# CS 305 Project One

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/21/2024** | **Darrell Walker** | **Initial Creation** |
| **1.1** | **9/21/2024** | **Darrell Walker** | **Added Information** |

## 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

Darrell Walker II

**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:** Artemis Financial needs strong encryption for data in transit, given that sensitive customer financial information is handled so the value is considered high.
* **International Transactions:** If the company processes international transactions, it needs to comply with GDPR and similar regulations for cross-border data transfers.
* **Governmental Restrictions:** U.S. regulations, such as HIPAA (if health-related financial data is involved), require compliance for secure communications.
* **External Threats:** Common threats include phishing, API endpoint attacks, cross-site scripting (XSS), and SQL injection. As the financial industry is a frequent target, securing against ransomware and data exfiltration is crucial.
* **Modernization Requirements:** Ensure the latest security patches are applied to all open source libraries and use secure web frameworks. Open-source components should be regularly reviewed to avoid known vulnerabilities some of these vulnerabilities can be identified by using OWASP.

**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.

* **Authentication and Authorization:** Ensure that all users are properly authenticated and that role-based access control is applied.T his area is about making sure that only the right people can log in and access certain parts of the system. It also ensures that users can only see or do what they are supposed to, based on their role. For a financial company, it’s crucial that customer information is only accessible to authorized staff and not to hackers.
* **Data Encryption:** Encrypt data at rest and in transit, especially given the sensitive nature of financial information. Encryption keeps information secret by turning it into a code. Artemis Financial deals with personal and financial data, so encrypting data ensures that even if someone intercepts it, they won’t be able to understand or use it. This applies both when data is stored and when it’s being sent over the internet.
* **Input Validation:** Protect against injection attacks by ensuring all inputs are sanitized. Input validation checks that users are entering the right type of information. This prevents hackers from entering harmful code that could break the system or steal data. For example, when customers input their account numbers, the system needs to ensure it’s not allowing bad code to slip in.
* **Session Management:** Ensure session timeouts and secure cookie handling to prevent session hijacking. When users log into the system, a session is created to track their activity. If sessions last too long without security measures, attackers might hijack them. It’s important to manage these sessions so that users don’t stay logged in forever and cookies are safe from attacks.
* **Dependency Management:** Regularly update third-party libraries to patch known vulnerabilities. Software often uses outside code called libraries. Sometimes, these libraries can have security problems. If they aren’t updated, hackers might exploit them. Keeping these libraries up to date ensures Artemis Financial isn’t using unsafe or vulnerable code in their system.
* **API Security:** Secure the RESTful API by implementing proper authentication (OAuth2) and securing endpoints. APIs allow different parts of the system to talk to each other, like the website and the database. If not secured properly, hackers can use APIs to get unauthorized access to data. Implementing strong security for APIs ensures that only trusted people and systems can use them.

**3. Manual Review**

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

After reviewing the code base provided, here are 9 key security vulnerabilities that were identified through manual inspection:

1. **Missing Input Validation (CRUDController.java)**
   * **Description:** In this file, there is no validation for user inputs before interacting with the database. This can leave the system vulnerable to SQL injection attacks.
   * **Recommendation:** Add input validation to ensure that data entered by users is properly sanitized before interacting with the database.
2. **Hardcoded Sensitive Data (customer.java)**
   * **Description:** Sensitive information, such as user IDs or passwords, appears to be hardcoded in the file. This exposes sensitive data to anyone with access to the code.
   * **Recommendation:** Store sensitive data in a secure, encrypted manner, such as in a configuration file or environment variable.
3. **Lack of Role-Based Access Control (CRUD.java)**
   * **Description:** There are no checks to ensure that users have the correct roles and permissions to perform actions, allowing anyone logged in to perform sensitive operations.
   * **Recommendation:** Implement role-based access control (RBAC) to restrict access to sensitive functions based on the user’s role.
4. **Unencrypted Sensitive Data (DocData.java)**
   * **Description:** Customer documents and other sensitive data are stored without encryption. If the system is breached, all sensitive data would be easily accessible.
   * **Recommendation:** Encrypt all sensitive data at rest using AES-256 encryption.
5. **Improper Error Handling (GreetingController.java)**
   * **Description:** When the application encounters an error, it outputs detailed error messages. These messages can expose sensitive information about the system architecture to attackers.
   * **Recommendation:** Ensure that error messages shown to users are generic and avoid revealing internal system details. Log the detailed error messages securely for internal review.
6. **Cross-Site Scripting (XSS) Vulnerability (GreetingController.java)**
   * **Description:** User inputs are reflected in the response without being sanitized, leading to a potential XSS vulnerability where malicious scripts can be executed in the user's browser.
   * **Recommendation:** Sanitize all user inputs before returning them in responses to prevent XSS attacks.
7. **No Session Expiry (myDateTime.java)**
   * **Description:** The application does not have proper session management in place, allowing users to stay logged in indefinitely without session expiration.
   * **Recommendation:** Implement session timeouts to limit how long a user can stay logged in. After a period of inactivity, require the user to log in again.
8. **Outdated BouncyCastle Library (org.bouncycastle:bcprov-jdk15on:1.46)**

* **Description:** The BouncyCastle version 1.46 used in the project is outdated and known to contain multiple vulnerabilities. This could expose the system to cryptographic flaws or allow attackers to bypass certain security mechanisms.
* **Recommendation:** Update the BouncyCastle library to the latest stable version. Regularly updating third-party libraries is essential to mitigate known vulnerabilities.

1. **Lack of Dependency Scanning Before Build (Maven Plugin Configuration)**

* **Description:** While there is an OWASP dependency-check plugin configured, there is no setup to ensure that dependency checks are enforced before the build process. This could allow vulnerable dependencies to be included without being flagged.
* **Recommendation:** Configure the OWASP dependency-check plugin to run in all build phases (e.g., validate, compile) to ensure that any vulnerabilities are caught early in the development cycle.

**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 error

Description automatically generated

A screenshot of a computer

Description automatically generated

**1. bcprov-jdk15on:1.46**

* **Vulnerability Codes:**
  + cpe:2.3:a:bouncycastle:bouncy\_castle:crypto:package:1.46
  + cpe:2.3:a:bouncycastle:bcprov-jdk15on:1.46
* **Description:** This version of Bouncy Castle contains known vulnerabilities related to cryptographic failures, which could lead to security weaknesses in encrypted communications.
* **Recommendation:** Update to the latest stable version of Bouncy Castle to mitigate these cryptographic vulnerabilities.

**2. hibernate-validator-6.0.18.Final.jar**

* **Vulnerability Codes:**
  + cpe:2.3:a:hibernate\_validator:6.0.18
* **Description:** This library version has been flagged for potential vulnerabilities in data validation, which could allow bypassing certain validation rules and lead to security issues.
* **Recommendation:** Update to a more secure version of Hibernate Validator to ensure proper input validation.

**3. jackson-databind-2.10.2**

* **Vulnerability Codes:**
  + cpe:2.3:a:jackson:modules:java:8.2.10
* **Description:** Older versions of Jackson Databind are vulnerable to deserialization attacks, where malicious actors can exploit how data is serialized and deserialized.
* **Recommendation:** Upgrade to the latest version of Jackson Databind to avoid deserialization vulnerabilities.

**4. log4j-api-2.12.1**

* **Vulnerability Codes:**
  + cpe:2.3:a:apache:log4j:2.12.1
* **Description:** Though this vulnerability is rated as low severity, older versions of Log4j have potential logging vulnerabilities that could expose sensitive information in logs.
* **Recommendation:** Update Log4j to the latest stable release to secure logging mechanisms.

**5. logback-core-1.2.3**

* **Vulnerability Codes:**
  + cpe:2.3:a:logback:core:1.2.3
* **Description:** Known vulnerabilities in Logback could lead to improper logging of sensitive data and potential exploitation by attackers.
* **Recommendation:** Update Logback to the latest version to enhance secure logging.

**6. spring-boot-2.2.4.RELEASE.jar**

* **Vulnerability Codes:**
  + cpe:2.3:a:vmware:spring\_boot:2.2.4
* **Description:** The Spring Boot framework version 2.2.4 has several critical vulnerabilities that could allow remote code execution or data leaks.
* **Recommendation:** Upgrade to the latest version of Spring Boot to patch these critical vulnerabilities.

**7. spring-core-5.2.3.RELEASE.jar**

* **Vulnerability Codes:**
  + cpe:2.3:a:vmware:spring\_framework:5.2.3
* **Description:** This version of the Spring Core library is vulnerable to injection and deserialization attacks.
* **Recommendation:** Upgrade to a newer version of Spring Core to prevent such attacks.

**8. spring-expression-5.2.3.RELEASE.jar**

* **Vulnerability Codes:**
  + cpe:2.3:a:vmware:spring\_expression:5.2.3
* **Description:** The Spring Expression module is vulnerable to expression language injection, which could allow attackers to execute arbitrary expressions.
* **Recommendation:** Update the Spring Expression module to a newer version to mitigate these risks.

**9. spring-web-5.2.3.RELEASE.jar**

* **Vulnerability Codes:**
  + cpe:2.3:a:vmware:spring\_web:5.2.3
* **Description:** This version of Spring Web is vulnerable to cross-site scripting (XSS) and injection attacks.
* **Recommendation:** Upgrade Spring Web to the latest version to avoid web-related vulnerabilities.

**10. tomcat-embed-core-9.0.30**

* **Vulnerability Codes:**
  + cpe:2.3:a:apache:tomcat:9.0.30
* **Description:** Older versions of Tomcat, including 9.0.30, have critical vulnerabilities related to remote code execution and session hijacking.
* **Recommendation:** Update to the latest version of Tomcat to prevent these severe vulnerabilities.

**11. tomcat-embed-websocket-9.0.30**

* **Vulnerability Codes:**
  + cpe:2.3:a:apache:tomcat-websocket:9.0.30
* **Description:** This version of Tomcat WebSocket has vulnerabilities related to WebSocket handling, potentially allowing attackers to intercept WebSocket communications.
* **Recommendation:** Upgrade Tomcat WebSocket to the latest secure version.

These findings show a significant number of vulnerabilities in critical libraries. Updating these dependencies to their latest secure versions is essential to protect Artemis Financial’s software application from potential exploits.

**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.

Based on the review and the static test results, here’s a simple plan to fix the security problems we found:

**1. Update Outdated Libraries**

* **Problem:** Some of the libraries the software uses are old, like **Bouncy Castle**, **Hibernate Validator**, **Jackson Databind**, and **Spring Framework**, and they have known security issues.
* **Solution:**
  + Update all of these libraries to their newest versions:
    - Upgrade **Bouncy Castle** to a newer version to fix cryptography (encryption) issues.
    - Upgrade **Hibernate Validator** and **Jackson Databind** to fix problems that could allow hackers to mess with data.
    - Update **Spring Boot**, **Spring Core**, and **Spring Web** to the latest versions to protect against critical vulnerabilities.

**2. Add Input Validation**

* **Problem:** There is no input validation, meaning the software doesn’t check if the data users enter is safe. This could allow hackers to enter harmful code.
* **Solution:**
  + Add validation to make sure that all user input is clean and safe before it’s processed, especially in the **CRUDController.java** file.
  + Use secure methods to communicate with the database to prevent harmful code injections.

**3. Encrypt Sensitive Data**

* **Problem:** Important data, like customer financial details, isn’t encrypted, meaning anyone could read it if they got access to the system.
* **Solution:**
  + Use AES-256 encryption to protect all sensitive information, especially personal and financial data.
  + Store encryption keys securely, in a special, safe place that only authorized systems can access.

**4. Implement Role-Based Access Control (RBAC)**

* **Problem:** Any user can perform sensitive actions in the system, as there are no restrictions based on their role.
* **Solution:**
  + Set up role-based access control (RBAC) so that only authorized users can access certain features or data. For example, regular employees shouldn’t have access to critical financial operations.

**5. Prevent Cross-Site Scripting (XSS)**

* **Problem:** User inputs are being displayed back to users without being checked, which could allow hackers to inject scripts into the system.
* **Solution:**
  + Make sure to sanitize and filter user inputs before showing them on the web interface. This will stop hackers from running dangerous scripts in the browser.

**6. Add Session Management**

* **Problem:** User sessions don’t expire, meaning a user can stay logged in forever, which increases the risk of session hijacking.
* **Solution:**
  + Implement session timeouts, meaning users will be logged out automatically after a period of inactivity (like 15 minutes).
  + Use secure cookie settings (like HttpOnly and Secure) to prevent hackers from stealing session data.

**7. Secure API Endpoints**

* **Problem:** The API (the system that allows different parts of the software to talk to each other) is not well protected, making it easy for hackers to misuse.
* **Solution:**
  + Use OAuth2 (a strong authentication method) to secure the API, so only trusted users and systems can access it.
  + Use HTTPS to protect communication between clients and the server to prevent data from being intercepted.

**8. Improve Error Handling and Logging**

* **Problem:** The system displays too much information in error messages, which could help hackers learn about the system’s internal structure.
* **Solution:**
  + Hide detailed error information from users and only show simple, generic messages.
  + Log detailed errors on the server for developers to check but don’t show these to users.

**9. Strengthen Cryptography**

* **Problem:** The cryptographic system (how data is secured) uses an outdated library, which may have weak encryption methods.
* **Solution:**
  + Update to the latest version of the Bouncy Castle cryptography library and make sure it uses strong encryption methods like AES-256 and RSA-2048.

**10. Run Dependency Checks Regularly**

* **Problem:** Some of the libraries and dependencies are outdated, but the OWASP dependency-check plugin isn’t set up to run all the time.
* **Solution:**
  + Set the OWASP dependency-check plugin to run during the build process so any vulnerabilities are detected early.
  + Regularly run this check to make sure all libraries stay updated and secure.

**Summary:**

By following this plan, Artemis Financial can reduce the risk of being hacked and better protect its sensitive financial data. Keeping the software updated and running regular security checks will help keep the system safe in the future.