# Project One

Katherine A. Cranford

Department of Computer Science, Southern New Hampshire University

CS 305: Software Security

Dr. Vivian Lyon

March 18, 2022

****

# CS 305 Project One

**Artemis Financial Vulnerability Assessment Report**

Table of Contents

[Document Revision History 4](#_Toc32574607)

[Client 4](#_Toc32574608)

[Instructions 4](#_Toc32574609)

[Developer 5](#_Toc32574610)

[1. Interpreting Client Needs 5](#_Toc32574611)

[2. Areas of Security 5](#_Toc32574612)

[3. Manual Review 6](#_Toc32574613)

[4. Static Testing 6](#_Toc32574614)

[5. Mitigation Plan 7](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **18 March 2022** | **Katherine Cranford** | **Updated all sections** |

## 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

Katherine Cranford

## 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?

Attackers are frequently financially motivated, making financial applications popular targets. These attackers will attempt to steal client information, such as social security numbers and banking information, as well as company information, such as employee login details and transaction records. Thus, secure communications are of utmost importance, not only for the company’s security, and that of their clients, but also to remain compliant with governmental regulations that have been put into place to regulate businesses that hold personal information.

Alabama requires that businesses “implement and maintain reasonable security measures…to protect sensitive personally identifying information against a breach of security”, and at least 24 additional states have similar requirements for these businesses (Greenberg 2019). Additionally, should the company choose to offer its clients the ability to invest in foreign stock, this would introduce additional cybersecurity requirements like addressing foreign government policies.

There are a wide range of potential threats the company is likely to face. Phishing, malware and ransomware, denial of service and data theft are all common attacks on financial institutions. Being aware of these threats and taking intentional steps towards preventing them is the only way to remain safe.

## 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 code of this application will have many areas of input in the RESTful API sections. Parameterized queries and variables defend against potential SQL injection (*OWASP Secure Coding Practices Quick Reference Guide*, 2010, p. 8). The number of returned results should be verified that it is within expectations and output should be forcibly converted to a formal output type (Manico & Detlefsen, 2014). Additionally, the strict use of authentication and authorization should be in place for account access and management, such as a DAO layer (Manico & Detlefsen, 2014).
* APIs: There are RESTful API sections in this application. Manico and Detlefsen (2014) suggest separate, server-side Policy Information Points (PIPs), Policy Enforcement Points (PEPs) and Policy Decision Points (PDPs). Ideally, the two advise, a permission-based PEP and attribute-based PDP should be used to defend against privilege escalation and denial of service as well as for flexibility for future policy changes. The application should be restricted from interacting directly with the operating system (*OWASP Secure Coding Practices Quick Reference Guide*, 2010, p. 8).
* Cryptography: To ensure secure data transfer, encryption should be deployed for transmission and storage of any sensitive information, even server-side (*OWASP Secure Coding Practices Quick Reference Guide*, 2010, p. 7). One thing needed to ensure this is the implementation of HTTPS protocol, allowing for confidentiality, integrity, and authentication (Manico & Detlefsen, 2014).
* Client/Server: All data validation and session ID creation should be performed on the server, with the necessary access controls (*OWASP Secure Coding Practices Quick Reference Guide*, 2010, pp. 4, 6). Manico and Detlefsen (2014) further expand on these requirements, noting that identity information, metadata, policy rules and other PIPs should come from trusted server-side resources.
* Code Quality: Private variables and classes should be used where necessary and object oriented principles should be executed when possible. The principle of least privilege (POLP) should be implemented, restricting users to only the functionality, system information and data that is required (*OWASP Secure Coding Practices Quick Reference Guide*, 2010, p. 7).
* Code Error: Exception handling should be in place in every instance necessary, such as user authentication, and should be handled by the application rather than relying on server configuration (*OWASP Secure Coding Practices Quick Reference Guide*, 2010, p. 7).

## 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 class “customer”, int account\_balance should be private. Account\_balance is not accessed in any other class and it is incredibly important that any changes to the variable is reflected in all instances. This applies to Code Quality.
* There are no instances of input validation in the application. This leaves the application open to a myriad of attacks, including SQL Injection and buffer overflow attacks, leading to a denial of service.
* The line “(@RequestParam (value = “name”, defaultValue = “World”) String name)” in the GreetingController class cannot be determined until runtime and is therefore susceptible to SQL injection. This applies to Input Validation and Code Quality.
* The CRUD class is incomplete as it does not allow deletion. It also does not include parameterization of input or require a form of authorization to access its functions. There should be exception handling present for each of the functions for unauthorized users. This applies to Input Validation, Code Quality and Code Error.

## 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

* **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.
* **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. 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.
* **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.
* **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.
* **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.
* **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. 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.
* **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.
* **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. An attacker can recover the private key from a vulnerable application. This vulnerability is referred to as "ROBOT."
* **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 the execution of unexpected code. This attack appears 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.
* **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.
* **CVE-2020-15522:** 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.
* **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.
* **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.
* **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.
* **CVE-2020-36518:** jackson-databind before 2.13.0 allows a Java StackOverflow exception and denial of service via a large depth of nested objects.
* **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.
* **CVE-2021-42550:** 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.
* **CVE-2017-18640:** The Alias feature in SnakeYAML 1.18 allows entity expansion during a load operation, a related issue to CVE-2003-1564.
* **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 with a jsessionid path parameter.
* **CVE-2021-22060:** In Spring Framework versions 5.3.0 - 5.3.13, 5.2.0 - 5.2.18, and older unsupported versions, it is possible for a user to provide malicious input to cause the insertion of additional log entries. This is a follow-up to CVE-2021-22096 that protects against additional types of input and in more places of the Spring Framework codebase.
* **CVE-2021-22096:** 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.
* **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.
* **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.
* **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 enough of such requests were made on concurrent HTTP/2 connections, the server could become unresponsive.
* **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 enough 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. Multiple requests with invalid payload lengths could lead to a denial of service.
* **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.
* **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.
* **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.
* **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; and processing any file in the web application as a JSP Further. If the web application allowed file upload and stored those files within the web application (or the attacker was able to control the content of the web application by some other means) then this, along with the ability to process a file as a JSP, made remote code execution possible. It is important to note that mitigation is only required if an AJP port is accessible to untrusted users. Users wishing to take a defense-in-depth approach and block the vector that permits returning arbitrary files and execution as JSP may upgrade to Apache Tomcat 9.0.31, 8.5.51 or 7.0.100 or later. Several 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.
* **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; b) the server is configured to use the PersistenceManager with a FileStore; 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.
* **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 behavior of the JRE API File.getCanonicalPath() which in turn was caused by the inconsistent behavior of the Windows API (FindFirstFileW) in some circumstances.
* **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.
* **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.
* **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.
* **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 honored the identify encoding; and Tomcat did not ensure that, if present, the chunked encoding was the final encoding.
* **CVE-2021-41079:** 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.
* **CVE-2021-42340:** The fix for bug 63362 present in Apache Tomcat 10.1.0-M1 to 10.1.0-M5, 10.0.0-M1 to 10.0.11, 9.0.40 to 9.0.53 and 8.5.60 to 8.5.71 introduced a memory leak. The object introduced to collect metrics for HTTP upgrade connections was not released for WebSocket connections once the connection was closed. This created a memory leak that, over time, could lead to a denial of service via an OutOfMemoryError.
* **CVE-2020-8022:** An Incorrect Default Permissions vulnerability in the packaging of tomcat allows local attackers to escalate from group tomcat to root.

## 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.

* Input Validation: Validate input against allowed characters, and ensure that strings are a reasonable length before passing them to functions (*OWASP Secure Coding Practices Quick Reference Guide*, 2010, pp. 4, 9).
* Client/Server: Limit checks should be in place to prevent overflow to avoid denial of service attacks.
* Code Error: Exception handling should be in place to prevent unexpected behavior and to defend against exploitation of untrusted data.
* Code Quality: Ensure that the buffer is as large as specified to avoid writing past allocated space and close resources individually rather than relying on garbage collection (*OWASP Secure Coding Practices Quick Reference Guide*, 2010, p. 9).
* Dependency Check for Files: NVD and CVE databases should be referenced for dependencies.

**References**

Manico, J., & Detlefsen, A. (2014). *Iron-Clad Java: Building Secure Web Applications (Oracle Press)* (1st ed.). McGraw Hill.

Greenberg, P. (2019, May 29). *Data Security Laws | Private Sector*. National Conference of State Legislatures. Retrieved March 18, 2022, from https://www.ncsl.org/research/telecommunications-and-information-technology/data-security-laws.aspx

*NVD - Search and Statistics*. (n.d.). National Vulnerability Database. Retrieved March 13, 2022, from https://nvd.nist.gov/vuln/search

*OWASP Secure Coding Practices Quick Reference Guide*. (2010, August). OWASP. Retrieved March 13, 2022, from https://owasp.org/www-pdf-archive/OWASP\_SCP\_Quick\_Reference\_Guide\_v1.pdf.