****

# 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** | **11/10/2022** | **Tim Brady** | **Initial Analysis** |

## 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

Tim Brady

## Interpreting Client Needs

As part of a project to modernize their operations, Artemis Financial would like to update the software security for their RESTful web application programming interface. As a financial institution, Artemis Financial has access to delicate information related to their customers’ financials. This is extremely crucial as the mishandling of data can lead to monetary theft as well as identity theft. The customer is not the only one at risk with this type of application. Other financial institutions all around the world can be affected in the case of international clients or international transactions between U.S. clients. To help guard the information of customers and institutions the system must be protected from various forms of threat from brute forcing passwords to malware and phishing. All of this must be addressed while adhering to specific governmental restrictions that are in place specifically for financial institutions like this one. Many actions can be taken that will lead to more secure data. Two of the easiest are two-factor authentication and ensuring up-to-date versions of things like programs and libraries.

## Areas of Security

The areas of security that are applicable to Artemis Financial’s software application are as follows:

* **Input Validation** – A financial app will have the ability for user input that ranges from login info to banking info. This input will need to be validated.
* **APIs –** The application is a RESTful web application, and the API should be secure.
* **Cryptography –** Financial data should always be encrypted to avoid being compromised. “Required by [ISO/IEC 27001](https://www.ekransystem.com/en/solutions/meeting-compliance-requirements/iso-compliance-solution), GLBA, GDPR, PCI DSS, and other standards and regulations, encryption is an efficient way to secure your data.” (Ekran, 2022).
* **Code Error –** Coding of the software must be validated for secure error handling.
* **Code Quality –** The quality of the code should follow secure coding practices and ensure that things like public and private methods are being used appropriately.
* **Encapsulation –** Data structures must be secure. Encapsulation is a way to ensure people cannot directly change code.

## Manual Review

Upon manual review, the following vulnerabilities were identified:

* Greeting.java contains a @GetMapping which will leak over HTTP referrer headers. A more secure request would be to use @PostMapping.
* There does not appear to be any input validation which leaves the system open to malicious data.

## Static Testing

Running a dependency test returned the following vulnerabilities:

* **CVE-2020-10693** - 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.
* **CVE-2022-42004** - In FasterXML jackson-databind before 2.13.4, resource exhaustion can occur because of a lack of a check in BeanDeserializer.\_deserializeFromArray to prevent use of deeply nested arrays. An application is vulnerable only with certain customized choices for deserialization.
* **CVE-2021-44832** - Apache Log4j2 versions 2.0-beta7 through 2.17.0 (excluding security fix releases 2.3.2 and 2.12.4) are vulnerable to a remote code execution (RCE) attack when a configuration uses a JDBC Appender with a JNDI LDAP data source URI when an attacker has control of the target LDAP server. This issue is fixed by limiting JNDI data source names to the java protocol in Log4j2 versions 2.17.1, 2.12.4, and 2.3.2.
* **CVE-2021-42550** - 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.
* **CVE-2022-38752 -** Using snakeYAML to parse untrusted YAML files may be vulnerable to Denial of Service attacks (DOS). If the parser is running on user supplied input, an attacker may supply content that causes the parser to crash by stack-overflow.
* **CVE-2022-38751** - Using snakeYAML to parse untrusted YAML files may be vulnerable to Denial of Service attacks (DOS). If the parser is running on user supplied input, an attacker may supply content that causes the parser to crash by stackoverflow.
* **CVE-2022-27772** - \*\* UNSUPPORTED WHEN ASSIGNED \*\* spring-boot versions prior to version v2.2.11.RELEASE was vulnerable to temporary directory hijacking. This vulnerability impacted the org.springframework.boot.web.server.AbstractConfigurableWebServerFactory.createTempDir method. NOTE: This vulnerability only affects products and/or versions that are no longer supported by the maintainer.
* **CVE-2022-22971** - In spring framework versions prior to 5.3.20+ , 5.2.22+ and old unsupported versions, application with a STOMP over WebSocket endpoint is vulnerable to a denial of service attack by an authenticated user.
* **CVE-2022-42252** - If Apache Tomcat 8.5.0 to 8.5.52, 9.0.0-M1 to 9.0.67, 10.0.0-M1 to 10.0.26 or 10.1.0-M1 to 10.1.0 was configured to ignore invalid HTTP headers via setting rejectIllegalHeader to false (the default for 8.5.x only), Tomcat did not reject a request containing an invalid Content-Length header making a request smuggling attack possible if Tomcat was located behind a reverse proxy that also failed to reject the request with the invalid header.

## Mitigation Plan

Using current versions of software is crucial to having a more secure product. Updates often come with corrections to security vulnerabilities as they are discovered. More than half of the vulnerabilities listed above can be addressed with updates. As highlighted above in the manual review section, there are also vulnerabilities with the user input and the request mapping. User input can be limited to certain characters as well as lengths to prevent malicious data being passed through. The request mapping should always use @PostMapping instead of @GetMapping when handling sensitive data online. Making these changes should resolve the vulnerabilities outlined.

**References**

*7 best practices for banking and financial cybersecurity compliance*. Banking & Financial Data Security Compliance: 12 Best Practices | Ekran System. (2022, September 29). Retrieved November 11, 2022, from https://www.ekransystem.com/en/blog/banking-and-financial-cyber-security-compliance