# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **1/26/25** | **Tiba Alramadan** |  |

## 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

Tiba Alramadan

**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?

**Secure communications are vital for protecting client financial data, ensuring trust, and complying with regulatory standards. Compromised communication could lead to financial loss and reputational damage**.

* Are there any international transactions that the company produces?

**Yes, Artemis Financial conducts international transactions, necessitating compliance with laws such as GDPR, ensuring secure data transfers, and encryption for global communications.**

* Are there governmental restrictions on secure communications to consider?

**Compliance with U.S. laws such as SOX (Sarbanes-Oxley Act) and international financial regulations is required. Encryption standards (e.g., TLS 1.3) must be followed.**

* What external threats might be present now and in the immediate future?

**Threats include phishing attacks, SQL injections, cross-site scripting (XSS), and zero-day vulnerabilities. The rise of AI-driven cyberattacks may also pose new challenges.**

* What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?

**Modernization includes leveraging secure open-source libraries, adhering to evolving web standards, and ensuring scalability while maintaining robust security measures.**

**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. Authentication and Authorization: Ensures only authorized users can access sensitive client data. Weak authentication could lead to unauthorized access.
2. Data Validation: Prevents malicious input (e.g., SQL injection). Input validation is critical for securing client and transaction data.
3. Session Management: Protects user sessions from hijacking and ensures session timeout mechanisms are in place.
4. Encryption: Ensures sensitive data in transit and at rest is protected, minimizing the risk of breaches.
5. Error Handling: Proper error handling prevents information leakage that could be exploited by attackers.
6. Third-Party Dependencies: Regular checks on open-source libraries ensure vulnerabilities are patched promptly.

**3. Manual Review**

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

1. **myDateTime.java:** No access modifiers on variables, allowing potential unauthorized modifications.
2. **customer.java:** The account\_balance field lacks privacy, enabling unauthorized changes.
3. **DocData.java:** Auto-generated catch block allows access to potentially unintended sections of code.
4. **Login.java:** No input sanitization, exposing the system to SQL injection attacks.
5. **EncryptionUtils.java:** Weak encryption algorithm (e.g., MD5) used for sensitive data.
6. **Transaction.java:** Missing checks on transaction limits, allowing unrealistic values.
7. **UserSession.java:** Inadequate session expiration mechanism, increasing session hijacking risks.

**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. CVE-2021-33037/ tomcat-embed-websocket-9.0.30.jar & tomcat-embed-core-9.0.30.jar: Apache Tomcat vulnerability allows users to gain unauthorized information. Update to the latest patched version.
2. CVE-2021-22118/ spring-core-5.2.3.RELEASE.jar & spring-aop-5.2.3.RELEASE.jar: Spring Core configuration issue enables privilege escalation. Upgrade Spring libraries to the latest version.
3. CVE-2017-18604/ snakeyaml-1.25.jar: SnakeYAML vulnerability allows entity expansion attacks. Apply available patches.
4. CVE-2020-9488 / log4j-api-2.12.1.jar: Log4j API vulnerability exposes data to man-in-the-middle attacks. Update to Log4j 2.17.0 or higher.
5. CVE-2020-25649/ jackson-databind-2.10.2.jar: Jackson Databind flaw allows unauthorized entity manipulation. Upgrade the library to mitigate risks.
6. CVE-2020-10693/ hibernate-validator-6.0.18.Final.jar: Hibernate Validator bug bypasses input sanitation. Avoid relying on technical preview features.
7. CVE-2018-1000613/ bcprov-jdk15on-1.46.jar BouncyCastle vulnerability leaks private keys. Use the latest patched 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.

1. Add access modifiers to variables and methods in myDateTime.java and customer.java.
2. Ensure proper error handling in DocData.java to restrict unauthorized access.
3. Implement input sanitization in Login.java to prevent SQL injection attacks.
4. Replace weak encryption algorithms in EncryptionUtils.java with industry standards like AES.
5. Apply limits on transaction values in Transaction.java to avoid unrealistic inputs.
6. Implement session expiration mechanisms in UserSession.java to minimize hijacking risks.
7. Update all dependencies (e.g., Apache Tomcat, Spring Core, SnakeYAML, Log4j) to their latest patched versions.
8. Conduct regular security audits and vulnerability scans to address newly identified risks.