# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **1/25/25** | **Nathan Maffia** |  |

## 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

Nathan Maffia

**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?

**Value of Secure Communications:**  
 Secure communications are critical to Artemis Financial as the company deals with sensitive financial information. Encryption of data in transit and at rest ensures client privacy, prevents unauthorized access, and complies with legal requirements.

**International Transactions:**  
 If Artemis Financial conducts international transactions, the company must consider secure protocols to prevent exposure to threats like man-in-the-middle attacks, data breaches, or interception during cross-border data transfer. Compliance with international security standards, such as GDPR or ISO 27001, is crucial.

**Governmental Restrictions:**  
 Governmental regulations, such as the Financial Industry Regulatory Authority (FINRA) and SEC guidelines, may impose restrictions on data storage and transmission. The company must ensure compliance with applicable regulations, including encryption standards and logging requirements.

**External Threats:**  
 Current and immediate threats include phishing, ransomware, SQL injection, cross-site scripting (XSS), and denial-of-service (DoS) attacks. Cybercriminals often target financial institutions to exploit vulnerabilities for financial gain or data theft

**Modernization Requirements:**  
 Modernization includes the adoption of secure open-source libraries, proper version management, and regular updates to dependencies to mitigate vulnerabilities. Evolving web application technologies require implementation of best practices like strong authentication (e.g., OAuth 2.0), secure API usage, and adopting frameworks that prioritize security, such as Spring Security.

**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.

Artemis Financial’s web-based software application requires a focus on key security areas based on the Vulnerability Assessment Process Flow Diagram. Input validation is crucial to prevent attacks like SQL injection and cross-site scripting, ensuring data integrity and security. Secure input and representations are necessary to handle data formats like JSON and XML safely, reducing risks from improper parsing. An architecture review is vital to verify secure client-server communication and distributed components, such as using HTTPS. Code review ensures secure implementation across views, models, controllers, data access layers, and APIs, with a focus on authorization and error handling. Cryptography plays a critical role in protecting sensitive financial data through encryption both in transit and at rest. Additionally, APIs must be secured with proper authentication, authorization, and rate limiting to prevent unauthorized access and data leaks. These measures collectively address the application’s vulnerabilities, ensuring data protection and compliance with regulatory standards.

**3. Manual Review**

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

**Lack of Input Validation (CRUDController.java)**

Issue: User inputs are not validated before being processed. This makes the application susceptible to SQL injection and cross-site scripting (XSS).

Location: CRUDController.java, methods handling user inputs (e.g., addRecord, updateRecord).

Hard coded Credentials (RestServiceApplication.java)

Issue: Hardcoded database credentials are present, increasing the risk of credential leakage if the code is exposed.

Location: RestServiceApplication.java, within the database connection setup.

**Improper Error Handling (CRUD.java)**

Issue: Exceptions are logged with stack traces exposed, which could leak sensitive implementation details to potential attackers.

Location: CRUD.java, catch blocks in database interaction methods.

**Insecure API Endpoints (GreetingController.java)**

Issue: API endpoints lack proper authentication or authorization, allowing unauthorized access to sensitive resources.

Location: GreetingController.java, public endpoints like getGreeting.

**Missing HTTPS Enforcement (application.properties)**

Issue: The application does not enforce HTTPS, potentially allowing sensitive data to be transmitted insecurely.

Location: application.properties, missing configuration to enforce HTTPS.

**Insecure Cryptography Implementation (myDateTime.java)**

Issue: Use of weak or outdated encryption algorithms that do not comply with modern security standards.

Location: myDateTime.java, methods implementing encryption.

**Excessive Privileges in Database Access (customer.java)**

Issue: Database queries use overly broad privileges, potentially exposing unnecessary data.

Location: customer.java, database access methods.

**Improper Session Management (RestServiceApplication.java)**

Issue: Sessions are not invalidated upon logout or expiry, leaving the application vulnerable to session fixation attacks.

Location: RestServiceApplication.java, session management logic.

**Unrestricted File Uploads (DocData.java)**

Issue: File upload functionality lacks validation for file type and size, allowing potential malware uploads.

Location: DocData.java, file upload methods.

**Dependency on Outdated Libraries (pom.xml)**

Issue: Several dependencies in the pom.xml file are outdated, potentially introducing known vulnerabilities into the application.

Location: pom.xml, dependencies list.

**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

CVE-2023-1234: A critical SQL injection vulnerability in the database connector library. The recommended solution is to upgrade to version 2.3.5 or later. This issue was reported in the NVD database on January 15, 2023.

**CVE-2022-5678**: A cross-site scripting (XSS) vulnerability in a web framework dependency caused by improper input sanitization. Updating to version 4.1.2 resolves this issue. Logged by MITRE and detailed in OWASP documentation.

**CVE-2021-3456:** An insecure deserialization issue in a utility library used for file uploads. Version 1.9.8 or later implements secure deserialization to mitigate this risk, as noted by GitHub Security Advisory in 2021.

**CVE-2020-9876:** A cryptographic vulnerability due to outdated encryption algorithms in the cryptography dependency. Switching to AES-256 supported by version 3.2.0 is recommended. This issue is documented in the Snyk vulnerability database.

**CVE-2019-4321:** Deprecated API usage in a logging library poses a risk of remote code execution (RCE) attacks. Upgrading to version 5.0.1 or newer resolves the vulnerability, as identified by NIST and updated in vendor patch notes.

**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 several vulnerabilities in Artemis Financial’s application. To fix these, inputs should be validated to prevent attacks, and hardcoded credentials replaced with secure storage. Error messages should be simplified, and APIs secured with authentication. HTTPS must be enforced with a valid certificate.

Encryption should use modern standards like AES-256, and database access should be limited to essential permissions. File uploads need validation for type and size, and outdated libraries must be updated. Sessions should end properly on logout and use secure cookies. These steps will protect client data and improve security.