# CS 305 Module Two Written Assignment

## Overview

As the lead person on this application, I must ensure the code is secure. This review will assess potential vulnerabilities in the code and create a mitigation plan to address them with the software development team. This review will leverage the Vulnerability Assessment Process Flow Diagram (VAPFD) to ensure thoroughness in identifying vulnerabilities in the submitted code.

## Areas of Security – Reviewed in the same order as listed on the VAPFD

**Input Validation –** It is important to ensure that the input provided to the application by the end user is the intended input. Accidental or intentionally malicious input should be disallowed by the input validation method. This practice will ensure the application is secure from attack through input. The input validation should include mitigation for buffer overflow, canonicalization, cross-site scripting and SQL injection attacks.

**APIs –** It is important to ensure the interactions and communications between systems are secure from attack and the data is encrypted if the data is sensitive. This application uses the Spring framework for interactions. Spring secure offers servlet and Web MVC integrations to ensure APIs communicate securely. Leveraging Web MVC should mitigate risk from attack by denial of service, distributed denial of service, man in the middle, injection, cross-site scripting and credential stuffing attacks.

**Cryptography –** As this application will have a public-facing interface, it is important to ensure that user credentials and sensitive user information submitted are encrypted. It is possible that data shared from server to client is also sensitive and should be encrypted to mitigate the data being intercepted. Proper encryption techniques should mitigate risk from attacks by brute force, plain text, cipher text, key and algorithm attacks.

**Client/Server –** As this application relies on a client/server relationship, the previous security areas are of increased importance as we know there is a high likelihood of public infrastructure between the client and server. Data sent over public infrastructure is susceptible to interception and injection-based attack methods. It is essential to mitigate these threats to keep user confidence high.

**Code Error –** In this application, code errors can be introduced by input or by API. Additional diligence and testing will be required to extend the traditional input testing to include extremely large values and typical injection attack strategies. Some out-of-the-box thinking will be required here as the vulnerability is more about unexpected values than expected ones.

**Code Quality –** High-quality code and adherence to industry best practices and our corporate style guide should be a standard operating procedure. Adhering to this will lessen the likelihood of vulnerabilities introduced by code quality issues. Pay special attention to the Spring framework and Spring security guidelines and syntax as using this framework is new and could introduce quality issues without extra diligence on this project.

**Encapsulation –** Ensuring that only the application and framework can access the objects is essential for good security during code development. This practice prevents the overall system security not from being undermined by improper object permissions.

## Areas of Security Justification

All seven areas of security are important to this project. Much of the justification and attack methodologies are outlined in the descriptions above. Based on the project requirements, the developers must focus on input validation, APIs, client/server and code quality. This does not de-emphasize the importance of the other areas but places some trust in the Spring framework instead.

## Code Review Summary

**Missing Input Functionality –** The current code contains a placeholder for input not representative of the expected values. The greeting message parser and variables appear to be implemented properly, but no input makes input validation testing impossible. It should also be noted that this data will be stored as plain text without code changes. As we don’t know the sensitivity of the data, this may introduce a vulnerability.

**Improper Access Declared –** After reviewing the constructor for the variables in Greeting.java, the “public“ declaration could be problematic and make the object vulnerable to attack.

**Spring Framework Version –** This was not part of the code review, but it was mentioned in the directions that the team is using Spring framework 2.6.5. This version is out of date and could suffer from vulnerabilities. This did not show up in the code review because I updated the code during the week one Maven update and see version 5.2.3, a current release.

## Mitigation Plan

The Spring framework for the project needs to be updated. There is a directory traversal CVE identified for 2.6.5 and it should no longer be used for new development. An entire framework update should mitigate the risk of incompatible components and versions.

Change the constructor within Greeting.java to private. This will hide the variables from external sources and improve the application’s security

Correct the application input by replacing the placeholder with the appropriate code. Ensure the code used contains input validation strategies, encapsulation strategies and encryption if necessary.