# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **03/21/2025** | **Jacob C Wilson** |  |

## 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

Jacob Wilson

**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 communication is critical for Artemis Financial due to the nature of its services. As a financial consulting firm, it handles sensitive customer data, including personally identifiable information, banking details, and investment records. Therefore, the value is high in secure communications for this company to exceed expectations in their business operations.

Artemis Financial may engage in international transactions and financial planning for clients across different jurisdictions. This would include cross border data transfers, encryption standards, financial regulations, and secure APIs which would ensure business standards, along with governmental restrictions for secure international transactions for the company.

Artemis’s web application faces multiple external threats, both present and emerging such as phishing attacks, SQL injection, cross site scripting, main in the middle attacks, API security risks, zero-day exploits, and Ransomware attacks. These attacks would attempt compromise user sessions, interceptions attacks, use of improperly secured APIs to expose sensitive customer data, undiscovered vulnerabilities in third-party software, exploiting web form vulnerabilities and many more.

Modernization requirements must be considered to ensure that Artemis Financial remains secure and compliant. The roles of open-source libraries should include dependency management, software composition analysis, and least privilege principle. These roles are detrimental to the quality of security that is a must in the banking web applications for the company by ensuring regular updates and vulnerability patching within the application, implementing tools like OWASP dependency check to identify vulnerable libraries, and restricting third-party dependencies to only those necessary for operations. Evolving web application technologies must be compliant with secure development practices, API security best practices, zero trust architecture, secure software development lifecycle, and cloud security measures involved.

**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**

Ensures all user-provided data is checked and sanitized to prevent SQL Injection, XSS, and CSRF attacks. Artemis Financial must use allowlists and parameterized queries to prevent malicious input.

**Secure Input & Representations**

User input should be properly formatted and securely processed to prevent manipulation. Encoding special characters and normalizing input data will ensure data integrity.

**Architecture Review**

A full system review helps identify vulnerabilities in client-server interactions, API security, and authentication. Security-by-design principles like Zero Trust and network segmentation should be enforced.

**Code Review**

Manual and automated code reviews will help detect logic errors, insecure authentication, and hardcoded credentials. Implementing security tools like SAST and DAST ensures best coding practices.

**Secure Error Handling**

Error messages should not reveal sensitive system details. Generic messages should be shown to users, while detailed logs should be recorded internally for security monitoring.

**Secure Coding Practices & Code Quality**

Strict coding standards must be followed to prevent common vulnerabilities. Developers should use secure development frameworks, apply least privilege access, and undergo security training.

**Secure Data Structures & Encapsulation**

Sensitive financial data must be securely stored and managed using encrypted databases and access controls to prevent unauthorized access and ensure compliance.

**Cryptography & Encryption**

Data must be encrypted at rest (AES-256) and in transit (TLS 1.3) to protect against breaches. Proper key management is essential to maintain data security.

**Secure API Interactions**

APIs must use OAuth 2.0, rate limiting, and API gateways to prevent unauthorized access and data leaks. Regular audits should ensure API security compliance.

**Plug-Ins & Third-Party Components**

Third-party libraries and plugins must be regularly updated and scanned for vulnerabilities using OWASP Dependency-Check to prevent supply chain attacks.

**3. Manual Review**

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

**SQL Injection Vulnerability (DocData.java)**

The read\_document() method establishes a direct database connection with hardcoded credentials.

This implementation is vulnerable to SQL injection if user input is used in SQL queries.

**Hardcoded Credentials (DocData.java)**

The database connection string contains hardcoded credentials which is a major security risk

**Insecure API Implementation (CRUDController.java)**

The /read endpoint in CRUDController does not perform any input validation on the business\_name parameter.

Potential for reflected cross-site scripting or information disclosure if user-controlled input is processed without sanitization.

**Lack of Authentication & Authorization**

There is no authentication or authorization mechanism in place for accessing API endpoints.

Anyone can access endpoints like /read and /greeting without authentication.

**Insecure Object Exposure (CRUD.java, Greeting.java)**

The CRUD and Greeting classes expose object properties without data protection.

If user input is directly passed into these objects, sensitive data leakage can occur.

**Unrestricted Data Modification (customer.java)**

The deposit() method allows unrestricted modification of account\_balance.

There are no validation checks to prevent negative deposits or unauthorized modifications.

**Missing Secure Error Handling**

No custom error-handling mechanism is present.

If an exception occurs (e.g., database connection failure), raw stack traces may be exposed to users.

**No HTTPS Enforcement**

There is no indication that HTTPS is enforced in the application.

**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

A list of information with text

AI-generated content may be incorrect.

**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.

Artemis Financial’s software application contains several security vulnerabilities identified through manual review and static testing. The primary concerns include SQL injection risks due to unparameterized queries, hardcoded credentials in the codebase, lack of input validation, and insecure API endpoints with no authentication or authorization. Additionally, dependencies such as Log4j, Jackson Databind, and SnakeYAML have known vulnerabilities, exposing the system to potential remote code execution, deserialization attacks, and data leaks. The use of outdated cryptographic libraries like Bouncy Castle further increases security risks.

To mitigate these issues, Artemis Financial should implement secure coding practices by using parameterized queries to prevent SQL injection, removing hardcoded credentials, and enforcing strong input validation. API security must be strengthened by integrating JWT-based authentication, role-based access control (RBAC), and rate limiting to prevent unauthorized access and abuse. Error handling should be improved to display generic messages instead of revealing sensitive system information, while HTTPS enforcement must be implemented to protect data in transit.

Updating vulnerable dependencies is also critical. Artemis Financial should upgrade Spring Boot, Log4j (to 2.17.1+), Jackson Databind, and SnakeYAML to their latest secure versions to prevent known exploits such as Log4Shell and unsafe deserialization attacks. Regular software composition analysis should be performed using tools like OWASP Dependency-Check and Snyk to detect outdated libraries before deployment.

Beyond code improvements, security monitoring and incident response should be enhanced. Implementing Web Application Firewalls (WAFs), SIEM solutions, and real-time logging will help detect and respond to threats quickly. Automated static and dynamic application security testing (SAST & DAST) should be integrated into the CI/CD pipeline to identify vulnerabilities early. By adopting these mitigation strategies, Artemis Financial can significantly strengthen its security posture, protect sensitive customer data, and ensure compliance with industry security standards.