# CS 305 Project

Alex Surprenant

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **11/13/2024** | **Alex Surprenant** |  |

## 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

Alex Surprenant

**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?
  1. As a financial institution, Artemis Financial places great emphasis on the security of its platform, which is apparent by the fact that they hired us to implement the latest and most effective software security system
* 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?

**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.

[Include your findings here.]

**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. In the CRUDController.java class, the class accepts the business\_name parameter from the user without any type of validation. This could allow for injection attacks from an attacker
2. In the GreetingController.java class, the greeting() function takes the “name” parameter to create a greeting message, again without validation. This can be used by attackers to inject bad code into the system.
3. Both the GreetingController.java and the CRUDController.java classes expose endpoints /greeting and /read respectively, and both classes do not have any access controls implemented or input validation as noted in 1 and 2. This can allow anyone to make calls to the API potentially causing a security risk,
4. The CRUDController.java class takes the data from the instance of DocData and returns it as a string via doc.toString(). If DocData included any sensitive information, this could cause a breach in security.
5. In the DocData.java class, the try catch block does not have good error handling. If there is a SQLExeption, the e.printStackTrace() function is called which could expose sensitive information about the database, like user information, into the error log.
6. In the application.properties file, the max-file-size is pretty large. This could allow an attacker to upload really large files consecutively and overwhelm the server, inflicting a denial-of-service attack.
7. The customer.java class has no checks for authorized users, which, in conjunction with the other potential risks in the system, can allow anyone to make changes to account balances or view account numbers.

**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:** Software communicates with a host that provides a certificate, but does not ensure certificate is associated with host.
      2. **hibernate-validator-6.0.18.Final.jar:** A flaw was found that would allow an attacker to bypass a validator by omitting a tag ending in a less-than character, allotting HTML injection or Cross-Site-Scripting attacks
      3. **jackson-databind-2.10.2.jar:** A flaw found in FasterXML Jackson Databind where it did not have entity expansion secured properly, which allows potential XML external entity attacks.
      4. **log4j-api-2.12.1.jar:** Improper validation of certificate with host mismatch in Apache Log4j SMTP appender, which could allow interception of connection from man in the middle attack.
      5. **logback-core-1.2.3.jar:** Serialization vulnerability in logback receiver would allow an attacker to mount a denial of service attack.
      6. **snakeyaml-1.25.jar:** SnakeYaml’s constructor() class does not restrict types which can be instantiated during deserialization. This can lead to remote execution by an attacker.
      7. **spring-boot-2.2.4.RELEASE.jar:** Older version of Spring Boot could be susceptible to a security bypass.
      8. **spring-boot-starter-web-2.2.4.RELEASE.jar:** Older version of Spring Boot could be susceptible to a security bypass.
      9. **spring-core-5.2.3.RELEASE.jar:** A Spring MVC or Spring WebFlux Application running on jdk 9+ may be vulnerable to remote execution.
      10. **spring-expression-5.2.3.RELEASE.jar:** A Spring MVC or Spring WebFlux Application running on jdk 9+ may be vulnerable to remote execution.
      11. **spring-web-5.2.3.RELEASE.jar:** Pivitol Spring Framework through version 5.3.16 can suffer from remote code execution if used for Java deserialization of untrusted data.
      12. **spring-webmvc-5.2.3.RELEASE.jar:** A Spring MVC or Spring WebFlux Application running on jdk 9+ may be vulnerable to remote execution.
      13. **tomcat-embed-core-9.0.30.jar:** When using Apache JServ Protocol, care must be taken in trusting incoming connection to Apache Tomcat. The Tomcat AJP Connector has a vulnerability that can allow an attacker to access files and execute remote code in the system.
      14. **tomcat-embed-websocket-9.0.30.jar:** When using Apache JServ Protocol, care must be taken in trusting incoming connection to Apache Tomcat. The Tomcat AJP Connector has a vulnerability that can allow an attacker to access files and execute remote code in the system.

**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.

[Include your findings here.]