****

# 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 4](#_Toc32574613)

[4. Static Testing 4](#_Toc32574614)

[5. Mitigation Plan 4](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **3/18/2021** | **Michael Bauer** | **Added client’s needs, security needs, manual review, static testing, and remediation for vulnerabilities** |

## 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

Michael Bauer

## 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?

Artemis Financial will need secure communications as a financial consultant company. They will need to follow the Gramm-Leach-Bliley Act regarding securing information, disclosure of private information, and accessing of private information. Encryption will have to be used to secure information, transactions, and communications. If there are international transactions then specific privacy laws will need to be considered for each country being allowed to have transactions. Using a RESTful API possible threats include injection, cross site request forgery, cross site scripting, and XML external entity to include others. Securing web applications now and as they change will require securing the vulnerabilities in code used within the application, such as open source libraries.

## 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.

APIs and securing the interactions is important since the web application is using a RESTful API and will be a main component used to interact with the system. Client/server security is important since a RESTful API separates the actual interface and data storage as a client and server. Encryption is important since much of the information, transactions, and communications needs to be secured and encryption should be used for data that is stored and transferred from users.

## 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 no input validation to ensure what is being passed in is acceptable for example in the CRUD class. There is no use of ciphers for encryption for customer data in the customer class. There should be more then one check to verify authentication and authorization and should be done with each request for data.

## 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

**log4j-api-2.12.1.jar**

* + **Low Severity**
  + **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.
  + **Solution -** Update Apache Log4j to 2.13.2
  + **Mitigation** -Previous versions can set the system property mail.smtp.ssl.checkserveridentity to true to globally enable hostname verification for SMTPS connections.

**snakeyaml-1.25.jar**

* + **High** **Severity**
  + **CVE-2017-18640** - The Alias feature in SnakeYAML 1.18 allows entity expansion during a load operation, a related issue to CVE-2003-1564.
  + **Solution –** Update SnakeYAML to version 1.26

**jackson-databind-2.10.2.jar**

* + **High Severity**
  + **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.
  + **Solution –** Update FasterXML Jackson Databind to 2.10.5.1

**hibernate-validator-6.0.18.Final.jar**

* + **Medium Severity**
  + **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.
  + **Solution –** Update to Hibernate Validator 6.1.5.Final
  + **Mitigation -** You can pass user input as an expression variable by unwrapping the context to HibernateConstraintValidatorContext

**spring-core-5.2.3.RELEASE.jar**

* + **Medium Severity**
  + **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.
  + **Solution –** Update Spring Framework to 5.2.9, 5.1.18, 5.0.19, or 4.3.29

**tomcat-embed-core-9.0.30.jar**

* + **Medium Severity**
  + **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.
  + **Solution –** Update Apache Tomcat to 8.5.54 or later, 9.0.31 or later, or a version after 7.0.99 or later
  + **High Severity**
  + **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.
  + **Solution -** Update Apache Tomcat to 10.0.0-M6 or later, 9.0.36 or later, or 8.5.56 or later
  + **High Severity**
  + **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.
  + **Solution –** Update Apache Tomcat to 10.0.0-M7 or later, 9.0.37 or later, or 8.5.57 or later
  + **High Severity**
  + **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.
  + **Solution -** - Update Apache Tomcat to 10.0.0-M7 or later, 9.0.37 or later, or 8.5.57 or later
  + **High Severity**
  + **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.
  + **Solution –** Install openSUSE-SU-2020:0911-1 security update
  + **Medium Severity**
  + **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.
  + **Solution -** Update Apache Tomcat to 10.0.0-M8 or later, 9.0.38 or later, or 8.5.58 or later
  + **High Severity**
  + **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.
  + **Solution -** Update Apache Tomcat to 10.0.0-M10 or later, 9.0.40 or later, or 8.5.60 or later
  + **Critical Severity**
  + **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.
  + **Solution –** Update Apache Tomcat to 9.0.31 or later, 8.5.51 or later, or 7.0.100 or later
  + **Mitigation -**
    1. If AJP support is not required, the Connector may be disabled e.g. by removing the AJP Connector element from the server.xml file
    2. If AJP support is required, untrusted users may be prevented from accessing the AJP port by one or more of the following means:
       - - configuring appropriate network firewall rules
       - - configuring an explicit address attribute to the connector so that the Connector listens on a non-public interface
       - - configuring a shared secret for the AJP connection
  + **Medium Severity**
  + **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.
  + **Solution -** Update Apache Tomcat to 9.0.31 or later, 8.5.51 or later, or 7.0.100 or later
  + **High Severity**
  + **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.
  + **Solution -** Update Apache Tomcat to 10.0.0-M5 or later, 9.0.35 or later, 8.5.55 or later, or 7.0.104 or later
  + **Mitigation -** Alternatively, users may configure the PersistenceManager with an appropriate value for sessionAttributeValueClassNameFilter to ensure that only application provided attributes are serialized and deserialized.
  + **Medium Severity**
  + **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.
  + **Solution -** Update Apache Tomcat to 10.0.0-M10 or later, 9.0.40 or later, 8.5.60 or later, or 7.0.107 or later
  + **High Severity**
  + **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.
  + **Solution -** Update Apache Tomcat to 10.0.2 or later, 9.0.43 or later, or 8.5.63 or later
  + **High Severity**
  + **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.
  + **Solution -** Update Apache Tomcat to 10.0.2 or later, 9.0.43 or later, 8.5.63 or later, or 7.0.108 or later

**bcprov-jdk15on-1.46.jar**

* + **Medium Severity**
  + **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, a related issue to CVE-2013-0169.
  + **Solution –** This solution was found in <http://www.isg.rhul.ac.uk/tls/TLStiming.pdf>

(AlFardan, N. J., & Paterson, K. G. (2013, February 27). *Lucky Thirteen: Breaking the TLS and DTLS Record Protocols*. http://www.isg.rhul.ac.uk/tls/TLStiming.pdf.)

* + 1. First sanity check the ciphertext: check that its length in bytes is a multiple of the block-size b and is at least max{b, t + 1} (for chained IVs) or b + max{b, t + 1} (for explicit IVs). If these conditions are not met, then return fatal error.
    2. Decrypt the ciphertext to obtain plaintext P; now plen will be a multiple of b and at least max{b, t + 1}.
    3. If t + padlen + 1 > plen, then the plaintext is not long enough to contain the padding (as indicated by the last byte of plaintext) plus a MAC tag. In this case, run a loop as if there were 256 bytes of padding, with a dummy check in each iteration. Then let P 0 denote the first plen − t bytes of P, compute a MAC on SQN||HDR||P 0 and do a constant-time comparison of the computed MAC with the last t bytes of P. Return fatal error.
    4. Otherwise (when t + padlen + 1 ≤ plen), check the last padlen + 1 bytes of P to ensure they are all equal (to the last byte of P), ensuring that the loop does check all the bytes (and does not stop as soon as the first mismatch is detected). If this fails, then run a loop as if there were 256 − padlen − 1 bytes of padding, with a dummy check in each iteration, and then do a MAC check as in the previous step. Return fatal error.
    5. Otherwise (the padding is now correctly formatted) run a loop as if there were 256 − padlen − 1 bytes of padding, doing a dummy check in each iteration. Then let P 0 denote the first plen − padlen − 1 − t bytes of P, and let T denote the next t bytes of P (the remainder of P is valid padding). Run the MAC computation on SQN||HDR||P 0 to obtain a MAC tag T 0 . Then set L1 = 13 + plen − t, L2 = 13 + plen − padlen − 1 − t, and perform an additional d L1−55 64 e−d L2−55 64 e MAC compression function evaluations (on dummy data). Finally, do a constant-time comparison of T and T 0 . If these are equal, then return P 0 . Otherwise, return fatal error
  + **Unknown Severity**
  + **CVE-2015-6644 (OSSINDEX) –** An information disclosure vulnerability in Bouncy Castle could enable a local malicious application to gain access to user’s private information.
  + **Solution –** Update Bouncy Castle to latest version.
  + **Mitigation –** Add code to patch vulnerability in current version.
  + **Medium Severity**
  + **CVE-2015-7940 (OSSINDEX) –** 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."
  + **Solution –** Update Bouncy Castle to 1.51 or later.
  + **High Severity**
  + **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.
  + **Solution –** Update Bouncy Castle to 1.56 or later.
  + **Medium Severity**
  + **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.
  + **Solution –** Update Bouncy Castle to 1.56 or later.
  + **Medium Severity**
  + **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.
  + **Solution –** Update Bouncy Castle to 1.56 or later.
  + **High Severity**
  + **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.
  + **Solution –** Update Bouncy Castle to 1.56 or later.
  + **High Severity**
  + **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 initialized 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.
  + **Solution –** Update Bouncy Castle to 1.56 or later.
  + **High Severity**
  + **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.
  + **Solution –** Update Bouncy Castle to 1.56 or later.
  + **Medium Severity**
  + **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.
  + **Solution –** Update Bouncy Castle to 1.56 or later.
  + **Low Severity**
  + **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.
  + **Solution –** Update Bouncy Castle to 1.56 or later.
  + **High Severity**
  + **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.
  + **Solution –** Update Bouncy Castle to 1.56 or later.
  + **Medium Severity**
  + **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."
  + **Solution –** Update Bouncy Castle to 1.59 or later
  + **Critical Severity**
  + **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. 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.
  + **Solution –** Update Bouncy Castle to 1.60 or later
  + **Critical Severity**
  + **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. All BKS-V1 keystores are vulnerable. Bouncy Castle release 1.47 introduces BKS version 2, which uses a 160-bit MAC.
  + **Solution –** Update Bouncy Castle to 1.47 or later
  + **Medium Severity**
  + **CVE-2020-26939 (OSSINDEX) –** 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.
  + **Solution –** Update Bouncy Castle to 1.62 or later

## 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.

**These are the steps that need to be taken**

1. Update Apache Log4j to 2.13.2 or later
2. Update SnakeYAML to version 1.26 or later
3. Update FasterXML Jackson Databind to 2.10.5.1 or later
4. Update to Hibernate Validator 6.1.5.Final or later
5. Update Spring Framework to 5.2.9, 5.1.18, 5.0.19, or 4.3.29 or later
6. Install openSUSE-SU-2020:0911-1 security update
7. Update Apache Tomcat to 10.0.2 or later, 9.0.43 or later, or 8.5.63 or later
8. Within Bouncy Castle padding length needs to be treated as having length of 0 instead of 1
9. Update Bouncy Castle to 1.62 or later
10. This is what needs to be done to Bouncy Castle to fix the CVE-2013-1624
    1. First sanity check the ciphertext: check that its length in bytes is a multiple of the block-size b and is at least max{b, t + 1} (for chained IVs) or b + max{b, t + 1} (for explicit IVs). If these conditions are not met, then return fatal error.
    2. Decrypt the ciphertext to obtain plaintext P; now plen will be a multiple of b and at least max{b, t + 1}.
    3. If t + padlen + 1 > plen, then the plaintext is not long enough to contain the padding (as indicated by the last byte of plaintext) plus a MAC tag. In this case, run a loop as if there were 256 bytes of padding, with a dummy check in each iteration. Then let P 0 denote the first plen − t bytes of P, compute a MAC on SQN||HDR||P 0 and do a constant-time comparison of the computed MAC with the last t bytes of P. Return fatal error.
    4. Otherwise (when t + padlen + 1 ≤ plen), check the last padlen + 1 bytes of P to ensure they are all equal (to the last byte of P), ensuring that the loop does check all the bytes (and does not stop as soon as the first mismatch is detected). If this fails, then run a loop as if there were 256 − padlen − 1 bytes of padding, with a dummy check in each iteration, and then do a MAC check as in the previous step. Return fatal error.
    5. Otherwise (the padding is now correctly formatted) run a loop as if there were 256 − padlen − 1 bytes of padding, doing a dummy check in each iteration. Then let P 0 denote the first plen − padlen − 1 − t bytes of P, and let T denote the next t bytes of P (the remainder of P is valid padding). Run the MAC computation on SQN||HDR||P 0 to obtain a MAC tag T 0 . Then set L1 = 13 + plen − t, L2 = 13 + plen − padlen − 1 − t, and perform an additional d L1−55 64 e−d L2−55 64 e MAC compression function evaluations (on dummy data). Finally, do a constant-time comparison of T and T 0 . If these are equal, then return P 0 . Otherwise, return fatal error

The static test should be rerun to ensure these vulnerabilities were remedied.