# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **6/1/2025** | **Jesse O’Barr** |  |

## 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

Jesse O’Barr

**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 is focused on modernizing its financial consulting services by upgrading its web-based RESTful API. As a company handling sensitive customer data related to savings, investments, and insurance, secure communication is important for maintaining customer trust and legal compliance. Secure communication ensures that personal and financial data is not leaked or altered during transmission.

The company may handle international transactions, meaning it needs to comply with global data protection laws like GDPR in Europe. This adds more pressure to make sure encryption and secure access control are in place.

Right now, the biggest threats Artemis might face include:

* Cross-site scripting (XSS)
* SQL injection
* Broken authentication
* Insecure data storage
* Outdated or vulnerable third-party libraries

When modernizing the app, they need to:

* Keep open-source libraries updated
* Make sure the app works safely with modern web technologies
* Follow secure coding practices so future problems don’t occur

**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.

Looking at the kind of app Artemis uses, these areas of security are important:

* **Input Validation**: Since the app takes user input, it needs to check that the data is safe. Without this, attackers could send harmful data.
* **Secure input and Representations:** The app should store and show data in a safe way to prevent corruption or any weird behavior.
* **Secure Error Handling**: Errors should be handled in a way that doesn’t give attackers helpful information.
* **Secure Coding Practices**: The app should be written using safe coding habits, like avoiding hardcoded passwords.
* **Cryptography**: Any sensitive data being sent or stored should be encrypted.
* **APIs**: Because it’s a web API, it needs to be protected so that only allowed users can access it.

These areas match what Artemis’s app does and helps reduce the most likely risks.

**3. Manual Review**

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

Here are the 8 problems I found while reading through the Java files:

1. **No Input Validation**

* File: GreetingController.java, CRUDController.java
* The app takes user input but doesn’t check or clean it first. This can lead to attacks like XSS or SQL injection.
* Fix: Add Input validation using Spring’s built-in tools or validation libraries.

1. **No Login or Access Control**

* File: GreetingController.java, CRUDController.java, RestServiceApplication.java
* Anyone can use the API. There are no checks for user roles or login.
* Fix: Use Spring Security to add authentication and role checks.

1. **Unsafe Responses**

* File: GreetingController.java
* The app sends back user input directly in the response. If someone sends bad code, it could show up in someone else’s browser.
* Fix: Sanitize or escape the data before sending it back.

1. **No Proper Error Handling**

* File: All controllers
* Errors might crash the app or show details in the response. This helps attackers learn how the app works.
* Fix: Add global error handler and show only simple error messages.

1. **Unprotected File Uploads**

* File: Possibly CRUD.java and DocData.java
* Files are saved in a folder without checking the path or file type. Attackers could upload harmful files or change paths.
* Fix: Check Filenames, file types, and save them in a secure location.

1. **Data Not Encrypted**

* File: customer.java, DocData.java
* Important data like names or emails are stored without encryption. If someone steals the data, they can read it.
* Fix: Encrypt or hash sensitive data.

1. **Weird File Naming**

* File: customer.java, DocData.java, myDateTime.java
* File names don’t match Java naming rules. This doesn’t hurt security directly, but it makes mistakes more likely.
* Fix: Rename files to follow Java conventions.

1. **Unused API Endpoints Are Open**

* File: GreetingController.java, CRUDController.java
* Some API routes are open but don’t seem to be used or protected.
* Fix: Remove or lock down those routes.

**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

I used OWASP Dependency-Check and found these problems:

**CVE-2018-1000613**

* Library: BouncyCastle 1.46
* Problem: It has a weak encryption and can crash.
* Fix: Update to version 1.65 or later.’

**CVE-2019-10744**

* Library: json-path
* Problem: It might evaluate unsafe expressions.
* Fix: Update to the latest version.

Source: These vulnerabilities were listed on the National Vulnerability Database and confirmed by the Dependency-Check tool.

**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.

Here are ways to fix the issues I found:

1. **Update Problem Libraries**

* Use newer versions of BouncyCastle and json-path.

1. **Add Input Validation**

* Use Spring or Apache tools to check user input.

1. **Force HTTPS**

* Make the app use HTTPS to protect data in transit.

1. **Make File Uploads Safe**

* Check file names and types, and save them securely.

1. **Add Login and Access Control**

* Use Spring Security to add login and role permissions.

1. **Handle Errors Better**

* Show simple error messages and keep details in logs only.

1. **Stop Hardcoding Ports**

* Use environment settings to control the app’s port.

1. **Clean Up Logging**

* Don’t log user input directly. Sanitize it first.

These fixes will help make Artemis Financials app safer and more reliable in the future.