Secure Coding Review Report

Selected Language: Python

Selected Application: Simple Login Authentication System

Reason for selection:

- Python is widely used and easy to understand
- Authentication systems are common and security-critical
- Contains typical beginner-level security issues suitable for review

Insecure Code Sample

```
username = input("Enter username: ")
password = input("Enter password: ")
if username == "admin" and password == "admin123":
    print("Login successful")
else:
    print("Login failed")
```

Identified Vulnerabilities:

Hardcoded Credentials

Username and password are directly written in the code

Plaintext Password Handling

Password is not encrypted or hashed

Lack of Input Validation

User input is taken without validation or sanitization

Findings:

- Hardcoded credentials
- Plaintext password handling
- Lack of input validation

Tools Used:

- **Manual Code Review**
  - Line-by-line inspection to identify insecure practices
- **Static Analysis Tool**

- **Bandit** (Python security linter)

Tool Command:

bandit insecure.py

Bandit detects:

- Hardcoded passwords
- Weak security practices

Recommendations:

- **Do not store credentials directly in code**
- **Hash passwords before storing or comparing**
- **Validate all user inputs**
- **Use environment variables for sensitive data**
- **Follow least privilege principles**
- **Perform regular security code review**

Secure Version of the Code

```
import hashlib

stored_password_hash = hashlib.sha256("admin123".encode()).hexdigest()
```

```
username = input("Enter username: ")
password = input("Enter password: ")
hashed_input = hashlib.sha256(password.encode()).hexdigest()
if username == "admin" and hashed_input == stored_password_hash:
    print("Login successful")
else:
    print("Login failed")
```

Conclusion:

Applying secure coding techniques significantly improves application security and reduces risk.