****

# 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 5](#_Toc102040759)

[3. Deploy Cipher 6](#_Toc102040760)

[4. Secure Communications 6](#_Toc102040761)

[5. Secondary Testing 6](#_Toc102040762)

[6. Functional Testing 9](#_Toc102040763)

[7. Summary 11](#_Toc102040764)

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

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **6/12/23** | **Quintin B. Rozelle** | **Initial Draft** |

## 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

Quintin B. Rozelle

## Algorithm Cipher

Artemis Financial is in the process of updating its business to be competitive in the current market. One aspect of this is to incorporate cipher algorithms in their software to increase security. Since Artemis Financial manages sensitive information for its clients, a strong encryption algorithm and hashing algorithm are needed. Additionally, these algorithms will be used with data transfer between the web server and clients. Based on this, speed is important. Given these requirements, it is recommended to use the Advanced Encryption Standard (AES) with a 128-bit key and the Secure Hash Algorithm 2 (SHA-2) with a 256-bit key which have become the standard of encryption and hashing across the internet.

Encryption Algorithm (AES)

The AES algorithm works by breaking the original message into 128-bit blocks. It is then combined with the first part of the key using XOR logic. This output is then arranged into a 4-by-4 grid where it is first run through a substitution function, then rearranged by shifting the rows and mixing the columns via matrix multiplication. The substituted and permuted information is then run through the process again starting the next section of the key. This substitution and permutation step is the hash function for this algorithm. This entire process repeats a total of 10 times for a 128-bit kit and the output is an entirely encrypted file. Different bit sizes can be used, but the process is identical with the exception that a 192-bit key has 12 cycles while a 256-bit key has 14 cycles. (Jena, 2023)

The key used for this algorithm could be any string of bits that is 128, 192, or 256 long, but a random string of this length is needed to be the most secure. Due to this, a random bit generator is used to create the key. This helps increase security by ensuring every bit in the key is not determinant on another which would make it easier to crack. (Smirnoff & Scholten, 2019)

The AES algorithm is a symmetric encryption algorithm. This means the same key is used to encrypt and decrypt the message. This contrasts with an asymmetric algorithm which uses a public key to encrypt a message and a separate private key to decrypt it. Based on this, it is important to ensure that the key used with AES is not compromised. (Poggi, 2021)

Before the creation of the AES algorithm, the Data Encryption Standard algorithm (DES) was primarily used. As time progressed and computers became faster, the DES algorithm began to become broken. It was first cracked in 1997 and is now capable of being broken in less than 5 minutes. To prevent this, a technique known as triple-DES was used which is essentially running the original message through the DES algorithm three times, but this came with the cost of increased encryption time. Due to this, a stronger algorithm that was faster and harder to crack was needed. In 1997, the National Institute of Standards and Technology (NIST) put out a request for an algorithm that would satisfy these requirements. Five final candidates were eventually received and reviewed with the Rijndael algorithm (now known as AES) ultimately becoming selected. (NIST, 2023)

Hashing Algorithm (SHA-2)

The SHA-2 algorithm is used to create a “file signature” to ensure that a file transferred across the internet hasn’t been modified in any unintentional way. It creates a fixed-length hexadecimal string from a file that can be compared to the known string to ensure no change has occurred. This hexadecimal string is created by breaking the file into 512-bit chunks. The first chuck is fed through a hash function along with a starting string to produce an initial checksum. This output is fed back through the function with the next 512-bit chunk to produce a new checksum output. This process is repeated until all 512-bit chunks are processed, and a final checksum is produced. Regardless of the file input size, the SHA-256 algorithm always produces an output of 256 bits. (Ellis, 2018)

Collision is a concern with hashing function since there are far more potential inputs (virtually infinite) than there are outputs (2^256). Despite this, the SHA-256 algorithm has a low chance of experiencing collision under normal conditions since there is only a 1 in 2^256 (1 with 77 zeros after) chance of two separate inputs producing the same output. While it is theoretically possible that someone with malicious intent could create a harmful file that does produce the same checksum as the intended file, the chances of this given the current state of computing power is negligible. (Ellis, 2018)

## 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

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

Description automatically generated

## Secure Communications

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

Note: Firefox is connected via HTTPS but does not list it as a secure webpage (see lock icon with warning symbol) since the certificate is self-signed.

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.

Screenshot of refactored code:

A screen shot of a computer program

Description automatically generated with low confidence

A picture containing text, screenshot, software, multimedia software

Description automatically generated

Problems tab showing no errors:

A screenshot of a computer

Description automatically generated with medium confidence

Console output showing correct execution:

A screenshot of a computer

Description automatically generated with medium confidence

Vulnerability report summary of starter code before any modifications:

A screenshot of a computer error

Description automatically generated with low confidence

Vulnerability report summary of refactored code showing no additional vulnerabilities added:

A screenshot of a computer error

Description automatically generated with low confidence

Vulnerability report summary of refactored code after updating dependencies showing a reduction in vulnerabilities:

A screenshot of a computer error

Description automatically generated with medium confidence

Refactored pom.xml file with updated dependencies:

A screen shot of a computer

Description automatically generated with medium confidence

## Functional Testing

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

Screenshot of refactored code:

A screen shot of a computer program

Description automatically generated with low confidence

A picture containing text, screenshot, software, multimedia software

Description automatically generated

Problems tab showing no errors:

A screenshot of a computer

Description automatically generated with medium confidence

Console output showing correct execution:

A screenshot of a computer

Description automatically generated with medium confidence

## Summary

Several areas of security were addressed during the refactoring of this code:

* APIs – The code was refactored to use RESTful APIs when sending and receiving data over the Internet. It does this by using a RestController to manage requests.
* Cryptography – Cryptography was incorporated using the SHA-256 algorithm. Without this, it would not be possible to ensure that data sent to or from the web server was what it was believed to be. The SHA-256 algorithm solves this by creating a checksum for data which can be used to confirm that data was not tampered with before it arrived at its destination. This algorithm is very secure (i.e., incredibly difficult to crack) and has a very low chance of collision.
* Client/Server – The use of certificates has been implemented. This helps to improve security by allowing for HTTPS connections as they act as confirmation that the site you are connected to is what you believe it to be. With HTTPS connections, all traffic between the client and server is encrypted.
* Code Quality – Dependencies were updated to reduce the number of known vulnerabilities in the application. By updating to the latest version, the known vulnerabilities went from 100 to just 1.

In short, layers of security were added by utilizing a certificate to ensure encrypted HTTPS connections, the SHA-256 hashing was included to ensure file transfer was completed without tampering, known vulnerabilities were eliminated by updating dependencies, and RESTful API communication was implemented. All refactored code can be seen in the attachment.

## Industry Standard Best Practices

As stated above, the dependencies used for this application were updated to the current version using Maven. This helped to maintain the system’s current security by removing known vulnerabilities. Additionally, the code was continually tested throughout the refactoring process to ensure that no bugs or additional vulnerabilities were introduced. This was confirmed by reviewing the OWASP Dependency Report before any refactoring to get a baseline vulnerability value then reviewed again during and after refactoring to show that the vulnerabilities did not increase. This process was also repeated while updating dependencies to confirm that any changes didn’t have detrimental effects.

Incorporating these and other secure programming practices is paramount to optimal software creation. Individuals with malicious intent are extremely prevalent on the internet and it must be assumed that if a vulnerability exists, it will be found and exploited for their gain. This could be as simple as vandalizing or taking down a website, or it could be as nefarious as data or financial theft. Regardless of the outcome, companies must do everything they can to prevent this before it happens as the damage from an attack could be catastrophic.

## References

Ellis, S. (2018, December 13). *The beautiful hash algorithm*. Medium. https://steviecellis.medium.com/the-beautiful-hash-algorithm-f18d9d2b84fb

Jena, B. K. (2023, February 9). What is AES encryption and how does it work?. Simplilearn. https://www.simplilearn.com/tutorials/cryptography-tutorial/aes-encryption

NIST. (2023, March 8). AES development. NIST. https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development

Poggi, N. (2021, June 15). Types of encryption: Symmetric or asymmetric? RSA or AES?. Prey Blog. https://preyproject.com/blog/types-of-encryption-symmetric-or-asymmetric-rsa-or-aes

Smirnoff, P., &amp; Scholten, U. (2019, April 23). The use of random number generators in cryptography. Cryptomathic. https://www.cryptomathic.com/news-events/blog/the-role-of-random-number-generators-in-cryptography