****

# Artemis Financial Vulnerability Assessment Report

Table of Contents

[Document Revision History 3](#_Toc32574607)

[Client 3](#_Toc32574608)

[Instructions 3](#_Toc32574609)

[Developer 4](#_Toc32574610)

[1. Interpreting Client Needs 4](#_Toc32574611)

[2. Areas of Security 4](#_Toc32574612)

[3. Manual Review 4](#_Toc32574613)

[4. Static Testing 5](#_Toc32574614)

[5. Mitigation Plan 5](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **[Date]** | **[Your name]** |  |

## Client



## Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In the report, identify your findings of security vulnerabilities and provide recommendations for 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 choose to include images or supporting materials. If you include them, make certain to insert them in all 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

Michael Brooks

## Interpreting Client Needs

## The company, as well as any company in the financing industry, should value security over everything. We need to make sure our communication and transactions, especially transactions between users, are secure. If these interactions are not secure, Artemis Financial, and Global Rain as well, will loose all credibility. We must also work within federal guidelines to make sure we are not in violation of any statutes or mandates regarding interactions such as communication, transaction, or verification. While following governmental guidelines, we must take steps to secure data stored in these systems, as well as data exchanged between clients. Packages of data in transit between the client and server are particularly vulnerable to cyber-attacks. Additionally, user input should be validated at several stages before it is introduced to the system to prevent a variety of attacks. While undertaking the modernization of Artemis Financial’s systems, we must consider how web applications continue to grow, and evolve. Perhaps, the implementation of AI, or biometric security. As we modernize their systems we should consider including new, but established, elements of computer science.

## Areas of Security

User input needs to be secure, so malicious users cannot use any prompts to attack or damage the system. Next, we need to ensure the API is functional, and that it is protected from cyber-attacks, as it serves as the link between the user interface and the server. Data packages sent or received from the API and server should be encrypted such that if they are intercepted they cannot be deciphered. Again, the connection from the client to the server should be secured and protected, but if that protection is overcome, or other wise rendered inoperable, encrypting the data should keep system information safe. Finally, the bulk of system data should be sectioned off to itself on the backend of the web application, so that it is unreachable to an external user, likely someone who is not connected by the company domain.

## Manual Review

After manually reviewing the project source code, I noticed a number of things. First, when trying to read documents from the DocData class, I was pleased to see that you were required to provide the database name, as well as a valid username, password pair. Avoiding hard coded parameters is a good way to increase security. While this is a step in the right direction, I did not see anything in any of the classes to verify user input. This is a potential hazard, and could lead to a breach in security. I also noticed that when using the showInfo function from the customer class, it does not require any form of user verification. In the interest of security, I would think this function should require some form of verification, even if that verification is redundant, or otherwise incontinent for the user. It may be seen as infuriating, and or, useless, but this way, if someone managed to access the system without a proper set of credentials, this might keep them from accessing sensitive information. Additionally, while this may just be a nitpick, I noticed the deposit function in the customer class does not verify that the deposited amount is positive. While the function would work in the case the user made a withdraw, and the value for a was set to a negative number, it wouldn’t be a deposit at that point. In the interest of properly naming functions, the function should either validate input data, or we should change the name of the function to transaction instead.

## Static Testing

After preforming static testing, a number of known vulnerabilities were found. I have listed them below, long with possible solutions for these vulnerabilities.

* [CVE-2022-22965](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-22965) This vulnerability, found in the VMware Spring Framework product, may allow for remote code execution via data binding. This can be resolved by deploying the application as a spring boot executable jar. However, there may still be ways to exploit this vulnerability.
* [CVE-2020-1938](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-1938) This vulnerability, found in the Apache Tomcat product, uses AJP connections, which tomcat gives higher levels of trust, to return arbitrary files, and under the right conditions, execute remote code. This vulnerability can be mitigated by upgrading to Apache Tomcat 9.0.31, 8.5.51 or 7.0.100 or later, though some configuration may need to be adjusted.

Note: these vulnerabilities came up in the dependency report multiple times for the same product.

## Mitigation Plan

Our first step to the mitigation plan should be to address known vulnerabilities, once these vulnerabilities have been addressed, we should consult the dependency check report to get a better idea of potential vulnerabilities, and how we can try to defend or remove them. Once we have delt with the vulnerabilities plaguing this application, I recommend adding input validation, user verification when retrieving data, and package protection to the top of our priority list. These additions need to be caried out so that they do not add new vulnerabilities.