****

# 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** | **05/24/2022** | **Alexandrea Teigeler** |  |

## 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

Alexandrea Teigeler

## 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?

The value of secure communications to the company is paramount, in order to keep their clients private information safe, as well to protect their employees personal identifying data, their websites application architecture hidden, and their database from being corrupted and compromised. If any of this were to occur, not only would there be a loss of financial concerns that would be difficult to recover, but their good reputation of the public, of their customers would be largely lost.

The company advises its customers with financial consultations such as retirement funds, savings accounts, where to invest their capital, as well as insurance policies. International transactions may occur if their customers were to invest in anything international, or if they had international clients wishing to invest and work with them. Cryptography is a possible need for this company for security, if they did business in this manner as it is a government regulated restriction.

External threats would be those trying to secure and steal funds from their clients or the business itself, their personal identification information, how their web application architecture worked, their databases. As modernization continues, it is paramount to keep any open source libraries limited, and to upgrade packages whenever possible to avoid vulnerabilities as they are discovered and fixed.

## 2. Areas of Security

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

Website applications must contend with security threats such as:

1. SQL Injection: When untrusted data is added to database queries, changing the structure of the application, and producing behaviors that are inconsistent with the applications design and purpose.
2. Broken Authentication: Have user sessions timeout, whitelist appropriate characters for passwords that are known to not cause issues through SQL injection attacks, and use multifactor authorization for an added layer of security.
3. XSS (Cross Site Scripting): Can be prevented by encoding user supplied data with query parameterization, sanitizing data, whitelisting input.
4. Insecure Direct Object References (IDOR): Access control checks and sanitize error messages to prevent the exploitation of the web applications URL.

Other considerations and areas that could be compromised that are not apart of the application:

1. Associated Software Information
2. Operating System and Associated Users
3. Users Systems
4. Other Applications in the shared Environment
5. The Backend Database

Specific areas of security applicable to Artemis Financials software application in relation to the Vulnerability Assessment Process Flow Diagram, given the security threats outline prior:

1. Input Verification: Verify that the returned values are within the legal range. Type safely by forcing output into a formal data type. Parameterize every variable for performance.
2. API Interactions: Spring Framework needs to support these interactions.
3. Code Quality

## 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. DocData.java:
   1. **e.printStackTrace** (CWE-209):The web application could generate a message that has information about the environment it runs in, the users, and its associated data. This discloses potentially sensitive information about the physical file paths, source code fragments, the version information about the package the web application used, database, and error messages.
   2. **Connection con = DriverManager.getConnection(“jdbc:mysql://localhost:3306/test”, “root”, “root”);** (CVE-2022-21724) & (CVE-2020-13692) & (CVE-2021-44228): JDBC vulnerability utilization URL attack. The driver does not verify if the class implements the expected interface by instantiating the class. This can lead to code being executed and loaded within arbitrary classes. This may allow injection arguments vulnerabilities and allow local users to execute code. Through this vulnerability, it is possible to provide an SSL factory, where the host’s name isn’t checked and then a man-in-the-middle attacker could masquerade as a trusted user by providing a certificate for the wrong host. Other vulnerabilities from this could be:
      1. Improper Authorization
      2. XXE (XL External Entity)
      3. Access Control Bypass
      4. SQL Injection
      5. Privilege Escalation
      6. Arbitrary Code Execution
      7. Improper Access Control
2. CrudController.java:
   1. @RequestMapping(“/read”) : Due to Spring MVC automatic binding request parameters, it is possible to feed unexpected fields on arguments of this method. Specially crafted user input can be used to change the content of unexpected fields into database. It can provide unauthorized remote code execution.
   2. .toString() : The input values for this function are never verified before the string conversion. SQL Injection is possible since this isn’t checked.
3. GreetingController:
   1. The variable name isn’t checked for the correct type of input. This could lead to a possible SQL Injection.
   2. .format : Since name isn’t check for the correct type of input, it could potentially allow for an information leak and disclosure.
   3. %s : could cause a denial of service if the attacker injected characters into the string.
4. Greeting.java:
   1. GreetingController.java sends an atomic Long to Greeting.java, but the function Greeting only accepts Long. Sending different types of data can cause an integer underflow, which could leak data into other parts of the memory, causing data corruption.

## 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. Bcprov-jdk15on-1.46.jar: Highest Confidence
   1. Code: CVE-2016-1000352:
      1. Severity: High
      2. Flaw: 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.
   2. Code: CVE-2016-1000338:
      1. Severity: Med
      2. Flaw: 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.
   3. Code: CVE-2016-1000344:
      1. Severity: Med
      2. Flaw: 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.
   4. Code: CVE-2016-1000341:
      1. Severity: Med
      2. Flaw: 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.
   5. Code: CVE-2016-1000345:
      1. Severity: Med
      2. Flaw: 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.
   6. Code: CVE-2017-13098
      1. Severity: Med
      2. Flaw: 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."
   7. Code: CVE-2020-15522:
      1. Severity: Med
      2. Flaw: 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.
   8. Code: CVE-2016-1000339:
      1. Severity: Med
      2. Flaw: 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.

## 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 Financials’ software application.

1. DocData.java:
   1. e.printStackTrace :
      1. To handle exceptions internally, do not display error messages containing sensitive information to the user.
   2. Connection con = DriverManager.getConnection(“jdbc:mysql://localhost:3306/test”, “root”, “root”); :
      1. Limit the amount of JNDI data source names to the java protocol in Log4j2, as well as upgrading the packages to the latest versions which fix these known vulnerabilities
2. CrudController.java:
   1. @RequestMapping(“/read”) :
      1. Upgrading the package to the latest version will help fix these issues.
   2. .toString() :
      1. Validate the input of any untrusted data.
3. GreetingController.java:
   1. Variable ‘name’ :
      1. Validate the input to make sure the application is only receiving viable characters for the system.
   2. .format :
      1. .format(“ ‘ “ + name “ ‘ “) will prevent any erroneous issues with the variable string from causing vulnerabilities.
   3. %s :
      1. Upgrading the packages would help alleviate the issue of string input.
4. Greeing.java:
   1. Make sure that input being sent is the same input being received. Validating this input is crucial to prevent integer underflow.