# CS 305 Module Two Written Assignment

SNHU CS-305

Professor Norman

Sean Jetté

## Areas of Security

Utilizing the Vulnerability Assessment Process Flow Diagram, we can see that there are 7 areas of security utilized when reviewing the architecture of this web application. For this application, focusing on architecture review, input validation, APIs, code error, code quality, and encapsulation are especially critical to ensure a review of potential vulnerabilities. Starting with Architecture Review, this program is utilizing the Spring Framework which provides configuration and modeling for developing various forms of applications. As such, it is important for us to understand its structure to identify any possible vulnerabilities or misconfigurations within that framework that may impact security. Next, since we are working with user inputs that are passed into methods, such as the methods greeting and number, ensuring the use of input validation will be critical for mitigating the potential forms of injection of malicious code, such as SQL injection or SpEL injection, which could exploit these inputs and compromise the applications security. Thus, proper input validation would help to mitigate the risks of such attacks. Since this program relies on APIs to handle user requests, securing API interactions would be necessary to protect from any unauthorized access or manipulation, especially since APIs can be a significant attack vector. For code error handling, lacking checks for aspects such as array bounds checking, could introduce some potential vulnerabilities and any unhandled exceptions might expose the application to unintended consequences or behaviors that could cause the program to abruptly end or not perform the intended work. Additionally, it is important to ensure code quality, where reviewing secure coding practices and patterns is followed. Within this program, the code lacks proper validation and some secure coding practices through which the intention would be to sanitize any user inputs before processing. Lastly, the usage of methods should utilize encapsulation for protection of the data that might lead to any unwanted changes and the usage of encapsulation effectively will help restrict unauthorized access to maintain data security for the application. .

## Areas of Security Justification

While the program’s architecture is based on the Spring Framework which provides components for dependency injection and APIs, our team should pay attention to prevent any security misconfigurations. The input validation for this program is relevant due to the application’s usage on user-provided data in the name parameter’s /greeting endpoint. Since the parameter is parsed directly as an expression, it poses a risk for injection attacks if it is not properly validated. Such improper validation might lead to some form of arbitrary code execution and expose sensitive data. Thus, by ensuring all inputs are validated and sanitized, our program will be protected against potential injection threats. Next, the Spring’s REST APIs in the application (/greeting and /number/{id} endpoints) are necessary for the secure API structure and from a security perspective, APIs can be a threat vector for unauthorized access or data exposure. Next, error handling is an essential part of our security apparatus because in its current state, the code lacks mechanisms to handle certain conditions. An example of this is within the number method where it might result in an out-of-bounds exception for the array. So, implementing error handling in the form of try-catch blocks will help to mitigate any unhandled exceptions that would result in termination of the application abruptly or not performing the intended work. In regard to code quality, parsing input directly using SpEL (Expression exp = parser.parseExpression(name);) instead of formatting it could open the code up to injections and would require input sanitization to help mitigate these risks to promote code stability and security. Last of all, the Greeting class exposes data directly through public getter methods (getId and getContent). By using encapsulation, we can limit direct access to sensitive data to prevent unwanted data modifications. Additionally, reviewing encapsulation in the program would help to prevent access to any private class data.

## Code Review Summary

Through the manual inspection of the program’s source code, I identified some potential vulnerabilities and areas that should be improved upon. Starting with methods and encapsulation for the Greeting class; where this class includes public getter methods (getId and getContent) and private parameters that follow basic encapsulation practices, but the associated GreetingController class does not utilize these getter methods which could lead to inconsistencies in access. Next, with respect to the GreetingController class, user input is not sanitized or validated before processing the methods. Specifically, in the number method, id is passed directly from the user to access an array element (myArray[id]) which would lead to an array out-of-bounds error. Additionally, in the greeting method, the input string name is parsed as an expression using SpEL which is creating a potential vulnerability for any sort of injection attack. Another issue identified within the greeting method is that it does not check the length of the name input. This can lead to potential buffer overruns or performance impacts on large inputs. Within the pom.xml file, the Spring Boot and Java JDK are both outdated. Outdated versions of this framework/software could allow the introduction of vulnerabilities without the requisite security patches applied through updated software. An additional impact of the Spring Expression Language (SpEL) is in parsing user inputs that make it susceptible to expression injection. Lack of proper input handling can lead to malicious injections for manipulation purposes of the application logic or data. The last area identified through the code review the lack of exception handling specific to the number method. This method does not handle any potential ArrayIndexOutOfBoundsException and as such, if an invalid id is provided, the application throwing an exception can cause unexpected crashes, reveal information about the array structure, or not perform its intended function.

## Mitigation Plan

For the mitigation plan, we should do the following:

* Safeguard that all user inputs, such as name and id, are validated and sanitized. An example being check that id is within the array bounds and that name contains only expected characters.
* Avoid parsing the user inputs as code through using safe string formatting methods rather than directly evaluating expressions.
* Upgrade to the latest stable versions of both Spring Boot and Java JDK to protect against known vulnerabilities while also improving the program’s performance.
* Implement try-catch blocks around user-dependent operations, such as array access, to prevent unwanted crashes or performance issues. Having the program return a generic message to avoid revealing system details will help to enhance security as well.
* Regularly review the code for all secure coding applicable standards and ensure focusing on input handling, exception handling, and any dependency updates as required.