****

# 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 6](#_Toc32574613)

[4. Static Testing 6](#_Toc32574614)

[5. Mitigation Plan 10](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **November 12, 2023** | **David Cruz** |  |

## Client



## Developer

David Cruz

## Interpreting Client Needs

The client, Artemis Financial, is a consulting company that creates financial plans that are specific to their clients. With the importance of software security growing larger in today’s current cyber climate, Artemis Financial is looking to bolster its software security and modernize its operation systems to the most effective security that is current and up to date. The client’s web application interface currently uses the REST API, which is a step in the correct direction but seeks expertise to examine their systems and find vulnerabilities as well as give suggestions about how to further improve the security of their systems to better protect against external threats.

In this vulnerability assessment, security vulnerabilities specific to the client’s web application will be analyzed and will be documented in this report. Moreover, there are a few characteristics of the client’s company that need to be considered in this report. For instance, the company has different clients that have different financial needs tailored to them, so effective communication with their customers holds great importance. That said, there always lies the possibility that sensitive financial information is transferred with Artemis Financial and their clients, which means that secure communications between them and their customers need to be built into the web application.

This is especially important due to the possibility of the client making international transactions. There may be different rules and regulations regarding security and privacy when dealing with clients in different parts of the world. In Europe for example, the General Data Protection Regulation (GDPR) “applies when ‘personal data’ are ‘processed’. The GDPR defines ‘processing’ as ‘any operation or set of operations which is performed on personal data or on sets of personal data, whether or not by automated means” (Hoofnagle et al., 2019, p.72). In addition, some countries may have stricter data policies that restrict encryption schemes or may require that data be surveilled according to their laws which may conflict with the customer’s and our client’s privacy. Therefore, Artemis Financial would need to either comply with these restrictions or deal with potential consequences due to not complying with different government rules and regulations. Additionally, there is always the threat of dealing with cyber threats from government adversaries that look to spy on, steal, or sabotage data.

Building from the cyber threats that can be seen from foreign adversaries, there is the possibility of seeing external threats in the homeland in the present and the immediate future. As stated earlier, the client has a RESTful web API which can be susceptible to API attacks like JSON injection. For instance, if the input isn’t properly sanitized or the system fails to use proper input validation, an unauthorized user can modify JSON messages maliciously and compromise the security of the REST API (Nidecki & Mussler, 2023).

To ensure that the client’s systems are modernized, it will be important to be sure that they are using updated web technologies. This will help both the functionality and the security of their web application but may require an occasional rework of the system to ensure the system stays compatible with the most recent web technologies.

## Areas of Security

The following areas of security apply to Artemis Financials’ web application:

Input Validation

When communicating with their clients, Artemis Financial will need to get detailed personal and financial information to best create personalized financial plans for their clients. This information will need to be collected through input forms, so measures to validate user input will be paramount. Because collecting client information is one of the first steps in the process, input validation can be seen as the “first line of defense” in a web application (Manico & Detlefsen, 2014, p.18). Sanitizing the input can prevent injection attacks like SQL injection and JSON injection which could compromise the web application.

APIs

The application is using a RESTful API when it connects to the database, so this area of security will also be a primary focus. Because the client is already using the REST API, they are already taking advantage of HTTPS and other protocols to prevent API attacks. There are certain characters used within the API that would need to be escaped in untrusted input to prevent an injection (Manico & Detlefsen, 2014, ch.7). This also goes for certain commands that are used to prevent command injections.

Cryptography

The data that Artemis Financial will be handling will most certainly be sensitive and confidential, so using encryption algorithms and encryption keys will be necessary to protect this data while in transit and while at rest. This will be especially important when performing communications internationally to protect against foreign hackers and adversaries. The system can use public key encryption to speed up computational time, and symmetric key cryptography can be used for encryption/decryption once the key exchange is made (Maqsood et al., 2017, p.442).

Code Error

The web application should be able to handle errors and exceptions properly. Failure to do so may expose sensitive financial information through exception handling, which may lead to further vulnerabilities in the web application down the line. For example, the user can run into an HTTP 404 error which is common, but if not handled correctly, it can expose information about the server such as the server version number (Manico & Detlefsen, 2014, p.231).

Code Quality

Quality code is the foundation of a secure web application. Following secure coding guidelines can enable the security of the code to be baked into the web application from its foundation. This will already put the web application in a safer spot in terms of security since the code has been created to minimize the exposure of vulnerabilities. Using guides like the OWASP Secure Coding Practices can help the client understand the importance of software security and prevent programmers from making coding errors that jeopardize the security of the application “The goal of software security is to maintain the confidentiality, integrity, and availability of information resources in order to enable successful business operations” (OWASP Foundation, 2010., p. 3).

## Manual Review

After reviewing the code, there are a few areas where the code is vulnerable. First, there does not seem to be any input validation implemented anywhere within the code. This will be an issue and open the web application to possible injection attacks (SQL, JSON, etc.). This can be seen in the GreetingController class when the name parameter is passed to the String.format method with no input validation.

Second, it doesn’t look like the code has any protocols implemented in the API. The code should use HTTPS and API keys at the very least to protect against attacks like brute force attacks and man-in-the-middle attacks. HTTPS can encrypt the data transfer between the web application and the server, and the API keys can be used to authenticate the users that use the web application.

Third, on the subject of encryption, the web application code does not implement any encryption algorithm to protect data in transit or at rest. Since Artemis Financial will be dealing with clients across the world, the use of encryption is a must-have. Failure to do so can expose the web application’s data, giving malicious users access to sensitive financial information. The most common are AES or RSA for encryption and decryption. Cryptography is one of the strongest and most common ways to secure data from attackers and prevents the attacker from having the ability to read the original data easily (Abdullah, 2017, para.2).

## Static Testing

Dependency Check

A dependency check was run on the web application code using the Maven Dependency Check. This dependency check uses the U.S. Government’s National Vulnerability Database (NVD) to identify any security issues seen within the code’s dependencies and provides a report describing the vulnerabilities that it found. Below is a screenshot of the report containing the vulnerabilities that were found.

A screenshot of a computer

Description automatically generated

Document Results

***Low Severity***

**CVE-2020-9488 Detail**

**Description**

Improper validation of certificate with host mismatch in Apache Log4j SMTP appender. This could allow an SMTPS connection to be intercepted by a man-in-the-middle attack which could leak any log messages sent through that appender. Fixed in Apache Log4j 2.12.3 and 2.13.1

**Mitigation:**

<https://issues.apache.org/jira/browse/LOG4J2-2819>

Upgrade to 2.13.2 which supports this feature. Previous versions can set the system property mail.smtp.ssl.checkserveridentity to true to globally enable hostname verification for SMTPS connections.

***Medium Severity***

**CVE-2020-10693 Detail**

**Description**

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.

**Mitigation:**

<https://bugzilla.redhat.com/show_bug.cgi?id=CVE-2020-10693>

You can pass user input as an expression variable by unwrapping the context to HibernateConstraintValidatorContext. Please refer to the https://in.relation.to/2020/05/07/hibernate-validator-615-6020-released/ and https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html\_single/#\_the\_code\_constraintvalidatorcontext\_code.

**CVE-2021-42550 Detail**

**Description**

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.

**Mitigation:**

Restrict the write access to the logback configuration file (logback.xml) to trusted personnel.

<https://cert-portal.siemens.com/productcert/pdf/ssa-371761.pdf>

***High Severity***

**CVE-2016-1000338 Detail**

**Description**

In Bouncy Castle JCE Provider version 1.55 and earlier the DSA does not fully validate ASN.1 encoding of signature on verification. It is possible to inject extra elements in the sequence making up the signature and still have it validate, which in some cases may allow the introduction of 'invisible' data into a signed structure.

**Mitigation:**

Before applying the update, back up your existing installation, including all applications, configuration files, databases and database settings, and so on. Installation instructions are located in the download section of the customer portal. The References section of this erratum contains a download link (you must log in to download the update). <https://access.redhat.com/errata/RHSA-2018:2669>

**CVE-2020-25649 Detail – FALSE POSITIVE**

**Description**

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.

**False Positive Justification:**

The Jackson-databind dependency is a library that coverts java objects to JSON. It doesn’t appear that the code uses JSON in any way, so this can be assumed to be a false positive.

***Critical Severity***

**CVE-2022-1471 Detail**

**Current Description**

SnakeYaml's Constructor() class does not restrict types which can be instantiated during deserialization. Deserializing yaml content provided by an attacker can lead to remote code execution. We recommend using SnakeYaml's SafeConsturctor when parsing untrusted content to restrict deserialization. We recommend upgrading to version 2.0 and beyond.

**Mitigation:**

SnakeYaml's Constructor() class does not restrict types which can be instantiated during deserialization. Deserializing yaml content provided by an attacker can lead to remote code execution. We recommend using SnakeYaml's SafeConsturctor when parsing untrusted content to restrict deserialization. <https://bitbucket.org/snakeyaml/snakeyaml/issues/561/cve-2022-1471-vulnerability-in#comment-64581479>

**CVE-2023-20873 Detail**

**Description**

In Spring Boot versions 3.0.0 - 3.0.5, 2.7.0 - 2.7.10, and older unsupported versions, an application that is deployed to Cloud Foundry could be susceptible to a security bypass. Users of affected versions should apply the following mitigation: 3.0.x users should upgrade to 3.0.6+. 2.7.x users should upgrade to 2.7.11+. Users of older, unsupported versions should upgrade to 3.0.6+ or 2.7.11+.

**Mitigation:**

No solution found for this vulnerability.

**CVE-2022-22965 Detail**

**Description**

A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e. the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it.

**Mitigation:**

Updating to the latest version can help mitigate the vulnerability.

<https://cert-portal.siemens.com/productcert/pdf/ssa-254054.pdf>

**CVE-2016-1000027 Detail**

**Description**

Pivotal Spring Framework through 5.3.16 suffers from a potential remote code execution (RCE) issue if used for Java deserialization of untrusted data. Depending on how the library is implemented within a product, this issue may or not occur, and authentication may be required. NOTE: the vendor's position is that untrusted data is not an intended use case. The product's behavior will not be changed because some users rely on deserialization of trusted data.

**Mitigation:**

No solution found for this vulnerability.

**CVE-2020-1938 Detail**

**Description**

When using the Apache JServ Protocol (AJP), care must be taken when trusting incoming connections to Apache Tomcat. Tomcat treats AJP connections as having higher trust than, for example, a similar HTTP connection. If such connections are available to an attacker, they can be exploited in ways that may be surprising. In Apache Tomcat 9.0.0.M1 to 9.0.0.30, 8.5.0 to 8.5.50 and 7.0.0 to 7.0.99, Tomcat shipped with an AJP Connector enabled by default that listened on all configured IP addresses. It was expected (and recommended in the security guide) that this Connector would be disabled if not required. This vulnerability report identified a mechanism that allowed: - returning arbitrary files from anywhere in the web application - processing any file in the web application as a JSP Further, if the web application allowed file upload and stored those files within the web application (or the attacker was able to control the content of the web application by some other means) then this, along with the ability to process a file as a JSP, made remote code execution possible. It is important to note that mitigation is only required if an AJP port is accessible to untrusted users. Users wishing to take a defense-in-depth approach and block the vector that permits returning arbitrary files and execution as JSP may upgrade to Apache Tomcat 9.0.31, 8.5.51 or 7.0.100 or later. A number of changes were made to the default AJP Connector configuration in 9.0.31 to harden the default configuration. It is likely that users upgrading to 9.0.31, 8.5.51 or 7.0.100 or later will need to make small changes to their configurations.

**Mitigation:**

The vulnerability may be mitigated in customer environments that use firewalls to limit access to the AJP ports. The potential impact is additionally mitigated by the requirement that the attacker must have permissions to upload documents. <https://support.blackberry.com/kb/articleDetail?articleNumber=000062739>

## Mitigation Plan

After performing a manual code review and analyzing the results of the Maven Dependency Check, here is the recommended mitigation plan for the identified vulnerabilities.

To mitigate future vulnerabilities, make sure input validation is implemented to sanitize inputs into the system using techniques like whitelisting and blacklisting to ensure that all data going in and out of the system is safe and secure. This will help defend the web application from a multitude of injection attacks like SQL injection and JSON injection. One suggestion would be to take advantage of some of the frameworks Spring has to offer to validate the user’s input and sanitize input and output messages.

Next, to better protect the RESTful API, we will use HTTPS to encrypt traffic between the web API and the application. Also, authentication and authorization will be implemented to authorize users who use the web application so only authenticated users have access to the web API. Encryption algorithms will need to be included in the web application to encrypt and decrypt data in transit and at rest. Using strong encryption schemes like AES or RSA and storing the encryption keys on a secure server will ensure that data is protected.

The results of the Maven Dependency check further emphasized the importance of code quality. Using the Maven Dependency check throughout the development process can help identify vulnerabilities in the NST, where solutions for these vulnerabilities can also be found as well. Following coding guidelines from the OWASP Secure Coding Practices can help developers write more quality code and avoid making common mistakes that make the programs they develop vulnerable. We will also ensure that all errors and exceptions that may arise in the code are handled correctly and securely to prevent unwanted details about the system from becoming exposed and exploited by malicious users.

References

Abdullah, A. M. (2017). Advanced encryption standard (AES) algorithm to encrypt and decrypt data.

*Cryptography and Network Security*, 16(1), 11.

Hoofnagle, C. J., Van Der Sloot, B., & Borgesius, F. Z. (2019). The European Union general data protection

regulation: what it is and what it means. *Information & Communications Technology Law*, 28(1), 65-98.

Manico, J., & Detlefsen, A. (2014). *Iron-clad java: Building secure web applications*. McGraw-Hill Education.

Maqsood, F., Ahmed, M., Ali, M. M., & Shah, M. A. (2017). Cryptography: A comparative analysis for modern techniques. *International Journal of Advanced Computer Science and Applications*, 8(6).

Nidecki, T. A., & Mussler, B. D. (2023, February 21). *JSON injection*. Invicti. https://www.invicti.com/learn/json-injection/

*Owasp secure coding practices-quick reference guide*. OWASP Secure Coding Practices-Quick Reference Guide | OWASP Foundation. (n.d.). https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/