****

# Artemis Financial Vulnerability Assessment Report

Table of Contents

[Artemis Financial Vulnerability Assessment Report 1](#_Toc146020667)

[Document Revision History 3](#_Toc146020668)

[Client 3](#_Toc146020669)

[Developer 3](#_Toc146020670)

[1. Interpreting Client Needs 3](#_Toc146020671)

[2. Areas of Security 3](#_Toc146020672)

[3. Manual Review 4](#_Toc146020673)

[4. Static Testing 4](#_Toc146020674)

[5. Mitigation Plan 4](#_Toc146020675)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **09/19/2023** | **Kellen Burke** |  |

## Client



## Developer

Kellen D. Burke

## Interpreting Client Needs

Secure communication ensure that the financial and personal data of customers protected, limiting exposure to theft and unauthorized access. Part of this security effort is compliance with any regulatory requirements and industry standards, limiting the exposure of Artemis Financial to potential fines and penalties in the financial industry. If the country is operating outside the U.S., it is imperative that Artemis’ program abides by any country-specific and/or international regulations. Operating in different markets also requires that any calculations and conversions be checked for accuracy and updated regularly for constantly changing variables. Additionally, any government clients come along with their own sets of standards that must be maintained can complied with. Consulting with legal counsel may be the best option, as they are experts in the matter and could focus their resources on this landscape.

Artemis may be vulnerable to attacks from cybercriminals that may use social engineering or exploit insider threats to cause data breaches. Threats from international organizations may be a factor, seeking to fulfill the interests of other national threat actors. Any third-party services the application may depend on should be verified and secured for any know vulnerabilities and assessed for any unknow attack surfaces. Artemis and the development team should be considering regular assessments in the future, employee training, patch management, encryption, access controls, planning for incident response, and a system to log potential attacks once the application is deployed. Any open-source libraries should be reviewed and updated regularly and implement automated tools that monitor and manage API dependencies. Serverless computing should be considered for certain functions or services to reduce management needs and improve scalability. Browser compatibility should be maintained for new capabilities and cross-browser utilization to reach a wider customer-base.

## Areas of Security

Artemis Financial should be most concerned with input validation, secure API interactions, encryption, and client/server communication security. User inputs should be properly validated and sanitized, preventing common attacks like an injection attack or cross-site scripting (XSS). Securing API interactions is critical. Insecure APIs can lead to data leaks, unauthorized access, and other security breaches. Proper encryption is crucial for protecting sensitive data, especially for financial data. Weak encryption can lead to harmful actions that reduce the trust of customers and can mean violating regulations that may be in place. Additionally, securing client/server communication is essential for secure data transmission and authentication.

## Manual Review

There are a few common mistakes and vulnerabilities that are readily apparent in the provided Artemis code base that I was able to identify:

* In the *CRUDController.java* class, the ‘name’ parameter lacks proper input validation. This can lead to injection vulnerabilities such as SQL injection and XSS.
* The is no exception handling or error messages in the code. Proper error handling provides meaningful messages in response to errors preventing the exposure of potentially sensitive code information.
* There are no functions meant to control user authentication and authorization.
* The classes *customer.java* and *myDateTime.java* appear to be unused. Unused code should be removed to improve code clarity.

## Static Testing

The dependency check that was conducted on the project found **13** vulnerable dependencies.

1. **Code**: bcprov-jdk15on-1.46.jar

**Summary**: The TLS implementation in the Bouncy Castle Java library before 1.48 and C# library before 1.8 does not properly consider timing side-channel attacks on a non-compliant MAC check operation during the processing of malformed CBC padding, which allows remote attackers to conduct distinguishing attacks and plaintext-recovery attacks via statistical analysis of timing data for crafted packets.

**Potential** **Mitigations**: Not Found

**Phase**: Cryptography

1. **Code**: spring-boot-2.2.4.RELEASE.jar

**Summary**: In Spring Boot versions 3.0.0 – 3.0.5, 2.7.0 – 2.7.10, and older unsupported versions, an application that is deployed to Cloud Foundry could be susceptible to a security bypass. Users of affected versions should apply the following mitigation.

**Potential Mitigations**: 3.0.x users should upgrade to 3.0.6+. 2.7.x users should upgrade to 2.7.11+. Users of older, unsupported versions should upgrade to 3.0.6+ or 2.7.11+.

**Phase**: Patch Management

1. **Code**: logback-core-1.2.3.jar

**Summary**: In logback version 1.2.7 and prior versions, an attacker with the required privileges to edit configurations files could craft a malicious configuration allowing to execute arbitrary code loaded from LDAP servers.

**Potential Mitigations**: If available, use the signing/sealing features of the programming language to assure that deserialized data has not been tainted. For example, a hash-based message authentication code (HMAC) could be used to ensure that data has not been modified.

**Phase**: Architecture and Design; Implementation

1. **Code**: log4j-api-2.12.1.jar

**Summary**: Improper validation of certificate with host mismatch in Apache Log4j SMTP appender. This could allow an SMTPS connection to be intercepted by a man-in-the-middle attack which could leak any log messages sent through that appender. Fixed in Apache Log4j 2.12.3 and 2.13.1

**Potential Mitigations**: Certificates should be carefully managed and checked to assure that data are encrypted with the intended owner’s public key.

**Phase**: Architecture and Design; Implementation

1. **Code**: snakeyaml-1.25.jar

**Summary**: SnakeYaml’s Constructor() class does not restrict types which can be instantiated during deserialization. Deserializing yaml content provided by an attacker can lead to remote code execution. Recommended to use SnakeYaml’s SafeConstructor when parsing untrusted content to restrict deserialization. Upgrade to version 2.0 and beyond.

**Potential Mitigations**: If available, use the signing/sealing features of the programming language to assure that deserialized data has not been tainted. For example, a hash-based message authentication code (HMAC) could be used to ensure that data has not been modified.

**Phase**: Architecture and Design; Implementation

1. **Code**: Jackson-databind-2.10.2.jar

**Summary**: A flaw was found in FasterXML Jackson Databind, where it did not have entity expansion secured properly. This flaw allows vulnerability to XML external entity (XXE) attacks. The highest threat from this vulnerability is data integrity.

**Potential** **Mitigations**: Many XML parsers and validators can be configured to disable external entity expansion.

**Phase**: Implementation; System Configuration

1. **Code**: tomcat-embed-core-9.0.30.jar

**Summary**: When using the Apache JServ Protocol (AJP), care must be taken when trusting incoming connection to Apache Tomcat. Tomcat treats AJP connections as having higher trust than, for example, a similar HTTP connection. In Apache Tomcat 9.0.0.M1 to 9.0.0.30, 8.5.0 to 8.5.50 and 7.0.0 to 7.0.99, Tomcat shipped with an AJP Connector enabled by default that listened on all configured IP addresses. This vulnerability report identified a mechanism that allowed: - returning arbitrary files from anywhere in the web application – processing any file in the web application as a JSP. Further, if the web application allowed file upload and stored those files within the web application (or the attacker was able to control the content of the web application by some other means) then this, along with the ability to process a file as a JSP, made remote code execution possible. Mitigation is only required if an AJP port is accessible to untrusted users.

1. **Code**: hibernate-validator-6.0.18.Final.jar

**Summary**: A flaw was found in Hibernate Validator version 6.1.2.Final. A bug in the message interpolation processor enables invalid EL expressions to be evaluated as if they were valid. This flaw allows attackers to bypass input sanitation (escaping, stripping) controls that developers may have put in place when handling user-controlled data in error messages.

**Potential Mitigations**: Use an input validation framework such as Struts or the OWASP ESAPI Validation API. Note that using a framework does not automatically address all input validation problems; be mindful of weaknesses that could arise from misusing the framework itself.

**Phase**: Architecture and Design

1. **Code**: spring-web-5.2.3.RELEASE.jar

**Summary**: Pivotal Spring Framework through 5.3.16 suffers from a potential remote code execution (RCE) issue if used for Java deserialization of untrusted data. Depending on how the library is implemented within a product, this issue may or may not occur, and authentication may be required. NOTE: the vendor’s position is that untrusted data is not an intended use case. The product’s behavior will not be changed because some users rely on deserialization of trusted data.

**Potential Mitigations**: If available, use the signing/sealing features of the programming language to assure that deserialized data has not been tainted. For example, a hash-based message authentication code (HMAC) could be used to ensure that data has not been modified.

**Phase**: Architecture and Design; Implementation

1. **Code**: spring-beans-5.2.3.RELEASE.jar

**Summary**: A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e., the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it.

**Potential Mitigations**: Refactor your program so that you do not have to dynamically generate code.

**Phase**: Architecture and Design

1. **Code**: spring-webmvc-5.2.3.RELEASE.jar

**Summary**: A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e. the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it.

**Potential Mitigations**: Refactor your program so that you do not have to dynamically generate code.

**Phase**: Architecture and Design

1. **Code**: spring-context-5.2.3.RELEASE.jar

**Summary**: In Spring Framework version 5.3.0 – 5.3.18, 5.2.0 – 5.2.20, and older unsupported versions, the patterns for disallowedFields on a DataBinder are case sensitive which means a field is not effectively protected unless it is listed with both upper and lower case for the first character of the field, including upper and lower case for the first character of all nested fields within the property path.

**Potential Mitigations**: Avoid making decisions based on names of resources (e.g. files) if those resources can have alternate names.

**Strategy**: Input Validation

1. **Code**: spring-expression-5.2.3.RELEASE.jar

**Summary**: In Spring Framework versions 5.3.0 – 5.3.16 and older unsupported versions, it is possible for a user to provide a specially crafted SpEL expression that may cause a denial of service (DoS) condition.

**Potential Mitigations**: Clearly specify the minimum and maximum expectations for capabilities, and dictate which behaviors are acceptable when resource allocation reaches limits.

**Phase**: Requirements

## Mitigation Plan

Based on these findings, vulnerable dependencies should be addressed by reviewing the documentation and release notes of each library to truly understand the vulnerabilities and fixes. Regular updates and patches should be pushed in conjunction with automated scanning tools to receive up-to-date vulnerability notifications. The base code should be reviewed for the use of user input, and input validation implemented where applicable. Alongside input validation, proper error handling and event logging should be utilized. This will prevent leakage of potentially critical information of the code base structure, and in the occurrence of similar errors thrown, logging tools can be used to assess patterned attacks and alert program custodians. Authentication and authorization should be enacted. Spring Security is a tool that can help manage these properties within the application. Ensure that best practices are used when accessing databases. Encryption of sensitive data within a database should be used to protect sensitive data at rest. Source code should be clear and meaningful, complete with comments, explaining the classes and methods in use throughout the code. Employee training should be conducted with regular occurrence or in the event of a major change to the application. Consultation would be well advised regarding any regulatory compliance that may be required by the financial industry and international markets. Lastly, an effort to improve performance should be made throughout the production and deployment of the program, a good user experience can make a large impact on client satisfaction.