****

# 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 5](#_Toc32574613)

[4. Static Testing 6](#_Toc32574614)

[5. Mitigation Plan 7](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/13/2023** | **Lane Berrevoets** | **Security documentation and mitigation plan added. Manually reviewed code. Performed Spring Boot Maven check utilizing version 8.4.0.** |

## 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

Lane Berrevoets

## Interpreting Client Needs

1. What is the value of secure communications to the company?
   1. Artemis Financial provides individualized financial plans for their customers. In order to maintain credibility with their customers and ensure that they maintain and grow their customer base, the value of secure communications to Artemis Financial must be high. A single slip has a high likelihood of losing customers to more secure financial institutions.
2. Does the company make any international transactions?
   1. Based on the prompt, it is unknown whether Artemis Financial makes international transactions. Assuming that they may, we should take international transaction restrictions into account when developing for them.
3. Are there governmental restrictions about secure communications to consider?
   1. Yes. That said, based on the prompt, it is unknown which country or countries and or states Artemis Financial operates, and therefore, specifics cannot be given. Recommend discussing further with Artemis Financial to ascertain the regions that they operate in order to ensure that all local rules and regulations are being followed.
4. What external threats might be present now and in the immediate future?
   1. Current and projected external threats will consist of SQL injection vulnerabilities, data breaches, denial-of-service attacks, to name a few. As libraries within the program update, external attacks may come in the form of zero-day attacks, when an issue is known, but no fix has been implemented.
5. What are the modernization requirements that you must consider?
   1. Evaluation of the in-use libraries within the program and any changing web application technologies must be employed. Using outdated libraries may lead to a security risk. Ensuring that the program stays up to date is vital to ensure vulnerabilities are maintained at a minimum.

## Areas of Security

The following areas of security apply to Artemis Financials’ web application:

Input Validation:

Artemis Financials’ web application should perform input validation to prevent injection-type attacks on the web application.

Application Programming Interface (API):

Artemis Financial will utilize a RESTful web API. However, we should still consider the use of input validation, access control, and encryption to minimize the likelihood of outside attacks to the API. The application Artemis Financial uses should be checked to ensure it is not providing attack opportunity.

Cryptography:

The web application should use encryption appropriately when handling customer information and data. All transactions should be encrypted to prevent attackers from gaining access to customer information, customer account information, or transaction history.

Client/Server:

Appropriate communication protocols should be followed in order to protect data during transmission. We should verify all data from external sources that may come into contact with Artemis Financials’ web application.

Code Error/Quality:

Code used in the web application should be checked for quality and efficiency. Additionally, we should verify that the code operates without error. Small errors within the code may lead to very large issues within customer accounts.

Encapsulation:

Due to the inherent nature of Java code, the program will be written with encapsulation in mind. Separations of classes, clear delineation of methods and attributes will allow future developers to make corrections and review the code with ease.

Based on the previous, it has been determined that the following areas of security will play a vital role in limiting the risk to Artemis Financials’ web application:

* + - Input validation
    - API – This will be dependent on which API the web application uses, and whether it is appropriate for the application.
    - Cryptography
    - Client/Server
    - Code Error/Quality

## Manual Review

Based on the provided code, the following vulnerabilities have been identified during the manual review:

Customer.java contains a public class customer with an open int account\_balance. Creating account\_balance as a private field, then writing to the field using setters and getters will assist in further security of the individuals account balance. If an outside attack were to occur, anyone would have the ability to retrieve account balance information.

DocData.java contains a read\_document method that takes a string key, and string value without first reviewing the input prior to insertion into the SQL query. This may lead to SQL injection vulnerabilities.

DocData.java contains a read\_document method that tries to establish a connection con=DriverManager.getConnection without attempting to establish a secure connection to the server.

Pom.xml lists the version of Spring Boot as version 2.2.4, which was released in early 2020. We should consider verifying whether known issues exist with that version, and whether Spring Boot should be updated to a more recent version.

## Static Testing

|  |  |
| --- | --- |
| **Dependency** | **Severity** |
| Bcprov-jdk15on-1.46 | 18 vulnerabilities classified as High severity |
| Hibernate-validator-6.0.18.Final | 1 vulnerability classified as Medium severity |
| Jackson-databind-2.10.2 | 6 vulnerabilities classified as High severity |
| Log4j-api-2.12.1 | 1 vulnerability classified as Low severity |
| Logback-core-1.2.3 | 1 vulnerability classified as Medium severity |
| Snakeyaml-1.25 | 8 vulnerabilities classified as Critical severity |
| Spring-boot-2.2.4.release | 3 vulnerabilities classified as Critical severity |
| Spring-boot-starter-web-2.2.4.Release | 3 vulnerabilities classified as Critical severity |
| Spring-core-5.2.3.Release | 11 vulnerabilities classified as Critical severity |
| Spring-web-5.2.3.Release | 12 vulnerabilities classified as Critical severity |
| Spring-webmvc-5.2.3.Release | 11 vulnerabilities classified as Critical severity |
| Tomcat-embeded-core-9.0.30 | 21 vulnerabilities classified as Critical severity |
| Tomcat-embeded-websocket-9.0.30 | 22 vulnerabilities classified as Critical severity |

The dependencies found represent a wide range of vulnerabilities from Low, Medium, High, and Critical severity classifications. The following lists the severity classifications, a brief description of the dependency, as well as recommended solutions listed in the National Vulnerability Database (NVD) and how the dependency was initially identified.

The following dependencies were listed as Critical Severity:

**SnakeYaml** – The constructor class does not restrict the type that can be instantiated during deserialization. Specifically, an attacker can deserialize content, and remote execute code. Recommended corrective action is to update SnakeYaml to version 2.0 or following.

**Spring Boot/Core/Web–** If Spring MVC is used together with a reverse proxy cache causing a potential for DDoS type attacks. There are no given fixes, however, it is assumed that updating the library to a version that is not listed, the issue would be corrected.

* + - High susceptibility to security bypasses, recommended correction is to update the library to a version that does not have the listed vulnerability.
    - Spring MVC or WebFlux applications running on JDK 9+ represent a vulnerability to remote code execution. Specifically, the exploit requires running Tamcat as a WAR deployment. If this is not the case, there is not an exploitable vulnerability.

**Tomcat core/websocket** – Tomcat treats Apache JServ Protocol as having higher trust than other HTTP connections, allowing for exploitation. Changes to the configurations of Tomcat may be required to prevent such exploitations.

The following dependencies were listed as High Severity:

**Bouncy Castle** – Bouncy Castle versions 1.55 and earlier do not fully validate encoded signatures during verification. Specifically, it is possible to inject elements within the sequence and still have validation occur. Though not specifically stated, updating the library to a newer version without the listed dependency should correct said issue.

**FasterXML Jackson-databind** – DDoS attacks may occur involving JsonNode JDK serialization, resource exhaustion may occur due to lack of a check in deserializers.

Medium and Low Severity dependencies were found in Hibernate Validator, Logback Core, and Log4j. The dependencies include allowing attackers to bypass input sanitation, remote code execution, server/client type vulnerabilities. Most listed dependencies list library updates to versions in which the vulnerabilities do not exist.

## Mitigation Plan

Based on both the manual code review and the dependency check using static testing, it has been determined that the following actions should be taken to minimize the risk association to the above security vulnerabilities:

Library updates – the libraries for various dependencies should be updated to versions in which the vulnerabilities do not exist.

Code review – the web application code should be checked, potentially re-written to provide various checks to be performed, as well as additional security steps be added to the code to provide a less vulnerable web application. Additionally, the DocData class should utilize parameterized queries to prevent SQL injection type attacks. Secure data communication protocols should be used when connecting to and transmitting from the server.

For the future state of the web application, we should ensure that regular checks are made of the web application and any future dependencies should be evaluated and the associate risk should be made small.