# CS 305 Project One

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **23 January 2025** | **Tony McCormack** |  |

## 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

Tony McCormack

**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 dealing in customized financial plans; hence, security in communications is vital to protect sensitive information about customers' savings, investments, and insurance plans. This need underlines the importance of maintaining customer trust and adherence to regulatory standards. Besides, Artemis Financial may conduct international transactions that require compliance with global regulations like GDPR and PCI DSS. Besides, there are governmental restrictions that must be considered in order not to incur fines for not following such data protection regulations as HIPAA and SOX by the U.S.

Regarding external threats, the most immediate risks include phishing attacks, SQL injection, cross-site scripting (XSS), and insecure API endpoints. Future risks may come in the form of zero-day vulnerabilities, APTs, and supply chain attacks. Modernization will be required to address these threats. This would involve leveraging secure and updated open-source libraries, among others, implementing evolving technologies like OAuth 2.0 for authentication and TLS 1.3 for encrypted communication. By looking at, and addressing these needs and threats, Artemis Financial can achieve a secure and modernized software environment.

**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.

The vulnerability scan indicates several security issues to Artemis Financial's software application. Input validation is critical to stop attacks like SQL injection and XSS by sanitizing user input on web forms and API calls. Authentication and authorization are also critical since enforcing role-based access control (RBAC) allows sensitive endpoints to be accessed by authorized users only. Secure settings must be ensured to harden the system settings, disable unused services, and enforce suitable HTTP security headers.

Data protection is another critical area that necessitates encryption of data in transit with TLS 1.3 and data at rest using AES-256 encryption. Error handling must be addressed to make error messages generic and not expose sensitive system information. Dependency management is necessary because out-of-date libraries are a frequent attack vector; keeping dependencies current will prevent known vulnerabilities. Last, logging and monitoring are critical to detection and response to abnormal activity, as an early indication of a potential compromise. These security factors in combination are the cornerstone of a sound defense for Artemis Financial's software application.

**3. Manual Review**

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

Most of the security vulnerabilities were identified during the manual review of the Java code base within CRUD.java, CRUDController.java, customer.java, DocData.java, Greeting.java, GreetingController.java, myDateTime.java, and RestServiceApplication.java. Major issues found include insecure data handling, SQL injection, authentication and authorization, XSS, too much exposure of APIs, and lack of logging/error handling.

Customer.java and DocData.java store sensitive information in plaintext, without encrypting it. This can easily be accessed by unauthorized users. Also, poor input validation in DocData.java may further cause data corruption and injection attacks. Encryption of information, such as AES-256, is recommended along with strong input validation for data sanitization and strict format enforcement.

The major issue in CRUD.java and CRUDController.java is direct concatenation in SQL query composition, which exposes the system to SQL injection attacks. Besides that, not using prepared statements raises the chances of malicious queries being executed. Solution: use parameterized queries or some ORM frameworks like Hibernate to avoid SQL injection attacks. Also, the input data should be sanitized before passing into database queries.

Weaknesses in authentication and authorization have been identified in RestServiceApplication.java and CRUDController.java, whereby critical endpoints are exposed without authentication, enabling unauthorized users to execute CRUD operations. Besides, RestServiceApplication.java may also be vulnerable due to the absence of RBAC, which may enable users to perform actions beyond their privilege levels. These need to be fixed by implementing an authentication mechanism like JWT or OAuth 2.0, among others, along with the implementation of RBAC policies that will ensure restriction of access based on user roles.

Cross-site scripting XSS vulnerabilities were present in both GreetingController.java and Greeting.java, where the user input is rendered directly without sanitization. This would expose the application to reflected and stored XSS attacks. Such threats can be prevented by first escaping user-generated content for display and secondly using the OWASP Java Encoder or Apache Commons Text library for input sanitization.

In RestServiceApplication.java and CRUDController.java, sensitive exposure of excessive API data was found. These return whole records from the database, making API responses a high risk for exposing sensitive information. Also, there is no limit to safeguarding the REST API from brute-force attacks. The API responses need to be limited to necessary fields. Rate limiting through an API gateway or any middleware, like the Spring Boot Rate Limiter, should be introduced.

Poor logging and error handling can be found in myDateTime.java and CRUDController.java, whereby very verbose error messages reveal the system stack trace to show internal implementation. Lack of centralized logging complicates security monitoring. This could be fixed by replacing detailed error messages with customized generic responses and doing centralized logging with security-focused frameworks such as Logback or ELK Stack.

Overall, the Java code base review has identified important security vulnerabilities, including SQL injection risks, authentication flaws, XSS vulnerabilities, excessive data exposure, and insufficient input validation. All these issues need to be fixed to keep the application secure. Authentication, role-based access control, prepared statements against SQL injections, input sanitizing against XSS, API response restrictions, enhanced logging, and error handling are immediate steps. The integration of secure coding best practices, frequent vulnerability assessment, and automated security scanning would go a long way in hardening the application from various potential cyber threats.

**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 report for Artemis Financials’ software application has identified several vulnerabilities, with multiple dependencies flagged for security risks. Notably, critical vulnerabilities were found in **Spring Boot, Spring Framework, and Apache Tomcat**, which could expose the application to remote code execution, authentication bypass, and data leakage. Additionally, the **Bouncy Castle Crypto Library** was found to have multiple security weaknesses that could lead to cryptographic attacks. **Logback and Jackson Databind** were also flagged for vulnerabilities related to log injection and insecure deserialization, respectively. These issues pose a significant risk to the application's security, making it imperative to upgrade the affected dependencies to their latest secure versions. The following table outlines the key vulnerabilities, their severity levels, and recommended actions.

|  |  |  |  |
| --- | --- | --- | --- |
| Library | Current Version | Vulnerability | Recommended Solution |
| |  |  |  | | --- | --- | --- | | |  | | --- | | Bouncy Castle (bcprov-jdk15on) |  |  | | --- | |  | |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | 1.70 |  |  | | --- | |  | |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | Cryptographic weaknesses exposing encryption vulnerabilities (5 CVEs) |  |  | | --- | |  | |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | Upgrade to the latest secure version |  |  | | --- | |  | |  |  | | --- | |  | |
| |  |  |  | | --- | --- | --- | | |  | | --- | | Jackson Databind |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | 2.15.0 |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | Deserialization vulnerabilities leading to remote code execution (1 CVE) |  |  | | --- | |  | |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | Update to the latest version |  |  | | --- | |  | |  |  | | --- | |  | |
| |  |  |  | | --- | --- | --- | | |  | | --- | | Logback Classic |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | 1.2.11 |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | Log injection vulnerabilities leading to potential code execution (1 CVE) |  |  | | --- | |  | |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | Upgrade Logback to a newer, secure version |  |  | | --- | |  | |  |  | | --- | |  | |
| |  |  |  | | --- | --- | --- | | |  | | --- | | Logback Core |  |  | | --- | |  | |  |  | | --- | |  | | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | |  | | --- | | 1.2.11 |  |  | | --- | |  | |  |  | | --- | |  | |  |  | | --- | |  | |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | Logging vulnerabilities affecting security (3 CVEs) |  |  | | --- | |  | |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | Update to the latest version |  |  | | --- | |  | |  |  | | --- | |  | |
| |  |  |  | | --- | --- | --- | | |  | | --- | | Spring Boot |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | 2.7.8 |  |  | | --- | |  | | |  |  |  | | --- | --- | --- | | |  | | --- | | Authentication bypass and data exposure risks (3 CVEs) |  |  | | --- | |  | |  |  | | --- | |  | | |  | | --- | | Upgrade Spring Boot to the latest version |  |  | | --- | |  | |

**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.

The security assessment of Artemis Financial’s software application identified multiple vulnerabilities across critical dependencies, particularly in cryptographic libraries, logging frameworks, and web application components. These vulnerabilities pose significant risks, including remote code execution, authentication bypass, and data exposure. To mitigate these threats effectively, the following steps will be taken:

**1. Upgrade Vulnerable Dependencies**

* **Bouncy Castle (bcprov-jdk15on 1.70)**: Identified cryptographic weaknesses. Upgrade to the latest stable version to ensure secure encryption practices.
* **Jackson Databind (2.15.0)**: Contains deserialization vulnerabilities that could allow remote code execution. Upgrade to the latest patched version.
* **Logback Classic & Core (1.2.11)**: Known log injection vulnerabilities. Upgrade to a newer secure version to prevent unauthorized access.
* **Spring Framework (Spring Boot 2.7.8, Spring Web 5.3.25, Spring WebMVC 5.3.25)**: High-risk vulnerabilities, including authentication bypass and remote code execution. Upgrade to the latest Spring Boot and Spring Framework releases.
* **Tomcat Components (Embed Core 9.0.73, WebSocket 9.0.71)**: Contains multiple critical vulnerabilities. Upgrade to a secured Tomcat version to mitigate risks.

**2. Apply Security Patches and Configuration Hardening**

* Implement **patch management policies** to ensure timely application of security updates for all dependencies.
* Harden **Spring Security** configurations by enforcing stricter authentication and authorization mechanisms.
* Enable **secure logging practices** to prevent log injection attacks.

**3. Implement Secure Coding Practices**

* Enforce **input validation and sanitization** to mitigate deserialization and injection vulnerabilities.
* Apply **least privilege principles** in authentication and authorization mechanisms.
* Enable **Content Security Policy (CSP)** and **Secure HTTP Headers** for better web security.

**4. Conduct Regular Security Testing**

* Perform **automated dependency scanning** and **manual code reviews** regularly.
* Implement **dynamic application security testing (DAST)** to detect vulnerabilities in runtime.
* Conduct **penetration testing** to simulate real-world attacks and assess system resilience.

**5. Strengthen Cryptographic Practices**

* Ensure **TLS 1.2 or higher** is enforced for secure communications.
* Replace outdated or weak cryptographic algorithms with **NIST-approved standards**.
* Regularly rotate and manage cryptographic keys using **secure key management practices**.

**6. Monitor and Respond to Threats**

* Integrate **Security Information and Event Management (SIEM)** tools to monitor for malicious activities.
* Establish **incident response protocols** to handle potential exploits efficiently.
* Provide **security awareness training** for developers to recognize and mitigate security threats proactively.

By implementing these mitigation steps, Artemis Financial can significantly reduce its exposure to external threats and strengthen its application security posture. Regular audits and proactive security updates will ensure the long-term integrity and resilience of the software.