# Vulnerability Assessment Report

## Report Scenario

As a software engineer, your assignment involves analyzing a piece of vulnerable code (imagine that the code you wrote was provided to you to analyze). Your task is to identify any vulnerabilities, discuss these findings as if you were analyzing them for the first time, and recommend mitigation strategies. It's crucial to approach this analysis with an objective lens to enhance the code's security and functionality.

Importantly, ensure you carefully read the content directions provided within square brackets ‘[ ]’ in the report template. Replace these instructions with your specific findings and recommendations, removing the brackets to streamline your report. This exercise aims to develop your skills in critical assessment and improvement of software from an analytical and objective standpoint.

## Submission Checklist

Complete this checklist, marking each requirement with an X if complete, after finishing this report and before you submit your assignment.

|  |  |  |
| --- | --- | --- |
| X | Req # | Description |
|  | 1 | 2 CWE/SANS Top 25 Vulnerabilities were selected from the correct category for this assignment. |
|  | 2 | Report is well written, text sections should be 1-3 paragraphs each and describe your specific vulnerability and mitigation steps required. |
|  | 3 | References are cited (minimum of 2 references) |
|  | 4 | Screenshots include the snippets of vulnerable and mitigated code and any needed to demonstrate your vulnerable code and mitigation was successful. Additionally, the screenshots are easily viewable without needed to zoom in (they should be sized to the margin-to-margin width of your document, and do not include an excessive amount of unnecessary areas of your screen). |
|  | 5 | VAR Template was followed and the highlighted sections with square brackets ‘[ ]’ were changed as required and the [ ] and template content were removed. |
|  | 6 | Source files are to be submitted as a ZIP, separately from your report, maintaining the necessary folder structures to run your code. |
|  | 7 | This report is to be submitted as a PDF (\*preferred) or a DOCX, as a separate file than your source code. |
|  | 8 | The Requirements section is filled out for each CWE. |

|  |
| --- |
| Computer script on a screen |
| Vulnerability Assessment Report  Assignment #[X] |
| |  |  |  | | --- | --- | --- | | [Ahmed Ashade] | 6/26/25 | UMGC SDEV 325 | |

# Executive Summary

This report demonstrates two examples of insecure interaction between components using Python code in the AWS Cloud9. The 2 selected vulnerabilities are:

1. CWE-89: Improper Neutralization of Special Elements used in a SQL Command.
2. CWE-78: Improper Neutralization of Special Elements used in an OS Command

Each of them includes a vulnerable code sample and an updated, secure version.

# Task 1 – CWE-89: SQL Injection

## Overview

This example is a simple user authentication lookup script written in Python using SQLite. The user is prompted to enter a username, which is used to query a local database. The application is vulnerable to SQL injection due to insecure query formatting.

**Vulnerable Code**

## Vulnerable Code

### Requirements List

1. Python
2. Sqlite3
3. A SQLite database file user. dB with a user’s table

### Analysis of the Vulnerability

This script directly embeds user input into an SQL query string. An attacker could enter a crafted string like admin OR 1 = 1, leading to a SQL injection. The resulting query would return all users, bypassing authentication.

A screenshot of a computer program

AI-generated content may be incorrect.

## Mitigated Code

### Requirements List

1. Python
2. Sqlite3
3. A SQLite database file user. dB with a user’s table

### Analysis of the Mitigation

The mitigation uses a parameterized query with a placeholder (?). The user input is passed separately as a tuple, which prevents the input from being interpreted as part of the SQL Logic. Thius approach effectively neutralizes special characters and prevents SQL Injection.

A screenshot of a computer program

AI-generated content may be incorrect.

# Task 2 – CWE-78: OS Command Injection

## Overview

This example demonstrates a command line tool written in Python that allows users to enter a file name and list its contents using the ls command. The input is directly passed to the shell, making it vulnerable to OS Command

## Vulnerable Code

### Requirements List

1. Python 3
2. Linux shell with ls command

### Analysis of the Vulnerability

This code passes user input directly into an os.system() call. An attacker could inject commands like : rm -rf /. Resulting in critical system compromise.

A screen shot of a computer

AI-generated content may be incorrect.

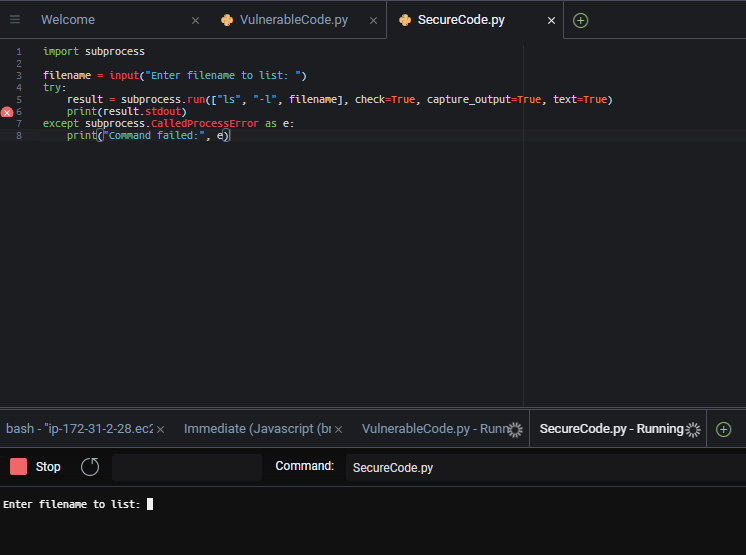
## Mitigated Code

### Requirements List

1. Python 3
2. Linux shell with ls command

### Analysis of the Mitigation

Using subprocess.run() with a list of arguments prevent shell interpretation of the input. The command is executed directly without invoking a shell, meaning injected characters or delimiters like :, &&, or | have no effect. This method eliminates the risk of OS command injection



# References

* CWE-89: SQL Injection
* CWE-78: OS Command Injection
* Python official documentation on [sqlite3](https://docs.python.org/3/library/sqlite3.html) and [subprocess](https://docs.python.org/3/library/subprocess.html)