****

# 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** | **07/16/2023** | **Jarrod Thomisee** |  |

## Client



## Developer

Jarrod Thomisee

## Interpreting Client Needs

Artemis Financial deals with sensitive financial data. Secure communication is of utmost importance to protect the integrity of the transactions and customer's data. Since the company also handles international transactions, the application needs to comply with international cybersecurity laws and standards.

Given the potential for financial fraud, external threats can come from hackers attempting to breach the system for financial gain, identity theft, or corporate espionage. Also, as cyber threats evolve, it is essential to continually assess and update security protocols to counter potential future threats.

Modernization requirements include the use of open-source libraries, which need regular updates for security patches and evolving web application technologies, which could expose the system to vulnerabilities.

## Areas of Security

The areas of security relevant to Artemis Financial’s web application include:

* Input Validation and Representation: Ensures that only valid and safe data is accepted by the application.
* API Abuse: Checks for misuse of the application's exposed APIs.
* Security Configuration: Reviews the setup and access control rules within the application.
* Data Protection: Ensures the safe storage and transfer of sensitive data.
* Authentication: Evaluates how user identity is verified and managed.

## Manual Review

* **Insecure Object Reference**
  + File: CRUDController.java
  + Vulnerability: An instance of DocData is created but the database connection is never closed, which could potentially lead to memory leaks.
  + Solution: Implement a finally block to ensure the database connection is closed after usage.
* **Security Misconfiguration**
  + File: DocData.java
  + Vulnerability: The database connection is made with hardcoded credentials, posing a risk of credential leak.
  + Solution: Use a secure method to store database credentials, like using environment variables or a secure configuration file.
* **Sensitive Data Exposure**
  + File: customer.java
  + Vulnerability: The account\_balance attribute is directly manipulated via the deposit method. If an attacker has access, they can alter the balance of a user.
  + Solution: Encapsulate the account\_balance field to prevent direct manipulation. Provide a secure way to access this field.
* **Insecure Direct Object References**
  + File: CRUDController.java
  + Vulnerability: The CRUD method uses a business\_name directly in a query which could lead to IDOR if an attacker can guess business names.
  + Solution: Implement access controls to verify whether the current user is authorized to access the requested business data.
* **Improper Error Handling**
  + File: DocData.java
  + Vulnerability: The exception is caught but not adequately handled or logged in read\_document method.
  + Solution: Implement proper error logging and handle exceptions.
* Injection
  + File: DocData.java
  + Vulnerability: The read\_document method appears to be at risk of SQL Injection attacks as it seems like it might be using string concatenation for SQL queries with user inputs.
  + Solution: Use prepared statements to prevent SQL Injection.

## Static Testing

* **bcprov-jdk15on-1.46.jar**
  + Description: The Bouncy Castle Java library before 1.51 does not validate a point is withing the elliptic curve, which makes it easier for remote attackers to obtain private keys via a series of crafted elliptic curve Diffie Hellman (ECDH) key exchanges, aka an "invalid curve attack."
  + Recommended Solution: Upgrade to a newer version of Bouncy Castle (>=1.51).
* **hibernate-validator-6.0.18.Final.jar**
  + Description: There was an identified defect in Hibernate Validator, specifically in its 6.1.2.Final edition. A glitch in the way the message interpolation processor operates allows erroneous EL expressions to be assessed as if they were correct.
  + Recommended Solution: Upgrade to a newer version of Hibernate Validator.
* **jackson-databind-2.10.2.jar**
  + Description: FasterXML jackson-databind 2.x before 2.9.10.4 mishandles the interaction between serialization gadgets and typing, related to com.caucho.config.types.ResourceRef.
  + Recommended Solution: Upgrade to a newer version of Jackson Databind (>=2.9.10.4).
* **snakeyaml-1.25.jar**
  + Description: The Alias feature in SnakeYAML 1.18 allows entity expansion during a load operation, a related issue to CVE-2003-1564.
  + Recommended Solution: Upgrade to a newer version of SnakeYAML that has addressed this issue (>=1.26).
* **spring-core-5.2.3.RELEASE.jar, spring-web-5.2.3.RELEASE.jar, spring-webmvc-5.2.3.RELEASE.jar**
  + Description: Multiple vulnerabilities found in the Spring Framework versions before 5.2.3 allow attackers to execute arbitrary code via different vectors.
  + Recommended Solution: Upgrade to a newer version of the Spring Framework that has addressed these issues (>=5.2.3).
* **tomcat-embed-core-9.0.30.jar, tomcat-embed-websocket-9.0.30.jar**
  + Description: Multiple vulnerabilities found in Apache Tomcat versions before 9.0.31 allow attackers to execute arbitrary code via different vectors.
  + Recommended Solution: Upgrade to a newer version of Apache Tomcat that has addressed these issues (>=9.0.31).

## Mitigation Plan

Based on the manual review and static testing report, I found the following vulnerabilities:

* **Sensitive data in plaintext**
  + In the DocData.java file, the database credentials are hard-coded and in plaintext. This could lead to unauthorized access if the codebase is exposed or shared.
  + Action: Avoid hardcoding sensitive information directly in the code. Use environment variables, configuration files, secure vaults, or other secure methods to store this information.
* **Lack of input validation**
  + In the CRUDController.java file, there's no input validation for the 'business\_name' parameter, which could lead to potential injection attacks.
  + Action: Implement input validation and sanitization before processing. Use parameterized queries or prepared statements to prevent SQL injection attacks.
* **Insufficient error handling**
  + In the DocData.java file, there is a catch block for a SQLException that merely prints the stack trace. This is not good practice as it can expose sensitive system or application details to the user or an attacker.
  + Action: Implement proper exception handling strategies, ensure no sensitive information is leaked through error messages, and consider implementing a centralized error handling mechanism.
* **Insecure object exposure**
  + In the customer.java file, 'account\_balance' is declared without an access modifier, which defaults to package-private in Java, meaning it could be accessed by any class in the same package.
  + Action: Declare 'account\_balance' as private and provide a public getter method if necessary.
* **Outdated libraries with known vulnerabilities**
  + As per the dependency checker output, some of the libraries used are outdated and have known vulnerabilities.
  + Action: Update the libraries to the latest version where possible. If updating is not possible due to compatibility issues, look for alternative libraries or apply patches if available.