Graded Assignment: Homework 2

University of Maryland Global Campus

SDEV 325: Detecting Software Vulnerabilities

# Executive Summary

(Updated A/N 5/9/2023): I again failed to make the codes work, but hopefully I have managed to at least somewhat accurately depict each vulnerability).

(A/N: I had great difficulty understanding this week’s materials, and began late, so didn’t have the time to ask for help. As a result, my homework is probably awful).

In this week’s assignment, I chose to demonstrate the vulnerabilities of SQL Injection (CWE-89) and OS Command Injection (CWE-78). The first occurs when an SQL query is written directly into the code, and can be externally modified by hackers, resulting in a program crash, data leaks, and other attacks. For the second one to occur, special elements that could alter a command are not handled correctly.

## Example 1- [ **Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') ]**

Overview:

To demonstrate this vulnerability, I first downloaded the MySQL server, and, in it, created the database “students”, with a table of the same name. The table contains the columns “student\_id”, “first\_name”, “last\_name”, and “graduation\_year”. I then attempted to establish a connection between my Java code and the database, intending to display the data for one of the students. Since I am not sure how to submit a database to the UMGC portal, I will just paste a screenshot of it here as proof of its creation.

Graphical user interface, text, application, email

Description automatically generated

Unfortunately, the code failed to load as expected, being unable to connect to the database:

Text

Description automatically generated

Analysis of the Vulnerability:

SQL Injection is the practice of injecting code (either malicious or one leading to undesirable consequences) into an unprotected SQL query. In this case, the query is simply written as a string, and not inside of a statement. An attacker could modify the program by injecting a command that is not supposed to be there, causing anything from data exposure to a crash.

Vulnerable Code:

A screenshot of a computer program

Description automatically generated with medium confidence



Vulnerable App Result:

A screen shot of a computer

Description automatically generated with medium confidence

Our program has failed to properly establish a connection with the database. However, we can still demonstrate how the code could be modified by inserting a command into the query.



By inserting the command name' OR 'a'='a into the program, we confuse it, and cause it to skip the rest of the query. This results in the program crashing.

Mitigation:

One of the ways to mitigate the issue of SQL Injection is to put the SQL query into a prepared statement. A prepared statement protects the query, not allowing attackers to inject any excess code in it.

Repaired Code:

A screen shot of a computer

Description automatically generated with medium confidence

After being written into the code as a string, the query is put into a prepared statement, avoiding attacks.

Repaired Code Result:

A computer screen with white text

Description automatically generated with low confidence

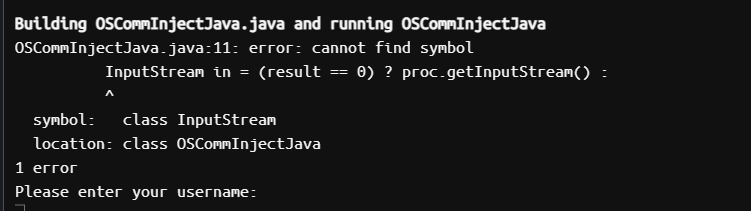
As we can see, the program still failed to connect to the database.

Example 2 – [CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')]

Overview:

To show the second vulnerability, I created a Java program that is supposed to load unrestricted data into the environment, which exposes it to potentially dangerous data and malware attacks.

Code failed to work correctly due to an error in the variables and symbols. However, I do suspect that it has many more errors than that.



Analysis of the Vulnerability:

OS Command Injection happens when an application does not have proper mechanisms to tackle injection of excess code into its operating system. In this case, the app accepts a big amount of unchecked, unregulated data from its environment. This can lead to the app becoming infected with malicious code, or receive dangerous commands.

Vulnerable Code:

A screen shot of a computer program

Description automatically generated with medium confidence

The code does not restrict what data can be accepted from the environment, therefore opening the program up for attacks. It is not exactly possible to fully demonstrate such an attack.

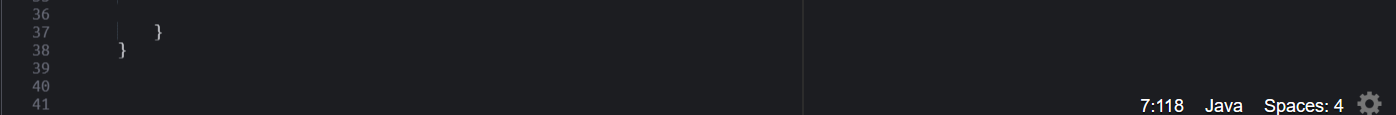
Mitigation:

One possible way to mitigate this is to restrict the characters that the app should accept from the environment. This way, any undesirable data will not be accepted, avoiding potential problems and attacks.

Repaired Code:

A screen shot of a computer code

Description automatically generated with low confidence



We now have specified a set of characters the program can use, so it will ignore all others.

References:

[CWE - 2022 CWE Top 25 Most Dangerous Software Weaknesses (mitre.org)](https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html)

[SQL Injection: The Equal Opportunity Vulnerability | Infosec Resources (infosecinstitute.com)](https://resources.infosecinstitute.com/topic/sql-injection-vulnerability/)

[OS Command Injection Defense - OWASP Cheat Sheet Series](https://cheatsheetseries.owasp.org/cheatsheets/OS_Command_Injection_Defense_Cheat_Sheet.html)

[CWE - 2022 CWE Top 25 Most Dangerous Software Weaknesses (mitre.org)](https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html)

[CWE - CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') (4.10) (mitre.org)](https://cwe.mitre.org/data/definitions/78.html)