GLOBALRAIN

Artemis Financial Vulnerability Assessment Report

Table of Contents

Document Revision History 3
Client 3
Instructions 3
Developer 4
1. Interpreting Client Needs 4
2. Areas of Security 4
3. Manual Review 4
4. Static Testing 5
5. Mitigation Plan 7

Document Revision History

Version	Date	Author	Comments
1.0	2022-11-11	Eric Slutz	Wrote the report.

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

Eric Slutz

1. Interpreting Client Needs

The client, Artemis Financial, develops savings, retirement, investment, and insurance financial plans for their customers. They are looking to update their business and get their system up-to-date and modernized. For a financial institution, such as Artemis Financial, secure communications is essential to the success of their business and the financial safety of their customers. While the customer base of Global Rain extends around the world, Artemis Financial does not specify the need to make international transactions. This simplifies the requirements in only needing to ensure compliance with local regulations. This includes and governmental restrictions dictating the requirements for secure communications. We must ensure all local laws and regulations are being followed.

The external threats to consider are vast when talking about a financial system. These include everything from DDOS attacks; malware, viruses, or trojan attacks; phishing and social engineering attacks to gain access to the system; ransomware attacks; supply chain attacks; or state-sponsored or radical group attacks.

When modernizing the system for Artemis Financial, the role of open-source libraries and evolving web application technologies must be considered. Open-source libraries can be a great benefit to the modernization project. However, when choosing such a library you should look for one that is actively being maintained and has a good user base. This is a good indication that any issues discovered with the library will be addressed. Evolving web application technologies should be considered to the modernization project. Newer apps are more likely to be employing current security standards. Additionally, when modernizing, it is a good idea to be forward looking. You don't want to adopt a new technology for support for it to drop shortly after completing the modernization.

2. Areas of Security

Regarding the areas of security for Artemis Financial and their system, the following areas should be relevant areas of security to assess:

- Input Validation: Secure input and representations
- API: Secure API interactions
- Cryptography: Ensure financial data is only seen by those authorized
- Code Error: Secure error handling
- Code Quality: Secure coding practices/patterns

The system will require some type of input to create the financial plans. Since there will be input, it must be validated to ensure it is safe. Additionally, since this is a RESTful web application, API security is another needed part to mitigate any security issues during REST API calls. Furthermore, as this is a financial system it is vital to keep information secure using cryptography. Checking for code error is also important. As this is a new function being added, you want to make sure any errors are caught to minimize unintended actions. Lastly, code quality is imperative. Again, since this is a new function, you want to ensure best practices are being followed while coding and that the appropriate patterns are used.

3. Manual Review

After reviewing the code there appears to be multiple issues with the code. The first issue is with the "/greeting" route in the GreetingController.java file. Regardless of if you attach a name query parameter to the route or not, it always gives an error. This seems to indicate that the controller is not set up correctly to output the expected result. Additionally, the Spring service may be being used incorrectly as well, which could be the reason a null exception gets thrown when attempting to access the "/greeting" route.

Looking through DocData.java you can see values hardcoded for connecting to the database, including the database URL, username, and password. It is a bad idea to hardcode usernames and passwords into you code. If any unauthorized person gets access to your source code, they would be able to get those values to login to your database on their own.

Lastly, in the CRUDController.java file, the "/read" route looks like it would allow anyone to access the data if they could provide a valid name with the request. This threat could allow someone to gain access to information about a business that they should not have.

4. Static Testing

The static testing for the system of Artemis Financials' found vulnerabilities with 11 of the packages. The package, packages description, and relates CVEs are listed below. The recommended solution in most cases is to update the package to a newer issue. This would resolve the security issue for those packages. In some cases, a configuration change is all that is needed to resolve the threat.

Dependency	Description	Vulnerability Code
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.	CVE-2016-1000338 CVE-2016-1000342 CVE-2016-1000343 CVE-2016-1000344 CVE-2016-1000352 CVE-2016-1000341 CVE-2016-1000345 CVE-2017-13098 CVE-2020-15522 CVE-2020-0187 CVE-2016-1000339 CVE-2020-26939 CVE-2015-7940 CVE-2018-5382 CVE-2013-1624 CVE-2016-1000346 CVE-2015-6644

hibernate-validator-6.0.18.Final.jar	Hibernate's Bean Validation (JSR-380) reference implementation.	CVE-2020-10693
jackson-databind-2.10.2.jar	General data-binding functionality for Jackson: works on core streaming API	CVE-2020-25649 CVE-2020-36518 CVE-2022-42003 CVE-2022-42004
log4j-api-2.12.1.jar	The Apache Log4j API	CVE-2020-9488
logback-core-1.2.3.jar	logback-core module	CVE-2021-42550
snakeyaml-1.25.jar	YAML 1.1 parser and emitter for Java	CVE-2017-18640 CVE-2022-25857 CVE-2022-38749 CVE-2022-38751 CVE-2022-38752 CVE-2022-38750
spring-boot-2.2.4.RELEASE.jar	Spring Boot	CVE-2022-27772
spring-core-5.2.3.RELEASE.jar	Spring Core	CVE-2022-22965 CVE-2021-22118 CVE-2020-5421 CVE-2022-22950 CVE-2022-22971 CVE-2022-22968 CVE-2022-22970 CVE-2021-22060 CVE-2021-22096
spring-web-5.2.3.RELEASE.jar	Spring Web	CVE-2016-1000027 CVE-2022-22965 CVE-2021-22118 CVE-2020-5421 CVE-2022-22950 CVE-2022-22971 CVE-2022-22968 CVE-2022-22970 CVE-2021-22060 CVE-2021-22096
tomcat-embed-core-9.0.30.jar	Core Tomcat implementation	CVE-2020-1938 CVE-2020-11996 CVE-2020-13934 CVE-2020-13935 CVE-2020-17527 CVE-2021-25122

tomcat-embed-websocket-9.0.30.jar	Core Tomcat implementation	CVE-2021-41079 CVE-2022-29885 CVE-2020-9484 CVE-2021-25329 CVE-2021-30640 CVE-2022-34305 CVE-2021-24122 CVE-2021-33037 CVE-2019-17569 CVE-2020-1935 CVE-2020-13943 CVE-2021-43980 CVE-2020-1938 CVE-2020-8022 CVE-2020-11996 CVE-2020-13934 CVE-2020-13935 CVE-2020-17527 CVE-2021-25122 CVE-2021-41079 CVE-2022-29885 CVE-2020-9484 CVE-2021-25329 CVE-2021-30640 CVE-2022-34305 CVE-2021-24122 CVE-2021-33037 CVE-2019-17569 CVE-2020-1935 CVE-2020-13943 CVE-2021-43980

5. Mitigation Plan

Many of the security vulnerabilities can be resolve by ensuring good security practices are followed. Ensuring all input is validated, extensive testing, and thorough code reviews will help to mitigate the threats from input, code errors, and code quality. Additionally, having standards in place to make sure project dependencies stay up to date will further increase the secureness of the system.

Action List:

1. Validate all input
2. Follow encryption standards for financial and customer data
3. Follow best practices for RESTful APIs

4. Thoroughly test code
5. Thoroughly review code changes
6. Update bcprov-jdk15on-1.46.jar to at least version 1.66 to resolve package vulnerabilities
7. Update hibernate-validator-6.0.18.Final.jar to at least 6.1.5 to resolve package vulnerabilities
8. Update jackson-databind-2.10.2.jar to at least version 2.10.5.1 to resolve package vulnerabilities
9. Update log4j-api-2.12.1.jar to at least version 2.12.3 to resolve package vulnerabilities
10. Update logback-core-1.2.3.jar to a version above 1.2.7 to resolve package vulnerabilities
11. Update snakeyaml-1.25.jar to at least version 1.32 to resolve package vulnerabilities
12. Update spring-boot-2.2.4.RELEASE.jar to at least version 2.2.11 to resolve package vulnerabilities
13. Update spring-core-5.2.3.RELEASE.jar to at least version 5.2.22 and use a Java framework newer than JDK 9
14. Update spring-web-5.2.3.RELEASE.jar to at least version 5.2.22 and use a Java framework newer than JDK 9
15. Update tomcat-embed-core-9.0.30.jar to at least version 9.0.64 to resolve package vulnerabilities
16. Update tomcat-embed-websocket-9.0.30.jar to at least version 9.0.68 to resolve package vulnerabilities