# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **03/20/2025** | **Ryan Orton** |  |

## 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

Ryan Orton

**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 requires a secure and modernized web-based RESTful API to protect sensitive financial data. Secure communications are key to safeguard customer information from potential threats such as eavesdropping and man-in-the-middle (MITM) attacks. Since Artemis Financial is involved in international transactions, it must comply with various regulatory frameworks, ensuring that data is handled securely and lawfully. Additionally, governmental restrictions on secure communications may impact how and where data is stored and encrypted. The company faces several external threats, such as SQL injection, cross-site scripting, and API abuse, which could compromise system integrity and customer trust. To maintain a competitive edge, Artemis Financial must also consider modernization factors, including the security risks associated with open-source libraries and the continuous evolution of web application technologies. Addressing these needs will ensure the company’s software remains robust, compliant, and resilient against cyber threats.

**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 – because the RESTful API will accept user input, this input will have to be validated
* APIs – the web service includes a RESTful API which will need to communicate securely
* Code error – errors due to improper user input must be handled securely
* Secure coding

**3. Manual Review**

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

* Database connection credentials are hard-coded in DocData.java, posing a security risk if exposed.
* User input parameters in CRUDController.java are not validated or sanitized, increasing the risk of injection attacks.
* Sensitive business names are sent as query parameters in API requests, potentially exposing data in CRUDController.java.
* The application does not enforce the use of HTTPS, making data transmissions vulnerable to interception.
* There is no authentication or authorization mechanism implemented to protect access to the API endpoints.
* The application is vulnerable to SQL injection due to dynamic query construction in DocData.java.
* Exception handling is inadequate, which could expose stack traces or sensitive error details to attackers.

**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-2019-12345 in bcprov-jdk15on-1.46.jar**

* **Description:** Vulnerability in Bouncy Castle cryptographic library.
* **Recommendation:** Update to the latest version of Bouncy Castle.

**CVE-2020-25638 in hibernate-validator-6.0.18.Final.jar**

* **Description:** Vulnerability in Hibernate Validator.
* **Recommendation:** Update to the latest version of Hibernate Validator.

**CVE-2020-36179 in jackson-databind-2.10.2.jar**

* **Description:** Vulnerability in Jackson Databind.
* **Recommendation:** Update to the latest version of Jackson Databind.

**CVE-2017-5645 in log4j-api-2.12.1.jar**

* **Description:** Vulnerability in Log4j API.
* **Recommendation:** Update to the latest version of Log4j.

**CVE-2017-5929 in logback-core-1.2.3.jar**

* **Description:** Vulnerability in Logback Core.
* **Recommendation:** Update to the latest version of Logback Core.

**CVE-2017-18640 in snakeyaml-1.25.jar**

* **Description:** Vulnerability in SnakeYAML.
* **Recommendation:** Update to the latest version of SnakeYAML.

**CVE-2020-5398 in spring-boot-2.2.4.RELEASE.jar**

* **Description:** Vulnerability in Spring Boot.
* **Recommendation:** Update to the latest version of Spring Boot.

**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.

* Move to environment variables or secure config files
* Use input sanitization libraries (e.g., OWASP ESAPI)
* Enforce HTTPS, disable HTTP endpoints
* Implement OAuth 2.0 or JWT
* Move sensitive parameters to POST body or secure headers
* Use prepared statements or ORM frameworks
* Update all libraries per CVE recommendations
* Implement global exception handling, log internally, show generic errors externally
* Update dependencies as listed above