****

# Artemis Financial Vulnerability Assessment Report

Ryan Rieth

Prof. Conlan — CS 305 — SNHU

Table of Contents

[Artemis Financial Vulnerability Assessment Report 1](#_Toc119234304)

[Document Revision History 3](#_Toc119234305)

[Client 3](#_Toc119234306)

[Developer 3](#_Toc119234307)

[Interpreting Client Needs 4](#_Toc119234308)

[Areas of Security 5](#_Toc119234309)

[Input Validation 5](#_Toc119234310)

[APIs 5](#_Toc119234311)

[Cryptography 5](#_Toc119234312)

[Code Error 5](#_Toc119234313)

[Code Quality 5](#_Toc119234314)

[Manual Review 6](#_Toc119234315)

[Static Testing 7](#_Toc119234316)

[Mitigation Plan 9](#_Toc119234317)

[References 10](#_Toc119234318)

## Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 0.0 | 11/09/2022 | Ryan Rieth | Document inception |
| 0.1 | 11/09/2022 | Ryan Rieth | *Interpreting Client Needs* |
| 0.2 | 11/10/2022 | Ryan Rieth | *Areas of Security* |
| 0.3 | 11/11/2022 | Ryan Rieth | *Manual Review* |
| 0.4 | 11/12/2022 | Ryan Rieth | *Static Testing* |
| 0.5 | 11/13/2022 | Ryan Rieth | *Mitigation Plan* |
| 1.0 | 11/13/2022 | Ryan Rieth | Document submission |

## 

## Client



## 

## Developer

Ryan Rieth

## Interpreting Client Needs

The client *Artemis Financial* — from here deemed *Client* — develops individualized financial plans for their customers that include savings, retirement, investments, and insurance. With these plans, *Client* handles highly sensitive customer information that includes, but is not limited to:

Social security numbers

Routing and account bank numbers

Credit cards

Money balances

401(k)

Addresses

Health information

Family members

Stock information

Passwords/Security questions

The sensitivity level of the information requires extensive cybersecurity measures to be implemented; we here at *Global Rain* are tasked with this mission. To script this reality, *Client* is willing to modernize their operations including software and hardware. Furthermore, they wish to employ the most up-to-date security measures available.

*Client* faces a great many types of threats and risks. These threats and risks include (Ozarslan, S., 2022):

Ransomware

Phishing

Web application attacks

Vulnerability exploitation attacks

Denial of Service (DoS) attacks

Insider threats

*Client* possesses vital needs from this project, both to operate generally and protect against these kinds of threats. These needs are:

Server-side focus of operations

Cloud-based services

Secure communications

CDNs, ACLs, or Load Balancers

WAFs

Secure communications are significantly valuable to *Client* considering sensitive information is communicated across multiple agencies. Several mitigation attempts exist that will be discussed later. *Client* also can make international transactions dependent on where their consumers are located or where companies’ stocks are located. This will be a tight area for development as there are several government restrictions centering around data privacy — the United States does not have a singular law that protects the privacy of multiple kinds of data (Klosowski, T., 2021).

Advanced Persistent Threat (APT) groups will arguably forever exist, and they will better their threats and attacks equally with technology’s own evolution. It is impossible to predict their level of security threat, but *Client* can employ security control validation to assess and verify their status of cybersecurity (Ozarslan, S., 2022). Open-source libraries, too, can allow an ever-evolving perspective on what successfully used code that has been modernized to the latest “standard” of software coding, and this perspective can help deduce what threats could come next. Even with this help, ever-changing technology and web application technologies mean that reliance on local-side data management should be minimal and the server-side should be focused on.

## Areas of Security

The Vulnerability Assessment Process Flow details seven areas of security involving development of the architecture for a project. These are Input Validation, APIs, Cryptography, Client/Server, Code Error, Code Quality, and Encapsulation. While all seven are important in their own rights, not all will necessarily require focus for *Client’s* concerns.

The five of the seven areas of security that will require relevant attention and implementation for *Client’s* scenario are as follows:

### Input Validation

A cornerstone security measure for systems. It provides anywhere from a base level to a complex measure of validating user ownership of data, as well as user existence as an authenticated person.

### APIs

While most data should be stored and function on the server-side for *Client*, there will still be some client-side implementation. APIs server as the functionality for server-client communication and utilizing proper, even customized APIs will solidify the privacy protection for *Client*.

### Cryptography

Encrypting data will add yet another layer of privacy and security to client and consumer data. Should data be accessible when it shouldn’t be, encryption can help ensure its utilization is minimal by “disguising” the data.

### Code Error

Understanding errors and how to handle them is vital to a software’s execution. The act of error handling will ensure proper scripting along the way. It can provide run-arounds or dead ends for attackers, providing another security measure before an issue could potentially occur.

### Code Quality

The quality of the existing code determines every other area of security that relies on or affects the code. Substantial code quality can ensure operation between objects as well as proper separation of them; in other words, only authorized users can view sensitive information, and authenticated but unauthorized users would be unable to view the same information.

## Manual Review

The code base presented for *Client* showcases some solid work, but there is improvement that must be implemented in a few capacities.

POST is not utilized in *Client’s* present API. POST sends data to a server to create or update a resource, while GET merely requests data. POST requests cannot be cached, stored in browser history, or bookmarked, while GET requests can.

Input Validation is severely lacking in the *GreetingController* file. A parameter request is made but parameter validation is not handled before the parameter is dealt with.

There appears to be no authentication present in the *Customer* file. Authorization and authentication need to go hand in hand to ensure proper Code Quality. Furthermore, there seems to be lacking authentication in the other files as well. Or, perhaps, it does exist and I haven’t discovered its uses, but this idea lends concern almost as much as deducing nonexistent authentication does.

*DocData* features some hard coding, namely with database connections — local host 3060 utilizes the database name to establish connection; this can be easily breached. Hard coding, generally, renders extreme difficulty in keeping up with developing safety measures and exponentially adds to the number of “endpoints” that will need to be manually re-scripted to include these updates.

Why are business names sent as request parameters in the *CRUDController* file? This is a method that can be easily manipulated and greenlit.

There doesn’t appear to be any sort of encryption or Cryptography present in, well, any of the files — at least as far as I can tell. Encryption is *vital* in safely communicating or storing data from server to client and client to server.

## Static Testing

A few vulnerable dependencies were discovered in running a dependency check on *Client’s* code:

The Bouncy Castle JCE Provider 1.46 has 16 vulnerabilities.

Method of solution: upgrade to **1.61** if possible (at least 1.60).

CVE-2013-1624

CVE-2015-6644

CVE-2015-7940

CVE-2016-1000338

CVE-2016-1000339

CVE-2016-1000341

CVE-2016-1000342

CVE-2016-1000343

CVE-2016-1000344

CVE-2016-1000345

CVE-2016-1000346

CVE-2016-1000352

CVE-2017-13098

CVE-2018-5382

CVE-2020-0187

CVE-2020-26939

Apache Spring Boot 2.2.4.RELEASE has 13 vulnerabilities.

Method of solution: adopt Spring Framework of at least version **5.3.20+**.

\*\* *THIS IS OF CRITICAL SEVERITY*. \*\*

CVE-2013-4152

CVE-2013-7315

CVE-2014-0054

CVE-2016-1000027

CVE-2018-11039

CVE-2018-11040

CVE-2018-1257

CVE-2020-5421

CVE-2022-22950

CVE-2022-22965

CVE-2022-22968

CVE-2022-22970

CVE-2022-27772

The Logback-Core Module 1.2.3 has one vulnerability.

Method of solution: update the module to at least version **1.2.8**.

CVE-2021-42550

The Apache Log4j API 2.12.1 has five vulnerabilities.

Method of solution: update to at least version **2.17.1**.

\*\* *THIS IS OF CRITICAL SEVERITY. \*\**

CVE-2020-9488

CVE-2021-44228

CVE-2021-44832

CVE-2021-45046

CVE-2021-45105

The Apache Snake YAML 1.25 parser and emitter for Java has 6 vulnerabilities.

Methods of solution: update to version **1.31** and implement *DoS attack preventions*.

CVE-2017-18640

CVE-2022-25857

CVE-2022-38749

CVE-2022-38750

CVE-2022-38751

CVE-2022-38752

The Apache FasterXML Jackson Databind API 2.10.2 has 4 vulnerabilities.

Methods of solution: update to at least version **2.14.0** and secure entity expansion.

CVE-2020-25649

CVE-2020-36518

CVE-2022-42003

CVE-2022-42004

The Apache Tomcat 9.0.30 core implementation has 20 vulnerabilities.

Method of solution: update to at least version **10.1.0-M6**.

\*\* *THIS IS OF CRITICAL SEVERITY. \*\**

CVE-2019-17569

CVE-2020-11996

CVE-2020-13934

CVE-2020-13935

CVE-2020-13943

CVE-2020-17527

CVE-2020-1935

CVE-2020-1938

CVE-2020-8022

CVE-2020-9484

CVE-2021-24122

CVE-2021-25122

CVE-2021-25329

CVE-2021-30640

CVE-2021-33037

CVE-2021-41079

CVE-2021-43980

CVE-2022-29885

CVE-2022-34305

CVE-2022-42252

The Apache Spring Boot Starter 2.2.4.RELEASE has 1 vulnerability.

Method of solution: update to at least version **v2.2.11.RELEASE**.

CVE-2022-27772

The Apache Hibernate Bean Validator 6.0.18.FINAL has 1 vulnerability.

Methods of solution: update to at least version **6.1.3** and implement further validation for EL expressions.

CVE-2020-10693

The Apache Spring Core 5.2.3.RELEASE has 10 vulnerabilities.

Method of solution: update to at least version **5.3.20+**.

\*\* *THIS IS OF CRITICAL SEVERITY. \*\**

CVE-2016-1000027

CVE-2020-5421

CVE-2021-22060

CVE-2021-22096

CVE-2021-22118

CVE-2022-22950

CVE-2022-22965

CVE-2022-22968

CVE-2022-22970

CVE-2022-22971

## Mitigation Plan

The first matter at hand for mitigation is to update the dependencies that are listed above. The vulnerabilities presented in the listed dependencies appear to rest in outdated formats or versions. The first step would be to update the dependencies before attempting to correct them as a secondary additive in updating them would be minimizing the number of false positives.

Cryptography is incredibly lacking in *Client’s* code. Since this is a heavy and successful manner of security and privacy assurance, encryption and decryption methods should be adopted and utilized. Something to consider could be E2E encryption unless access to whichever end of encryption would be necessary, but then further security measures must be taken to protect the means of encrypting/decrypting.

While some hard coding is permissible, hard coding database connections is a significant flaw. Delving into the *DocData* file and rectifying this dangerous scripting should be prioritized. Perhaps passing the database as an encrypted reference parameter instead of hard coding the database name with localhost:3060 can be a method utilized to rectify.

Authentication and validation methods should be implemented across the board. Input Validation is lacking in *GreetingController*, where a parameter should be verified before being requested. Some manner of authentication should occur in the *Customer* file since that file will be a driver in which type of user is accessing the information. Proper authentication and validation methods implemented in the *Customer* class will mitigate a lot of potential attacks right out of the gate.

GET is a successful HTML method of accessing data, but it is certainly not as safe privacy- and security-wise as a method like POST. GET requests the data and allows caching or storing of this data — subsequently, repeated requesting or adjusting of this data. POST does not allow this same caching and storing. POST allows a lot more parameter data that GET, where GET can be overloaded more easily. Rescripting the programs — especially those that handle passwords, usernames, and any sensitive information — in a POST manner should be absolutely prioritized to mitigate security concerns for *Client*.

## References

Klosowski, T. (2021, September 6). *The State of Consumer Data Privacy Laws in the US (and Why it Matters)*. The New York Times. Retrieved November 9, 2022, from <https://www.nytimes.com/wirecutter/blog/state-of-privacy-laws-in-us/>.

Manico, J., & Detlefsen, A. (2015). *Iron-Clad Java: Building Secure Web Applications*. McGraw-Hill Education.

Ozarslan, S. (2022, March 24). *Key Threats and Cyber Risks Facing Financial Services and Banking Firms in 2022*. Picus Security. Retrieved November 9, 2022, from <https://www.picussecurity.com/key-threats-and-cyber-risks-facing-financial-services-and-banking-firms-in-2022>.