Secure Coding Review Report

1. Introduction

In this report, I have conducted a secure coding review of a simple Python-based Flask login application. The goal of the review is to identify common security vulnerabilities, suggest best practices for mitigation, and demonstrate how static code analysis tools can assist in strengthening code security.

2. Application Overview

The application reviewed is a basic login system where users submit their credentials to access a protected dashboard. The backend is written using the Flask web framework in Python.

The application's key features:
- Accepts username and password from users.
- Authenticates credentials.
- Redirects authenticated users to a dashboard page.

3. Code Under Review

Code snippet has been provided within the report document due to size limitations.

4. Vulnerabilities Identified

- Hardcoded Credentials
- Plaintext Password Storage
- No Brute Force Protection
- Debug Mode Enabled
- No HTTPS Enforcement
- Missing CSRF Protection

5. Recommendations for Secure Coding

- Use Database and Environment Variables
- Hash Passwords
- Implement Login Rate Limiting

- Disable Debug Mode
- Enforce HTTPS
- Add CSRF Protection

6. Static Code Analysis

Tool Used: Bandit

Command Used: bandit -r flask_app/

Findings:

- B105: Hardcoded password.
- B201: Debug mode is enabled.

This confirms manual review results.

7. Improved Secure Code

Improved code provided separately with security fixes:

- Passwords hashed
- Session management added
- Debug mode disabled

8. Conclusion

Through this secure coding review, significant flaws were identified and solutions proposed. Static code analysis tools like Bandit help to catch vulnerabilities early. Secure coding practices are essential for protecting applications and user data.