# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **7/18/2024** | **Rex Green** |  |

## 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

Rex Green

**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?

The client wishes to modernize its operations through a secure web application. This application will be used to handle sensitive financial data, and therefore requires robust security measures. The application will utilize a RESTful API, which will need protection from external threats.

* Secure communication is critical to Artemis Financial as it protects sensitive financial data from unauthorized access or interception. This includes user login credentials, account details, investment information, transaction data.
* The scenario doesn’t explicitly mention international transactions, however, given that the software is for entrepreneurs, business, and government agencies around the world, it is likely that there are international transactions produced by the company.
* If the software is indeed used by government agencies, then there are indeed governmental restrictions to consider. Artemis Financial should comply with relevant data privacy regulations.
* Some external threats include data breaches, injection attacks, DoS attacks, compromised third-part libraries or APIs, and phishing or other social engineering tactics.
* Some modernization requirements to consider are the security vulnerabilities of open-source libraries, maintaining up-to-date versions of libraries to address unknown vulnerabilities, and consider the license terms of open-source libraries to ensure compatibility with Artemis Financial’s needs.

**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.

Areas of security which apply to Artemis Financial’s software application:

* Input Validation: This is crucial for Artemis Financial’s application, as it handles sensitive financial data. Malicious users may try to inject code or exploit vulnerabilities through user input fields, and therefore all data received from users must be sanitized and met expected formats.
* APIs: Since the application uses RESTful API, which needs protection, Secure API interaction practices like authentication, authorization, and encryption are essential to prevent unauthorized access and data breaches.
* Cryptography: Financial data needs to be encrypted both at rest and in transit. Cryptography ensures that confidentiality and integrity of sensitive information.
* Client-Server Distributed Composing: The application likely consists of a client-side interface and a server-side component. Secure communication between these parts is essential. This area of security focuses on secure data transmission methods and potential vulnerabilities in these mechanisms.
* Code Review: Code review helps identify and fix vulnerabilities in the application’s code.
* Data Access Control: As this application stores sensitive financial data, only authorized users must be able to access specific data and prevent unauthorized modification.

**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. The read\_document method in DocData.java seems to be building a connection string directly in the code. This is insecure as credentials like username and password are exposed in plain text. This should use a separate configuration file to store sensitive information
      2. In CRUDController.java, the /read endpoint might be vulnerable to open redirects if user input is used to construct a URL for redirection. An attacker could inject malicious URLs to redirect users to phishing sites
      3. Lack of input validation. There is no apparently input validation throughout the code.
      4. DocData.java catches a SQLException but does not handle it properly.
      5. The CRUD.java class constructor takes user-provided content and assigns it to two member variables. Without proper validation, this may expose sensitive information through getters (getContent and getContent2).
      6. DocData.java could be vulnerable to SQL injection if input is not properly sanitized.
      7. GreetingController.java uses an AtomicLong counter for generating greetings. While this is not a critical vulnerability, generating greetings this way may not be secure for applications requiring strong randomness.

**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

Here is the summary of the security vulnerabilities in the code:  
Graphical user interface, text, application

Description automatically generated

****

The dependency check found 13 vulnerable dependencies with a total of 86 vulnerabilities. Here is a breakdown of the most critical ones:

* bcprov-jdk15on-1.46.jar (HIGH): This cryptography library has vulnerabilities that could allow attackers to compromise your system. Upgrade to Bouncy Castle version 1.48 or later.
* spring-boot-2.2.4.RELEASE.jar (CRITICAL): Spring Boot has critical vulnerabilities. Update to a newer version.
* log4j-api-2.12.1.jar (CRITICAL): This logging library has critical vulnerabilities. Refer to the full report for details and update recommendations.
* snakeyaml-1.25.jar (CRITICAL): This YAML processing library has critical vulnerabilities. Upgrade to a newer version.
* tomcat-embed-core-9.0.30.jar (CRITICAL): This Apache Tomcat embedded library has critical vulnerabilities. Upgrade to a newer 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.

High-severity vulnerabilities:

* Insecure Credential Storage (DocData.java): Storing database credentials directly in the code exposes them to anyone with access to the codebase.
* Open Redirect Vulnerability (CRUDController.java): Building redirect URLs from user input can lead attackers to redirect users to malicious websites.
* Lack of Input Validation: The code doesn't validate user input, potentially allowing injection attacks (SQL injection, XSS) and manipulation of application data.
* Improper Exception Handling (DocData.java): Catching a SQLException but not handling it properly can mask underlying issues and prevent proper logging or error reporting.
* Potential Information Disclosure (CRUD.java): Assigning user-provided content directly to member variables without validation could expose sensitive information through getters.
* SQL Injection Vulnerability (DocData.java): If user input is not sanitized before building SQL queries, it can be exploited for SQL injection attacks.

Recommendations:

* Secure credential storage in a separate configuration file
* Prevent open redirects
* Implement input validation
* Properly handle exceptions
* Validate user input before use or assignment
* Prevent SQL injection attacks by using parameterized queries or prepared statements