# CS 305 Module Two Written Assignment

## Instructions

Replace the bracketed text with the relevant information in your own words. If you choose to include images or supporting materials, make certain to insert them in all the relevant locations in the document.

## Areas of Security

**Input Validation**: Relevant because user inputs are directly processed by the application in the GreetingController class.

**APIs**: The application exposes endpoints via REST APIs, which should be secure.

**Code Error**: Potential for code execution vulnerabilities and logic errors, especially in the number method.

**Code Quality**: The use of Spring Expression Language (SpEL) and general coding practices require review to ensure secure implementation.

**Encapsulation**: Ensures that data handling within classes like Greeting is properly secured.

## Areas of Security Justification

**Input Validation**: Direct evaluation of user input using SpEL in GreetingController.greeting() poses a risk of SpEL injection.

**APIs**: REST APIs must validate and sanitize inputs to prevent security vulnerabilities such as injection attacks.

**Code Error**: Potential for ArrayIndexOutOfBoundsException in GreetingController.number() due to lack of bounds checking for myArray.

**Code Quality**: Best practices like using secure libraries and avoiding direct parsing of user inputs need to be followed to prevent vulnerabilities.

**Encapsulation**: Encapsulation helps manage data flow securely and prevents unintended access or modification of class fields.

## Code Review Summary

**GreetingController.java**:

**Vulnerability**: SpEL injection risk in greeting() method by directly evaluating user input.

**Vulnerability**: Lack of bounds checking in number() method may lead to ArrayIndexOutOfBoundsException.

**Application.java**:

The direct use of SpEL to parse static expressions is not a vulnerability in itself but becomes risky when used with user input.

**Greeting.java**:

No vulnerabilities identified; class is well encapsulated with appropriate accessors.

**application.properties**:

No configurations or sensitive data present as the file is empty.

**ApplicationTests.java**:

Contains a basic context load test, no security-specific test cases.

## Mitigation Plan

1. **Input Validation**:

Sanitize and validate user inputs before processing. Avoid direct evaluation of user inputs using SpEL. Implement input validation to ensure only safe and expected inputs are accepted.

1. **APIs**:

Use validation annotations (e.g., @Valid, @NotNull, @Size) in Spring to ensure inputs meet specified criteria. Employ input sanitization libraries to prevent injection attacks.

1. **Code Error**:

Add bounds checking in the number() method to ensure that the id is within the valid range of the myArray indices. Use try-catch blocks to handle exceptions gracefully.

1. **Code Quality**:

Adopt secure coding practices, such as avoiding dynamic expressions unless necessary and always sanitizing inputs when using SpEL or similar features.

1. **Encapsulation**:

Continue following proper encapsulation practices to manage data securely. Ensure that sensitive data is protected and only accessible through well-defined interfaces.