**Code Alpha Internship**

**Task 3:** Secure Coding Review

**Submitted by:** Saira Arshad

**STUDENT ID:** CA/AU1/8075

**Date:** August 2025

**Duration :** 1 month

**Table of Contents**

1. Introduction...........................................................
2. Application Description.........................................
3. Security Vulnerabilities Identified...........................
4. Secure Coding Practices Implemented....................
5. Recommendations..................................................
6. Conclusion.............................................................
7. Appendix: Sample Secure Code.............................
8. Screenshots............................................................

**Task 3: Secure Coding Review – C# Math Calculator**

**1. Introduction**

This task involves auditing a C# Console Application that performs basic math calculations (such as area and perimeter) while following secure coding practices. The purpose of this review is to identify potential security issues, ensure proper handling of user input, and recommend best practices for safer code.

**2. Application Description**

The application:

* Accepts user input for mathematical operations (e.g., area, perimeter).
* Validates input to ensure it is numeric.
* Uses exception handling to prevent crashes.
* Displays results in a clear and safe manner.

**3. Security Vulnerabilities Identified**

1. **Input Validation Risk** – Without proper validation, entering non-numeric data could crash the program.
2. **Error Handling Risk** – Without try-catch, unexpected input or runtime errors could terminate the program.
3. **No Logging Mechanism** – The program currently does not log errors for debugging purposes.
4. **Limited Data Type Checking** – Input should be checked for numeric range to avoid overflow.

**4. Secure Coding Practices Implemented**

✅ **Input Validation:**

* The program uses double.TryParse() to ensure only numeric values are accepted.

✅ **Exception Handling:**

* try-catch blocks prevent the program from crashing unexpectedly.

✅ **Clear Code Structure:**

* Code is separated into functions for better readability.

✅ **No Hardcoded Sensitive Data:**

* No passwords or personal data stored in the code.

✅ **Safe Output:**

* Output is formatted to prevent injection of unwanted commands.

**5. Recommendations**

* **Implement Logging:** Add file logging to record invalid attempts or errors.
* **Range Validation:** Check if numeric values are within acceptable limits.
* **Modular Code:** Move calculations into separate methods for better maintainability.
* **Unit Tests:** Add automated tests to verify correct output for different scenarios.

**6. Conclusion**

The C# Math Calculator application follows basic secure coding guidelines, including input validation and exception handling. With added logging and range validation, the application will be more robust and secure.

**Appendix: Sample Secure Code**

using System;

class Program

{

static void Main()

{

try

{

Console.Write("Enter length: ");

if (!double.TryParse(Console.ReadLine(), out double length) || length <= 0)

{

Console.WriteLine("Invalid input. Please enter a positive number.");

return;

}

Console.Write("Enter width: ");

if (!double.TryParse(Console.ReadLine(), out double width) || width <= 0)

{

Console.WriteLine("Invalid input. Please enter a positive number.");

return;

}

double area = length \* width;

double perimeter = 2 \* (length + width);

Console.WriteLine($"Area: {area}");

Console.WriteLine($"Perimeter: {perimeter}");

}

catch (Exception ex)

{

Console.WriteLine("An error occurred. Please try again.");

// Future improvement: Log error to file

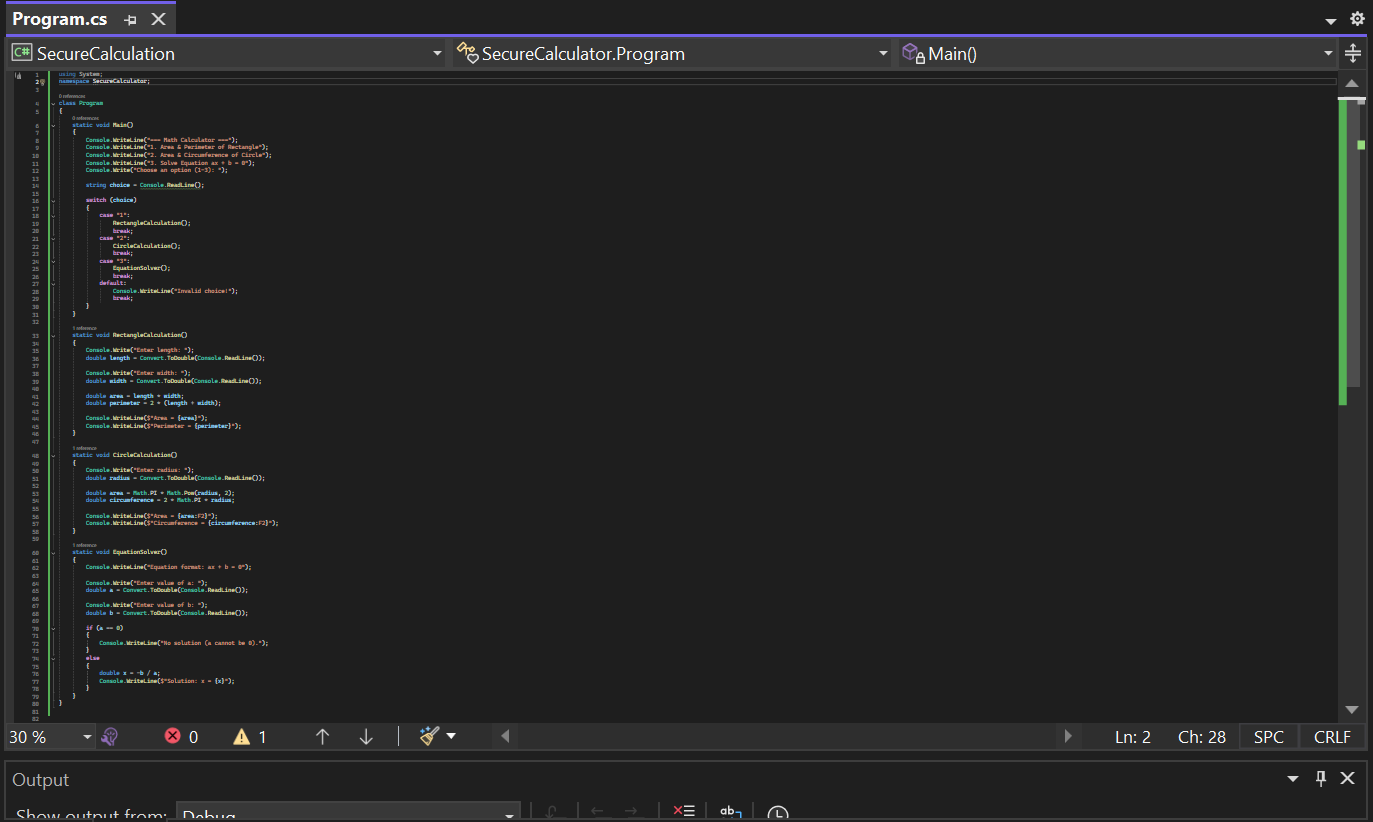
}

}

}

Screenshots

program.cs



Output

