****

# 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** | **November 11, 2023** | **Gabrielle Maitland** |  |

## Client



## Developer

Gabrielle Maitland

## Interpreting Client Needs

Secure communications is a valuable goal of the company. Because Artemis Financial offers customers solutions on their savings, retirements, investments and more, it is crucial that customers feel a sense of security when sending sensitive data back and forth through the client’s API.

Since Artemis Financial also covers customers savings accounts, they may engage in international transactions if the customer has an account in another country due to dual citizenship, or if a customer has family living elsewhere that they send/receive money from. In that case, the client should be aware of what international regulations there may be globally if they permit such transactions. For example, the European Union may have particular guidelines around how electronic banking occurs from a security standpoint that may differ from the requirements of the United States. Depending on the country of operation for Artemis Financial, there may or may not be governmental restrictions surrounding secure communication.

Now and in the future, Artemis Financial faces the threat of malicious attacks from external sources. These attacks can target any web application, and because Artemis works with sensitive customer information and finances, it is even more vulnerable to being targeted. These attacks come in many forms, such as SQL injections, denial-of-service attacks, click jacking, and more.

Artemis Financial seeks to modernize their operations. This may come in the form of open source libraries or other evolving web application technologies. It is important that when using these features that developers make note of which libraries have known exploits that are susceptible to attacks. It is also key to stay afloat with updates in the cyber security world to measure which technologies are deprecated due to vulnerabilities and which are recommended due to security practices.

## Areas of Security

Input Validation: Any application that requires users to input any sort of information should always be validated to deter attacks

APIs: Since Artemis Financial is using a REST API, it is important to secure it properly.

Cryptography/Encryption: Artemis Financial is working with very sensitive information. It is recommended that they use some form of encryption for important data.

Client/Server: Information is being passed back and forth within the client’s application. Any data communicated between the client and server should be properly secured.

Code Error Handling: When an error is thrown, it is important that there are no sensitive data leaks in the error, or any opportunity for attackers to gain unwanted access through the error.

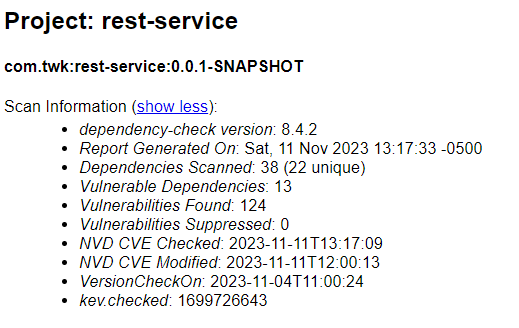
## Manual Review

After going through each class, I have found a few issues:

Customer.java: There is a public method for ShowInfo() that displays the customer’s sensitive information. There is no authentication feature implemented to ensure that it is actually the customer viewing their info.

DocData.java: There is a direct line of connection to a SQL database. This is dangerous because it is directly coded into the class and it has no security features to deter attackers from accessing the database.

## Static Testing



|  |  |
| --- | --- |
| **Vulnerability** | **Description** |
| bcprov-jdk15on-1.46.jar | The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.5 to JDK 1.7. |
| hibernate-validator-6.0.18.Final.jar | Hibernate's Bean Validation (JSR-380) reference implementation. |
| jackson-databind-2.10.2.jar | General data-binding functionality for Jackson: works on core streaming API |
| log4j-api-2.12.1.jar | The Apache Log4j API |
| logback-core-1.2.3.jar | logback-core module |
| snakeyaml-1.25.jar | YAML 1.1 parser and emitter for Java |
| spring-boot-2.2.4.RELEASE.jar | Spring Boot |
| spring-boot-starter-web-2.2.4.RELEASE.jar | Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container |
| spring-core-5.2.3.RELEASE.jar | Spring Core |
| spring-web-5.2.3.RELEASE.jar | Spring Web |
| spring-webmvc-5.2.3.RELEASE.jar | Spring Web MVC |
| tomcat-embed-core-9.0.30.jar | Core Tomcat implementation |
| tomcat-embed-websocket-9.0.30.jar | Core Tomcat implementation |

## Mitigation Plan

After reviewing the Dependency Report, we can solve many of these issues by keeping dependencies up to date. Bouncy Castle is the most critical vulnerability listed in the report due to information leaks and it is simply because it is out of date. SnakeYaml is also listed as a critical vulnerability, susceptible to remote code execution because it is also out of date. Aside from that, it is recommended that we fix each vulnerability in the code aforementioned (customer.java, docdata.java) in order to mitigate attacks. We can implement an additional layer of user authentication before showing a customer their bank information. We can also explore different options to access the SQL database without hardcoding it, such as with query parameterization or role based access.

Resources:

Detlefsen, A., & Manico, J. (2014). *Iron-Clad java: Building Secure Web Applications*. McGraw-Hill Education.