Secure Code Review

Manual Code Review for Security and Performance Issues on Flask Web Application

---

Identified Issues and Recommendations

1. SQL Injection Vulnerability (Critical)

Issue:

The code directly concatenates user input into the SQL query:

```
query = "SELECT * FROM users WHERE username = '" + username + "' AND password = '" + password + "'"cursor.execute(query)  # SQL Injection vulnerability
```

Why is this a problem?
An attacker can manipulate the input to execute arbitrary SQL commands, exposing sensitive data or modifying the database.

Solution: Use Parameterized Queries

```
query = "SELECT * FROM users WHERE username = ? AND password = ?"

cursor.execute(query, (username, password))
```

---

2. Plaintext Password Storage (Critical)

Issue:

The application compares passwords in plaintext without hashing.

Solution: Hash Passwords Using werkzeug.security

Modify the code to store and verify hashed passwords.

```
from werkzeug.security import check_password_hash

query = "SELECT * FROM users WHERE username = ?"

cursor.execute(query, (username,))

user = cursor.fetchone()

if user and check_password_hash(user["password"], password):

    return "<h1>Login successful</h1>"

else:

    return "<h1>Invalid credentials</h1>"
```

Use generate_password_hash(password) when storing user passwords.

---

3. Debug Mode Enabled (High Risk)

Issue:

app.run(debug=True)

Running Flask in debug mode exposes the interactive debugger, which can be exploited in production.

Solution:

Use environment variables to control debug mode:

import os

debug_mode = os.getenv("FLASK_DEBUG", "False").lower() == "true"

app.run(debug=debug_mode)

---

4. Connection Pooling for Performance Optimization

Issue:

The function get_db_connection() opens a new connection every time, which can slow down the application.

Solution:

Use sqlite3 with connection pooling or switch to SQLAlchemy for better performance:

from flask_sqlalchemy import SQLAlchemy

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///database.db'

db = SQLAlchemy(app)

---

5. XSS Vulnerability (Medium Risk)

Issue:

Using render_template_string directly can allow XSS attacks if user input is included in the HTML.

return render_template_string('<form>...</form>'

Solution:

Use proper HTML templates stored in a separate .html file.

```
from flask import render_template
@app.route('/login', methods=['GET', 'POST'])
def login():
    return render_template("login.html")
```