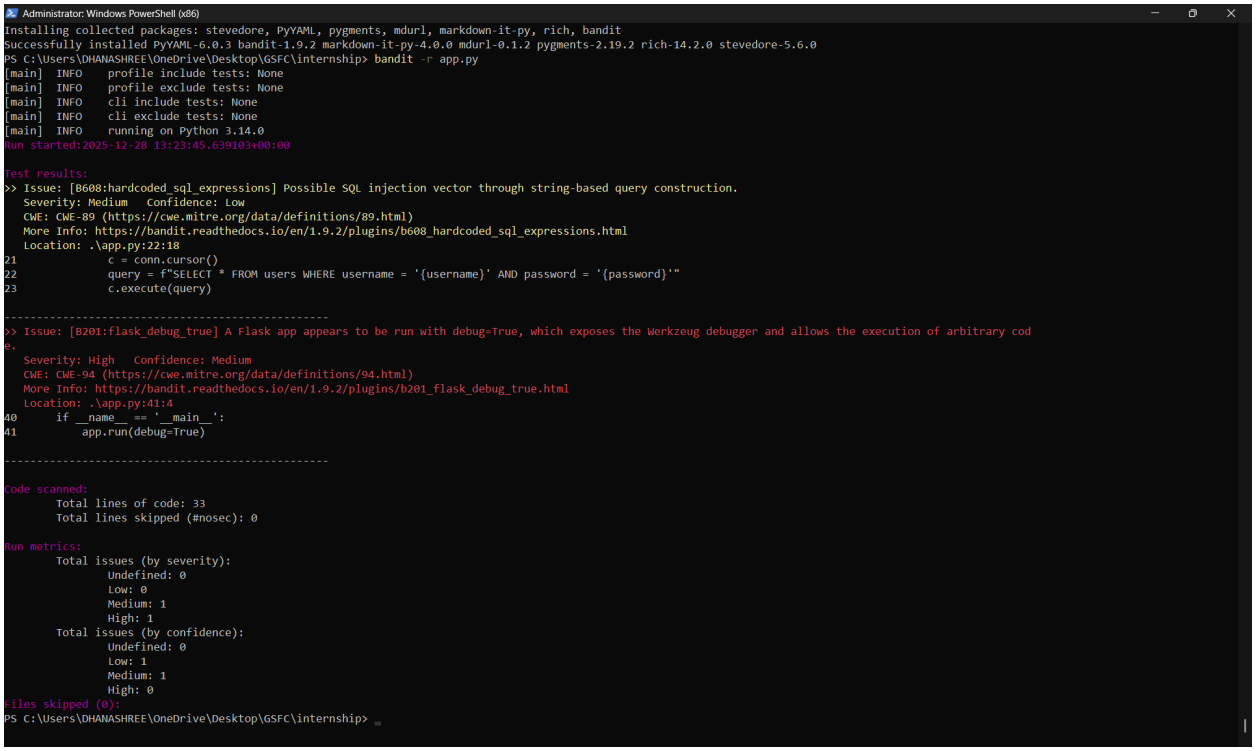
```python
from flask import Flask, request, render_template_string
import sqlite3

app = Flask(__name__)

# Create a simple database
conn = sqlite3.connect('users.db')
c = conn.cursor()
c.execute('''CREATE TABLE IF NOT EXISTS users (username TEXT, password TEXT)''')
c.execute("INSERT INTO users VALUES ('admin', 'password123')")
conn.commit()
conn.close()

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

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

        if result:
            return "Login successful!"
        else:
            return "Invalid credentials."

    return '''
    <form method="post">
        Username: <input type="text" name="username"><br>
        Password: <input type="text" name="password"><br>
        <input type="submit" value="Login">
    </form>
    '''

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

OUTPUT:



3. Bandit results – Before fix

[Paste or screenshot the original Bandit output showing B608 here]



4. Remediation steps applied

- Used parameterized queries (? placeholders) → prevents SQL injection
- Stored passwords using bcrypt hashing + salt
- Compared passwords securely with bcrypt.checkpw()
- Disabled debug mode (app.run(debug=False))
- Added basic required fields in HTML form

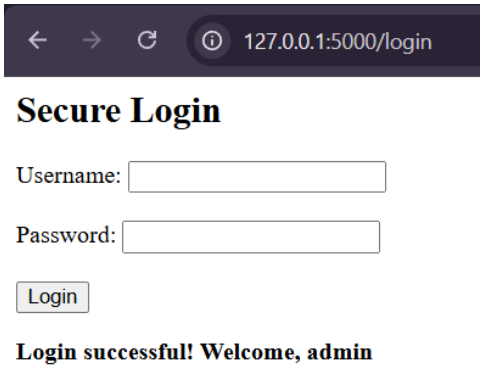
REVISED CODE:

app.py

C: > Users > DHANASHREE > OneDrive > Desktop > GSFC > internship > task3 > app.py > init_db

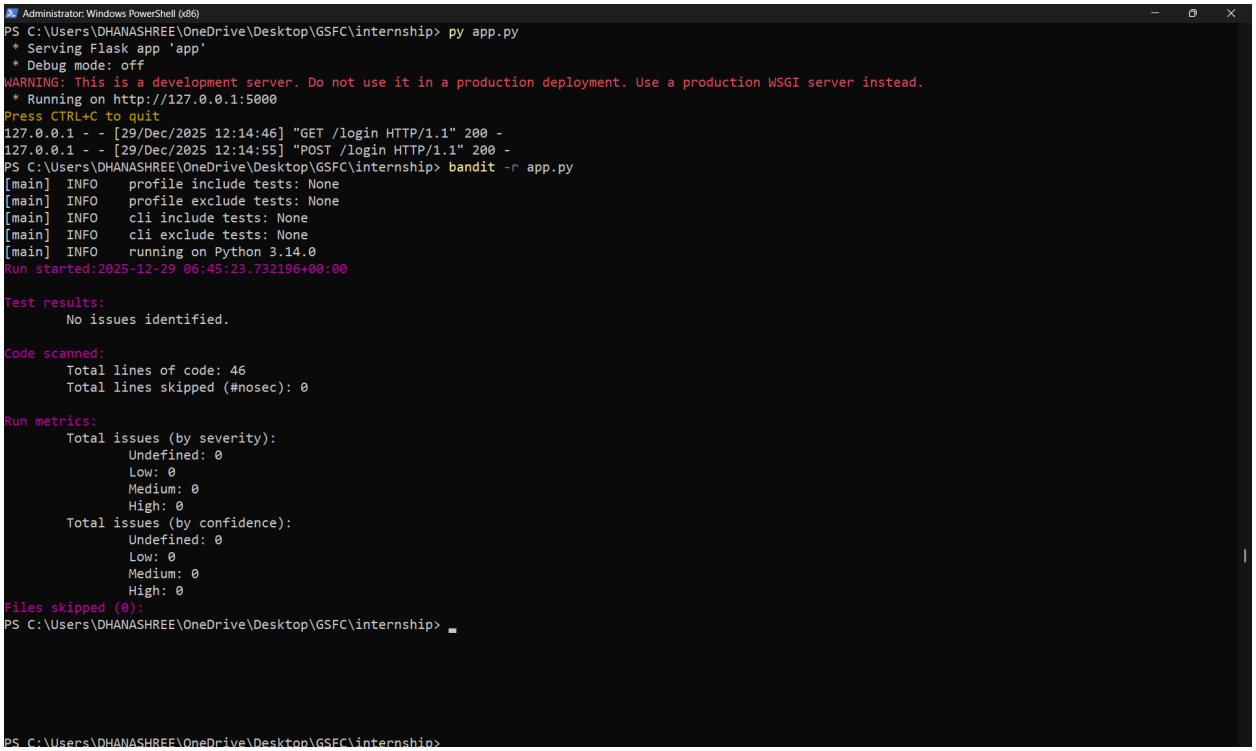
```python
from flask import Flask, request
import sqlite3
import bcrypt

app = Flask(__name__)

# Initialize database - store hash as BLOB (bytes) for proper bcrypt handling
def init_db():
    conn = sqlite3.connect('users.db')
    conn.execute("PRAGMA foreign_keys = 1")
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS users
                 (username TEXT UNIQUE, password BLOB)''')  # Changed to BLOB

    # Hash password as bytes
    password = 'password123'.encode('utf-8')
    hashed = bcrypt.hashpw(password, bcrypt.gensalt())

    # Insert admin user (as bytes)
    try:
        c.execute("INSERT INTO users (username, password) VALUES (?, ?)",
                  ('admin', hashed))  # hashed is already bytes
    except sqlite3.IntegrityError:
        pass

    conn.commit()
    conn.close()

init_db()

@app.route('/login', methods=['GET', 'POST'])
def login():
    message = ''
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password'].encode('utf-8')

        conn = sqlite3.connect('users.db')
        c = conn.cursor()
        c.execute("SELECT password FROM users WHERE username = ?", (username,))
        result = c.fetchone()
        conn.close()

        if result and bcrypt.checkpw(password, result[0]):  # result[0] is now bytes!
            message = f"Login successful! Welcome, {username}"
        else:
            message = "Invalid credentials."

    return '''
    <h2>Secure Login</h2>
    <form method="post">
        Username: <input type="text" name="username" required><br><br>
        Password: <input type="password" name="password" required><br><br>
        <input type="submit" value="Login">
    </form>
    <p><strong>{}</strong></p>
    '''.format(message)

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

OUTPUT:



5. Bandit results – After fix

[Paste or screenshot the clean Bandit output here – should show 0 high/medium issues]



6. Best practices learned / recommended

- Never concatenate user input into SQL queries → always use parameters
- Never store plaintext passwords → use bcrypt, Argon2 or similar
- Disable debug mode in any non-development environment
- Validate/sanitize inputs and use CSRF protection in real apps
- Run static analyzers (Bandit, Semgrep) regularly in CI/CD

7. Conclusion

The original code was vulnerable to easy authentication bypass via SQL injection.
After fixes, the application is significantly more secure.
Tools like Bandit help catch issues early.