****

# 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** | **9/14/2022** | **Jason Veno** |  |

## 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

Jason Veno

## Interpreting Client Needs

Artemis Financial needs an application that can manage different types of secure accounts for many users. The nature of this application includes storing extremely sensitive information regarding the owners of the accounts, so security is the absolute highest priority in this scenario. This is also because there could be numerous different types of transactions as well, some of which would be international, which leads to further possible types of attacks. This is because there is so much potential gain for a hacker, as they could potentially break through the system and acquire access to the funds within said transactions. For example, an account owner could invest in an international company, only for a hacker to intercept the investment and claim it for themselves. For all the reasons listed above, extra care must be taken for the highest possible level of security. The client already made the wise move of using a RESTful API, which supports Transport Layer Security encryption, which monitors and secures the data involved in all interactions. However, there are so many possible forms of attacks against this application, each type of transaction involving the transfer of data should be uniquely and individually assessed for the best possible way to secure it.

## Areas of Security

**Input Validation**: Users should be limited in how they can interact with the system, and the system should both detect and deter bad input.

**API**: They already have a RESTful API, but additional attention to detail is needed to ensure that it properly secures its many interactions, as the risk is so high in this case.

**Cryptography**: The RESTful API uses Transport Layer Security encryption, but it should be tested to ensure that it encrypts every type of interaction, including investments, insurance, savings, etc.

## Manual Review

Firstly, the **request parameters are not validated** in the pom.xml file. This will help to ensure that proper input validation is used. Specifically, users will be much more restricted in regards to what they can include in their input, which greatly increases the security of the system.

Also, the application uses HTTP, whereas **HTTPS would be the better option**. This is because HTTPS uses Secure Sockets Layer (SSL), which keeps the internet connection secure and protects any transferring data.

In CRUDController.java, the **business name is part of the request parameter**, which means it is part of the URL. Though the business name alone is not sensitive data, any information an attacker has can be used to breach the system.

## Static Testing

**bcprov-jdk15on-1.46.jar**

**Description:**

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.

**Recommended Solution:** Update Bouncycastle to newest version (1.70 as of 9/14/2022)

**spring-boot-2.2.4.RELEASE.jar**

**Description:**

Spring Boot

**Recommended Solution:** Update Spring Boot to newest version (2.7.3 as of 9/14/2022)

**logback-core-1.2.3.jar**

**Description:**

logback-core module

**Recommended Solution:** Update Logback Core Module to newest version (1.4.0 as of 9/14/2022)

**log4j-api-2.12.1.jar**

**Description:**

The Apache Log4j API

**Recommended Solution:** Update Apache API to newest version (1.18.0 as of 9/14/2022)

**snakeyaml-1.25.jar**

**Description:**

YAML 1.1 parser and emitter for Java

**Recommended Solution:** Update SnakeYAML to newest version (1.32 as of 9/14/2022)

**jackson-databind-2.10.2.jar**

**Description:**

General data-binding functionality for Jackson: works on core streaming API

**Recommended Solution:** Update Jackson Datamind to newest version (2.13.4 as of 9/14/2022)

**tomcat-embed-core-9.0.30.jar**

**Description:**

Core Tomcat implementation

**Recommended Solution:** Update Tomcat Embed Core to newest version (10.1.0-M17 as of 9/14/2022)

**spring-boot-starter-validation-2.2.4.RELEASE.jar**

**Description:**

Starter for using Java Bean Validation with Hibernate

Validator

**Recommended Solution:** Update Spring Boot Starter to newest version (2.7.3 as of 9/14/2022)

**hibernate-validator-6.0.18.Final.jar**

**Description:**

Hibernate's Bean Validation (JSR-380) reference implementation.

**Recommended Solution:** Update Hibernate Validate Engine to newest version (8-final as of 9/14/2022)

**spring-core-5.2.3.RELEASE.jar**

**Description:**

Spring Core

**Recommended Solution:** Update Spring Core to newest version (5.3.2 as of 9/14/2022)

## Mitigation Plan

Firstly, each of the dependencies listed in the Dependency Check Report should be updated to the newest possible version, to ensure that an attacker cannot take advantage of any of them. Next, each of the three vulnerabilities found in the manual review need to be individually fixed. The request parameters need to be validated, which can be done by adding @RequestParam and @Validated to the pom.xml. HTTPS should be used instead of HTTP. Any sensitive information (in this case the business name) should be removed from the request parameter. Once all of these changes are made, an additional review of the application should be made to ensure that the changes were made correctly, and that they didn’t create any new vulnerabilities.