Secure Code Review Report

Task 3 – Secure Coding Review

Reviewer: Natinael Tesfaye

Language: Python

Tool Used: Bandit (via `python -m bandit vuln_script.py`)

Date: July 15, 2025

📁 Application Reviewed:

Filename: vuln_script.py

Description: A simple Python script simulating user input, password handling, and file listing functionality.

🔍 Vulnerabilities Identified:

1. Hardcoded Password

Location: Line 5

Code:

```
password = "123456"
```

Issue: Passwords should never be hardcoded in source code. If the code is exposed, attackers gain instant access.

Risk Level: Medium

Recommendation:

Use environment variables or secure vaults like AWS Secrets Manager or .env files.

Example:

```
import os
password = os.getenv("APP_PASSWORD")
```

2. Insecure Use of eval()

Location: Line 8

Code:

print(eval(user_input))

Issue: eval() executes arbitrary Python code, which can be abused by attackers to run malicious commands.

Risk Level: High

Recommendation:

Use safer alternatives like ast.literal_eval() if evaluating simple expressions.

Example:

import ast

print(ast.literal_eval(user_input))

3. Command Injection via sub process.call ()

Location: Line 12

Code:

subprocess.call("ls " + filename, shell=True)

Issue: If the user inputs a malicious command (e.g., ; rm -rf /), it gets executed on the system.

Risk Level: High

Recommendation:

Avoid shell=True when using subprocess.

Use list format to separate command and arguments safely.

Example:

subprocess.run(["ls", filename])

🛠️ Secure Coding Best Practices (General Recommendations):

- Never hardcode sensitive information like passwords or tokens.
- Avoid using dangerous functions like eval() unless strictly necessary.
- Always sanitize and validate user input.
- Use secure coding tools like Bandit regularly to scan your code.
- Follow the principle of least privilege — don’t allow more than necessary.
- Keep dependencies up to date and use vulnerability scanners for third-party packages.

Conclusion:

This code review identified three critical security issues. By following the provided recommendations and adopting secure coding practices, the code will become more secure, resilient, and safe from common exploits like injection attacks and code execution.

Fixed Version of vuln_script.py

```
import os
import subprocess
import ast

✅ Secure way to handle passwords using environment variables
password = os.getenv("APP_PASSWORD")

✅ Safe use of user input with ast.literal_eval()
user_input = input("Enter math expression: ")
try:
    result = ast.literal_eval(user_input)
    print(result)
except (ValueError, SyntaxError):
```

```
    print("Invalid input!")

✅ Secure subprocess call without shell=True
filename = input("Enter file name: ")
try:
    subprocess.run(["ls", filename], check=True)
except subprocess.CalledProcessError:
    print("Error listing files.")
```