****

# 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** | **05/08/2023** | **Phillip Cabaniss** | **Completed** |

## 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

Phillip Cabaniss

## Interpreting Client Needs

* What is the value of secure communications to the company?

Being a financial corporation, almost all the data being processed will most likely be extremely confidential. Having secure communications with the servers and when using APIs would be a crucial feature of the company. Meaning, that if the information were to leak, the damage to the company's reputation would be extensive. The security of the code should not be overlooked.

* Does the company make any international transactions?

It is not uncommon for financial institutions make global transactions. For instance, banks normally will wire transfer money, or, Artemis Financial may be a global company that sends data across countries to other branches.

* Are there governmental restrictions about secure communications to consider?

Always. How many regulations depends on if the company is global or not. A global company will have to deal with the individual laws of each country it does business in. I will list the most common laws with financial companies because we know for certain that we will have to adhere to these. This information is coming from UpGuard.com.

* + - **Payment Card Industry (PCI) Data Security Standards (DSS)** - PCI DSS for short - is a set of standards for reducing credit card fraud and protecting the personal details of credit cardholders.
    - **The Gramm–Leach–Bliley Act (GLBA)** requires financial institutions to protect customer data and honestly disclose all data-sharing practices with customers.
    - **The Bank Secrecy Act (BSA)**, also known as the Currency and Foreign Transactions Reporting Act, aims to prevent financial institutions from laundering money, either willfully or through force during a cyberattack.

While there are more for other countries, this is the minimum we will have to adhere to.

* What external threats might be present now and in the immediate future?
  + - Malware/Ransomware
    - Phishing
    - Cyber Attacks
    - DDos attacks
    - Cryptojacking
    - AI-based attacks
* What are the modernization requirements that you must consider? For example:
  1. The role of open-source libraries
     + 1. In due diligence, we must take the time to properly vet how secure and trustworthy any library or framework is.
       2. We are using a RESTful API, we have to ensure the communication with the interface is secure and untouched to and from the user.
  2. Evolving web application technologies
     + 1. Making sure these libraries are up to date with the most recent security patches and browser standards, and implementing access control will ensure a more secure application.

## Areas of Security

Since this is a financial corporation, the information used in transactions, and individual user data, the security of the application is a top priority. Using the Vulnerability Assessment Process Flow chart, we will examine the most vulnerable areas.

* Input Validation – A necessity when making an app with user interaction, and the fit defense in securing the application.
* APIs – We are using a RESTful API in the application.
* Code quality – Goes hand-in-hand with using APIs and ensuring secure practices are implemented.
* Code Error – Proper error handling and contingency policies must be in place to ensure the safety of user’s data in the event of an attack or failure.
* Encapsulation – Proper protection of data during the transfer process is a crucial feature in financial applications.

## Manual Review

Basic access control errors:

* + - 1. In DocData.java, you have unfinished functions, and the database server address should not be hardcoded in.
      2. In CRUD.java, better naming conventions should be implemented. “content1” can easily be mistaken for “content2”.
      3. In GreetingsController.java, we see no validation of input, and could be open to a privilege escalation. Also, the keyword GET is used in the api to retrieve data but should instead be POST. This ensures there is no trace of the data left on any browser or web server log.

## Static Testing

**bcprov-jdk15on-1.46.jar**: The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.5 to JDK 1.7. Some vulnerabilities in your code include:

**CVE-2013-1624**

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 noncompliant 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, a related issue to CVE-2013-0169.

**CVE-2015-6644** (OSSINDEX)

Bouncy Castle in Android before 5.1.1 LMY49F and 6.0 before 2016-01-01 allows attackers to obtain sensitive information via a crafted application, aka internal bug 24106146.

**CVE-2016-1000338**

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.

**spring-boot-2.2.4.RELEASE.jar:** Spring Boot has one vulnerability:

**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.

**logback-core-1.2.3.jar**: logback-core module has one vulnerability:

**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.

**log4j-api-2.12.1.jar:** The Apache Log4j API has multiple issues, here are a few:

[**CVE-2020-9488**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-9488)

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

[**CVE-2021-44228**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-44228)

Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0 (along with 2.12.2, 2.12.3, and 2.3.1), this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.

**snakeyaml-1.25.jar:** YAML 1.1 parser and emitter for Java has multiple issues, here are a few:

[**CVE-2017-18640**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2017-18640)

The Alias feature in SnakeYAML before 1.26 allows entity expansion during a load operation, a related issue to CVE-2003-1564.

[**CVE-2021-4235**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-4235)

Due to unbounded alias chasing, a maliciously crafted YAML file can cause the system to consume significant system resources. If parsing user input, this may be used as a denial of service vector.

**jackson-databind-2.10.2.jar:** General data-binding functionality for Jackson: works on core streaming API has multiple issues, here are a few:

[**CVE-2020-25649**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-25649)

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.

**[CVE-2020-36518](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-36518" \t "_blank)**

jackson-databind before 2.13.0 allows a Java StackOverflow exception and denial of service via a large depth of nested objects.

**tomcat-embed-core-9.0.30.jar:** Core Tomcat implementation has multiple issues, here are a few:

[**CVE-2019-17569**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2019-17569)

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.

[**CVE-2020-11996**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-11996)

A specially crafted sequence of HTTP/2 requests sent to Apache Tomcat 10.0.0-M1 to 10.0.0-M5, 9.0.0.M1 to 9.0.35 and 8.5.0 to 8.5.55 could trigger high CPU usage for several seconds. If a sufficient number of such requests were made on concurrent HTTP/2 connections, the server could become unresponsive.

**hibernate-validator-6.0.18.Final.jar:** Hibernate's Bean Validation (JSR-380) reference implementation has one issue:

[**CVE-2020-10693**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=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.

**spring-web-5.2.3.RELEASE.jar:** Spring Web has many issues, here are a few:

**CVE-2016-1000027**

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 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.

**CVE-2020-5421**

In Spring Framework versions 5.2.0 - 5.2.8, 5.1.0 - 5.1.17, 5.0.0 - 5.0.18, 4.3.0 - 4.3.28, and older unsupported versions, the protections against RFD attacks from CVE-2015-5211 may be bypassed depending on the browser used through the use of a jsessionid path parameter.

**spring-beans-5.2.3.RELEASE.jar:** Spring Beans has one issue:

**CVE-2022-22965**

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.

**spring-webmvc-5.2.3.RELEASE.jar:** Spring Web MVS has one issue:

**CVE-2021-22060**

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.

**spring-context-5.2.3.RELEASE.jar:** Spring Context has one issue:

**CVE-2022-22968**

In Spring Framework versions 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.

**spring-expression-5.2.3.RELEASE.jar:** Spring Expression Language (SpEL) has many issues, here are a few:

**CVE-2022-22950**

n 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 condition.

**CVE-2023-20861**

In Spring Framework versions 6.0.0 - 6.0.6, 5.3.0 - 5.3.25, 5.2.0.RELEASE - 5.2.22.RELEASE, 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.

## Mitigation Plan

The bulk of these static test issues stem from out of data dependencies and older vulnerable versions. Go through and update them and the issues should be resolved. It is important to keep your libraries up to date with the most recent patches.

* + - bcprov-jdk15on-1.46.jar: Upgrade to 1.70
    - spring-boot-2.2.4.RELEASE.jar: Upgrade to 3.0.6
    - logback-core-1.2.3.jar: Upgrade to 1.4.7
    - log4j-api-2.12.1.jar: Upgrade to 2.20.0
    - snakeyaml-1.25.jar: Upgrade to 2.0
    - jackson-databind-2.10.2.jar: Upgrade to 2.15.0
    - tomcat-embed-core-9.0.30.jar: Upgrade to 11.0.0-M5
    - hibernate-validator-6.0.18.Final.jar: Migrate to org.hebernate.validator
    - spring-web-5.2.3.RELEASE.jar: Upgrade to 6.0.8
    - spring-beans-5.2.3.RELEASE.jar: Upgrade to 6.0.8
    - spring-webmvc-5.2.3.RELEASE.jar: Upgrade to 6.0.8
    - spring-context-5.2.3.RELEASE.jar: Upgrade to 6.0.8
    - spring-expression-5.2.3.RELEASE.jar: Upgrade to 6.0.8

As for the manual review of the code, the functions need to be completed and exposed information needs to be encapsulated in variables. Proper access control would ensure that all input and output is validated and cannot be breached by any outside sources. We need to talk to the team to start using POST instead of GET and to use proper naming conventions when naming variables.

1. **Citations**

Kost, E. (2023, March 31). *Top 8 Cybersecurity Regulations for Financial Services: Upguard*. RSS. https://www.upguard.com/blog/cybersecurity-regulations-financial-industry#toc-5