****

# 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 5](#_Toc32574614)

[5. Mitigation Plan 6](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **3/24/2024** | **Joshua Perry** |  |

## 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

Joshua Perry

## Interpreting Client Needs

Artemis Financials’ imperative to modernize its operations with a strong emphasis on security underscores the need for a comprehensive approach to safeguard its web-based applications. The protection of sensitive financial data, adherence to international and domestic regulations, and resilience against external threats are paramount. The firm’s engagement in international financial transactions necessitates a robust framework for secure communications, mindful of the legal complexities of cross-border data handling. As such, our assessment will focus on identifying vulnerabilities that could compromise data integrity, confidentiality, and availability, ensuring that Artemis Financials’ digital transformation aligns with its security-first mission.

## Areas of Security

Based on the analysis of Artemis Financial's web application, the following areas of security are identified as critical and should be addressed to ensure the robustness and resilience of the application:

**Input Validation**: The application must enforce strict input validation, especially in CRUD operations. Currently, there is no clear evidence of input validation in CRUDController, which can open doors to injection attacks or other forms of input-based vulnerabilities.

**Authentication and Authorization**: The code does not detail how users are authenticated and what authorization checks are in place, especially in sensitive operations like reading and modifying customer data. Ensuring that only authorized users can perform specific actions is crucial to protect user data.

**Data Protection**: Sensitive data such as customer account numbers and balances are handled within the application. It is imperative to ensure that this data is encrypted, both at rest and in transit, and that access to it is controlled and monitored.

**Error Handling**: The DocData class indicates a lack of robust error handling, particularly in database operations. Proper error handling mechanisms are essential to prevent the leakage of sensitive information and to ensure the application's stability and reliability.

**Secure Database Access**: The read\_document method in DocData reveals hard-coded credentials and a lack of secure database access practices. It's vital to employ secure methods for database connectivity, avoiding hard-coded credentials and ensuring that database queries are resistant to SQL injection.

**Code Quality**: Ensuring that the code follows secure coding practices can mitigate a wide array of potential vulnerabilities. The initial review suggest there might be room for enforcing better coding standards that emphasize security, such as avoiding hard-coded credentials and ensuring encapsulation is properly implemented.

**API Security**: The application's use of APIs (particularly in CRUDController) must ensure secure API interactions. This includes validating inputs, sanitizing outputs, and employing proper authentication and authorization checks for API access.

## Manual Review

Upon a detailed examination of the code, several potential vulnerabilities and areas of concern have been identified that warrant immediate attention:

**CRUD and CRUDController**: The CRUD operations seem to handle sensitive data, but there's no clear indication of input validation or sanitization. Without proper validation, the system may be vulnerable to injection attacks, especially if user input is utilized in constructing database queries or commands.

**Customer Class**: The customer class exposes account\_number and account\_balance as private but provides public methods to access and modify these values without any validation or checks. This could lead to unauthorized access or modification of sensitive financial data.

**DocData Class**: The read\_document method attempts to establish a database connection using hard-coded credentials, which is a severe security risk. Hard-coded credentials can lead to credential leakage and should be replaced with secure credential storage and retrieval mechanisms.

The method also lacks proper exception handling, potentially exposing sensitive information about the database structure or server configuration in case of an error. The commented-out Class.forName("com.mysql.jdbc.Driver"); suggests potential use of outdated or deprecated methods for database connectivity.

## Static Testing

The static analysis conducted using the Dependency-Check tool revealed several security vulnerabilities across various dependencies in the rest-service project. The tool scanned 38 dependencies (22 unique) and identified 13 vulnerable dependencies, with a total of 131 vulnerabilities found. Here are some highlights from the static analysis report:

**Bouncy Castle Crypto (bcprov-jdk15on-1.46.jar)**: This library was found to have multiple vulnerabilities, including issues that could potentially allow an attacker to inject extra elements into a signature sequence or exploit weak private key generation in DSA. The most severe vulnerability had a CVSSv3 base score of HIGH (7.5).

**Hibernate Validator (hibernate-validator-6.0.18.Final.jar)**: A vulnerability was identified that could allow an attacker to bypass input sanitation controls by crafting malicious error messages. This vulnerability was rated with a CVSSv2 base score of MEDIUM (5.0).

**Jackson Databind (jackson-databind-2.10.2.jar)**: Several vulnerabilities were found in this library, including one that could allow an attacker to conduct a denial of service attack through a crafted object using cyclic dependencies. Another significant vulnerability related to improper restriction of XML external entity references was also identified.

**Log4j API (log4j-api-2.12.1.jar)**: A vulnerability was discovered that could allow an SMTPS connection to be intercepted by a man-in-the-middle attack, potentially leaking log messages sent through the affected appender.

**Logback Core (logback-core-1.2.3.jar)**: This component was found to be vulnerable to a deserialization issue that could lead to a denial-of-service attack if poisoned data is sent to the logback receiver component.

**SnakeYAML (snakeyaml-1.25.jar)**: The analysis uncovered vulnerabilities in the SnakeYAML library that could lead to remote code execution or denial of service when parsing untrusted YAML files.

**Spring Boot and Related Libraries**: Several critical vulnerabilities were identified in Spring Boot and associated libraries, emphasizing the need for an urgent upgrade to secure the application further.

## Mitigation Plan

Based on the findings from the manual review and static testing report, a comprehensive mitigation plan is essential to address and rectify the identified vulnerabilities in Artemis Financials’ web-based application. This action list outlines the steps necessary to mitigate the vulnerabilities.

**Manual Review Findings and Mitigation Steps**:

**CRUD and CRUDController**: Implement input validation to ensure that only expected data formats are accepted. Sanitize user inputs to prevent injection attacks.

**Customer Class**: Ensure that sensitive data, such as account\_number and account\_balance, is adequately protected and not exposed to unauthorized users. Implement proper access controls to restrict the modification of account details to authorized users only.

**DocData Class**: Remove hard-coded credentials and replace them with a secure method of credential storage and retrieval. Implement exception handling to prevent leakage of sensitive information in error messages. Ensure that database queries are parameterized to prevent SQL injection attacks.

**Greeting and GreetingController Classes**: Review the usage context of these classes to ensure they do not inadvertently expose sensitive information.

**myDateTime Class**: Validate the implementation to ensure there are no vulnerabilities related to time manipulation.

**Static Testing Findings and Mitigation Steps**:

**Bouncy Castle Crypto (bcprov-jdk15on-1.46.jar)**: Upgrade to a newer, non-vulnerable version of the library. Verify the security fixes by reviewing the library's release notes and testing for regression.

**Hibernate Validator (hibernate-validator-6.0.18.Final.jar)**: Update the library to the latest secure version to mitigate the bypass vulnerability.

**Jackson Databind (jackson-databind-2.10.2.jar)**: Apply patches or upgrade to a version where known vulnerabilities are addressed. Ensure safe handling of data when using Jackson for processing JSON.

**Log4j API (log4j-api-2.12.1.jar) and Logback Core (logback-core-1.2.3.jar)**: Update these logging frameworks to the latest versions to resolve identified vulnerabilities. Configure the logging frameworks securely, avoiding exposure of sensitive information.

**SnakeYAML (snakeyaml-1.25.jar)**: Upgrade to a version that mitigates the identified vulnerabilities, particularly those related to remote code execution.

**Spring Boot and Related Libraries**: Update Spring Boot and its related dependencies to the latest versions to address the critical vulnerabilities found. Test the application thoroughly after updating the libraries to ensure that the updates do not introduce new issues.

**General Mitigation Steps**: Implement a process for regular updates and vulnerability scanning of dependencies. Integrate security tools and practices into the development lifecycle to identify and address vulnerabilities proactively. Conduct training sessions for the development team to enhance their awareness of secure coding practices and vulnerability mitigation strategies. By systematically addressing each identified vulnerability following this action list, Artemis Financial can significantly enhance the security and resilience of its web-based application, aligning with its commitment to modernization and security.