****

# 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 6](#_Toc102040759)

[3. Deploy Cipher 6](#_Toc102040760)

[4. Secure Communications 7](#_Toc102040761)

[5. Secondary Testing 7](#_Toc102040762)

[6. Functional Testing 8](#_Toc102040763)

[7. Summary 10](#_Toc102040764)

[8. Industry Standard Best Practices 11](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **[Date]** | **John Brungard** | **Document Template supplies rationale for the chosen hashing algorithm applied to Artemis Financial. Template includes measures of security to include SSL Certificate Generation, SHA-256 Hashing of Data, use of a Key Store to include a Private and Public Key, HTTPS usage for Client/Server connections, manual reviewing of code for defects, and a generated OWASP Dependency Report to identify vulnerabilities. Document closes with a supplied Summary of security uses within the product as well as evidence of Industry Best Practices.** |

## 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

John Brungard

## Algorithm Cipher

Given the scenario and Oracle’s listed Java Algorithm Ciphers, I would choose the Secure Hash Algorithm (SHA) as the cipher for file encryption as it is most appropriate for Artemis Financial’s needs. Specifically, I would choose SHA-256 which is the 256-bit block as more bits typically lead to a higher difficulty for the encryption to be hacked. I also chose SHA-256 due to its collision chances being extremely low. A collision is an instance where an input produces the same digest/hash value as another input given the same exact hash algorithm. This is problematic for decryption purposes and shows a vulnerability susceptible to a collision attack. It can also increase the chances of transmission errors that will make your customers or clients unsure if your original message is unchanged or absent. For this, SHA-2 is appropriate to apply to this situation due to its key length size, collision resistance, and reputation, making it seemingly and to this date under the correct conditions, unbreakable.

Larger bit blocks have a longer length of their encryption key. For a 256-bit block, an attacker would need at least 2^256 combinations to hack a message. By today’s standards, this is something that cannot be accomplished computationally. Its hash function, which is used to convert data into an unreadable output of fixed length in a deterministic manner, makes it so the hashing scheme is applied to each message in the same way, yet another message will not contain the same hash value (*Hash Functions*). The block size used for the hash function will determine the length of the output regarding bit size. For SHA-256, our input after becoming output from the hash function will be 256 bits. For these reasons, keys must be secure yet unpredictable. Many algorithms use a Random Number Generator (RNG) to create a random and secure key.

One difference that a symmetric algorithm such as AES has compared to asymmetric algorithms such as Rivest, Shamir, Adleman (RSA) and Secure Hashing Algorithm (SHA) is that symmetric cryptography, the encryption key, and decryption key are usually the same to where asymmetrical cryptography makes more use of private (encryption) keys and public (decryption) keys. The private key is known strictly by the sender while the public key can be known by anyone and is primarily used to validate the sender of the message (Manico & Detlefsen, 2014).

History of Encryption in various natures can be traced back to before the common era. For instance, around 60 BCE, Julius Caesar would encrypt messages by shifting characters three places (*A brief history of encryption (and cryptography),* 2021). Techniques advanced over time as an encryption device known as the Enigma machine would substitute characters in a message while changing its technique and key daily (*A brief history of encryption (and cryptography),* 2021). Today, we abide by encryption standards such as the Advanced Encryption Standard used by organizations for varying uses (*A brief history of encryption (and cryptography),* 2021). For our chosen algorithm, SHA2 stands for Secure Hash Algorithm and was published in 2001 by the NSA and NIST to succeed a previous version of the SHA1 that was becoming more susceptible to brute force attacks (Jena, 2022). It uses a five-segment process of padding bits, adding a modulus value, initializing buffers, compressing functions and then outputs the final hash digest (Jena, 2022). All of this is formulated into a secure digest to date that will build confidence that the asymmetrical algorithm will show if a message is unchanged or from its intended sender

References

*Hash Functions*. (n.d.). <https://www.freesoft.org/CIE/Topics/142.htm>

Manico. J & Detlefsen, A. (2014, September). *Iron-Clad Java: Building Secure Web Applications.* McGaw Hill Computing.<https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/>

*A brief history of encryption (and cryptography)*. (2021, October 1). Thales Group. https://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-history-encryption

Jena, B. K. (2022, November 11). *A Definitive Guide to Learn The SHA-256 (Secure Hash Algorithms)*. Simplilearn.com. https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm

## Certificate Generation

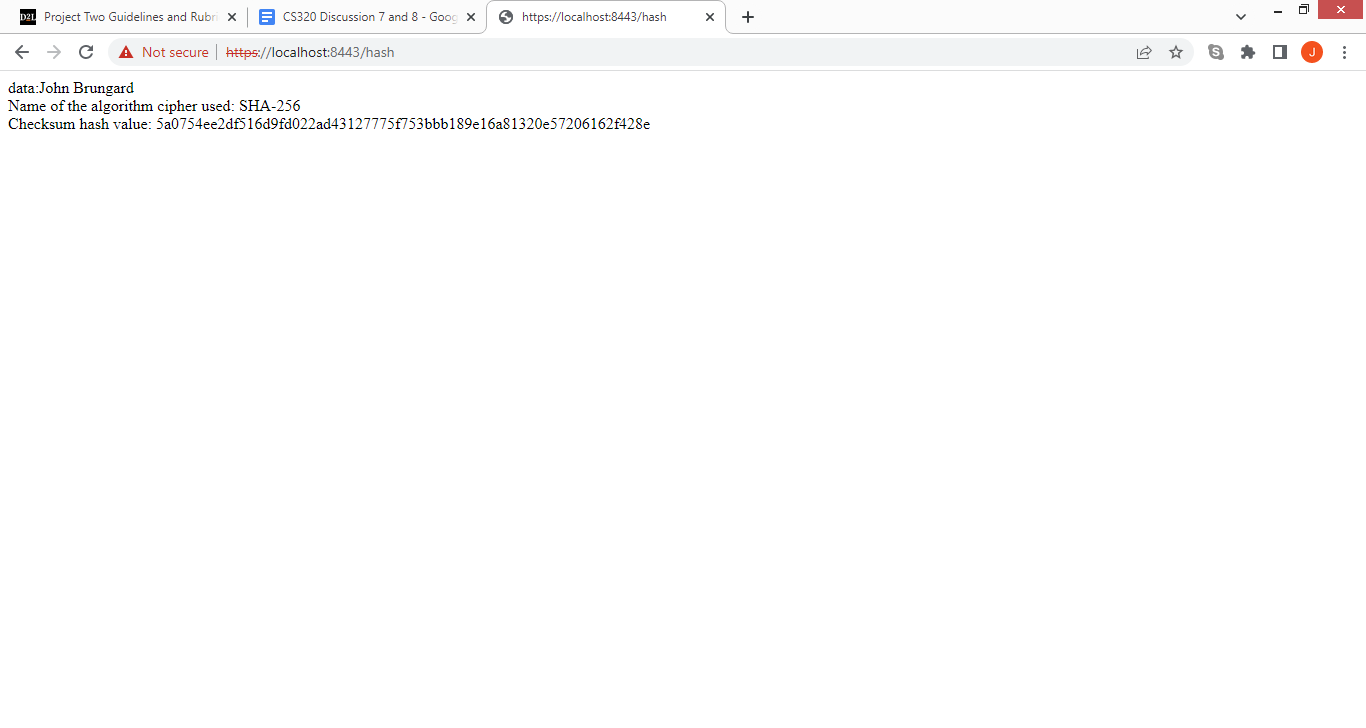
Insert a screenshot below of the CER file.

A screen shot of a computer

Description automatically generated with low confidence

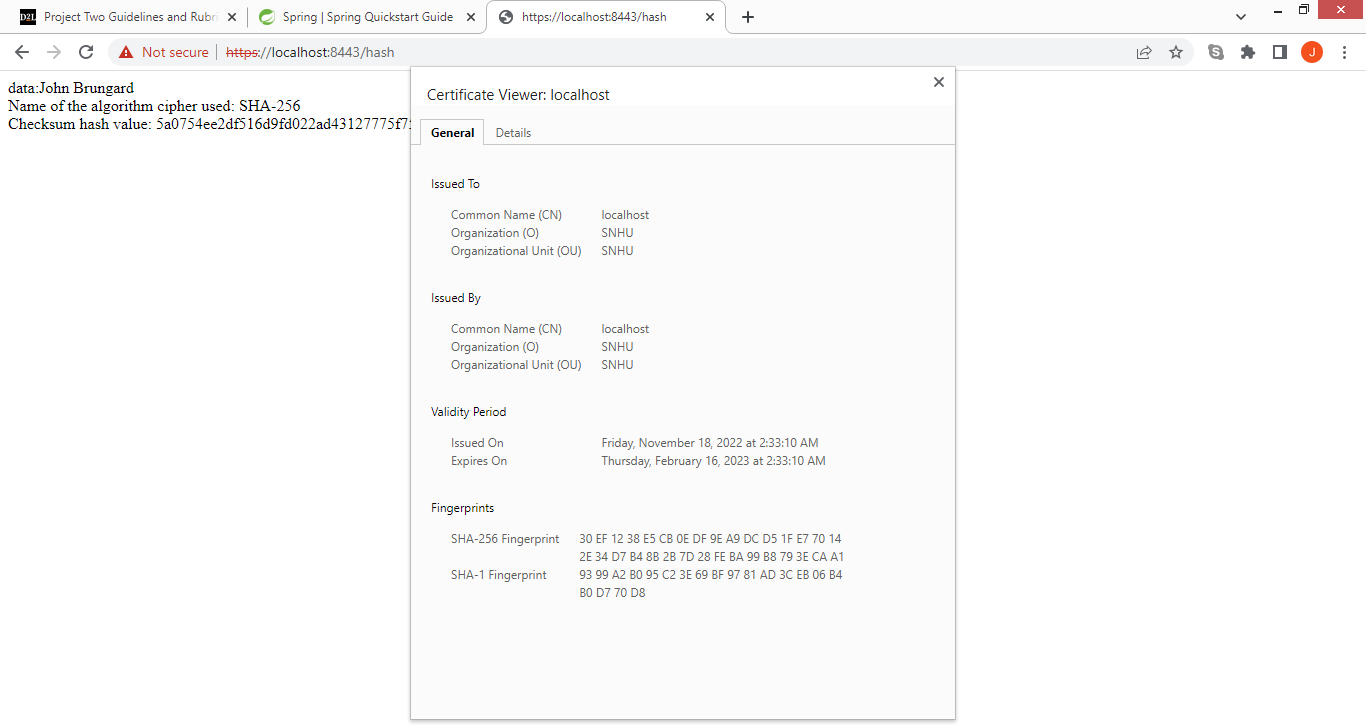
## Deploy Cipher

Insert a screenshot below of the checksum verification.



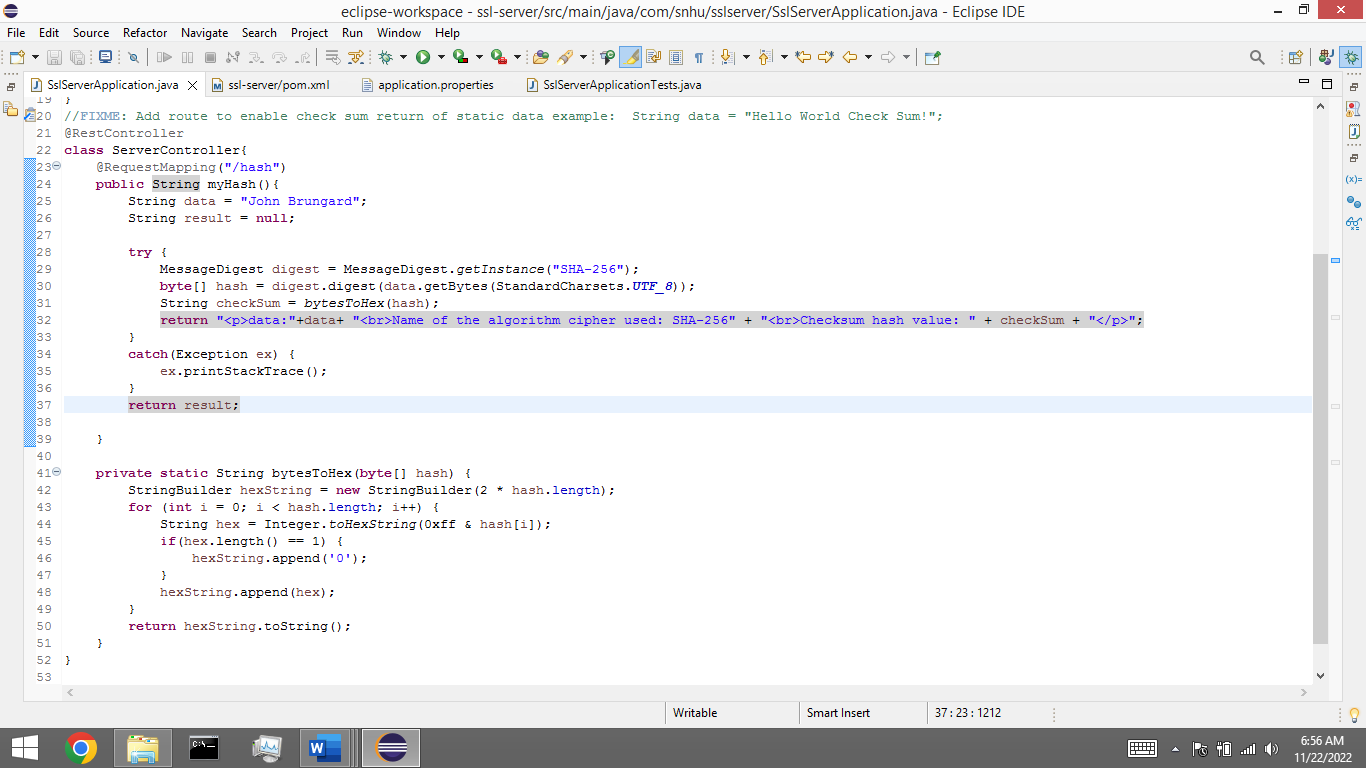
## Secure Communications

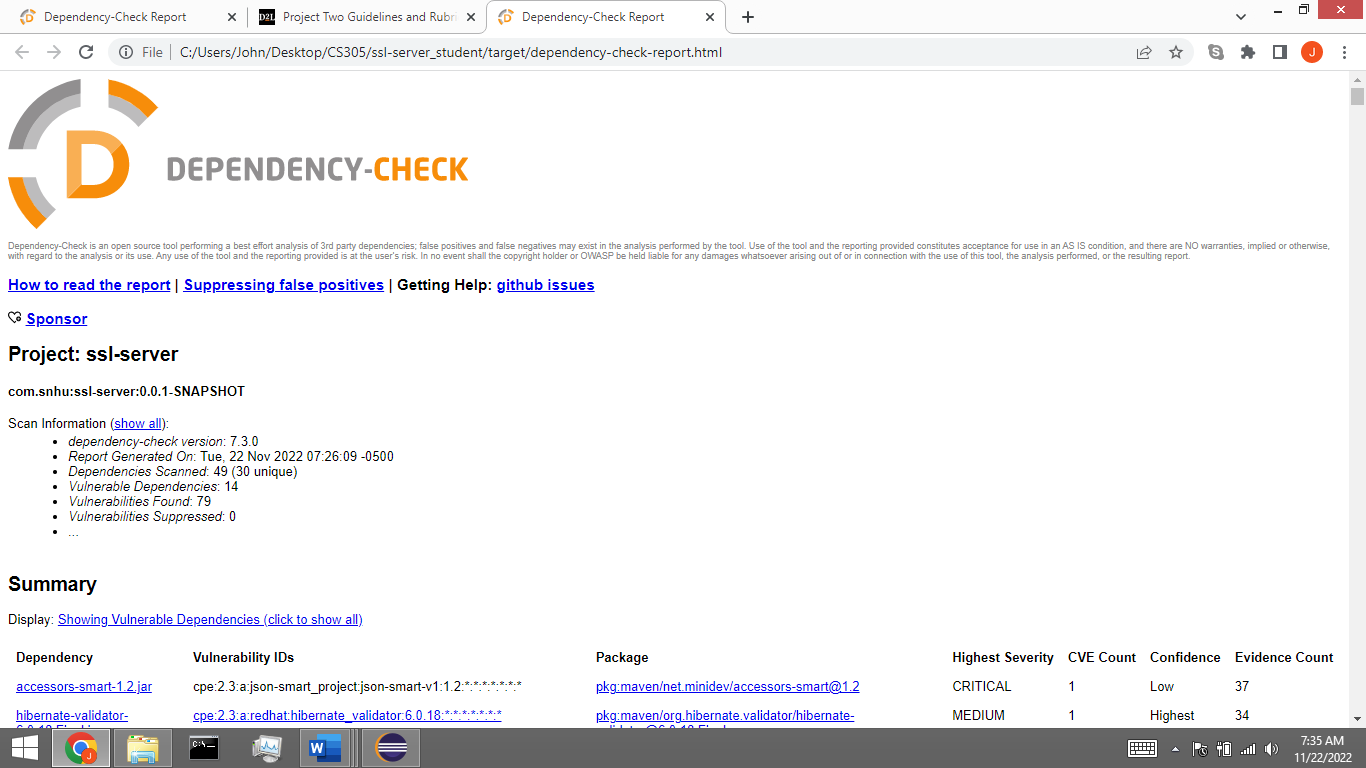
Insert a screenshot below of the web browser that shows a secure webpage.



## Secondary Testing

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





## Functional Testing

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

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, Word

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## Summary

According to the Vulnerability Process Flow Diagram, I addressed Cryptography, Client/Server, and Code Quality. I addressed Cryptography by utilizing an SHA-256 hashing algorithm. This allows Artemis Financial to encrypt data before being stored that could be considered sensitive such as client financial information. I addressed Client/Server by creating an SSL certificate that is used to secure the communication and connection between a user and the server. This will allow clients to view the certificate to evaluate the authenticity of the server and whether or not it has been signed by a CA authority. Finally, I addressed Code Quality by using the appropriate coding patterns and practices in relation to the application.

The first layer of security was choosing an appropriate algorithm pertaining to Financial Artemis’s needs. I decided on SHA-256 due to its reputation for not being breakable under correct conditions and for its 256-bit block. Computationally, the block size makes brute force attacks near impossible to be the measure for uncovering the actual value of encrypted data. Furthermore, its size makes it collision resistant, meaning the chances of two inputs of data that are different in characters producing the same digest is very low, if not impossible.

The second layer of security was using an SSL certificate. The certificate ensures to users that the organization presented is authentic, the connection is secure, and that the CA authority has reviewed and approved the server. If another website pretending to be Artemis Financial tries to retrieve sensitive data from users, the user(s) will be able to verify that they are unauthentic according to the certificate. The certificate and connection were then further enhanced to use HTTPS instead of HTTP.

The final layer of security was using the best practices of coding within my application. Using the Spring Framework helped simplify the authentication process. Due to its reputation, extensive testing and open source framework, it helped simplify the authentication process while still making it very secure, leaving time to focus on the quality of code.

## Industry Standard Best Practices

Within the project, I attempted to provide Industry Standard Best Practices to make a secure product. One practice was making sure the resources involving the project were up to date. An example of this is utilizing the latest version of the OWASP Dependency Check by setting the most recent version in the pom.xml file (7.3.0). This is useful for determining the appropriate dependencies and CVEs within the product.

Another practice was using the appropriate annotations for the Spring Framework. This allowed me to establish a RESTful API that implemented HTTPS and is scalable for any purpose Artemis Financial sees fit. It also offers flexibility for using different data inputs. An example of this is the @RestController annotation used in the Server Application Class. I also used security.require-ssl = true for an extra measure of security within the API.

Finally, I used a function-based approach for my hashing algorithm in the Server Application Class. I did this by using my Bytes To Hex function as well as my function named MyHash(). This made the program modular to where a user can change the data value field and does not have to worry about changing anything else due to function calling and appropriate arguments/parameters.