****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **08/15/2024** | **Devin Burdick** |  |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the 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 Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Devin Burdick

## Algorithm Cipher

To keep Artemis Financial's client data secure, I'd recommend using AES (Advanced Encryption Standard) for encryption. It's a solid choice because it's fast, reliable, and widely trusted. AES is a symmetric key algorithm, which means the same key is used to both encrypt and decrypt data. It works with 128-bit blocks of data and lets you choose key lengths of 128, 192, or 256 bits, balancing security and performance nicely.

For added security, pairing AES with the SHA-256 hash function is a good move. This ensures data integrity by generating a unique 256-bit hash, so you can check if anything's been tampered with. Also, using random numbers in AES is important to avoid patterns in the encrypted data that could be exploited.

Since AES relies on symmetric keys, you'll need to handle key management. Carefully keeping that key safe is crucial because it's used for both locking and unlocking the data. AES took over from the older DES algorithm and has become the industry standard because it’s strong and efficient.

## Certificate Generation

Insert a screenshot below of the CER file.

Below is attached screenshot of the CER file. I used projecttwoburdick as the keypass. As you can see in the lower left corner of the screenshot where I command keytool to store it in sever.cer.

A screenshot of a computer

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

Description automatically generated

## Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.

I struggled to get the webpage to show as secure. I tried adding the certificate directly to the Trusted Root Certification Authorities on my system as well.

A screenshot of a computer

Description automatically generated

## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Functional Testing

Insert a screenshot below of the refactored code executed without errors.

A screenshot of a computer

Description automatically generated

Functional Testing led me to make a few improvements to the code. These improvements included making ‘data’ a parameter and making exception handling cleaner with an error message that is handled more securely, and no sensitive data can be exposed.

## Summary

In the process of refactoring the code, I took a deep dive to make sure the software meets strict security standards, in line with the Vulnerability Assessment Process Flow. Here's a breakdown of the key steps I followed to enhance security across the application's layers:

**Architecture Review:**

First off, I analyzed the application's architecture to spot any weak points. By understanding how the app is designed and how data flows through it, I could identify where we needed to beef up security, especially in handling sensitive data and external communications.

**Input Validation:**

Next, I tackled input validation to protect against injection attacks and other malicious inputs. By ensuring that all user inputs are properly validated, the app now has a strong defense against bad data getting through.

**APIs:**

I then focused on securing the API interactions. This involved refactoring API endpoints to make sure they only accept requests from authenticated and authorized users. I added token-based authentication and improved error handling to safeguard against common API vulnerabilities.

**Cryptography:**

A big part of the refactoring was enhancing the app’s cryptographic protocols. I reviewed the existing encryption methods and implemented AES encryption to protect sensitive data both at rest and in transit. I also set up self-signed certificates to enable HTTPS, ensuring secure client-server communication.

**Client/Server:**

To secure the data exchanges between client and server, I made sure everything is encrypted using HTTPS. This step protects against man-in-the-middle attacks, keeping data safe during transmission.

**Code Error and Quality:**

I went through the code with a fine-tooth comb to find vulnerabilities. By applying secure coding practices, I ensured that exceptions are handled properly, resources are managed effectively, and common coding mistakes that could lead to security breaches are avoided. I also checked that the code meets industry-standard secure coding guidelines.

**Encapsulation:**

I applied encapsulation techniques to secure the app’s data structures. By ensuring data is accessed only through well-defined interfaces, I minimized the risk of unauthorized access and modification.

**Code Review:**

After following the vulnerability assessment process, I conducted targeted manual code reviews. These reviews were guided by the architecture review and static testing outputs, ensuring all code met the required security standards.

**Summary of Findings:**

In the end, the refactoring process effectively addressed the vulnerabilities identified earlier. The app now follows best practices in secure coding, with multiple security layers in place to protect against potential threats. I documented the findings from the code reviews and developed a mitigation plan to keep security tight and make improvements as needed.

## Industry Standard Best Practices

In the process of refactoring the code, I made sure to follow industry-standard best practices to address known security vulnerabilities effectively. Here's a summary of the key actions taken:

**Application of Industry Standard Best Practices:**

* **Secure Data Transmission**: HTTPS and AES encryption were used to secure data in transit and at rest, protecting it from interception and unauthorized access.
* **Error Handling**: Proper error handling was implemented to avoid leaking sensitive information through error messages, ensuring that attackers can’t gather useful data from them.
* **Dependency Management**: Tools like OWASP Dependency-Check were utilized to scan third-party libraries for vulnerabilities, allowing us to patch or replace any insecure dependencies.

**Maintaining Existing Security:**

* **Code Reviews**: Regular code reviews were conducted to ensure that the refactoring process didn’t introduce new security issues and that all changes complied with secure coding standards.
* **Documentation**: I made sure to document all security-related changes, making it easier for future developers to understand and maintain the current security posture.

**Value of Industry Standard Best Practices:**

By applying these secure coding practices, we’ve added significant value to the company:

* **Data Protection**: Sensitive data is better protected against breaches and unauthorized access.
* **Regulatory Compliance**: Following these practices helps ensure compliance with industry regulations and standards.
* **Risk Mitigation**: The proactive approach reduces the risk of security incidents, which in turn lowers the potential costs associated with data breaches.
* **Futureproof**: The software is now better equipped to handle emerging security challenges, contributing to the company’s long-term success.

These steps collectively enhance the software’s security and provide a solid foundation for future development efforts.