# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **5/2025** | **Gigi Cruz** |  |

## 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

Gigi Cruz

**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?

Given that Artemis Financial handles sensitive personal data like personal identity information, financial records, insurance police, etc., so secure communications will be essential. Artemis Financial will need a means of preventing unauthorized access and ensuring data can’t be altered. This will allow them to build trust with their clients and to comply with government regulations.

It’s unclear whether the company engages in international transactions, but planning for scalability would be useful. With international transactions, extra precautions may need to be taken to protect data communications. The company may also consider a way to support secure cross-border communication protocols. Of course, awareness of data sovereignty laws like the GDPR in Europe or PIPEDA in Canada will be beneficial.

As mentioned, Artemis Financial will need to comply with governmental restrictions in relation to secure communications. For example, the Gramm-Leach-bliley Act(GLBA), requires safeguarding of customer financial information (ftc.gov). The company must regard all regulations necessary as non-compliance can result in legal penalties, lawsuits and damage to their reputation.

There are many external threats that Artemis Financial should be aware of and prepare for:

* Phishing attacks: This can target employees and clients to gain access to the systems.
* API attacks: Improperly secured endpoints, like a broken object-level authorization, can expose data.
* Ransomware: encrypted internal data can be held hostage
* Credential stuffing: This can occur if users reuse passwords or authentication lacks multi-factor verification.
* Man-in-the-middle attacks: This can occur if communications are not properly encrypted.

There are a few more threats, but mitigating these threats requires constant monitoring, patching, penetration testing, and employee education.

To modernize effectively and securely, the company should consider that with open-source tools like open-source libraries, you need to make sure things are regularly updated, that the tool is vetted for security vulnerabilities using tools like OASP Dependency Check, and ensure that they are ed according to best practices. With evolving web applications, it's important to ensure RESTful API’s are protected with input validation, rate limiting, logging and anomaly detection, and HTTPS-only policies.

**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.

1. Architecture Review

Security Area: System-wide security

This is relevant to Artemis Financial because the application architecture determines how components interact, including client-facing portals, API’S, and third-party services. As a financial institution, misconfigured architecture can expose data pathways or create trust boundary violations.

1. Input Validation

Security Area: Data integrity and injection prevention

This is relevant to the company because financial data entry points(customer information, transaction details) are high-value targets for injection attacks.

Proper input validation prevents malformed or malicious inputs that could lead to data corruption or system compromise.

1. API’s

Security Area: Access control, data leakage prevention

This is highly relevant because Artemis relies on a RESTful API to exchange financial data, which must be protected with strong authentication, rate limiting, and access controls.

1. Cryptography

Security Area: Data confidentiality and compliance

This is essential because encryption (TSL for data transit, AES for data at rest) is necessary to secure financial and personal data. Cryptography also ensures compliance with regulations.

1. Code Error:

Security Area: Error handling and exception management

This is important for the company because unhandled exceptions could expose stack traces or crash systems, leaving sensitive implementation details. If crashes or errors do occur, they will want to ensure data isn’t exposed in the process.

1. Code Quality :

Security Area: consistent, safe codebase

It's important to use a consistent, secure design pattern to minimize the attack surface.

**3. Manual Review**

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

Crud.java

1. Lacks input validation; accepts unsanitized data via constructor, could be risk depending on how it's used elsewhere

CRUDController.java

1. Unvalidated input; may allow injection depending on DocData usage.
2. Exposes internal data (doc.toString()); could leak sensitive or unintended information.
3. No access control, unprotected public endpoint

Customer.java

1. No Input validation in deposit(); allows negative deposits, risks logic abuse.
2. Exposes account number(showInfo); could expose sensitive data without authentication.
3. Poor encapsulation, account\_balance is not private; risk of direct manipulation.

DocData.java

1. Hardcoded database credentials; full database credentials in source code
2. Uses root database user; root has full privileges, which increases the impact if compromised.
3. Incomplete method; missing prepared statements opens the door to SQL injection.

RestServiceApplication.java

1. No Spring Security enabled; the application exposes all endpoints by default.

**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.(CVE-2015-6644, CVE-2018-1000180)**

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.

The vulnerability is due to a weakness in cryptographic implementation

Recommended to upgrade to 1.68+

* **jackson-databind-2.10.2.jar(CVE-2020-8840, CVE-2020-9547)**

General data-binding functionality for Jackson: works on the core streaming API

The vulnerability here is in data integrity. Entity expansion isn’t properly secured, so the XML external entity is vulnerable to attacks. Recommended to upgrade to 2.13.4+

* **log4j-api-2.12.1.jar (CVE-2020-9488)**

This has an SMTPS hostname verification issue. Leaves it open to middle-man attack, which could leak data. Recommended upgrade to 2.17.2+

* **logback-core/logback-classic-1.2.3.jar(CVE-2021-42550)**

A serialization vulnerability in the logback receiver part of logback version 1.4.11 allows an attacker to mount a Denial-of-Service attack by sending poisoned data. Essentially arbitrary file write via crafted config. Recommended to update to 1.2.11+

* **snakeyami-1.25.jar(CVE-2022-1471)**

Constructor class doesn’t restrict the types that can be instantiated during deserialization, which can lead to remote code execution. Recommend to upgrade to 2.0+

* **spring-boot/spring-core/spring-web(5.2.3/2.2.4)(CVE-2022-22965 *(Spring4Shell)*, CVE-2020-5398)**

RCE via unsafe property binding; classLoader exposure. Recommended upgrade to Spring Boot 2.6.7+ and Spring Framework 5.3.18+

* **tomcat-embed-core-9.0.30.jar ( CVE-2020-13943, CVE-2020-1938)**

HTTP vulnerabilities, Denial of Service attack, path traversal. Recommended upgrade to Tomcat 9.0.60+

**Attributes**

CVE’s were: Referenced from the [National Vulnerability Database (NVD)](https://nvd.nist.gov)

Cross-linked via CPE strings and Maven coordinates

Detected by OWASP Dependency-Check

Publicly documented in CVE records or vendor security advisories

**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.

| **Vulnerability** | **Recommended Fix** |
| --- | --- |
| **Unvalidated user input** | Add input validation for all @RequestParams using Spring’s @Validated annotation and validation annotations like @Size, @Pattern, etc. |
| **Exposed internal data via toString()** | Replace doc.toString() with sanitized, structured response objects (DTOs). |
| **No access control / authentication** | Integrate Spring Security. Secure all endpoints with roles, authentication, and CSRF protection. |
| **SQL connection without prepared statement** | Implement PreparedStatement with parameterized queries to prevent SQL Injection. |
| **Hardcoded database credentials** | Move credentials to a secure application.properties file or environment variables. Use Spring Boot's configuration abstraction. |
| **No logging or improper exception handling** | Use a logging framework (e.g., SLF4J + Logback). Never expose stack traces to the client. |
| **Insecure data exposure (account info)** | Add authorization checks before returning any customer/account data. |
| **Poor encapsulation of fields** | Mark fields as private and provide controlled accessors with validation. |
| **Spring4Shell (CVE-2022-22965), others** | Upgrade to Spring Boot 2.6.7+ and Spring Framework 5.3.18+ |
| **CVEs: 32–33 issues** | Upgrade to Tomcat 9.0.65+ |
| **RCE vulnerability (CVE-2022-1471)** | Upgrade to SnakeYAML 1.33+ |
| **Unsafe deserialization (CVE-2020-8840)** | Upgrade to 2.13.4+ |
| **File overwrite (CVE-2021-42550)** | Upgrade to 1.2.11+ |
| **Constraint validation flaws** | Upgrade to 6.1.7+ |
| **Cryptographic weaknesses (CVE-2015-6644, etc.)** | Upgrade to BouncyCastle 1.68+ |