**CS 305 Project One Template**

Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| **1.0** | **9/18/2024** | **Lyric Hart** |  |

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

Lyric Hart

**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?

**Value of Secure Communications:** Artemis Financial deals with sensitive financial information, so secure communications are essential to safeguard client data, financial plans, and transactions from potential breaches.

**International Transactions:** If Artemis were to do business internationally, there must be secure communications to abide by global data protection policies and standards. This will help manage credit card transactions.

**Governmental Restrictions:** If Artemis works with U.S. government entities or international clients, it must comply with regulations such as the Gramm-Leech-Bliley Act (GLBA) and possibly the International Traffic in Arms Regulation (ITAR).

**External Threats:** There can be a range of potential threats that may affect Artemis during its business dealings with clients either nationally or globally. Such attacks can be data breaches, phishing, financial fraud, man-in-the-middle attacks (MITM), etc. However, in the future, these threats may evolve into advanced espionage or other cyber malware.

**Modernization Requirements:** As Artemis modernizes its software keep in mind the requirements that it should have to keep it up to date. The first should be to evaluate open-source libraries which could lead to unsecured vulnerabilities. The second would be to set up proper frameworks so that there is a structured foundation within the software. Popular APIs like React, Angular, or anything cloud-based will help to bring it to completion.

**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 step involves analyzing the overall application architecture to locate vulnerabilities early on. It's the foundation that helps determine the areas needed for deeper review.

**Input Validation:** This ensures that all data input, like user submissions, is properly verified and validated to avoid attacks such as SQL injections or cross-scripting (XSS). Especially, for Artemis Financial, this is crucial for data sensitivity and security.

**APIs:** Having secure API applications is vital in web applications. If Artemis wants to integrate third parties or external services, then they need a proper API framework. Proper authentication, authorization, encryption, and many other methods are important for them to contribute to this phase.

**Cryptography:** This focuses entirely on the stage of data encryption. Especially, if Artemis works with a multitude of clients, they need to heavily encrypt all customer financial data, so that it is not compromised. A misuse of cryptography could lead to multiple data breaches.

**Client/Server:** This deals with secured computing that is distributed throughout servers. As mentioned earlier, attacks like man-in-the-middle (MITM), or even session handling can be a huge risk. It's better to have Artemis utilize cloud servers for their financial dealings.

**Code Error:** Secure code handling is critical to prevent exposing sensitive information, that could aid cyber-attackers in exploiting vulnerabilities.

**Code Quality:** This promotes secure coding practices and patterns for developers. The Artemis codebase needs to follow proper coding measures so that nothing is out of place.

**Encapsulation:** Focuses on securing data structures and ensuring that internal data is not exposed to external entities.

**3. Manual Review**

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

**1. Lack of Input Validation (CRUDController.java, GreetingController.java)**

**Description:** User input received via @RequestParam in both files is not validated. This can make the application susceptible to attacks, that being cross-scripting (XSS), SQL injections, and other malicious attacks.

**Recommendation:** Implement proper input validation tactics, by the use of frameworks. Examples of these can be Spring's @valid annotation or manually validated parameters to ensure safety. The use of libraries like JSoup can help to sanitize user inputs when needed.

**2. SQL Injection Vulnerability (DocData.java)**

**Description:** In the 'read\_document' method, SQL queries are constructed without any prepared statements or parametrized queries. This makes the application vulnerable to SQL injections.

**Recommendation:** Use prepared statements or an ORM to avoid SQL injection future attacks. Prepared statements help to handle user inputs before they are executed within the system.   
  
**3. Improper Error Handling (DocData.java)   
 Description:** The error handling for database connection failures is incomplete, as shown in the 'read\_document' method. While the application does detect 'SQL injection', it only prints the stack trace which may expose sensitive information.

**Recommendation:** To fix this, simply start by logging errors to a secure location and provide generic-user-friendly messages that don't reveal information about the internal system.

**4. Unprotected Financial Data (customer.java)  
 Description:** The customer class handles customer financial data that is sensitive. This includes things such as account numbers and balances but without any sort of secure encryption. This makes the data susceptible to tampering or invasion of privacy.   
  
 **Recommendation:** The best suggestion would be to implement encryption algorithms so that no customer financial data is at risk of being exposed to the online web. Such controls like role-based access controls (RBAC) help to prevent unauthorized access.   
  
**5. Potential Logic Flaws in Deposit Handling (customer.java)  
 Description:** The deposit method allows for direct modification of the 'account\_balance', without validation. This can lead to logic flaws in overdrawing accounts or adding negative values.   
  
 **Recommendation:** Implement validation checks on deposit amounts to prevent negative deposits and ensure accurate financial transactions. Also, enforce logging for auditing purposes.   
  
**6. Missing Security Headers (RestServiceApplication.java)  
 Description:** The application for Artemis does not enforce the use of HTTPS or implement common security security headers. This won't be able to protect the application from attacks such as MITM or clickjacking.   
  
 **Recommendation:** The proper method would be to enforce HTTPS for all communications and configure common security headers using Spring security to protect against these vulnerabilities.   
  
**7. Absence of Authentication and Authorization (General)  
 Description:** There are no checks for authentication and authorization in any of the controllers. This means that anyone can access the endpoints within Artemis without verifying their identity or permissions.   
  
 **Recommendations:** Simply, implement Spring security or a similar framework to ensure only authorized users can access sensitive endpoints.

**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



**Dependency One Code:** CVE-2023-33202  
**Description:** This class parses OpenSSL PEM encoded streams containing X.509 certificates, PKCS8 encoded keys, and PKCS7 objects. Parsing a file that has crafted ASN.1 data through the PEMParser causes an OutOfMemoryError, which can enable a denial of service attack. (For users of the FIPS Java API: BC-FJA 1.0.2.3 and earlier are affected; BC-FJA 1.0.2.4 is fixed.)  
  
**Solution:** Updating or filtering PEM requests with EXTERNAL tagged encodings will prevent the attack. Although the problem initially appeared with PEM parsing, it finally turned out to be a consequence of a technique in the ASN.1 SET class, which has been resolved in the change that follows: <https://github.com/bcgit/bc-java/commit/0c576892862ed41894f49a8f639112e8d66d229c> (Hook, 2023)  
  
**Dependency Two Code:** CVE-2020-10693  
**Description:** A flaw was found in Hibernate Validator version 6.1.2.Final. A bug in the message interpolation processor enables invalid EL expressions to be evaluated as if they were valid. This flaw allows attackers to bypass input sanitation (escaping, stripping) controls that developers may have put in place when handling user-controlled data in error messages.  
  
**Solution:** Mitigation:

You can pass user input as an expression variable by unwrapping the context to HibernateConstraintValidatorContext. Please refer to the <https://in.relation.to/2020/05/07/hibernate-validator-615-6020-released/> and <https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/#_the_code_constraintvalidatorcontext_code>. (Jongseok Won, 2020)

**Dependency Three Code:** CVE-2021-44832  
**Description:** Apache Log4j2 versions 2.0-beta7 through 2.17.0 (excluding security fix releases 2.3.2 and 2.12.4) are vulnerable to a remote code execution (RCE) attack when a configuration uses a JDBC Appender with a JNDI LDAP data source URI when an attacker has control of the target LDAP server. This issue is fixed by limiting JNDI data source names to the Java protocol in Log4j2 versions 2.17.1, 2.12.4, and 2.3.2. (Sicker, 2021)  
  
**Solution:** Fixed Bugs

LOG4J2-3293: JdbcAppender now uses JndiManager to access JNDI

resources. JNDI is only enabled when system property

log4j2.enableJndiJdbc is set to true.

LOG4J2-3290: Remove unused method.

LOG4J2-3292: ExtendedLoggerWrapper.logMessage no longer double-logs

when location is requested.

LOG4J2-3289: log4j-to-slf4j no longer re-interpolates formatted

message contents.

LOG4J2-3204: Correct SpringLookup package name in Interpolator. Thanks

to Francis-FY.

LOG4J2-3284: log4j-to-slf4j takes the provided MessageFactory into

account Thanks to Michael Vorburger.

LOG4J2-3264: Fix MapLookup to lookup MapMessage before DefaultMap

Thanks to Yanming Zhou.

LOG4J2-3274: Buffered I/O checked had inverted logic in

RollingFileAppenderBuidler. Thanks to Faisal Khan Thayub Khan.

: Fix NPE when input is null in StrSubstitutor.replace(String, Properties).

LOG4J2-3270: Lookups with no prefix only read values from the

configuration properties as expected.

LOG4J2-3256: Reduce ignored package scope of KafkaAppender. Thanks to

Lee Dongjin. (Sicker, 2021)  
  
**Dependency Four Code:** CVE-2023-6378  
**Description:** A serialization vulnerability in logback receiver component part of logback version 1.4.11 allows an attacker to mount a Denial-Of-Service attack by sending poisoned data. (Switzerland Government Common Vulnerability Program, 2023)  
  
**Solution:** Fixed vulnerability of a potential denial of service attack on a centralized logback receiver when a third party controlling a remote appender connects to said receiver and could shut down or slow down the logging of events. This problem was reported by Yakov Shafranovich, Amazon Web Services. It has been reported under the reference CVE-2023-6378 (Shafranovich, 2023).

**Dependency Five Code:** CVE-2022-1471  
**Description:** 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. We recommend upgrading to version 2.0 and beyond. (Google Inc., 2022)  
  
**Solution:** We have now fixed this issue in the new version by replacing to `jackson-dataformat-yaml`.

This issue affects Apache Submarine: from 0.7.0 before 0.8.0. Users are recommended to upgrade to version 0.8.0, which fixes this issue.

If using the version smaller than 0.8.0 and not want to upgrade, you can try cherry-pick PR <https://github.com/apache/submarine/pull/1054> and rebuild the submart-server image to fix this. (Chen, 2023) and GHSL team member Jorge Rosillo  
  
**Dependency Six Code:** CVE-2023-20883   
**Description:** In Spring Boot versions 3.0.0 - 3.0.6, 2.7.0 - 2.7.11, 2.6.0 - 2.6.14, 2.5.0 - 2.5.14 and older unsupported versions, there is potential for a denial-of-service (DoS) attack if Spring MVC is used together with a reverse proxy cache. (VMware, 2023)  
  
**Solution:** Mitigation

Users of affected versions should apply the following mitigations:

3.0.x users should upgrade to 3.0.7+

2.7.x users should upgrade to 2.7.12+

2.6.x users should upgrade to 2.6.15+

2.5.x users should upgrade to 2.5.15+

Users of older, unsupported versions should upgrade to 3.0.7+ or 2.7.12+.

Workarounds: configure the reverse proxy not to cache 404 responses and/or not to cache responses to requests to the root (/) of the application. (Smedshammer, 2023)  
  
**Dependency Seven:** CVE-2023-46589   
**Description:** Improper Input Validation vulnerability in Apache Tomcat.Tomcat from 11.0.0-M1 through 11.0.0-M10, from 10.1.0-M1 through 10.1.15, from 9.0.0-M1 through 9.0.82, and from 8.5.0 through 8.5.95 did not correctly parse HTTP trailer headers. A trailer header that exceeded the header size limit could cause Tomcat to treat a single request as multiple requests leading to the possibility of request smuggling when behind a reverse proxy. Users are recommended to upgrade to version 11.0.0-M11 onwards, 10.1.16 onwards, 9.0.83 onwards or 8.5.96 onwards, which fixes the issue. (Apache Software Foundation, 2023)  
  
**Solution:** Users are recommended to upgrade to version 11.0.0-M11 onwards, 10.1.16

onwards, 9.0.83 onwards or 8.5.96 onwards, which fixes the issue. (**Thomas, 2023) and (Aimoto, 2023)**

**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.

* **Lack of Input Validation:**   
   Using Spring's @valid and annotation and validation APIs to enforce strict validation on all input fields, especially those received via request parameters. Also, implement input sanitization like JSoup to prevent cross-scripting (XSS). Artemis needs to make sure that all input data is properly formatted for financial purposes, and cleaned before processing.
* **SQL Injection Vulnerability:**  
  It would be better to refactor the 'read\_document' method in 'DocData.java' to utilize prepared statements or parametrized queries. This shields the Artemis application against SQL injections, by ensuring that all user inputs are parametrized before being processed on Artemis' servers.
* **Improper Error Handling:**  
  Implement a global error handling system using Spring's @ControllerAdvice and @ExceptionHandler exceptions to properly handle all exceptions. All errors that occur in Artemis need to be logged in a secure location, and user-friendly error messages need to be generic and not reveal any internal system details.
* **Unprotected Financial Data:**   
  Financial data such as account balances and account numbers must be encrypted. This can be done usually by AES-256 to have it stored in Artemis' database. In terms of transmission, this can be completed by the use of HTTPS. For only authorized personnel to access this type of data, there needs to be an implementation of role-based access control (RBAC).
* **Potential Logic Flaws in Deposit Handling:**  
  The introduction of input validation would be a huge benefit to ensure that deposits are being processed correctly. This can help to avoid negative numbers being calculated or the overdrawing of deposits.
* **Missing Security Headers:**   
  The use of Spring security would help to install HTTPS, by using the 'requiresChannel()' method to reroute all HTTP to HTTPS. The practice of configuration of security headers can help to prevent clickjacking.
* **Absence of Authentication and Authorization:**  
  All controllers in the Artemis network need proper authentication and authorization. This can be done by Spring security to have a secure communications mechanism in place.