# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **7/20/2025** | **Mitchell Flint** |  |

## 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

Mitchell Flint

**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 modernizing their operations, which frequently handles sensitive financial data. The value placed on secure communications is about as high as it gets, considering the software will be expected to handle sensitive customer data, integrate with a database, and support file uploads. Although I do not see it explicitly stated that international transactions are handled, it can be assumed that there likely will be some. As governmental restrictions are likely to exist, and are not always the same for each of the countries involved, it may be a good idea to inquire further about which specific countries are a guarantee.

Given the sensitivity of the data being handled, it can be assumed that there will be people relentlessly attempting to get in and acquire information. Some of the threats that are particularly relevant here include injection attacks, insecure endpoints, and insecure file handling. Also, any open-source libraries must be kept as up to date as possible, to protect against known exploits that may be patched out in the latest version. Some of the modernization requirements that must be included are proper validation, secure storage practices, and implementation of authentication and access control.

**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.

On review of the flowchart, every section applies here.

* Input Validation – Financial applications process and store sensitive information, and user input can open up risks like injection attacks if not properly sanitized.
* APIs – The company’s services are expected to be used via APIs, likely by user applications. Ensuring proper access control is essential to keeping information that the user doesn’t need safe.
* Cryptography – Strong cryptographic practices are necessary to protect data. Again, given the sensitivity of the data being transmitted here, this would also be essential.
* Client/server – the architecture is likely to involve a distributed setup, with client all over the internet accessing services. Session management would be a key component for reducing risk.
* Code Error – error messages should be secured to ensure they do not expose system internals.
* Code Quality – make sure the code is clean and maintainable is a given essential to all programs, even the simplest and most secure should be done in this way.
* Encapsulation – properly implementing this ensures that business logic and financial records cannot be exposed or altered. This is generally a good practice anyway for keeping the code clean, but has added emphasis here.

**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. Beginning with input validation, I noticed issues with both CRUDController and GreetingController. Each captures input from the URL, without any sort of input validation. As the first line of defense, this should be the first priority for the mitigation plan.
2. Currently there is neither authentication nor authorization, as all end points are publicly accessible.
3. In the application.properties file, I noticed file uploading is enable with a pretty generous limit of 200MB. Without the authentication implemented, it would not be difficult to upload malware or launch a DoS attack.
4. The DocData class logs full stack traces to the console from SQL exceptions errors. If this is left there, internal implementation details will be exposed.
5. The customer class has no proper getting/setter methods, and only the account number is marked private. This can enable access to the application state.
6. Currently in the DocData.java file, credentials are being hardcoded, which is a violation of encryption hygiene.
7. While not a particularly major security issue, it could be worth mentioning the customer and myDateTime classes are not capitalized as per java convention. I might just be being a little nitpicky here, but it would violate code quality standards.

**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

After running the dependency check on this assignment, there were many vulnerabilities discovered, many of critical or high severity. Much of these can once again be fixed by upgrading to the latest versions.

* The two biggest flags were for the tomcat embedded server, each were given critical severity with over 40 CVEs. Upgrading these two modules should be of the highest priority.
* Next was with the spring framework, which had 7 modules flagged. Every one of these were of critical severity, most of which with 13 or more CVEs. Upgrading these share the same priority as the Tomcat server modules.
* One last critical module is the snakeYAML. It has 8 CVEs, including unsafe YAML parsing. This too should upgraded as soon as possible.
* The Bouncy Castle module has a high severity, with 18 CVEs. While not currently used in the program, future use is expected. This should be upgraded soon, though is less of a priority than the previous mentions.

**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.

1. The first step that should be done is to upgrade all vulnerable dependencies flagged during the static testing. Though some are clearly a higher priority than others, due to the sensitive nature of the data, no known vulnerabilities should be allowed to persist, especially if all that is needed is just to update it.
2. After this is completed, Input Validation must be added to the CRUDController and GreetingController files. Field length limitations, restrictions on usable characters, and type enforcement are a start.
3. Next would be to Implement authentication and authorization. Authentication should be required for all endpoints, and access should be restricted by user roles.
4. The next action to fix is the removal of the hardcoded credentials. Credentials should be moved to a secure configuration.
5. Since file uploading is enabled, this should be secured by implementing file type whitelisting, size checks, and stored in an isolated directory.
6. Next item to tackle would be the full stack trace printing in DocData. Instead generic messages should be used, that don’t expose the internal logic.
7. In the customer class, all fields should be marked as private, and the getters and setters should be added as needed.
8. Lastly, customer and myDateTime should be renamed to Customer and MyDateTime, to follow naming conventions.