# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **03-20-25** | **Christian Clark** | **Initial Version** |

## 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

Christian Clark

**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?

Since Artemis Financial is a finance-based company, there should be extreme importance and value being placed in secure communication and security of software. Though all companies that use communication and software should be secure, finance is a particularly lucrative area for criminals to try to exploit for maximum gain. From all the information given, I do not have the knowledge of whether or not Artemis Financial does perform international transactions.

There are governmental restrictions and regulations on financial institutions such as the Safeguards Rule the FTC has to protect customer information (Federal Trade Commission, 2024). These will have to be followed if Artemis Financial is to create an online portal for customers to perform online banking with.

There are many external threats to consider. Some may be domestic, citizens from the country attempting to breach security for gain, or international. International threats can be either state actors or independents looking for either financial gain, or as an attack on the state.

Any software or dependencies that are used in this online software must always be in their most secure state. Whether that is updates to patch known or unknown vulnerabilities or having software developers patch vulnerabilities directly in the codebase. Any software with dependencies are always at risk for zero-day vulnerabilities popping up without warning, so we must be diligent in catching them as fast as possible to minimize any risk of leaking data.

**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.

Out of the seven areas of security presented, we must take into consideration the use of the software we’re trying to write. All seven areas are extremely important, but the most important are Input Validation, APIs, and Client/Server communication, which can also have an emphasis on Cryptography as well. We’re writing a forward-facing public program where users are encouraged to log into their finance accounts. This can present an ‘in’ for criminals looking to exploit something for access to the server itself. We must make sure we’re taking input validation seriously and prevent any and all exploits like that from occurring. Secure API interaction also falls into that circle as well, as improper use of APIs can create a situation where input validation may not catch an exploit. We must make sure to take care when use APIs, especially since we’re going to be working with a RESTful server.

Making sure both the Client and Server are communicating properly and securely is also important. Should communications be insecure, instead of Artemis Financial’s data being leaked, it could be that one customer’s data instead. That is just as bad as our server being breached, as if a single customer is at risk, we are also at risk. This is where Client/Server communication and Cryptography go hand in hand. We need to make sure all communication is end-to-end encrypted to prevent man in the middle attacks and prevent any data from being stolen.

**3. Manual Review**

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

A screen shot of a computer

AI-generated content may be incorrect.

Starting with DocData.java, it is extremely concerning that we have a URL that allows for access to our database. This is very easily findable by trying known testing words as the path for our main URL. Additionally, having root as both the username and password can easily be guessed and tried to allow for access to the database. This is unacceptable and should be changed immediately. The printstacktrace function call is also concerning, as there can be the potential for an attacker to gain access to the stack trace from some fashion and use it to break into the system further or leak information from either us or customers.

A screen shot of a computer program

AI-generated content may be incorrect.

Another small vulnerability in GreetingController.java, but one that can be avoided, is putting user input directly into String.format(). While not extremely dangerous, it can be leveraged by an attacker to put extremely large amounts of data into the function to cause a DOS attack if done repeatedly. This also has the possibility of adding an additional exploit later on if no validation is done before being put into String.format().

A screenshot of a computer program

AI-generated content may be incorrect.

Another vulnerability is the deposit function in customer.java. There are no checks to make sure that the number is not negative, and we are just modifying the account\_balance value directly. This is unsafe, as an attacker can just either drain an account into whatever value they want or put an infinite amount of money into their own account. There is no logging either, as any unauthorized access will be untraceable.

A screen shot of a computer program

AI-generated content may be incorrect.

While CRUDController.java is not necessarily vulnerable, it does leave a potential in for attackers as business\_name does not have any validation. This value is never passed anywhere and is never used but still presents a potential for a vulnerability if this goes unchecked in the future. This returned page will also present an error page that will expose the necessary string parameter required to make the read page work. This can present a potential leak of information as we do not want attackers to know the correct way to work our program from the outside. When business\_name is provided, the page will provide internal working of the doc data we create and pass into CRUD. While not necessarily problematic at the moment, it could provide a vulnerability later. This is especially concerning as DocData hold the username, password, and location of our database.

**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

A screenshot of a computer

AI-generated content may be incorrect.



This project has a total of 15 dependencies that have a combined total of 157 vulnerabilities. 7 of these dependencies have vulnerabilities that have been known to have been exploited. 10 dependencies have critical level vulnerabilities. The combined total of this many extremely critical vulnerabilities is an absolute priority to solve. An example of one of the vulnerabilities is CVE-2020-1938, which is a known exploited vulnerability that can let AJP connections have a higher trust than normal (National Vulnerability Database, 2025). CVE-2020-1938 is a vulnerability not only in the tomcat-embed-websocket-9.0.30.jar dependency but also in tomcat-embed-core-9.0.30.jar. This can present a problem as if we only update one of these dependencies, the other can remain with that same vulnerability if we ignore it. Another known exploited vulnerability is CVE-2022-22965 in the spring-webmvc-5.2.3.RELEASE.jar dependency. This vulnerability is an extremely bad vulnerability that allows for remote code execution (NATIONAL VULNERABILITY DATABASE, 2025). We must do everything in our power to patch and prevent this vulnerability, as RCE is the most powerful access an attacker is allowed to have. It is recommended to update all dependencies to an updated state that has either no vulnerabilities, or low risk vulnerabilities that can be mitigated against. All code using those dependencies should be looked over to ensure that functionality remain the same after the updates, and for any vulnerabilities that remain that can be patched in the code.

**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.

From the provided static test and manual code review, there are many vulnerabilities that have been found and must be fixed. At least 7 vulnerabilities from the code review have been found and must be fixed as soon as possible. Additionally, there are over 150 vulnerabilities in our dependencies, and 10 of them are critical level. This is unacceptable and cannot remain in this state or else our customer’s data is in danger.

Our first and most obvious step is to update our dependencies to a state where they have the least amount and effective vulnerabilities. This will solve quite a lot of our problems as a great majority of the weaknesses we have are through vulnerabilities that have been actively exploited in the past. Additionally, cleaning our code to prevent these obvious and not so obvious exploits is also key. Things such as input validation, proper API use, and not leaving full database access in the code are key. Clean and secure code is the number one way to prevent a majority of these exploits. Going forward, all software engineers should be following a secure code workflow to prevent any and all potential exploits that may crop up in the future.

# Works Cited

Federal Trade Commission. (2024, December). *FTC Safeguards Rule: What Your Business Needs to Know*. Retrieved from Federal Trade Commission: https://www.ftc.gov/business-guidance/resources/ftc-safeguards-rule-what-your-business-needs-know

National Vulnerability Database. (2025, February 6). *CVE-2020-1938 Detail*. Retrieved from National Vulnerability Database: https://nvd.nist.gov/vuln/detail/CVE-2020-1938

NATIONAL VULNERABILITY DATABASE. (2025, January 29). *CVE-2022-22965 Detail*. Retrieved from NATIONAL VULNERABILITY DATABASE: https://nvd.nist.gov/vuln/detail/CVE-2022-22965