****

# 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** | **3/23/2024** | **Nick Apodaca** |  |

## Client



## Developer

Nick Apodaca

## Interpreting Client Needs

Artemis Financial, our new client, is a consulting firm that helps create individualized financial plans for their various clients. They do this with a focus on various account types ranging from investing to insurance. With banking fraud at all new levels, it is vitally important that they can keep their client’s information secure and protected. Not only to protect themselves from potential losses due to leaks or lawsuits, but Artemis also has a reputation to uphold when it comes to providing reliable and safe advice to their clients. That is why they have come to us, to ensure all their data and communications remain secure.

While Artemis has not yet specified if they will be handling international transactions, in today’s financial world it is important to prepare for such possibilities. This would mean ensuring that any program follows privacy laws like Europe’s General Data Protection Regulation (GDPR).

For here in the United Sates, there is not one encompassing law or regulation that must be followed. Each state has different rules and regulations regarding communication. There are also a variety of federal laws that must be followed when dealing with financial institutions. These include the Fair Credit Reporting Act, the Gramm-Leach-Bailey Act, the Payment Card Industry Data Security Standard, in addition to any of those laws that specifically apply to particular states.

Two important threats that need to be watched out for are injection attacks and authentication vulnerabilities. SQL injection is an especially important threat. Not protecting input fields and parameters could allow an attacker to gain access to the system. Additionally, weaknesses in user authentication or password encryption could enable an attacker to brute force their way into the system, gaining access to all of Artemis’ client’s banking information.

Artemis has also asked us to help modernize their operations. Moving towards using open-source libraries will help ensure that there are regular updates to the security of said libraries. In addition to that, with others using the same libraries, it will become easier to identify future security risks and be able to add extra protection or remove those vulnerabilities until an update becomes available. Considering a move to cloud-based storage and servers is another new technology that has become available most likely since their last update in technology. Not only do these provide added features like limited downtime of the system or easier ways to increase load on or size of servers and databases; it also comes with added levels of security since they are often managed by third parties that also have a reputation to maintain for their clients.

## Areas of Security

There are a few areas of vulnerability that should be addressed when looking at Artemis’ system.

* + **Input Validation**- Attackers can use privilege escalation to try to access data, signing up for an account and then attempting to gain administrative privileges via manipulative input. They may also try to cause a fail-open with their input which if not handled properly can give them access. Customers will no doubt need an account to view their financial plan. They will also most likely need areas to enter in sensitive information. Both of these provide areas of untrusted data entering the system and the system therefore need adequate protection against attackers.
  + **API’s** – Artemis has mentioned that they use RESTful API. API security breaches are a common way of accessing sensitive information. SQL injections, cross-site scripting and other forms of attacks will be used to attempt to gain access to the system. Ensuring that untrusted data does not change the behavior of the system as it is designed is very important.
  + **Cryptography**- Just as Artemis will need to protect itself when untrusted data attempts to enter the system, Artemis will also be sending out sensitive data between its employees and their customers. Whether it is sharing financial plans or sharing sensitive banking information, there is a need to protect that information as it is sent out. This is where cryptography comes into play. Ensuring outgoing data is properly encrypted to help ensure that attackers are unable to glean any sensitive customer information in transit.
  + **Code Quality**- A large part of Artemis requesting our assistance was their desire to upgrade their system. Secure coding practices change rapidly to accommodate the changes in habits of attackers. Ensuring that this new system is up to current industry standards and allows room for improvement and future upgrades to the software being used is very important. If proper coding practices are not added or followed, then these upgrades to the system will be mute in the near future opening up the possibility of a data breach.

## Manual Review

**Unfinished code** – In DocData.java, line 29, a catch block was never completed. If code was run and an exception thrown from the try statement. This failure could possibly allow sensitive information to become accessible.

**catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

**No encryption of sensitive data**- In the entire system, there is no encryption being used, of note is no use of HTTPS. This is especially important because an attacker could easily access customer account information, one possible way is being able to view it through the URL. The public class, customer.java, has a function that simply reveals a customer’s account information. This unprotected data could be easily stolen or altered.

**public** **int** showInfo() {

//code to show customer information

**return** **this**.account\_number;

}

**Poor Request Parameters**- In CRUDcontroller.java, a business’ name is being used as a request parameter in line 13. In a GET request, this can be unnecessarily exposed. It can also help an attacker better take advantage of sensitive data that they may acquire.

**public** CRUD CRUD(@RequestParam(value="business\_name") String name)

**Lack of input validation**- When requesting untrusted data from outside the system, there is no validation, authentication or authorization. This can open up the system to attacks by injection and possibly expose sensitive information.

## Static Testing

|  |  |  |  |
| --- | --- | --- | --- |
| **Dependency** | **Issue** | **Severity** | **Resolution** |
| bcprov-jdk15on-1.46.jar | Older implementation of Bouncy Castle allows for remote attackers to conduct distinguishing attacks and plain-text recovery attacks | HIGH | Update to newest version of library |
| spring-boot-2.2.4.RELEASE.jar | Versions prior to v2.2.11 are vulnerable to temporary directory hijacking | CRITICAL | Upgrade to latest version available |
| logback-core-1.2.3.jar | Attackers with the required privileges to edit configurations could craft malicious configuration allowing to execute arbitrary code loaded from LDAP servers. | HIGH | Ensure latest version is being used, upgrade if necessary. |
| log4j-api-2.12.1.jar | Improper validation can allow interception by man-in-the-middle-attacks. | CRITICAL | Upgrade to latest version which will support this feature. |
| snakeyaml-1.25.jar | Allows for entity expansion during load operation. | CRITICAL | Upgrade to version without CVE-2017-18640 |
| jackson-databind-2.10.2.jar | Used for core streaming API. Flaw did not have entity expansion secured properly. Allows vulnerability to XML external entity attacks | HIGH | Upgrade to latest version which includes patch. |
| tomcat-embed-core-9.0.30.jar | Version allows for possibility of HTTP Request Smuggling. | CRITICAL | Upgrade to latest version of Tomcat for patch. |
| hibernate-validator-6.0.18.Final.jar | Bean validation implementation. Bug enables invalid EL expressions to be evaluated as if valid. | MEDIUM | Upgrade to latest version. |
| spring-web-5.2.3.RELEASE.jar | Suffers from potential remote code execution issue if used for Java deserialization of untrusted data. | HIGH | Using authentication can avoid this danger in program. Upgrading to newest version also addresses issue. |
| spring-beans-5.2.3.RELEASE.jar | Spring running on JDK may be vulnerable to remote code execution via data binding. | HIGH | If used as a Spring boot jar, the default is not vulnerable. Consider upgrading to newest version. |
| spring-webmvc-5.2.3.RELEASE.jar | On older versions of spring framework. It is possible for a user to provide malicious input to cause the insertions of additional log entries. | MEDIUM | Upgrade to newest version of Spring framework. |
| spring-context-5.2.3.RELEASE.jar | In older versions, the patterns for disallowedFields on a DataBinder are case sensitive which can cause it to not be effectively protects. | MEDIUM | Upgrade to latest version to remedy issue. |
| spring-expression-5.2.3.RELEASE.jar | In older versions, a user can provide a SpEL expression that may cause denial of service. | MEDIUM | Upgrade to latest version of spring framework |

## Mitigation Plan

* Upgrading the various libraries and frameworks to their latest versions should be a primary focus. Setting up alerts for new versions or scheduling a regular update period into the maintenance schedule of the program would help avoid possible future security issues.
* Ensure that all code is properly completed.
  + Make sure that any time the code has a try statement there is a corresponding catch. Ideally this would use whitelisting validation rather than blacklisting certain characters or lines of code.
  + Add input validation and employing HTTPS protocol. When asking for input, have clear expectations of what the user is able to enter.
  + Consider moving sensitive parameters to the body of a request rather than the URL to prevent an attacker stumbling into the sensitive information.
  + Since the data is banking related as well, use a cache control header so that the browser will not cache sensitive information.
* Introduce query parameterization into the code. Since the program will be asking for untrusted data, this protection will help build a foundational defense against injection attacks.
* In addition to implementing proper HTTPS protocol, consider additional encryption on data being sent to clients to help prevent it from being intercepted by attackers. TLS or SSL protocols are two such examples.

**Sources:**

Klosowski, T. (2021, September 6). *The state of consumer data privacy laws in the US (and why it matters)*. Wirecutter; New York Times. <https://www.nytimes.com/wirecutter/blog/state-of-privacy-laws-in-us/>

Nair, S. V. (n.d.). *What are the common REST API security vulnerabilities?* What Are the Common REST API Security Vulnerabilities? https://beaglesecurity.com/blog/article/rest-api-security-vulnerability.html