****

# 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** | **12/9/23** | **Bryce Jensen** |  |

## 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

Bryce Jensen

## Algorithm Cipher

 Recommend an appropriate encryption algorithm cipher to deploy, given the security vulnerabilities, and justify your reasoning. Review the scenario and the supporting materials to support your recommendation. In your practices for secure software report, be sure to address the following:

1. Provide a brief, high-level overview of the encryption algorithm cipher.
2. Discuss the hash functions and bit levels of the cipher.
3. Explain the use of random numbers, symmetric versus non-symmetric keys, and so on.
4. Describe the history and current state of encryption algorithms.

It is recommended that AES-256 is used for Artemis Financials’ security needs. AES is a symmetric encryption algorithm that is widely accepted as one of the most secure block ciphers for encryption. NIST deemed AES as the standard for encryption in 2001. (Computer Security Resource Center (CSRC), 2023)

While AES is not a hash function, a separate hash function – sometimes SHA-256 – can be used with it to ensure the authenticity of transmitted data and ensure it was not tampered with or corrupted. The bit level of AES-256 determines the strength of security. This determines the length of the key, in this case, 256 bits, or 32 bytes. The reason I would recommend AES-256 over AES-128 is that Artemis Financial deals in the banking sector and the 256 version is far more secure. They also wouldn’t need to encrypt and decrypt data with any excessive speed, so the slower performance of AES-256 over the faster 128 would be fine in this instance.

Random numbers are what make key generation work. They need to be unpredictable to keep malicious attempts on breaking security. AES is a symmetric algorithm. This means that the same key is used for both encryption and decryption. This also means that they are faster and more efficient when handling large amounts of data encryption. Of course, this means that there needs to be a secure form of distributing the keys; otherwise – if intercepted – the encryption is useless.

In short, before AES was used as the standard for data encryption, the DES (Data Encryption Standard) was used. The DES used a 56-bit key size, which eventually became obsolete after determining that most computers today can break it pretty quickly (Simplilearn, 2023). Today, the standard is AES (Computer Security Resource Center (CSRC), 2023) and it is used in a variety of applications.

## Certificate Generation

Insert a screenshot below of the CER file.

A black screen with white text

Description automatically generated

## Deploy Cipher

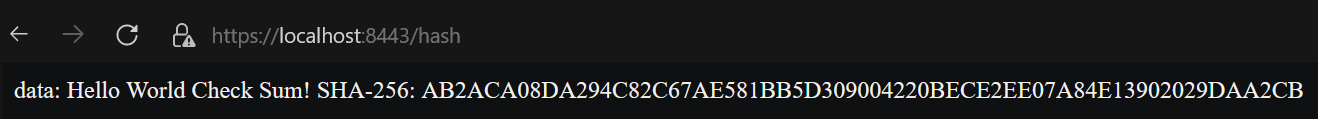
Insert a screenshot below of the checksum verification.

The code:

A screenshot of a computer program

Description automatically generated

The verification:



## Secure Communications

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

From what I understand it won’t display as secure because it is a self-signed certificate.

The certificate information:

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 black screen with white text

Description automatically generated

Dependency Check BEFORE the refactored code:

A screenshot of a computer

Description automatically generated

Dependency Check AFTER the refactored code:

A screen shot of a computer

Description automatically generated

Dependency check AFTER suppressions:

## Functional Testing

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

A group of lights on a black background

Description automatically generated

## Summary

To refactor the code and eliminate vulnerabilities, I updated the dependencies to their most recent versions. This allowed me to get the vulnerabilities down to 2 without suppressing them.

I also added the checksum verification of a string. The checksum is made with SHA-256 and has a VERY small chance (read: near impossible) of collisions.