****

# 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 4](#_Toc32574614)

[5. Mitigation Plan 4](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **3/22/2023** | **Janai Cano** | **First dependency check** |

## Client



## 

## Developer

Janai Cano

## Interpreting Client Needs

Artemis Financial wants to modernize their operations and have reached out to Global Rain to create a custom software with current and effective software security. I will be looking to identify any software security vulnerabilities in the web-based software application that we have developed for them. Since they conduct personalized financial advising, every communication between client and Artemis Financial needs to be secure. International transactions are made every day, so we must abide by international laws and standards to ensure security in bounds of the law. Personal information, especially financial information, is always under attack, so we must make sure no account information is leaked. Names of clients should also be obscured or encrypted to make sure if there ever is a breach, the data will be anonymous and cannot be linked to specific clients. By modernizing Artemis Financial’s web app, we must be careful in using open-source libraries. We should constantly test for any discrepancies or security issues with any libraries we use, and again, ensure they are not linked to or can access client account information.

## Areas of Security

Users must all have unique login identifiers to access account information. Two-factor or multiple factor authentication can be provided upon request if necessary for each account, or certain types of accounts. Between the web application, client local servers, and user devices, API connections must be secure. Any access point should be secured and handled accordingly. We also should encrypt any information that is stored server-side, as well as encrypting any sensitive information that is transmitted. This all starts with secure coding practices. At Global Rain, we pride ourselves on developing secure, personalized software, so let’s make sure we are developing a RESTful application, using OOP principles, including encapsulation. Any information that should not be visible to users or clients should be coded accordingly.

## Manual Review

In the customer.java file, the account\_balance needs to be private. showInfo() function should have a secure access point or some sort of input validation to show information. The entire customer class needs input validation to access this customer class. The ShowInfo and deposit function can be made private, correct?

In DocData.java file, SQL injection could be used to access ID. Please secure this connection. and do not use user input to validate or authenticate.

Using a RESTful API is great, and including CRUD is essential, and we have that here, so good work.

## Static Testing

Conducting a dependency check, I found 111 vulnerabilities, with 14 vulnerable dependencies being used in our code. The dependency check file will be attached to this report, but here are some of the issues listed:

1. tomcat-embed-websocket-9.0.30.jar : CVE-2020-1938. This is the Core Tomcat implementation dependency. We should update to tomcat 9.0.31 to make sure unauthorized users do not have access to files stored in the web application. Updating should fix this issue.
2. spring-webmvc-5.2.3.RELEASE.jar: CVE-2020-22965. This is the Spring Web MVC dependency. This may render our web application vulnerable to remote code execution (RCE). Applying an update to this dependency will fix this issue.
3. spring-web-5.2.3.RELEASE.jar: CVE-2016-1000027. Again, outdated Pivotal Spring Framework through 5.3.16 has a potential for RCE. We need to update this dependency.
4. spring-expression-5.2.3.RELEASE.jar: CVE-2020-22965. Same as #2 above, we need to update this dependency.
5. spring-core-5.2.3.RELEASE.jar: CVE-2020-22965. Same as #2 and #4. We need to update this dependency.
6. spring-boot-starter-web-2.2.4: CVE-2020-27772. This vulnerability allows unauthorized users to temporarily hijack directories. This is an outdated version of this dependency, we need to update to newest version.
7. snakeyaml-1.25.jar: CVE-2022-1471. SnakeYaml’s Constructor class does not restrict deserialization, which can lead to RCE. We should use SnakeYaml’s SafeConstructor to restrict deserialization.
8. Jackson-databind-2.10.2.jar: CVE-2020-25649. In FasterXML Jackson Databind, the entity expansion was not secured properly. This allows XML external entity attacks, which can lead to data leaks. We should suppress this dependency.
9. bcprov-jdk15on-1.46.jar: CVE-2016-1000338. This is a Bouncy Castle JCE version that does not validate ASN, which can lead to injection of “invisible” data into our database. We should update to a newer version of this dependency.

Graphical user interface, text, application, email

Description automatically generated

## Mitigation Plan

Most of the vulnerabilities found were because of using outdated versions of these dependencies. As listed above, the solutions are mostly updating to the newest versions of these dependencies. We should always make sure we are using the most current version of all dependencies and constantly check for updates to all that we are using. In doing so, we can make sure that known vulnerabilities are avoided and data stays secure.