****

# 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** | **[JAN 23, 2024]** | **[Victor Udeh]** | **First Release** |

## 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

Victor Udeh

## Interpreting Client Needs

1. **Secure Communications Importance**: Evaluate how critical secure communication is to Artemis Financial. Given the company's involvement in financial services, secure communication is paramount not only to protect financial assets but also to maintain customer confidentiality and trust.
2. **International Transactions**: Investigate whether Artemis Financial engages in cross-border financial activities. Such transactions could subject the company to international regulations and laws, including those pertaining to Personally Identifiable Information (PII) and secure communication standards.
3. **Governmental Restrictions and Compliance**: Examine the impact of legal frameworks like the Electronic Communications Privacy Act (ECPA) in the USA on Artemis Financial's operations. Compliance with laws such as the CFAA, SOX, and Gramm-Leach-Bliley Act is vital. Consulting with external legal counsel for comprehensive compliance is advisable.
4. **External Threat Landscape**: Analyze the range of cyber threats that Artemis Financial may face. These could range from minor yet disruptive attacks like Denial-of-Service (DOS) to more severe threats such as data breaches, which could lead to significant financial and reputational damage.
5. **Modernization and Technological Evolution**: Consider the implications of incorporating modern technologies, including open-source libraries, in Artemis Financial's operations. This includes understanding the responsibilities that come with open-source licenses and the necessity of regular updates and maintenance. Highlight the importance of advanced encryption for data protection, noting the evolving landscape of cryptographic security.
6. **Risk of Open Source and Library Maintenance**: Address the specific risks and management needs associated with using open-source libraries, as exemplified by incidents like the widespread Log4J vulnerabilities. Emphasize the need for vigilant maintenance and frequent updates of all software components, whether open-source or proprietary, to safeguard against vulnerabilities.

## Areas of Security

1. **Input Validation**: This is a critical security measure for any API. It involves ensuring that all user-provided data is checked and "sanitized" to prevent malicious inputs that could cause unexpected behavior or vulnerabilities in the system. Given that even data from trusted users can be malformed, either intentionally or accidentally, robust input validation is essential to maintain the integrity and security of the API.
2. **Secure API Interactions**: Securing the API involves more than just validating inputs; it also encompasses ensuring that the interactions with the API are secure. This includes authenticating and authorizing users and systems that interact with the API, as well as securing the data transmitted between the client and the server. While frameworks like Spring offer some built-in security features, it is important to implement additional, customized error handling that does not reveal sensitive information about the system architecture to potential attackers.
3. **Code Errors**: It is important to manage and mitigate code errors effectively. These errors can often provide attackers with insights into the system's structure or offer them entry points for exploits. The error handling should be generic enough to avoid revealing system details but specific enough to aid in debugging and maintaining system integrity.
4. **Encapsulation**: Utilizing encapsulation in software development, especially in the context of a method-based architecture, is vital. This involves setting variables to private and using getter and setter methods for data access and manipulation. Encapsulation helps in safeguarding sensitive data from unauthorized access and modifications, thereby maintaining data integrity.
5. **Cryptography**: Considering the nature of Artemis Financial’s operations, implementing strong encryption is indispensable. All data transmissions and database storage should employ robust encryption standards, such as RSA-2048, to protect sensitive financial information. Additionally, it’s crucial to ensure that the chosen encryption methods comply with the legal requirements in each operating country, as laws and regulations regarding cryptography can vary significantly.

## Manual Review

Upon manually inspecting the code base for Artemis Financial's RESTful API, several key vulnerabilities have been identified:

1. **Inadequate Use of Get/Set Methods**: The application shows inconsistent implementation of getter and setter methods. For example, while the **Greeting** class appropriately sets its parameters to private and utilizes these methods, the **GreetingController** fails to do so. This inconsistency can lead to uncontrolled access to class fields, potentially resulting in unauthorized data manipulation.
2. **Unsanitized User Input**: User inputs are not properly sanitized in the application. In the **number** method, user-provided data is directly used in a string, creating a risk of injection attacks. Similar vulnerabilities are observed in the **CRUDController**'s **CRUD** method. These vulnerabilities could lead to serious security breaches like SQL injection or cross-site scripting (XSS).
3. **Insecure Handling of Sensitive Data**: The **Customer** class exposes the **account\_balance** as a public member. This field should be private with access controlled through getter and setter methods to prevent unauthorized access or manipulation of sensitive financial data.
4. **Coding Flaws in myDateTime and DocData Classes**: Issues were noted in the **myDateTime** class, including incomplete methods like **setMyDateTime**. Additionally, the **DocData** class contains hardcoded, easily guessable credentials for testing ("root", "root"). This approach is insecure; credentials should be obfuscated, and all passwords should be salted and hashed for security, even in testing scenarios.
5. **Lack of Adequate Error Handling and Messaging**: There is a notable deficiency in error handling and messaging across various classes and methods. An example of this is the SQL database connection failure in **DocData** --> **read\_document**, which results in a full stack trace being revealed. This inadequate error messaging could inadvertently leak sensitive information or provide clues about the system's architecture, potentially aiding attackers in exploiting the system.

To enhance the security of Artemis Financial's software application, it is crucial to address these vulnerabilities through comprehensive code review, implementation of secure coding practices, robust input validation, and effective error handling.

## Static Testing

A screenshot of a computer screen

Description automatically generated

**Dependencies, Vulnerability Code, Description, and Mitigation Documentation**

1. SLF4J API

* **Dependency**: **slf4j-api-1.7.30.jar**
* **Vulnerability IDs**: None
* **Package**: **pkg:maven/org.slf4j/slf4j-api@1.7.30**
* **Highest Severity**: None
* **CVE Count**: 0
* **Confidence**: Not Applicable
* **Evidence Count**: 29
* **Description**: No known vulnerabilities.
* **Mitigation**: Ensure regular updates and monitor for future vulnerabilities.

2. Bouncy Castle Provider

* **Dependency**: **bcprov-jdk15on-1.46.jar**
* **Vulnerability IDs**: **cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.46:\*:\*:\*:\*:\*:\*:\***
* **Package**: **pkg:maven/org.bouncycastle/bcprov-jdk15on@1.46**
* **Highest Severity**: HIGH
* **CVE Count**: 17
* **Confidence**: Highest
* **Evidence Count**: 37
* **Description**: Cryptography API known to have high-severity vulnerabilities.
* **Mitigation**: Update to a later, secure version. Audit usage for exposure to known vulnerabilities.

3. Spring Boot

* **Dependency**: **spring-boot-2.2.4.RELEASE.jar**
* **Vulnerability IDs**: **cpe:2.3:a:vmware:spring\_boot:2.2.4:release:\*:\*:\*:\*:\*:\***
* **Package**: **pkg:maven/org.springframework.boot/spring-boot@2.2.4.RELEASE**
* **Highest Severity**: CRITICAL
* **CVE Count**: 3
* **Confidence**: Highest
* **Evidence Count**: 32
* **Description**: Critical vulnerabilities affecting various functionalities.
* **Mitigation**: Urgently update to the latest version. Review application for potential security breaches.

4. Logback Core

* **Dependency**: **logback-core-1.2.3.jar**
* **Vulnerability IDs**: **cpe:2.3:a:qos:logback:1.2.3:\*:\*:\*:\*:\*:\*:\***
* **Package**: **pkg:maven/ch.qos.logback/logback-core@1.2.3**
* **Highest Severity**: HIGH
* **CVE Count**: 2
* **Confidence**: Highest
* **Evidence Count**: 32
* **Description**: Logging framework with high-severity issues.
* **Mitigation**: Update to a safer version and verify logging configurations for security.

5. Log4j API

* **Dependency**: **log4j-api-2.12.1.jar**
* **Vulnerability IDs**: **cpe:2.3:a:apache:log4j:2.12.1:\*:\*:\*:\*:\*:\*:\***
* **Package**: **pkg:maven/org.apache.logging.log4j/log4j-api@2.12.1**
* **Highest Severity**: CRITICAL
* **CVE Count**: 5
* **Confidence**: Highest
* **Evidence Count**: 46
* **Description**: Critical vulnerabilities in logging library.
* **Mitigation**: Immediate update required. Review configurations and logs for anomalies.

6. JUL to SLF4J Bridge

* **Dependency**: **jul-to-slf4j-1.7.30.jar**
* **Vulnerability IDs**: None
* **Package**: **pkg:maven/org.slf4j/jul-to-slf4j@1.7.30**
* **Highest Severity**: None
* **CVE Count**: 0
* **Confidence**: Not Applicable
* **Evidence Count**: 28
* **Description**: No known vulnerabilities.
* **Mitigation**: Regular monitoring for updates and vulnerabilities.

7. Jakarta Annotation API

* **Dependency**: **jakarta.annotation-api-1.3.5.jar**
* **Vulnerability IDs**: None
* **Package**: **pkg:maven/jakarta.annotation/jakarta.annotation-api@1.3.5**
* **Highest Severity**: None
* **CVE Count**: 0
* **Confidence**: Not Applicable
* **Evidence Count**: 32
* **Description**: No known vulnerabilities.
* **Mitigation**: Keep the package updated and monitor for new releases.

8. SnakeYAML

* **Dependency**: **snakeyaml-1.25.jar**
* **Vulnerability IDs**: **cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25:\*:\*:\*:\*:\*:\*:\***, **cpe:2.3:a:yaml\_project:yaml:1.25:\*:\*:\*:\*:\*:\*:\***
* **Package**: **pkg:maven/org.yaml/snakeyaml@1.25**
* **Highest Severity**: CRITICAL
* **CVE Count**: 10
* **Confidence**: Highest
* **Evidence Count**: 28
* **Description**: YAML parser with critical security vulnerabilities.
* **Mitigation**: Update to a more secure version, validate and sanitize all YAML input.

9. Jackson Databind

* **Dependency**: **jackson-databind-2.10.2.jar**
* **Vulnerability IDs**: **cpe:2.3:a:fasterxml:jackson-databind:2.10.2:\*:\*:\*:\*:\*:\*:\***
* **Package**: **pkg:maven/com.fasterxml.jackson.core/jackson-databind@2.10.2**
* **Highest Severity**: HIGH
* **CVE Count**: 6
* **Confidence**: Highest
* **Evidence Count**: 39
* **Description**: High-severity vulnerabilities in data-binding library.
* **Mitigation**: Upgrade to a patched version, review data processing workflows.

10. Jackson Annotations

* **Dependency**: **jackson-annotations-2.10.2.jar**
* **Vulnerability IDs**: None
* **Package**: **pkg:maven/com.fasterxml.jackson.core/jackson-annotations@2.10.2**
* **Highest Severity**: None
* **CVE Count**: 0
* **Confidence**: Not Applicable
* **Evidence Count**: 38
* **Description**: No reported vulnerabilities.
* **Mitigation**: Maintain updates, monitor for new releases.

11. Jackson Core

* **Dependency**: **jackson-core-2.10.2.jar**
* **Vulnerability IDs**: None
* **Package**: **pkg:maven/com.fasterxml.jackson.core/jackson-core@2.10.2**
* **Highest Severity**: None
* **CVE Count**: 0
* **Confidence**: Not Applicable
* **Evidence Count**: 45
* **Description**: No known vulnerabilities.
* **Mitigation**: Regular updating and monitoring.

12. Jackson Datatype JDK8

* **Dependency**: **jackson-datatype-jdk8-2.10.2.jar**
* **Vulnerability IDs**: **cpe:2.3:a:fasterxml:jackson-modules-java8:2.10.2:\*:\*:\*:\*:\*:\*:\***
* **Package**: **pkg:maven/com.fasterxml.jackson.datatype/jackson-datatype-jdk8@2.10.2**
* **Highest Severity**: None (Low Confidence)
* **CVE Count**: 0
* **Confidence**: Low
* **Evidence Count**: 39
* **Description**: No high-severity vulnerabilities identified, but caution advised.
* **Mitigation**: Stay updated with the latest versions, perform regular security checks.

13. Tomcat Embed Core

* **Dependency**: **tomcat-embed-core-9.0.30.jar**
* **Vulnerability IDs**: **cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\***, **cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\***
* **Package**: **pkg:maven/org.apache.tomcat.embed/tomcat-embed-core@9.0.30**
* **Highest Severity**: CRITICAL
* **CVE Count**: 26
* **Confidence**: Highest
* **Evidence Count**: 39
* **Description**: Multiple critical vulnerabilities in Tomcat server component.
* **Mitigation**: Immediate update required, ensure secure configuration, monitor for patches.

## Mitigation Plan

To address the security vulnerabilities identified in Artemis Financial’s software application, a systematic approach should be taken. The process involves understanding the vulnerabilities, prioritizing them based on severity and impact, and then systematically addressing each one. Here are the steps to remedy the identified security vulnerabilities:

* + Categorize and Prioritize Vulnerabilities
    - Severity Assessment: Classify the vulnerabilities based on their severity (Critical, High, Medium, Low). This classification can be based on the potential impact, exploitability, and the sensitivity of the affected data or system functionality.
    - Impact Analysis: Assess the business and operational impact of each vulnerability. Consider factors such as data loss, system downtime, compliance issues, and reputational damage.
    - Prioritization: Prioritize the vulnerabilities for remediation. Critical and high-severity vulnerabilities that impact key business processes or sensitive data should be addressed first.
  + Develop Remediation Plans
    - Patch Management: For vulnerabilities related to outdated software or known bugs, apply patches or updates provided by the vendors. Ensure that all dependencies are also updated.
    - Code Fixes: If the vulnerabilities are due to flaws in the custom code, prepare a plan to modify the code. This might involve rewriting certain parts, input validation, implementing proper error handling, etc.
    - Configuration Changes: In cases where vulnerabilities are due to misconfigurations (like insecure server settings), reconfigure the systems according to best security practices.
  + Implement Remediation Measures
    - Testing Before Deployment: Before applying patches or pushing code changes to production, test them in a controlled environment to ensure they don't introduce new issues.
    - Rollout Patches and Updates: Deploy the patches and updates in a phased manner, starting with the most critical systems.
    - Code Deployment: For code changes, use a CI/CD pipeline to ensure smooth deployment. Monitor the application for any unexpected behavior post-deployment.
  + Validate Remediation
    - Reassessment: After remediation, conduct another round of testing to ensure that the vulnerabilities have been effectively addressed. This can include manual review and automated scanning.
    - Regression Testing: Perform regression testing to ensure that the changes have not adversely affected existing functionalities.
  + Update Security Policies and Practices
    - Lessons Learned: Analyze the root cause of the vulnerabilities and update your development and operational practices to prevent similar issues in the future.
    - Training: Provide training and awareness sessions for the development and operations teams to understand the security best practices and the importance of secure coding.
  + Continuous Monitoring
    - Regular Scans: Implement regular vulnerability scanning and security assessments as part of your ongoing security strategy.
    - Anomaly Detection: Use monitoring tools to detect any abnormal activities that might indicate a breach or an unsuccessful remediation.
  + Documentation and Reporting
    - Documentation: Keep detailed records of the vulnerabilities, the steps taken to address them, and the outcomes of the remediation efforts.
    - Compliance and Reporting: Report the findings and remediation actions to relevant stakeholders, including compliance teams if necessary.

**Conclusion**

Addressing security vulnerabilities is an ongoing process that requires a structured approach. By following these steps, Artemis Financial can effectively mitigate the risks associated with these vulnerabilities, thereby protecting their assets and maintaining trust with their clients and stakeholders. Regular review and adaptation of security practices are essential to stay ahead of evolving threats.