CS 305 Project One Template

Document Revision History

Version	Date	Author	Comments
1.0	7/13/2024	Alexander Strevel	

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
Alexander Strevel

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 firm that specializes in personalized financial planning. Artemis Financial is seeking to modernize its operations with enhanced security measures, particularly for its RESTful web application. The value of secure communications for Artemis cannot be overstated. This ensures the confidentiality and integrity of sensitive financial data, which is not only essential for maintaining the trust of the client, but also for complying with both domestic and international financial regulations. Considering the possibilities for international transactions, Artemis Financial must prioritize robust encryption protocols and secure data transmission methods to protect against data breaches, phishing, ransomware, and DDoS attacks. Modernization will also involve evaluating the use of open-source libraries to ensure they do not introduce vulnerabilities. Furthermore, compliance with governmental restrictions on secure communications must be managed with great detail, to avoid any legal issues. This approach to security will be important in minimizing any kind of external threats and ensuring the success of Artemis Financial's modernization efforts.

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.

In the vulnerability assessment process flow of Artemis Financial's software application, these are some significant areas:

Input Validation:

- This is crucial due to the fact that financial applications handle sensitive data input by users, such as personal and financial information. Validating input can prevent SQL injection, cross-site scripting (XSS), and other injection attacks.

APIs (Secure API Interactions):

- Since Artemis Financial uses a RESTful API, ensuring secure API interactions is important. To ensure security, this would include implementing authentication, authorization, and data validation to prevent unauthorized access and data breaches.

Cryptography (Encryption Use and Vulnerabilities):

- Utilizing strong encryption for data at rest and in transit is essential to protect sensitive financial data against data theft.

Client/Server (Secure Distributed Composing):

- Ensuring the security of both client and server components in distributed systems helps minimize risks associated with data exposure and manipulation through man-in-the-middle attacks.

Code Review:

- Regular code reviews across different components can help identify and resolve security vulnerabilities, logic errors, or any other issues that could compromise the application.

Code Quality (Secure Coding Practices/Patterns):

- Following secure coding standards and best practices significantly reduces the risk of introducing security flaws during development. This includes practices like using prepared statements in database access to avoid SQL injection.

Encapsulation:

- Proper encapsulation in software design ensures that not only data structures, but operators are safe from unintended use, which can prevent sensitive information from being leaked along with ensuring data integrity.

Overall, these cover many types of security concerns that Artemis Financial needs to think about, making sure they are safe from outside threats. Each area has importance because of the kind of data handled by financial institutions and rules controlling it. Making strong security in these areas will assist Artemis Financial to update its operations, all while keeping a top level of security.

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, here are my findings from my manually inspection:

GreetingController.java:

- Hard-coded credentials were found in the database connection strings.
- Lack of input validation for the inputs received via GET and POST requests.

myDateTime.java:

- Use of outdated libraries that do not support the latest security practices.
- The system time is being exposed directly, which might lead to information leakage.

RestServiceApplication.java:

- Missing exception handling which can lead to unhandled errors possibly exposing sensitive information to the end-user.

CRUD.java:

- SQL queries are constructed using string concatenation, which makes them susceptible to SQL injection attacks.
- Sensitive data like user IDs are logged without masking.

DocData.java:

- Sensitive information is not encrypted before it's saved to the database.

customer.java:

- Customer's passwords are being stored in plaintext in the database.
- Email validation is missing, which can lead to spam or malicious attacks.

CRUDController.java:

- No rate limiting on endpoints, which makes them vulnerable to brute-force attacks.
- API endpoints expose detailed error messages that could aid an attacker in crafting further attacks.

Greeting.java:

- Insecure serialization processes that could lead to remote code execution if exploited.

These findings from my manual code review would suggest that there are many areas in the code that need immediate attention to improve security. As mentioned before these areas would include, input validation, secure handling of sensitive data, and the use of modern cryptographic methods. Addressing these issues are important in protecting Artemis Financial's application.

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

After running the dependency checker on Artemis Financial's application, it identified multiple critical vulnerabilities in Artemis's application dependencies. Below are the following findings from the report:

1. **Vulnerability**: org.bouncycastle:bcprov-jdk15on:1.46
   Severity: HIGH
   CVE Count: 22
   Description:
   - Contains cryptographic weaknesses that could potentially be exploited to compromise secure communications.

   Recommended Solution:
   - Upgrade to the latest version that resolves these known vulnerabilities.

   Attribution:
   - Documented under various CVE identifiers due to recognized cryptographic flaws.

2. **Vulnerability**: com.fasterxml.jackson.core:jackson-databind:2.10.2

Severity: HIGH
CVE Count: 6
Description:

- Vulnerable to deserialization attacks that could allow unauthorized remote code execution.

Recommended Solution:

- Update to a more recent, secure version of Jackson Databind.

Attribution:

- Widely recognized in security advisories and CVE databases for its vulnerabilities.

3. **Vulnerability**: org.apache.logging.log4j:log4j-core:2.13
   Severity: HIGH
   CVE Count: 2
   Description:
   - Log injection could lead to remote code execution through the logging of malicious input.

   Recommended Solution:
   - Patch to the latest version addressing these security flaws.

   Attribution:
   - CVEs associated with the Log4Shell incident highlight the risks.

4. **Vulnerability**: org.springframework:spring-webmvc:5.2.3.RELEASE
   Severity: CRITICAL
   CVE Count: 14
   Description:
   - Known vulnerabilities in this version of Spring Web MVC could allow remote code execution via specially crafted requests.

   Recommended Solution:
   - Upgrade to the latest version of Spring Web MVC that addresses these vulnerabilities.

   Attribution:
   - Documented in multiple CVE records, reflecting widespread recognition of the risks associated with this vulnerability.

5. **Vulnerability**: org.springframework:spring-web:5.2.3.RELEASE

Severity: CRITICAL
CVE Count: 11
Description:
- Remote code execution vulnerabilities due to improper input validation.

Recommended Solution:
- Migrate to a secured version of Spring Web.

Attribution:
- Known through various security advisories and CVE listings for its critical security flaws.

6. **Vulnerability**: org.apache.tomcat.embed:tomcat-embed-core:9.0.30
Severity: CRITICAL
CVE Count: 26
Description:
- Multiple security flaws including buffer overflows and privilege escalation.

Recommended Solution:
- Implement the most recent version of Tomcat that resolves these vulnerabilities.

Attribution:
- Extensively documented in the National Vulnerability Database (NVD) and other security platforms.

7. **Vulnerability**: org.yaml:snakeyaml:1.23
Severity: CRITICAL
CVE Count: 8
Description:
- This component can execute code when processing malicious YAML files.

Recommended Solution:
- Upgrade to a secure version that prevents unauthorized code execution.

Attribution:
- Identified through security research and multiple documented CVEs.

8. **Vulnerability**: org.springframework:spring-framework-bom:5.2.3.RELEASE
Severity: CRITICAL
CVE Count: 11
Description:
- Multiple vulnerabilities that could lead to the application becoming compromised.

Recommended Solution:
- Update to the latest secure release of Spring Framework.

Attribution:
- Documented across multiple CVEs related to various vulnerabilities in the Spring Framework.

9. **Vulnerability**: org.springframework:spring-beans:5.2.3.RELEASE

Severity: CRITICAL
CVE Count: 11
Description:
- Issues that could allow attackers to manipulate beans leading to code execution.

Recommended Solution:
- Update to the latest version of Spring Beans that patches known security issues.

Attribution:
- Part of the broader Spring vulnerabilities documented in CVEs.

10. **Vulnerability**: org.springframework:spring-aop:5.2.3.RELEASE
Severity: CRITICAL
CVE Count: 11
Description:
- Aspect-oriented programming library vulnerabilities that could be exploited.

Recommended Solution:
- Update to a newer, secure version of Spring AOP.

Attribution:
- CVEs outlining potential exploits in Spring AOP.

11. **Vulnerability**: org.springframework:spring-context:5.2.3.RELEASE
Severity: CRITICAL
CVE Count: 11
Description:
- Vulnerabilities in the application context can lead to security breaches.

Recommended Solution:
- Secure update to the latest version of Spring Context.

Attribution:
- Recognized through various CVEs detailing the vulnerabilities.

12. **Vulnerability**: org.hibernate.validator:hibernate-validator:6.0.18.Final
Severity: MEDIUM
CVE Count: 1
Description:
- Potential for invalid data to bypass validation checks.

Recommended Solution:
- Implement the updated version of Hibernate Validator.

Attribution:
- Documented in security advisories highlighting this specific flaw.

13. **Vulnerability**: jakarta.validation:jakarta.validation-api:2.0.2
Severity: LOW

CVE Count: 0
Description:

- Minor issues that could affect data validation processes.

Recommended Solution:

- Update to the latest version to ensure compliance with newer standards.

Attribution:

- Noted in general discussions around validation standards and their implementations.

These vulnerabilities represent a significant risk, especially in a financial application where data security is incredibly important. Immediate action is required to update or replace the vulnerable components to ensure the security of the system and to protect sensitive client information. Making sure to update and patch when they become available, as recommended by the developers of these libraries, is crucial to maintaining the security of the application.

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 following steps that I would take to mitigate the identified security vulnerabilities for Artemis Financial's software application, would be the following:

Update and Patching: The first step that I would take is to update all the third-party libraries and frameworks to their latest versions. This would include critical updates for Spring components, Jackson Databind, Log4j, and Tomcat Embed. These updates will protect against known vulnerabilities that could otherwise allow remote code execution, data breaches, and other malicious activities.

Code Refactoring and Input Validation: After the manual review, I identified severe issues such as hard-coded credentials, lack of input validation, and insecure data handling practices. The code needs to be refactored to remove hard-coded credentials and replace them with secure environmental variables. Secondly, the input fields should be validated both on the client-side and server-side to prevent SQL injection and cross-site scripting attacks. This includes implementing strict type, format, and content checks.

Secure Cryptographic Practices: We should replace the outdated cryptographic library (Bouncy Castle) with the latest version to ensure encryption standards are maintained. It's essential to enforce strong encryption protocols for both data at rest and data in transit, especially since the data that Artemis is handling is financial data.

Error Handling and Logging: If we modify the application's error handling and logging mechanisms, we can prevent the disclosure of sensitive information through detailed error messages or logs. This can

be Implemented by centralized logging that is secure and monitors log files for signs of tampering attempts.

Secure Serialization Processes: We should Address the insecure serialization processes found in the following file “Greeting.java,” along with other components by implementing safe serialization practices. Wherever possible, we should use safer alternatives to Java's native serialization or employ libraries that explicitly prevent serialization gadgets from being exploited.

Enhance API Security: APIs are a critical component of Artemis Financial’s application. We need to increase the security of their API as it’s very important. This would include the following: implementing rate limiting, proper authentication mechanisms like OAuth, and ensuring API keys are not exposed in the client code.

Conduct Regular Security Audits: Schedule regular security audits and penetration testing to identify and minimize risk of compromise, due to new vulnerabilities that could emerge. This should be integrated into the software development lifecycle to ensure continuous assessment and improvement of overall security.

Developer Training and Security Awareness: Conducting regular training sessions for developers and associated staff to enhance their awareness of security best practices and emergent threats, is very important. This training should cover secure coding practices, common vulnerabilities and how to prevent them, and updates on compliance and regulatory requirements.

Establish a Security Incident Response Plan: By developing and maintaining an incident response plan that can quickly be initiated just in case there is a security breach. This plan should include procedures for containment, investigation, mitigation, and communication with stakeholders, ensuring minimal impact on Artemis’s operations and reputation.

In conclusion, by implementing these steps, Artemis can significantly enhance the security of its software application, which in turn would protect its data and systems from potential threats. This would also help Artemis to align with industry best practices in terms of cybersecurity.