# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **1/24/25** | **Drina Baptiste** |  |

## 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

Drina Baptiste

**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?

Secure communications are vital to Artemis Financial to protect sensitive customer data, ensure regulatory compliance, and maintain trust. As the company may handle international transactions, compliance with regulations like GDPR and PCI-DSS is crucial. External threats such as phishing, SQL injection, and zero-day exploits pose immediate risks to the organization’s operations and customer data. Modernization efforts should focus on securing open-source libraries through dependency scanning and adopting robust API security measures like OAuth2 and input validation. Additionally, leveraging evolving web application technologies, such as serverless architectures, requires secure coding practices to mitigate vulnerabilities. Two-Factor authentication should be implemented as well to combat shoulder surfing. These measures will ensure Artemis Financial stays resilient against current and future security challenges.

**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.

Several areas of security are crucial for Artemis Financials’ software application based on the vulnerability assessment process. **Secure coding** is vital to ensure structured code that prevents external interference and protects sensitive company and client data. Proper **error handling** is also necessary to manage potential issues securely without exposing vulnerabilities. Since Artemis Financial uses a RESTful API, **API security** is a key focus, requiring secure communication protocols to prevent unauthorized access and data breaches. Additionally, **input validation** is critical to sanitize and verify user inputs, protecting the system from injection attacks and other threats. These measures collectively strengthen the software application against external threats and ensure the confidentiality, integrity, and availability of sensitive financial data.

**3. Manual Review**

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

The vulnerabilities that I see in the code base is the following:

* There is no authentication system in the code base
* In the DocData.java, there needs to be improvement on secure coding. The method uses MySQL JDBC driver without specifying secure configurations like SSL encryption. This could expose the data to interception in transit.
* DocData- There is also potential for a sql injection in the method read\_document, in which it accepts user-provided inputs but there’s no query implementation method.
* In the pom.xl file, the Bouncy Castle Library is outdated and has a known vulnerability for the current version that is in the file, which allows remote attackers to execute arbitrary code or bypass and encryption.
* The java version is also hard coded and outdated
* In the customer class, The account\_balance field is package-private , which allows access from other classes in the same package, potentially leading to unauthorized modification.
* HTTPS is not utilized which helps with securing information

**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

1. Hibernate Validator (6.0.18.Final.jar)

* **Vulnerability IDs**: cpe:2.3:a:redhat:hibernate\_validator:6.0.18:\*:\*:\*:\*:\*:\*:\*
* **Description**: Potential vulnerabilities in input validation may allow attackers to bypass validation checks or exploit other features in the application.
* **Recommendation**: Update to a newer, secure version of Hibernate Validator (e.g., 6.2.x or later).

2. Jackson Databind (2.10.2.jar)

* **Vulnerability IDs**: cpe:2.3:a:fasterxml:jackson-databind:2.10.2:\*:\*:\*:\*:\*:\*:\*
* **Description**: Multiple deserialization vulnerabilities could allow attackers to execute arbitrary code by injecting malicious input.
* **Recommendation**: Upgrade to the latest Jackson Databind version (e.g., 2.15.x).

3. Spring Boot (2.2.4.RELEASE.jar)

* **Vulnerability IDs**: cpe:2.3:a:vmware:spring\_boot:2.2.4:\*:\*:\*:\*:\*:\*:\*
* **Description**: Vulnerabilities in the Spring framework might allow remote code execution (RCE) or privilege escalation attacks.
* **Recommendation**: Update to a supported and patched version of Spring Boot (e.g., 2.5.x or later).

4. Snakeyaml (1.25.jar)

* **Vulnerability IDs**: cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25:\*:\*:\*:\*:\*:\*:\*
* **Description**: Vulnerabilities in YAML parsing could allow attackers to execute arbitrary code or cause denial of service (DoS).
* **Recommendation**: Upgrade to a secure version of Snakeyaml (e.g., 2.x or later).

5. Log4j API (2.12.1.jar)

* **Vulnerability IDs**: cpe:2.3:a:apache:logging:log4j:2.12.1:\*:\*:\*:\*:\*:\*:\*
* **Description**: Certain versions of Log4j may have issues that can expose applications to vulnerabilities like information leakage.
* **Recommendation**: Upgrade to the latest version of Log4j (e.g., 2.17.x).

6. Tomcat Embed (9.0.30.jar)

* **Vulnerability IDs**: cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*
* **Description**: Vulnerabilities in Tomcat, such as remote code execution and bypassing authentication, could allow attackers to compromise the application server.
* **Recommendation**: Update to a more secure Tomcat version (e.g., 9.0.71 or later).

7. JSON Path (2.4.0.jar)

* **Vulnerability IDs**: cpe:2.3:a:json-path:jayway\_jsonpath:2.4.0:\*:\*:\*:\*:\*:\*:\*
* **Description**: Vulnerabilities may exist in JSON parsing, leading to potential injection or denial of service (DoS) attacks.
* **Recommendation**: Update JSON Path to the latest secure version.

8. JSON Smart (2.3.jar)

* **Vulnerability IDs**: cpe:2.3:a:json-smart\_project:json-smart:2.3:\*:\*:\*:\*:\*:\*:\*
* **Description**: Issues in JSON Smart could expose the application to injection or deserialization vulnerabilities.
* **Recommendation**: Upgrade to a secure version of JSON Smart.

**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.

After reviewing the codebase and the static testing report. There are a few steps that we need to do to mitigate the known vulnerabilities. We need to address the outdated libraries, risks with user inputs, and the chance of exposing sensitive information. To fix these, the first step is to update the old libraries, like Hibernate Validator and Jackson Databind, to their newest versions and check for updates regularly. It’s also important to validate any data users enter to make sure it’s safe and to avoid risks like injection attacks. Setting up secure communication, like using HTTPS, will help keep data safe while it’s being sent. The logging system should also be updated to avoid storing sensitive information that could get leaked. Finally, testing the software regularly for security issues and following good practices during development will help keep everything secure.