****

# 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 5](#_Toc32574613)

[4. Static Testing 5](#_Toc32574614)

[5. Mitigation Plan 8](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **07-12-2023** | **Justin Hancock** | **Initial Review** |

## 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

Justin Hancock

## Interpreting Client Needs

*Review the scenario to determine your client’s needs and potential threats and attacks associated with their application and software security requirements. Document your findings in your vulnerability assessment report. Consider the scenario information and the following questions regarding how companies protect against external threats:*

* *What is the value of secure communications to the company?*
* *Does the company make any international transactions?*
* *Are there governmental restrictions about secure communications to consider?*
* *What external threats might be present now and in the immediate future?*
* *What are the modernization requirements that you must consider? For example:*
  + *The role of open-source libraries*
  + *Evolving web application technologies*

Artemis Financial is a financial consulting company that develops plans for their clients that include savings, retirement, investments, and insurance. Financial companies hold some of the most important information of their clients, so when it comes to secure communications, the value needs to be held to the highest standard regardless of if the communication is internal or external. We don’t know for sure based on the information provided, but we can be safe to assume that there could potentially be some international transactions with markets overseas or even clients that could be overseas. The Financial Industry Regulatory Authority, FINRA, is a governmental entity that does have some restrictions on communication with social media, but secure communication between the business and the client does not have specific regulations. Artemis Financial does need to have priority on the security of the correspondence between consultant and client to secure any information from being accessed by an unwanted party or having a client’s personal information leaked. Financial companies hold personal information like bank accounts, social security numbers, addresses, etc. that should be held securely. There is always a potential external threat of someone attempting to access this secure information, but with the proper security in place, we can avoid these attempts. The modernization requirements that must be considered are to continuously stay up to date with their open-source libraries, so any bugs or security threats are covered and controlled. With technology constantly changing, the system needs to be kept up to date on a regular basis to keep all the data stored and communicated secure for the business and the clients.

## Areas of Security

*Use what you’ve learned in step 1 and refer to the Vulnerability Assessment Process Flow Diagram provided. Think about the functionality of the software application to identify which areas of security apply to Artemis Financial’s web application. Document your findings in your vulnerability assessment report and justify why each area is relevant to the software application.*

Upon the investigation of the areas of security for Artemis Financial, I believe the potential vulnerabilities are input validation, API’s, cryptography, code error, and code quality. Input validation is one of the most important areas of security because you want to know that when you enter your username and password you see your information and not someone else's. You also want to know that no one else could have access to your account without that validation. The API’s determine how an end user interacts with the system. There should be methods in place to access acceptable data for a specified user. Many software applications are run with third party software, so we will need to make sure that the API’s are secure. Cryptography will be important to satisfy any international transactions that will include a client’s personal information. The information should be secured by complying with rules and regulations set by the U.S. and whatever country the information is being sent. Code errors and code quality go together in my opinion. They should work with the input validation and API to make sure that proper handling of any error not only prevents unwanted access, but also insures no unwanted data breaches. These two subjects will keep methods that we have created inaccessible by unauthorized users.

## Manual Review

*Refer to the seven security areas outlined in the Vulnerability Assessment Process Flow Diagram. Use what you’ve learned in steps 1 and 2 to guide your manual review. Identify all vulnerabilities in the Project One Code Base, linked in Supporting Materials, by manually inspecting the code. Document your findings in your vulnerability assessment report. Be sure to include a description that identifies where the vulnerabilities are found (specific class file, if applicable).*

From what we have learned in this course, the first thing that I look for when identifying vulnerabilities is looking into the POM.XML file for any Apache validator. I was able to find version 4.0.0 Maven Apache in the POM.XML file. I then looked for any input validation within the greeting and greeting controller files; and no where in those two files is there a line of code that 1. Asks for input 2. Validates the input or 3. Provides an output. I have been unable to find an API in the program. Upon running the program as it is, there seems to be some information being gathered from a URL, but there is no way for the user to understand how to interact with the program. There should be a specific way that a user interacts with the Restful API. There is no type of encryption key built into the program that I could find which means that there is no cryptography within the program. The DocData class uses try and catch methods, but those methods do not include an error catching procedures within the code. The code quality needs improvement as well. Inline comments need to be added so others working within the program can pick up on what the creator is trying to do in certain areas of the development process. The code is also not fully functioning, so there needs to be some additional work to complete the program. The vulnerabilities that I have listed throughout my manual review do leave Artemis Financial at risk of breaches and unwanted data being retrieved.

## Static Testing

*Integrate the dependency-check plug-in into Maven by following the instructions outlined in the Integrating the Maven Dependency-Check Plug-in tutorial provided in Supporting Materials. Run a dependency check on Artemis Financial’s software application to identify all security vulnerabilities in the code. Specifically, identify all vulnerabilities in the code base by analyzing results from running the code through a static test. Include these items from the dependency-check report in your vulnerability assessment report:*

* *The names or vulnerability codes of the known vulnerabilities.*
* *A brief description and recommended solutions that are found in the dependency-check report.*
* *Attribution (if any) that documents how this vulnerability has been identified or how it was documented in the past.*

1. bcprov-jdk15on-1.46.jar
   1. 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.
   2. 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.
2. hibernate-validator-6.0.18.Final.jar
   1. Hibernate's Bean Validation (JSR-380) reference implementation.
   2. 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.
3. jackson-databind-2.10.2.jar
   1. General data-binding functionality for Jackson: works on core streaming API
   2. 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.
4. log4j-api-2.12.1.jar
   1. The Apache Log4j API
   2. 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
5. logback-core-1.2.3.jar
   1. logback-core module
   2. 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.
6. snakeyaml-1.25.jar
   1. YAML 1.1 parser and emitter for Java
   2. 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.
7. spring-boot-2.2.4.RELEASE.jar
   1. Spring Boot
   2. \*\* 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.
8. spring-boot-starter-web-2.2.4.RELEASE.jar
   1. Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container.
   2. \*\* 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.
9. spring-core-5.2.3.RELEASE.jar
   1. Spring Core
   2. 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.
10. spring-web-5.2.3.RELEASE.jar
    1. Spring Web
    2. 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.
11. spring-webmvc-5.2.3.RELEASE.jar
    1. Spring Web MVC
    2. 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.
12. tomcat-embed-core-9.0.30.jar
    1. Core Tomcat implementation
    2. When using the Apache JServ Protocol (AJP), care must be taken when trusting incoming connections to Apache Tomcat. Tomcat treats 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 in ways that may be surprising. In Apache Tomcat 9.0.0.M1 to 9.0.0.30, 8.5.0 to 8.5.50 and 7.0.0 to 7.0.99, Tomcat shipped with an AJP Connector enabled by default that listened on all configured IP addresses. It was expected (and recommended in the security guide) that this Connector would be disabled if not required. This vulnerability report identified a mechanism that allowed: - returning arbitrary files from anywhere in the web application - processing any file in the web application as a JSP Further, if the web application allowed file upload and stored those files within the web application (or the attacker was able to control the content of the web application by some other means) then this, along with the ability to process a file as a JSP, made remote code execution possible. It is important to note that mitigation is only required if an AJP port is accessible to untrusted users. Users wishing to take a defence-in-depth approach and block the vector that permits returning arbitrary files and execution as JSP may upgrade to Apache Tomcat 9.0.31, 8.5.51 or 7.0.100 or later. A number of changes were made to the default AJP Connector configuration in 9.0.31 to harden the default configuration. It is likely that users upgrading to 9.0.31, 8.5.51 or 7.0.100 or later will need to make small changes to their configurations.
13. tomcat-embed-websocket-9.0.30.jar
    1. Core Tomcat implementation
    2. When using the Apache JServ Protocol (AJP), care must be taken when trusting incoming connections to Apache Tomcat. Tomcat treats 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 in ways that may be surprising. In Apache Tomcat 9.0.0.M1 to 9.0.0.30, 8.5.0 to 8.5.50 and 7.0.0 to 7.0.99, Tomcat shipped with an AJP Connector enabled by default that listened on all configured IP addresses. It was expected (and recommended in the security guide) that this Connector would be disabled if not required. This vulnerability report identified a mechanism that allowed: - returning arbitrary files from anywhere in the web application - processing any file in the web application as a JSP Further, if the web application allowed file upload and stored those files within the web application (or the attacker was able to control the content of the web application by some other means) then this, along with the ability to process a file as a JSP, made remote code execution possible. It is important to note that mitigation is only required if an AJP port is accessible to untrusted users. Users wishing to take a defence-in-depth approach and block the vector that permits returning arbitrary files and execution as JSP may upgrade to Apache Tomcat 9.0.31, 8.5.51 or 7.0.100 or later. A number of changes were made to the default AJP Connector configuration in 9.0.31 to harden the default configuration. It is likely that users upgrading to 9.0.31, 8.5.51 or 7.0.100 or later will need to make small changes to their configurations.

## Mitigation Plan

*Interpret the results from the manual review and static testing report. Identify steps to mitigate the identified security vulnerabilities by creating an action list that documents how to fix each vulnerability in your vulnerability assessment report.*

Based on the manual review and static testing for Artemis Financial, the mitigation plan of attack is going to begin with creating input validation within the program to verify and validate user information upon login attempts. There will also need to be API’s created to run the program with specific interactions for the user when using the system. Code quality can be fixed by adding inline comments to clean up what the creator is attempting to complete with the sections of code. Code errors need to be put into place to help validate certain areas of data access to verify the data is being pulled from the correct location with the correct user and so on. When it comes to the information from the static test, most of the vulnerabilities can be resolved by updating to the most current version of that software. Updating to the most current version of the software just allows the program to run with the latest bug fixes and resolved issues that could have been a vulnerability.