# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/09/2024** | **Jorge Flores** | **Initial vulnerability assessment report.** |

## 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

Jorge Flores

**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?

1. Financial relies on secure communication to protect sensitive customer financial data such as retirement plans, insurance details, and investments. Breaches in communication security could result in reputational damage, loss of customers, and potential legal liabilities.

2. Yes, Artemis Financial's client base spans various regions, and international transactions are part of its operations. Secure encryption protocols must be used to prevent man-in-the-middle attacks during such communications.

3. Yes, compliance with regulations such as GDPR for international clients and PCI DSS for financial transactions is essential. Adhering to these standards ensures legal compliance and secure handling of data.

4. Current threats include phishing attacks, SQL injection, cross-site scripting (XSS), and vulnerabilities in outdated libraries. As the company grows, targeted attacks from advanced persistent threats (APTs) may also become a concern.

5. - Open-source libraries: Dependencies must be updated regularly to eliminate vulnerabilities.

- Evolving web application technologies: Implementing secure API interactions and modern authentication methods like OAuth or SAML will strengthen application security.

**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.

Using the Vulnerability Assessment Process Flow, the following areas of security are relevant to Artemis Financial’s web application:

1. Input Validation: Ensures only valid and sanitized data is accepted by the application, reducing the risk of SQL injection and XSS attacks.
2. APIs: Secures API interactions to prevent unauthorized access or data leaks.
3. Cryptography: Encrypts sensitive data to prevent unauthorized access or tampering.
4. Code Quality: Implements secure coding practices to avoid common vulnerabilities.

**3. Manual Review**

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

During the manual review of the provided codebase, the following vulnerabilities were identified:

1. Lack of Input Validation in CRUDController.java: User inputs are directly passed without validation, increasing the risk of SQL injection.
2. Improper Error Handling in CRUD.java: Sensitive error details are exposed, which could aid attackers.
3. Outdated Dependencies in POM.xml: Several library versions have known vulnerabilities (e.g., Spring Boot version).
4. Hardcoded Values in customer.java: Storing sensitive information as plaintext in the code.
5. Unsecured API Endpoints in GreetingController.java: No authentication mechanism is implemented.
6. Redundant Code in DocData.java: Increases complexity and may introduce logic errors.
7. Missing Logging in CRUDController.java: Fails to log critical activities, which affects auditing and monitoring.

**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

From the Dependency Check reports, the following vulnerabilities were identified:

CVE-2020-9484: Vulnerable Apache Tomcat versions allowing remote code execution.

//Solution - Update to the latest Apache Tomcat version.

CVE-2021-22118 - Spring Boot versions below 2.5.0 have a known deserialization vulnerability.

//Solution - Upgrade Spring Boot to version 2.5.0 or higher.

CVE-2020-5398: Jackson Databind vulnerability allowing potential DOS attacks.

//Solution - Upgrade Jackson Databind to a secure version.

**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 following steps will address the identified vulnerabilities:

1. Input Validation: Implement a robust validation mechanism in all user input fields using frameworks like Hibernate Validator.
2. Update Dependencies: Upgrade all outdated libraries to their latest secure versions, ensuring compatibility.
3. Error Handling: Use generic error messages for end users and log detailed errors securely.
4. Secure API Endpoints: Introduce authentication mechanisms, such as OAuth2, for all API interactions.
5. Encrypt Sensitive Data: Replace hardcoded sensitive information with encrypted or environment-based configurations.
6. Code Optimization: Refactor redundant code to improve maintainability and reduce complexity.
7. Enable Logging: Add logging for all critical operations to support monitoring and incident response.