📑 **Secure Coding Review Report**

Internship Task 3: Code Review & Secure Development

Submitted by: Shubham Srivastva
Organization: CodeAlpha
College: Budge Budge Institute of Technology
Date: 10/07/2025

🚀 Cover Page

Secure Coding Review Report
Internship Task 3: Code Review & Secure Development

Submitted by:
Shubham Srivastva
Cybersecurity Intern, CodeAlpha

College:
Budge Budge Institute of Technology

Date:
10/07/2025

📌 Project Overview

- **Programming Language:** Python
- **Framework:** Flask
- **Module Reviewed:** Web-based Login System
- **Tools Used:** Bandit, PyLint, Manual Inspection
- **Objective:** Identify and remediate security vulnerabilities in a basic login system

---

🧩 Original Application Code

python

CopyEdit

```
@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')
```

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

    if user:
        return "Login successful!"
    else:
        return "Invalid credentials."
```

🔎 Security Vulnerability Analysis

Issue	Description	Severity
SQL Injection	Raw SQL query uses user input directly	High
Plaintext Passwords	Passwords stored and compared in plaintext	High
Lack of Input Validation	Inputs not sanitized or validated	Medium
No HTTPS Enforcement	Credentials may be intercepted over insecure connections	High
Hardcoded SQL Strings	Query construction via string formatting	Medium

---

🛠️ Tools Used for Review

- ✅ Bandit (Python Security Linter)
- ✅ PyLint (style and error checking)
- ✅ Manual Code Inspection

🛡️ Recommendations with Secure Code Examples

✅ 1️⃣ Use Parameterized Queries

Avoid string formatting in SQL:

python

CopyEdit

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

✅ 2️⃣ **Hash Passwords Securely**

Never store plaintext passwords. Use bcrypt:

python

CopyEdit

import bcrypt

Registration

hashed_pw = bcrypt.hashpw(password.encode(), bcrypt.gensalt())

Store hashed_pw in DB

Login comparison

bcrypt.checkpw(input_pw.encode(), stored_hash)

✅ 3️⃣ **Input Validation**

Always validate user inputs:

python

CopyEdit

if not username.isalnum():

  return "Invalid input."

✅ 4️⃣ **Enforce HTTPS & Secure Headers**

Use Flask-Talisman:

python

CopyEdit

from flask_talisman import Talisman

Talisman(app)

✅ 5️⃣ **Secure Session Management**

Replace insecure responses with session handling:

python

CopyEdit

from flask import session

session['user'] = username

✅ Always set app.secret_key securely.

📈 **Remediation Summary Table**

Issue	Recommended Remediation
SQL Injection	Use parameterized queries
Plaintext Passwords	Hash passwords using bcrypt
Input Validation Missing	Add length/type/format checks
Insecure Transport	Enforce HTTPS with Flask-Talisman
Poor Session Handling	Use Flask sessions with secure secret_key

---

📌 **Final Recommendations**

Security is an ongoing process. Adopting secure coding practices helps protect user data, maintain trust, and reduce risk. Developers should prioritize code reviews, static analysis, and secure design patterns in production applications.

📁 **Deliverables Included**

✅ Secure Coding Report (this document)
✅ Remediated Sample Code (.py file)
✅ Static Analysis Logs (Bandit output, optional)
✅ Google Slides version (available on request)

✅ **Prepared By**

Shubham Srivastva
Cybersecurity Intern
Organization: CodeAlpha
College: Budge Budge Institute of Technology
Date: 10/07/2025
