****

# 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 6](#_Toc32574614)

[5. Mitigation Plan 13](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **09/13/2021** | **Sam Brasher** |  |

## 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

Sam Brasher

## 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?

Security is of utmost importance in the financial industry. We are trusted with customer’s financial information, and in many cases their security and future. Compromising any of these could have serious legal repercussions. Even if none of our systems are seriously affected, potential negative publicity could affect customer loyalty and therefore our return business. As the world of financial technology grows, so too does the drive to regulate it (InnReg, n.d.). Varying national governments will have a myriad of evolving rule sets to govern the licensing and regulation of financial technology, and, assuming Artemis would like to deal in international investments, we must comply with all regulations (United States Government Accountability Office, 2005). Due to the nature of our business, hackers and other malicious parties may try to compromise our system. We cannot let down our guard against any type of attack, whether physical or digital. Phishing attacks are by far the most prevalent type of attacks and seem to be increasing year over year. The FBI reports that there were 11 times as many phishing attacks in 2020 as in 2016 and was the most documented type of attack in 2020 (Rosenthal, 2021). Not only must we stay vigilant about protecting our own software, but we will also potentially be integrating with technology controlled by a third party. We must ensure that whatever companies we do business with are also up to date and proactive about their cybersecurity practices. This includes a continuous compliance review, accounting for new governmental regulations and newly discovered vulnerabilities in both our own software and any third parties we do business with.

## 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
  + It is fundamental that we do not directly execute any input received from a client. A maliciously crafted input could wreak havoc on our system, including compromising customer’s data or affecting usage of the system.
* Secure APIs
  + Building a secure API is the sole focus of our project here. It is incredibly important and must be considered at every turn. A centralized attribute-based access control layer will greatly enhance our security.
* Cryptography
  + While cryptography is incredibly important, it is outside the scope of this RESTful API. We should encrypt all stored data and network traffic, but secure data structures are more relevant to our scope.
* Secure Distributed Composing
  + Secure client/server interaction is fundamental in composing a RESTful API. We must account for untrusted data in the system, and not let a compromised client affect the server.
* Code Error
  + Improper error handling can cause unforeseen results in our system and should be avoided. This could compromise our data like customer’s financial information, or cause outages in our systems, which could affect a customer’s finances and ultimately affect our business.
* Code Quality
  + Secure coding practices need to be standard operating procedure. This area covers a lot of the other areas: proper error handling, input sanitization and validation, secure API interaction, etc.
* Encapsulation
  + While encapsulation is an important part of cybersecurity, it is a bit beyond the scope of this project. We will be interacting with the existing custom system and must use the data structures given.

## 3. Manual Review

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

In CRUDcontroller.java, the CRUD function does not sanitize the input. In addition, the data seems not to be used. I believe we should either sanitize and use the data, or not take unnecessary input. The CRUD function also creates an empty Doc\_Data object, which may or may not be the intended functionality. There needs to be error handling regarding an empty Doc\_Data object wherever it may be used, or here upon creation. In GreetingController.java, we again do not sanitize the input string. We should sanitize in either the controller, or in the Greeting object constructor function. In both cases, we should be validating input both by providing a validation check (like that one Doc\_Data object was created) and confirming the check is an expected value. This also includes accounting for null byte input strings, which could produce unexpected values later in the system.

In the myDateTime.java file, we should implement input sanitization for the mutator methods. In addition, we should probably store times in a predefined type – the LocalTime class in the java.time package. We should also make sure we’re returning a proper value for the accessor method. The current implementation involves passing an empty array, which could cause unexpected results. Note: this function has been marked as TODO.

In the Doc\_Data.java file, we make a connection to a database accounting for SQL errors but make no further action. We should finish the function and close the connection. In addition, depending on how we are going to use the Doc\_Data class, we should further fill out the constructor method – as it stands now a Doc\_Data object will have a null value for “id”, which could cause unexpected results. Also, it appears that we are broadcasting username and password (“root”) in plaintext which is not secure.

While seemingly not used, our customer class is not secure. There seems to be no system-wide access control, and we should implement a centralized controller to secure the customer class (and potentially other classes in the future). For example, not everyone should be able to access a customer’s account number. In addition, not everyone should be able to make a deposit into a customer’s account.

## 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:

1. The names or vulnerability codes of the known vulnerabilities
2. A brief description and recommended solutions provided by the dependency check report
3. Attribution (if any) that documents how this vulnerability has been identified or documented previously

**Bcprov-jdk15on-1.46.jar**

* 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, which allows remote attackers to conduct distinguishing attacks and plaintext-recovery attacks via statistical analysis of timing data for crafted packets.
  + <https://nvd.nist.gov/vuln/detail/CVE-2013-1624>
* CVE-2015-6644
  + 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2015-6644>
* 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."
  + <https://nvd.nist.gov/vuln/detail/CVE-2015-7940>
* 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2016-1000338>
* 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. There was also a leak in AESEngine although it was substantially less. AESEngine has been modified to remove any signs of leakage (testing carried out on Intel X86-64) and is now the primary AES class for the BC JCE provider from 1.56. Use of AESFastEngine is now only recommended where otherwise deemed appropriate.
  + <https://nvd.nist.gov/vuln/detail/CVE-2016-1000339>
* CVE-2016-1000341
  + In the Bouncy Castle JCE Provider version 1.55 and earlier 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2016-1000341>
* 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. 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2016-1000342>
* 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. If the JCA key pair generator is not explicitly initialised with DSA parameters, 1.55 and earlier generates a private value assuming a 1024 bit key size. In earlier releases this can be dealt with by explicitly passing parameters to the key pair generator.
  + <https://nvd.nist.gov/vuln/detail/CVE-2016-1000343>
* 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2016-1000344>
* CVE-2016-1000345
  + In the Bouncy Castle JCE Provider version 1.55 and earlier the DHIES/ECIES CBC mode vulnerable to padding oracle attack. For BC 1.55 and older, in an environment where timings can be easily observed, it is possible with enough observations to identify when the decryption is failing due to padding.
  + <https://nvd.nist.gov/vuln/detail/CVE-2016-1000345>
* CVE-2016-1000346
  + In the Bouncy Castle JCE Provider version 1.55 and earlier the other party 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. As of release 1.56 the key parameters are checked on agreement calculation.
  + <https://nvd.nist.gov/vuln/detail/CVE-2016-1000346>
* 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2016-1000352>
* 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. An attacker can recover the private key from a vulnerable application. This vulnerability is referred to as "ROBOT."
  + <https://nvd.nist.gov/vuln/detail/CVE-2017-13098>
* CVE-2018-1000613
  + Legion of the Bouncy Castle 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. This attack appear to be exploitable via A handcrafted private key can include references to unexpected classes which will be picked up from the class path for the executing application. This vulnerability appears to have been fixed in 1.60 and later.
  + <https://nvd.nist.gov/vuln/detail/CVE-2018-1000613>
* CVE-2018-5382
  + The default BKS keystore use an HMAC that is only 16 bits long, which can allow an attacker to compromise the integrity of a BKS keystore. Bouncy Castle release 1.47 changes the BKS format to a format which uses a 160 bit HMAC instead. This applies to any BKS keystore generated prior to BC 1.47. For situations where people need to create the files for legacy reasons a specific keystore type "BKS-V1" was introduced in 1.49. It should be noted that the use of "BKS-V1" is discouraged by the library authors and should only be used where it is otherwise safe to do so, as in where the use of a 16 bit checksum for the file integrity check is not going to cause a security issue in itself.
  + <https://nvd.nist.gov/vuln/detail/CVE-2018-5382>
* CVE-2020-26939
  + 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. This occurs in org.bouncycastle.crypto.encodings.OAEPEncoding. Sending invalid ciphertext that decrypts to a short payload in the OAEP Decoder could result in the throwing of an early exception, potentially leaking some information about the private exponent of the RSA private key performing the encryption.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-26939>

**Log4j-api-2.12.1.jar**

* CVE-2020-9488
  + 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-9488>

**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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2017-18640>

**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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-25649>

**Tomcat-embed-core-9.0.30.jar**

* 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. Such a reverse proxy is considered unlikely.
  + <https://nvd.nist.gov/vuln/detail/CVE-2019-17569>
* CVE-2020-11996
  + 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-11996>
* 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-13934>
* 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. Multiple requests with invalid payload lengths could lead to a denial of service.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-13935>
* 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. This could lead to users seeing responses for unexpected resources.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-13943>
* 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. 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-17527>
* 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. Such a reverse proxy is considered unlikely.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-1935>
* 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. 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 defence-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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-1938>
* CVE-2020-8022
  + A Incorrect Default Permissions vulnerability in the packaging of tomcat on SUSE Enterprise Storage 5, SUSE Linux Enterprise Server 12-SP2-BCL, SUSE Linux Enterprise Server 12-SP2-LTSS, SUSE Linux Enterprise Server 12-SP3-BCL, SUSE Linux Enterprise Server 12-SP3-LTSS, SUSE Linux Enterprise Server 12-SP4, SUSE Linux Enterprise Server 12-SP5, SUSE Linux Enterprise Server 15-LTSS, SUSE Linux Enterprise Server for SAP 12-SP2, SUSE Linux Enterprise Server for SAP 12-SP3, SUSE Linux Enterprise Server for SAP 15, SUSE OpenStack Cloud 7, SUSE OpenStack Cloud 8, SUSE OpenStack Cloud Crowbar 8 allows local attackers to escalate from group tomcat to root. This issue affects: SUSE Enterprise Storage 5 tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server 12-SP2-BCL tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server 12-SP2-LTSS tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server 12-SP3-BCL tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server 12-SP3-LTSS tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server 12-SP4 tomcat versions prior to 9.0.35-3.39.1. SUSE Linux Enterprise Server 12-SP5 tomcat versions prior to 9.0.35-3.39.1. SUSE Linux Enterprise Server 15-LTSS tomcat versions prior to 9.0.35-3.57.3. SUSE Linux Enterprise Server for SAP 12-SP2 tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server for SAP 12-SP3 tomcat versions prior to 8.0.53-29.32.1. SUSE Linux Enterprise Server for SAP 15 tomcat versions prior to 9.0.35-3.57.3. SUSE OpenStack Cloud 7 tomcat versions prior to 8.0.53-29.32.1. SUSE OpenStack Cloud 8 tomcat versions prior to 8.0.53-29.32.1. SUSE OpenStack Cloud Crowbar 8 tomcat versions prior to 8.0.53-29.32.1.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-8022>
* CVE-2020-9484
  + 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-9484>
* CVE-2021-24122
  + 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. The root cause was the unexpected behaviour of the JRE API File.getCanonicalPath() which in turn was caused by the inconsistent behaviour of the Windows API (FindFirstFileW) in some circumstances.
  + <https://nvd.nist.gov/vuln/detail/CVE-2021-24122>
* CVE-2021-25122
  + 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2021-25122>
* CVE-2021-25329
  + The fix for CVE-2020-9484 was incomplete. When using Apache Tomcat 10.0.0-M1 to 10.0.0, 9.0.0.M1 to 9.0.41, 8.5.0 to 8.5.61 or 7.0.0. to 7.0.107 with a configuration edge case that was highly unlikely to be used, the Tomcat instance was still vulnerable to CVE-2020-9494. Note that both the previously published prerequisites for CVE-2020-9484 and the previously published mitigations for CVE-2020-9484 also apply to this issue.
  + <https://nvd.nist.gov/vuln/detail/CVE-2021-25329>
* CVE-2021-30640
  + 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. This issue affects Apache Tomcat 10.0.0-M1 to 10.0.5; 9.0.0.M1 to 9.0.45; 8.5.0 to 8.5.65.
  + <https://nvd.nist.gov/vuln/detail/CVE-2021-30640>
* CVE-2021-33037
  + Apache Tomcat 10.0.0-M1 to 10.0.6, 9.0.0.M1 to 9.0.46 and 8.5.0 to 8.5.66 did not correctly parse the HTTP transfer-encoding request header in some circumstances leading to the possibility to request smuggling when used with a reverse proxy. Specifically: - Tomcat incorrectly ignored the transfer encoding header if the client declared it would only accept an HTTP/1.0 response; - Tomcat honoured the identify encoding; and - Tomcat did not ensure that, if present, the chunked encoding was the final encoding.
  + <https://nvd.nist.gov/vuln/detail/CVE-2021-33037>

**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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-10693>

**Spring-core-5.2.3.RELEASE.jar**

* 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-5421>
* CVE-2021-22118
  + 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2021-22118>

**Spring-jcl-5.2.3.RELEASE.jar**

* 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.
  + <https://nvd.nist.gov/vuln/detail/CVE-2020-5421>

## 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.

I will cover the various security vulnerabilities by package, starting with the package directly utilized by our application:

* Spring-core-5.2.3.RELEASE.jar / Spring-jcl-5.2.3.RELEASE.jar
  + CVE-2020-5421
    - This vulnerability allows bypass of protections from a previous vulnerability regarding JSON responses. We do not use JSON responses yet, so this can be considered a false positive. Luckily we can ensure that Spring Framework is updated to 5.2.9 to ensure this vulnerability is corrected (VMWare, 2020).
  + CVE-2021-22118
    - This vulnerability concerns WebFlux applications, which we do not use. This was fixed in version 5.2.15 of Spring Framework, which could make for an easy fix should we end up needing WebFlux.
* Bcprov-jdk15on-1.46.jar – BouncyCastle is an open-source cryptography library for Java. As we will cover, the version we use is critically out-of-date and vulnerable to several serious attacks. We do not use any encryption at the time of this writing, but we should, and we should take great care to either implement the referenced mitigations or update BouncyCastle to the latest version which patches these issues.
  + CVE-2013-1624
    - TLS is a standard of encryption succeeding SSL. We absolutely want to make sure our traffic is secure and encrypted, and we should update our Bouncy Castle version to the latest 1.69 to solve this timing issue. Otherwise, we should block the ports used by TLS (465, 993, 995).
  + CVE-2015-6644
    - This vulnerability affects Bouncy Castle in Android, which we are not using. Should we decide to code an Android client application, we should be aware of this vulnerability and assure our users are using Android 6.0 or later.
  + 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
    - These vulnerabilities have to do with key generation/exchange/validation and signature validation in DSA and AES encryption protocols. We can sidestep these vulnerabilities by using the RSA protocol or updating to the latest version of BouncyCastle.
  + CVE-2017-13098
    - Here again we have a vulnerability with TLS. We should block the applicable ports (465, 993, 995) and disable TLS RSA if we cannot update to the latest secure version of Bouncy Castle.
  + CVE-2018-1000613
    - This vulnerability allows malicious users to execute unexpected code via references to unexpected classes. If we, again, cannot update Bouncy Castle to the latest secure version, we should ensure that, if we do use XMSS serialization, that we validate every class referenced by the user’s input.
  + CVE-2018-5382
    - Version 1 BKS files use a very small 16 bit HMAC. The recommended mitigation is to not use version 1 BKS keystore files – we should instead be using a more robust format like BCFKS. Otherwise, if we updated BouncyCastle past version 1.47, this issue has been patched with a 160 bit HMAC.
  + CVE-2020-26939
    - This vulnerability allows users to potentially extract information from an SMTPS connection. This type of connection is used to validate electronic mail, which we are not using. This vulnerability can safely be regarded as a false positive.
* Log4j-api-2.12.1.jar
  + CVE-2020-9488
    - This vulnerability allows users to potentially extract information from an SMTPS connection. Since we are not using electronic mail communication, we can regard this vulnerability as a false positive.
* Snakeyaml-1.25.jar
  + CVE-2017-18640
    - SnakeYAML is a library that helps in reading YAML configuration files. I do not believe we are accepting YAML files from an unknown source, only local trusted sources. We could consider this a false positive, or update SnakeYAML to version 1.26 which has patched this issue.
* Jackson-databind-2.10.2.jar
  + CVE-2020-25649
    - This vulnerability allows attackers to expand entities in XML requests to unsafe levels, potentially compromising data integrity. We do not accept any XML requests in our code base, so we could regard this as a false positive. We do not know if/how Spring Framework uses Jackson Databind, so we should take care to update both to the latest secure version.
* Hibernate-validator-6.0.18.Final.jar
  + CVE-2020-10693
    - This vulnerability allows malicious users to bypass input sanitization, potentially causing unexpected results. We should verify that the input gives a valid output – like that a function requesting one data point outputs only one data point. This vulnerability has been fixed as of version 6.0.20.Final.
* Tomcat-embed-core-9.0.30.jar
  + CVE-2019-17569; CVE-2020-1935; CVE-2021-33037
    - These are a series of vulnerabilities allowing an attacker to manipulate HTTP request headers to expose information. To mitigate, we should not use Tomcat behind a reverse proxy. If we must, we can update Tomcat to version 9.0.48 where these three issues have been patched.
  + CVE-2020-8022
    - This vulnerability only applies to SUSE Linux Enterprise Server, which we are not using. This can safely be considered a false positive.
  + CVE-2021-30640
    - Our software does not use the JNDI realm, nor an LDAP directory. If we decide to use this functionality later, we should update Tomcat to version 9.0.46 or later.
  + CVE-2021-25329; CVE-2020-9484
    - This vulnerability allows attackers to execute code on a server to which they already have access. The current mitigation is to configure the PersistenceManager with an appropriate value for sessionAttributeValueClassNameFilter. The more recommended fix is to update to Tomcat version 9.0.35 or later.
  + CVE-2020-11996; CVE-2020-13934; CVE-2020-13935
    - These vulnerabilities allow for a denial of service attack if multiple carefully crafter inputs are sent to the server. We must ensure that we’re monitoring resource usage by a particular connection, as well as monitoring the frequency of inputs to our server. We could check payload length first to mitigate this issue, but the issue has been patched as of Tomcat version 9.0.37 or later.
  + CVE-2020-13943
    - This vulnerability stems from exceeding the maximum number of concurrent streams in an HTTP/2 connection. We haven’t written any code for maintaining the number of connections, but we should enforce the maximum limit. If we are unable to do so, this issue has been patched in Tomcat version 9.0.38 or later.
  + CVE-2020-17527
    - This vulnerability arises from a bug that allows Tomcat to reuse an HTTP request header value from a previous stream which, while would mostly lead to an error and closure, could sometimes leak related information. We should prioritize updating to Tomcat 9.0.40 or later.
  + CVE-2020-1938
    - This vulnerability concerns the Tomcat AJP protocol and gives AJP connections a higher trust level than HTTP connections. We should, as is recommended, disable the AJP protocol to block all AJP connections.
  + CVE-2021-24122
    - This bug in Windows API made it possible to bypass security constraints and view Java Server Pages. Updating Tomcat to version 9.0.40 or later will mitigate the issue.
  + CVE-2021-25122
    - This bug allowed new h2c connections to expose a previous request’s information. By ensuring our proxy server has HTTP Upgrade Protocol implemented in the server.xml file, we can safely assure this will be a false positive.

In summation, quite a few of these vulnerabilities can be disregarded as false positives. The others can be fixed simply with a few adjustments to the code base or updating the open-source libraries we use. As part of our regularly delivery pipeline, we should be reviewing a current dependency report and responding to vulnerabilities in our code base.

# References

*CVE-2021-33037 Apache Tomcat HTTP request smuggling*. (2021, July 12). Retrieved from Apache: https://lists.apache.org/thread.html/r612a79269b0d5e5780c62dfd34286a8037232fec0bc6f1a7e60c9381%40%3Cannounce.tomcat.apache.org%3E

InnReg. (n.d.). *2021 Fintech Complicance Regulatory Actions and Trends*. Retrieved from INNREG: https://www.innreg.com/blog/2021-fintech-compliance-regulatory-actions-and-trends

Rosenthal, M. (2021, September 16). *Must-Know Phishing Statistics: Updated 2021*. Retrieved from Tessian: https://www.tessian.com/blog/phishing-statistics-2020/

Tyler Technologies. (2020, November 12). *Security Incident*. Retrieved from tylerTech: https://www.tylertech.com/security-incident

United States Government Accountability Office. (2005, February). *Core Financial System Requirements - Checklist for Reviewing Systems under the Federal Financial Mangement Improvement Act*. Retrieved from United States Govenment Accountability Office: https://www.gao.gov/assets/gao-05-225g.pdf

VMWare. (2020, September 17). *CVE-2020-5421: RFD Protection Bypass via jsessionid*. Retrieved from Tanzu: https://tanzu.vmware.com/security/cve-2020-5421