Branden, Daniel
4/21/24
CTEC 230
Application of Data Structure

Group Project

Programming Language

Java

Small Coding Project

Simple Calculator

Part 1: Understanding Secure Programming

Writing Response

Briefly explain the importance of secure programming practices in software development. (2 points)

- When developing any system or software it's best to make sure that you follow secure practices to not only maintain the integrity of your system but to also keep the system confidential. If an individual does not follow the practices then they leave their system open to an attack that could be as simple as a virus or black hat hacker.

What are some characteristics of a secure program? (3 points)

- Some of the characteristics of a secure program are input validation, secure configuration, secure communication, patch management, authentication, and authorization.

Describe the difference between defensive programming and robust programming. (2 points)

- Defensive programming relates to more of just stopping the threat or vulnerability in its track but still leads the system open for another attack. However, a robust program can handle the threat but also prevent the threat from recurring. A defensive program may only detect the system while a robust program can prevent and stop the vulnerability from recurring.

Explain the concept of information leakage and how it can be a security risk. (3 points)

- A few concepts of information leakage are lost confidentially, exposing vulnerabilities, and regulatory compliance violations.
- Losing confidentially can lead to identity theft, financial fraud, and reputation damage.
- Exposing vulnerabilities is a security risk because the attackers can exploit this information to perform sophisticated attacks and identify the weaknesses in the system.
- Regulatory compliance violations can lead to penalties imposed by regulatory authorities and damage your organization's reputation.

Coding

```
import java.util.Scanner;


public class SecureCalculator

{


   // Input validation method to ensure the user enters valid numeric values

   private static double validateInput(String prompt, Scanner scanner)

{

      double number = 0;

      boolean isValidInput = false;
```

```
        while (!isValidInput)
    {
            System.out.print(prompt);
            if (scanner.hasNextDouble())
    {
                number = scanner.nextDouble();
                isValidInput = true;
            } else
    {
                System.out.println("Invalid input! Please enter a valid numeric value.");
                scanner.next(); // Clear the invalid input
            }
        }
        return number;
    }

    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);

        double num1 = validateInput("Enter the first number: ", scanner);
        double num2 = validateInput("Enter the second number: ", scanner);

        System.out.println("Select operation: ");
        System.out.println("1. Addition");
        System.out.println("2. Subtraction");
        System.out.println("3. Multiplication");
        System.out.println("4. Division");
        System.out.print("Enter your choice: ");

        int choice = scanner.nextInt();
        double result = 0;

        // Handling exceptions gracefully
        try {
            switch (choice)
{
                case 1:
                    result = num1 + num2;
                    break;
                case 2:
                    result = num1 - num2;
                    break;
                case 3:
                    result = num1 * num2;
                    break;
                case 4:
                    if (num2 == 0) {
```

```
                throw new ArithmeticException("Cannot divide by zero!");
            }
            result = num1 / num2;
            break;
        default:
            System.out.println("Invalid choice!");
    }
    System.out.println("Result: " + result);
  } catch (ArithmeticException e)
{
    System.out.println("Error: " + e.getMessage());
  } finally
{
    scanner.close(); // Closing scanner to prevent resource leak
  }
 }
}
```

Part 2: Secure Coding Techniques

Written Component (10 points)

- Why is static code analysis a valuable tool for secure programming? (2 points)

Static Code Analysis helps to analyze and debug the code without ever executing the code. This allows for any syntax errors or logic errors to be debugged before implementing the code.

- Briefly describe two advantages and two disadvantages of using data obfuscation techniques. (4 points)
    - Advantages of data obfuscation maintaining confidential information with encryption. And also preventing unwanted and unauthorized access to data from users by masking or reduction of sensitive information.
    - Disadvantages of data obfuscation include potential data loss through encryption and reciprocation with others. Also, potentially causes the system to malfunction due to over-processing the system and overworking the resources due to the complexity of data.
- Explain the role of secure programming paradigms (pair programming, code reviews, test-driven development) in building secure software. (4 points)
    - Pair programming: With two programmers working together, there's constant code review and discussion, which helps catch security flaws early.
    - Code Reviews: Code reviews involve peers scrutinizing each other's code for bugs, including security vulnerabilities.
    - Test-Driven Development (TDD): TDD involves writing tests before writing the actual code, which helps in identifying security requirements early in the development process.

Coding

```
import java.util.Scanner;

public class SecureCalculator
{
    // Input validation method to ensure the user enters valid numeric values
    private static double validateInput(String prompt, Scanner scanner)
  {
        double number = 0;
        boolean isValidInput = false;
        while (!isValidInput)
  {
            System.out.print(prompt);
            if (scanner.hasNextDouble())
  {
                number = scanner.nextDouble();
                isValidInput = true;
            } else
  {
                System.out.println("Invalid input! Please enter a valid numeric value.");
                scanner.next(); // Clear the invalid input
            }
        }
        return number;
    }

    public static void main(String[] args)
  {
        Scanner scanner = new Scanner(System.in);

        double num1 = validateInput("Enter the first number: ", scanner);
        double num2 = validateInput("Enter the second number: ", scanner);

        System.out.println("Select operation: ");
        System.out.println("1. Addition");
        System.out.println("2. Subtraction");
        System.out.println("3. Multiplication");
        System.out.println("4. Division");
        System.out.print("Enter your choice: ");

        int choice = scanner.nextInt();
        double result = 0;

        // Handling exceptions gracefully
        try {
            switch (choice)
    {
                case 1:
                    result = num1 + num2;
                    break;
                case 2:
                    result = num1 - num2;
                break;
            case 3:
                result = num1 * num2;
                break;
            case 4:
                if (num2 == 0)
{
                    throw new ArithmeticException("Cannot divide by zero!");
                }
                result = num1 / num2;
                break;
            default:
                System.out.println("Invalid choice!");
        }
        System.out.println("Result: " + result);
      } catch (ArithmeticException e)
{
        System.out.println("Error: " + e.getMessage());
      } finally
{
        scanner.close(); // Closing scanner to prevent resource leak
      }
   }
}
```