****

# 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** | **[2/23/2024]** | **[Alexander Griffith]** |  |

## Client



## Instructions

Submit these 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

[Alexander Griffith]

## Algorithm Cipher

1. AES, otherwise known as Advanced Encryption Standard, is a solid standard used by many organizations as their algorithm encryption cipher of choice, so it has been tested in the past. It is secure if it is implemented correctly and even then, cryptographers have tested it extensively and continue to give it a stamp of approval. It is a symmetric block cipher that “chunks” data into 128-bit blocks and can be configured for three different key lengths 128, 192 ,256 with 256-bits being the strongest. This is due to the length of the key as well as the number of iterations that must be performed during encryption and decryption which increases with each key-length reaching 14 iterations at 256 bits.
2. Hash functions are used primarily to confirm data integrity and identity verification. It does this by taking an input and “hashing” it to a hexadecimal value. This hexadecimal value can then be stored and used to verify that data did in fact come from the proper source or that the data has not been modified by checking it against the stored hash value. The bit levels of a cipher are the amount of possible key combinations, therefore, the more possible keys there are the harder it is to crack through brute force attacks. An example of a hash function algorithm would be SHA-256.
3. Random numbers are generally used in cryptographic applications to reinforce security through a myriad of ways. Whether it is used to differentiate similar passwords through the introduction of “salt” junk data using random numbers or in the generation of the cryptographic keys to make the keys as secure as possible.

Symmetric key based algorithms are private key algorithms, that use the same key to decrypt and encrypt data and are less computationally expensive compared to asymmetric key based algorithms. This comes at the cost of the private key needing to be a secret key that must be protected.

While asymmetric keys are also known as public key cryptography and is typically used to sign data rather than encrypt it to ensure that the sender is truly who they say they are. It is comprised of a duo of keys; one is kept secret known only to the original sender. While the second key is the public key and used to validate the data signature. It can also be used to encrypt data with the public key but can only be decrypted by the private key. However, not every asymmetric algorithm is capable of this.

1. Encryption algorithms were in use before the advent of computing, as early as Spartans using a form involving a rod and a leather strip. However, computer-based encryption truly doesn’t start until the 1970’s, when IBM began to work on a block cypher now known as DES (Data Encryption Standard). DES stayed in use until DES was cracked in 1997. The invention of the idea that there would be one private key and one public key was published in a research paper during 1976 by Whitfield Diffie and Martin Hellman, which enables asymmetric ciphers. In 2000, AES is created through an open competition. Encryption will more than likely continue to advance as needed when computational power becomes stronger and more available.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

Description automatically generated



A screenshot of a certificate

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

Description automatically generated

## Secure Communications

(Had issues here not sure on how to solve or what I missed)

A screenshot of a computer

Description automatically generated

## Secondary Testing

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

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Functional Testing

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

A screen shot of a computer

Description automatically generated

## Summary

The code has been refactored to enable HTTPS by updating the configuration of the application.properties file with the line server.ssl.redirect-https=true. It has been modified to utilize a specific keystore. Further enhancements could involve implementing authentication requirements or access denial using the line server.ssl.client-auth=need.

However, attempts to resolve issues encountered during development, achieving compatibility between a self-signed certificate and my Chrome browser remains vexing with no solution coming to mind. Various methods, configurations, and even installing certificates to the trusted certificates within Chrome's security settings have been tried without much luck, leaving me frustrated and leaving it out of the refactoring attempts.

Concerning the vulnerability assessment process flow diagram and the security aspects addressed by the code refactoring, several branches of the diagram were touched upon:

APIs: Additional elements from the Spring Framework libraries have been imported as required and in use.

Cryptography: Hashed checksums (using SHA-256) of text within the code have been utilized. Input validation has not been addressed, given the absence of user input calls or any evident need for it.

Code Quality and Error Handling: Implementation of cryptographic capabilities as per project requirements, thorough testing to ensure error-free execution, and proper configuration of the application.properties file to ensure secure and accessible webpages upon program execution (or at least as much as I could did not think this would be the part that tripped me up as hard as it did).

## Industry Standard Best Practices

A . I used industry standard best practices to maintain the security of the project by making sure to not introduce unnecessary dependencies, since the cryptography could have also been implemented with a bouncy castle library as well and instead made use of the spring framework ones that were already a part of the dependencies in use. I also attempted to update the Spring Boot dependency within the project but that lead to errors that I could not find any solutions to and so reverted it back for the time being.

While checking the dependency report there were vulnerabilities found, however after reading the actual risks involved with those vulnerabilities. I found that they were outside the scope of this project at this time. Therefore, I saw little reason to address them nor any reason to create a suppression file, since if development continued past this point and additional features used this application in a cloud foundry deployment. The suppression of those vulnerabilities would hide potential risks that are now prevalent.

B . Implementing industry standard best practices for secure coding is of extreme importance to a company’s wellbeing in the long run for many reasons. By using secure coding from the beginning risks can be mitigated from the start of development and avoid potential scandals that could arise from neglecting security. When one neglects security it can lead to a firm’s image being damaged and remove consumer trust, which could lead to a firm’s downfall.

Especially in this case where the firm is one within the financial field, who would want to put their money in an unlocked and unwatched safe? No one, which is why to even operate as a financial firm there are regulatory bodies that demand certain security measures be put in place or a firm will face consequences and in turn is an industry standard.

Also, coding with industry standards and security in mind from the beginning also reduces the likelihood of wasting labor and funds on work that may need to be revisited due to a lack of foresight as development progresses. So all in all, a company should keep industry standard best practices in mind and by extension all of their developers if they wish to operate at their best.

Sources

ClickSSL. (n.d.). 256 Bit Encryption: The Evolution, Benefits & Drawbacks. ClickSSL. Retrieved from https://www.clickssl.net/blog/256-bit-encryption#:~:text=There%20have%20been%20a%20few,not%20been%20broken%20to%20date.

AppSealing. (n.d.). AES 128 Encryption: All You Need to Know. AppSealing. Retrieved from https://www.appsealing.com/aes-128-encryption/#:~:text=AES%20has%20never%20been%20cracked,AES%20for%20securing%20sensitive%20information.

SIMMS. (n.d.). Understanding the Levels of Encryption. SIMMS. Retrieved from https://www.simms.co.uk/tech-talk/understanding-the-levels-of-encryption/#:~:text=Modern%20AES%20encryption%20uses%20128,the%20code%20is%20to%20crack.

Oracle. (n.d.). Cipher Algorithm Names. Oracle. Retrieved from https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names

O'Reilly Media. (n.d.). Chapter 6. Security. Iron-Clad Java. Retrieved from https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch06.html#ch06lev2sec7

Thales Group. (n.d.). A Brief History of Encryption. Thales Group. Retrieved from https://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-history-encryption

Cloud Foundry. (n.d.). Getting Started Deploying Apps with the Spring Framework. Cloud Foundry. Retrieved from https://docs.cloudfoundry.org/buildpacks/java/getting-started-deploying-apps/gsg-spring.html