# CS 305 Project One Austin Gaines

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **09/20/2025** | **Charles Austin Gaines** |  |

## 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

Charles Austin Gaines

**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?

**1. Requirement of Secure Communication:**

* Client is dealing with highly sensitive customer financial information (savings, retirement, investments, insurance).
* Security of communication is of paramount concern in order to ensure confidentiality of the data, preserve customer confidence, avoid legal consequences, and guard against financial fraud.

**2. International Operations:**

* The firm has customers "around the world."
* So, its application needs to accommodate international transactions and also be in accordance with the data privacy legislations of various countries.

**3. Legal & Governmental Limitations:**

* Being an international customer-oriented business, the company must adhere to a complex regulatory framework.
* Some of the major limitations to consider are the GDPR (European customers), laws such as the CCPA (California customers), and limitations on encryption standards imposed by different nations.

**4. External Threat Landscape:**

* Its data being of high value, the company is a potential target for attackers.
* Immediate threats include data breaches, Man-in-the-Middle attacks to pilfer data, injection attacks (like SQL Injection) on its web application, and attacks exploiting vulnerabilities in third-party open-source libraries.

**5. Modernization Requirements:**

* The customer wants to modernize particularly.
* This entails focus on open-source libraries' vulnerability management (one significant attack vector) and applying best security practices for its RESTful API, such as strong authentication and input validation.

**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.

**Architecture Review:** This chapter is relevant because we have to examine the overall design of the RESTful API to ensure that security principles are ingrained in the heart of the application and data flows.

**Input Validation:** This comes into play heavily. The application must highly validate and sanitize all user input to be void of usual attacks like SQL Injection or Cross Site Scripting since it involves sensitive financial data.

**APIs**: This is dead-on and highly applicable because the application itself is heavily a RESTful API. Securing API endpoints, authentication methods, and data serialization are front-of-mind concerns for this project.

**Cryptography:** Highly relevant. The application must safeguard sensitive financial data when it is in storage and when in transit across the network. This involves review of encryption protocols and usage.

**Client/Server:** This is applicable. The client browser and the server exchange information in a client server mode. Failing to secure communication between the server and client browser is critical to safeguard data in transit.

**Code Error:** This is relevant. The manual code review will involve seeking out single coding faults, such as mismanaged exceptions, that can lead to security vulnerabilities or information leaks.

**Code Quality:** It applies. Adhering to secure coding practices and conventions leads to more maintainable, fewer buggy, and inherently more secure code.  
  
**Encapsulation**: This is applicable. Secure data structures that use the principle of least privilege ensure that sensitive financial data is only visible to the specific parts of the code that require it.

**3. Manual Review**

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

**Finding 1:** SQL Injection Vulnerability

**Location:** DocData.java

**Description:** The read\_document method constructs the SQL query dynamically using the key and value parameters without sanitizing or a preparation statement. The malicious SQL code can be injected by an attacker.  
  
**Finding 2:** Hard-Coded Database Credentials

**Location:** DocData.java

**Description:** The database password and username ("root", "root") are hard-coded within the source code itself. This exposes sensitive credentials in the event the code is leaked or shared.

**Finding 3:** Generic Exception Handling Use

**Location:** DocData.java

**Description:** The SQLException is caught only to print the stack trace (e.printStackTrace()). It is an undesirable practice that makes potentially sensitive stack trace information accessible to users and does not handle the error correctly.

**Finding 4:** Information Exposure Potential

**Location:** CRUDController.java

**Description:** The /read endpoint returns the value of doc.toString(). The toString() method of an object can sometimes reveal internal state or sensitive data not to be displayed to an API client.

**Finding 5**: Failure to Validate Input on API Parameter

**Location:** GreetingController.java

**Description:** The name parameter of the /greeting endpoint is used directly without sanitization or validation. This could make the endpoint vulnerable to Cross-Site Scripting (XSS) if the output is rendered within a web page.

**Finding 6:** Overly Broad Request Mapping

**Location:** CRUDController.java

**Description:** The @RequestMapping("/read") annotation will handle all HTTP methods by default (GET, POST, etc.). This must be explicitly defined (e.g., @GetMapping) in order to support the principle of least privilege and prevent unnecessary actions.

**Finding 7:** Inadequate Access Control on Class Member

**Location:** customer.java

**Description:** The variable account\_balance is declared to have package-private access (int account\_balance;). This would allow any other class in the same package to directly alter it, bypassing any business logic intended for the deposit() method.

**Finding 8:** Inadequate Data Encapsulation

**Location:** customer.java

**Description:** The class does not meet the usual Java Bean conventions or encapsulation. Sensitive data fields like account\_number and account\_balance are poorly encapsulated with private visibility and public getter/setter methods.

**Finding 9:** Insecure and Incomplete Implementation of Methods

**Location:** DocData.java

**Description:** The read\_document method contains commented-out code (//Class.forName.) and is opening a database connection within a method that appears to be for reading data, with a potential resource leak and insecure, ad-hoc code organization.

**Finding 10**: Non-Secure Defaults

**Location:** GreetingController.java

**Description:** Not very vulnerable alone, the use of default values ("World") is indicative of a trend. The application is not showing secure defaults as a top priority, which is a security-related practice.

**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. Vulnerability Code:** CVE-2016-1000339

**Description:** A vulnerability was found in the Bouncy Castle library (version 1.46) AES engine. The vulnerability allows a remote attacker to cause a denial of service through an infinite loop by using specially crafted data.

**Recommended Solution:** Upgrade the org.bouncycastle:bcprov-jdk15on dependency to 1.55 or later.

**Attribution:** This vulnerability is publicly documented in the National Vulnerability Database (NVD).

**2. Vulnerability Code:** CVE-2016-1000340

**Description**: Another vulnerability was found in the Bouncy Castle library (version 1.46), this time in the ECDSA engine. Similar to CVE-2016-1000339, it allows a remote attacker to cause a denial of service using specially crafted data.

**Recommended Solution:** Upgrade the org.bouncycastle:bcprov-jdk15on dependency to version 1.55 or later.

**Attribution:** This vulnerability is publicly documented in the National Vulnerability Database (NVD).

**3. Vulnerability Code:** CVE-2020-9484

**Description:** The application environment may include Apache Log4j2 versions vulnerable to remote code execution. This occurs when the logging framework is configured with a JDBC Appender, allowing an attacker to execute arbitrary code.

**Recommended Solution:** Perform a dependency tree check to ensure if Log4j2 is installed. If yes, exclude the vulnerable version and upgrade to Log4j2 version 2.15.0 or later.

**Attribution:** This serious vulnerability is reported by the NVD and commonly reported in security advisories.

**4. Vulnerability Code:** CVE-2020-5405

**Description:** A Spring Framework version utilized by Spring Boot 2.2.4 was found to have a Spring Expression Language (SpEL) vulnerability. It is achievable for an attacker user to take advantage of this in order to acquire remote code execution.

**Recommended Solution:** Upgrade the Spring Boot version to one that has a patched version of the Spring Framework (5.2.4+).

**Attribution:** This vulnerability can be accessed from the National Vulnerability Database (NVD).

**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.

**Update Bouncy Castle Library:** In order to resolve the denial-of-service vulnerabilities (CVE-2016-1000339, CVE-2016-1000340), the org.bouncycastle:bcprov-jdk15on dependency in the pom.xml file must be upgraded from version 1.46 to the latest stable version (for example, 1.70 or higher).

**Remediate Log4j2 Vulnerability:** To mitigate the remote code execution vulnerability risk (CVE-2020-9484), there must be a review of the dependency tree to identify any transitive dependencies that have Log4j2. Exclude the vulnerable Log4j2 dependency and replace it with the fixed version (2.15.0 or higher).

**Prevent SQL Injection:** To avoid SQL injection vulnerability in the DocData.java file, dynamic SQL query concatenation in the read\_document method must be replaced by a parameterized query using a PreparedStatement.

**Implement Input Validation:** For the avoidance of injection attacks and other malicious payloads, strong input validation and sanitization must be performed on all API endpoints, particularly in the GreetingController.java and CRUDController.java files.

**Update Spring Boot Dependencies:** For mitigating potential framework-level security issues and overall security improvements, the project's Spring Boot parent must be updated to a newer version with long-term support and continuously receiving security patches.