

**Artemis Financial Vulnerability Assessment Report**

Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| **1.0** | **9/16/23** | **colin aheron** |  |

Client



Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In the report, identify your findings of security vulnerabilities and provide recommendations for 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 choose to include images or supporting materials. If you include them, make certain to insert them in all 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

Colin Aheron

Client: Artemis Financial

Web-Based Software Application

Interpreting Client Needs

Artemis Financial requires robust security measures to protect their web-based software application. Key considerations include:

Value of Secure Communications

Secure communications are crucial for Artemis Financial due to the sensitive financial information they handle. Implementing secure communication protocols, such as HTTPS/TLS, ensures the confidentiality and integrity of data transmitted between clients and the application, safeguarding against eavesdropping and data tampering.

International Transactions

Artemis Financial engages in international transactions, making it essential to consider the security requirements imposed by different jurisdictions. Adherence to governmental restrictions and data protection regulations ensures compliance and protects customer data during cross-border transactions.

Artemis Financial faces various external threats, including:

- Unauthorized Access: Malicious individuals attempting to gain unauthorized access to sensitive financial data, compromising client confidentiality and integrity.

- Data Breaches: The theft or exposure of confidential customer information, such as personal details and financial accounts, leading to financial loss and reputational damage.

- Denial of Service (DoS) Attacks: Malicious actors overwhelming the application with excessive traffic or requests, resulting in service disruption and potential financial losses.

- Phishing Attacks: Attempts to deceive users into revealing sensitive information, such as login credentials or financial data, through fraudulent emails or websites.

- Malware and Ransomware: Installation of malicious software or encryption of critical data for ransom purposes, disrupting operations and causing financial harm.

- Man-in-the-Middle Attacks: Interception and modification of communication between the application and clients, enabling attackers to eavesdrop, alter, or steal sensitive information.

Areas of Security

Input validation is relevant because there is a lack of input validation in the CRUDController class where the name parameter from the URL query is used without validation. This vulnerability could potentially lead to injection attacks.

APIs are relevant because the code includes a REST API endpoint (CRUDController) that interacts with external clients by accepting input parameters and returning data. Vulnerabilities in API design or security can impact the overall security of the application.

Cryptography is relevant because there is a lack of data encryption for sensitive information transmitted over the network (Data Encryption vulnerability). Proper cryptography practices should be implemented to secure data in transit and at rest.

Code errors are relevant because the code base contains several issues, including SQL injection vulnerabilities, lack of exception handling, and hardcoded credentials. These code errors can lead to security vulnerabilities.

Code quality is relevant because the code base has multiple issues related to security, maintainability, and best practices. Addressing these code quality concerns is essential for improving overall application security.

Manual Review

In the DocData class, the read\_document method attempts to connect to a MySQL database but lacks proper SQL query handling. This code is susceptible to SQL injection attacks since it doesn't use prepared statements or input validation. Also in the DocData class, the database connection string includes the root username and password in plain text. Storing database credentials in source code is a security risk; they should be stored securely, such as in environment variables or a configuration file.

In the CRUDController class, the code directly uses the name parameter from the URL query without input validation. This can lead to potential injection attacks or unexpected behavior if the input is not sanitized.

Static Testing



75 vulnerabilities were found in the maven dependency check report.

The vulnerable dependencies are

bcprov-jdk15on-1.46.jar

The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.5 to JDK 1.7.

CVEs -

<https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Abouncycastle&cpe_product=cpe%3A%2F%3Abouncycastle%3Alegion-of-the-bouncy-castle-java-crytography-api&cpe_version=cpe%3A%2F%3Abouncycastle%3Alegion-of-the-bouncy-castle-java-crytography-api%3A1.46>

spring-boot-2.2.4.RELEASE.jar

Spring Boot

CVEs -

<https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Avmware&cpe_product=cpe%3A%2F%3Avmware%3Aspring_boot&cpe_version=cpe%3A%2F%3Avmware%3Aspring_boot%3A2.2.4>

logback-core-1.2.3.jar

logback-core module

CVEs -

<https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aqos&cpe_product=cpe%3A%2F%3Aqos%3Alogback&cpe_version=cpe%3A%2F%3Aqos%3Alogback%3A1.2.3>

log4j-api-2.12.1.jar

The Apache Log4j API

CVEs -

<https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aapache&cpe_product=cpe%3A%2F%3Aapache%3Alog4j&cpe_version=cpe%3A%2F%3Aapache%3Alog4j%3A2.12.1>

snakeyaml-1.25.jar

YAML 1.1 parser and emitter for Java

CVEs - <https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Asnakeyaml_project&cpe_product=cpe%3A%2F%3Asnakeyaml_project%3Asnakeyaml&cpe_version=cpe%3A%2F%3Asnakeyaml_project%3Asnakeyaml%3A1.25>

CVE-2022-1471 SnakeYaml's Constructor() class does not restrict types which can be instantiated during deserialization. Deserializing yaml content provided by an attacker can lead to remote code execution. We recommend using SnakeYaml's SafeConsturctor when parsing untrusted content to restrict deserialization.

Mitigation Plan

Based on the vulnerability assessment, the following mitigation plan is recommended:

Best Practices

- Implement industry-standard best practices for secure software development, including secure coding practices, input validation, output sanitization, and secure authentication and authorization mechanisms.

Security Updates and Patches

- Regularly update the web application, operating systems, and third-party libraries or components to address known vulnerabilities and reduce the risk of exploitation.

Encryption and Data Protection

- Implement encryption for sensitive data at rest and during transit, using strong encryption algorithms and proper key management practices.

Secure Configuration

- Apply secure configurations to servers, databases, firewalls, and other network components, including the use of secure communication protocols and appropriate access controls.

Regular Security Assessments

- Conduct periodic security assessments, including penetration testing and vulnerability scanning, to identify and remediate any new vulnerabilities that may arise over time.