****

# CS 305 Project One

**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 5](#_Toc32574613)

[4. Static Testing 5](#_Toc32574614)

[5. Mitigation Plan 8](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **January 24, 2021** | **Derek Nill** |  |

## 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

Derek Nill

## 1. Interpreting Client Needs

Determine your client’s needs and potential threats and attacks associated with their application and software security requirements. Consider the following regarding how companies protect against external threats based on the scenario information:

* What is the value of secure communications to the company?
* Are there any international transactions that the company produces?
* Are there governmental restrictions about secure communications to consider?
* What external threats might be present now and in the immediate future?
* What are the “modernization” requirements that must be considered, such as the role of open source libraries and evolving web application technologies?

Secure communications are highly valuable to Artemis Financial. Sensitive information will be communicated to and from the server. This may include users financial and personal information such as bank accounts, investments, social security numbers and addresses. It is important that this data is transferred in a secure manner.

At this time, there are no known international transactions that the company produces. There are also no governmental restrictions about secure communications to consider.

There are many external threats that might be present now and in the future. A database will be required, so general database security as well as protection from SQL injection will be important. Other possible threats are broken authentication and sensitive data exposure.

When modernizing the web application, it will be important to keep in mind that any new libraries and frameworks used may come with their own vulnerabilities. Especially with the newest web application technologies that have only recently been used in production.

## 2. Areas of Security

Referring to the Vulnerability Assessment Process Flow Diagram, identify which areas of security are applicable to Artemis Financial’s software application. Justify your reasoning for why each area is relevant to the software application.

* **Input Validation:** The application will be accepting input from users. This includes personal information such as name and address as well as uploads of financial information. This input must be properly validated to prevent attempts to maliciously input information.]
* **APIs:** Artemis Financial is already using a RESTful API with their web application. It is important to make sure the API does not contain any vulnerabilities, especially in HTTP requests.
* **Cyrptography:** Sensitive information will be transmitted between client and server. It is important that this information is encrypted between communications instead of being transmitted in plain text.
* **Client/Server:** The web application will involve using a server and clients will connect to the server when in use. It is important that these connections are secure.
* **Code Quality:** Proper coding practices and patterns must be used, especially to ensure that proper authorization is in place.

## 3. Manual Review

Continue working through the Vulnerability Assessment Process Flow Diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

* There is currently no authentication/authorization system in place.
* The GET Mapping for /greeting takes in a variable titled name. There is no validation on this variable. A user could attempt using very long variables for /greeting?name=(thousands of characters).
* The CRUDController class has a mapping for /read which takes in a variable. There is no input validation on this variable.
* There needs to be exceptions for error handling when /read and /greeting are passed illegal variables.

## 4. Static Testing

Run a dependency check on Artemis Financial’s software application to identify all security vulnerabilities in the code. Record the output from dependency check report. Include the following:

* **bcprov-jdk15on-1.46.jar:** The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms.
  + **CVE-2013-1624:** 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.
  + **CVE-2015-6644:** An information disclosure vulnerability in Bouncy Castle could enable a local malicious application to gain access to user’s private information.
  + **CVE-2015-7940:** 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."
  + **CVE-2016-1000338:** 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.
  + **CVE-2016-1000339:** 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.
  + **CVE-2016-1000341:** In the Bouncy Castle JCE Provider version 1.55 and earlier DSA signature generation is vulnerable to timing attack.
  + **CVE-2016-1000342:** In the Bouncy Castle JCE Provider version 1.55 and earlier ECDSA does not fully validate ASN.1 encoding of signature on verification.
  + **CVE-2016-1000343:** In the Bouncy Castle JCE Provider version 1.55 and earlier the DSA key pair generator generates a weak private key if used with default values.
  + **CVE-2016-1000344:** In the Bouncy Castle JCE Provider version 1.55 and earlier the DHIES implementation allowed the use of ECB mode. This mode is regarded as unsafe and support for it has been removed from the provider.
  + **CVE-2016-1000345:** In the Bouncy Castle JCE Provider version 1.55 and earlier the DHIES/ECIES CBC mode vulnerable to padding oracle attack.
  + **CVE-2016-1000346:** In the Bouncy Castle JCE Provider version 1.55 and earlier the other party DH public key is not fully validated.
  + **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.
  + **CVE-2017-13098:** 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.
  + **CVE-2018-1000613:** Legion of the Bouncy Castle Java Cryptography APIs 1.58 up to but not including 1.60 contains a CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') vulnerability in XMSS/XMSS^MT private key deserialization that can result in Deserializing an XMSS/XMSS^MT private key can result in the execution of unexpected code.
  + **CVE-2018-5382:** Bouncy Castle BKS version 1 keystore (BKS-V1) files use an HMAC that is only 16 bits long, which can allow an attacker to compromise the integrity of a BKS-V1 keystore.
* **log4j-api-2.12.1.jar:** The Apache Log4j API.
  + **CVE-2020-9488**: Improper validation of certificate with host mismatch in Apache Log4j SMTP appender.
* **snakeyaml-1.25.jar:** YAML 1.1 parser and emitter for Java.
  + **CVE-2017-18640:** The Alias feature in SnakeYAML 1.18 allows entity expansion during a load operation, a related issue to CVE-2003-1564.
* **jackson-databind-2.10.2.jar:** General data-binding functionality for Jackson: works on core streaming API.
  + **CVE-2020-25649:** A flaw was found in FasterXML Jackson Databind, where it did not have entity expansion secured properly.
* **tomcat-embed-core-9.0.30.jar:** Core Tomcat implementation
  + **CVE-2019-17569:** The refactoring present in Apache Tomcat 9.0.28 to 9.0.30, 8.5.48 to 8.5.50 and 7.0.98 to 7.0.99 introduced a regression. The result of the regression was that invalid Transfer-Encoding headers were incorrectly processed leading to a possibility of HTTP Request Smuggling if Tomcat was located behind a reverse proxy that incorrectly handled the invalid Transfer-Encoding header in a particular manner.
  + **CVE-2020-11996:** 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.
  + **CVE-2020-13934:** 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.
  + **CVE-2020-13935:** 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.
  + **CVE-2020-13943:** If an HTTP/2 client connecting to Apache Tomcat 10.0.0-M1 to 10.0.0-M7, 9.0.0.M1 to 9.0.37 or 8.5.0 to 8.5.57 exceeded the agreed maximum number of concurrent streams for a connection (in violation of the HTTP/2 protocol), it was possible that a subsequent request made on that connection could contain HTTP headers - including HTTP/2 pseudo headers - from a previous request rather than the intended headers.
  + **CVE-2020-17527:** 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.
  + **CVE-2020-1935:** In Apache Tomcat 9.0.0.M1 to 9.0.30, 8.5.0 to 8.5.50 and 7.0.0 to 7.0.99 the HTTP header parsing code used an approach to end-of-line parsing that allowed some invalid HTTP headers to be parsed as valid. This led to a possibility of HTTP Request Smuggling if Tomcat was located behind a reverse proxy that incorrectly handled the invalid Transfer-Encoding header in a particular manner.
  + **CVE-2020-1938:** 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.
  + **CVE-2020-8022:** An Incorrect Default Permissions vulnerability in the packaging of tomcat.
  + **CVE-2020-9484:** Possible to trigger remote execution via deserialization of the file under control.
* **spring-core-5.2.3.RELEASE.jar:** Spring Core.
  + **CVE-2020-5421:** 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.
* **Solutions:**
  + **Bouncy Castle:** All listed vulnerabilities can be avoided by updating bouncy castle to the latest version, 1.68.
* **Apache Log4j API:** Log4J has fixed this issue in version 2.13.2, recommend update.
  + Source: <https://issues.apache.org/jira/browse/LOG4J2-2819>
* **SnakeYAML:** According to the supplied source, unless the YAML file is coming from an untrusted source, then this is a false positive.
  + - Source: <https://bitbucket.org/asomov/snakeyaml/wiki/Billion%20laughs%20attack>
* **Jackson-Databind:** Various versions of Jackson have been patched to fix this vulnerability. Recommend update to 2.6.7.4, 2.9.10.7, 2.10.5.1, or 2.11.0 and later.
  + Source: <https://github.com/FasterXML/jackson-databind/issues/2589>
  + **Tomcat Core:** Updating to 9.0.36 or later will fix all of the vulnerabilities listed for tomcat.
    - **Source:** <https://lists.opensuse.org/opensuse-security-announce/2020-03/msg00025.html>
    - **Source:** <https://lists.opensuse.org/opensuse-security-announce/2020-07/msg00084.html>
  + **Source:** <https://lists.apache.org/thread.html/r5541ef6b6b68b49f76fc4c45695940116da2bcbe0312ef204a00a2e0%40%3Cannounce.tomcat.apache.org%3E>
* **Spring Core:** The best that one can do is update to the latest version of the spring framework. However, more components keep popping up with this vulnerability. The most recent report is from about three months ago.
  + Source: <https://lists.apache.org/thread.html/r1c679c43fa4f7846d748a937955c7921436d1b315445978254442163@%3Ccommits.ambari.apache.org%3E>

## 5. Mitigation Plan

After interpreting your results from the manual review and static testing, identify the steps to remedy the identified security vulnerabilities for Artemis Financial’s software application.

* Use the Spring framework’s authentication and authorization system to create a username and password system for authentication, and a roles system for authorization.
* Put a character limit on /greeting?=name. Only take the first x characters of input and discard anything else.
* Add exceptions and error handling for /greeting and /read.
* Consider rewriting the code to use POST instead of GET, as it is generally a more secure way to transport data.
* Update all dependencies to latest versions.