****

# Practices for Secure Software Report

Table of Contents

[Practices for Secure Software Report 1](#_Toc148104860)

[Document Revision History 3](#_Toc148104861)

[Client 3](#_Toc148104862)

[Instructions 3](#_Toc148104863)

[Developer 4](#_Toc148104864)

[1. Algorithm Cipher 4](#_Toc148104865)

[The recommended algorithm cipher that avoids collisions is SHA-256. 4](#_Toc148104866)

[2. Certificate Generation 4](#_Toc148104867)

[3. Deploy Cipher 6](#_Toc148104868)

[4. Secure Communications 6](#_Toc148104869)

[5. Secondary Testing 7](#_Toc148104870)

[6. Functional Testing 10](#_Toc148104871)

[7. Summary 14](#_Toc148104872)

[8. Industry Standard Best Practices 14](#_Toc148104873)

**9. References…………………………………………………………………………………………………………………………………16**

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **10-15-2023** | **Justin Starr** | **Updated** |

## 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

Justin Starr

## Algorithm Cipher

## The recommended algorithm cipher that avoids collisions is SHA-256.

SHA-256 stands for Secure Hash Algorithm 256 (256 is its digest size in bits) and is a widely used hash algorithm from the SHA 2 family of algorithms (Jena, 2023). As a combined effort from the NSA (National Security Administration) and NIST (National Institute of Standards and Technology), it was published in 2001 because SHA-1 was becoming increasingly weak due to brute force attacks (Jena, 2023). Hashing is performed by scrambling raw information to the point that it cannot be reproduced back to its original form (Jena, 2023). The output it produces of a fixed length is called a hash or message digest (Rhodes, 2023). The output generated from the hashing algorithm will always be the same length, 256 bits, which is 32 bytes or 64 alphanumeric characters (Rhodes, 2023). Significantly, SHA-256 is deterministic meaning it will always produce the same output when it is given the exact same input and there is no way of reverse engineering an input because the knowledge of the output is available (Rhodes, 2023).

When considering an appropriate encryption algorithm, it is important to take into consideration collisions. Collisions occur when you take two completely different inputs, put them through a hashing algorithm, and the same output is produced in each instance (Espinosa, 2021). The higher the amount of bits of the encryption algorithm, the less likely there is a chance of a collision occurring (Espinosa, 2021).

With SHA-256 there are 2^256 (1.157920892e77) possible hash values making it nearly impossible for two different inputs to have the exact same hash value or message digest (Callaghan, 2023).

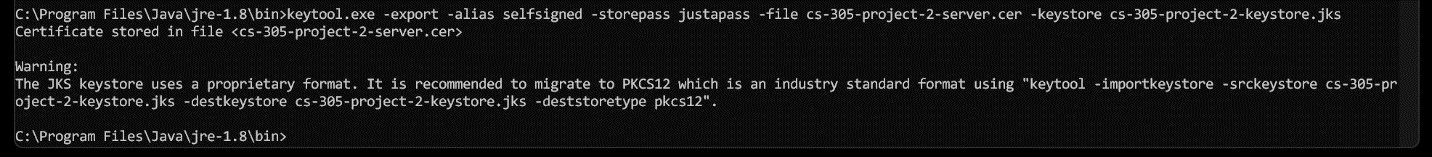
The most significant reason why SHA-256 is so popular and widely utilized is because it has no known vulnerabilities making it insecure, and the algorithm has not yet been broken (Callaghan, 2023). This makes it a leading algorithm over other popular algorithms like SHA-1, which has been broken. It was reported in February of 2017 that Google and CWI announced that they broke SHA-1 encryption and warning signs of its vulnerability began appearing sometime around 2005 (Forbes Technology Council, 2017).

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screen shot of a computer

Description automatically generated

## Secure Communications

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

The following is a screenshot of the refactored application.properties (Line 7 enables HTTPS):

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

The following screenshot shows the server is running on port 8443 making use of the HTTPS protocol:

A screen shot of a computer screen

Description automatically generated

## Secondary Testing

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

The following is a screen shot of the dependency check report before the code base was refactored:

A screenshot of a computer

Description automatically generated

A total of 15 dependencies with a total of 106 vulnerabilities were identified.

The following is a screen shot the refactored code running without errors:

A screenshot of a computer program

Description automatically generated

The following is a screenshot of the dependency check report after the code has been refactored which shows that no new vulnerabilities were introduced:

A screenshot of a computer

Description automatically generated

Again, 15 dependencies with a total of 106 vulnerabilities were identified showing that the refactored code did not introduce any new vulnerabilities.

## Functional Testing

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

The following is a screen shot of the Junit test that is created as part of the spring-boot application.

A screenshot of a computer

Description automatically generated

The execution of this code shows that the spring-boot application runs and that it runs a test to show that the context loads.

The following is a screenshot of the line coverage for the contextLoads Junit test.

A screenshot of a computer program

Description automatically generated

While this test does not test any of the methods, it does show that it tests the spring-boot application and the RestController.

Also, in an effort to maintain industry standards and best practices, after identifying the software application’s syntactical, logical, and security vulnerabilities, we can refactor the code further by updating to a more recent, stable version of spring-boot. The following is a screenshot of the .pom file updated to a recent version of spring-boot (Version 3.0.8) that removes nearly all vulnerabilities that had been previously identified.

A screenshot of a computer program

Description automatically generated

The following is a screen shot of the dependency report’s results after updating spring boot.

A screenshot of a computer

Description automatically generated

Important to note here is that we did not have to suppress any vulnerabilities to achieve this result. We simply had to update the spring-boot version to 3.0.8. This screen shot is also significant because it shows that when we use a more current version in our software we see that the software has much fewer vulnerabilities.

## Summary

According to the Vulnerability Assessment Process Flow Diagram, the areas of security that has been addressed in Artimus Financial’s application by refactoring the code are: Input validation, APIs, Cryptography, and Code Quality.

The process used for adding layers of security to the software application began by making use of a known microservice web-based application framework called spring-boot. It was important to use a recent stable version of the software to ensure that the code was not susceptible to a large number of known vulnerabilities.

The code for Artimus Financial has been refactored in a number of areas. First, to check the code base against known vulnerabilities, the .pom file was updated with the latest version of the OWASP (Open Web Application Security Project) plugin from version 5.3.0 to version 8.4.0. This ensures the code base will be checked against an up-to-date list of known vulnerabilities. Second, the SslServerApplication has been refactored to include a RestController that contains a method for encrypting data with the SHA-256 Algorithm. By incorporating a rest controller into the application, we can help ensure secure communications and transfers of data between servers and clients. We then look at the applications use of cryptography. It includes a method “myHash” that when a user maps to the “/hash” path, the method will take a String of data, and using the calculateHash method, the String of Data is encrypted. Both methods ensure that the software will throw an exception if and “when a particular cryptographic algorithm is requested but is not available in the environment,” (Oracle, 2023). After the data is encrypted the checksum for the encrypted data is displayed to the user. Additionally, the code has been refactored to make use of the secured communications protocol for the web, HTTPS. We can see this by navigating to the application.properties file where the server is configured to run on port 8443 with secured sockets layer enabled (SSL). It incorporates the use of the generated keystore and self-signed certificate that has been previously generated for this project and imported. Lastly, by building a spring-boot application using the spring initializer, a Junit test class was automatically created which can be used to ensure that the SpringBoot application’s context (SprintBootApplication and the RestController) does in fact load.

## Industry Standard Best Practices

Ways that industry standards and best practices were used to maintain the software application’s current security were by first ensuring that the application makes use of an up-to-date version of the spring-bot dependency. This goes hand in hand with ensuring that the software is maintained. If the software is not maintained it could become susceptible to new vulnerabilities and exploits. The less the software is maintained, the greater the risk there is of this happening. Similarly, the more vulnerable web applications are the more likely they are to have sensitive data exposed or unauthorized activities can occur. It is also important to ensure code quality practices are applied. When code is written poorly it makes software more susceptible to attackers. The code itself becomes hard to secure because common practices are not followed such as input validation, error handling, output encoding, or even secure storage (Forsbak, O., 2021).

Keeping security at the forefront of developers' minds helps to reduce security risks. Security should be implemented during each phase of the software development lifecycle. By taking a DevSecOps approach to the software development lifecycle, security is managed all throughout the software’s lifespan. Tools such as static tools can be implemented into code to help determine how vulnerable code is before it is ever deployed in a live environment. One essential tool is the OWASP dependency check plugin that can be implemented into the code base.

There does exist a significant value in applying industry standard best practices for secure coding to the company’s overall well-being. Because Artimus Financial operates in the financial services sector they do handle sensitive information that relates to their clients. Such information includes personal data like names, addresses, birth dates, and social security numbers; all types of information that will always be sought after by attackers/hackers who are willing to try anything to get their hands on this type of information. Also, another category of attacks that the company is potentially susceptible to is Denial of Service (DoS) or Distributed Denial of Service (DDoS) attacks. It is not outside the scope of possibility that others would stand to gain/benefit from Artimus Financials’ financial web services becoming inoperable. It is imperative that Artimus Financial maintains a strict level of security to ensure its clients alongside the company maintain and adhere to strict security protocol. By continuously applying industry standards and best practices in software development, the risk of exposure of sensitive information can be reduced; further reducing the likely high costs associated with data breaches. While not all vulnerabilities can be avoided, practicing secure coding will not only benefit Artimus Financial, but also its clients.

**References**

Callaghan, P. (2023). *Why You Should Use SHA-256 In Evidence Authentication*. Pagefreezer Blog.

<https://blog.pagefreezer.com/sha-256-benefits-evidence-authentication>

Espinosa, C. (2021, September 5). *Explanation of Cybersecurity Hashing and Collisions*. Christian

Espinosa. <https://christianespinosa.com/blog/explanation-of-cybersecurity-hashing-and-collisions/>

Forbes Technology Council. (2017, April 13). *SHA-1 Encryption Has Been Broken: Now*

*What?* <https://www.forbes.com/sites/forbestechcouncil/2017/04/13/sha-1-encryption-has-been-broken-now-what/?sh=4377c2077ee7>

Forsbak, O. (2021, November 29). *10 Best Practices for Software Development Security.* Top Software

Outsourcing Company in Vietname – Orient Software. <https://www.orientsoftware.com/blog/software-development-security/>

Jena, B. K. (2023, August 29). *What is SHA-256 Algorithm: How It Works and Applications: Simplilearn*.

Simplilearn.com. <https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm>

NIST - National Institute of Standards and technology. (n.d.). National Vulnerability Database.

<https://nvd.nist.gov/>

*Open Source Vulnerability Database: MEND*. Mend.io. (2022, May 25).

https://www.mend.io/vulnerability-database/

Oracle. (2023, July 10). *API reference for Java Platform, Standard Edition*. Java Security

Standard Algorithm Names. <https://docs.oracle.com/en/java/javase/11/docs/specs/security/standard-names.html#cipher-algorithm-names>

Oracle. (2023, June 14). *Class NoSuchAlgorithmException*. Nosuchalgorithmexception (java platform SE 8

). https://docs.oracle.com/javase/8/docs/api/java/security/NoSuchAlgorithmException.html

OWASP. (2023). How To Read the Reports. dependency-check – How To Read The Reports.

https://jeremylong.github.io/DependencyCheck/general/thereport.html

Rhodes, D. (2023, August 31). *SHA-256 Cryptographic Hash Algorithm*. Komodo Academy | En.

<https://komodoplatform.com/en/academy/sha-256-algorithm/>