****

# 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** | **01/28/24** | **Joey Paul E. Haynes** |  |

## Client



## Developer

Joey Paul E. Haynes

## Interpreting Client Needs

Upon interpreting the security concerns and client needs of Artemis Financial, it is clear that secure communications are crucial to the success of the organization’s software offerings and valued reputation. Secure communications pave the way to protecting sensitive customer data and maintaining their trust, thus strengthening the reputation of the company while ensuring local and international regulatory compliance. Given that the client likely deals in international transactions, this latter point is crucial to consider, as many nations/regions have specific data protection laws (such as the GDPR in Europe). This is in addition to other standard governmental regulations, such as those to reduce credit card fraud (PCI DSS) and guarantee transparent data-sharing practices (GLBA).

Pertinent external threats for Artemis Financial’s RESTful web API include, broadly, potential data breaches, injection attacks, and unauthorized access through, among other possible sources, weakness in authentication mechanisms. These threats continue to evolve as technology advances, thus underscoring the importance of the company implementing a routine security protocol for countering future threats, such as sophisticated malware or even AI-driven attacks. As attacks continue to evolve and become more sophisticated, so too must the organization’s security standards and procedures.

## Areas of Security

With respect to the given scenario, where we seek the secure implementation of a RESTful web application programming interface (API), the areas I expect need be particularly emphasized are input validation, APIs, cryptography, and code quality. This follows from the first level of architecture review in the VAPFD document, with input validation, APIs, and cryptography at the beginning of the process flow and code quality in the terminal stages of the review. Areas like client/server, code error, and encapsulation, while still important in the broader context of security, may not directly impact the RESTful API functionality in this application specifically, and thus are not included in the scope of this current vulnerability assessment report.

Here, to further justify these areas of focus, I elaborate on why the four out of seven criteria were chosen. Input validation, as the first step, acts as a first line of defense against common vulnerabilities, including SQL injection and other maliciously formatted inputs. Because we are implementing a RESTful web API, secure API interactions are the next logical step in the process flow; this ensures that sensitive data is securely accessed and handled. Moreover, because we are dealing with sensitive financial and customer-identity data, robust cryptography measures are crucial to implement in the first iteration of the API to protect from interception or tampering in the event that data is compromised. Finally, code quality, through secure coding patterns, ensures that the initial implementation of this API prevents initial security flaws and ensures the application’s future maintainability. This provides a holistic starting point for narrowing down the relevant security concerns for a RESTful web API in the financial sector.

## Manual Review

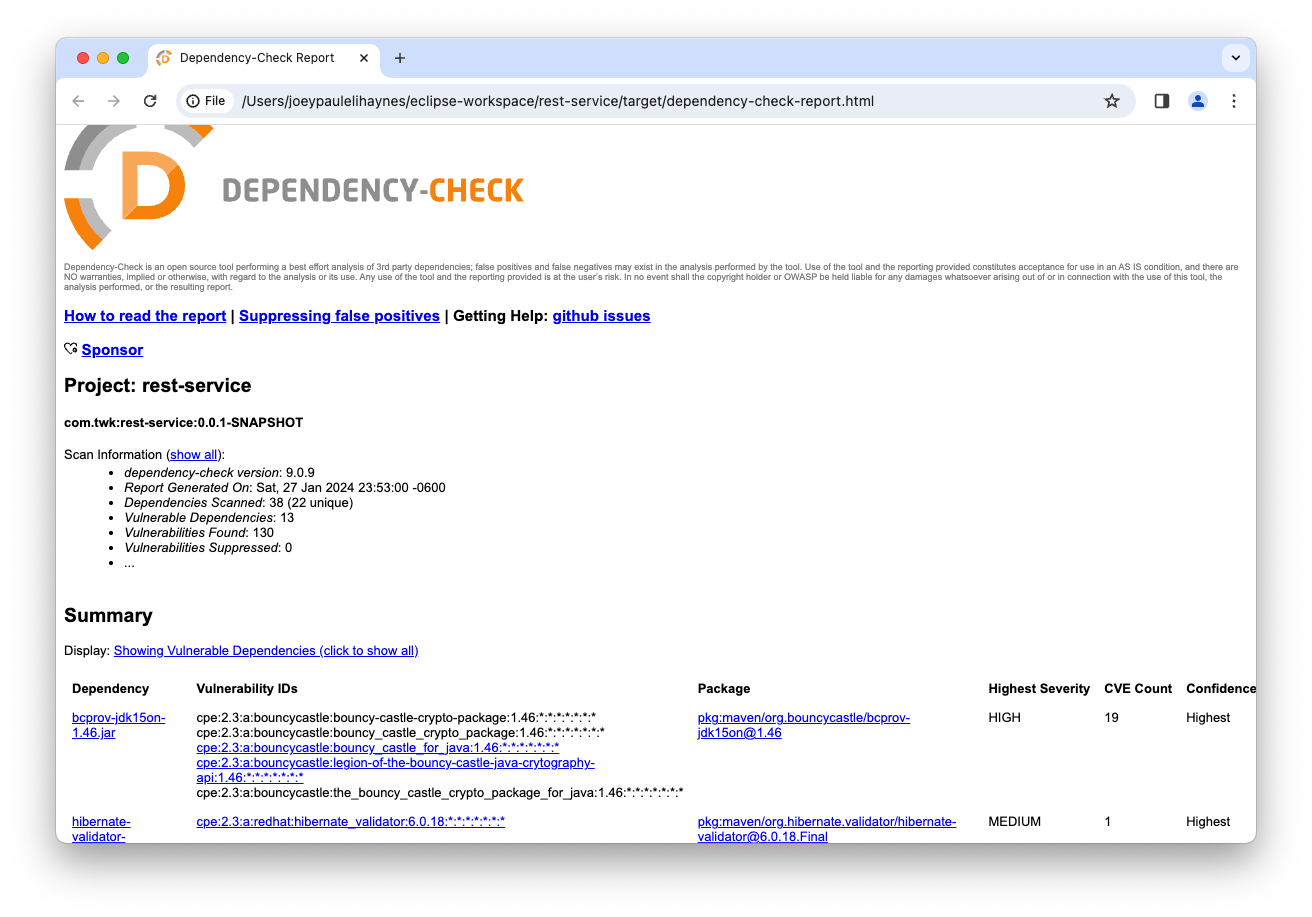
During my manual review of the codebase, I uncovered the following potential security vulnerabilities:

1. **Hardcoded credentials in DocData.java**. On lines 26 and 27, this file establishes a database connection using hardcoded data, which is a security risk as it openly exposes database access credentials.
2. **Insufficient exception handling in DocData.java**. The *read\_document* method catches an exception (SQLException), but does not handle it securely.
3. **Insecure object design for the customer object in Customer.java**. This class exposes *account\_number* and *account\_balance* with limited encapsulation or authentication measures. Likewise, the *showInfo* method returns customer information and the *deposit* method modifies account balance without any authentication check.
4. **No input validation in controllers (CRUDContoller and GreetingController)**. These files do not perform any validation of input from request parameters, which could leave the API vulnerable to injection attacks.
5. **Overall lack of authentication and authorization structure**. Neither of the controllers (CRUDController, GreetingController) implements a systematic authentication/authorization scheme when dealing with the user’s personal and financial data. Moreover, many class fields throughout the codebase are public, which could lead to unauthorized access or modification from other classes within the package.

## Static Testing

Through the implementation of static testing protocols, via a dependency check, I uncovered a total of 13 vulnerable dependencies, with 130 total vulnerabilities. The results of this most recent dependency check are shown below:

***Figure 1. Maven Dependency Check Report***



***Figure 2. Dependency Summary (includes name and vulnerability codes)***

A screenshot of a computer

Description automatically generated

Additionally, a brief description of each dependency, written by referencing known information from the NVD and CVE databases, is shown below:

***Table 1. REST API Vulnerabilities with Description***

|  |  |
| --- | --- |
| **Name** | **Description** |
| bcprov-jdk15on-1.46.jar | This is a Java library for cryptographic API functionalities, which may have had security issues due to flaws in cryptographic algorithm implementation in an earlier version. |
| hibernate-validator-6.0.18.Final.jar | Hibernate-validator is used for object validation in Java. This version may contain vulnerabilities related to input validation and error handling. |
| jackson-databind-2.10.2.jar | Jackson Databind is a library for data parsing/serialization. Known vulnerabilities are related to flaws in deserialization. |
| log4j-api-2.12.1.jar | Apache Log4j is a logging framework, which may have been susceptible to remote code execution. |
| logback-core-1.2.3.jar | Logback is a logging library where data leakage and unsafe log handling are reported. |
| snakeyaml-1.25.jar | SnakeYAML is a YAML processor for the Java Virtual Machine (JVM), which may have had potential DDoS exploits. |
| spring-boot-2.2.4.RELEASE.jar | Spring Boot is the main framework of this app, which may have exposed sensitive information through remote code execution attacks. |
| spring-boot-starter-web-2.2.4.RELEASE.jar | This is a module of the Spring Framework for building web applications, and may have improperly handled user inputs. |
| spring-core-5.2.3.RELEASE.jar | The core of the Spring Framework for building web applications; this dependency may have remote code execution and binding issues. |
| spring-web-5.2.3.RELEASE.jar | A component of the Spring Framework’s support for web applications, which may have inadequate exception handling and insecure defaults. |
| spring-webmvc-5.2.3.RELEASE.jar | Spring Web MVC is another part of the Spring Framework that deals with web applications; this dependency may have security concerns related to path traversal and validation of request parameters. |
| tomcat-embed-core-9.0.30.jar | The core dependency of the Tomcat server, which may have had issues related to session management and privilege escalation. |
| tomcat-embed-websocket-9.0.30.jar | This library is used to enable WebSocket communication in Tomcat. Issues include possible DDoS attacks and insecure message handling. |

As noted in an earlier dependency check, an element of this type of test to take note of is the possible presence of false positives. Dependency-Check-Maven most often reports false positives on CPE values, and these can be suppressed by generating an XML file for use in future scans. Filtering false positives in this way can reduce time spent considering irrelevant issues and enhance the accuracy of the report.

## Mitigation Plan

In light of the overall business needs, the manual review, and static testing, a path forward could include a combination of dependency updates, security patches in the codebase, policy design/enforcement, and regular review and audit of dependencies. Newer dependency versions generally contain patches for known vulnerabilities; thus, keeping versions up to date can reduce the record of known vulnerabilities in a subsequent dependency check. In the event that major security concerns remain after dependency updates, techniques like whitelisting/blacklisting and code encapsulation methods like execute-around can be implemented in the codebase to mitigate additional security concerns, including those outlined above related to inadequate input validation and exception handling. At this point, further review and risk assessment are needed; an organizational security policy can help, as this establishes clear criteria for acceptable risk levels and protocols for continued updates and security reviews.

To mitigate the aforementioned security concerns, the following action plan may be implemented:

1. Ensure dependencies are up to date; update and suppress false positives as needed
2. Remove hardcoded credentials, such as the database connection in DocData.java
3. Improve exception handling to avoid exposing sensitive error details in DocData.java
4. Ensure greater input validation in the class controllers to avoid injections/other malicious inputs
5. Refactor the API’s *customer* object to encapsulate information more securely.
6. Implement a robust authentication and authorization system throughout the package such that only authorized users/account owners can access sensitive financial information.
7. Continue with regular security audits, code reviews, and developer education; this will help the organization stay on top of new and emerging threats and ensure adherence to a standard security policy.