****

# 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** | **7/18/2021** | **Michael Jordan** | **Initial release. Includes client needs, areas of security, manual review, static testing, and mitigation plan.** |
| **1.1** | **7/18/2021** | **Michael Jordan** | **Updated the mitigation plan to account for the manual code review.** |

## 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 Jordan

## 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?

1. There appear to be two major items that contribute to the value of secure communications for this company. The first item is maintaining the trust of their patrons by ensuring their private information is not compromised. The second item is maintaining the accuracy of their individualized financial plans by ensuring they have not been tampered with in transit.
2. With the information given, there is no indication that the company produces international transactions.
3. According to [this site](https://www.tcdi.com/information-security-compliance-which-regulations/), it appears that the company is subject to the Gramm-Leach-Bliley Act, and may be subject to the American Institute of Certified Public Accountants SOC2, the California Consumer Privacy Act and/or the Payment Card Industry Data Security Standard. Some of the requirements from the [Investment Advisers Act of 1940](https://www.sec.gov/divisions/investment/iaregulation/memoia.htm) may also be relevant for secure communications, such as the suitability and custody requirements.
4. One type of threat could be “phishing” attacks, where an outside threat imitates company communication to patrons. The threat could gain the patron’s credentials, then pose as the patron within the company’s system to obtain personal information or affect their finances. Or the threat could install malware on the patron’s system, which is potentially even more damaging. The contact details of patrons present within the client’s application could lead to an attack of this sort if a vulnerability within the application allowed those details to be leaked. Another type of threat would directly target the application or the data behind it. For example, a denial-of-service attack may stop patrons from interacting with the application for several hours or days. Another type of attack called “[ransomware](https://www.reuters.com/technology/hackers-demand-70-million-liberate-data-held-by-companies-hit-mass-cyberattack-2021-07-05/)” could encrypt the data behind the application, which would also stop patrons from interacting with the application. The financial nature of the business makes this type of attack particularly enticing.
5. Although using open-source libraries can save a significant amount of time when developing applications, especially when specialized functions are needed, they can introduce deadly vulnerabilities to the application. Using such libraries may be unavoidable, but examining the known vulnerabilities allows developers to take action to mitigate the risks while reaping the benefits. Due to the nature of the libraries, they are constantly evolving and updating, which will remove some known vulnerabilities and introduce unknown ones, which may not be discovered for several months. Because of this, a secure web application may become vulnerable over its lifetime if it is not properly maintained.

## 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.

1. As a web application that is likely to accept input from users, Input Validation is an applicable area to assess. Much of the application is likely to be available only to authenticated users, but ensuring their input is validated would be another layer of a defense in depth strategy.
2. Examining the source code reveals that the application uses the Spring framework and intends to use the Bouncy Castle library. The API interactions should be examined to ensure they are done securely.
3. Since secure communications are vital for the client’s application, Cryptography is an applicable area to assess. Effectively encrypting the communications is one of the best ways to make sure they stay secure.
4. As a web application, Client/Server is an applicable area to assess. This also ties in with the necessity for secure communications, as they will be passing between the client and the server.
5. Since the application will be handling private and/or financial information, Code Error is an applicable area to assess. Properly aborting transactions when an application error occurs and avoiding the inclusion of private information will be important.
6. For similar reasons, Code Quality is also an applicable area to assess. The application should ensure that private and financial information is secure during standard operation as well.
7. Controlling access to this data is important for tracking who made changes to it or even viewed it, so Encapsulation is also an applicable area to assess.

## 3. Manual Review

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

1. Examination of the code found in the “CRUD.java” class has revealed no vulnerabilities.
2. Examination of the code found in the “CRUDController.java” class has revealed 1 vulnerability. The usage of “doc.toString()” populates the CRUD object with the [hexadecimal representation of the hash code](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#toString--) of the document object, instead of the expected document contents. Additionally, the code appears to be incomplete as the “CRUD” function does not use the “name” parameter that it is given. This section must be re-examined as changes are made to complete the codebase.
3. Examination of the code found in the “customer.java” class has revealed 2 vulnerabilities. First, the variable “account\_balance” is not correctly encapsulated, which allows it to be manipulated by sources that should not have access. Second, the variable “account\_balance” is of type int and is likely used to track money. Due to the maximum size of the int data type ( or 2,147,483,647), accounts with a large amount of money in them could cause an overflow and severely misrepresent the amount of money a patron has in their account. Additionally, the code is not yet complete, and this section must be re-examined as changes are made to complete the codebase.
4. Examination of the code found in the “DocData.java” class has revealed 1 vulnerability. Although the database connection is clearly a testing mechanism, the database credentials are passed as string literals. Using string literals makes it difficult to change credentials in the future and may expose the database to severe risks if the credentials are pulled from the application. Additionally, the code is not yet complete, and this section must be re-examined as changes are made to complete the codebase.
5. Examination of the code found in the “Greeting.java” class has revealed no vulnerabilities.
6. Examination of the code found in the “GreetingController.java” class has revealed no vulnerabilities.
7. Examination of the code found in the “myDateTime.java” class has revealed 2 vulnerabilities. First, the variables “mySecond”, “myMinute” and “myHour” are not correctly encapsulated, which allow them to be manipulated by sources that should not have access. Second, the function “retrieveDateTime()” does not produce the expected result of an array populated with the current time variables. Additionally, the code is not yet complete, and this section must be re-examined as changes are made to complete the codebase.
8. Examination of the code found in the “RestServiceApplication.java” class has revealed no vulnerabilities.

## 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
4. CVE-2013-1624 – A vulnerability in the Bouncy Castle Java library that allows attackers to conduct distinguishing attacks and plaintext-recovery attacks. Several implementation solutions are provided by [this paper](http://www.isg.rhul.ac.uk/tls/TLStiming.pdf), but updating the library to 1.48 or later is also a solution.
5. CVE-2015-6644 – An information disclosure vulnerability in Bouncy Castle could enable a local malicious application to gain access to user’s private information. No solutions are provided by the report.
6. CVE-2015-7940 – The Bouncy Castle Java library does not validate a point is within the elliptic curve, making it easier for attackers to obtain private keys. The report recommends updating the library to 1.51 or later.
7. CVE-2016-1000338 – The Bouncy Castle Java library does not fully validate the ASN.1 encoding of a DSA signature, allowing for the introduction of ‘invisible’ data into a signed structure. The report recommends updating the library to 1.56 or later.
8. CVE-2016-1000339 – The Advanced Encryption Standard (AES) engine used by the Bouncy Castle Java library had a potential AES key leak. The report recommends updating the library to 1.56 or later.
9. CVE-2016-1000341 – The Bouncy Castle Java library has a vulnerability that allows a timing attack that reveals information about a DSA signature’s k value and private value. The report recommends updating the library to 1.56 or later.
10. CVE-2016-1000342 – The Bouncy Castle Java library does not fully validate the ASN.1 encoding of a ECDSA signature, allowing for the introduction of ‘invisible’ data into a signed structure. The report recommends updating the library to 1.56 or later.
11. CVE-2016-1000343 – The Bouncy Castle Java library DSA key pair generator generates a weak private key if used with default values. The report recommends explicitly passing parameters to the key pair generator or updating the library to 1.56 or later.
12. CVE-2016-1000344 – The DHIES implementation in the Bouncy Castle Java library allowed the use of ECB mode which is regarded as unsafe. The report recommends updating the library to 1.56 or later.
13. CVE-2016-1000345 – The DHIES/ECIES CBC mode is vulnerable to a padding attack in the Bouncy Castle Java library. The report recommends updating to 1.56 or later.
14. CVE-2016-1000346 – The Bouncy Castle Java library does not fully validate the public DH key of the other party, which can reveal details about the private key of the other party. The report recommends updating the library to 1.56 or later.
15. CVE-2016-1000352 - The ECIES implementation in the Bouncy Castle Java library allowed the use of ECB mode which is regarded as unsafe. The report recommends updating the library to 1.56 or later.
16. CVE-2017-13098 [(A.K.A. “ROBOT”)](https://robotattack.org/) – The TLS used by the Bouncy Castle Java library contains a vulnerability that allows an attacker to recover the application’s private RSA key. The report recommends updating the library to 1.59 or later.
17. CVE-2018-1000613 – The Bouncy Castle Java library has a vulnerability related to private key deserialization that can result in the execution of unexpected code. The report recommends updating the library to 1.60 or later.
18. CVE-2018-5382 – The default BKS keystore in the Bouncy Castle Java library uses an HMAC that is only 16 bits long, which can allow an attacker to compromise the integrity of a BKS keystore. The report recommends updating the library to 1.47 and creating a new BKS keystore.
19. CVE-2020-26939 – The Bouncy Castle Java library contains faulty error handling related to OAEP that could leak information about the private exponent of the RSA private key. The report recommends updating the library to 1.61 or later.
20. CVE-2020-10693 – The Hibernate Validator library contains a flaw that allows attackers to bypass input validation. The report recommends updating the library to 6.1.5.Final or later.
21. CVE-2020-25649 – The FasterXML Jackson Databind library contains an entity expansion flaw that allowed XML external entity attacks. The report recommends updating the library to 2.10.5.1 or later.
22. CVE-2020-9488 – The Apache Log4j library contained a vulnerability that could allow for log messages to be intercepted. The report recommends updating the library to 2.13.2 or later.
23. CVE-2017-18640 – The SnakeYAML library allows entity expansion during a load operation. The report recommends updating the library to 1.26 or later.
24. CVE-2020-5421 – The Spring Framework library contains a vulnerability that potentially allowed Reflected File Download (RFD) protections to be bypassed. The report recommends updating the library to version 5.2.9 or later.
25. CVE-2021-22118 – The Spring Framework library contains a privilege escalation vulnerability for WebFlux applications that allows a locally authenticated user to read or modify uploaded files or overwrite arbitrary files. The report recommends updating the library to 5.2.15 or later.
26. CVE-2019-17569 – The Apache Tomcat library contains a flaw related to refactoring that could lead to HTTP request smuggling if a reverse proxy incorrectly handled invalid Transfer-Encoding headers. The report recommends updating the library to 9.0.31 or later.
27. CVE-2020-11996 – The Apache Tomcat library contains a flaw related to HTTP/2 requests that could lead to unresponsive servers due to high CPU usage. The report recommends updating the library to 9.0.36 or later.
28. CVE-2020-13934 – The Apache Tomcat library contains a flaw related to h2c direct connections that could lead to a denial-of-service attack. The report recommends updating the library to 9.0.37 or later.
29. CVE-2020-13935 – The Apache Tomcat library contains a flaw related to invalid payload lengths in WebSocket frames that could lead to a denial-of-service attack. The report recommends updating the library to 9.0.37 or later.
30. CVE-2020-13943 – The Apache Tomcat library contains a flaw related to a HTTP/2 client violating the protocol by exceeding the agreed maximum number of streams could result in users seeing responses for unexpected resources. The report recommends updating the library to 9.0.38 or later.
31. CVE-2020-17527 – The Apache Tomcat library contains a flaw related to reusing request header values received on an HTTP/2 that could allow information to leak between requests. The report recommends updating the library to 9.0.40 or later.
32. CVE-2020-1935 – The Apache Tomcat library contains a flaw related to header parsing that could lead to HTTP request smuggling if a reverse proxy incorrectly handled invalid Transfer-Encoding headers. The report recommends updating the library to 9.0.31 or later.
33. CVE-2020-1938 [(A.K.A. “Ghostcat”)](https://snyk.io/blog/ghostcat-breach-affects-all-tomcat-versions/) – The Apache Tomcat library contains a flaw related to the Apache JServ Protocol that could lead to obtaining arbitrary files or malicious code execution. The report recommends updating the library to 9.0.31 or later.
34. CVE-2020-8022 – The SUSE packaging of the Apache Tomcat library contains a flaw related to incorrect default permissions that could allow local attackers to escalate their permissions to root. The report recommends updating the SUSE product.
35. CVE-2020-9484 – The Apache Tomcat library contains a flaw related to the PersistenceManager FileStore objects that could allow remote code execution. The report recommends updating the library to 9.0.43 or later.
36. CVE-2021-24122 – The Apache Tomcat library contains a flaw related to unexpected behavior of a JRE API filesystem function that could lead to JSP source code disclosure. The report recommends updating the library to 9.0.40 or later.
37. CVE-2021-25122 – The Apache Tomcat library contains a flaw related to new h2c connection requests that could allow a second party to also see the results of a request. The report recommends updating the library to 9.0.42 or later.
38. CVE-2021-25329 – The Apache Tomcat library contains a flaw related to fixing CVE-2020-9484, allowing for the same vulnerability to be exploited. The report recommends updating the library to 9.0.42 or later.
39. CVE-2021-33037 – The Apache Tomcat library contains a flaw related to incorrectly parsing the HTTP transfer-encoding request that could lead to HTTP request smuggling. The report recommends updating the library to 9.0.47 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.

The codebase in its current state does not utilize the Bouncy Castle library, and therefore is not vulnerable to any of the CVE’s related to it. However, it is likely to utilize the library in the future as the codebase grows.

The remaining vulnerabilities are related to Spring Core and other Spring dependencies. Although the codebase is not affected by many of the vulnerabilities listed, it is likely that implementing the recommended solutions for the ones it is affected by will also satisfy the solutions for the ones it is not affected by. For efficiency, I will evaluate the Spring Core vulnerabilities first. The application does not use WebFlux, so implementing the solution for that is unnecessary. The current codebase does not appear to be vulnerable to the RFD attacks mentioned in CVE-2020-5421, but it may become vulnerable [if it uses callbacks in URLs, or if it has URLs that are accessible by authenticated users](https://www.davidsopas.com/reflected-file-download-cheat-sheet/), which is very likely. I would suggest implementing the solution for this vulnerability.

1. The first step to remedy the identified security vulnerabilities is to update the version of the Bouncy Castle library. Since there are no existing usages in the codebase, updating to the latest version would have minimal to no impact on the codebase and is what I recommend. As of writing this report, version 1.69 is the latest version available.
2. The second step is to update the version of Spring Boot, which will also update Spring Core. Since the codebase does have existing usages of the Spring Framework, updating to the earliest recommended version for solving the relevant vulnerability will have minimal impact. The earliest version of Spring Core recommended is 5.2.9, and the earliest version of Spring Boot which contains it is 2.2.10.
3. The third step is to [examine the dependencies of Spring Boot 2.2.10](https://docs.spring.io/spring-boot/docs/2.2.10.RELEASE/reference/html/appendix-dependency-versions.html#appendix-dependency-versions) and the recommended solutions of the remaining vulnerabilities.
   1. The version of Hibernate Validator is 6.0.20.Final, which satisfies the related vulnerability solution.
   2. The version of FasterXML Jackson Databind is 2.10.5, which does not satisfy the related vulnerability solution.
   3. The version of Apache Log4j is 2.12.1, which does not satisfy the related vulnerability solution.
   4. The version of SnakeYAML is 1.25, which does not satisfy the related vulnerability solution.
   5. The version of Apache Tomcat is 9.0.38, which satisfies approximately half of the related vulnerability solutions.
4. The fourth step is to examine the remaining vulnerabilities to determine which ones are relevant to the codebase. The codebase does not utilize Jackson Databind, Log4j, or SnakeYAML, so the solutions for those vulnerabilities do not need to be implemented. Similarly, the codebase does not utilize HTTP/2 or h2c connections, so the solutions to vulnerabilities #28 and #34 do not need to be implemented. The codebase is not a SUSE package, so vulnerability #31 is unrelated. The codebase does not meet the strict requirements including the use of a PersistenceManager and a FileStore to be vulnerable to #32 and #35. The codebase does not contain JSP source code that could be exposed by #33.
5. Vulnerability #36 appears to affect the codebase. The recommended solution is to update the Apache Tomcat library, which is a dependency of Spring Boot. However, it is possible to obtain a specific version of a Spring Boot dependency using Maven as implied at the top of [this page](https://docs.spring.io/spring-boot/docs/2.4.7/reference/html/appendix-dependency-versions.html). Although the recommended version is 9.0.47 or later, the earliest version available that meets this criterion is version 9.0.48. The fifth step is then to adjust the pom.xml file to specify that version for tomcat-embed-core and tomcat-embed-websocket as follows.Graphical user interface, text

   Description automatically generated
6. Next, I shall address the vulnerabilities found in the manual code review. In the “CRUDController.java” class, the method most likely to produce the intended functionality is “doc.read\_document()”, which is not yet completed. This should be revisited after the codebase is more complete. In the “customer.java” class, I recommend setting the “account\_balance” variable to private, then adjusting the implementation to utilize [JSR-354](https://www.baeldung.com/java-money-and-currency). In the “DocData.java” class, a couple approaches to avoid hardcoded credentials include using a secret manager or programmatic in-memory storage as [described here](https://www.linkedin.com/pulse/storing-database-credentials-securely-siddhesh-jog). In the “myDateTime.java” class, I recommend setting the “mySecond”, “myMinute” and “myHour” variables to private, then adjusting the “retrieveDateTime()” method to set each entry in the array to one of those variables.
7. After implementing these solutions, the final step is to run the static testing tool once more to ensure no new vulnerabilities were introduced by updating the dependencies.