# CS 305 Project One

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **7/21/2024** | **Jordan Santiago** | **Initial Report** |

## Client



## Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In this report, identify your security vulnerability findings and recommend the next steps to remedy the issues you have found.

* Respond to the five steps outlined below and include your findings.
* Respond using your own words. You may also include images or supporting materials. If you include them, make certain to insert them in the relevant locations in the document.
* Refer to the Project One Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Jordan Santiago

**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?

Artemis financial is a consulting company that creates financial plans for its customers, which means there is a valid concern for software security. The financial plans include sensitive information about its customer’s financial data, which can be intercepted and stolen by hackers should they be able to exploit any vulnerabilities in the software. For this reason, it is paramount that all communications between Artemis Financial and their customers be absolutely secure. Artemis financial conducts business and a global capacity, with entrepreneurs, businesses commas and government agencies around the world being their customers. For this reason, we will need to consider any governmental restrictions on secure communications for the United States as well as any foreign countries that the customers are located in. An external threat that present now, or may be in the immediate future, Could be hackers we're looking for financial data in order to potentially blackmail victims, or still there banking data allowing them access to Artemis financials customers money. Some modernization requirements that should be considered would be the usage of a secured HTTPS web application, an authorization structure to limit access to the data, and from encryption of the data.

**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
  + Input will need to be validated to ensure that only authorized personnel are inputting data into the system and that hackers are not able to input their own malicious code.
* APIs
  + Artemis Financial will be using restful web application programming interface (API) for their software, which means the application will need to integrate with that API securely.
* Cryptography
  + Since this data will be sent to various places throughout the world, adding encryption can help protect stolen data from being read.
* Code Error
  + The application will need to be able to handle erroneous inputs in order to ensure the application is able to detect and handle anomalies, which can lead to data corruption or intrusion.
* Code Quality
  + Using higher quality code and secure coding practices, loopholes and vulnerabilities can be minimized.
* Encapsulation
  + Encapsulation is relevant to the security of Artemis Financial’s software application because it will segregate data and methods, protecting from external access.

**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. I was not able to identify any encryption for any stored data
2. No input validation was found throughout the system
3. Spring is not the latest version and should be updated
4. There is no integration with the RESTful API, even though Artemis Financial stated they use it.
5. There is no output to verify the functionality of the code.
6. DocData has error handling, however there is no other error handling in the other classes
7. The CRUD class has two methods named “CRUD” and the duplication should be resolved

**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

1. bcprov-jdk15on-1.46.jar
   1. **Description**: In Bouncy Castle JCE Provider 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.
   2. **Recommended Solution**: Update to 1.78
2. hibernate-validator-6.0.18.Final.jar
   1. **Description**: 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.
   2. **Recommended Solution**: Update to 8.0.1.Final
3. jackson-databind-2.10.2.jar
   1. **Description**: 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.
   2. **Recommended Solution**: Update to 2.17.2
4. log4j-api-2.12.1.jar
   1. **Description**: 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
   2. **Recommended Solution**: Update to 3.0.0-beta2
5. logback-core-1.2.3.jar
   1. **Description**: A serialization vulnerability in logback receiver component part of
   2. logback version 1.4.11 allows an attacker to mount a Denial-Of-Service
   3. attack by sending poisoned data.
   4. **Recommended Solution**: Update to 1.5.6
6. snakeyaml-1.25.jar
   1. **Description**: 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.
   2. **Recommended Solution**: Update to 2.2
7. spring-boot-2.2.4.RELEASE.jar
   1. **Description**: 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+.
   2. **Recommended Solution**: Update to 3.3.2
8. spring-boot-starter-web-2.2.4.RELEASE.jar
   1. **Description**: 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+.
   2. **Recommended Solution**: Update to 3.3.2
9. spring-core-5.2.3.RELEASE.jar
   1. **Description**: 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.
   2. **Recommended Solution**: Update to 6.1.11
10. spring-web-5.2.3.RELEASE.jar
    1. **Description**: 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.
    2. **Recommended Solution**: Update to 6.1.11
11. spring-webmvc-5.2.3.RELEASE.jar
    1. **Description**: 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.
    2. **Recommended Solution**: Update to 6.1.11
12. tomcat-embed-core-9.0.30.jar
    1. **Description**: 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.
    2. **Recommended Solution**: Update to 11.0.0-M22
13. tomcat-embed-websocket-9.0.30.jar
    1. **Description**: 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.
    2. **Recommended Solution**: Update to 11.0.0-M22

**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.

There are several things that can be done to help mitigate some of the vulnerabilities that were identified. First and foremost, Spring should be updated to the latest version, 6.1.11. That alone will likely clear up a lot of the identified vulnerabilities. However, the remaining vulnerabilities can be updated individually in accordance with the above. Closing out those vulnerabilities will strengthen the software application’s security greatly. For the actual code, integrating encryption, input validation, and cleaning up the code quality will mitigate some more vulnerabilities.