Secure Coding Review — Mahmoud Saad Sarhan

CodeAlpha Cyber Security Internship

Objective

The goal of this secure coding review is to check a simple Python Flask application for common security weaknesses.
The review identifies the problems, explains their impact, and shows how to fix them using secure coding practices.

Reviewed Code (Before Fixing)

castillo.py

```python
from flask import Flask, request
import sqlite3

app = Flask(__name__)
conn = sqlite3.connect('users.db', check_same_thread=False)
cursor = conn.cursor()

@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    query = f"SELECT * FROM users WHERE username='{username}' AND password='{password}'"
    cursor.execute(query)
    user = cursor.fetchone()
    if user:
        return "Login successful"
    else:
        return "Invalid credentials"

if __name__ == "__main__":
    app.run(debug=True)
```

Application Description

The reviewed code is a basic login feature built with Flask and SQLite.
It allows users to submit their username and password through a POST request and verifies them directly from the database.

Main Issues Found

1. SQL Injection

- **Problem:** The SQL query includes user input directly inside the string.
- **Impact:** Attackers can inject SQL commands to bypass authentication or read sensitive data.
- **Fix:** Use parameterized queries with placeholders instead of string concatenation.

Example (before):

```
query = f"SELECT * FROM users WHERE username='{username}' AND password='{password}'"

cursor.execute(query)
```

Example (after):

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

---

2. Plaintext Passwords

- **Problem:** Passwords are stored and compared in plain text.
- **Impact:** If the database is leaked, all user passwords are exposed.
- **Fix:** Use hashing with salt, for example bcrypt.

Example fix:

```
import bcrypt

hashed = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, hashed)) # Parameterized Query to prevent SQL Injection
```

---

3. Missing Input Validation

- **Problem:** No validation on form data.
- **Impact:** Attackers can send unexpected or malicious input that may break the system.
- **Fix:** Validate input length, type, and format before processing.

---

4. Debug Mode Enabled

- **Problem:** debug=True in Flask can expose detailed error messages and stack traces.
- **Impact:** Leaks sensitive technical information.
- **Fix:** Always disable debug mode in production by setting debug=False or removing the flag.

Improved Code Summary

The corrected version of the application:

- Uses parameterized SQL queries.
- Stores passwords securely with hashing.
- Validates user inputs.
- Runs Flask safely without debug mode.

Reviewed Code (After Fixing)

castillo.py

```python
from flask import Flask, request
import sqlite3, bcrypt

app = Flask(__name__)

@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']

    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()

    cursor.execute("SELECT password FROM users WHERE username=?", (username,))
    user = cursor.fetchone()

    if user and bcrypt.checkpw(password.encode('utf-8'), user[0].encode('utf-8')):
        return "Login successful"
    else:
        return "Invalid credentials"

if __name__ == "__main__":
    app.run()
```

Conclusion

Following secure coding practices greatly reduces the risk of attacks like SQL injection and data exposure.
Simple steps such as validating inputs, hashing credentials, and avoiding debug mode make web applications safer and more reliable.

This review demonstrates how small code improvements can prevent serious security issues and ensure better data protection for users.

Acknowledgement

I extend my sincere gratitude to the **CodeAlpha** for this valuable opportunity, which allowed me to deepen my understanding and practical application of secure coding best practices.
