# CS 305 Module Two Written Assignment Template

## 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

## Based on this app’s architecture and scenario, the following areas from the Vulnerability Assessment Process Flow Diagram are relevant:

## Architecture Review: Look at the overall structure, since the app exposes REST endpoints using Spring and lets users send input.

## Input Validation: The endpoints take user data, so it’s critical to check for malicious input.

## APIs: Since this is a REST API, making sure requests are handled securely is a big deal.

## Code Error: Check for logic mistakes, missing exception handling, and that libraries are used safely.

## Code Quality: Assess secure coding practices and patterns to ensure maintainability and reduce risk.

## Areas of Security Justification

* Architecture Review: Because the app is a Spring Boot REST API and uses SpEL, understanding how everything connects makes it easier to spot possible risks.
* Input Validation: if we don’t filter user input, we could end up with code injection or other attacks.
* APIs: Since outside users can call these endpoints, we need to be careful how requests are handled to keep out bad actors.
* Code Error: Not checking things like valid array indices, or not handling errors, can cause crashes or leak info about our system.
* Code Quality: Stuff like using SpEL on user input is a code quality issue and can make it way easier for someone to find a vulnerability.

## Code Review Summary

Three main concerns were identified in the code review:

1. Expression Injection: In the /greeting endpoint, whatever someone sends as the “name” gets run through a parser and evaluated. That means if someone sends something malicious instead of just a name, the app will actually try to execute it. This could let someone run their own code on the server.
2. Array Index Not Checked: The /number/{id} endpoint takes whatever number the user puts in the URL and uses it to look up an item in an array. There’s nothing stopping someone from using a number that’s out of range. If that happens, the program just crashes or gives a confusing error, which isn’t great from a user or security standpoint.
3. No Error Handling: Neither of the endpoints has any real error handling. If someone sends bad input or does something unexpected, the app throws an error and could even show details about the system. This makes it easier for an attacker to figure out what’s going on behind the scenes, and it’s also a bad user experience.

## Mitigation Plan

To mitigate these risks:

SpEL Expression Injection:

* The biggest thing is to never directly run user input through SpEL or any kind of expression parser. Instead, just treat whatever someone sends as regular text. If we absolutely need to let users send in something that gets evaluated, we should lock it down, only allowing specific, safe things using a whitelist and never just running whatever is sent.

Array Index Validation:

* For the array lookup, we should make sure the number the user sends is a valid index before trying to use it. So if the array has seven items, only numbers 0–6 should work. If someone puts in something outside that range or not a number, just return a normal error message, not a crash or a stack trace.

Add Error Handling:

* We should add proper error handling to both endpoints, so if something goes wrong, like bad input or an exception, the app returns a simple error message. No technical details should ever be shown to users. If we need the details for troubleshooting, we can log them somewhere the user can’t see.