****

# 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** | **02/21/2024** | **Binaya Rimal** | **Project-2** |

## 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

Binaya Rimal

## Algorithm Cipher

I recommend Artemis to use AES (Advanced Encryption Standard) as the encryption algorithm. AES is a symmetric encryption algorithm meaning it uses the same key to encrypt and decrypt data while the key is kept secret. Asymmetric algorithms like RSA which uses a private and a public key also has its benefits, but for Artemis AES is the recommended choice due to its high security, speed, and ability to handle large amounts of data. AES divides data into bit blocks and transforms them into cipher blocks through mathematical operations. It uses fixed key sizes which can be 128, 192 or 256 bits to encrypt data. These sizes determine the strength of the encryption, with higher number providing the most security. Using an algorithm, the original data is transformed into seemingly random values through hashing.

Financial services are specially targeted by hackers. Despite requiring high computing power and difficulty, hackers might attempt to brute force their way into Artemis’s system in hopes for financial gains. AES prevents brute force attacks because its large key sizes could include 2^128, 2^192, and 2^256 possible keys, such a size will be practically impossible to decrypt with brute force. AES is also backed by the National Institute of Standards and Technology (FIPS) and approved by the Federal Information Processing Standards (FIPS). It is chosen as the standard encryption algorithm by the United States government for protection of data.

AES is good for financial services like Artemis because it requires low computational power, allowing Artemis and its customers to communicate and perform transactions fast and securely. AES can be used for various functionalities like authentications, data encryption while rest or while on storage, financial transactions, encrypted browsing, and general file encryption. With its versatility AES is perfect for Artemis’s need of encryption in many fronts.

As computing evolves, the encryption methods of security will also need to evolve. For most cases AES-128 is good enough; however, it has been cracked by Quantum computing. I recommend Artemis to use AES-256 as it is yet to be broken. Theoretically, any encryption algorithm can be broken, but the amount of computational power it will take to do so is too high for it to be a viable option for AES-256.

## Certificate Generation

A computer screen with white text

Description automatically generated

A screenshot of a certificate

Description automatically generated

## Deploy Cipher

A screenshot of a computer

Description automatically generated

## Secure Communications

## HTTPS in browser:

A screenshot of a computer

Description automatically generated

**Security Information on browser:**

A screenshot of a computer screen

Description automatically generated

**Self signed certificate displayed on browser :**

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.

**Refactored Code without Errors:**

A screenshot of a computer

Description automatically generated

**Dependency Check before code refactoring:**

A screenshot of a computer

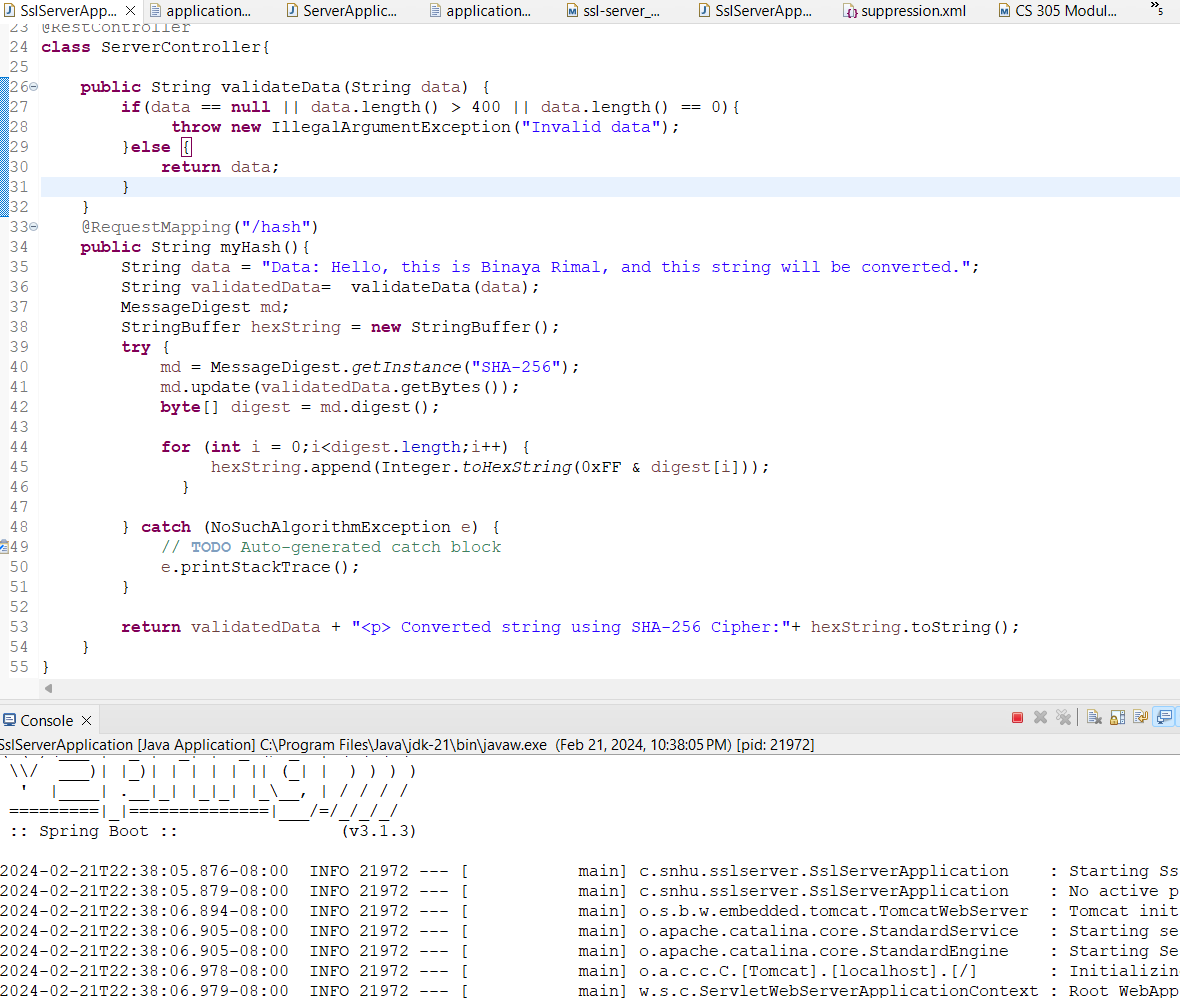
Description automatically generated

**Dependency Check after refactoring code:**

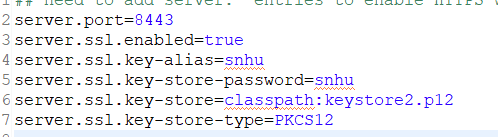


## Functional Testing

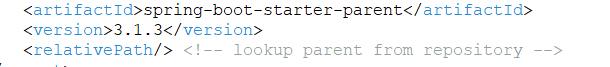
**Refactored code running without error:**



**Server Configuration:**



**Spring Boot Updated:**



**Suppressions:**

****

## Summary

I’ve added multiple layers of security on different fronts. I made sure that the API connection is secure by creating a key store to generate a self-signed certificate using java keytool which allows the connection to operate on HTTPS as opposed to an HTTP browser. Once the app is outside of the local machine and in production server, it will need to be verified from a third-party CA (Certificate Authority). The certificate initiates secure sessions with browser by the secure sockets layer (SSL) protocol.

The code uses Spring Boot for configuration which I’ve updated with the latest version. This prevents the app from running into unnecessary vulnerabilities. When updating to later versions, there is a chance of incompatibility; however, this was not the case with our app, and the program runs error free. After the update, the vulneraries significantly decreased. There were some vulnerabilities that were not relevant to our project, so I decided to suppress them. For instance, there was a vulnerability([CVE-2023-34055](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-34055)) where it was possible for a user to provide specially crafted HTTP requests that may cause a denial-of-service (DoS) condition. This only occurs on applications that uses Spring MVC or Spring WebFlux, which was not something that is used in our project.

The main functionality added through refactoring the code was the addition of a functionality to hash the static data. I used SHA-256 cryptographic hash function that takes an input data and scrambles it to produce a fixed-size hash value. SHA-256 is implemented because of its resistance to collision. A collision occurs when an entity can produce the same value as the hash value. Due to the large size of SHA-256, collision is nearly impossible, and has never occurred.

We are using static data and not user input; therefore, it wasn’t of utmost importance to validate the data as we have full control over it. However, I still implemented it for potential changes in future. The current data validation throws error when the string data is null, has length that is greater than 500, or is empty.

## Industry Standard Best Practices

Developers must follow industry standards for a secure software application. One way to do so is by running a dependency check to identify any vulnerabilities. Vulnerabilities are not always relevant so we must suppress any false positives. Many of the vulnerabilities can easily be resolved by simply updating with the latest version, which is exactly what I’ve done for this project. Not all vulnerabilities can be resolved from updating, in which case it is up to us to fix it or take the risk.

Browsers like Chrome, Firefox, and Safari will flag websites without a certification with insecure connection and notify users. To prevent this, it is standard practice to be certified through Certificate Authority. After the CA verifies your information as secure and within their guidelines, they will issue the website a digital certificate. The browser will then recognize the website as legitimate and will allow the HTTPS secure connection sign to the visitors.

Developers must perform secure communication by hashing data on transit so no hackers can access and decipher it. When storing data, it must be properly encapsulated to prevent data leakage, and data manipulation. Information like ID should never be updated so they should be set to private. All user inputs must be properly validated to prevent hackers from exploiting vulnerabilities such as SQL injection and command injection.

**Citations**

*How does SSL work?*. Entrust. (n.d.). https://entrust.com/resources/certificate-solutions/learn/how-does-ssl-work

*Key advantages and disadvantages of using Spring Boot*. SCAND. (n.d.). https://scand.com/company/blog/pros-and-cons-of-using-spring-boot/

*Keytool - key and certificate management tool*. keytool-Key and Certificate Management Tool. (n.d.). https://docs.oracle.com/javase//7/docs/technotes/tools/windows/keytool.html

N-able. (2021, April 1). *SHA-256 algorithm - N-able*. N. https://www.n-able.com/blog/sha-256-encryption

*Secure coding practice guidelines*. Secure Coding Practice Guidelines | Information Security Office. (n.d.). https://security.berkeley.edu/secure-coding-practice-guidelines

What is SHA 256 Algorithm? Functions & Applications | upGrad blog. (n.d.). *What is SHA 256 algorithm? Functions & Applications*. upGrad blog. https://www.upgrad.com/blog/sha-256-algorithm/