****

# 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** | **7-13-23** | **Nicholas Nevins** |  |

## 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

Nicholas Nevins

## Interpreting Client Needs

As a financial institution, Artemis Financial should consider secure communications paramount to success. Their entire business involves holding and moving money, as well as servers and servers worth of sensitive information. Besides the actual value of the finances and consumer information, the firm’s reputation is a massive asset. If we are lax about secure communication, customers will cease to trust Artemis Financial and that would be disastrous to their company. Suffice it to say, secure communications are vital to Artemis Financial.

Artemis Financial is likely to engage in transactions both nationally and internationally, and as such there is added complexity and opportunity for failure. While I am unaware of any federal regulations specifically restricting their secure communications, the Gramm-Leach-Bliley Act (GLBA) requires financial institutions like Artemis Financial to ensure that the sensitive data of their company and customers are protected. I personally would rather be on the “too protected” side rather than the “too little protected” one.

External threats will always be a risk, and as a financial institution they will be particularly targeted. Not only do they control assets digitally, they also house extremely valuable data about their customers. This sensitive data must be protected. It can do great harm to individuals and businesses if it is not. Attacks like DoS, phishing, and error or deficient security utilization are some of the more common threats they might face. Phishing in particular needs to be a matter of concern and training, as many companies are targeted by these attempts.

APIs and web application technologies are wonderful tools, and we should embrace them fully. We should utilize them in ways that allow for easy and efficient upgrading, patching, and securing. However, we need to be vigilant with these tools. As products of the internet, they carry inherent risks that must be properly mitigated. They provide added connectivity from third parties to our system, and potentially enable more points of attack for threats. We have to be extremely careful about which APIs and tools we use and ensure that they are implemented in ways that are safe, secure, and as the program was intended to be used.

## Areas of Security

* Input Validation
  + Improperly validated input is a concern for any system, as even non-malicious entries could cause harm to the system. We have to ensure that inputs are valid and regulated to prevent unintended reactions by the system or malicious code being used and leveraged against the system through the input.
* Secure APIs
  + Since we will be relying on APIs, we need to ensure that the ones we pick are up to date, secure, and used properly. APIs can introduce third parties to our system, and we need to be sure that we are careful with the access that we give outside parties either intentionally or unintentionally.
* Cryptography
  + As we find ourselves in the field of finances and sensitive information, cryptography is a must. It is essential that sensitive information, such as social security addresses, birth days, physical addresses, and so on can never be sent without encryption. If, somehow, an attacker ends up with some of our transmissions, it is imperative that it be unusable to them.
* Code Error
  + Our code, as always, should be error free and should handle errors it could reasonably encounter properly. If it doesn’t, we could see the system vulnerable in ways we hadn’t intended. While we implement APIs, we should do so properly and ensure that our own code utilizes them in a safe, accurate manner.
* Code Quality
  + Code quality is always a concern. We need to produce code that is properly encapsulated and as condensed and streamlined as possible. By following quality standards, we limit the amounts of errors and complexities in the system that might leave us more vulnerable than we realized.

## Manual Review

Upon inspection, it does not appear that any input validation is in use. Very little error handling is present, save a single catch and release process in DocData.java. There is also no obvious ability to utilize cryptography, and that is a major flaw in this system. We need to ensure that we are encrypting user information. For instance, CRUDController.java sends the business name directly as a parameter, which is a vulnerability. Utilizing HTTPS is a good idea that we are not using for data transmission currently.

## Static Testing

|  |  |  |
| --- | --- | --- |
| **Dependency** | **Severity** | **Count** |
| tomcat-embed-websocket-9.0.30.jar | Critical | 21 |
| tomcat-embed-core-9.0.30.jar | Critical | 20 |
| spring-web-5.2.3.RELEASE.jar | Critical | 12 |
| spring-webmvc-5.2.3.RELEASE.jar | Critical | 11 |
| spring-core-5.2.3.RELEASE.jar | Critical | 11 |
| snakeyaml-1.25.jar | Critical | 8 |
| bcprov-jdk15on-1.46.jar | High | 18 |
| jackson-databind-2.10.2.jar | High | 6 |
| spring-boot-2.2.4.RELEASE.jar | High | 2 |
| spring-boot-starter-web-2.2.4.RELEASE.jar | High | 2 |
| hibernate-validator-6.0.18.Final.jar | Medium | 1 |
| logback-core-1.2.3.jar | Medium | 1 |
| log4j-api-2.12.1.jar | Low | 1 |

* tomcat-embed-websocket-9.0.30.jar
  + Apache Tomcat treats Apache JServ Protocol (AJP) connections as having higher trust than, for example, a similar HTTP connection. If such connections are available to an attacker, they can be exploited.
  + Apply Updates
* tomcat-embed-core-9.0.30.jar
  + Apache Tomcat treats Apache JServ Protocol (AJP) connections as having higher trust than, for example, a similar HTTP connection. If such connections are available to an attacker, they can be exploited.
  + Apply Updates
* spring-web-5.2.3.RELEASE.jar
  + Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding.
  + Apply Updates
* spring-webmvc-5.2.3.RELEASE.jar
  + Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding.
  + Apply Updates
* spring-core-5.2.3.RELEASE.jar
  + Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding.
  + Apply Updates
* snakeyaml-1.25.jar
  + 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. We recommend using SnakeYaml's SafeConsturctor when parsing untrusted content to restrict deserialization. We recommend upgrading to version 2.0 and beyond.
  + Apply Updates
* bcprov-jdk15on-1.46.jar
  + In Bouncy Castle JCE Provider version 1.55 and earlier the DSA does not fully validate ASN.1 encoding of signature on verification. It is possible to inject extra elements in the sequence making up the signature and still have it validate, which in some cases may allow the introduction of 'invisible' data into a signed structure.
  + Apply Updates
* jackson-databind-2.10.2.jar
  + 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.
  + Apply Updates
* spring-boot-2.2.4.RELEASE.jar
  + 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.
  + Apply Updates
* spring-boot-starter-web-2.2.4.RELEASE.jar
  + 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.
  + Apply Updates
* hibernate-validator-6.0.18.Final.jar
  + 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
  + Apply Updates
* logback-core-1.2.3.jar
  + 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.
  + Apply Updates
* log4j-api-2.12.1.jar
  + 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
  + Apply Updates

## Mitigation Plan

When all is said and done, our main mitigation plan involves developing the software to the standards and practices we should already be implementing. In more detail, I think the following steps are areas that should be our biggest concern.

* Ensure that all input is being properly validated throughout the entire system
* Ensure that sensitive data is being properly encrypted before being sent
* Ensure that we are using a secure communication protocol, such as HTTPS
* Ensure that all APIs and dependencies are up to date and properly supported by the developer. If they are not, we need to examine alternate APIs to fulfill our needs
* Ensure that code is developed to test, and that we are properly error testing our code

I feel that if we follow these guidelines, we’ll be very close to a secure, stable system for Artemis Financial.