# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **March 20, 2025** | **Christopher Braman** |  |

## 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

Christopher Braman

**1. Interpreting Client Needs**

For Artemis Financial specifically, the value of secure communication between clients and the server is of dire importance. This is due to the type of company that Artemis Financial is. This company handles individual’s savings, retirement, investments, and insurance, so almost all of the person’s personal income. Ensuring the safety of the customers personal information and finances should be the top priority of Artemis Financial. The company does make internal transactions through the use of all of their investments and retirement. These two types of transactions can be spent with the customers’ personal incomes and then sold or traded after the purchase. With this in mind it means additional advice would be some sort of 2FA (Two Factor Authentication) but that will be delved further into the mitigation plan. With being a financial institution there are government guidelines and restrictions that they must adhere by, however; with the company attempting to be global it will need to adhere to whichever country it is based in. Since I am US based, I shall be using our government’s current guidelines. <https://www.ffiec.gov/pdf/efs.pdf> The preceding URL is for the US electronic financial guidelines.By adhering to these guidelines and restrictionsthe company could even become part of the FDIC which would help ensure and insure their customer’s safety so long as they are within the United States, which would also give their customer a sense of relief. External threats for this type of company would be plentiful considering that they will be dealing with personal finances. Obtaining personal logins via HTTP is one of the security concerns, and we will talk about that later in the coding mitigation, along with the admin rights. These two different styles of attacks would allow hackers to get personal information and then transfer funds to different stocks and trade, transfer money, cancel insurances causing a DoS (denial of service), and potentially allow them to sell other types of information on the ‘dark web.’ As for the admin accessibility, this will allow hackers to change passwords on a whim if that access is granted to the admin, accounts could be deleted on put on hold, and the whole system could be deleted if proper API protection is not taken. These two types of attacks should be the main concern for the company, with which the obtaining of personal information should be their first focus. Using open-source libraries could lead to the cause of admin accessibility, so understating and using the correct libraries is going to be the next crucial step. A good way to go about how/what libraries should be used is through the government guidelines and restrictions. Having that base line will help Artemis Financial choose which libraries would be best suited for what they need. This goes hand in hand with how the web is ever evolving. New applications and libraries will become available and choosing the brightest and newest one may not always be the best option because those could lead to new and undiscovered holes in security. Using a relatively newer/older (depending on perspective) will allow for Artemis Financial to make the security walls that they will need in order to safeguard the customers as well as their own information because they have been tried and tested over the years.

**2. Areas of Security**

Upon looking at the code the areas of security I would like to point out are as follows: Input Validation [customer.java], Secure API, Cryptography [DocData.java], Client/Server [DocData.java], Code Quality [customer.java] [DocData.java]. These areas are a major concern to the customers’ personal information, and the type of information that an outside or even an internal source could manipulate. Input Validation is one of the first I noticed, and with this it allows hackers to inject their own code or even different variables into the system causing either unwanted transfers or even DoS. The second is Secure API this comes with the Dependency check that had been ran with all of the possible vulnerabilities. This goes to show that an updated API is going to be needed and could be a concern in the future due to some of the vulnerabilities being caused from a lack of support. Cryptography, I believe, is the highest risk thus far because of the lack of obscurity or protection with the customers usernames and password. Client/Server ties into the same issue of cryptography because the clients username and password are used in the HTTP address allowing easy traversal through the server itself if those two pieces are leaked. Code Quality all around is okay, it looks to be unfinished in a lot of ways, customer’s aren’t able to make a withdraw request, or any other function other than deposit.

**3. Manual Review**

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

I have briefly gone over the areas of security and specifically where to find them/how to mitigate in the portion above in **2. Areas of Security** and in **5.Mitigation Plan.**

**Customer.java –** input validation. Int a is never parsed and could still have a character passed in the input even though it is initialized with the int value.

Incomplete code – there is no other function available to the user other than deposit. Artemis Financial should allow withdraws, stock transfers, insurance to align with their goals.

**DocData.java –** Users’ name and password are not encrypted in anyway when appended to the directory.

**CRUD.java** – there appears to be an extra chunk of code that assigns content to content from both content and content 2 to content. Then assigning it to another variable of content1 and content2. Very confusing why there is that extra step of adding it to content if it is being assigned to content1 when that is never returned.

**4. Static Testing**

The following images are recorded Maven Dependency checks, they list the potential vulnerabilities that the current program faces. Many of the different vulnerabilities come from Spring and Tomcat, both of which are known vulnerabilities and could be suppressed.

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

****

However, even with most of these dependencies are able to be suppressed, some could still pose a threat to security specifically through the HTTP requests.

A close-up of a message

AI-generated content may be incorrect.

Above is an example of one of dependency that is able to be suppressed but should still be considered due to it being a known CISA vulnerability (as denoted in the red text). There were a total of three separate CVE’s that came up: CVE-2022-22965 a TomCat exploit that deal could result in remote code execution, however if it is deployed via Sprin Boot its safe. It looks as though yours is going to be executed via Spring but the awareness needs to be there, CVE-2020-1938 a TomCat web application exploit where the hacker could use or pull information from the website, and CVE2 023-44487 this is an HTTP exploit that could lead to a DOS as I had mentioned prior with the clients username and password.

**5. Mitigation Plan**

With the three CVE’s I had mentioned just above the best way to avoid two of them would be through the use of Spring instead of Tomcat. That will make sure those vulnerabilities won’t be an issue. The third CVE 2023-44487 can be prevented by ensuring the clients information is obscured in the HTTP address, versus appending it to the end of the directory like how Artemis Financial currently has it set up. Having the Username/Password appear as “ \*\*\*\* ” or even go through a key like SHA-256 key encryption to ensure some type of safety when navigating to or from the customer personal profiles will help mitigate any type of leaks. The next part of the mitigation plan is with the code itself, starting with the customer.java class there is no check for the input. This could lead to improper variable being entered into their account. If the checks aren’t made from the inputs then a hacker would be able to use an injection type attack to either withdraw or send money to a separate account (with the use of the http issue earlier) with the type of input that they enter. Having a Try-Catch statement set up in order to parse for integers only would be a good first step, but also black listing known characters like ‘%’ or even ‘–‘, could lead to safe code practices. The next issue resides in the DocData.java class, I have mentioned multiple times that the user’s name and password should not be included or at least obscured in the root directory. Currently, “ "jdbc:mysql://localhost:3306/test","root","root")”, is how the users name and password will be going into the HTTP request, with their name and password replacing root. This makes it exceedingly easy for a hacker to obtain that personal information to either then login to that account or even transfer funds from that account. Obscuring the names/passwords will prevent that information from leaking. Another issue with DocData.java is that it looks unfinished.

References

(2000) *Guidance on electronic financial services and consumer ...*, *FEDERAL FINANCIAL INSTITUTIONS EXAMINATION COUNCIL GUIDANCE ON ELECTRONIC FINANCIAL SERVICES AND CONSUMER COMPLIANCE*. <https://www.ffiec.gov/pdf/efs.pdf>

Manico, J. and Detlefsen, A. (2014) *Iron-clad java: Building secure web applications*. O’Reilly. <https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/?sso_link=yes&sso_link_from=SNHU>