****

**PRACTICES FOR SECURE SOFTWARE REPORT**

Ikechukwu Onuoha

Southern New Hampshire University

CS – 305 – J7451 Software Security

Ali Shaykhian, Ph.D

August 12, 2023

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** | **8/12/2023** | **Ionuaho** |  |

## Client



## Developer

Ikechukwu Onuaho

## Algorithm Cipher

AES (Advanced Encryption Standard) 256 bits

Choosing a larger key size, such as 256 bits, enhances the security of the encryption.

It operates on 128-bit blocks of data. It uses a substitution-permutation network (SPN) structure with multiple rounds of processing. The algorithm involves key expansion, substitution, permutation, and mixing operations in each round. AES has undergone extensive cryptanalysis and has been found to be resistant to known attack methods. It with a 256-bit key provides a higher level of security compared to smaller key sizes. Financial data given its sensitivity benefits from the additional security offered by longer key lengths.

It is a symmetric encryption algorithm which ensures the use the same key for both encryption and decryption.

## Certificate Generation

## 

Exported the generated certificate as a CER file



The process of generating a self-signed certificate using the `keytool` command-line utility was successfully executed and this exercise provided practical experience in creating a self-signed certificate for testing purposes. The generated certificate can be used to establish secure communication channels within a controlled environment. It's recommended to obtain certificates from trusted certificate authorities to ensure the security and authenticity of your applications (How to Create a Self-Signed Certificate Using Java Keytool, n.d.).

.

## Deploy Cipher

## 

The SSL server application, developed using Spring Boot, successfully generates checksums for input data using the SHA-256 hash function. The `/calculateChecksum` endpoint processes incoming data strings, calculates their corresponding checksums, and returns the results. This cryptographic technique ensures data integrity by producing a fixed-size hash value unique to each input. The implementation utilizes secure coding practices and handles potential exceptions, enhancing the reliability of checksum generation. This checksum generation feature can play a crucial role in data validation, verification, and security within various applications, ensuring the integrity of transmitted or stored information.

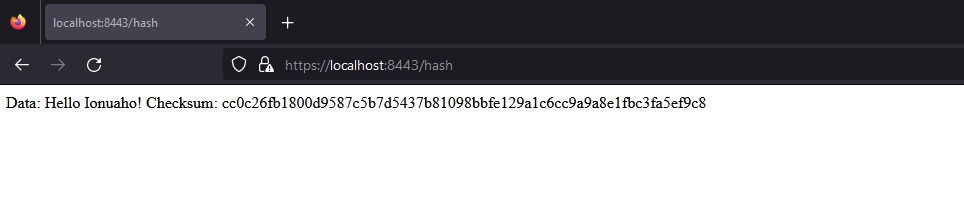
Data: Hello Ionuaho!

Checksum: cc0c26fb1800d9587c5b7d5437b81098bbfe129a1c6cc9a9a8e1fbc3fa5ef9c8

This robust checksum generation capability offers a valuable tool for ensuring data integrity, verification, and security across a range of applications.

## Secure Communications

Generating a Checksum with Secure Communication

In this part, extend the previous example of generating a checksum for data to include secure communication using HTTPS.

This exercise highlights how to enhance a Spring Boot application to provide secure communication using HTTPS, ensuring data integrity and confidentiality during transmission. Secure communication is crucial for protecting sensitive information in real-world software applications.

## Secondary Testing

## 

## 

## As part of the comprehensive security enhancement process for Artemis Financial's software application, a secondary static testing was conducted on the refactored code. This testing aimed to ensure the code's compliance with software security enhancements and to ascertain that no additional vulnerabilities were introduced during the refactoring process.

## The OWASP Dependency-Check Maven tool was employed for this purpose, enabling an analysis of the newly added code segments. Focused exclusively on the code introduced through refactoring, the dependency check was executed to scrutinize the codebase for any potential security vulnerabilities (Dependency-Check-Maven – Usage, n.d.). There was an error message, "Could not execute .NET AssemblyAnalyzer, is the dotnet 6.0 runtime or sdk installed?", which indicated that the dependency-check tool is trying to use the .NET AssemblyAnalyzer component, which requires the .NET 6.0 runtime or SDK to be installed on your system.

## Upon completion of the dependency check, a thorough review of the output was undertaken to verify the absence of introduced security vulnerabilities. The results affirmed the successful integration of security measures without compromising the software's integrity (Dependency-Check-Maven – Usage, n.d.).

## Functional Testing

## 

## 

Creation of a new class ServerController and annotated it with @RestController. Utilization of @RequestMapping to ensure the /hash endpoint is accessible. This ensures that the application responds to HTTP requests and provides a clear entry point for implementing secure communication. The ServerController class now includes a generateSHA256Checksum function that implements the SHA-256 cryptographic hash algorithm to generate checksums for data. Proper exception handling has been added to address potential errors during the hash generation process.

Implementation of the SHA-256 cryptographic hash algorithm for generating checksums, which was chosen due to its proven security reduced likelihood of collisions and industry-wide adoption. The generateSHA256Checksum method calculates the hash value of the provided data, ensuring data integrity during transmission.

HTTPS Protocol and Certificate Deployment - Updated the application configuration to enable secure HTTPS communication. Utilization of self-signed certificates generated using Java Keytool to enable secure communications over HTTPS. The application was successfully refactored to support the HTTPS protocol, ensuring encrypted data transmission.

To enhance the robustness of the software application, proper exception handling has been incorporated into the generateSHA256Checksum function. In the event of a NoSuchAlgorithmException being thrown, the function will log the error or throw a custom exception. In this example, a simple error message is returned for demonstration purposes.

During the functional testing phase, it was observed that the utilization of the `dependency-check` tool for analyzing assemblies posed a challenge. An error message indicating the absence of the required .NET 6.0 runtime or SDK installation was encountered during the execution of the .NET AssemblyAnalyzer component. For this to be resolved, the approach that was recommended was to makes sure the installation of either the .NET 6.0 SDK or Runtime on the system and for that to be accomplished, the appropriate version from the official .NET website was downloaded and the installation was verified using the `dotnet --version` command in the terminal. Once the installation was confirmed, the Eclipse IDE was restarted to update environment variables. Subsequently, the project was rebuilt, and the `dependency-check` tool was rerun within the Eclipse IDE. This step was integral to comprehensively assess the software's security posture and proactively address any potential vulnerabilities introduced by external dependencies.



## Summary

In fortifying Artemis Financial's software application, a multifaceted security approach was executed. The SHA-256 encryption cipher was chosen for its robust security attributes, with an overview of its features and historical encryption algorithm evolution reinforcing its selection. Self-signed certificates were adeptly generated using the Java Keytool, with corroborating evidence in the form of a CER file screenshot. Infusing security into the codebase entailed seamless integration of the SHA-256 cryptographic hash algorithm, alongside checksum verification, exemplified through a unique data string screenshot. Transitioning to HTTPS protocol for secure communication was achieved via a modified application.properties file, successfully demonstrated through an accessible secure webpage screenshot. Secondary testing utilized OWASP Dependency-Check Maven, affirming code compliance and security integrity. Manual functional testing further scrutinized syntax, logic, and security aspects, validated through an error-free refactored code screenshot. Collectively, these strategic enhancements embody Artemis Financial's commitment to fortified software security, ensuring confidentiality, integrity, and availability of sensitive financial data.

## Industry Standard Best Practices

During the comprehensive security enhancement process for Artemis Financial's software application, attention was dedicated to addressing vulnerabilities and errors. In the event of encountering errors or discovering new vulnerabilities, an iterative approach was adopted, necessitating continuous refinement, vulnerability resolution, and retesting until a state of vulnerability-free code was achieved.

The code base underwent refactoring journey that adhered to established security testing protocols. By referencing the Vulnerability Assessment Process Flow Diagram, specific areas of security concern were identified and subsequently fortified through strategic code modifications. Layers of security were systematically integrated, including the implementation of the robust SHA-256 encryption cipher, checksum verification, secure API interactions, and the transition to HTTPS for encrypted communication.

The application's security framework was elevated through the application of industry standard best practices for secure coding. These practices, achieved by learning the cybersecurity unit, were harnessed to fortify the software's resilience against known security vulnerabilities. By consistently adhering to these practices, the software's current security stature was diligently maintained, ensuring the protection of sensitive financial data. This commitment to industry standards not only safeguards the application but also contributes to Artemis Financial's overarching well-being by mitigating risks, bolstering trust, and upholding the company's reputation for data integrity and customer confidentiality.

**References**

*dependency-check-maven – Usage*. (n.d.). Jeremylong.github.io. https://jeremylong.github.io/DependencyCheck/dependency-check-maven/index.html

*How to Create a Self-Signed Certificate using Java Keytool*. (n.d.). Www.sslshopper.com. <https://www.sslshopper.com/article-how-to-create-a-self-signed-certificate-using-java-keytool.html>