****

**Southern New Hampshire University**

# Practices for Secure Software Report

**Jordan Ballard**

**Dr. Lyon**

**CS 305 – Software Security**

**November 13, 2022**

Table of Contents

[Practices for Secure Software Report 1](#_Toc121673529)

[Document Revision History 3](#_Toc121673530)

[Client 3](#_Toc121673531)

[Instructions 3](#_Toc121673532)

[Developer 4](#_Toc121673533)

[1. Algorithm Cipher 4](#_Toc121673534)

[2. Certificate Generation 5](#_Toc121673535)

[3. Deploy Cipher 6](#_Toc121673536)

[4. Secure Communications 6](#_Toc121673537)

[5. Secondary Testing 6](#_Toc121673538)

[6. Functional Testing 7](#_Toc121673539)

[7. Summary 8](#_Toc121673540)

[8. Industry Standard Best Practices 8](#_Toc121673541)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/11/22** | **Jordan Ballard** |  |

## 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

Jordan Ballard

## Algorithm Cipher

To meet Artemis Financial’s security requirements, I would recommend the SHA-256 algorithm cipher. SHA-256 is used to hash information using a message schedule of sixty-four words, eight working variables, and a hash value of eight words, all of which are 32 bits each, resulting in a message digest of 256 bits (Dang, 2015). SHA-256 is part of the SHA-2 family which succeeds SHA-1 and its 160-bit output, and is considered more secure due to the larger output size and increased collision resistance (Steven, 2021). SHA-256 is the most widespread algorithm of the SHA-2 family and is used in security protocols such as Transport Layer Security, Internet Protocol Security, or Secure Shell and it also has uses in data authentication, blockchain technologies, and protecting government data (Lake, 2022). Aside from SHA-256, SHA-2 includes a group of other functions identified as SHA-224, 384, and 512 which all use the same algorithm but have varying output hash lengths as indicated by their names (Lake, 2022). SHA-224 and 256 have a block size of 512 bits with the input processed as 32 bit words, and operates on the data in 64 rounds before producing the final hash (Lake, 2022). SHA-384 and 512 have a block size of 1,024 bits and process inputs as 64-bit words, operating on the data for 80 rounds before the final hash (Lake, 2022). Larger output size means increased collision resistance, but since SHA-256 is still considered secure and is the most widely used function of the family, it is the best option for Artemis Financial.

Random numbers are used in some encryption algorithms, such as RSA, to generate encryption keys, but SHA algorithms do not use random numbers themselves and are instead more useful for generating random numbers; for example, a pseudo-random number can be acquired by taking the SHA-1 hash of an entropy pool’s contents which keeps the pool confidential (Sidhpurwala, 2019). Likewise, many encryption algorithms are either symmetric, when data is encrypted and decrypted using a single key, or asymmetric, when data is encrypted and decrypted with two linked public and private key pairs (Daniel, 2021). Symmetric encryption is useful because it is fast due to shorter key lengths, yet still secure and considered industry standard, while asymmetric encryption is useful since key distribution is not needed nor is exchanging private keys (Daniel, 2021). When it comes to SHA-256 however, it cannot be considered symmetric or asymmetric since cryptographic hash functions are one-way and have no direct way of decrypting the message (Lake, 2022).

Encryption dates back for thousands of years, with some of the first known uses being found in ancient Egyptian hieroglyphics and later the use of the Caesar cipher, neither of which were very secure (Sidhpurwala, 2013). The Vigenere cipher was among the first to use the idea of an encryption key which helped set the stage for encryption algorithms that we use today (Sidhpurwala, 2013). World War I saw the invention of the Enigma machine which built on the foundations of security laid before it but it was still eventually broken and considered obsolete (Sidhpurwala, 2013). In more modern times, the DES block cipher was set as the national standard for encryption until it was broken some years later, followed by AES which is still the standard today (Sidhpurwala, 2013). Looking at the history and progression of encryption algorithms, it is clear that they become progressively stronger as vulnerabilities are found and that the current state of encryption is stronger than it has ever been. However, it also shows that encryption needs to continue to advance as technology does to avoid breaches in security.

## Certificate Generation

Insert a screenshot below of the CER file.

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

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

Description automatically generated

## Secure Communications

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

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

Description automatically generated

## Secondary Testing

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

Graphical user interface, application

Description automatically generated

Dependency Check Before: Dependency Check After:

Text

Description automatically generatedText

Description automatically generated

## Functional Testing

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

Text

Description automatically generated

**Manual Review and Mitigation Plan**

* pom.xml
  + 
  + This is an older version of Spring Boot. Support ended 11/18/21 and now only has commercial support, which ends 2/23/23, so will only receive releases for critical bug fixes or security issues when requested (*Spring Boot,* n.d.).
  + Mitigation plan is to consider updating to Spring Boot 3.0 which has support until 11/24/23 and commercial support until 2/24/25 (*Spring Boot,* n.d.).
  + Also in pom.xml, an old version of the Maven dependency check was specified. This was mitigated before running the dependency checks by using the most current version.
* application.properties
  + Text

    Description automatically generated
  + There are many more properties available to be specified, including ones regarding security (*Common Application Properties,* n.d.).
  + Mitigation plan is to consider adding additional properties, if they apply, especially ones that add additional security.

## Summary

Areas of security from the Vulnerability Process Flow Diagram that were addressed when refactoring the code were: secure API interactions since Spring was used, encryption use since SHA-256 was used, client/server since the code is the server side and the browser is the client side, secure error handling since exceptions were used, and code quality since secure practices and patterns are always crucial.

To add layers of security to the application, the code has been refactored by utilizing the MessageDigest library and a bytes to hex function to deploy and implement the SHA-256 cryptographic hash algorithm. The @RequestMapping method was used to create a RESTful route to the web browser. In order to convert from an HTTP to an HTTPS protocol, I generated a self-signed certificate and installed it to my machine as a trusted root certificate authority. Then I added keysotre.jks to my project and refactored the code in the application.properties file with the correct entries to enable HTTPS with the SSL keystore. A dependency check was run before and after refactoring the code to ensure that no new vulnerabilities were introduced, and functional testing through a manual code review was conducted to identify additional vulnerabilities.

## Industry Standard Best Practices

Industry standard best practices were followed in refactoring the code base. Appropriate naming conventions were used as well as comments on code functionality, both of which are helpful in understanding what the code does. bytesToHex was encapsulated into its own method and was declared as private. In the myHash method, a try-catch block was used for NoSuchAlgorithmException handling. Also in the myHash method, the static method getInstance was accessed in a static way by not using the object to call the method. In application.properties, the entries were changed so that an HTTPS protocol was used for additional security and dependency checks were run to ensure that no new vulnerabilities were added after code refactoring.

Applying industry standard best practices for secure coding is extremely valuable in a software application for a company’s overall wellbeing. Security bugs can cause issues like theft of confidential data, abuse of system resources, opening of security holes, and many other issues (*Secure Coding Guidelines for Java SE*, 2022). If any of this were to happen to a company it would affect not only them but their customers as well, significantly reducing trust and compromising personal data. Additionally, secure coding helps to ensure reduced failure rate of the software, optimized development time, more secure software, and an overall company culture of secure coding (*Security by Design: The Advantages of Secure Coding Best Practices,* 2020).

**References**

*Common Application Properties.* (n.d.). Spring. https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html

Dang, Q. (2015). Secure Hash Standard. *National Institute of Standards and Technology*. https://doi.org/10.6028/NIST.FIPS.180-4

Daniel, B. (2021, May 4). *Symmetric vs. Asymmetric Encryption: What’s the Difference?* Trenton Systems. https://www.trentonsystems.com/blog/symmetric-vs-asymmetric-encryption

Lake, J. (2022, February 17). *What is SHA-2 and how does it work?.* Comparitech. https://www.comparitech.com/blog/information-security/what-is-sha-2-how-does-it-work/

*Secure Coding Guidelines for Java SE.* (2022, October). Oracle. https://www.oracle.com/java/technologies/javase/seccodeguide.html

*Security by Design: The Advantages of Secure Coding Best Practices* (2020, December 17). Gold Sky Security. https://www.goldskysecurity.com/security-by-design-the-advantages-of-secure-coding-best-practices/

Sidhpurwala, H. (2019, June 5). *Understanding random number generators, and their limitations, in Linux.* Red Hat. https://www.redhat.com/en/blog/understanding-random-number-generators-and-their-limitations-linux

*Spring Boot*. (n.d.). Spring. https://spring.io/projects/spring-boot

Steven, J. (2021, October 18). *SHA2 ‘vs.’ SHA1*. Application Security Blog. https://www.synopsys.com/blogs/software-security/sha2-vs-sha1/