****

# 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** | **11/8/2023** | **Jason Buol** |  |

## 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

Jason Buol

## Interpreting Client Needs

Secure communications are crucial for maintaining the confidentiality, integrity, and availability of the company’s data. It ensures that sensitive financial transactions and client information are protected from eavesdropping, tampering, and other forms of cyber threats. For a financial company like Artemis Financial, secure communication is invaluable for maintaining trust and complying with industry regulations such as PCI DSS for payment data, GDPR for data privacy, and possibly other depending on the regions they operate in.

If Artemis Financial makes international transactions, it must adhere to international standards and regulations for secure communications, such as ISO/IEC 27001 for information security management and consider the specific cybersecurity laws and regulations of the countries it operates in. Restrictions like those in the EU’s GDPR or China’s Cybersecurity Law impact how data is transferred and stored and require specific protective measures.

The common external threats to web applications include but are not limited to:

* Phishing attacks targeting employees or customers.
* Distributed Denial of Service (DDoS) attacks that could disrupt services.
* Man-in-the-middle attacks intercepting communications.
* Zero-day exploits targeting unpatched vulnerabilities in software.
* Advanced Persistent Threats (APTs) that may target financial institutions specifically.

The immediate future may see the rise of AI-powered attacks, more sophisticated ransomware, and threats exploiting IoT devices.

Modern web application technologies evolve rapidly, and so does the landscape of potential vulnerabilities. While beneficial for rapid development, open-source libraries can introduce vulnerabilities if not properly vetted and kept updated. New Frameworks and APIs may have undiscovered vulnerabilities. Regular security reviews and updates are necessary.

## Areas of Security

Artemis Financial’s application architecture must be thoroughly reviewed for security considerations. This includes the structure of the network, the configuration of the servers and services, and the data flow between components. For a financial company, ensuring a secure and robust architecture is paramount to prevent systemic vulnerabilities that could be exploited.

All user-generated inputs should be validated to prevent common vulnerabilities such as SQL injection, Cross-Site Scripting, and Command Injection. Artemis Financial must ensure that: Client-side validation is complemented by server-side checks as the former can be bypassed. Whitelisting allowed characters to only allow known safe characters to be used. They have prepared statements and ORMs for database interactions to prevent injection attacks.

APIs are integral to modern financial applications for transactions and data exchange. Secure API interactions are necessary to prevent attacks such as Man-in-the-Middle or unauthorized data access. This can be accomplished with the use of authentication tokens like OAuth to control access to API endpoints, rate limiting to prevent abuse of the API which could lead to denial-of-service conditions, and input sanitization to ensure that data from API requests is treated as untrusted and sanitized.

The use of encryption to secure data both in transit (via SSL/TLS) and at rest is critical. Artemis Financial should implement strong encryption standards and key management practices such as using up-to-date protocols, cipher suites, and strong keys.

Ensuring that the components of the application do not expose the internal workings to other parts or users that do not need access to that information. Good encapsulation practices can prevent a wide range of security issues. By using the principle of least privilege, you limit the access rights for users to the bare minimum they need to complete their tasks. Service abstraction keeps the implementation details of a service hidden from the consumers.

## Manual Review

In the DocData class, there are multiple potential security vulnerabilities. The database URL, username, and password are hardcoded in the “read\_document’ method. This is a significant security risk because anyone with access to the codebase can see these credentials. It is better to store sensitive information in environment variables or a configuration file that is not included in the source control. The ‘Connection’ object ‘con’ is created but never closed. This can lead to resource leaks where database connections remain open and eventually exhaust the database resources. The ‘read\_document’ method implies that it will perform a database operation with the provided ‘key’ and ‘value’. If these parameters are concatenated directly into a SQL query, it could lead to SQL injection vulnerabilities.

## Static Testing

DEPENDENCY Description

|  |  |
| --- | --- |
| Spring-boot-2.2.4.RELEASE.jar | The Spring boot version that is being used is out of date, which is a critical-severity vulnerability. |
| Log4j-api-2.12.1.jar | This critical-severity vulnerability is due to using an outdated version of Log4j. |
| Snakeyaml-1.25.jar | This critical-severity vulnerability is due to using an outdated version of SnakeYAML. |
| Tomcat-embed-core-9.0.30.jar | This critical-severity vulnerability is due to using an outdated version of tomcat-embed-core. |
| Bcprov-jdk15on-4.46.jar | This high-severity vulnerability is due to using an outdated version of bcprov-jdk15on. |
| Jackson-databind-2.10.2.jar | This high-severity vulnerability is due to using an outdated version of Jackson-databind. |
| Spring-web-5.2.3.RELEASE.jar | This high-severity vulnerability is due to using an outdated version of spring-web. |
| Spring-beans-5.2.3.RELEASE.jar | This high-severity vulnerability is due to using an outdated version of spring-beans. |
| Logback-core-1.2.3.jar | This medium-severity vulnerability is due to using an outdated version of logback-core. |
| Hibernate-validator-6.0.18.Final.jar | This medium-severity vulnerability is due to using an outdated version of hibernate-validator. |

## Mitigation Plan

It is recommended that the vulnerabilities be addressed in order of severity to ensure that the most severe cases are addressed. Going through the dependency check report shows that the largest problem with this code is that most of the dependencies are outdated leaving them vulnerable to known attacks. To protect from already-known vulnerabilities it is vitally important to ensure that all the dependencies are using the latest versions that have been patched to protect from all the known vulnerabilities. It is also highly recommended to routinely monitor for any updates to the dependencies that patch any vulnerabilities that are found in the future. After updating the dependencies, it is important to thoroughly test the full application, with added attention to functions that directly interact with the updated libraries, to ensure that the application still fully functions with the new versions.

Using an integrated tool such as OWASP Dependency-Track can help with the continuous monitoring of the application to ensure that any new vulnerabilities are caught as soon as possible. Having a formal policy for regularly updating and patching software dependencies instead of relying on someone remembering to check for updates and patch the application will make the process more streamlined and less likely to be forgotten.

Implement a training program to ensure that all developers are aware of the best practices for using and updating third-party libraries. Also, train developers on secure coding practices to prevent the introduction of new vulnerabilities. Having developers who are completely trained and knowledgeable with both the use of third-party libraries and secure coding practices most vulnerabilities can be prevented prior to the application ever being deployed and having exposed vulnerabilities.