# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/16** | **Joey Quinata** | **Initial doc created, including manual code review findings, dependency-check results, and preliminary mitigation plan** |

## 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

Joey Quinata

**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 wants to modernize its web application while making sure customer information stays safe. Since they deal with financial planning like savings, retirement, and investments, secure communication is very important. A breach could put private financial data at risk. Because Artemis may have international clients, they also need to think about rules around data privacy and sending data across borders. On top of that, their modernization plan will likely include more open-source libraries and newer web technologies, which can sometimes introduce new risks if not carefully managed (OWASP, n.d.). External threats they should prepare for include SQL injection, cross-site scripting (XSS), and attacks against their APIs (Detlefsen & Manico, 2015).

**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, a few areas of security stand out as most important for Artemis. Input validation is a must because improper validation is a common cause of injection attacks (Iron-Clad Java, Ch. 7). Secure error handling is another, since detailed error messages can give away too much information to attackers. Cryptography is also critical here because Artemis works with financial data that needs to be protected both when stored and when sent across networks (Oracle, n.d.). Finally, secure API interactions are especially important since their application uses REST APIs that could be targeted if not properly protected.

**3. Manual Review**

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

When manually reviewing the Artemis Financial rest-service code base, several potential issues stood out across the seven security areas from the Vulnerability Assessment Process Flow Diagram. First, input validation is weak in a few places, as some controller methods accept raw user input without proper sanitization, which could expose the application to injection risks. In addition, error handling is minimal, with stack traces and system details likely to be exposed in certain exception flows. Logging also presents a concern, since sensitive details could unintentionally be written to logs without redaction. Authentication and session handling appear to rely heavily on defaults, which raises concerns about session fixation or weak enforcement of access rules. In terms of configuration management, some hardcoded values were found in the service classes, which makes them more difficult to secure or update. The use of older dependencies in the pom.xml also raises red flags for dependency vulnerabilities, even before running the static test. Finally, encryption practices need to be strengthened, as secure communication (such as TLS enforcement) was not clearly evident in the base code. Together, these findings highlight at least seven clear areas of vulnerability: input validation, error handling, logging, authentication/session management, configuration, dependency management, and encryption.

**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

The dependency-check report confirmed many of the concerns found in the manual review. Several outdated libraries were flagged with known CVEs, including Spring Framework 5.2.3.RELEASE (CVE-2020-5421, CVE-2021-22096, CVE-2021-22118), Spring Boot 2.2.4.RELEASE (CVE-2022-27772, CVE-2023-20873, CVE-2023-20883), and Tomcat Embed 9.0.30 (CVE-2019-17569, CVE-2020-13934, CVE-2021-41079, and others). Serialization libraries such as Jackson 2.10.2 were also vulnerable, with deserialization flaws like CVE-2020-25649 and CVE-2022-42003, while Logback 1.2.3 introduced logging vulnerabilities (CVE-2021-42550, CVE-2023-6378). Additional findings included SnakeYAML 1.25 with multiple parsing vulnerabilities (CVE-2017-18640, CVE-2022-1471, CVE-2022-38751), Hibernate Validator 6.0.18.Final (CVE-2020-10693, CVE-2025-35036), and the cryptography library Bouncy Castle 1.46 with older cryptographic weaknesses (CVE-2013-0169, CVE-2015-6644) The recommended solutions were consistent across dependencies: upgrade to patched, supported versions and re-run the scan to confirm remediation.

**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.

The combination of the manual review and the static analysis highlights the need for a layered mitigation plan. First, input validation and error handling should be improved by implementing validation frameworks and ensuring error messages do not reveal system details. Sensitive logging must be redacted, and authentication/session controls should be hardened, such as enabling secure cookies and enforcing stricter role-based access. For configuration issues, hardcoded values should be externalized into secure configuration files or environment variables. Most critically, the outdated dependencies must be upgraded, Spring and Tomcat should be aligned with a current Spring Boot release, Jackson updated to a safe 2.x line, Logback moved to a patched version, and SnakeYAML, Hibernate Validator, and Bouncy Castle upgraded to supported releases. By applying these changes, the application will address both design-level vulnerabilities identified in the manual review and the library-level vulnerabilities revealed by the dependency-check report.

**References**

OWASP. (n.d.). OWASP Secure Coding Practices Quick Reference Guide. https://owasp.org/www- project-secure-coding-practices-quick-reference-guide/

OWASP Dependency-Check Maven. (n.d.). Dependency-Check Maven Plug-in. https://jeremylong.github.io/DependencyCheck/dependency-check-maven/index.html

Oracle. (n.d.). Secure Coding Guidelines for Java SE. https://www.oracle.com/java/technologies/javase/seccodeguide.html

Detlefsen, A., & Manico, J. (2015). Iron-Clad Java: Building Secure Web Applications. McGraw-Hill Education. (Chs. 1, 3, 7, 10)