**Unit 5 Assignment**

Laurence T. Burden

Purdue Global University

IN300: Data Governance

John Brooks

May 23, 2021

1. **Using the Java code below, describe the vulnerability that exists in the query and what can be done to fix the issue(s).**
2. **import** java.util.Scanner;
3. **public** **class** IN300\_Unit5\_A1 {
4. **public** **static** **void** main(String[] args) {
5. Scanner sc = **new** Scanner(System.***in***);
6. System.***out***.println("Enter employee name");
7. String employee = sc.nextLine();
8. String query = "SELECT \* FROM Employee where employeeName = '" + employee + "'";
9. System.***out***.println("The query is: " + query);
10. }
11. }

The above code contains a SQL Injection vulnerability. If a malicious user enters “Bob’ or ‘1’=’1” then they can return every employee name from the database. They may be able to run arbitrary SQL commands against the database as well.

To fix this issue, the programmer can do one of a few things. The programmer could parameterize the statement, create a stored procedure, whitelist acceptable values, or escape the text (*SQL Injection Prevention Cheat Sheet*). The best fit for this particular code snippet is to parameterize the statement. Below is a corrected version of the snippet.

**import** java.util.Scanner;

**import** java.sql.\*;

**public** **class** IN300\_Unit5\_A1 {

**public** **static** **void** main(String[] args) {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Enter employee name");

String employee = sc.nextLine();

String query = "SELECT \* FROM Employee where employeeName = '" + employee + "'";

// Create prepared statement to prevent injection attack

PreparedStatement preqparedSql = connection.prepareStatement(query);

preqparedSql.setString(1, employee);

preqparedSql.executeQuery();

}

}

1. **Given the below Python code and execution, describe the results and what input validation countermeasures you might use to ensure you get the proper results:**
2. result = input("Do you want to continue [Yes] or [No]: ")  
     
   print(result)

The above code will allow arbitrary code to run against the terminal from which the Python file is run. Input validation should be implemented to prevent this. This could involve using type coercion to ensure the proper type is entered, or a list of acceptable values can be checked. A whitelist of values would work best here, such as:

# Get user input  
result = input("Do you want to continue [Yes] or [No]: ")  
  
# Check input against acceptable values  
if (result == "Yes" or result == "yes" or result == "No" or result == "no"):  
 print(result)  
  
# Do nothing if incorrect input

**3. In two paragraphs, describe the best practice of running code with least privilege**

Any vulnerability that can use your program to alter the operating system in some way will be running with the same permissions as your code. This can lead to issues where a malicious action can delete system files or access peripherals on a mobile device that is not needed by the program. Because of this, all code should be running with the least privilege necessary to accomplish its requirements.

A program that does not access files outside of its folder should not be granted administrative rights. A phone app that displays the weather should not request access to the camera. Ignoring this security tenant will introduce vulnerabilities to the user and possibly the backend of the application.

**4. In two paragraphs, describe the best practice of not trusting infrastructure.**

Not trusting infrastructure is also called the Zero Trust Architecture (*What is a Zero Trust Architecture*). This method of constructing code ensures that all necessary security features are built into the code. Code can run on many different types of machines with different configurations. Not knowing if a security feature is installed on one versus another can cause vulnerabilities.

To prevent this, the programmer should include all security functions within the code. All input into the computer program from the environment should also be validated by code. Adding this additional layer of security adds to the overall defense-in-depth approach.

**5. In one paragraph, describe the best practice of avoiding security by obscurity.**

Secrets have a nasty habit of being found out. Because of this, security through obscurity is a bad practice. Developers should avoid rolling their security methods to prevent introducing vulnerabilities. Instead, the developer should rely on well-tested security functions built by security professionals. These can be cryptographic functions such as AES and RSA. Also, the use of security APIs from trusted sources, such as OWASP, will help increase the security of the application.

**References:**

*SQL Injection Prevention Cheat Sheet*. OWASP Cheat Sheet Series. (n.d.). https://cheatsheetseries.owasp.org/cheatsheets/SQL\_Injection\_Prevention\_Cheat\_Sheet.html.

*What is a Zero Trust Architecture*. Palo Alto Networks. (n.d.). https://www.paloaltonetworks.com/cyberpedia/what-is-a-zero-trust-architecture.