# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **[Date]** | **[Your name]** |  |

## 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

Christopher Harris

**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?

Artemis Financial should ensure that the security of its RESTful web application programming interface, or API, is higher to protect sensitive financial data from any threat. Protection of customer data via secure communication is an essential issue to maintain client trust and also not to violate the law of regulatory compliance. Since the company is dealing with international transactions, it needs to be compliant with both domestic and international laws related to secure communications, such as GDPR and PCI DSS.

Other external threats include the possibility of man-in-the-middle attacks, injection vulnerabilities, and abuse of APIs. The other immediate threats involve very outdated open-source libraries that could introduce vulnerabilities to the application. Modernization requirements include secure coding practices, encryption of data in transit and at rest, and proper input validation. Additionally, the application has to keep up with the evolution of web application technologies in terms of 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.

Based on the vulnerability assessment process flow diagram, areas of security that are pertinent to Artemis Financial include:

* **Input Validation:** Ensuring all the users' inputs are sanitized to block injections and corruption of data.
* **APIs:** Securing API endpoints against unauthorized access and abuse.
* **Cryptography:** Applying encryption to protect sensitive data both in transit and at rest.
* **Client/Server:** Ensuring secure communication between clients and servers by using HTTPS and secure authentication mechanisms.
* **Code Quality:** Following secure coding practices to minimize vulnerabilities that might be introduced.
* **Encapsulation:** Protecting sensitive data structures through secure access controls.

Each of these areas is important in maintaining the integrity and confidentiality of financial data while supporting Artemis Financial's modernization goals.

**3. Manual Review**

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

Following are some of the vulnerabilities identified during manual inspection of the code base:

* **Input Validation:** Some input fields aren't sanitized, making them susceptible to SQL injection attacks.
* **Hardcoded Credentials:** Sensitive credentials are embedded directly into the source code.
* **Unsecured API Endpoints:** Some API endpoints lack adequate authentication and authorization mechanisms.
* **Outdated Libraries:** Dependencies such as spring-core and jackson-databind are outdated, introducing known vulnerabilities.
* **Missing Error Handling:** Certain sections of the code fail to catch exceptions, potentially exposing sensitive information to attackers.
* **Weak Cryptography:** The application relies on outdated cryptographic algorithms that are vulnerable to exploitation.
* **Exposed Sensitive Data:** Debug logs contain sensitive information, including customer details.
* **Weak Session Management:** Sessions are not properly invalidated upon logout, increasing the risk of unauthorized access.
* **Insecure File Upload:** File upload endpoints do not validate uploaded files, potentially allowing malicious files to be uploaded.
* **Excessive Permissions:** Some services operate with more permissions than necessary, violating the principle of least privilege.

**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 screenshot of a computer

Description automatically generated

**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 following mitigation steps are recommended to address the identified vulnerabilities:

* **Input Validation:** Apply stringent validation and sanitization to all user inputs.
* **Remove Hardcoded Credentials:** Securely store credentials using environment variables or a secrets management service.
* **Secure API Endpoints:** Implement authentication and authorization for all API endpoints.
* **Update Dependencies:** Upgrade to the latest versions of outdated libraries to eliminate known vulnerabilities.
* **Implement Error Handling:** Incorporate comprehensive exception handling to prevent information disclosure.
* **Improve Cryptography:** Transition to modern, secure cryptographic algorithms and protocols.
* **Protect Sensitive Data:** Remove sensitive information from debug logs and adopt proper logging practices.
* **Improve Session Management:** Enforce session expiration and ensure sessions are invalidated after logout.
* **Secure File Uploads:** Validate file types and scan uploaded files for potential threats.
* **Apply Principle of Least Privilege:** Review and minimize permissions for services and processes to the least necessary for functionality.