****

# 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** | **12/7/2022** | **Hunter Richards** | **Updated the Algorithm Cipher, Certification Generation, Deployed Cipher, Secure Communication** |
| **1.1** | **12/10/2022** | **Hunter Richards** | **Updated the Functional Testing, Summary, and Industry Best Practices** |
| **1.2** | **12/11/2022** | **Hunter Richards** | **Proof-read and adjusted each section accordingly** |

## 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

Hunter Richards

## Algorithm Cipher

Cipher Chosen: **SHA-256**

SHA-256 is a hashing algorithm to validate files by providing a unique irreversible hash for an application/file. Every modification to the data input into the hashing function adjusts the uniquely given hash. This is integral to mitigate man-in-the-middle attacks (by hijacking the file and modifying the data) or to prevent general file manipulation through verification. SHA-256

SHA-256 has a very low collision rate. This means that having malicious data hash to a verified hash value is very unlikely. SHA-256 has possible hash values (bit level), making any chance of a collision almost entirely impossible. This is important as the same checksum values (i.e. collisions) can be exploited to send malicious payloads. If the checksum value is the same, then an unsuspecting user may not think twice of the file’s contents.

SHA-256 isn’t asymmetric or symmetric as it doesn’t use public or private keys yet rather maps an arbitrary length input into a fixed length output (hashing). Asymmetric/symmetric algorithms are typically used for transferring/storing data versus verification. AES-256, for example, is a symmetric algorithm because it has the same public/private key. This means that the key to encrypt and decrypt data uses the same AES-256 key. As stated in the name, AES-256 uses 256 bits to encrypt data.

Both SHA-256 and AES-256 randomize their data during the encryption process. SHA-256 appears randomized to the observer (pseudo-random) who does not know the given input (as stated previously by how hashing operates). AES-256 randomizes through multiple ‘rounds’ (specifically 14 in the case of 256-bit AES) through sub bytes, round shifting, column mixing, and round keys (keys specific to the round).

SHA-256 (Secure Hashing Algorithm) is a cryptographic hash function designed by the United States National Security Agency in 2001. It has become an industry standard to provide verification through hashing for checksums. SHA-2 is the family where SHA-256 resides. SHA-2 has six hash functions, each with different bit levels, including: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256. SHA-3 was released by the NIST in 2015. It was designed fundamentally different from SHA-2 and has its own advantages/disadvantages, which is why that family was not chosen for this scenario.

AES (Advanced Encryption Standard), or its original name of Rijndael, is an encryption standard established by the US National Institute of Standards and Technology in 2001. AES was built to replace DES (Data Encryption Standard) as the leading standard for cryptography. DES’s 56-bit key was found vulnerable to cracking and couldn’t withstand modern compute cracking capabilities. AES is considered the most common, effective, and secure cryptography algorithms available. It’s become an industry standard among many institutions which wish to adopt some form of secure data encryption.

## Certificate Generation

Insert a screenshot below of the CER file.

Text

Description automatically generated

Text

Description automatically generated

CERTIFICATE FILE:



## Deploy Cipher

Insert a screenshot below of the checksum verification.

Graphical user interface, text, application, email

Description automatically generated

## Secure Communications

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

**<< Can’t get Chrome to accept the locally created CA, however, chrome is clearly picking up on the self-signed certificate per the below screenshot >>**

Graphical user interface, application

Description automatically generated

## Secondary Testing

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

Text

Description automatically generated

Text

Description automatically generated

A screenshot of a computer

Description automatically generated

## Functional Testing

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

Text

Description automatically generated

## Summary

**Refer to the Vulnerability Assessment Process Flow Diagram. Highlight the areas of security that you addressed by refactoring the code.**

The primary area of security that was addressed by refactoring the code was **Cryptography** (according to the Vulnerability Assessment Process Flow Diagram). SHA-256 is a cryptographic hashing function to securely identify a file’s content. This is done by initially hashing the file’s/data’s content and obtaining the fixed checksum value (in this instance, it was a string input). Then this checksum value is provided to the user who can use the checksum value to verify the files contents are, in fact, untampered with. Specifically, the MessageDigest class is utilized to convert the string of data into a fixed length output checksum value. The MessageDigest class uses the java.security.MessageDigest library. This library doesn’t have any reported vulnerabilities noted according to the static test performed on the refactored code. Furthermore, the codebase utilizes **encapsulation** by separating the bytesToHex class created to convert the byte value to hexadecimal for the final checksum value. This prevents those who contact the webserver from accessing that particular section of code. Moreover, the refactored codebase utilizes secure **API**’**s** by ensuring proper HTTPS protocol when contacting the webserver. This is done by implementing a self-signed certificate to authenticate/validate any data transferred by the website.

**Discuss your process for adding layers of security to the software application.**

The codebase went through an iterative process to identify key vulnerabilities. The process began with identifying a secure cryptographic algorithm to fulfill the needs of the client. Specifically, Artemis Financial desired a way to verify files/data through a form of checksum. Thus, the SHA-256 hashing algorithm was chosen for this task.

The applicated first needed to be reached securely using HTTPS. HTTPS requires a hashing algorithm (such as RSAwithSHA256 for this scenerio). This algorithm has a private and public key. The private key is hidden inside the websever to decrypt information encrypted by the public key. The public key is available to everyone who interacts with the webserver. This key only allows for data decryption. This form of relationship is called asymmetric as there are two separate keys: a public and private key. This relationship is essential to HTTPS and implementing Public Key Infrastructure (PKI) security.

Then, a hashing algorithm is implemented to provide a checksum value for the given data. This is where SHA-256 hashing comes into play. The data is given to the algorithm, which is subsequently converted to a hash value, then output as a hexadecimal value per the bytesToHex function.

Once completed, the codebase underwent static testing to determine vulnerabilities detected in the declared libraries. This is useful to determine publicly known vulnerabilities for early mitigation/prevention. The code was refactored according to dangerous vulnerabilities detected in this test and manually found upon further review of the code.

## Industry Standard Best Practices

**Explain how you used industry standard best practices to maintain the software application’s current security.**

The cryptographic method chosen for the hashing function was specially selected for this scenario. SHA-256 has possible hash values. Thus, collisions are almost impossible. This is important as collisions can allow a malicious entity to make a supposed safe verified file (via the checksum value) appear correct yet have hidden malicious code underneath. This cryptographic hashing algorithm is considered one of the premiere industry standard cryptographic algorithms for checksum values. Furthermore, this application was tackled using a layered security approach. During each step of the process, a new layer of security was added to the web application. This includes the cryptographic hashing algorithm, the self-signed HTTPS certificate for secure connections, a static report created to indicate vulnerable libraries and potentially exploitable routes, and manual code review to ensure that the code quality achieves industry standard quality.

**Explain the value of applying industry standard best practices for secure coding to the company’s overall wellbeing.**

The aforementioned layers of security provide an in-depth approach to tackling general software security. These practices create a level of integrity and validity for the company image and brand. Not only does implementing strong application security mitigate external threats, it also increases trust between the company and the end users (whom are essential for the company’s very existence). Moreover, these software security strategies help mitigate major data breaches which could end up causing litigious action against the company for reckless behavior; especially in regard to a financial company such as the hypothetical Artemis Financial. Thus, the value of applying industry standard best practices for secure coding should be self-apparent.