****

# 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** | **10/22/2022** | **Malcolm McGee** |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## 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

Malcolm McGee

## Algorithm Cipher

First, I want to look at encryption algorithms to show their state of them and lead into what led me to my decision. When looking at encryption algorithms, we can have either a symmetric or asymmetric encryption algorithm. Symmetric encryption will use the same secret key in both encryption and decryption of the message. While asymmetric encryption will have a different key for both encryption and decryption. The public key is used to encrypt the message, and the private key is used to decrypt the message. The public key is derived from the private key but using a method to make it difficult to find the private key. ([Article about Encryption Algorithms], (n.d.)).

The next part of encryption that I want to look at is things like random numbers. Random numbers can help create another layer of security, making the encryption algorithm more difficult to solve. We can use random numbers in plaintext, and this is then either encrypted or hashed. It helps allow for a different layer in case the decryption keys are stolen, or someone can reverse the hashing process or encryption process. (Crane,2021).

The last fundamental part I want to discuss is bits and what role they play in encryption and hashing functions. A bit is considered the smallest piece of information and it is represented as a number. Because of this, the larger the bit length, the more numbers are present. Therefore, a 56-bit encryption protocol like Data Encryption Standard (DES) would be less secure than some 128-bit or more protocols. (ClickSSL, 2012).

With some processes explained, I can go into discussing hash functions and encryption. When looking at a hash function, is a process that takes some plaintext data of any size and converts it into a unique ciphertext that has a length specified by the bit length chosen. Hashing is considered a one-way function, unlike encryption. It doesn’t have the option to use a key to decrypt it and so we shouldn’t use it for things like security. Hash functions are a great option for us to ensure that data integrity is kept. It shows if the document was altered because it would create a new hash, and this helps us know if unauthorized modifications were created and alert us to those changes. When looking at options for hashing protocols, I chose SHA-256 because it has great collision resistance and a relatively fast speed, allowing us to complete the hashing quickly. If Artemis was to send any messages or emails, deploying a hashing function can be a useful part of security to alert us to any altered communications.

When looking at the history of Encryption algorithms, we can start with Data Encryption Standard (DES). This encryption algorithm was used by the federal government and was created by IBM using a 56-bit symmetric key. It was used from the 1970s until the National Institute of Standards and Technology (NIST) started the development of AES in 1997. The reason being was that DES was becoming vulnerable to brute-force attacks to decode it. While looking for a replacement option for DES, the NIST evaluated 15 different algorithms. The selection process was then narrowed down to 5 different algorithms and Rijndael was chosen to become the successor. Rijndael then was renamed to AES and was published by the U.S. NIST in 2002. The AES algorithm is approved by the NSA for handling top-secret information and the bit length requirements are very based on the classification of the material. There are three different AES lengths of 128-bit, 192-bit, and 256-bit. 256-bit is the longest key and provides the strongest level of encryption, but it comes with a tradeoff of speed. With most modern devices, the encryption speed is fast enough that I would still recommend AES 256 for our choice because of the greater security it provides. When looking at the bit size, it correlates to the number of rounds are done in the encryption. AES 128 has 10 rounds, AES 192 has 12 rounds, and AES 256 has 14 rounds. Each round will comprise several processing steps like substitution, mixing, and transposition of the input plaintext to transform it into our ciphertext. The more rounds, the more complex the encryption will be, and the more resources needed on the device to perform the encryption. AES is a symmetric encryption algorithm, so those rounds are done on just the plaintext input and provides our one key. (N-able, 2019).

Artemis Financial provides individual investment plans and this data could be seen as very valuable for third parties. With that in mind, one of the most important aspects of keeping that data safe will be security and our Algorithm Cipher will play a large role in that. The cipher that I would recommend is Advanced Encryption Standard (AES) as the best option and I would choose AES 256-bit. “With the right quantum computer, AES-128 would take about 2.61\*10^12 years to crack, while AES-256 would take 2.29\*10^32 years.” (Tobias, 2021). With that in mind, if we properly implement AES and keep the keys secure it would be one of the best options for security for Artemis Financial. (Tobias, 2021).

## Certificate Generation

Insert a screenshot below of the CER file.

Graphical user interface, text, application

Description automatically generated

## Deploy Cipher

This shows the checksum verification being completed

Graphical user interface, text

Description automatically generated with medium confidence

## Secure Communications

This shows the connection is secure and it is using https security

Text

Description automatically generated

## Secondary Testing

This is a screenshot of my code running and the output in the bottom console.

A picture containing text, screenshot, computer, computer

Description automatically generated

This is a screenshot of the first dependency check so I can verify no other dependencies were introduced.

Graphical user interface, text, application

Description automatically generated

This is the second dependency check showing no additional dependencies were introduced.

Graphical user interface

Description automatically generated

## Functional Testing

A picture containing text, screenshot, computer, computer

Description automatically generated

When looking through the code, the main vulnerabilities I see are coming from our use of outdated dependencies. While looking through the dependency report that I will attach, it shows that for the dependencies, one of the best solutions we can implement is updating to later versions. Another security vulnerability is having the key-store-password directly in the file. This is seen in the application.properties file on line 5. A recommendation that I would make is for us to find a way to not have it present or ensure that the file is secured. Text

Description automatically generated

## Summary

When looking at the Vulnerability Assessment Process Flow Diagram, there are certain parts that I want to go over and how we met them. When writing the code, I ensured that our hash function was implemented properly and ensured that the output was correct by checking the output to other SHA-256 encryption implementations. This helped ensure that anything we run through the program for hashing produces the correct output. I didn’t hard code any type of user input into the fields or pass user input into applications which can allow for SQL injections and other forms of attacks. I ensured that the client/server was secure by changing the application.properties file to check for the certification and, if valid, continue running the hash function. The certification was signed and had SAN setup, so we can test the HTTPS connection and verify it works properly for the application. I also checked my code quality by ensuring no additional dependencies were added after checking the code. I made sure that the code was executed properly after editing with no errors as well.

When looking at our layers of security, we had a couple that helped ensure the site was secure. The first one was our self-signed certificate. This allowed users to know that the site they were accessing was the correct site by checking the signed certificate. This can act as an added value in that users can trust communications that are sent to the site are the correct site and trust us with important financial information that they are sending. The next layer of security I want to look at is our SHA-256 hashing cipher. The cipher shows the users that any communications we send to them or vice versa aren’t altered. They or we can verify the cipher and if we notice any changes, it serves to alert us to a system vulnerability or unauthorized access.

When looking at ways to add layers of security to the application, the first one I want to look at is encryption. When Artemis adds encryption in the future, we need to ensure that we keep the encryption keys in a safe location and aren’t on a public site that is just obfuscated through a different site that is hard to find. Next, we need to ensure that we continue to run the dependency check and update our dependencies when applicable. One of the best ways for finding vulnerabilities and ensuring they no longer affect our application is to update to known versions that patch that vulnerability.

## Industry Standard Best Practices

When looking at secure coding, covers things like error handling, logic flaws, input validation, and more. When creating my code, I understood that following the industry standard and best practices would help me create code that can continue to be used. One way that I did this was my way of handling data in the output. I tried to store most data in the output into a container so that if we need to change things like the name or the item being hashed, it’s easier to change the one container instead of looking throughout the code. This helps make it more modular while also allowing it to be more easily checked to find any errors. Another aspect was when creating the functions. I wanted to approach making our hashing function as simple as possible while also having it well documented. This allows others to understand my process in making the function as well as allows them to make any necessary edits and changes when auditing or implementing it. The code doesn’t have any type of injections that the user is passing through as input and we also can use the containers made for things like name or data to sanitize any user input in the future. This helps us further follow secure practices and helps protect us from things like input injection. Another major part of industry-standard best practices is to have secure communication, and we did this by ensuring that all communications are through HTTPS. By creating the self-signed certificate and editing the application.properties file to use HTTPS, we help secure any communications to a user from the site and vice versa.

By us using industry-standard best practices allows our code to be secure. When looking at security, it shouldn’t be an effort in the end, but we should create our software with security in mind from the beginning. By following industry standards, we are attempting to ensure that security is at the forefront of our application.

References:

Bernstein, C. and Cobb M. (n.d.) *Advanced Encryption Standard (AES)*

<https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard>

Crane, C. (2021, January 25) *What Is a Hash Function in Cryptography? A Beginner’s Guide*

<https://www.thesslstore.com/blog/what-is-a-hash-function-in-cryptography-a-beginners-guide/>

ClickSSL (2012, January 27) *What is bit encryption, and how does it work?*

<https://www.clickssl.net/blog/what-is-bit-encryption-and-how-does-it-work>

N-Able (2019, July 29) *Understanding AES 256 Encryption*

<https://www.n-able.com/blog/aes-256-encryption-algorithm>

Tobias, E. (2021, February 15) *128 or 256 bit Encryption: Which Should I Use?*

https://www.ubiqsecurity.com/128bit-or-256bit-encryption-which-to-use/

[Article about Encryption Algorithms] (n.d.) *Types of Encryption Algorithms + Pros and Cons for Each*

<https://www.keyfactor.com/resources/types-of-encryption-algorithmns/>