****

# CS 305 Project One

**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 5](#_Toc32574612)

[3. Manual Review 5](#_Toc32574613)

[4. Static Testing 6](#_Toc32574614)

[5. Mitigation Plan 8](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **05/22/2022** | **Gabriel Romero** | **Initial security report** |

## Client



## Instructions

Deliver this completed vulnerability assessment report, identifying your findings of security vulnerabilities and articulating recommendations for next steps to remedy the issues you have found.

Respond to the five steps outlined below and include your findings. Replace the bracketed text on all pages with your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Gabriel Romero

## 1. Interpreting Client Needs

Determine your client’s needs and potential threats and attacks associated with their application and software security requirements. Consider the following regarding how companies protect against external threats based on the scenario information:

* What is the value of secure communications to the company?
* Are there any international transactions that the company produces?
* 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 must be considered, such as the role of open-source libraries and evolving web application technologies?
* As a Financial company secure communication would be to the utmost importance. The data we will be using would contain very detailed personal information that must be treated as securely as possible. In every line of code that is written, security needs to be at the forefront of the thought behind it.
* Through investments and insurance policies it goes without saying that there would be some level of international transactions. This means there must be some thought put into adherence to international banking laws at there are some very strict stipulations when it comes to applications dealing with finances. This goes doubly in the EU.
* When dealing with international transactions there would be several restrictions to be concerned about and our team would need to be privy of all of them. This would only benefit us moving forward if there is a desire to maintain support for the application that is developed and ensure that we are following along best practices and standards.
* The main threats would be the integrity of the data we are transmitting. We are working with tons of personal identifiable information (PII) and this data needs to be kept extremely secure and never transmitted as is. We are talking heavy levels of encryption, cryptography, use of appropriate roles and groups. Everything in our tool kit to keep this data safe as this is the data that is most sought after by attackers. Not only that but our application could come under attack via distributed denial of service, injection attacks to get information out and backdooring via error codes if not correctly handed.
* Modernization means we must build an application that can change with the changing technology. We cannot just rely on static versions of libraries or fad tech frameworks. We must work with what has been tested and is true to what we need. Using “new” tech could lead to holes in our security that we would have no known due to lack of exposure. Using “old” libraires could leave us vulnerable to older bugs that were never patched. We need to find the middle ground when it comes to our libraries and be ready to make a swap out if needed. The application must be fluid enough to allow for us to focus on security and ensure that the data we are putting out there is completely secure.

## 2. Areas of Security

Referring to the Vulnerability Assessment Process Flow Diagram, identify which areas of security are applicable to Artemis Financials’ software application. Justify your reasoning for why each area is relevant to the software application.

* **Input Validation**: We will be having users log in and enter data as needed. This could be searching their transaction history or looking for help on the website. We need to be ready to insure what we are getting back is what we are looking for. Sanitization of the input is important and necessary.
* **APIs**: We are using a RESTful application so there will be calls to an API which we have to ensure goes through securely. Which leads to the next area of assessment.
* **Cryptography**: There will be loads of PII data and financial information floating around which means that every single bit of data that comes from our app and is in transit with API calls, MUST be encrypted to the utmost degree. There needs to be several levels of the most modern and best practice encryption. Which also means that there will be extensive pen testing when the time comes.
* **Client/Server**: This goes hand in hand with APIs and Cryptography as the data we are feeding the user needs to be up to par with our security and ensure we are using the proper certifications.
* **Code Error**: Having input validations, sanitization and API calls all require us to ensure that the error messages we are sending back are not leaking data that could be used against our application. This is one of the easier ways to let information slip out and it is best to stay vigilant when working with error cases.
* **Code Quality**: Everything we write for this application needs to have the idea of security first. Specifically with the data that we are working with. Best practices must be used, and test cases must be developed to ensure we are not leaking data at every turn. It will be a tedious bridge to cross, but the result will be a higher quality secure application.
* **Encapsulation**: API calls and our data need to be created and maintained to a high degree. We need to ensure that the data we handle can be encrypted successfully without issue and passed through our application as needed.

## 3. Manual Review

Continue working through the Vulnerability Assessment Process Flow Diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

* The first few things that I noticed were that a lot of the classes contained public methods to serve the data right up. This could be dangerous as an attacker, if the found there way into our systems, could simply call our methods without accessors, and retrieve data. The following classes contain public classes that need to be reviewed:
  + CRUD.java
  + Customer.java
  + DocData.java
  + Greeting.java
  + myDateTime.java
  + RestServiceApplication.java

Having these methods public and not belonging to groups can cause issues down the line with our application and must be reviewed. There should be proper accessors and setters in place.

* There is no input validation for our controllers. In crud controller, DocData, and GreetingController.java the input is just taking the string as is with no sanitization or limit to what can be sent. This leaves us open to huge data breaches in the form of sql injection attacks. Especially when it comes to the DocData.java connection it makes to our database. An attacker could drop all our tables and our information would be gone.
* Some of the methods are still incomplete and need to be finished., For example the DocData.java a catch block has not been set which can be dangerous as any error pushed to the user could give away damaging information about our app. It currently is printing the entire stack trace so for testing its fine, but it must be fixed going into production.

## 4. Static Testing

Run a dependency check on Artemis Financials’ software application to identify all security vulnerabilities in the code. Record the output from dependency check report. Include the following:

1. The names or vulnerability codes of the known vulnerabilities
2. A brief description and recommended solutions provided by the dependency check report
3. Attribution (if any) that documents how this vulnerability has been identified or documented previously

The following is a list of dependency vulnerabilities caught by our static testing, ordered from critical to low in descending order (A screenshot will be provided right after the posting of the dependencies):

* tomcat-embed-websocket-9.0.30.jar
  + CVE-2022-29885: Low severity, does not protect while clustering and running over untrusted networks. Recommended to us VPN when clustering over untrusted networks. There are also 16 other vulnerabilities associated with this CPE dependency which can be found here: [LINK](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)
* tomcat-embed-core-9.0.30.jar
  + In the list of 17 vulnerabilities which can be found in the link provided above.
* spring-core-5.2.3.RELEASE.jar
  + CVE-2022-22950: medium severity, there is an issue with spring frameworks version 5.3.0 – 5.3.16 where a user can provide a specially crafted SpEL expression that can cause an DOS condition. Recommended to upgraded to at least 5.3.17+ for 5.3 users while 5.2.x users can go to 5.2.20+.
* spring-aop-5.2.3.RELEASE.jar
  + CVE-2022-22968: Medium severity, disallowedFields has an issue where it was case sensitive meaning that if it was not fully sanitized it could be used to gain access. Recommended to upgrade past 5.3.19+ or 5.2.21+. There is also a list of 6 other vulnerabilities with this dependency which can be found here: [LINK](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)
* bcprov-jdk15on-1.46.jar
  + CVE-2016-1000352: High severity, enabled the use of ECB mode which is regarded as high unsafe to use and there is currently no support for it. Advised to update to a more current version which removes this issue. A list of 11 other CVE can be found here with this dependency: [LINK](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)
* spring-boot-2.2.4.RELEASE.jar
  + CVE-2022-27772: high severity, version v2.2.11.RELEASE was vulnerable to temporary directory hijacking. This vulnerability impacted the org.springframework.boot.web.server.AbstractConfigurableWebServerFactory.createTempDir method. This issue was patched in later versions.
* snakeyaml-1.25.jar
  + CVE-2017-18640: severity high, The Alias feature in SnakeYAML 1.18 allows entity expansion during a load operation, a related issue to CVE-2003-1564. Upgrade to at least 1.26 or later to fix this.
* jackson-databind-2.10.2.jar
  + CVE-2020-36518: before version 2.13.0 allowed for a java stackoverflow exception that led to DOS attacks. Upgrading the version corrected this problem.
* logback-core-1.2.3.jar
  + CVE-2021-42550: medium severity, anyone with access could potential edit config files that would execute code from a LDAP servers that could cause issues with the application. Fixed in version 1.3.0-alpha11 and onward.
* hibernate-validator-6.0.18.Final.jar
  + CVE-2020-10693: medium severity, allowed for invalid EL expressions to be passed as valid. Fixed in version 7.0.0 Alhpa2, 6.1.5 Final, 6.0.20 Final.
* log4j-api-2.12.1.jar
  + CVE-2021-44832: medium severity, This made apache vulnerable to remote code execution when an attackers has use of an LDAP server they can manipulate the code from. Fixed in subsequent versions. There are also a list of other issue pertaining to this dependency which can be found here: [LINK](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)

Graphical user interface, text, application, email

Description automatically generated

## 

## 5. Mitigation Plan

After interpreting your results from the manual review and static testing, identify the steps to remedy the identified security vulnerabilities for Artemis Financial’s software application.

* Firstly, proper data structures and class best practices need to be implemented. Getters and Setters need to properly be set along with ownership to enable data returns.
* Input validation is a must especially when dealing with sql queries. We need to sanitize and validate the input that we are passing into our DB call.
* We must make sure that our errors are not just generic stack traces that we are sending back to the user as this could leak information to attackers.
* Even though this is just the start of the application, it still needs to maintain a high level of security with every piece of code written.
* As for most of the dependency checks require an update. I would recommend focusing on the next best version of the dependencies so that we are working with tested versions with the fix applied. Can deep dive on most and test each case depending on then next version we use.
* Will have to continue to do testing and ensure each version we use is up to par with our security needs.