Secure Coding Review Report

Intern: Utsav Shrivastav

Organization: CodeAlpha Internship

Task 3: Secure Coding Review

Date: 20th Sep, 2025

1. Application Overview

- Language: Python 3.x
- Application: Simple login system with username/password authentication
- Purpose: Demonstrate user authentication process and handling of credentials

Python Code (Vulnerable Version):

```
# login.py

# Hardcoded credentials
username_db = "admin"
password_db = "admin123"

def login():
    username = input("Enter username: ")
    password = input("Enter password: ")

    # Vulnerable comparison
    if username == username_db and password == password_db:
        print("Login successful!")
    else:
        print("Invalid credentials!")

if __name__ == "__main__":
    login()
```

2. Methodology

Manual code inspection: Looked for:
Input validation
Hardcoded credentials

Authentication issues
Error handling

Automated tool: Bandit
pip install bandit
bandit -r SecureApp/
Identified security vulnerabilities and classified them based on severity (High, Medium, Low).

3. Findings Table

Vulnerability	Location	Risk Level	Description	Recommendation
Hardcoded password	login.py line 4	High	Password stored as plain text	Use hashed passwords (bcrypt) stored securely
No input validation	login.py line 7	Medium	User inputs directly compared	Sanitize and validate all inputs
Plain error messages	login.py line 10	Low	Generic error messages could expose info	Log internally; show generic message to users

4. Recommendations & Remediation Steps

Short-Term Fixes:
1. Remove hardcoded credentials; store in a secure file or database.
2. Use hashed passwords with bcrypt:
import bcrypt
password = input("Enter password: ")
hashed = bcrypt.hashpw(password.encode(), bcrypt.gensalt())
3. Validate all user inputs; disallow suspicious characters.
4. Show generic error messages, log actual errors internally.

Long-Term Fixes:
- Implement secure coding standards.
- Conduct regular static analysis using tools like Bandit or SonarQube.
- Maintain an updated dependency list to avoid vulnerable libraries.

5. Conclusion

The review identified hardcoded passwords, lack of input validation, and insecure error handling as the primary vulnerabilities. Implementing the recommended remediation steps will significantly improve the security posture of the application and prevent common attacks such as credential theft and injection attacks.