# Artemis Financial Vulnerability Assessment Report

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **25 May 2024** | **Kyunghoon Lee** | **Conducting initial assessment.** |

## Client



## Developer

Kyunghoon Lee

**1. Interpreting Client Needs**

Determine your client’s needs and potential threats and attacks associated with the company’s application and software security requirements. Consider the following questions 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 on secure communications to consider?
* What external threats might be present now and in the immediate future?
* What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?

The client (known from here on as Artemis Financial) is a consulting company based primarily around developing tailored financial plans (savings, retirement, investments, insurance) for its customers. As with any company involved in the finance industry, secure communications are of the utmost importance to the client. It is expected that they their daily operations will involve handling of sensitive and personal information pertaining to its customers, such as identification or Social Security Numbers, banking account information, names and addresses, or credit information.

It is undisclosed whether Artemis Financial specifically conducts international transactions but given how common international transactions are in the modern world and that we at Global Rain support customers around the world, it is a safe assumption that international transactions must be covered as well. For United States-based transactions, a major point of legislation to consider would be the Financial Modernization Act of 1999, containing the Financial Privacy Rule; this governs collection and disclosure of consumers’ personal financial information. It also includes the Safeguards Rule, setting in place requirements for financial institutions to protect their customers’ data (*Financial Privacy*, 2021). Additionally, international transactions must adhere to the regulations within the regions involved- a common example of this would be the European Union’s General Data Protection Regulation (GDPR), detailing data handling and privacy notices within the EU (*EU GDPR | Office of Ethics*, n.d.).

External threats to consider in this assessment can range from individual criminals seeking out compromised data for purposes like identity theft to potentially hostile foreign nations conducting full-scale cyberwarfare. As such, modernization techniques should be implemented at every level to maintain the best security possible. One such example of a modernization technique would be the use of open-source libraries, which offer the advantage of potentially thousands of code reviewers on top of strong community support and transparency. Another point of modernization may be in the form of regular security audits to be conducted by IT and cybersecurity teams to reveal potential vulnerabilities before they can be exploited.

**2. Areas of Security**

Refer to the vulnerability assessment process flow diagram. Identify which areas of security apply to Artemis Financial’s software application. Justify your reasoning for why each area is relevant to the software application.

* Input Validation can be expected as a relevant area of security given that the application involves entry and return of customer data and account information as strings and integers, respectively.
* APIs are a relevant area of security as the application involves interfacing with and accessing both internal and external data.
* Cryptography is a relevant area of security as internal data should be safely encrypted, as well as any data in transit.
* Code Error is a relevant area of security as proper and secure error handling within the application should minimize exploitation of code-related vulnerabilities, before any sensitive information is compromised outside of the application.
* Code Quality is a relevant area of security to provide a method for control or permissions handling internally, ensuring data is accessed only when the appropriate and correct conditions are met.

**3. Manual Review**

Continue working through the vulnerability assessment process flow diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

* Within Greeting.java and GreetingController.java, it would be worth implementing a form of input validation regarding the data passed by the getter methods (hibernate-validator?)

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

* There does not seem to be any form of cryptography implemented into the application. This could potentially be combined with input validation, if implemented.
* Within the controllers, consistently using @GetMapping as opposed to @RequestMapping may be more specific and improve readability.

A screen shot of a computer program

Description automatically generated

* We should ensure the API endpoints have appropriate security measures, such as authentication.
* There is an error handling block in DocData.java, but it may be worth implementing global error handling to account for the rest of the application.
* Some of the content within the pom.xml are using outdated versions:
  + spring-boot-starter-parent (2.2.4.RELEASE), newest is 3.3.0
  + Java (v1.8/Java 8) is almost a decade old
  + bcprov-jdk15on 1.46 contains vulnerabilities and should be changed to bcprov-jdk18on
  + dependency-check-maven (5.3.0), current is 9.2.0. I updated this myself to run the static testing.

A screen shot of a computer program

Description automatically generated

**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 the dependency-check report. Include the following items:

* The names or vulnerability codes of the known vulnerabilities
* A brief description and recommended solutions provided by the dependency-check report
* Any attribution that documents how this vulnerability has been identified or documented previously

A screenshot of a computer

Description automatically generated

Following are the highest severity vulnerabilities for each package, as well as mitigation recommendations.

* bcprov-jdk15on-1.46: bouncycastle - Improper Validation of Certificate with Host Mismatch. The software communicates with a host that provides a certificate, but the software does not properly ensure that the certificate is actually associated with that host. Migrate to bcprov-jdk18on.
* hibernate-validator-6.0.18.Final: 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. Update to newest version (8.0.x)
* jackson-databind-2.10.2: 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. Update to newest version (2.17.x)
* log4j-api-2.12.1: 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. Fixed in Apache Log4j 2.12.3 and 2.13.1. Update to newest non-beta (2.23.x)
* logback-core-1.2.3: A serialization vulnerability in logback receiver component part of logback version 1.4.11 allows an attacker to mount a Denial-Of-Service attack by sending poisoned data. Update to newest version (1.5.x)
* snakeyaml-1.25: SnakeYaml's Constructor() class does not restrict types which can be instantiated during deserialization. Deserializing yaml content provided by an attacker can lead to remote code execution. We recommend using SnakeYaml's SafeConsturctor when parsing untrusted content to restrict deserialization. We recommend upgrading to version 2.0 and beyond. Update to newest version (2.2)
* spring-boot-2.2.4.RELEASE: In Spring Boot versions 3.0.0 - 3.0.5, 2.7.0 - 2.7.10, and older unsupported versions, an application that is deployed to Cloud Foundry could be susceptible to a security bypass. Users of affected versions should apply the following mitigation: 3.0.x users should upgrade to 3.0.6+. 2.7.x users should upgrade to 2.7.11+. Users of older, unsupported versions should upgrade to 3.0.6+ or 2.7.11+. Update to newest supported version (3.3.x)
* spring-boot-starter-web-2.2.4.RELEASE: In Spring Boot versions 3.0.0 - 3.0.5, 2.7.0 - 2.7.10, and older unsupported versions, an application that is deployed to Cloud Foundry could be susceptible to a security bypass. Users of affected versions should apply the following mitigation: 3.0.x users should upgrade to 3.0.6+. 2.7.x users should upgrade to 2.7.11+. Users of older, unsupported versions should upgrade to 3.0.6+ or 2.7.11+. Update to newest supported version (3.3.x)
* spring-core-5.2.3.RELEASE: A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e. the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it. Update to newest version (6.1.3+)
* spring-web-5.2.3.RELEASE: Pivotal Spring Framework through 5.3.16 suffers from a potential remote code execution (RCE) issue if used for Java deserialization of untrusted data. Depending on how the library is implemented within a product, this issue may or not occur, and authentication may be required. NOTE: the vendor's position is that untrusted data is not an intended use case. The product's behavior will not be changed because some users rely on deserialization of trusted data. Update to newest version (6.1.6+)
* spring-webmvc-5.2.3.RELEASE: A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e. the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it. Update to newest version (6.0.14+)
* tomcat-embed-core-9.0.30: 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. Update to newest supported version (11.0.0-M17+)
* tomcat-embed-websocket-9.0.30: 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. Update to newest supported version (11.0.0-M17+)

**5. Mitigation Plan**

Interpret the results from the manual review and static testing report. Then identify the steps to mitigate the identified security vulnerabilities for Artemis Financial’s software application.

The dependency vulnerabilities can be addressed by updating each of the impacted dependencies or their associated packages to the newest supported version. <https://mvnrepository.com> was used as the primary reference for the latest available versions of each package. For any packages that have been migrated but cannot be supported within our application, we should consider alternative libraries as substitutes. It also appears the Java version used in our application is 1.8 or Java 8, which is severely outdated. At minimum, we should upgrade to a newer version with long-term support (LTS) if necessary, such as Java 17. Using hibernate-validator, we should implement use of constraint annotations within the controllers as a form of input validation. Global error handling could be implemented by adding a dedicated exception handler class, or implementing it specific to our existing classes (though that may be more difficult). To secure our API endpoints, Spring offers a Security package we could add to our application to implement an authentication system.

**References**

*EU GDPR | Office of Ethics*. (n.d.). <https://ethics.berkeley.edu/privacy/international-privacy-laws/eu-gdpr>

*Financial Privacy*. (2021, July 16). Federal Trade Commission. <https://www.ftc.gov/news-events/topics/protecting-consumer-privacy-security/financial-privacy>

Maven Repository. (n.d.). Retrieved May 25, 2024, from <https://mvnrepository.com>