****

# 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 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** | **2022-01-23** | **Andrew Black** | **Initial revision** |

## 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

Andrew Black

## 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 dealing with customers’ financial data and personal information on a day-to-day basis, security should be of the utmost importance to every employee, developer, and software make-up. What Artemis Financial needs is a secure system from A to Z; every step of the process should be heavily scrutinized and inspected for vulnerabilities, especially pertaining to user input. There’s been no information to hint towards Artemis Financial dealing with international transactions, nor any explicit government restrictions that need to be considered. One external threat that will always exist is malicious actors looking to socially engineer employees or customers into giving information that grant access to the system, so that it can be compromised from the inside-out. A solution to this is granting only what information is required for the employee to complete their job and training them to look out for potential social engineering attacks. Finally, modernization requires engineers to stay on top of up-and-coming technologies, especially as they pertain to security of the overall system. By using open-source libraries, the code used to drive the system is completely open to everyone for auditing, meaning someone either in or out of the development team can report a vulnerability and strengthen security in the library, benefiting everyone using it—including Artemis Financial.

## 2. Areas of Security

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

After describing the applicable areas, it seems as though every single area of security is applicable to Artemis Financial’s software application. Starting at the first area: input validation. When customers interact with the system, it needs to be secure from the get-go—users should not be able to input data that breaks the system, either for them or anyone else. The next three applicable areas are secure API interactions, cryptography, and client/server—secure distributed composing. One common stereotype of these being broken is someone banking at a coffee shop. If the end user is not using HTTPS (with TLS encryption) to interact with the API, any malicious actor with the right program for sniffing data packets can intercept the data being transferred and potentially break into their account. The composition of this security falls under client/server, where the client needs to make an encrypted call to a secure API on the server. The next applicable area is code error, where any errors that occur on the back end need to be handled properly, such as hardware hiccups, performance issues, and timing discrepancies. If the error is the fault of the user, they should be told what went wrong and how they can rectify the situation themselves. The next applicable area is arguably the most important: code quality. By developing a financial system that directly deals with personal information, it is critical *not only to the business* but to customers that the system is developed with secure coding practices and patterns in mind. Finally, encapsulation is important so that if any aspect of the system is compromised, it does not lead to the *entire* system being compromised. For example, a single user account being accessed illegitimately should not allow attackers to access more user accounts.

## 3. Manual Review

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

* In DocData.java, the database username and password are directly embedded in the code. In a production environment, this should not be the case; it should be stored in environment variables or another secure method.
* In CRUDController.java, parameters can be read by anyone on the client network.
* There is no indication of the application making use of HTTPS or any secure methods to transmit data.

## 4. Static Testing

Run a dependency check on Artemis Financial’s 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

* Ran with dependency-check version 6.5.3
* Scanned 38 dependencies (20 unique)
* Found 10 vulnerable dependencies
* Found 63 vulnerabilities
  + bcprov-jdk15on-1.46.jar: [12 CVEs listed](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) with the highest being 9.8 **CRITICAL**.
  + hibernate-validator-6.0.18.Final.jar: [CVE-2020-10693](https://nvd.nist.gov/vuln/detail/CVE-2020-10693) at 5.3 **MEDIUM**; “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 sanitization (escaping, stripping) controls that developers may have put in place when handling user-controlled data in error messages.”
  + jackson-databind-2.10.2.jar: [CVE-2020-25649](https://nvd.nist.gov/vuln/detail/CVE-2020-25649) at 7.5 **HIGH**; “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.”
  + log4j-api-2.12.1.jar: [5 CVEs listed](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) with the highest being 10.0 **CRITICAL**.
  + logback-core-1.2.3.jar: [CVE-2021-42550](https://nvd.nist.gov/vuln/detail/CVE-2021-42550) at 6.6 **MEDIUM**; “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.”
  + snakeyaml-1.25.jar: [CVE-2017-18640](https://nvd.nist.gov/vuln/detail/CVE-2017-18640) at 7.5 **HIGH**; “The Alias feature in SnakeYAML 1.18 allows entity expansion during a load operation, a related issue to [CVE-2003-1564](https://nvd.nist.gov/vuln/detail/CVE-2003-1564).”
  + spring-aop-5.2.3.RELEASE.jar + spring-core-5.2.3.RELEASE.jar: [CVE-2020-5421](https://nvd.nist.gov/vuln/detail/CVE-2020-5421) at 6.5 **MEDIUM**; “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](https://nvd.nist.gov/vuln/detail/CVE-2015-5211) may be bypassed depending on the browser used through the use of a jsessionid path parameter.”
  + tomcat-embed-core-9.0.30.jar + tomcat-embed-websocket-9.0.30.jar: [17 CVEs listed](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) with the highest being 9.8 **CRITICAL**.

## 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.

Based on manual review:

* The database username and password should be moved to environment variables rather than embedded directly in the code, so that the credentials are only specific to the system that requires it and not the codebase itself.
* Instead of accepting a parameter for the business name in CRUDController.java, accept a POST request with the data embedded so that it cannot be sniffed on the network.
* Ensure that HTTPS is used between the client and server in a production environment.

Based on static testing:

* Upgrade spring-boot-starter-parent to version 2.6.3 and bcprov-jdk15on to version 1.69 to fix all known vulnerabilities found through static testing.