****

# 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** | **January 25, 2024** | **Ryan Hatch** | * **Reanalyze and format code due to basic flaws. OWASP scan reveals 13 vulnerable dependencies and 81 total vulnerabilities.** |

## 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

Ryan Hatch

## Interpreting Client Needs

**Secure Communications:**

**Value:** Essential for protecting client data and financial transactions. Ensures confidentiality and integrity of data in transit.

International Transactions: If present, consider encryption standards and compliance with international regulations.

**Governmental Restrictions:** Research and comply with laws related to data security and privacy in jurisdictions where Artemis Financial operates.

**External Threats:** Phishing, Man-in-the-Middle attacks, data breaches, etc., could be immediate threats.

**Modernization Requirements:**

**Open-Source Libraries:** Ensure they are regularly updated and free from known vulnerabilities.

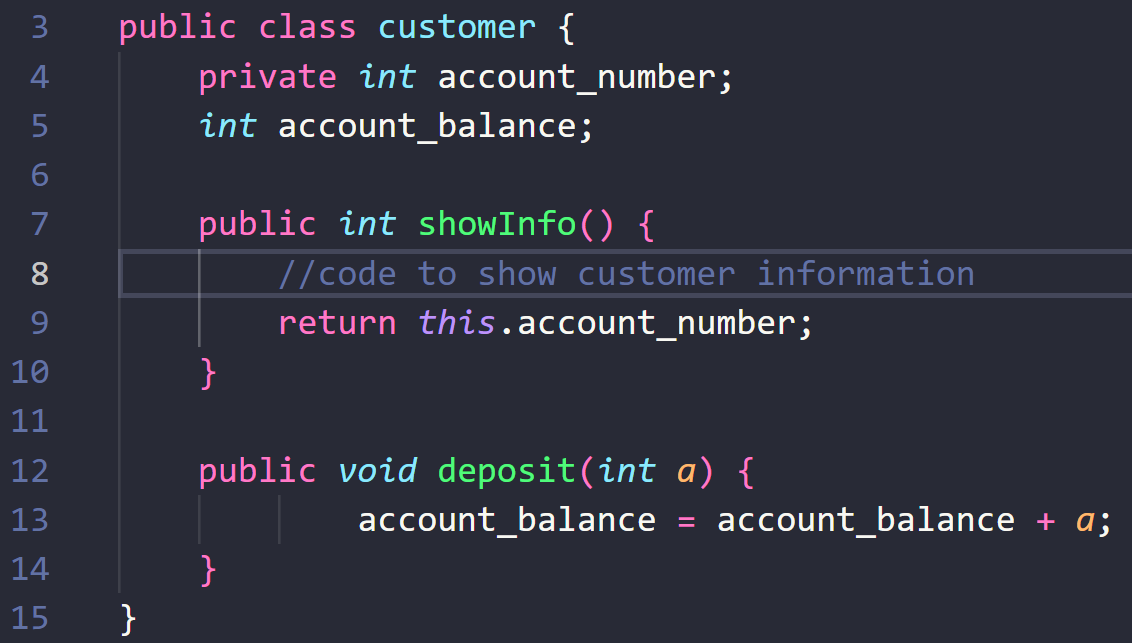
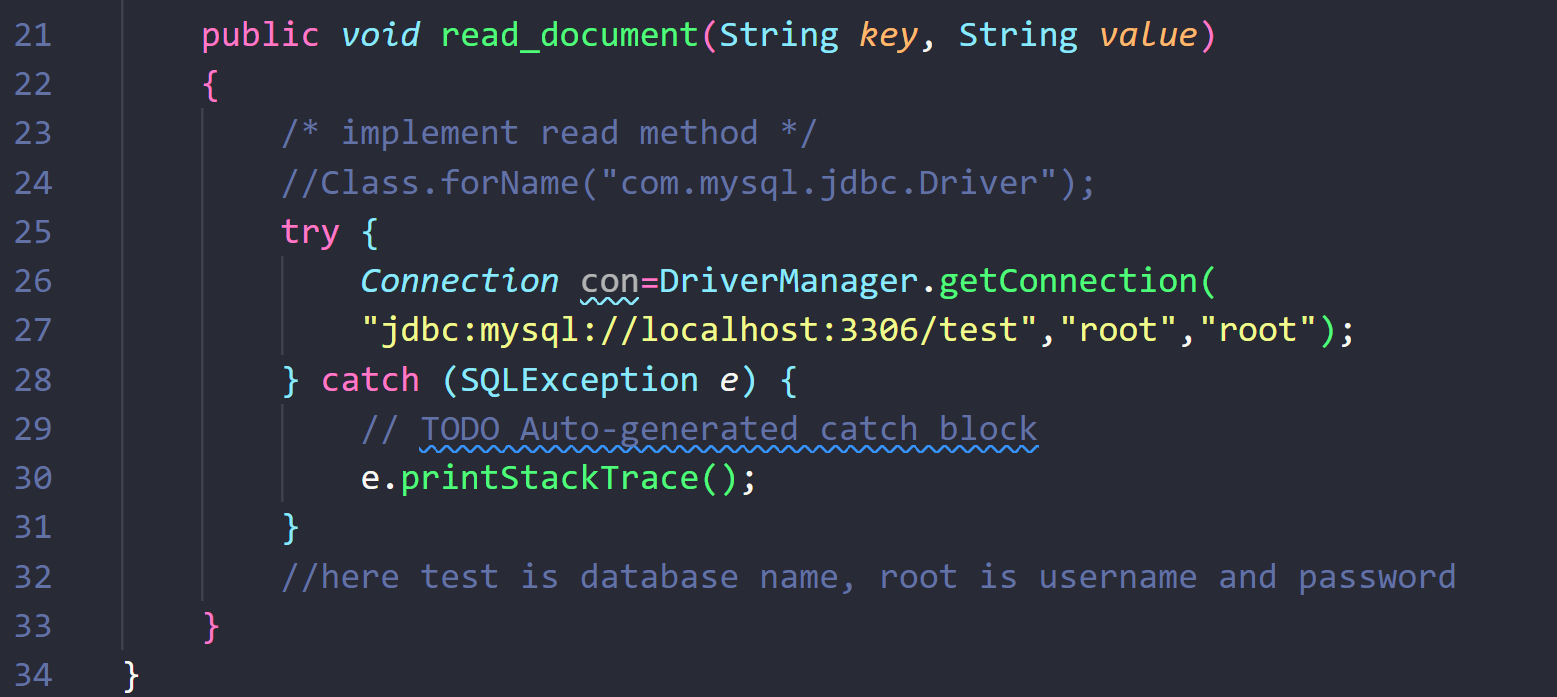
**Evolving Web Application Technologies**: Adopt and keep up with the latest secure coding practices and technologies, for example SSL, HTTPS, OAuth, etc...

## Areas of Security:

* **Architecture Review:** Assess application architecture for secure design principles.
* **Input Validation:** Ensure all user input is validated to prevent injection attacks.
* **APIs:** Secure API interactions to protect against unauthorized access.
* **Cryptography:** Use robust encryption methods for data at rest and in transit.
* **Client/Server:** Secure distributed computing, including session management.
* **Code Error:** Handle errors securely to prevent leakage of sensitive information.
* **Code Quality:** Implement secure coding practices/patterns.

## Manual Review

* **Input Validation Flaws:** Look for SQL injection, XSS vulnerabilities.
* **Improper Error Handling:** Check for detailed error messages exposing system information.
* **Insecure Authentication/Authorization:** Verify authentication flows and access controls.
* **Misconfigured Security Settings:** Review security configurations in the code and deployment environment.
* **Outdated Libraries:** Identify usage of outdated or vulnerable third-party libraries.

**Customer.java:**  
  
**Insecure Data Handling**: The **customer** class exposes the “**account\_balance”** without proper validation in the “**deposit”** method, which could potentially allow a threat actor to arbitrarily modify to account balances​.  
  
  
**DocData.java:**  
  
**Hardcoded Credentials/ SQL Injection**: The “**DocData”** class contains hardcoded database credentials within the “**read\_document”** method. This practice exposes sensitive information within the code base. Also, without the proper implementation there is a n open risk of SQL injection if user input is not properly sanitized.  
  
**Pom.xml:**  
  
**Outdated Libraries:** The project uses an outdated version of the **Bouncy Castle library (v1.46),** along with an old version of **Java (v1.8)**. Both which poses potentially opening up to the risk of exploiting known vulnerabilities. It is imperative to update to a more recent versions of any libraries used and the java version.

**4. Static Testing**

After integrating the Maven Dependency-Check plugin and running a scan, the following vulnerabilities were identified:

* **Vulnerabilities**: Several dependencies, including Bouncy Castle, Spring Boot, Logback, Log4J, and others, were found to be vulnerable, with vulnerabilities ranging from LOW to CRITICAL severity.
* **Description and Solutions**: Each vulnerability is associated with specific CVEs, detailing cryptographic issues, information exposure, and more. Solutions typically involve upgrading to a safer version of the affected libraries.
* **Attribution**: The vulnerabilities were identified through CVE databases and other security advisories.

1. **Mitigation Plan**
2. **Prioritized Action List**: Start with vulnerabilities classified as CRITICAL and work down to LOW, focusing on components used for security-sensitive operations first.
3. **Updating Libraries**: Upgrade Bouncy Castle, Spring Boot, Logback, Log4J, and other vulnerable dependencies to their latest, non-vulnerable versions.
4. **Code Changes**:
   * **Customer.java:** Implement input validation in the deposit method to ensure only positive amounts can be added to the account balance. This prevents erroneous or malicious attempts to manipulate account balances through SQL injections.
   * **DocData.java:** Remove hardcoded credentials from the “**read\_document"** method. Instead, utilize environment variables or secure vault solutions to handle sensitive information. Also, prepare for the implementation of this method with SQL queries by using “PreparedStatement” to prevent SQL injection attacks.
5. **Enhancements in Security Configurations**:
   * **General Application Security**: Enforce the principle of least privilege across all application components. Ensure that error messages are generic to prevent information leakage while still being helpful for debugging purposes.
   * **Database Security**: Secure database connections by using SSL and regularly rotate credentials. Ensure that database queries are performed using parameterized queries or ORM frameworks to prevent SQL injection.
   * **Encryption Enhancements**: Evaluate the use of updated encryption libraries and algorithms, especially for sensitive data handling. Upgrade the Bouncy Castle library to the latest version to mitigate vulnerabilities found in older versions.
   * **Dependency Management**: Integrate tools like OWASP Dependency-Check in the CI/CD pipeline to automatically check for vulnerable dependencies as part of the build process. This helps in keeping the project dependencies up-to-date and secure.
6. **Regular Security Audits and Updates**:

* Establish a regular schedule for security audits that includes both automated scanning and manual code review. This ensures that new code changes are vetted for security issues before being deployed.
* Keep abreast of the latest security advisories for the libraries and frameworks used in the project. Subscribe to security bulletins or use tools that notify you of relevant vulnerabilities.