****

# 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** | **July 15, 2023** | **Easton Kyhl** | **Initial draft created** |

## 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

Easton Kyhl

## Interpreting Client Needs

It is crucial that financial companies especially are secure from any sort of attack or threat as they typically hold the most sensitive information of all of their users like transaction history, card numbers, tax information, and even potentially social security numbers. Even one slip in the realm of security could result in catastrophe for the business since many are not willing to risk that information and will simply go elsewhere. In the continually evolving age of cyber attacks and cybersecurity, there are more and more threats to need to be prepared for and protect against.

The Gramm-Leach-Bliley Act (GLBA) is one that stuck out immediately to me as it rather directly deals with financial institutions and how they handle sensitive information. They would also have to explain how they use their customer’s data because of this act. There are other regulations that they may have to be careful of as well depending on what information or from who they collect from, most notably acts like COPPA. Not only are there federal laws and acts to be aware of, but many states also have their own regulations in place along with international laws to be aware of if they were to operate in different counties.

Open-source is a rather double-edged sword of risk in terms of it can be incredibly helpful with some of the functions of the libraries. It can be the backbone of some programming functions while on the other hand can produce some additional vulnerabilities to the application. Luckily, the support groups behind large libraries are quite substantial, so it’s less of a problem, but there may be some unexpected loopholes or known exploits that aren’t published from bad acting creators.

## Areas of Security

Areas of security I could think to focus on would be Input Validation, Secure APIs, Cryptography, Code error, and Encapsulation.

Inputs will always need validation, even from trusted sources. Without proper validation, any sort of unexpected input could produce some very unexpected results and even create some vulnerabilities to attacks later on.

Focusing on securing API interactions goes a bit further and confirms the connection to the API is secure in the first place.

Cryptography becomes incredibly important when dealing with any sort of financial or sensitive information. Companies need to make sure information cannot be traced or tracked in any way.

Code errors is rather straight forward and is somewhat built-in to the API, but need to be handled in such a way that it would not give off information a potential attacker could use as an exploit.

The last is encapsulation. Since we are working with some different methods, we want to make sure any data passed through these methods is protected and cannot be changed without the proper means to do so.

## Manual Review

First thing I noticed from the POM file is that there are a couple out of date files like the Maven release or the Spring Boot files which will come into play later looking at the static testing.

Same as module two, Getters and Setters are used in the initial Greeting class, but not in GreetingControl. In fact, there is an input variable (id) that is not validated. Without validation, the program becomes susceptible to injection. An index out of bounds error message would be displayed that, if seen, could open the door for exploits of the program

There were plenty of methods that were also super clearly out of date with plenty of known exploits to their names from the dependency report, again it will come up later.

## Static Testing

A screenshot of a computer

Description automatically generated

hibernate-validator-6.0.18.Final.jar

A close up of a text

Description automatically generated

There was one vulnerability found here with a medium severity that essentially allowed attackers to bypass any sort of input sanitation in relation to error messages. It was found back in mid-2020 though so it seems the best action would be a simple update

jackson-databind-2.10.2.jar

A screenshot of a computer

Description automatically generated

Multiple high-level vulnerabilities, six to be exact, including two found within the last couple months both relating to DOS attacks. An update would probably be fine, but it may be safest to find a different library entirely.

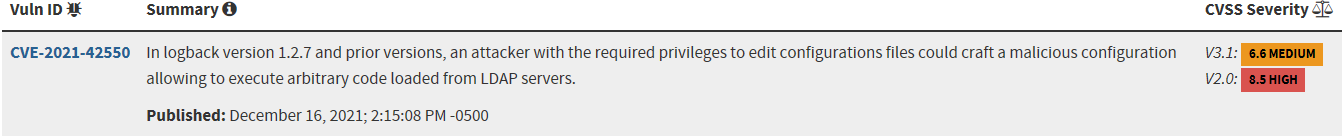
log4j-api-2.12.1.jar

A screenshot of a computer

Description automatically generated

Our first critical vulnerability shows up here. The NVD shows five vulnerabilities with the highest level critical but the most recent was late 2021. Since then, they disabled the feature providing the critical, so it seems an update here would be just fine.

logback-core-1.2.3.jar



Logback only displays one vulnerability related to executing arbitrary code if the attacker had the required privileges. This did earn a high severity, but it is the only exploit found and it was also found back in late 2021. I would also say an update here is fine.

snakeyaml-1.25.jar

A screenshot of a computer

Description automatically generated

A vulnerability was found at a critical level in late 2022 as the most recent, the other seven were of medium severity including four of the same error logged in the database. An update is likely fine, the exploit was due to a constructor class not restricting types for instantiation during deserialization.

spring-boot-2.2.4.RELEASE.jar

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

A screenshot of a computer

Description automatically generated

Two vulnerabilities relating to these, both considered a high severity, including one found back in March of this year. This recognized a potential for DOS attacks in older versions of spring boot. Strong suggestions to update for this very reason. Many vulnerabilities are found in older software and applications.

spring-core-5.2.3.RELEASE.jar

spring-web-5.2.3.RELEASE.jar

spring-webmvc-5.2.3.RELEASE.jar

A screenshot of a computer

Description automatically generated

Each of these had the same vulnerabilities. Most recently, a condition for a DOS attack could be found in earlier releases. Most of the 12 vulnerabilities found in the spring API seem to be medium level, but it does include two critical level threats. One of them was logged in early 2020 relating to untrusted data but could depend on how the library was implemented. The other opened a vulnerability with the potential to be exploited in many different ways which earned it the critical severity ranking.

tomcat-embed-core-9.0.30.jar

tomcat-embed-websocket-9.0.30.jar

A screenshot of a computer

Description automatically generated

Tomcat showed 21 vulnerabilities, topping the list by quite the margin, the most recent logged in March of this year. This vulnerability allowed the session cookie to be transferred over an insecure channel. It also has a known critical flaw logged back in February of 2020 which AJP connections could, if made available to an attacker, exploited in many different ways, also earning it the critical severity ranking.

bcprov-jdk15on-1.46.jar

A screenshot of a computer

Description automatically generated

This was swapped from module two for the mango dependency. It has a total of 12 vulnerabilities with virtually the highest being the most recent with a high severity ranking. The most recent in this case is actually found way back in mid-2018 and allowed the use of ECB mode. This was regarded as unsafe, and support was removed from the provider.

## Mitigation Plan

Updates are desperately needed for every library and that would be the first focus for me as a developer. This code had a staggering 114 known vulnerabilities, and most were from simply having old versions of code. There are also some improvements to be made in terms of encapsulating some of the methods and validating the users inputs to ensure we aren’t allowing direct access to a method of our code.