****

# Artemis Financial Vulnerability Assessment Report

Table of Contents

[Artemis Financial Vulnerability Assessment Report 1](#_Toc140097884)

[Document Revision History 3](#_Toc140097885)

[Client 3](#_Toc140097886)

[Developer 4](#_Toc140097887)

[1. Interpreting Client Needs 4](#_Toc140097888)

[2. Areas of Security 4](#_Toc140097889)

[3. Manual Review 6](#_Toc140097891)

[4. Static Testing 6](#_Toc140097892)

[5. Mitigation Plan 8](#_Toc140097893)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **7/16/2023** | **Thi Hong Van Nguyen** | **N/A** |

## Client



## Developer

Thi Hong Van Nguyen

## Interpreting Client Needs

Artemis Financial has identified specific priorities to meet their needs. These priorities encompass establishing secure communication channels, ensuring compliance with international regulations, and proactively mitigating external threats by modernizing their systems and adhering to industry-recognized software security best practices.

Protecting sensitive financial information is extremely important to Artemis Financial, and secure communications serve as a vital safeguard. By ensuring the confidentiality, integrity, and availability of client data, secure communications establish a foundation of trust and elevate the organization's reputation.

Artemis Financial, in its consulting services involving international transactions, faces security considerations related to potential threats and risks of financial data breaches and fraudulent activities. To address these concerns, the company must prioritize compliance with international regulations and establish secure practices for data transmission and handling across various networks and jurisdictions. By implementing secure communication channels, encryption protocols, and adhering to data protection regulations, Artemis Financial can protect the integrity and confidentiality of data during international transactions, preventing unauthorized access and maintaining client trust.

Considering secure communications, Artemis Financial must also factor in governmental restrictions, which vary across different countries. These restrictions may encompass encryption standards, data localization requirements, and privacy regulations. Compliance with these regulations is crucial to ensure the legality and security of international transactions.

Artemis Financial's web-based software application is constantly exposed to a variety of external threats, both existing and potential. The organization faces significant risks from cyber attacks that exploit vulnerabilities to gain unauthorized access, steal sensitive financial data, or disrupt the application's availability. Phishing and social engineering tactics also pose dangers as they can deceive employees and clients, resulting in unintentional disclosure of confidential information and compromising the application's security. Data breaches, denial of service (DoS) attacks, and insider threats further contribute to the overall risks, potentially compromising the organization's data security and operational stability.

To meet Artemis Financial's modernization requirements, it is crucial to consider the role of open-source libraries and effectively manage them to avoid security vulnerabilities. Regular monitoring, updates, and patch management are necessary for maintaining the security of these libraries. Additionally, adopting secure development frameworks, following secure coding practices, and staying up-to-date with technology versions are essential in the evolving landscape of web application technologies. Conducting security audits, penetration testing, and code reviews helps identify vulnerabilities and weaknesses in the application. Employee training and awareness programs are also vital to educate and cultivate a security-conscious workforce, mitigating risks from human error and social engineering attacks. By addressing these considerations, Artemis Financial can enhance its overall security posture during the modernization process.

## Areas of Security

**Input Validation**

Input validation is a critical area of security that holds relevance for Artemis Financial's web application. Implementing proper input validation measures ensures that all user-provided data is thoroughly examined and validated, significantly reducing the risk of malicious code execution, data manipulation, and unauthorized access to sensitive information. This process involves checking the format, length, and content of user input, as well as applying necessary sanitization techniques to remove potentially harmful or malicious content. By validating and sanitizing user input, input validation acts as a safeguard against prevalent attacks like SQL injection, cross-site scripting (XSS), and command injection. This process minimizes the chances of data manipulation and unauthorized access to enhance security.

**API security**

Secure API interactions are crucial for Artemis Financial, given their utilization of a RESTful web API. The security of these interactions plays a vital role in protecting sensitive data transmitted between different systems and components. To achieve this, implementing robust authentication and authorization mechanisms is essential. By ensuring that only authorized users and applications can access and interact with the API, the organization can mitigate the risk of unauthorized access, data breaches, and malicious attacks.

In addition to authentication and authorization, implementing secure communication protocols, such as HTTPS, is important to safeguard the confidentiality and integrity of the transmitted data. This ensures that sensitive financial information remains protected during transit. It is also crucial to protect against API abuse, such as API key exposure or unauthorized access to sensitive resources. By implementing proper API security measures, Artemis Financial can establish a secure and trusted communication channel, preventing unauthorized access and ensuring the integrity of their web application's data.

**Cryptography**

The implementation of robust encryption measures is critical for safeguarding the confidentiality of sensitive financial information, including financial data, customer records, and communication channels, within Artemis Financial's web application. By utilizing strong encryption algorithms, ensuring proper key management, and addressing potential cryptographic vulnerabilities, such as weak ciphers or inadequate key sizes, the organization can establish a secure environment, even in the event of data compromise. This reinforces the protection of valuable data assets and enhances the overall security posture of the application.

**Client/Server Security**

Ensuring the security of client/server interactions is of utmost importance for Artemis Financial's application. It requires the implementation of secure communication protocols such as HTTPS, along with client-side and server-side verification. These measures effectively protect the data exchanged between clients and servers and prevent unauthorized interception, unauthorized alteration, and falsification attacks. To establish a secure distributed composing environment, Artemis Financial should prioritize secure session management, implement robust communication protocols, and take proactive measures to protect against prevalent vulnerabilities such as cross-site scripting (XSS) and cross-site request forgery (CSRF).

**Code Error Handling**

Code error handling is crucial for Artemis Financial's web application as it prevents the exposure of sensitive information and potential security vulnerabilities. By effectively handling errors, such as employing graceful error messages and avoiding detailed error stack traces, the risk of information leakage is reduced. This, in turn, mitigates the potential exploitation of code errors by attackers seeking insights into the system's architecture or implementation.

## Manual Review

First of all, there is no explicit input validation implemented in the code. It is important to validate and sanitize user input, especially in the CRUDController class where the business\_name parameter is accepted as a request parameter. Lack of input validation can lead to security risks such as SQL injection or cross-site scripting (XSS) attacks. In the customer class, the "deposit" method currently accepts an integer "a" as an argument. However, it lacks input validation to ensure that the deposit amount is greater than zero. It is important to validate the input to ensure that a deposit is always a positive value.

From Client/Server Security perspective, the code does not explicitly show the implementation of secure communication protocols (e.g., HTTPS) between the client and server components. It is important to establish secure communication channels to protect sensitive data from spoofing attacks, eavesdropping and tampering.

In terms of API security, the code in the code base lacks a comprehensive implementation of authentication or authorization measures to secure the API endpoints. It is crucial to integrate authentication and authorization mechanisms to establish a secure framework, allowing only authorized users to access and interact with the API endpoints, thus preventing unauthorized access and potential security breaches.

Last but not least, a vulnerability has been found in the DocData class related to the data access method, specifically the use of hard-coded credentials (root and root) for establishing the database connection. Storing credentials directly in the code is considered insecure as it exposes sensitive information and heightens the possibility of unauthorized access. To ensure proper security measures, it is important to adopt secure configuration practices, such as utilizing external configuration files or environment variables, to securely store and manage credentials. Moreover, the use of default or easily predictable credentials like “root” significantly amplifies the risk of attacks and should be avoided.

## Static Testing

The Dependency Check Report has identified a total of 13 vulnerabilities, with 6 of them being critical severity. It is crucial to address and patch these vulnerabilities to ensure the security of your application.

**bcprov-jdk15on-1.46.jar**

The Bouncy Castle JCE Provider versions 1.55 and earlier have multiple security vulnerabilities. These include incomplete validation of ASN.1 encoding in DSA and ECDSA signatures, weak private key generation in DSA, support for unsafe ECB mode in DHIES and ECIES implementations, timing attack vulnerability in DSA signature generation, padding oracle attack vulnerability in DHIES/ECIES CBC mode, weak Bleichenbacher oracle in RSA key exchange, timing issue exposing private key information in EC math library, incorrect cryptographic algorithm selection, leakage of AES key information, observable differences leading to private exponent exposure in RSA encryption, LDAP injection vulnerability, invalid curve attack vulnerability, integrity compromise in default BKS keystore, timing side-channel attacks in TLS implementation, and incomplete validation of other party DH public key.

These vulnerabilities could potentially result in data manipulation, key compromise, information disclosure, and plaintext recovery attacks. It is crucial to update to a secure version (1.56 or later) of the Bouncy Castle library to mitigate these risks.

**hibernate-validator-6.0.18.Final.jar**

An issue was discovered in Hibernate Validator version 6.1.2.Final. Due to a bug in the message interpolation processor, incorrect EL (Expression Language) expressions can be treated as valid, leading to a vulnerability. Exploiting this flaw enables attackers to circumvent input sanitation measures (such as escaping or stripping) implemented by developers when dealing with user-supplied data in error messages.

**jackson-databind-2.10.2.jar**

Several vulnerabilities have been found in FasterXML Jackson Databind, a general data-binding library for Jackson. These vulnerabilities include insecure entity expansion, allowing for XML external entity (XXE) attacks and potential data integrity issues. The library is also susceptible to Java StackOverflow exceptions and denial-of-service attacks due to excessive nested objects or deep wrapper array nesting.

Additionally, resource exhaustion can occur in certain situations involving deeply nested arrays during deserialization. While there is some dispute about the impact of a vulnerability related to cyclic dependencies, it highlights the risk of using untrusted input with the library. These issues fall under the CWE-502 category, which pertains to the deserialization of untrusted data.

**log4j-api-2.12.1.jar**

A vulnerability was discovered in the Apache Log4j SMTP appender where it improperly validates certificates with host mismatches. This flaw could be exploited by a man-in-the-middle attacker to intercept SMTPS connections and potentially expose any log messages transmitted through the affected appender. The issue has been resolved in Apache Log4j versions 2.12.3 and 2.13.1.

**logback-core-1.2.3.jar**

A security vulnerability was identified in logback versions 1.2.7 and earlier. If an attacker possesses the necessary privileges to modify configuration files, they could create a malicious configuration that enables the execution of arbitrary code retrieved from LDAP servers.

**snakeyaml-1.25.jar**

Multiple vulnerabilities have been identified in SnakeYaml versions prior to 2.0. The Constructor() class lacks proper type restrictions during deserialization, allowing remote code execution when handling YAML content provided by an attacker.

Additionally, the Alias feature in earlier versions enables entity expansion, leading to potential security issues. Furthermore, versions before 1.31 suffer from a Denial of Service (DoS) vulnerability due to a missing nested depth limitation for collections. It is recommended to use SnakeYaml's SafeConstructor for parsing untrusted content and to upgrade to version 2.0 or newer.

**spring-boot-2.2.4.RELEASE.jar**

There have been vulnerabilities identified in different versions of Spring Boot. In versions prior to v2.2.11.RELEASE, there was a temporary directory hijacking vulnerability in the AbstractConfigurableWebServerFactory.createTempDir method, but this only affects unsupported versions. Additionally, in versions 3.0.0 - 3.0.6, 2.7.0 - 2.7.11, 2.6.0 - 2.6.14, 2.5.0 - 2.5.14, and older unsupported versions, there is a potential denial-of-service (DoS) vulnerability when using Spring MVC with a reverse proxy cache.

**spring-boot-starter-web-2.2.4.RELEASE.jar**

Spring Boot is a framework for building web applications, particularly RESTful ones, using Spring MVC and Tomcat as the default embedded container. However, there have been identified vulnerabilities in certain versions. In versions prior to v2.2.11.RELEASE, there was a vulnerability related to temporary directory hijacking in the org.springframework.boot.web.server.AbstractConfigurableWebServerFactory.createTempDir method, but this only affects unsupported versions. Additionally, in versions 3.0.0 - 3.0.6, 2.7.0 - 2.7.11, 2.6.0 - 2.6.14, 2.5.0 - 2.5.14, and older unsupported versions, there is a potential risk of denial-of-service (DoS) attacks when using Spring MVC with a reverse proxy cache.

**spring-core-5.2.3.RELEASE.jar**

There are multiple vulnerabilities identified in different versions of Spring Framework. In some versions, a Spring MVC or Spring WebFlux application running on JDK 9+ is vulnerable to remote code execution (RCE) via data binding. There is also a vulnerability in WebFlux applications that allows a locally authenticated user to read, modify, or overwrite files in the application's temporary storage directory. Additionally, there are vulnerabilities related to privilege escalation, bypassing RFD attack protections, denial-of-service (DoS) conditions, and log manipulation. It is recommended to apply updates as per the vendor's instructions to address these vulnerabilities.

**spring-web-5.2.3.RELEASE.jar**

This version of Spring Framework has potential remote code execution (RCE) issues, privilege escalation, denial-of-service (DoS) attacks, bypassing security protections, and log manipulation.

**spring-webmvc-5.2.3.RELEASE.jar**

Multiple vulnerabilities have been identified in different versions of the Spring Framework's web components, including Spring MVC and Spring WebFlux. These vulnerabilities include potential remote code execution (RCE), privilege escalation, denial-of-service (DoS) attacks, bypassing security protections, and log manipulation. It is crucial to apply updates according to the vendor's instructions.

**tomcat-embed-core-9.0.30.jar**

Core Tomcat vulnerabilities include remote code execution (RCE), denial-of-service (DoS) attacks, request smuggling, HTTP header parsing issues, session cookie transmission over insecure channels, and concurrency bugs.

Similar to Spring Framework, updates per the vendor's instructions are necessary.

**tomcat-embed-websocket-9.0.30.jar**

Core Tomcat’s vulnerabilities include improper privilege management, denial-of-service attacks, request smuggling, cross-site scripting (XSS), information leakage, and memory-related issues.

To mitigate these vulnerabilities, updates according to vendor’s instructions are imporant. The vulnerabilities affect various versions of Apache Tomcat, and some may require configuration changes to address them. Care should be taken with AJP connections, as they are treated with higher trust and can be exploited. Additionally, there are vulnerabilities related to WebSocket frame validation, HTTP/2 requests, TLS packet validation, JNDI Realm authentication, and the handling of HTTP headers.

## Mitigation Plan

**Input Validation**

Use input validation techniques such as whitelisting, blacklisting, or regular expressions to ensure that the input is safe and does not contain any malicious content. For the CRUDController class, we can use Spring's validation annotations. For instance, adding @Validated annotation at the class level will enable validation support, or @NotBlank annotation is used to ensure that the "business\_name" parameter is not null or empty. Additionally, we can also add @Size annotation to limit the size of the "business\_name" parameter to a maximum of 50 characters or any size based on the requirements.

By using these validation annotations, we ensure that the "business\_name" parameter passed in the request is validated for not being blank and within the specified size limit. If the validation fails, Spring will automatically return a validation error response to the client.

For the customer class, we can add a condition to check if the amount parameter is greater than zero before updating the accountBalance. If the amount is zero or negative, an IllegalArgumentException is thrown. By adding this validation check, we ensure that the deposit amount provided to the deposit method is always a positive value. If a non-positive value is passed, an exception will be thrown, indicating the invalid input.

**API security**

To address the API security issues and implement authentication and authorization measures, we can utilize a widely adopted authentication framework like JSON Web Tokens (JWT) along with role-based access control.

**Data Access Credentials**

We can update the application to load the credentials from the configuration file or environment variables. The method depends on the chosen framework or configuration management approach. In this case we can utilize Spring Boot to separate the sensitive credentials from the code and store them in a securely.

**bcprov-jdk15on-1.46.jar**

Based on the information available on MVN Repository, version 1.70 was released on December 01, 2021 and does not contain any reported vulnerabilities until present (“Bouncy Castle Provider,” n.d.). Updating to this version would be a prudent step to ensure the software remains secure and up to date.

**hibernate-validator-6.0.18.Final.jar**

Although Version 8.0.0 is not the latest release, it is worth noting that there have been no reported vulnerabilities for this version. Additionally, it has been widely adopted and has garnered significant usage since 2020. Therefore, considering its stability and widespread usage, Version 8.0.0 might be a reliable choice for the software (“Hibernate Validator Engine,” n.d.).

**jackson-databind-2.10.2.jar**

Version 2.15.2 released on May 31, 2023 is the latest and has no vulnerability at the moment so it is recommended to update Jackson Databind to this version.

**log4j-api-2.12.1.jar**

As of its official release date in February 21, 2023, Version 2.20.0 has demonstrated a clean track record with no reported vulnerabilities (“Apache Log4j API,” n.d.). Given this information, it is highly recommended to consider using this version to patch any vulnerabilities associated with the dependency. **logback-core-1.2.3.jar**

Both version 1.4.7 and 1.4.8 have not been reported to have any vulnerabilities. Both versions were released in 2023. While 1.4.7 has more usage since it is an older version, both versions can be considered as strong candidates for our solution options. Their clean vulnerability records and recent release dates make them reliable choices for our software solution.

**snakeyaml-1.25.jar**

Unfortunately, all previous versions of SnakeYAML have been found to contain vulnerabilities, with the exception of the latest release, version 2.0 (“SnakeYAML,” n.d.). Therefore, it is highly recommended that we prioritize updating to the latest version. By doing so, we can ensure that our software remains secure and free from any known vulnerabilities.

**spring-boot-2.2.4.RELEASE.jar**

All versions within the 3.1.x series, which were released in 2023, have not encountered any reported vulnerabilities thus far (“Spring Boot,” n.d.). This makes them suitable candidates for updating our dependencies.

**spring-boot-starter-web-2.2.4.RELEASE.jar**

The same with Spring Boot, versions 3.1.x are latest and might be the best choice.

**spring-core-5.2.3.RELEASE.jar**

Among the available versions, 6.0.10, 6.0.9, and 6.0.8 stand out as the most recent releases that have no reported vulnerabilities. These versions provide a secure option for our dependency update.

**spring-web-5.2.3.RELEASE.jar**

All 6.9.x are secure options for our application as they do not have any vulnerabilities.

**spring-webmvc-5.2.3.RELEASE.jar**

Since March 2022, Spring Web MVC has maintained stable versions. You have several good options to choose from, particularly version 6.0.9, which has gained significant popularity and usage among the latest releases. Considering the stability and widespread adoption, it is recommended to consider version 6.0.9 or any subsequent versions for your project.

**tomcat-embed-core-9.0.30.jar**

11.0.x versions are out in the market since 2023 and seem to be stable enough to be our options.

**tomcat-embed-websocket-9.0.30.jar**

The 11.0.x versions of the software have been proven to be stable and have not encountered any reported issues. Among these versions, the latest release is 11.0.0-M9. It is important to consider using this latest version.

**References**

*Bouncy Castle Provider.* (n.d.). MVN Repository. <https://mvnrepository.com/artifact/org.bouncycastle/bcprov-jdk15on>

*Hibernate Validator Engine.* (n.d.). MVN Repository. <https://mvnrepository.com/artifact/org.hibernate.validator/hibernate-validator>

*Apache Log4j API*. (n.d.). MVN Repository. <https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-api>

*SnakeYAML*. (n.d.). MVN Repository. <https://mvnrepository.com/artifact/org.yaml/snakeyaml>

*Spring Boot.* (n.d.). MVN Repository. <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot>