****

# 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** | **Feb 19, 2023** | **Eric Florence** |  |

## 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

Eric Florence

## Algorithm Cipher

One commonly used encryption algorithm for SSL/TLS is Advanced Encryption Standard (AES), which is a symmetric key encryption algorithm. AES is widely considered to be secure and is recommended by the National Institute of Standards and Technology (NIST). AES uses block ciphers with key lengths of 128, 192, or 256 bits, which are considered to be strong enough for most applications.

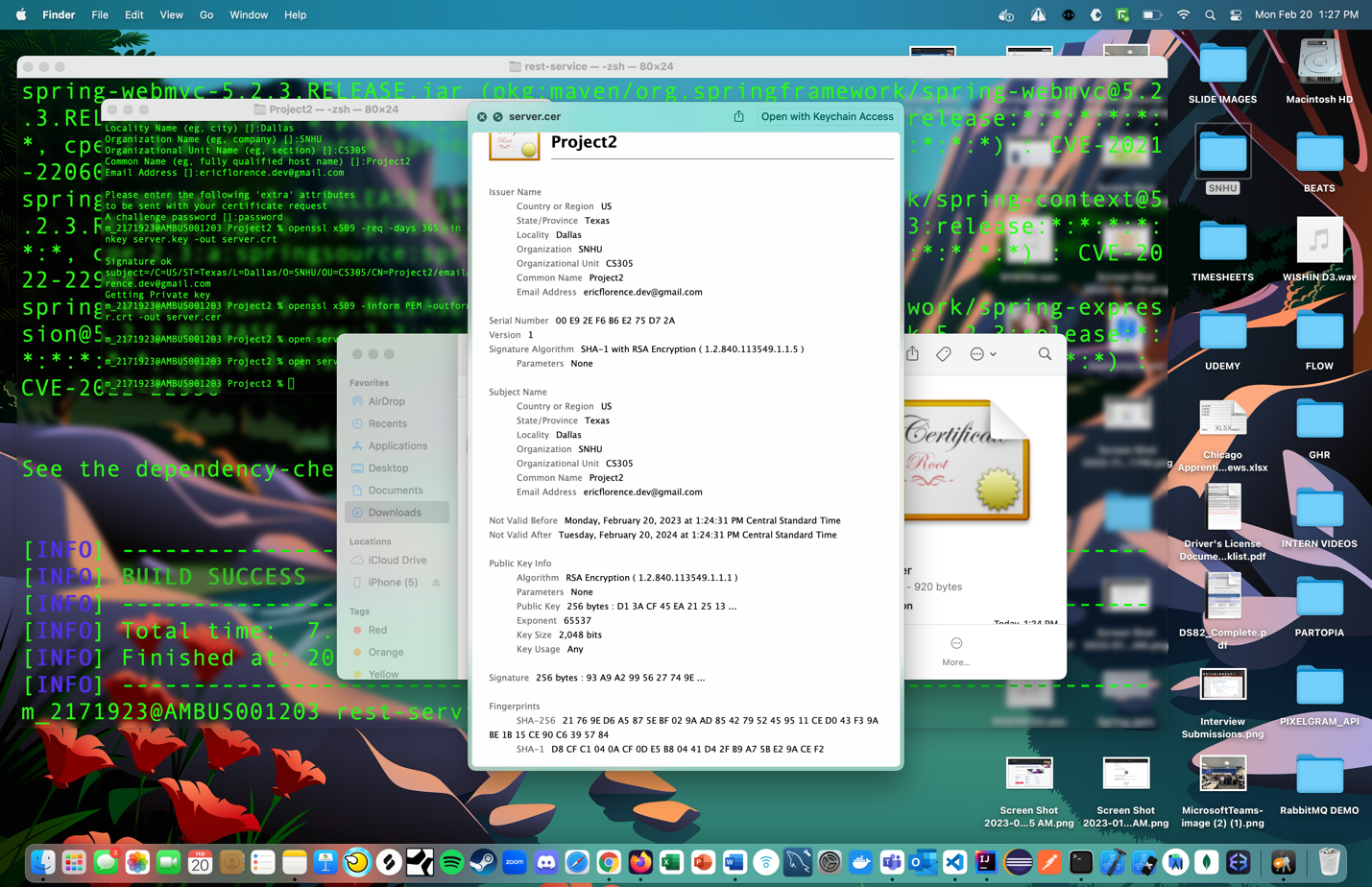
In terms of hash functions, SHA-256 or SHA-3 are commonly used with AES for message authentication and integrity checks. These hash functions provide strong collision resistance and are widely implemented in cryptographic libraries.

In terms of key management, random numbers are essential for generating strong encryption keys. It is important to use high-quality sources of randomness to ensure that the generated keys are not predictable. Symmetric key encryption algorithms like AES require the same key to be used for encryption and decryption, which means that the key must be securely distributed to all parties that need to communicate. Non-symmetric or asymmetric key encryption algorithms like RSA can be used for key exchange, but are typically slower and more computationally intensive.

Encryption algorithms have a long and constantly evolving history, with new algorithms being developed to replace older ones that have been found to be vulnerable or no longer secure enough. For example, the Data Encryption Standard (DES) was widely used in the 1970s and 1980s but is now considered to be too weak for modern applications. Similarly, the original RSA algorithm has been largely replaced by newer, stronger versions.

In summary, based on the information provided and the best practices in the industry, an appropriate encryption algorithm cipher for SSL/TLS communication would be AES with a key length of at least 128 bits, combined with a strong hash function like SHA-256 or SHA-3 for message authentication and integrity checks. Random numbers should be used for key generation, and symmetric key encryption is recommended for performance and efficiency.

## Certificate Generation



## Deploy Cipher

Graphical user interface, application

Description automatically generated

## Secure Communications

Graphical user interface, text, application, chat or text message

Description automatically generated

## Secondary Testing

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

## Functional Testing

## All functions are working so I went back in and updated all the versions for the dependencies to resolve all the security vulnerabilities.

## Other considerations:

1. The application is using a hard-coded string "Hello World Check Sum!" to compute the checksum. This is not secure as it makes the application vulnerable to dictionary attacks and brute force attacks. A better approach would be to use a randomly generated string or a user input to compute the checksum.
2. The server SSL certificate is using a self-signed certificate which is not trusted by browsers by default. This makes the application vulnerable to man-in-the-middle attacks. A better approach would be to use a certificate signed by a trusted certificate authority (CA).
3. The keystore password is hard-coded in the application.properties file. This is not secure as it makes the password vulnerable to attacks if the code is decompiled or if the properties file is accessed by an unauthorized user. A better approach would be to store the password in a secure location such as a password manager or a secure vault.

Text

Description automatically generated

## Summary

The code has been refactored to enable HTTPS with SSL keystore, which adds security to the software application. The following changes were made to achieve this:

1. In the **application.properties** file, the **server.port** was changed to 8443, and the following properties were added to enable HTTPS with SSL keystore:

server.ssl.key-alias=sslserver

server.ssl.key-store-password=password

server.ssl.key-store=ssl-server.p12

server.ssl.key-store-type=PKCS12

server.ssl.enabled=true

These properties specify the SSL certificate alias, password, keystore file, keystore type, and enable SSL for the server.

1. The **SslServerApplication** class was annotated with **@SpringBootApplication** to enable Spring Boot's auto-configuration capabilities.
2. The **ServerController** class was created and annotated with **@RestController** to handle HTTP requests to the server.
3. A **GetMapping** method was added to the **ServerController** class to handle the **/hash** endpoint. This method computes the SHA-256 checksum of the string "Hello World Check Sum!" and returns it to the client.

The refactored code complies with security testing protocols by adding security to the software application through HTTPS with SSL keystore. The following areas of security were addressed by the refactoring process:

1. Confidentiality: HTTPS with SSL keystore is used to encrypt data sent between the client and server, ensuring that it cannot be intercepted and read by unauthorized parties.
2. Integrity: The SHA-256 checksum is computed on the server to ensure the integrity of the data sent between the client and server. This prevents unauthorized modification of the data in transit.
3. Authentication: The SSL certificate is used to authenticate the server to the client, ensuring that the client is communicating with the intended server and not a malicious one.

The process for adding layers of security to the software application involved configuring the server to use HTTPS with SSL keystore, which encrypts data sent between the client and server and ensures that the server is authenticated. Additionally, computing the SHA-256 checksum of the data ensures its integrity. By addressing these security concerns, the refactored code complies with security testing protocols and provides a more secure software application.

## Industry Standard Best Practices

To maintain the software application's security, several industry standard best practices for secure coding were applied. These practices include:

1. Using HTTPS protocol: The code was refactored to convert HTTP to HTTPS, which is a more secure protocol that ensures confidentiality, integrity, and authenticity of data exchanged between the client and server.
2. Using SSL/TLS encryption: SSL/TLS encryption was used to encrypt the communication between the client and server. This ensures that sensitive data transmitted between the client and server cannot be intercepted or read by unauthorized third parties.
3. Using secure passwords: The password used to secure the keystore was changed to a strong and secure password that is not easily guessable.
4. Using secure algorithms: The SHA-256 algorithm was used to compute the hash value. This algorithm is a secure cryptographic hash function that provides strong collision resistance and is less vulnerable to attacks.

By applying these industry standard best practices for secure coding, the software application's security was maintained and its resilience against known security vulnerabilities was increased.

The value of applying industry standard best practices for secure coding to a company's overall wellbeing cannot be overemphasized. Security breaches can result in loss of sensitive data, reputational damage, legal liabilities, and financial losses. Therefore, it is crucial for companies to prioritize security in their software development lifecycle to protect their customers' data and preserve their reputation. By implementing industry standard best practices for secure coding, companies can avoid security breaches, ensure data privacy and confidentiality, and demonstrate their commitment to security, which can boost customer trust and loyalty.