****

# 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** | **09/18/2022** | **Nicolas Blaisdell** | **Initial Draft** |

## 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

Nicolas Blaisdell

## Interpreting Client Needs

For the client, Artemis Financial, secure communications, among other modern security solutions, are of great importance since they are involved with individual’s financial information which they’ll want to keep secure, as well as the fact that they are reaching out to our software engineering company, Global Rain, which specializes in software for businesses with security at front-of-mind.

While the prompt only specifies that they are working with individualized financial plans, it doesn’t necessarily say if the transactions will be international or not. However, since people’s lives and families can vary greatly, as well as them reaching out to our company which works with businesses around the world, it’s probably safe to assume that they will likely be making international transactions.

There are no governmental restrictions mentioned regarding how communications should be secured in the application/software. However, security measures to keep those communications safe at rest and in transit throughout the application will be considered first and foremost while developing the application anyway.

There are many external threats that exist now and will exist in the future, regarding the data and communications held within Artemis Financial’s software. There are unfortunately many bad actors in the world which use their knowledge for stealing information, whether to attempt to steal the actual money in some way by intercepting the data communication by the software, or perhaps with the intention of stealing the data and trying to sell that to other third parties. Lastly, it could just be out of boredom and to simply wreak havoc on some company/piece of software. For these reasons, and many more not mentioned here, the security of this application must be one of the most important aspects, to keep the sensitive financial data safe.

As a result of this application wanting to stay up-to-date and using modern solutions and technology, it will most likely consist of using one or more third-party dependencies, or code written by somebody other than the original developers used within the software being developed. Because this was written elsewhere, it’s even more important that the code is tested thoroughly, both manually and with testing tools, to make sure there aren’t any vulnerabilities that might compromise our application by including it. Most of the time, these issues can be mitigated by making sure any third-party software included in our application is upgrade to the latest version.

## Areas of Security

After considering Artemis Financial’s use-case, reviewing the codebase for the API they have already developed, and consulting the Vulnerability Assessment Process Flow Diagram, it appears that the following Areas of Security are relevant and will apply to the software being created for them:

* *Input Validation*
  + This is crucial, as we’ll want to make sure that only the data that’s expected to come into our application/API is what gets processed, and anything else is rejected
  + This will help to avoid things like SQL Injection, which might allow a hacker to access or manipulate the data held within the application
  + It will also help less malicious, but still bad, results such as incorrect output coming from the API
* *APIs*
  + While there will be a front-end part of this software, the main aspect is the API which handles all the work in the backend and serves the data back to its users. As a result, the “APIs” Area of Security will be of great relevance, and the security around its use should be considered with great care.
  + This will include making sure the endpoints which are made public are only accessible by those with the proper authorization, and the endpoints which are private and internal to the API itself are not accessible by the public.
* *Cryptography*
  + Since there is a potential for data to be sent internationally, Cryptography will be of relevance to the software, so that the appropriate regulations for data security as followed.
  + This also means that the data should always be encrypted, unless absolutely required, in which case, it should be decrypted at that point, and not before, so the data stays encrypted at rest and in transit as much as possible.
* *Client/Server*
  + Once the application exists, the communication between that software and the API it interacts with will need to be secured as well. Only as much information as necessary should be sent to the API, so extra details aren’t leaked to any potential external threats reading those communications.
  + Even without that part of the software, the public will still be able to access the API, and so the communication between those users will need to be secured, as well.
* *Code Error*
  + Together with input validation, any invalid input or other unexpected errors need to be handled gracefully, and in conjunction with the user’s authorization.
  + The lack of correct error handling might lead to vulnerabilities in our own software, allowing hackers to exploit this.

## Manual Review

With the Areas of Security in mind, I started the Manual Review of the codebase containing the API for Artemis Financial. The first place I checked was the pom.xml file, which includes references to all the dependencies used within the application. Since nothing was mentioned in the prompt directly relating to any specific dependency, it appears that the file was in good condition, at least for now.

In the GreetingController.java file, there is an example where input validation needs to be considered, and currently there is no validation being done on the data in this file. The file consists of a single API endpoint “/greeting”, which optionally accepts a “name” parameter, and is used directly and returned to the user in a JSON object, with the message “Hello {name}”, where {name} is replaced with the user’s input, or “World”, if left blank. Because there is no validation done on this input, it could potentially be used by hackers in a SQL Injection attack to compromise our application or the data held within it.

For the “/greeting” and “/read” endpoints, they are both completely open to the public with no authorization, so that should also be considered for this, and any additional endpoints that are created. Additionally, the parameters that are passed to the API are given in plain text directly in the URL. That information will be saved to the browser, which could be used by external threats.

## Static Testing

After integrating the dependency-check plugin into the API codebase, and analyzing the results from the Dependency Check Report, the following vulnerabilities were discovered:

|  |  |  |  |
| --- | --- | --- | --- |
| **Dependency** | **Vulnerability IDs** | **Description** | **Solution** |
| bcprov-jdk15on-1.46.jar | [cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.46:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Abouncycastle&cpe_product=cpe%3A%2F%3Abouncycastle%3Alegion-of-the-bouncy-castle-java-crytography-api&cpe_version=cpe%3A%2F%3Abouncycastle%3Alegion-of-the-bouncy-castle-java-crytography-api%3A1.46) | In Legion of the Bouncy Castle BC before 1.61 and BC-FJA before 1.0.1.2, attackers can obtain sensitive information about a private exponent because of Observable Differences in Behavior to Error Inputs. This occurs in org.bouncycastle.crypto.encodings.OAEPEncoding. Sending invalid ciphertext that decrypts to a short payload in the OAEP Decoder could result in the throwing of an early exception, potentially leaking some information about the private exponent of the RSA private key performing the encryption. | Upgrade to latest version |
| spring-boot-2.2.4.RELEASE.jar | [cpe:2.3:a:vmware:spring\_boot:2.2.4:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Avmware&cpe_product=cpe%3A%2F%3Avmware%3Aspring_boot&cpe_version=cpe%3A%2F%3Avmware%3Aspring_boot%3A2.2.4) [cpe:2.3:a:vmware:spring\_framework:2.2.4:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Avmware&cpe_product=cpe%3A%2F%3Avmware%3Aspring_framework&cpe_version=cpe%3A%2F%3Avmware%3Aspring_framework%3A2.2.4) | \*\* 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. | Upgrade to latest version.  Also, from GitHub (<https://github.com/JLLeitschuh/security-research/security/advisories/GHSA-cm59-pr5q-cw85>):  *Setting the java.io.tmpdir system environment variable to a directory that is exclusively owned by the executing user will fix this vulnerability for all operating systems.* |
| logback-core-1.2.3.jar | [cpe:2.3:a:qos:logback:1.2.3:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aqos&cpe_product=cpe%3A%2F%3Aqos%3Alogback&cpe_version=cpe%3A%2F%3Aqos%3Alogback%3A1.2.3) | 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. | Upgrade to latest version |
| log4j-api-2.12.1.jar | [cpe:2.3:a:apache:log4j:2.12.1:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aapache&cpe_product=cpe%3A%2F%3Aapache%3Alog4j&cpe_version=cpe%3A%2F%3Aapache%3Alog4j%3A2.12.1) | 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. | Upgrade to latest version |
| snakeyaml-1.25.jar | [cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Asnakeyaml_project&cpe_product=cpe%3A%2F%3Asnakeyaml_project%3Asnakeyaml&cpe_version=cpe%3A%2F%3Asnakeyaml_project%3Asnakeyaml%3A1.25) [cpe:2.3:a:yaml\_project:yaml:1.25:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Ayaml_project&cpe_product=cpe%3A%2F%3Ayaml_project%3Ayaml&cpe_version=cpe%3A%2F%3Ayaml_project%3Ayaml%3A1.25) | 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. | Upgrade to latest version |
| jackson-databind-2.10.2.jar | [cpe:2.3:a:fasterxml:jackson-databind:2.10.2:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Afasterxml&cpe_product=cpe%3A%2F%3Afasterxml%3Ajackson-databind&cpe_version=cpe%3A%2F%3Afasterxml%3Ajackson-databind%3A2.10.2) | jackson-databind before 2.13.0 allows a Java StackOverflow exception and denial of service via a large depth of nested objects. | Upgrade to latest version |
| tomcat-embed-core-9.0.30.jar | [cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aapache&cpe_product=cpe%3A%2F%3Aapache%3Atomcat&cpe_version=cpe%3A%2F%3Aapache%3Atomcat%3A9.0.30) [cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aapache_tomcat&cpe_product=cpe%3A%2F%3Aapache_tomcat%3Aapache_tomcat&cpe_version=cpe%3A%2F%3Aapache_tomcat%3Aapache_tomcat%3A9.0.30) | The refactoring present in Apache Tomcat 9.0.28 to 9.0.30, 8.5.48 to 8.5.50 and 7.0.98 to 7.0.99 introduced a regression. The result of the regression was that invalid Transfer-Encoding headers were incorrectly processed leading to a possibility of HTTP Request Smuggling if Tomcat was located behind a reverse proxy that incorrectly handled the invalid Transfer-Encoding header in a particular manner. Such a reverse proxy is considered unlikely. | Upgrade to latest version |
| spring-boot-starter-validation-2.2.4.RELEASE.jar | [cpe:2.3:a:vmware:spring\_boot:2.2.4:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Avmware&cpe_product=cpe%3A%2F%3Avmware%3Aspring_boot&cpe_version=cpe%3A%2F%3Avmware%3Aspring_boot%3A2.2.4) | \*\* 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. | Upgrade to latest version |
| hibernate-validator-6.0.18.Final.jar | [cpe:2.3:a:redhat:hibernate\_validator:6.0.18:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aredhat&cpe_product=cpe%3A%2F%3Aredhat%3Ahibernate_validator&cpe_version=cpe%3A%2F%3Aredhat%3Ahibernate_validator%3A6.0.18) | 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. | Upgrade to latest version |
| spring-core-5.2.3.RELEASE.jar | [cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Apivotal_software&cpe_product=cpe%3A%2F%3Apivotal_software%3Aspring_framework&cpe_version=cpe%3A%2F%3Apivotal_software%3Aspring_framework%3A5.2.3) [cpe:2.3:a:springsource:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aspringsource&cpe_product=cpe%3A%2F%3Aspringsource%3Aspring_framework&cpe_version=cpe%3A%2F%3Aspringsource%3Aspring_framework%3A5.2.3) [cpe:2.3:a:vmware:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Avmware&cpe_product=cpe%3A%2F%3Avmware%3Aspring_framework&cpe_version=cpe%3A%2F%3Avmware%3Aspring_framework%3A5.2.3) | In Spring Framework versions 5.3.0 - 5.3.13, 5.2.0 - 5.2.18, and older unsupported versions, it is possible for a user to provide malicious input to cause the insertion of additional log entries. This is a follow-up to CVE-2021-22096 that protects against additional types of input and in more places of the Spring Framework codebase. | Upgrade to latest version |

## Mitigation Plan

After analyzing all the results from the static analysis, most of the mitigation can be handled by upgrading the libraries used within the API to their latest versions.