# CS 305 Project One - Jordan Bankston

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **January 31, 2025** | **Jordan Bankston** |  |

## 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 Bankston

**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?

Secure communications are essential for safeguarding sensitive corporate information, encompassing client data, financial records, and proprietary assets. Within the company's REST service structure, ensuring the confidentiality and integrity of data transmission is essential for averting data breaches and cultivating stakeholder trust. Robust encryption techniques, HTTPS protocols, and secure token-based authentication systems can do this. The existing software does not specifically accommodate foreign transactions; however, web-based apps inherently have the capacity to cater to a global audience. Consequently, if the organization participates in international transactions, adherence to legislation such as the General Data Protection Regulation (GDPR) and other regional data protection statutes is essential.

Government laws are essential in formulating safe communication solutions. Organizations must comply with legislative frameworks such as the Health Insurance Portability and Accountability Act (HIPAA), Payment Card Industry Data Security Standard (PCI DSS), or the Federal Information Security Management Act (FISMA), depending on the nature of the data processed. These requirements require procedures including encrypted data storage, secure key management, and routine compliance audits to protect against potential attacks. External threats to the company's application manifest in various forms, including SQL and NoSQL injections resulting from inadequate input validation, cross-site scripting (XSS) vulnerabilities stemming from insufficient output sanitization, phishing attacks targeting employee credentials, and advanced persistent threats (APTs) capable of circumventing standard security protocols without comprehensive monitoring systems.

To mitigate these dangers, firms must prioritize modernization and ongoing security improvements. Incorporating safe open-source libraries, routinely upgrading dependencies, and doing vulnerability assessments with tools such as OWASP Dependency-Check are critical practices. Moreover, the adoption of contemporary security protocols like OAuth 2.0 and OpenID Connect for secure authentication, along with the implementation of API gateways for improved access control, can substantially enhance security. Cloud-hosted applications must adhere to the security protocols established for cloud environments. This encompasses encrypting data during transmission and storage, securely managing identities, and automatically detecting security vulnerabilities. The organization must prioritize encrypted communications, comply with international and regulatory security standards, and aggressively counteract growing cyber threats through ongoing updates, secure coding methods, and regular security evaluations.

**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.

Utilizing the Vulnerability Assessment Process Flow, various critical security domains pertain to the company's software application. Initially, input validation is crucial to guarantee that all data entering the system is pristine, properly structured, and secure, hence reducing dangers such as SQL injection and cross-site scripting (XSS). Implementing robust validation checks diminishes the likelihood of harmful data exploitation.

Secondly, API security is crucial due to the application's dependence on REST services. Secure API interactions are essential to avert unauthorized access, data breaches, and the exploitation of weaknesses such as compromised authentication or insecure endpoints. Implementing robust authentication procedures and rate limits can improve API security.

Cryptography is a crucial domain that emphasizes the appropriate application of encryption for both stored and transmitted data. Unauthorized entities are unable to intercept or access confidential information, particularly about personal or financial data.

Implementing error management in code is essential for detecting and managing problems securely, hence preventing the disclosure of critical system information. Robust error handling obstructs attackers from acquiring knowledge about the application's internal mechanisms via error messages.

Ultimately, Code Quality ensures the uniform implementation of secure coding methodologies across the development lifecycle. This entails compliance with coding standards, frequent code evaluations, and static analysis to identify and rectify problems promptly.

Although not all seven parts of the process flow are relevant, concentrating on these essential security domains fosters a formidable defense against potential threats, hence ensuring the application's security and resilience.

**3. Manual Review**

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

Upon manually inspecting the codebase, the following vulnerabilities were identified:

* Lack of Input Validation: The CRUDController.java accepts user input directly via the @RequestParam without validating the input. This could lead to injection attacks if malicious data is processed.
* Exposure of Sensitive Information: Returning doc.toString() without sanitizing its output might expose sensitive data inadvertently.
* Missing Authentication and Authorization: There is no authentication mechanism to restrict access to the /read endpoint, which could allow unauthorized users to retrieve data.
* Insecure API Endpoints: The REST endpoints do not enforce HTTPS, leaving data vulnerable to man-in-the-middle (MitM) attacks during transmission.
* Hardcoded Configuration: If sensitive information like database credentials is stored in plain text within configuration files, it increases the risk of credential leakage.
* Improper Error Handling: The code lacks structured error handling, which might expose stack traces or detailed error messages to end-users, providing attackers with insights into the application structure.
* Potential Insecure Data Handling: The data flow between classes like DocData and CRUD isn’t clearly secured, raising concerns about data integrity and confidentiality.
* Absence of Logging and Monitoring: There are no logging mechanisms in place to track suspicious activities, making it harder to detect security breaches.
* Unvalidated Redirects and Forwards: If not controlled properly, redirects and forwards can be exploited to redirect users to malicious sites.
* Inconsistent Use of Secure Coding Practices: The code lacks uniform implementation of secure coding guidelines, increasing the likelihood of vulnerabilities due to inconsistent development practices.

**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

The dependency check on Artemis Financial's software application revealed the following vulnerabilities:

1. **bcprov-jdk15on-1.46.jar**
   * **Severity:** High
   * **Vulnerability IDs:** 20 CVEs
   * **Description:** Issues related to cryptographic implementations that could lead to security weaknesses.
   * **Recommendation:** Upgrade to the latest stable version of Bouncy Castle.
2. **spring-boot-2.2.4.RELEASE.jar**
   * **Severity:** Critical
   * **Vulnerability IDs:** 3 CVEs
   * **Description:** Vulnerabilities affecting the Spring Boot framework, potentially allowing remote code execution.
   * **Recommendation:** Upgrade to a more recent, patched version of Spring Boot.
3. **log4j-api-2.12.1.jar**
   * **Severity:** Critical
   * **Vulnerability IDs:** 5 CVEs
   * **Description:** Log4j vulnerabilities, particularly related to remote code execution through JNDI lookups.
   * **Recommendation:** Upgrade to Log4j 2.17.1 or later.
4. **snakeyaml-1.25.jar**
   * **Severity:** Critical
   * **Vulnerability IDs:** 10 CVEs
   * **Description:** YAML parsing vulnerabilities that may lead to arbitrary code execution.
   * **Recommendation:** Update to the latest version of SnakeYAML.
5. **jackson-databind-2.10.2.jar**
   * **Severity:** High
   * **Vulnerability IDs:** 6 CVEs
   * **Description:** Insecure deserialization vulnerabilities that could be exploited for remote code execution.
   * **Recommendation:** Upgrade to Jackson-databind version 2.9.10.5 or later.
6. **tomcat-embed-core-9.0.30.jar**
   * **Severity:** Critical
   * **Vulnerability IDs:** 27 CVEs
   * **Description:** Vulnerabilities in Apache Tomcat allowing potential request smuggling and other attacks.
   * **Recommendation:** Upgrade to Apache Tomcat 10.0.6 or later.
7. **hibernate-validator-6.0.18.Final.jar**
   * **Severity:** Medium
   * **Vulnerability IDs:** 2 CVEs
   * **Description:** Bugs in message interpolation that could bypass validation checks.
   * **Recommendation:** Upgrade to Hibernate Validator 6.0.20 or newer.
8. **spring-web-5.2.3.RELEASE.jar**
   * **Severity:** High
   * **Vulnerability IDs:** 8 CVEs
   * **Description:** Vulnerabilities related to improper input handling and security bypass issues.
   * **Recommendation:** Update to the latest stable version of Spring Web.
9. **spring-beans-5.2.3.RELEASE.jar**
   * **Severity:** High
   * **Vulnerability IDs:** 1 CVE
   * **Description:** Security flaws in Spring Beans potentially leading to privilege escalation.
   * **Recommendation:** Upgrade to the latest Spring Beans release.
10. **spring-webmvc-5.2.3.RELEASE.jar**
    * **Severity:** High
    * **Vulnerability IDs:** 2 CVEs
    * **Description:** Issues with web request processing that can be exploited for unauthorized access.
    * **Recommendation:** Update to the latest version of Spring Web MVC.

**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.

To rectify the identified security vulnerabilities, it is essential to implement robust input validation, ensure the utilization of HTTPS with secure API authentication, and encrypt data during transmission and storage. Consistently updating dependencies, eliminating hardcoded credentials, and improving error handling will mitigate hazards. Implementing logging and monitoring tools, along with regular security evaluations and developer training, will enhance the overall security framework of Artemis Financial's software application.