****

# 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** | **11/11/2023** | **Dylan Cavazos** |  |

## 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

Dylan Cavazos

## Interpreting Client Needs

Analyzing the security for Artemis Financial as a software developer my aim is to provide insightful feedback into the relative risk associated with the companies financial and software conduct. Given the sensitive nature of the financial information secure communications must be properly audited to ensure the confidentiality and integrity of transmitted data as well as safeguarded client information is properly followed. These principles help to maintain the trust of customers and are especially crucial for complying with the various regulatory standards, especially within the financial sector.

Since the company regularly engages with international transactions, clients often from various geographic locations which can increase the need for secure financial planning. International transactions are often accompanied by a litany of security considerations, which increased the need for robust encryption protocols and compliance with data protection regulations that are often specific to different countries **(Manico, n.d.)**. Moreover, governmental restrictions related to secure communications must be considered as well. It is important to consider the various standards and laws that accompany each country that the company interacts with.

Artemis Financial conducts business with international clients. This can often lead to unpredicted exposure to unsecured communication and potential vulnerabilities. Considering that the financial industry is a top target for potential threat actors, Artemis is no exception and faces external threats perhaps both currently and in the future. The known attack vectors are indeed limitless but the common attacks in which Artemis might be exposed to include common web application security threats like SQL injection cross-site scripting (XSS), and data breaches. Each of which pose risks to the confidentiality and integrity of financial information. To protect against these, it is important that the company implements robust security measures to mitigate the threats and protect themselves against any potential cyberattacks.

In terms of modernization requirements, Artemis Financial must consider the role of open-source libraries and evolving web application technologies. Open-source libraries, while do provide cost-effective solutions can be vulnerable to security attacks or compromises, so must be continuously monitored. Evolving web application technologies require staying up to date on the latest security practices to mitigate any emerging threats, ensuring a resilient defense against potential attacks. Artemis Financial should consider implementing regular audits of open-source libraries and evolving web applications.

## Areas of Security

In conducting a vulnerability assessment for Artemis Financials’ web application, it is important to identify the specific areas of security based on the application’s functionality. Using the Vulnerability Assessment Process Flow Diagrams as a guide to systematically determine which security aspects are most important to Artemis Financial.

For example, one critical area to consider is Input Validation, which helps ensure that user-provided data is properly validated before being processed by the application. Regarding financial planning software from Artemis, where sensitive client information is continuously processed, proper input validation is essential to prevent common security vulnerabilities such as SQL injections and cross-site scripting(CSS) attacks. This ensures that client information can remain secure stored within the applications database and that user input for the financial software is adhered to strong security standards.

Considering that Artemis Financial uses a RESTful web API service for their application this can often lead to unsecure methods within the program. API security helps to ensure that data exchanged between different components within the system are protected against any unauthorized access or potential exploits. API communication security methods are regulatory requirements and any failure to address the API security of an application can expose the program to vulnerabilities in data transmission, leading to unauthorized access or data breaches **(*Secure Coding Guidelines for Java SE*, n.d.).**

In addition, another pertinent area is Cryptography and Encryption Use. Financial data, especially personalized financial plans that contains sensitive customer info must adhere to robust security practices. Assessing how cryptography is implemented within the application is crucial to safeguarding the confidentiality and integrity of any data transmitted or stored within the application. Potential weaknesses within cryptographic implementations could result in data exposure or compromise, posing a significant risk to the reputation of Artemis Financial or its clients.

Another aspect that is crucial to security assessment is adhering to secure coding practices and code quality. Examining the overall codebase for adherence to secure coding standards ensures that vulnerabilities arising from poor coding practices are mitigated. To prevent this, it is important to review code for errors involving input validation or insecure dependencies, which can introduce vulnerabilities or malware. Reviewing code and adhering to secure coding standards is a priority for companies that use advanced web application technologies to implement their software and following strict guidelines can often save companies like Artemis Financial from common cybersecurity attacks or data breaches.

## Manual Review

Upon reviewing the software application’s code for Artemis Financial, I have identified some potential security vulnerabilities. To address these, it is important to incorporate principles provided from the vulnerability process flow diagram, as well as other references to secure coding standards.

The **CRUDController class** for instance, manages specific API interactions, but the security of these interactions relies on proper input validation. Common preventative measures for this include sanitizing user input to prevent common attacks like SQL injections. Another potential security flaw is the use of the **DocData class**, which interacts with a database using SQL queries is not parameterized, potentially exposing the application to SQL injection. “Because of the level of the risk posed by SQL injection, query parameterization is the most important technique a developer can use to build a secure website. ” **(Manico, n.d.-e).**

Since Artimis Financial deals with sensitive customer information, it is important for the application to adhere to strict cryptographic principles with sending information back and forth. This can often be done using advanced cryptographic methods like using HTTPS as well as utilizing cryptographic protocols like Secure Sockets Layer (SSL) or Transport Layer Security (TLS) **(Manico, n.d.-d).**

The code provided deals with client-server communication. When overseeing sensitive client information and communicating across networks it is vital to use secure communication practices. As mentioned earlier the use of HTTPS can increase the security of network communication as opposed to standard HTTP.

Another aspect of the Vulnerability diagram that is of use here is the principle of Code Quality. In this instance, the DocData class could benefit from more secure coding practices. For instance, the database connection credentials are hardcoded, and exception handling could be improved. The DocData class could also benefit from using “parameterized SQL statements from libraries like java.sql.PreparedStatement or java.sql.CallableStatement.” “As these are more well-written, higher-level libraries to insulate application code from SQL **(*Secure Coding Guidelines for Java SE*, n.d.-b**).”

Encapsulation is another crucial principle when adhering to strict secure coding standards. As the use of database operations using SQL queries within the DocData class is a potential insecure method. Proper encapsulation and access control using the appropriate methods should be prioritized. Additional secure coding principles include “Closing the connection as soon as possible,” making sure that the “application connects to the database with different credentials for every trust distinction (e.g., user, read-only user, guest, administrators). **(OWASP)”**

## Static Testing

**Vulnerability 1: bcprov-jdk15on-1.46.jar - CVE-2013-0169**

**Description:** The bcprov-jdk15on library version 1.46 has a high-severity vulnerability (CVE-2013-0169). It allows remote attackers to conduct distinguishing attacks and plaintext-recovery attacks via statistical analysis of timing data for crafted packets (NVD - CVE-2013-1624, n.d.).

I recommend upgrading to a patched version of the Bouncy Castle library (bcprov-jdk15on) that addresses the identified CVE-2013-0169 vulnerability.

**Vulnerability 2: spring-boot-2.2.4.RELEASE.jar - CVE-2022-27772**

**Description:** The spring-boot library version 2.2.4.RELEASE has a vulnerability that made this current version vulnerable to temporary directory hijacking. (CVE-2022-27772). Upgrade to version 2.2.11.RELEASE or later to mitigate the risk. (NVD - CVE-2022-27772, n.d.).

**Vulnerability 3: logback-core-1.2.3.jar - CVE-2021-42550**

**Description:** The logback-core library version 1.2.3 has a vulnerability that enabled an attacker with the right privileges to edit configuration files that could craft a malicious configuration allowing to execute arbitrary code loaded from LDAP servers. (CVE-2021-42550). Upgrade to version 1.2.7 or later. (**NVD - CVE-2021-42550, n.d.).**

**Vulnerability 4:** **log4j-api-2.12.1.jar - CVE-2020-9488**

**Description:** The log4j-api library version 2.12.1 has a vulnerability that could allow an SMTPS connection to be intercepted by a man-in-the-middle attack, potentially leaking log messages. (CVE-2020-9488). Upgrade to version 2.12.3 or 2.13.1 for Apache log4j to mitigate the risk of interception through a man-in-the-middle attack (NVD - CVE-2020-9488, n.d.).

**Vulnerability 5: snakeyaml-1.25.jar - CVE-2017-18640**

**Description:** The snakeyaml library version 1.25 has a vulnerability that allows entity expansion during a load operation. (CVE-2017-18640). Upgrade to version 1.26 or later to mitigate the risk associated. (NVD - CVE-2017-18640, n.d.).

**Vulnerability 6: jackson-datatype-jdk8-2.10.2.jar - CVE-2020-25649**

**Description:** The jackson-databind library version 2.10.2 has a vulnerability whether it did not have entity expansion secured properly. This allowed XML external entity (XXE) attacks. (CVE-2020-25649). Upgrade to a fixed version to prevent XML external entity attacks and safeguard data integrity (NVD - CVE-2020-25649, n.d.).

**Vulnerability 7:** **tomcat-embed-core-9.0.30.jar - CVE-2019-17569**

**Description:** Tomcat-embed-core is a core Tomcat implementation. The result of the regression led to a possibility of HTTP request smuggling if Tomcat was behind a reverse proxy that incorrectly managed invalid Transfer-Encoding headers. (CVE-2019-17569). Review and update to a version beyond the one where the regression was introduced to eliminate the potential risk of HTTP Request Smuggling (NVD - CVE-2019-17569, n.d.).

**Vulnerability 8: hibernate-validator-6.0.18.Final.jar - CVE-2020-10693**

**Description:** TA flaw was found in Hibernate Validator version 6.1.2.Final. A bug enables invalid EL expressions. This flaw allowed attackers to bypass input sanitation. (CVE-2020-10693). Update to a version that addresses this flaw (NVD - CVE-2020-10693, n.d.).

**Vulnerability 9:** **spring-web-5.2.3.RELEASE.jar - CVE-2016-1000027**

**Description:** The spring-web library version 5.2.3.RELEASE is vulnerable to remote code execution (CVE-2016-1000027, OSSINDEX). Assess the specific implementation and consider restricting or reconsidering untrusted data deserialization.

**Vulnerability 10: spring-beans-5.2.3.RELEASE.jar - CVE-2022-22965**

**Description:** A Spring MVC or Spring WebFlux application running JDK9+ is vulnerable to remote code execution via data binding. (CVE-2022-22965, OSSINDEX). Assess deployment configurations and monitor for updates to mitigate the risk of remote code execution.

**Vulnerability 11: spring-webmvc-5.2.3.RELEASE.jar - CVE-2021-22060**

**Description:** Older versions that are unsupported allow a user to provide malicious input to cause the insertion of additional log entries. (CVE-2021-22060, OSSINDEX). Update to a version beyond 5.3.13 or 5.2.18 to mitigate the risk of log manipulation.

**Vulnerability 12: spring-context-5.2.3.RELEASE.jar - CVE-2022-22968**

**Description:** Unsupported older version of the Spring Framework versions had patterns that allowed fields to be unprotected unless listed with both upper and lower case for the first characters. Improper handling of case sensitivity. (CVE-2022-22968, OSSINDEX). Update to a version beyond 5.3.18 or 5.2.20 to address the case-sensitive disallowedFields patterns.

## Vulnerability 13: spring-expression-5.2.3.RELEASE.jar - CVE-2022-22950

## Description: “In Spring Framework versions 5.3.0-5.3.16 and older versions, it is possible for a user to provide a specially crated SpEL expression that may cause a denial-of-service condition. (Dependency Check)” (CVE-2022-22950, OSSINDEX). Update to a version beyond 5.3.16 to mitigate the risk of a denial-of-service condition. Review and validate user-input SpEL expressions for potential attacks (Sonatype, Inc., n.d.-c).

## Mitigation Plan

**CRUDController Input Validation:**

We need to check how the system checks information as it is currently not strong enough. This makes it easy for potential threat actors to inject harmful commands into the database for a data breach or other attacks. To combat this, I suggest we improve how the information coming in is checked. Also, using a more advanced method to access the database is recommended. This will make it more difficult for attackers to access.

**DocData Class Security Flaws:**

Right now, the use of non-parameterized SQL queries in the DocData class is a risk for SQL injection attacks. To reduce the possibility of this happening I recommend we use Parameterized SQL statements using libraries like java.sql.PreparedStatement or java.sql.CallableStatement to help prevent injection attacks.

**Secure Communication:**

As of now there is a lack of secure communication practices involved with client-server interactions. To help prevent attacks I recommend implementing HTTPS for client-server communications to enhance network security. This will help in ensuring that sensitive client information is transmitted securely.

**Code Quality:**

Regarding code quality concerns, it is essential to enhance the security of the DocData class by securely storing database connection credentials instead of hardcoding them. Additionally, improving exception handling practices will help contribute to increased code resilience and overall system reliability.

**Static Testing Vulnerabilities:**

**bcprov-jdk15on-1.46.jar - CVE-2013-0169 (Bouncy Castle library):**

Upgrade the Bouncy Castle library to the latest secure release.

**spring-boot-2.2.4.RELEASE.jar - CVE-2022-27772 (spring-boot library):**

Upgrade the spring-boot library to version 2.2.11.RELEASE or later.

**logback-core-1.2.3.jar - CVE-2021-42550 (logback-core library):**

Upgrade the logback-core library to version 1.2.7 or later.

**log4j-api-2.12.1.jar - CVE-2020-9488 (log4j-api library):**

Upgrade the log4j-api library to version 2.12.3 or 2.13.1.

**snakeyaml-1.25.jar - CVE-2017-18640 (snakeyaml library):**

Upgrade the snakeyaml library to version 1.26 or later.

**jackson-datatype-jdk8-2.10.2.jar - CVE-2020-25649 (jackson-databind library):** Upgrade the jackson-databind library to a fixed version.

**tomcat-embed-core-9.0.30.jar - CVE-2019-17569 (Apache-Tomcat Library):**

Review and update the jackson-datatype-jdk8 library to a version beyond the one where the regression was introduced.

**hibernate-validator-6.0.18.Final.jar - CVE-2020-10693 (hibernate-validator library):**

Update the hibernate-validator library to a version that addresses the identified flaw.

**spring-web-5.2.3.RELEASE.jar - CVE-2016-1000027 (spring-web library):**

Assess the specific implementation of the spring-web library, restricting or reconsidering untrusted data deserialization.

**spring-beans-5.2.3.RELEASE.jar - CVE-2022-22965 (spring-beans library):**

Assess deployment configurations and monitor for updates to mitigate the risk of remote code execution.

**spring-webmvc-5.2.3.RELEASE.jar - CVE-2021-22060 (spring-webmvc library):** Update the spring-webmvc library to a version beyond 5.3.13 or 5.2.18.

**spring-context-5.2.3.RELEASE.jar - CVE-2022-22968 (spring-context library):** Update the spring-context library to a version beyond 5.3.18 or 5.2.20.

spring-expression-5.2.3.RELEASE.jar - CVE-2022-22950 **(spring-expression library):**

Update the spring-expression library to a version beyond 5.3.16.

Regularly review and validate user-input SpEL expressions to prevent potential attacks.

[Microsoft Word - OWASP SCP Quick Reference Guide\_v1.doc](https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v1.pdf)

Manico, J. (n.d.). *Iron-Clad java*. O’Reilly Online Learning. <https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch10.html#ch10lev2sec14>

Manico, J. (n.d.-c). *Iron-Clad java*. O’Reilly Online Learning. <https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch06.html#ch06lev2sec11>

Manico, J. (n.d.-d). *Iron-Clad java*. O’Reilly Online Learning. <https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch06.html#ch06lev1sec1>

Manico, J. (n.d.-e). *Iron-Clad java*. O’Reilly Online Learning. https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch07.html#ch07lev1sec3

*NVD - CVE-2013-1624*. (n.d.). <https://nvd.nist.gov/vuln/detail/CVE-2013-1624>

*NVD - CVE-2022-27772*. (n.d.). <https://nvd.nist.gov/vuln/detail/CVE-2022-27772>

*NVD - CVE-2021-42550*. (n.d.). <https://nvd.nist.gov/vuln/detail/CVE-2021-42550>

*NVD - CVE-2020-9488*. (n.d.). <https://nvd.nist.gov/vuln/detail/CVE-2020-9488>

*NVD - CVE-2017-18640*. (n.d.). <https://nvd.nist.gov/vuln/detail/CVE-2017-18640>

*NVD - CVE-2020-25649*. (n.d.). <https://nvd.nist.gov/vuln/detail/CVE-2020-25649>

*NVD - CVE-2019-17569*. (n.d.). <http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2019-17569>

*NVD - CVE-2020-10693*. (n.d.). https://nvd.nist.gov/vuln/detail/CVE-2020-10693

*Secure Coding Guidelines for Java SE*. (n.d.). <https://www.oracle.com/java/technologies/javase/seccodeguide.html>

Sonatype, Inc. (n.d.). *[CVE-2016-1000027] CWE-502: Deserialization of Untrusted Data*. Sonatype OSS Index. <https://ossindex.sonatype.org/vulnerability/CVE-2016-1000027?component-type=maven&component-name=org.springframework%2Fspring-web&utm_source=dependency-check&utm_medium=integration&utm_content=5.3.0>

Sonatype, Inc. (n.d.-b). *[CVE-2022-22965] CWE-94: Improper control of generation of Code ('Code injection’)*. Sonatype OSS Index. <https://ossindex.sonatype.org/vulnerability/CVE-2022-22965?component-type=maven&component-name=org.springframework%2Fspring-beans&utm_source=dependency-check&utm_medium=integration&utm_content=5.3.0>

Sonatype, Inc. (n.d.-b). *[CVE-2021-22060] CWE-117: Improper Output Neutralization for Logs*. Sonatype OSS Index. <https://ossindex.sonatype.org/vulnerability/CVE-2021-22060?component-type=maven&component-name=org.springframework%2Fspring-webmvc&utm_source=dependency-check&utm_medium=integration&utm_content=5.3.0>

Sonatype, Inc. (n.d.-d). *[CVE-2022-22968] CWE-178: Improper Handling of Case Sensitivity*. Sonatype OSS Index. <https://ossindex.sonatype.org/vulnerability/CVE-2022-22968?component-type=maven&component-name=org.springframework%2Fspring-context&utm_source=dependency-check&utm_medium=integration&utm_content=5.3.0>

Sonatype, Inc. (n.d.-c). *[CVE-2022-22950] CWE-770: Allocation of Resources Without limits or throttling*. Sonatype OSS Index. https://ossindex.sonatype.org/vulnerability/CVE-2022-22950?component-type=maven&component-name=org.springframework%2Fspring-expression&utm\_source=dependency-check&utm\_medium=integration&utm\_content=5.3.0