****

# Artemis Financial Vulnerability Assessment Report

Table of Contents

[Artemis Financial Vulnerability Assessment Report 1](#_Toc150535207)

[Document Revision History 3](#_Toc150535208)

[Client 3](#_Toc150535209)

[Developer 4](#_Toc150535210)

[1. Interpreting Client Needs 4](#_Toc150535211)

[2. Areas of Security 4](#_Toc150535212)

[3. Manual Review 5](#_Toc150535213)

[4. Static Testing 8](#_Toc150535214)

[5. Mitigation Plan 12](#_Toc150535215)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **11/11/2023** | **Allan O’Driscoll** | **Project One Vulnerability analysis report** |

## Client



## Developer

Allan O’Driscoll

## Interpreting Client Needs

This project aims to provide a security analysis of an application developed by Artemis Financial. The application supports the creation of personalized financial plans for the company’s customers. These financial plans help customers prepare for the future and include details about their savings, retirement, investments, and insurance policies.

A few key concerns for this application are protecting the integrity of the customer’s account, keeping information confidential, and maintaining the privacy of the customer’s data. If the application fails in any of these key areas, the company will lose business and potentially suffer legal and financial damage. To protect the customer’s data, the application must be written with security in mind. All communications must use secure channels to transfer data, and the information must be encrypted while at rest. The application must also be secured by a strong authentication mechanism. Otherwise, an attacker could easily gain access to the customer’s data. Keeping the customer’s data secure will protect all parties involved.

Given that this is a financial software application, the company may be involved in international transactions. They may conduct business with foreign customers or institutions. This may introduce additional security requirements related to data privacy, reporting, and other government regulations. For example, customers doing business in the European Union are required to comply with the General Data Protection Regulation (GDPR), which regulates how companies handle data and the measures they must take to protect the privacy of individuals (GDPR.EU, n.d.). It will be important to develop the application with these regulations in mind.

The security landscape is constantly shifting, and bad actors are always looking for ways to take advantage of weaknesses in technology. This is a big concern for financial institutions because a successful attack may result in big wins for the attacker. The OWASP organization tracks the most common threats found in web applications. This changes over time as software and technology evolve. In 2021, the top ten issues included Broken Access Control, Cryptographic Failures, and Injection vulnerabilities (OWASP, 2021). This assessment will review each of these and others listed in OWASP’s top-ten list.

The application under study is built on open-source components, including the Spring Framework. Using open-source libraries allows the company to reuse high-quality code that is backed by a community of skilled developers. However, this doesn’t make it immune to security vulnerabilities. In fact, Vulnerable and Outdated Components was number six on OWASP’s top-ten list (OWASP, 2021). Open-source software has many benefits, but it also takes effort to ensure that all the dependencies are free from critical vulnerabilities and that libraries are kept current. In addition, technology is constantly evolving, especially in areas related to security. Today’s solution may not be relevant tomorrow. As a result, developers need to be aware of changes in technology and adapt as necessary.

## Areas of Security

Given the scenario provided by Artemis Financial, this security review should focus on the following areas:

* **Input Validation**: The application is a RESTful web service that accepts input from the user via the query string or the payload of the request. Validating all untrusted input data is often considered the “first line of defense” for an application (Manico & Detlefsen, 2015, p. 16). This is especially true for financial applications.
* **Cryptography**: The application collects and manages sensitive data, including personally identifiable information and financial records. Data of this nature must be encrypted whenever it is sent over the wire and when it is at rest. This helps protect the client and their customers in the event of a data breach.
* **Client/Server**: The application has a three-tier architecture, including a browser-based client or mobile application, a server component (running under the Spring Framework), and a backend database. Communication between each of these tiers will need to be reviewed to ensure that it is secure.
* **Code Error**: The application will inevitably generate errors caused by bad input, network and infrastructure issues, and other exceptional conditions. The code will need to be analyzed to ensure that these conditions are handled securely and that messages presented to the user do not include sensitive information.
* **Code Quality**: This is a general topic that applies to all areas of software development, including security. High-quality code follows best practices that are used consistently throughout the code base. It is the result of careful planning, design, and adherence to industry standards.
* **APIs**: As mentioned earlier, the application is a RESTful web service. It provides a set of APIs that must be able to securely transfer data between the various application tiers. In addition, it must control data through a secure authentication mechanism and role-based access controls. Broken Access Control and Identification and Authentication Failures are two of the items on the OWASP top-ten list (OWASP, 2021). It will be important to verify that the application is secure in these areas.

## Manual Review

The following errors have been identified based on a manual review of the code.

|  |  |  |
| --- | --- | --- |
| **Location** | **Vulnerability Type** | **Description** |
| application.properties line 14 | Insecure Communication | The application is configured to operate on an insecure port and network protocol. Data transferred over the wire will be visible in plain text. |
| CRUDController.java line 1 | Code Quality | This class has no documentation. There may be functionality within this class that has security implications. According to Oracle’s secure coding guidelines, all security-related information should be documented (Oracle, 2023, Guideline 0-7). |
| CRUDController.java line 12 | No Authentication for API | The application does not require any authentication to access document data in the system. |
| CRUDController.java line 12 | No Access Control for API | The application does not implement role-based access controls to secure resources within the API. This means that users have unrestricted access to any business documents managed by the application. Access controls should be implemented for all protected resources (Manico & Detlefsen, 2015, p. 62). |
| CRUDController.java line 12 | No Input Validation | The application does not provide validation for the business\_name parameter, which could be passed on the URL or within form data. All untrusted input should be validated before it is used (Oracle, 2023, Guideline 5-1). |
| CRUDController.java | Information Leakage | The CRUD controller exposes internal class names and data structures to the client. These details could be used by an attacker to gather information about the system. |
| CRUDController.java | Information Leakage | The application does not provide a custom error page. If an exception was thrown during the execution of the code, it would be displayed on this page and could potentially expose sensitive information to the client.    Detailed error messages (such as stack traces) should not be displayed on the client side (OWASP, 2010). |
| customer.java line 1 | Code Quality | This class has no documentation. There may be functionality within this class that has security implications. According to Oracle’s secure coding guidelines, all security-related information should be documented (Oracle, 2023, Guideline 0-7). |
| customer.java line 12 | No Input Validation | The deposit method has no validation on the amount parameter. For example, the parameter could be a negative value, resulting in a debit. It may also be vulnerable to integer overruns (or underruns). The code should throw an error instead of executing this transaction. All untrusted input should be validated before it is used (Oracle, 2023, Guideline 5-1). |
| customer.java line 5 | Incorrect Data Type | An integer is not the correct data type to store an account balance. |
| customer.java line 5 | Incorrect Access Modifier | Data in the customer class is not properly encapsulated. Specifically, the account\_balance field should be private to protect the integrity of the data. |
| customer.java line 4 | Incorrect Construction | The account\_number and account\_balance fields are never initialized. This is a code quality issue. |
| DocData.java line 1 | Code Quality | This class has no documentation. There may be functionality within this class that has security implications. According to Oracle’s secure coding guidelines, all security-related information should be documented (Oracle, 2023, Guideline 0-7). |
| DocData.java line 6 | Code Quality | The id field is never initialized. According to the OWASP Secure Coding Practices, all variables should be explicitly initialized before they are used (OWASP, 2010). This is a code quality issue. |
| DocData.java line 26 | Hardcoded Credentials | The JDBC getConnection() method is called using hard-coded credentials for the root user. Passwords should not be included directly in the code, and they should never be stored in plain text (OWASP, 2010). |
| DocData.java line 26 | Insecure Credentials | The credentials used in the call to the JDBC getConnection() method are easy to guess (root, root). Passwords should meet minimum complexity standards (OWASP, 2010). |
| DocData.java line 26 | Memory Management | The JDBC database connection is not explicitly closed. Resources should be released when they are no longer needed (Oracle, 2023, Guideline 1-2). |
| DocData.java line 30 | Exception Handling | Exceptions generated by database connection errors are not properly handled. The code simply prints the stack trace and continues executing the code. The exception handling should do something more than just print the error. Exceptions must be handled in a secure and consistent way (Oracle, 2023, Guideline 1-4). |
| DocData.java line 21 | Code Quality | The read\_document method is incomplete. |
| Greeting.java line 1 | Code Quality | This class has no documentation. There may be functionality within this class that has security implications. According to Oracle’s secure coding guidelines, all security-related information should be documented (Oracle, 2023, Guideline 0-7). |
| GreetingController.java line 1 | Code Quality | This class has no documentation. There may be functionality within this class that has security implications. According to Oracle’s secure coding guidelines, all security-related information should be documented (Oracle, 2023, Guideline 0-7). |
| GreetingController.java line 16 | No Authentication for API | The application does not require any authentication to access this API. |
| GreetingController.java line 16 | No Access Control for API | The application does not implement role-based access controls to secure resources within the API. Access controls should be implemented for all protected resources (Manico & Detlefsen, 2015, p. 62). |
| GreetingController.java line 16 | No Input Validation | The application does not provide validation for the name parameter, which could be passed on the URL or within form data (Oracle, 2023, Guideline 5-1). |
| myDateTime.java line 1 | Code Quality | This class has no documentation. There may be functionality within this class that has security implications. According to Oracle’s secure coding guidelines, all security-related information should be documented (Oracle, 2023, Guideline 0-7). |
| myDateTime.java line 4 | Code Quality | This class has no constructor, and the two methods are left unimplemented. |
| pom.xml | Vulnerable Libraries | The application uses out-of-date open-source libraries with known vulnerabilities. Developers should ensure that all frameworks and libraries are kept up to date (OWASP, 2010). |

## Static Testing

Static testing was conducted using the OWASP dependency-check plugin developed by Jeremy Long. The generated report shows a total of 78 vulnerabilities across 13 dependencies.

A screenshot of a computer

Description automatically generated

Each of these vulnerabilities is listed in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dependency** | **Library Description** | **Published Vulnerabilities (NIST, n.d.)** | **Brief Vulnerability Description and Solution** |
| bcprov-jdk15on-1.46.jar | The Bouncy Castle Cryptographic Package | CVE-2013-1624,  CVE-2015-6644,  CVE-2015-7940,  CVE-2016-1000338,  CVE-2016-1000339,  CVE-2016-1000341,  CVE-2016-1000342,  CVE-2016-1000343,  CVE-2016-1000344,  CVE-2016-1000345,  CVE-2016-1000346,  CVE-2016-1000352,  CVE-2017-13098,  CVE-2018-5382,  CVE-2020-0187,  CVE-2020-26939,  CVE-2023-33201 | Allows the use of insecure ciphers and weak or vulnerable encryption algorithms. Versions up to and including 1.55 are vulnerable (<https://nvd.nist.gov/vuln/detail/CVE-2016-1000352>, <https://nvd.nist.gov/vuln/detail/CVE-2016-1000344>, <https://nvd.nist.gov/vuln/detail/CVE-2016-1000343>, <https://nvd.nist.gov/vuln/detail/CVE-2016-1000342>, <https://nvd.nist.gov/vuln/detail/CVE-2016-1000338>). Upgrade to a newer version. |
| spring-boot-2.2.4.RELEASE.jar | Spring Boot | CVE-2022-27772,  CVE-2023-20873,  CVE-2023-20883 | Vulnerable to denial of service attacks and directory hijacking. Versions up to and including 3.0.6 are vulnerable (<https://nvd.nist.gov/vuln/detail/CVE-2023-20883>, <https://nvd.nist.gov/vuln/detail/CVE-2023-20873>, <https://nvd.nist.gov/vuln/detail/CVE-2022-27772>). Upgrade to a newer version. |
| logback-core-1.2.3.jar | The Logback Core Module | CVE-2021-42550 | Allows arbitrary code execution under some conditions. Versions up to and including 1.2.7 are vulnerable (<https://nvd.nist.gov/vuln/detail/CVE-2021-42550>). Upgrade to a newer version. |
| log4j-api-2.12.1.jar | The Apache Log4j API | CVE-2020-9488,  CVE-2021-44228,  CVE-2021-44832,  CVE-2021-45046,  CVE-2021-45105 | Vulnerable to remote code execution. Versions up to 2.17.1 are vulnerable (<https://nvd.nist.gov/vuln/detail/CVE-2021-44832>, <https://nvd.nist.gov/vuln/detail/CVE-2021-45046>, <https://nvd.nist.gov/vuln/detail/CVE-2021-44228>). Upgrade to a newer version. |
| snakeyaml-1.25.jar | YAML 1.1 parser and emitter for Java | CVE-2017-18640,  CVE-2021-4235,  CVE-2022-1471,  CVE-2022-25857,  CVE-2022-3064,  CVE-2022-38749,  CVE-2022-38750,  CVE-2022-38751,  CVE-2022-38752,  CVE-2022-41854 | Vulnerable to denial of service and remote code execution attacks. Versions up to 2.0 are vulnerable (<https://nvd.nist.gov/vuln/detail/CVE-2022-1471>, <https://nvd.nist.gov/vuln/detail/CVE-2022-25857>, <https://nvd.nist.gov/vuln/detail/CVE-2017-18640>). Upgrade to a newer version. |
| jackson-databind-2.10.2.jar | General data-binding functionality for Jackson | CVE-2020-25649,  CVE-2020-36518,  CVE-2021-46877,  CVE-2022-42003,  CVE-2022-42004,  CVE-2023-35116 | Subject to denial of service and resource exhaustion attacks. Versions up to 2.13.4.1 are vulnerable (<https://nvd.nist.gov/vuln/detail/CVE-2021-46877>, <https://nvd.nist.gov/vuln/detail/CVE-2022-42004>, <https://nvd.nist.gov/vuln/detail/CVE-2022-42003>, <https://nvd.nist.gov/vuln/detail/CVE-2020-36518>, <https://nvd.nist.gov/vuln/detail/CVE-2020-25649>). Upgrade to a newer version. |
| tomcat-embed-core-9.0.30.jar | Core Tomcat implementation | CVE-2019-17569,  CVE-2020-11996,  CVE-2020-13934,  CVE-2020-13935,  CVE-2020-13943,  CVE-2020-17527,  CVE-2020-1935,  CVE-2020-1938,  CVE-2020-8022,  CVE-2020-9484,  CVE-2021-24122,  CVE-2021-25122,  CVE-2021-25329,  CVE-2021-30640,  CVE-2021-33037,  CVE-2021-41079,  CVE-2021-43980,  CVE-2022-29885,  CVE-2022-34305,  CVE-2022-42252,  CVE-2023-28708,  CVE-2023-41080,  CVE-2023-42795,  CVE-2023-44487,  CVE-2023-45648 | Vulnerable to input validation, resource exhaustion, denial of service, incorrect request parsing, and other types of attacks. Versions up to an including 10.1.13 are vulnerable (<https://nvd.nist.gov/vuln/detail/CVE-2020-1938>, <https://nvd.nist.gov/vuln/detail/CVE-2020-11996>, <https://nvd.nist.gov/vuln/detail/CVE-2020-8022>, <https://nvd.nist.gov/vuln/detail/CVE-2022-42252>). Upgrade to a newer version. |
| hibernate-validator-6.0.18.Final.jar | Hibernate's Bean Validation (JSR-380) reference implementation. | CVE-2020-10693 | Vulnerable to errors in expression parsing. Versions up to 6.1.5 are vulnerable (<https://nvd.nist.gov/vuln/detail/CVE-2020-10693>). Upgrade to a newer version. |
| spring-web-5.2.3.RELEASE.jar | Spring Web | CVE-2016-1000027,  CVE-2020-5421,  CVE-2021-22096,  CVE-2021-22118 | Vulnerable to remote code execution and privileged escalation attacks. Versions up to and including 5.3.16 are vulnerable (<https://ossindex.sonatype.org/vulnerability/CVE-2016-1000027>, <https://ossindex.sonatype.org/vulnerability/CVE-2021-22118>). Upgrade to a newer version. |
| spring-beans-5.2.3.RELEASE.jar | Spring Beans | CVE-2022-22965 | Vulnerable to remote code execution attacks (<https://ossindex.sonatype.org/vulnerability/CVE-2022-22965>). Upgrade to a newer version. |
| spring-webmvc-5.2.3.RELEASE.jar | Spring Web MVC | CVE-2021-22060 | Vulnerable to malicious input attacks that can cause extra log messages to be generated. Versions up to 5.3.13 are vulnerable (<https://ossindex.sonatype.org/vulnerability/CVE-2021-22060>). Upgrade to a newer version. |
| spring-context-5.2.3.RELEASE.jar | Spring Context | CVE-2022-22968 | Contains errors in handling case sensitivity of fields. Versions up to and including 5.3.18 are vulnerable (<https://ossindex.sonatype.org/vulnerability/CVE-2022-22968>). Upgrade to a newer version. |
| spring-expression-5.2.3.RELEASE.jar | Spring Expression Language (SpEL) | CVE-2022-22950,  CVE-2023-20861,  CVE-2023-20863 | Subject to denial of service attacks caused by malicious expressions. Versions up to and including 5.3.27 are vulnerable (<https://ossindex.sonatype.org/vulnerability/CVE-2022-22950>, <https://ossindex.sonatype.org/vulnerability/CVE-2023-20863>). Upgrade to a newer version. |

## Mitigation Plan

The following is an action plan to address the issues documented in this security review.

* Secure the communications layers with HTTPS and TLS. The application currently listens on port 8081, which is a non-secure port. Information sent over the wire can be traced. Spring has support for HTTPS/TLS but must be configured to use it (Frederick, 2023).
* Encryption database sessions with TLS to ensure that communications are secure between the application and database layers. MySQL has support for this, but it must be configured (MySQL, n.d.).
* Ensure that code is well documented, especially in any areas with security implications (Oracle, 2023, Guideline 0-7). This will improve the overall quality of the code.
* Require authenticated sessions for all API calls. Currently, APIs in CRUDController.java and GreetingController.java allow unrestricted access. This could give an attacker access to protected resources and may be a violation of privacy laws.
* Implement role-based access controls for all API calls. Currently, there are no security checks in the application. Designing and following a security model from the beginning is preferred (Manico & Detlefsen, 2015, p. 62).
* Validate all untrusted input data before it is used (Oracle, 2023, Guideline 5-1). Currently, there is no validation on any of the inputs. This is a major flaw that could enable attacks or allow the application to be misused. In addition, inputs to some methods need extra validation (i.e., the deposit method in customer.java) to prevent abuse.
* Ensure that internal data structures and ID values are not leaked. For example, the CRUDController returns internal class names and reference IDs as part of the response. This is an error that should be corrected. Even if no sensitive data is returned, this information could help further an attack.
* Create a custom page that is used for error responses and exceptional conditions. Avoid including detailed error messages such as stack traces in the response. This information should be logged but not returned to the client (OWASP, 2010).
* Ensure that data types and access modifiers are used appropriately. In general, data should be kept private and encapsulated within the class that manages it. An example of a problem in the code is the account\_balance field in the customer class. This field is not private, so it could be modified externally. The field also uses an integer type, which is not appropriate for currency data.
* Ensure that all fields are explicitly initialized before they are used (OWASP, 2010). For example, the customer class suffers from this problem. The account\_number and account\_balance fields are never initialized, which could lead to unexpected results.
* Do not hard code credentials in the source code. Someone with access to the class or jar files could decompile them and discover this sensitive information. Passwords should never be stored in plain text (OWASP, 2010). Passwords should be moved to a secure location outside of the code and should meet minimum complexity standards (OWASP, 2010).
* Ensure that all resources are closed when they are no longer in use (Oracle, 2023, Guideline 1-2). For example, the JDBC connection in the DocData class is never released. This could result in an exhaustion of resources in the JDBC connection pool and could cause problems at the database layer.
* Ensure that practices for secure exception handling are followed (Oracle, 2023, Guideline 1-4). For example, in the DocData class, an exception is caught when the JDBC connection attempt fails. The code simply prints the stack trace and continues. This is an error; it should do more to handle the failure than print it to the console.
* Remove unnecessary code (OWASP, 2010). There are a number of classes, methods, and even libraries that are not utilized. For example, the bouncy castle library is included but never actually used. Another example is myDataTime.java, which is never referenced. Removing unnecessary code and libraries will reduce the attack surface and make vulnerability scanning easier in the future.
* Upgrade the Spring Framework to version 3.1.5 (or newer). Most of the dependencies within the application are transitive. They are picked up because one of the direct dependencies includes them. As a result, many of the vulnerabilities can be cleared by upgrading the version of the direct dependencies.
* Upgrade Bouncy Castle to version 1.76 or newer. Upgrading to a newer version will reduce the number of vulnerabilities found in the application.
* Upgrade SnakeYAML to version 2.2 or newer.
* Upgrade to a newer version of Java (at least Java 11).
* Ensure that the application runs as a non-privileged user (Oracle, 2023, Guideline 0-3). This limits the damage that a vulnerability can have on a system.

**References**

Frederick, S. (2023, June 7). Securing Spring Boot Applications With SSL. Spring Blog. https://spring.io/blog/2023/06/07/securing-spring-boot-applications-with-ssl

GDPR.EU. (n.d.). General Data Protection Regulation (GDPR). GDPR.eu. https://gdpr.eu/tag/gdpr/

Long, J. (n.d.). Jeremylong/DependencyCheck: Owasp dependency-check is a software composition analysis utility that detects publicly disclosed vulnerabilities in application dependencies. GitHub. https://github.com/jeremylong/DependencyCheck

Manico, J., & Detlefsen, A. (2015). Iron-clad java: Building secure web applications. McGraw-Hill Education.

MySQL. (n.d.) Configuring MySQL to Use Encrypted Connections. MySQL. https://dev.mysql.com/doc/refman/8.0/en/using-encrypted-connections.html

NIST. (n.d.). NATIONAL VULNERABILITY DATABASE. NVD. https://nvd.nist.gov/

Oracle. (2023, May). Secure Coding Guidelines for Java SE. Secure coding guidelines for java SE. https://www.oracle.com/java/technologies/javase/seccodeguide.html

OWASP. (2010, August). OWASP SCP quick reference guide V1 - owasp foundation. https://owasp.org/www-pdf-archive/OWASP\_SCP\_Quick\_Reference\_Guide\_v1.pdf

OWASP. (2021). OWASP Top 10:2021. https://owasp.org/Top10/