# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **May 23rd, 2025** | **Jordan Walker** | **Wrote up the first version of the report after going through the code, running tests, and checking what the client needs.** |

## Client



## Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In this report, identify your security vulnerability findings and recommend the next steps to remedy the issues you have found.

* Respond to the five steps outlined below and include your findings.
* Respond using your own words. You may also include images or supporting materials. If you include them, make certain to insert them in the relevant locations in the document.
* Refer to the Project One Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Jordan Walker

**1. Interpreting Client Needs**

Determine your client’s needs and potential threats and attacks associated with the company’s application and software security requirements. Consider the following questions regarding how companies protect against external threats based on the scenario information:

* What is the value of secure communications to the company?
* Are there any international transactions that the company produces?
* Are there governmental restrictions on secure communications to consider?
* What external threats might be present now and in the immediate future?
* What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?

[Artemis Financial builds software that handles sensitive financial and personal data, so keeping user input secure, encrypting communications, and avoiding anything that could lead to code execution are key priorities. Even though there is no clear sign of international transactions, strong security is still important to meet compliance standards like SOX or GDPR.

The software needs protection from common web threats like injections, remote code execution, and weak authentication. Since the app is built using Spring, which is popular but sometimes has vulnerabilities, it is important to keep everything updated. The open source libraries in use also need regular security checks and updates to keep the system safe and reliable.]

**2. Areas of Security**

Refer to the vulnerability assessment process flow diagram. Identify which areas of security apply to Artemis Financial’s software application. Justify your reasoning for why each area is relevant to the software application.

[Looking at the Vulnerability Assessment Process Flow Diagram, there are a few key areas that definitely apply to the Artemis Financial app. Input validation is a big one since the app takes raw user input and runs it through Spring Expression Language. Without proper checks, that kind of setup can lead to serious problems like remote code execution. It also uses path variables from the URL to access array positions, and there are no safeguards to make sure those values are safe, which could cause the app to crash.

An architecture review is also a good idea because the app uses a REST API with user-driven routes and dynamic expression parsing, which adds more risk if things are not tightly controlled. A lot of the code could use stronger coding practices too. There is not much input validation, and the error handling mostly just uses print statements. Cryptography might not come up much in this project, but it is still important for anything dealing with financial or personal information. The app should also handle API calls securely, especially if it connects with third-party services, by using proper authentication and secure headers. On top of all that, the way it handles errors needs improvement. Showing stack traces and printing errors to the console just gives attackers more information to work with.]

**3. Manual Review**

Continue working through the vulnerability assessment process flow diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

[- SpEL input execution: The greeting method uses Spring’s expression parser directly on raw user input without safety filters.

- Unvalidated path variables: The /number/{id} endpoint takes the path variable and uses it directly as an array index.

- No input sanitization: Inputs are not filtered or validated.

- No error handling: There are no try-catch blocks or recovery logic.

- Verbose output: Stack traces are shown on error.

- Print statements: Logging uses println instead of a secure, structured logging library.

- No CSRF protection: Forms and endpoints do not appear to use CSRF tokens.

- No authentication or role-based access: Sensitive endpoints are not protected.]

**4. Static Testing**

Run a dependency check on Artemis Financial’s software application to identify all security vulnerabilities in the code. Record the output from the dependency-check report. Include the following items:

* The names or vulnerability codes of the known vulnerabilities
* A brief description and recommended solutions provided by the dependency-check report
* Any attribution that documents how this vulnerability has been identified or documented previously

[Vulnerabilities found:

* log4j-api-2.12.1: CVE-2021-44228 (Log4Shell) allows remote code execution.
* jackson-databind-2.10.2: CVE-2020-8840 allows remote code execution via unsafe deserialization.
* hibernate-validator-6.0.18.Final: CVE-2020-10693, bypasses input validation logic.
* commons-collections4-4.0: CVE-2015-7501, remote code execution from unsafe deserialization.
* spring-expression-4.3.25.RELEASE: CVE-2018-1275, allows dangerous input parsing.
* snakeyaml-1.15: CVE-2017-18640, denial of service through recursive structures.
* mongo-java-driver-2.4: CVE-2016-6494, weak authentication behavior.

Recommendations:

* Update log4j to 2.17.1 or later.
* Update jackson-databind to at least 2.13.
* Patch hibernate-validator and spring-expression to latest secure versions.
* Remove or replace older libraries like commons-collections4 and snakeyaml.
* Evaluate if mongo-java-driver is required, and upgrade or remove it.
* Continue running dependency scans as part of the CI process.]

**5. Mitigation Plan**

Interpret the results from the manual review and static testing report. Then identify the steps to mitigate the identified security vulnerabilities for Artemis Financial’s software application.

[To cut down on security risks, the app should stop using Spring Expression Language to evaluate raw user input. If it really has to use it, there should be strict limits on what users can actually input. Every piece of user input should be checked and cleaned up before it gets used, especially things like path variables that are used in array access. Without those checks, it’s way too easy for something to go wrong or be exploited.

The app should also have CSRF protection on all its forms and endpoints to help block fake requests. Logging should be handled by a proper system like SLF4J instead of just using print statements, which are not secure or helpful long-term. Any libraries or tools with known vulnerabilities need to be updated as soon as possible. It’s also a good idea to include secure headers in the response to help protect against common web attacks.

Errors shown to users should be simple and vague, while the actual details are logged safely in the background. The app should always use HTTPS to keep data safe during transmission. And finally, regular security scans should be part of the workflow to catch any new issues before they become serious problems.]