****

# CS 305 Project One

**Artemis Financial Vulnerability Assessment Report**

**Table of Contents**

[Document Revision History 3](#_heading=h.gjdgxs)

[Client 3](#_heading=h.30j0zll)

[Instructions 3](#_heading=h.1fob9te)

[Developer 4](#_heading=h.3znysh7)

[1. Interpreting Client Needs 4](#_heading=h.2et92p0)

[2. Areas of Security 4](#_heading=h.tyjcwt)

[3. Manual Review 4](#_heading=h.3dy6vkm)

[4. Static Testing 4](#_heading=h.1t3h5sf)

[5. Mitigation Plan 4](#_heading=h.4d34og8)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **01/23/2022** | **William Marshburn** |  |

## 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

William Marshburn

## 1. Interpreting Client Needs

Artemis Financial is a financial consulting company that develops individualized financial plans for savings, retirement, investments, and insurance. Artemis is seeking to modernize operations. As we know, financial firms, banks and monetary institutions can be at higher risk of cyber-attack and exploitations due to the sensitive information they carry and the assets they hold and/or manage.

While generally every business should highly prioritize security, Artemis Financials’ value of secure communications is paramount. Security emphasis should be placed on both internal and external attacks. It is safe to assume that this company will likely be dealing with international transactions as well as domestic and while there are laws and regulations the federal government will have for domestic transactions, Artemis must remain resilient when dealing with international affairs as any breach of customer credentials would significantly reduce user confidence and reputation of this financial institution.

For financial institutions, external threats are always present. Many attempts will be made to gain unauthorized access via injection attacks, input manipulations, man-in-the-middle attacks, etc. (Manico & Detlefsen, 2015). Since financial institutions carry information related, but not limited to, user SSN, sign-in/biometric information, bank account numbers, or other financially related information, making sure secure measures are always utilized especially during the exchange of information between client to user is paramount. Any vulnerabilities that allow exploits from the user/client end or the potential of a hacker to see leaked information from an insecure exchange would allow unauthorized access and abuse to what should remain a secure system.

Regarding the “modernization” requirements that must be considered, attackers are always reinventing the wheel and utilizing new avenues to exploit and hack into software to obtain information, financial gain, etc. While the use of third-party and open-source libraries can be very helpful, it is very important that the use of outside resources is thoroughly vetted before being used at Artemis Financial. Furthermore, since new threats are emerging each day, it is of utmost importance to keep all software and applications up to date as updated software will have patches that increase security and stop formerly known exploits from occurring.

## 2. Areas of Security

**Input Validation** – Input validation is very important as many attacks can start from logging credentials to access Artemis Financials’ web application. Anywhere the code takes input or data is passed into the system, specifically in API or database sections, it is imperative that validation and error handling is applied (Manico & Detlefsen, 2015).

**APIs** – APIs generally define how a user interacts and exchanges information with a client. Ensuring the API remains secure is necessary because of the information that can be intercepted if the API and additional software working in tandem with the API is not properly secured.

**Cryptography** – Given that Artemis Financial will be dealing with both domestic and international clients, utilizing cryptography will be necessary as this will encrypt the data sent to and from users. Encrypting the data adds another layer of security as anyone that may be trying to gain access to the data will likely be unable to decrypt it.

**Code Error** – This should be implemented throughout. Code error should properly display errors to the user and log them for review. Using error handling in combination with API’s and input validation will help the application remain secure and prevent unauthorized privileges or access.

**Code Quality** – Since the application will be using multiple architectures from the Vulnerability Assessment Flow Diagram, it is important to make sure the code quality is correctly used, implemented, and updated accordingly if not correctly implemented.

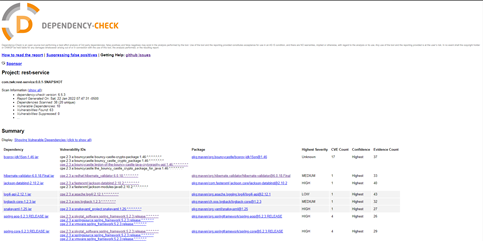
## 3. Manual Review

Utilizing the VAFPD, the manual review was first conducted to assess any vulnerabilities that may arise from lack of proper input validation. In the greetingController.java and CRUDController.java files, there does not appear to be any input validation and this needs to be addressed. Both files contain passage of type “String” data input. Both files could fall victim to an injection attack if not properly validated (Manico & Detlefsen, 2015).

Although the program takes input and data, there is no API in the code base currently, successful implementation of a secure API will be necessary for Artemis Financial to move forward securely (Manico & Detlefsen, 2015). There does not appear to be any forms of cryptography used and this will be necessary given Artemis Financial will be conducting both domestic and international transactions.

There is only one error handling technique used and this is in DocData.java. In myDateTime.java the initialized fields are not set to private, and this could be exploited and manipulated (OWASP Secure Coding Practices Quick Reference Guide, n.d.). The code quality thus far is fair, but implementing input validation, including error handling, including a secure API, utilizing secure techniques when accepting data and privatizing the access of fields and variables throughout the program will be necessary to keep security strong.

**4. Static Testing**



## 

## bcprov-jdk15on-1.46.jar (CVE-2016-1000352) - In the Bouncy Castle JCE Provider version 1.55 and earlier the ECIES implementation allowed the use of ECB mode. This mode is regarded as unsafe and support for it has been removed from the provider.

**hibernate-validator-6.0.18.Final.jar (CVE-2020-10693) -** 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.

**jackson-databind-2.10.2.jar (CVE-2020-25649)** - 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 (CVE-2021-44832)** - Apache Log4j2 versions 2.0-beta7 through 2.17.0 (excluding security fix releases 2.3.2 and 2.12.4) are vulnerable to a remote code execution (RCE) attack when a configuration uses a JDBC Appender with a JNDI LDAP data source URI when an attacker has control of the target LDAP server. This issue is fixed by limiting JNDI data source names to the java protocol in Log4j2 versions 2.17.1, 2.12.4, and 2.3.2.**logback-core-1.2.3.jar (CVE-2021-42550) -** 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)** - The Alias feature in SnakeYAML 1.18 allows entity expansion during a load operation, a related issue to CVE-2003-1564.

**spring-aop-5.2.3.RELEASE.jar (CVE-2021-22060)** - In Spring Framework versions 5.3.0 - 5.3.13, 5.2.0 - 5.2.18, and older unsupported versions, it is possible for a user to provide malicious input to cause the insertion of additional log entries. This is a follow-up to CVE-2021-22096 that protects against additional types of input and in more places of the Spring Framework codebase.

**spring-core-5.2.3.RELEASE.jar (CVE-2021-22060 & CVE-2020-5421) -** In Spring Framework versions 5.3.0 - 5.3.13, 5.2.0 - 5.2.18, and older unsupported versions, it is possible for a user to provide malicious input to cause the insertion of additional log entries. This is a follow-up to CVE-2021-22096 that protects against additional types of input and in more places of the Spring Framework codebase.

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 may be bypassed depending on the browser used using a jsessionid path parameter.

**tomcat-embed-core-9.0.30.jar & tomcat-embed-websocket-9.0.30.jar (CVE-2021-42340)** - The fix for bug 63362 present in Apache Tomcat 10.1.0-M1 to 10.1.0-M5, 10.0.0-M1 to 10.0.11, 9.0.40 to 9.0.53 and 8.5.60 to 8.5.71 introduced a memory leak. The object introduced to collect metrics for HTTP upgrade connections was not released for WebSocket connections once the connection was closed. This created a memory leak that, over time, could lead to a denial of service via an OutOfMemoryError.

## 5. Mitigation Plan

**Input Validation:** Incorporate input validation, when doing so make sure to validate all client data properly before processing any data. Validate input with characters that you are expecting to receive (whitelisting) vs what you think someone could use (Manico & Detlefsen, 2015). Validate date and data ranges of inputs. Define wrappers around methods. Make further use of query parameterization provided by Java. If any inputs fail, do not allow the application to run (OWASP Secure Coding Practices Quick Reference Guide, n.d.).

**APIs:** Implement task-specific API’s and do not allow applications to issue commands directly to the OS (OWASP Secure Coding Practices Quick Reference Guide, n.d.).

**Cryptography:** Include cryptography to keep data safe. Utilize HTTPS to ensure secure and safe transport of data and information. Submit sensitive data over HTTPS POST (Manico & Detlefsen, 2015).

**Code Error:** Improve error checking and logging by incorporating additional error handling techniques in the code base. Make sure that errors, untrusted data, and unusual events are not able to run. Make sure all errors are correctly logged (OWASP Secure Coding Practices Quick Reference Guide, n.d.).

**Code Quality:** Continue to ensure that error checking and input validation are implemented properly and efficiently. Make sure code in use is not consuming unnecessary amounts of memory. For example, checking the buffer boundaries when using a loop and/or making sure there is no chance of writing past allowed space. Properly free used space once done. When possible, use prior tested and safe blocks of code rather than creating new, untested code (OWASP Secure Coding Practices Quick Reference Guide, n.d.).

**Dependency Check for Files:** In reference to the static report above, please update and continue to update all dependencies as the updates include patches or fixes that will safeguard the application from vulnerabilities listed in the static report.

**bcprov-jdk15on-1.46.jar (CVE-2016-1000352**) – Update to later version.

**logback-core-1.2.3.jar (CVE-2021-42550) –** Update to later version. Only provide as much authorization and privileges to complete a task as necessary (principle of least privilege), in the event someone does gain access to a system, they will already be limited in what they can do when they’ve accessed the system illegally (Manico & Detlefsen, 2015).

**snakeyaml-1.25.jar (CVE-2017-18640) –** Upgrade to a later version. Apply input validation techniques that verifies trusted user input and validates that it is of acceptable size. This is vulnerable YAML expansion and can potentially result in memory wasting and DoS.

**spring-aop-5.2.3.RELEASE.jar (CVE-2021-22060) –** Update to later version.

**spring-core-5.2.3.RELEASE.jar (CVE-2021-22060 & CVE-2020-5421)** – Update to later version.

**tomcat-embed-core-9.0.30.jar & tomcat-embed-websocket-9.0.30.jar (CVE-2021-42340) –** Update to later version.

**References**

Manico, J., & Detlefsen, A. (2015). *Iron-clad Java building secure web applications*. New York, Ny Mcgraw-Hill Education.

*OWASP Secure Coding Practices Quick Reference Guide*. (n.d.). OWASP; OWASP. Retrieved Autumn 1 C.E., from https://owasp.org/www-pdf-archive/OWASP\_SCP\_Quick\_Reference\_Guide\_v1.pdf