****

# 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** | **03/22/2024** | **Matt Knutson** | **N/A** |

## Client



## Developer

Matt Knutson

## Interpreting Client Needs

Artemis Financial is a large consulting firm that handles the financial portfolio of their customers. These portfolios include extremely sensitive customer data, which consists of their savings and retirement accounts, their personal investments, and their insurance policies. With that in mind, they are looking to upgrade and revamp their system in order to offer top notch security and a modern user experience.

## Areas of Security

* Input Validation: Artemis Financial will use Input Validation to ensure that a user is only able to enter the proper or expected data values throughout the software. Input Validation will help to prevent nefarious data from being injected into the system. This helps prevents attacks that use data tampering as a means to exploit vulnerabilities and gain unauthorized access. A denial-of-service attack is commonly used by flooding the system with improper data.
* Secure API Interactions: Since the API is the link between the system (i.e. the server and database) and the user or intranet, its security should be taken very seriously. Therefore, this project will use a RESTful API and a Spring Architecture in order to implement security measures such as requiring Authorization and Authentication, implementing Access Controls, and using protocols such as TLS. This will help prevent a multitude of attacks, including the Man-in-the-Middle attack, SQL Injection, and the more common Denial of Service attack.
* Cryptography: Since Artemis Financial will be handling and transmitting some of the most important and confidential information a user can possess, they will need to utilize heavy encryption policies. Transport Layer security will be used between the server/database and the internet. While End-to End Encryption will be used between the client and the company to ensure data integrity.
* Secure Error Handling: This project will use Exception and Error Handling throughout the code in collaboration with Input Validation to ensure that warnings are triggered when improper input is entered. This assists with documenting and testing the program and can be insightful when implemented with informative comments.
* Secure Coding Practices: Artemis Financial will rigorously use Encapsulation throughout the code. All programmers working on the project will need to use Secure Coding Practices such as creating “private” and “final” variables, as well as use Query Parameterization to parametrize all variables and completely eliminate the threat of a SQL attack.
* Secure Data Structures: Artemis Financial will use Secure Data Structures in order to keep the system from being manipulated and used for privilege escalations. Access Controls will be implemented by enforcing strict authorization practices, allowing only certain groups and users access to particular data. Authentication will be required for access from any user of the system.

## Manual Review

A screenshot of a computer

Description automatically generated

As we can see, the Artemis Financial application is relatively small, only consisting of 8 source classes and 1 test class. The Controller and RestService classes are the main contact points for the RESTful API. The Customer class gathers and stores information pertaining to the customer, while the Greeting class delivers a default, or custom greeting to the customer.

A computer screen shot of a program code

Description automatically generated

A black screen with white text

Description automatically generated

By looking at the source code, it is apparent that Artemis Financial is already using the Spring Architecture and a RESTful API. By calling the RestController annotation, requests are being redirected and processed using REST.

A black background with white text

Description automatically generated

A black background with white text

Description automatically generated

A black background with text

Description automatically generated

The prior programmers have already been consistent when using Encapsulation practices within the program. By using Private and Final variables in the application, the risk for attacks and privilege escalations is dramatically reduced. This tactic keeps malicious users from tampering with and changing Global variables in the system.

## Static Testing

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A close-up of a computer screen

Description automatically generated

1. The Bouncy Castle Crypto Package

A screenshot of a computer

Description automatically generated

Issues:

* 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.
* ECDSA does not fully validate ASN.1 encoding of signature on verification.
* DSA key pair generator generates a weak private key if used with default values.
* ECIES implementation allowed the use of ECB mode. This mode is regarded as unsafe and support for it has been removed from the provider.
* DSA signature generation is vulnerable to timing attack. Where timings can be closely observed for the generation of signatures, the lack of blinding in 1.55, or earlier, may allow an attacker to gain information about the signature's k value and ultimately the private value as well.
* DHIES/ECIES CBC mode vulnerable to padding oracle attack.
* BouncyCastle TLS prior to version 1.0.3, when configured to use the JCE (Java Cryptography Extension) for cryptographic functions, provides a weak Bleichenbacher oracle when any TLS cipher suite using RSA key exchange is negotiated. An attacker can recover the private key from a vulnerable application. This vulnerability is referred to as "ROBOT."
* Bouncy Castle BC Java before 1.66, BC C# .NET before 1.8.7, BC-FJA before 1.0.1.2, 1.0.2.1, and BC-FNA before 1.0.1.1 have a timing issue within the EC math library that can expose information about the private key when an attacker is able to observe timing information for the generation of multiple deterministic ECDSA signatures.
* Bouncy Castle BC Java before 1.66, BC C# .NET before 1.8.7, BC-FJA before 1.0.1.2, 1.0.2.1, and BC-FNA before 1.0.1.1 have a timing issue within the EC math library that can expose information about the private key when an attacker is able to observe timing information for the generation of multiple deterministic ECDSA signatures.
* In engineSetMode of BaseBlockCipher.java, there is a possible incorrect cryptographic algorithm chosen due to an incomplete comparison. This could lead to local information disclosure with no additional execution privileges needed. User interaction is not needed for exploitation.
* Bouncy Castle for Java before 1.73 contains a potential Denial of Service (DoS) issue within the Bouncy Castle org.bouncycastle.openssl.PEMParser class.
* In Legion of the Bouncy Castle BC before 1.61 and BC-FJA before 1.0.1.2, attackers can obtain sensitive information about a private exponent because of Observable Differences in Behavior to Error Inputs.
* Bouncy Castle For Java before 1.74 is affected by an LDAP injection vulnerability. The vulnerability only affects applications that use an LDAP CertStore from Bouncy Castle to validate X.509 certificates.
* In the Bouncy Castle JCE Provider version 1.55 and earlier the primary engine class used for AES was AESFastEngine. Due to the highly table-driven approach used in the algorithm it turns out that if the data channel on the CPU can be monitored the lookup table accesses are sufficient to leak information on the AES key being used.
* The Bouncy Castle Java library before 1.51 does not validate a point is withing the elliptic curve, which makes it easier for remote attackers to obtain private keys via a series of crafted elliptic curve Diffie Hellman (ECDH) key exchanges, aka an "invalid curve attack."
* The default BKS keystore uses an HMAC that is only 16 bits long, which can allow an attacker to compromise the integrity of a BKS keystore.
* The TLS implementation in the Bouncy Castle Java library before 1.48 and C# library before 1.8 does not properly consider timing side-channel attacks on a noncompliant MAC check operation during the processing of malformed CBC padding, which allows remote attackers to conduct distinguishing attacks and plaintext-recovery attacks via statistical analysis of timing data for crafted packets.
* The other party’s DH public key is not fully validated. This can cause issues as invalid keys can be used to reveal details about the other party's private key where static Diffie-Hellman is in use.
* Bouncy Castle in Android before 5.1.1 LMY49F and 6.0 before 2016-01-01 allows attackers to obtain sensitive information via a crafted application, aka internal bug 24106146.

2. Hibernator’s Bean Validator

A screenshot of a computer

Description automatically generated

Issues:

* 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.

3. Jackson Data Binding

A screenshot of a computer

Description automatically generated

Issues:

* 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.
* Jackson-databind before 2.13.0 allows a Java StackOverflow exception and denial of service via a large depth of nested objects.
* Allows attackers to cause a denial of service (2 GB transient heap usage per read) in uncommon situations involving JsonNode JDK serialization.
* In FasterXML jackson-databind before versions 2.13.4.1 and 2.12.17.1, resource exhaustion can occur because of a lack of a check in primitive value deserializers to avoid deep wrapper array nesting, when the UNWRAP\_SINGLE\_VALUE\_ARRAYS feature is enabled.
* Resource exhaustion can occur because of a lack of a check in BeanDeserializer.\_deserializeFromArray to prevent use of deeply nested arrays. An application is vulnerable only with certain customized choices for deserialization.
* Jackson-databind through 2.15.2 allows attackers to cause a denial of service or other unspecified impact via a crafted object that uses cyclic dependencies.

4. The Apache Log4j API

A screenshot of a computer

Description automatically generated

Issues:

* 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.

5. Logback-Core Module

A screenshot of a computer

Description automatically generated

Issues:

* A serialization vulnerability in logback receiver component part of

logback version 1.4.11 allows an attacker to mount a Denial-Of-Service

attack by sending poisoned data.

* 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.

6. YAML 1.1 Parser & Emitter For Java

A screenshot of a computer

Description automatically generated

Issues:

* 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.
* The Alias feature in SnakeYAML before 1.26 allows entity expansion during a load operation.
* The package org.yaml:snakeyaml from 0 and before 1.31 are vulnerable to Denial of Service (DoS) due missing to nested depth limitation for collections.
* Using snakeYAML to parse untrusted YAML files may be vulnerable to Denial-of-Service attacks (DOS). If the parser is running on user supplied input, an attacker may supply content that causes the parser to crash by stackoverflow.

7. Spring Boot & Spring-Boot-Starter

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Issues:

* 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.
* Spring-boot versions prior to version v2.2.11.RELEASE was vulnerable to temporary directory hijacking.
* In Spring Boot versions 3.0.0 - 3.0.6, 2.7.0 - 2.7.11, 2.6.0 - 2.6.14, 2.5.0 - 2.5.14 and older unsupported versions, there is potential for a denial-of-service (DoS) attack if Spring MVC is used together with a reverse proxy cache.

8. Spring-Core

A screenshot of a computer

Description automatically generated

Issues:

* 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.
* In Spring Framework, versions 5.2.x prior to 5.2.15 and versions 5.3.x prior to 5.3.7, a WebFlux application is vulnerable to a privilege escalation: by (re)creating the temporary storage directory, a locally authenticated malicious user can read or modify files that have been uploaded to the WebFlux application or overwrite arbitrary files with multipart request data.
* 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 through the use of a jsessionid path parameter.
* It is possible for a user to provide a specially crafted SpEL expression that may cause a denial-of-service condition.
* In spring framework versions prior to 5.3.20+ , 5.2.22+ and old unsupported versions, applications with a STOMP over WebSocket endpoint is vulnerable to a denial-of-service attack by an authenticated user.
* The patterns for disallowedFields on a DataBinder are case sensitive which means a field is not effectively protected unless it is listed with both upper and lower case for the first character of the field, including upper and lower case for the first character of all nested fields within the property path.
* In spring framework versions prior to 5.3.20+ , 5.2.22+ and old unsupported versions, applications that handle file uploads are vulnerable to DoS attack if they rely on data binding to set a MultipartFile or javax.servlet.Part to a field in a model object.
* It is possible for a user to provide malicious input to cause the insertion of additional log entries.

9. Spring Web & Spring Web MVC

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Issues:

* 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.
* 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.
* Applications that use UriComponentsBuilder to parse an externally provided URL (e.g. through a query parameter) AND perform validation checks on the host of the parsed URL may be vulnerable to a open redirect https://cwe.mitre.org/data/definitions/601.html  attack or to a SSRF attack if the URL is used after passing validation checks.
* In Spring Framework, versions 5.2.x prior to 5.2.15 and versions 5.3.x prior to 5.3.7, a WebFlux application is vulnerable to a privilege escalation: by (re)creating the temporary storage directory, a locally authenticated malicious user can read or modify files that have been uploaded to the WebFlux application or overwrite arbitrary files with multipart request data.
* In Spring Framework versions 5.3.0 - 5.3.18, 5.2.0 - 5.2.20, and older unsupported versions, the patterns for disallowedFields on a DataBinder are case sensitive which means a field is not effectively protected unless it is listed with both upper and lower case for the first character of the field, including upper and lower case for the first character of all nested fields within the property path.
* It is possible for a user to provide malicious input to cause the insertion of additional log entries.
* In Spring Framework versions 5.3.0 - 5.3.10, 5.2.0 - 5.2.17, and older unsupported versions, it is possible for a user to provide malicious input to cause the insertion of additional log entries.

10. Tomcat Embed Core & Tomcat Embed Websocket

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Issues:

* 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.
* A specially crafted sequence of HTTP/2 requests sent to Apache Tomcat 10.0.0-M1 to 10.0.0-M5, 9.0.0.M1 to 9.0.35 and 8.5.0 to 8.5.55 could trigger high CPU usage for several seconds. If a sufficient number of such requests were made on concurrent HTTP/2 connections, the server could become unresponsive.
* An h2c direct connection to Apache Tomcat 10.0.0-M1 to 10.0.0-M6, 9.0.0.M5 to 9.0.36 and 8.5.1 to 8.5.56 did not release the HTTP/1.1 processor after the upgrade to HTTP/2. If a sufficient number of such requests were made, an OutOfMemoryException could occur leading to a denial of service.
* The payload length in a WebSocket frame was not correctly validated in Apache Tomcat 10.0.0-M1 to 10.0.0-M6, 9.0.0.M1 to 9.0.36, 8.5.0 to 8.5.56 and 7.0.27 to 7.0.104. Invalid payload lengths could trigger an infinite loop. Multiple requests with invalid payload lengths could lead to a denial of service.
* While investigating bug 64830 it was discovered that Apache Tomcat 10.0.0-M1 to 10.0.0-M9, 9.0.0-M1 to 9.0.39 and 8.5.0 to 8.5.59 could re-use an HTTP request header value from the previous stream received on an HTTP/2 connection for the request associated with the subsequent stream. While this would most likely lead to an error and the closure of the HTTP/2 connection, it is possible that information could leak between requests.
* When responding to new h2c connection requests, Apache Tomcat versions 10.0.0-M1 to 10.0.0, 9.0.0.M1 to 9.0.41 and 8.5.0 to 8.5.61 could duplicate request headers and a limited amount of request body from one request to another meaning user A and user B could both see the results of user A's request.
* Apache Tomcat 8.5.0 to 8.5.63, 9.0.0-M1 to 9.0.43 and 10.0.0-M1 to 10.0.2 did not properly validate incoming TLS packets. When Tomcat was configured to use NIO+OpenSSL or NIO2+OpenSSL for TLS, a specially crafted packet could be used to trigger an infinite loop resulting in a denial of service.
* Tomcat did not reject a request containing an invalid Content-Length header making a request smuggling attack possible if Tomcat was located behind a reverse proxy that also failed to reject the request with the invalid header.
* The HTTP/2 protocol allows a denial of service (server resource consumption) because request cancellation can reset many streams quickly, as exploited in the wild in August through October 2023.
* When using Apache Tomcat versions 10.0.0-M1 to 10.0.0-M4, 9.0.0.M1 to 9.0.34, 8.5.0 to 8.5.54 and 7.0.0 to 7.0.103 if a) an attacker is able to control the contents and name of a file on the server; and b) the server is configured to use the PersistenceManager with a FileStore; and c) the PersistenceManager is configured with sessionAttributeValueClassNameFilter="null" (the default unless a SecurityManager is used) or a sufficiently lax filter to allow the attacker provided object to be deserialized; and d) the attacker knows the relative file path from the storage location used by FileStore to the file the attacker has control over; then, using a specifically crafted request, the attacker will be able to trigger remote code execution via deserialization of the file under their control. Note that all of conditions a) to d) must be true for the attack to succeed.
* A vulnerability in the JNDI Realm of Apache Tomcat allows an attacker to authenticate using variations of a valid username and/or to bypass some of the protection provided by the LockOut Realm.
* The Form authentication example in the example’s web application displayed user provided data without filtering, exposing a XSS vulnerability.
* URL Redirection to Untrusted Site ('Open Redirect') vulnerability in FORM authentication feature Apache Tomcat.
* When serving resources from a network location using the NTFS file system, Apache Tomcat versions 10.0.0-M1 to 10.0.0-M9, 9.0.0.M1 to 9.0.39, 8.5.0 to 8.5.59 and 7.0.0 to 7.0.106 were susceptible to JSP source code disclosure in some configurations.
* Generation of Error Message Containing Sensitive Information vulnerability in Apache Tomcat.

## Mitigation Plan

* First, it is necessary to handle the overwhelming number of vulnerabilities that were discovered during the first dependency check. Many of these will be false positives, and once they are determined to be irrelevant, they can simply be suppressed. After all false positives have been identified and suppressed, another dependency check will be necessary. The remaining vulnerabilities will need to be patched, updated, or delt with appropriately according to their documented solutions. All currently unsupported technologies need to be eliminated and replaced as well.
* The Spring Architecture and RESTful API are already in place and will be utilized to enforce user and group roles, the principle of least privilege, user authentication, and Transport Layer Security between the company’s system and the web. End-to-End encryption will also be used to protect data in transit between the customers and the company site. Due to the sensitivity of the data being transmitted, and the legal ramifications of improperly handled data, 256-bit encryption will be used. Hashed passwords will be required for all user accounts.
* Encapsulation practices will continue to be used when programming to ensure that all the proper variables are privatized and finalized. Error checking and exception handling will need to be used in every class of the program. Tests will need to be built for every class as well, NOT just the RestServiceApplication class. This will increase the success of input validation throughout the entire program. All variables need to be parameterized in order to eliminate the threat of an attack on the company database. Any third-party libraries used by the system will need to be updated and monitored as time goes by and the program evolves.
* The software needs to be adaptable and fluid, continuously protecting any data involved. Therefore, the principle of least common mechanism will be used within the system. This will force every component of the system (libraries, drivers, operating system processes, etc....) to use a centralized security mechanism. This will enhance security, increase reliability, and allow the system to be adaptable.
* Artemis Financial will need to invest in proper training for their current and new employees. Educating the staff of the company is the only line of defense when it comes to attacks like shoulder surfing, dumpster diving, phishing emails, and all other attacks that rely on the manipulation of human beings.
* Artemis Financial will need to heavily investigate the laws and regulations that apply to their company domain. Since they will not currently be taking international customers, only US laws and laws concerning customer data and web activities will need to be considered. A few of the practices and standards are listed below:
  + GLBA - Enforces customer privacy and sharing practices.
  + FISMA - Protects Federal data and information such as a customer’s social security number.
  + HIPPA & HITECH – Protects customer’s health care data.
  + Sarbox - Ensures the accuracy and reliability of a customer’s finances.
  + PCIDSS – The credit card industry standard for online purchases.
* Finally, pen testing will need to be conducted on the finished system to verify it’s security. Dependency checks will also need to be occasionally run to check for new technologies, updates, and patches.

**Citations**

Manico, J., & Detlefsen, A. (2015). Iron-clad java: Building secure web applications. McGraw-Hill Education.

Spring. (2024). Spring Framework6.1.5. Spring Framework.

<https://spring.io/projects/spring-framework>

Oracle. (2023, May). Secure Coding Guidelines for Java SE.

<https://www.oracle.com/java/technologies/javase/seccodeguide.html>

SoftKraft. (2023, February 11). Web application architecture [complete guide & diagrams]. Medium.

<https://medium.com/@softkraft/web-application-architecture-complete-guide-diagrams-1b2d77fe3d2e>

Ruppert, S. (2024, January 16). Secure coding practices – input validation. Sven Ruppert.

<https://svenruppert.com/2023/12/13/secure-coding-practices-validation>.

Microsoft. (2024). What is Data Protection? Microsoft security. What Is Data Protection? | Microsoft Security. (n.d.).

<https://www.microsoft.com/en-us/security/business/security-101/what-is-data-protection>

Wallis, C. (2024). How to perform a vulnerability assessment: Step-by-step. RSS.

<https://www.intruder.io/guides/vulnerability-assessment-made-simple-a-step-by-step-guide>

Yasar, K. (2023, May 22). What is API security? Definition from TechTarget. App Architecture.

<https://www.techtarget.com/searchapparchitecture/definition/API-security>

Benzel, T., Irvine, C., & Levin, T. (n.d.). (PDF) design principles for security. faculty.nps.edu.

<https://www.researchgate.net/publication/265224436_Design_Principles_for_Security>

Srinivas. (2020, December 1). How is cryptography used in applications? Infosec.

<https://www.infosecinstitute.com/resources/cryptography/how-is-cryptography-used-in-applications>

Esplanada, R. Y. (2021, July 27). @Controller and @RestController annotations in Spring Boot. Stack Abuse.

<https://stackabuse.com/controller-and-restcontroller-annotations-in-spring-boot>

CSO Staff. (2023, September 12). Security and privacy laws, regulations, and compliance: The complete guide. CSO Online.

<https://www.csoonline.com/article/570281/csos-ultimate-guide-to-security-and-privacy-laws-regulations-and-compliance.html>