****

# 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** | **[Date]** | **Michael Clerico** |  |

## 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

Michael Clerico

## Algorithm Cipher

I am utilizing SHA-256 (Secure Hash Algorithm 256-bit). SHA is a cryptographic hash function that creates fixed 256-bit outputs. It was developed by the National Security Agency (NSA) and published by NIST. It is set up one way, to make SHA collision resistant. By minimizing collision, we are deterring hackers from accessing signatures. This makes SHA-256 a secure way to store passwords. Each unique input produces a unique output, which further ensures data integrity and password storage. With collisions, hackers are also able to access passwords that have the same value but different because of the collision. In the end we want to limit collisions, SHA-256 has been widely adopted and vetted. For most applications it is considered secure for most applications. With the 256-bit size, it offers a good balance of performance and security. With 1.16 x 10 ^ 77 combinations, SHA-256 makes it impossible to attack with brute force or collisions on any realistic time frame. The hash function takes the input and creates a fixed length output. This output is known as a digest. Again SHA-256 is not an encryption, it is a hash function that does not use keys. Symmetric keys are shared. This is used when encrypting large data, and you need faster access. Some examples of symmetric encryption are AES and DES. Asymmetric encryption is when you have a pubic and private key pair. Because of the complex math that is involved, Asymmetric encryption can be much slower. This is a good option when digital signatures are required. Modern cryptography standards started with DES in 1977 but later replaces with AES as it had a stronger algorithm. Today we utilize the SHA-2 family as the standard that is adopted by NIST.

## Certificate Generation

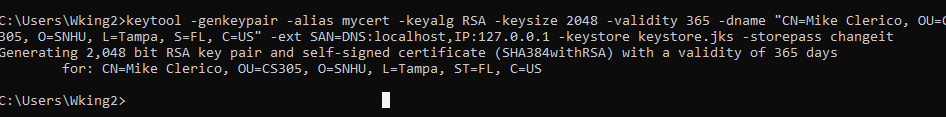
Insert a screenshot below of the CER file.

A screenshot of a certificate

AI-generated content may be incorrect.

## Deploy Cipher

Insert a screenshot below of the checksum verification.



A screenshot of a computer

AI-generated content may be incorrect.

## Secure Communications

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

A screenshot of a computer

AI-generated content may be incorrect.

## Secondary Testing

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

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Functional Testing

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

A screenshot of a computer

AI-generated content may be incorrect.

## Summary

The refactored code followed secure coding practices by utilizing SHA-256 hashing, HTTPS communication, and running vulnerability assessment. The application is configured with SSL/TLS encryption with an SSL certificate identified as keystore.jks. Application.properties are used to ensure it only runs on HTTPS on port 8443. The /hash endpoint generates SHA-256 hash. Using this hash prevents collisions. The hash does not store passwords. This made it possible to remove hardcoded sensitive data like passwords. I then ran a OSWASP dependency check to see the applications vulnerability assessment. The functional testing verified if the hashing worked correctly. The use of HashController.java ensures secure input handling. With these security systems in place, the application meets the standards needed against outside threats. SslServerApplication.java only area of exposure is in its APIs, I added authentication and authorization using spring security. This enables a check where a password is generated and required to enter the application.

## Industry Standard Best Practices

Enforcing Https with TLS encryption using keystore.jks has secured the applications communication. SHA-256 was used to institute collision resistance, which ensures data integrity. Removing hardcoded passwords increases security. Prevented injection attacks by utilizing validated user input in the/hash endpoint. Scanned for vulnerabilities with OWASP dependency check. These protocols will keep data safe and ensure confidentiality. Regular testing will help prevent future attacks by reducing vulnerable dependencies. These enhancements will translate into increased user trust because data leaks will be minimized. Then lastly companies make more money because you will minimize downtime with decreasing cyber threats. With utilizing the SAN and SSL certificate, I can ensure domain names and Ip addresses are validated. In the future I would suggest running dependency checks at least twice a month to reduce new vulnerabilities. This will help minimize down time and increase profitability.

References

National Institute of Standards and Technology (NIST). (2015). Secure hash standard (SHS) (FIPS PUB 180-4). U.S. Department of Commerce. Retrieved from https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

Schneier, B. (1996). Applied cryptography: Protocols, algorithms, and source code in C. Retrieved from https://www.schneier.com/books/applied-cryptography/

Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of applied cryptography. Retrieved from https://cacr.uwaterloo.ca/hac/

Koblitz, N., & Menezes, A. J. (2015). A survey of public-key cryptosystems. Retrieved from https://www.ams.org/notices/200811/tx081101388p.pdf

OWASP Foundation. (2023). Cryptographic storage cheatsheet. Retrieved from https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic\_Storage\_Cheat\_Sheet.html.