# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/18/25** | **Nikki Yanez** |  |

## 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

Nikki Yanez

**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 protect sensitive financial data, maintain customer trust, and ensure the company remains compliant with strict financial regulations. The company must comply with relevant global data privacy and security laws like GDPR. Yes, depending on the countries where Artemis operates, there may be governmental restrictions or requirements on the type and strength of encryption used for financial data. The company must also comply with national regulations like the GLBA. Artemis Financial faces threats like injection attacks cross-site scripting, malicious inputs, and API vulnerabilities. They are also vulnerable to zero-day exploits in the open-source frameworks they use. The company must have a continuous process for updating open-source libraries to patch security flaws and must implement modern security protocols for its APIs and cloud infrastructure to prevent data breaches as technologies evolve.

**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.

Input Validation

The application must check all incoming data from users to prevent malicious code or commands from getting in. This stops attacks like SQL injection and cross-site scripting.

Data Security

Since the application deals with sensitive financial information, it must protect data while it's being used and stored. This means making sure data isn't exposed or corrupted.

Client / Server

The application's overall design, including its use of frameworks like Spring Boot, needs to be built with security in mind. A solid foundation prevents major design flaws that hackers could exploit.

Code Error

When something goes wrong, the application's error messages should be generic. They shouldn't reveal sensitive details about the system that could help an attacker.

Code Quality

Developers must follow secure coding rules to avoid mistakes that create vulnerabilities, such as weak access controls or improper exception handling.

Encapsulation

The application's internal data structures must securely contain sensitive information, so private data isn't accidentally exposed to other parts of the system.

Cryptography

All financial data, whether it's being sent over the internet or sitting in a database, needs to be encrypted using strong methods. This protects data in transit and at rest.

APIs

If the application communicates with other systems (like banks or payment processors), those connections must be secure. This prevents unauthorized access and data leaks through external links.

**3. Manual Review**

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

By manually inspecting the code and considering the dependencies identified earlier, I found the following vulnerabilities in Artemis Financial’s code base:

Use of outdated Spring Framework (5.2.3) with known exploited vulnerabilities.

Use of outdated Tomcat (9.0.30), which contains multiple critical CVEs.

SnakeYAML (1.25) vulnerable to deserialization flaws that can lead to remote code execution.

Jackson Databind (2.10.2) susceptible to deserialization attacks if misconfigured.

Logback (1.2.3) with known issues that can be exploited for denial-of-service attacks.

Log4j (2.12.1) vulnerable to information disclosure and potential remote execution.

Bouncy Castle (1.46) outdated cryptographic library with weaknesses in encryption handling.

Improper error handling in code that may expose stack traces or sensitive system details to users.

These findings show that the application relies on multiple outdated dependencies and has several risks related to secure coding and architecture, all of which should be addressed in the mitigation plan.

**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

|  |  |  |
| --- | --- | --- |
| bcprov-jdk15on-1.46.jar | The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.5 to JDK 1.7. | CVE-2024-34447 |
| hibernate-validator-6.0.18.Final.jar | Hibernate's Bean Validation (JSR-380) reference implementation. | CVE-2025-35036 |
| jackson-core-2.10.2.jar | Core Jackson processing abstractions (aka Streaming API), implementation for JSON | CVE-2025-52999 |
| jackson-databind-2.10.2.jar | General data-binding functionality for Jackson: works on core streaming API | CVE-2020-25649 |
| log4j-api-2.12.1.jar | The Apache Log4j API | CVE-2020-9488 |
| logback-classic-1.2.3.jar | logback-classic module | CVE-2023-6378 |
| logback-core-1.2.3.jar | logback-core module | CVE-2023-6378 |
| snakeyaml-1.25.jar | YAML 1.1 parser and emitter for Java | CVE-2022-1471 |
| spring-aop-5.2.3.RELEASE.jar | Spring AOP | CVE-2022-22965 |
| spring-boot-2.2.4.RELEASE.jar | Spring Boot | CVE-2023-20873 |
| spring-boot-starter-web-2.2.4.RELEASE.jar | Starter for building web, including RESTful, applications using Spring  MVC. Uses Tomcat as the default embedded container | CVE-2023-20873 |
| spring-context-5.2.3.RELEASE.jar | Spring Context | CVE-2022-22965 |
| spring-core-5.2.3.RELEASE.jar | Spring Core | CVE-2022-22965 |
| spring-expression-5.2.3.RELEASE.jar | Spring Expression Language (SpEL) | CVE-2022-22965 |
| spring-web-5.2.3.RELEASE.jar | Spring Web | CVE-2016-1000027 |
| spring-webmvc-5.2.3.RELEASE.jar | Spring Web MVC | CVE-2022-22965 |
| tomcat-embed-core-9.0.30.jar | Core Tomcat implementation | CVE-2020-1938 |
| tomcat-embed-websocket-9.0.30.jar | Core Tomcat implementation | CVE-2020-1938 |

**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 review found outdated and vulnerable libraries in Artemis Financial’s application, some with known exploits. These must be fixed quickly to protect financial data.

The first step is updating critical components like Spring, Tomcat, and SnakeYAML to their latest secure versions. Other high-risk libraries, such as Jackson Databind and Logback, should also be upgraded right away. After that, all remaining dependencies, including bcprov, Hibernate Validator, Jackson Core, Log4j, and Spring Boot, should be patched to supported versions.

Next, the system should be hardened with secure configurations. This includes using tools like OWASP Dependency-Check, Snyk, or Dependabot to scan for risks, enforcing strong security settings in Spring, and tightening Tomcat by disabling unused features and requiring HTTPS.

Finally, Artemis should set up regular monitoring and testing, track new CVEs, and update quickly when patches are released. A software bill of materials (SBOM) should be maintained, and developers should be trained on secure coding and dependency management.

The most urgent task is upgrading the critical libraries, followed by ongoing updates and security practices to reduce future risks.