

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  X
app.py > login
1  from flask import Flask, request, render_template_string
2  import sqlite3
3
4  app = Flask(__name__)
5
6  # Create a simple database
7  conn = sqlite3.connect('users.db')
8  c = conn.cursor()
9  c.execute('''CREATE TABLE IF NOT EXISTS users (username TEXT, password TEXT)''')
10 c.execute('INSERT INTO users VALUES ('admin', 'password123')')
11 conn.commit()
12 conn.close()
13
14 @app.route('/login', methods=['GET', 'POST'])
15 def login():
16     if request.method == 'POST':
17         username = request.form['username']
18         password = request.form['password']
19
20         conn = sqlite3.connect('users.db')
21         c = conn.cursor()
22         query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
23         c.execute(query)
24         result = c.fetchone()
25         conn.close()
26
27         if result:
28             return "Login successful!"
29         else:
30             return "Invalid credentials."
31
32     return '''
33     <form method="post">
34         Username: <input type="text" name="username"><br>
35         Password: <input type="text" name="password"><br>
36         <input type="submit" value="Login">
37     </form>
38     '''
39
40 if __name__ == '__main__':
41     app.run(debug=True)
```

OUTPUT:

Login

Username:

Password:

Login

3. Bandit results – Before fix

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

```
Administrator: Windows PowerShell (x86)
Installing collected packages: stevedore, PyYAML, pygments, mdurl, markdown-it-py, rich, bandit
Successfully installed PyYAML-6.0.3 bandit-1.9.2 markdown-it-py-4.0.0 mdurl-0.1.2 pygments-2.19.2 rich-14.2.0 stevedore-5.6.0
PS C:\Users\DHANASHREE\OneDrive\Desktop\GSFC\internship> bandit -r app.py
[main] INFO     profile include tests: None
[main] INFO     cli include tests: None
[main] INFO     cli exclude tests: None
[main] INFO     running on Python 3.14.0
Run started:2025-12-28 13:23:45.639103+00:00

Test results:
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
Severity: Medium    Confidence: Low
CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
More Info: https://bandit.readthedocs.io/en/1.9.2/plugins/b608_hardcoded_sql_expressions.html
Location: .\app.py:22:18
21         c = conn.cursor()
22         query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
23         c.execute(query)

>> Issue: [B201:flask_debug_true] A Flask app appears to be run with debug=True, which exposes the Werkzeug 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/1.9.2/plugins/b201_flask_debug_true.html
Location: .\app.py:41:4
40     if __name__ == '__main__':
41         app.run(debug=True)

Code scanned:
Total lines of code: 33
Total lines skipped (#nosec): 0

Run metrics:
Total issues (by severity):
  Undefined: 0
  Low: 0
  Medium: 1
  High: 1
Total issues (by confidence):
  Undefined: 0
  Low: 1
  Medium: 1
  High: 0
Files skipped (0):
PS C:\Users\DHANASHREE\OneDrive\Desktop\GSFC\internship>
```

4. Remediation steps applied

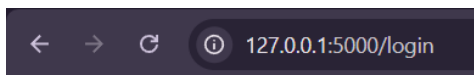
- Used parameterized queries (? placeholders) → prevents SQL injection
- Stored passwords using bcrypt hashing + salt
- Compared passwords securely with bcrypt.checkpw()
- Disabled debug mode (app.run(debug=False))
- Added basic required fields in HTML form

REVISED CODE:

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

1 from flask import Flask, request
2 import sqlite3
3 import bcrypt
4
5 app = Flask(__name__)
6
7 # Initialize database - store hash as BLOB (bytes) for proper bcrypt handling
8 def init_db():
9     conn = sqlite3.connect('users.db')
10    conn.execute("PRAGMA foreign_keys = 1")
11    c = conn.cursor()
12    c.execute('CREATE TABLE IF NOT EXISTS users
13              (username TEXT UNIQUE, password BLOB)') # Changed to BLOB
14
15    # Hash password as bytes
16    password = 'password123'.encode('utf-8')
17    hashed = bcrypt.hashpw(password, bcrypt.gensalt())
18
19    # Insert admin user (as bytes)
20    try:
21        c.execute("INSERT INTO users (username, password) VALUES (?, ?)",
22                  ('admin', hashed)) # hashed is already bytes
23    except sqlite3.IntegrityError:
24        pass
25
26    conn.commit()
27    conn.close()
28
29 init_db()
30
31 @app.route('/login', methods=['GET', 'POST'])
32 def login():
33     message = ''
34     if request.method == 'POST':
35         username = request.form['username']
36         password = request.form['password'].encode('utf-8')
37
38         conn = sqlite3.connect('users.db')
39         c = conn.cursor()
40         c.execute("SELECT password FROM users WHERE username = ?", (username,))
41         result = c.fetchone()
42         conn.close()
43
44         if result and bcrypt.checkpw(password, result[0]): # result[0] is now bytes!
45             message = f"Login successful! Welcome, {username}"
46         else:
47             message = "Invalid credentials."
48
49     return '''
50     <h2>Secure Login</h2>
51     <form method="post">
52         Username: <input type="text" name="username" required><br><br>
53         Password: <input type="password" name="password" required><br><br>
54         <input type="submit" value="Login">
55     </form>
56     <p><strong>{}</strong></p>
57     '''.format(message)
58
59 if __name__ == '__main__':
60     app.run(debug=False)
```

OUTPUT:



Secure Login

Username:

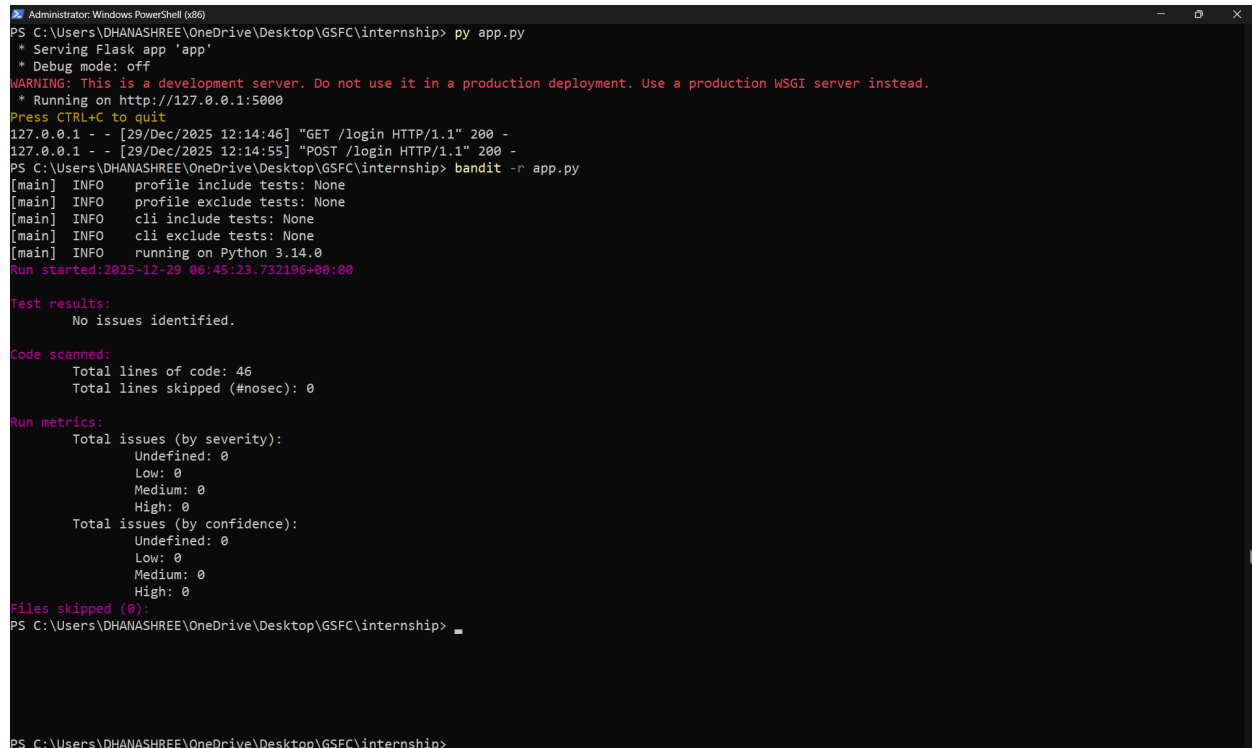
Password:

Login

Login successful! Welcome, admin

5. Bandit results – After fix

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



```
Administrator: Windows PowerShell (x86)
PS C:\Users\DHANASHREE\OneDrive\Desktop\GSFC\internship> py app.py
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [29/Dec/2025 12:14:46] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [29/Dec/2025 12:14:55] "POST /login HTTP/1.1" 200 -
PS C:\Users\DHANASHREE\OneDrive\Desktop\GSFC\internship> 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 3.14.0
Run started:2025-12-29 06:45:23.732196+00:00

Test results:
No issues identified.

Code scanned:
Total lines of code: 46
Total lines skipped (#nosec): 0

Run metrics:
Total issues (by severity):
Undefined: 0
Low: 0
Medium: 0
High: 0
Total issues (by confidence):
Undefined: 0
Low: 0
Medium: 0
High: 0
Files skipped (0):
PS C:\Users\DHANASHREE\OneDrive\Desktop\GSFC\internship> _
```

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.