# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **19/5/2024** | **Samuel Rincon** | **Vulnerability assessment report** |

## 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

Samuel Rincon

**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:

* Value of Secure Communications: Secure communications are vital to protect sensitive financial data, maintain client trust, and comply with regulatory requirements.
* International Transactions: If Artemis Financial conducts international transactions, it must ensure compliance with global security standards and regulations, such as GDPR for European clients.
* Governmental Restrictions: Depending on the regions in which Artemis Financial operates, there may be specific governmental restrictions on encryption and data handling that must be considered.
* External Threats: Potential threats include phishing attacks, data breaches, ransomware, and DDoS attacks. Emerging threats like zero-day vulnerabilities and advanced persistent threats (APTs) also pose risks.
* Modernization Requirements: The software should incorporate secure open-source libraries, keep up with evolving web technologies (e.g., frameworks like React or Angular), and ensure that all dependencies are regularly updated.

**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.

* Architecture Review: Essential for understanding the overall structure and identifying any architectural weaknesses.
* Input Validation: Critical to prevent injection attacks such as SQL injection, XSS, and command injection.
* APIs: Ensuring secure API interactions is crucial for protecting data exchanged between client and server.
* Cryptography: Necessary to protect data at rest and in transit using strong encryption algorithms.
* Client/Server: Focus on securing communications between clients and servers to prevent man-in-the-middle attacks.
* Code Quality: Ensures adherence to secure coding practices, reducing the likelihood of vulnerabilities due to poor coding standards.
* Encapsulation: Helps in securing data structures and preventing unauthorized access or modifications.

**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. SQL Injection: User inputs are directly concatenated into SQL queries without proper sanitization (e.g., in DatabaseHelper.java).
2. Cross-Site Scripting (XSS): User inputs are reflected in HTML responses without proper encoding (e.g., in UserProfileServlet.java).
3. Hardcoded Credentials: Hardcoded credentials found in the source code (e.g., in Config.java).
4. Sensitive Data Exposure: Sensitive information is logged or displayed in error messages (e.g., in ErrorHandler.java).
5. Unrestricted File Uploads: Lack of validation for file uploads, allowing potentially malicious files (e.g., in FileUploadServlet.java).
6. Insecure Direct Object References (IDOR): Direct access to objects based on user-supplied input without authorization checks (e.g., in AccountServlet.java).
7. Inadequate Error Handling: Detailed error messages revealing stack traces or internal implementation details (e.g., in ExceptionHandler.java).

**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:

Vulnerabilities that I found:

* CVE-2021-44228 (Log4j): A critical vulnerability in Log4j allowing remote code execution. Update to the latest version.
* CVE-2021-45046 (Log4j): An additional vulnerability in Log4j related to the previous CVE. Also mitigated by updating Log4j.
* CVE-2020-11979 (Apache Commons BeanUtils): An exposure in BeanUtils which can lead to arbitrary code execution. Update to the latest version.

Solutions:

* Update Log4j to version 2.17.1 or later.
* Regularly review and update dependencies to their latest secure versions.
* Use tools like OWASP Dependency-Check to automate vulnerability detection in dependencies.

**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.

* SQL Injection: Implement prepared statements and parameterized queries in all database interactions.
* XSS: Encode all user inputs before reflecting them in HTML responses.
* Hardcoded Credentials: Use environment variables or secure vaults to manage sensitive credentials.
* Sensitive Data Exposure: Ensure sensitive data is not logged or displayed in error messages. Implement proper logging practices.
* Unrestricted File Uploads: Validate file types and use antivirus software to scan uploaded files.
* IDOR: Implement access controls to check user permissions before granting access to objects.
* Inadequate Error Handling: Display generic error messages to users and log detailed errors internally.