****

# 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** | **2/14/2023** | **Richard Howell** | **Final Edit** |

## 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

Richard Howell

## Algorithm Cipher

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

I have chosen to implement the AES encryption method to safeguard Artemis Financials’ files and other information the company uses. AES is known as the Advanced Encryption Method and is the standard for the United States Government, military, as well as many financial institutions to comply with government regulations. AES utilizes 128-bit, 196-bit, and 256-bit key combinations that are considered unbreakable by the U.S. Government and other entities. A brute force attack that is trying to break AES utilizing a 128-bit key would take 13.75 billion years with current computing abilities and this increases exponentially when using 196-bit and 256-bit keys (Kryptall Secure Communication Services, n.d.). I have chosen the SHA-256 hash function for this specific scenario applying to Artemis Financial.

The hash function takes raw data input of any size and produces a fixed-length enciphered text. Hash functions are used to ensure data integrity, verify digital signatures, and facilitating secure password storage (Mehta, 2020). An example of this is a password that has been stored using a hash value. Securing a plane text password for users would be unsecure. Once the password is created the hash value, not the plain text, is stored and the only way to replicate that hash value is to have the exact same input. If just one character is off a different hash value will be created and the password will not be accepted. Bit levels are used to describe the order of magnitude of the resources required to break the security (Staff, 2020). AES with a bit level of 256 would require a potential attacker to do 2^256 computations to reverse engineer the cipher text into plain text (Staff, 2020). The main difference between hash functions and bit levels is that hash functions cannot be reverse engineered back into plain text and can only be replicated by inputting the exact same string whether that is one word or entire paragraphs. Bit levels can be unencrypted with the use of a secure key and are a way to transmit data securely.

The use of random numbers or random number generators (RNG) is needed in the key generation process to create secure and random keys (Scholten, n.d.). This ensures that the keys being used are truly random and cannot be simply guessed by potential attackers leaving information being stored or transmitted vulnerable. Symmetric keys use the same secure key (secret key) to encrypt and decrypt the data being transmitted or stored (Manico & Detlefsen, 2014). Non-symmetric keys on the other hand use public key cryptography. This is essential to the signing process. Non-symmetric or public key cryptography uses two different keys. The private key is known only to the sender and is kept private. This key is used to sign the data being transmitted. The public key is known by anyone and is used to validate the signature of the private key. There is no known way to obtain the private key from the public key currently with modern computing abilities (Manico & Detlefsen, 2014).

The first known evidence of cryptography was found in an inscription from ancient Egypt and was believed to be transcribed around the year 1900 BC (A Brief History of Cryptography, 2023). This was not used to hide the message but was used to make the message appear more dignified. If we fast forward around 1800 years to the time of the roman empire and Julies Caesar forms of encryption were used to send secret messages across the battlefield to his generals with orders and directions. This and different forms of written cypher texts were used to secretly transmit information all the way until around the 19th century. This is when rotor machines such as the Herbern rotor machine and Enigma machines were used to transmit secret messages through radios. A specific key was needed to encrypt and decrypt the message (A Brief History of Cryptography, 2023). Modern day computer encryption was started in 1973 with a cipher called Lucifer and was created by IBM. The NIST (Nation Bureau of Standards) then put out a request for proposals for a national standard block cipher and would later adapt the Lucifer encryption method to DES (Data Encryption Standard) (A Brief History of Cryptography, 2023). DES was eventually broken because of the increase in computing power and the NIST once again put out a request for a new proposal. “It received 50 submissions. In 2000, it accepted Rijndael, and christened it as AES or the Advanced Encryption Standard. Today AES is a widely accepted standard used for symmetric encryption” (A Brief History of Cryptography, 2023).

## Certificate Generation

* Insert a screenshot below of the CER file.

Text

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

## Deploy Cipher

* Graphical user interface, text, application

  Description automatically generatedInsert a screenshot below of the checksum verification.

## Secure Communications

* Graphical user interface, text, application

  Description automatically generatedInsert a screenshot below of the web browser that shows a secure webpage.

## Secondary Testing

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

* A screenshot of the refactored code executed without errors.

Graphical user interface, application

Description automatically generatedGraphical user interface, text

Description automatically generated

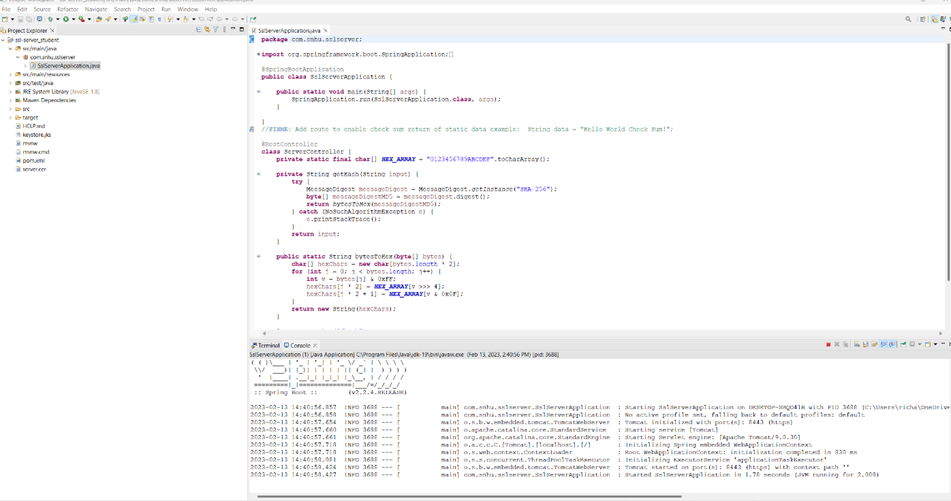
* A screenshot of the report of the output from the dependency-check static tester

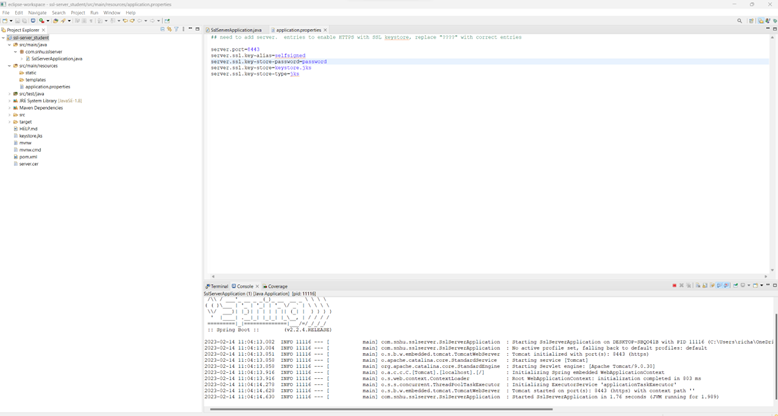
Graphical user interface, text, application

Description automatically generated

## Functional Testing

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





## Summary

* Refer to the Vulnerability Assessment Process Flow Diagram. Highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application.

The first area of security that I addressed from the Vulnerability Assessment Process Flow Diagram was cryptography. I refactored the code to include a hash function, specifically the SHA-256 hash function, to encrypt the data transmitted and stored by Artemis Financial. I also created a secure RESTful controller which enabled a more secure API and addressed that portion of the vulnerability assessment flow diagram. The client/server connection and security were improved by adding a certificate that used a self-signed key which provided a secure connection for both the client and server. This gives both Artemis Financial and its users more secure connections as well as providing a sense of security provided by HTTPS. I also attempted to update the spring-boot to a newer version but to ensure I was running the most secure versions of the software, but I was unable to get the Maven dependencies file updated withing my project. I was however able to update the dependency check to a much newer version of 6.4.1. This enabled me to see more potential risks and vulnerabilities and potential solutions. The process I used to add secure layers to the software application was the use of the hash function. A secure RESTful API as well as a self-signed certificate and keystore. This provides a defense in depth strategy and provides the best layers of security possible for the Artemis Financial software application. We should also ensure that we run a dependency check on a routine basis to ensure that new threats do not propagate as well. The version of spring-boot should also be updated at the earliest time possible to ensure that the software application is running on the newest available software that is as free of bugs and vulnerabilities as possible. Using all the strategies above I believe we have created a viable layering of defenses for Artemis financial and their needs for the application.

## Industry Standard Best Practices

* Explain how you used industry standard best practices to maintain the software application’s current security.
* Explain the value of applying industry standard best practices for secure coding to the company’s overall wellbeing.

I used the industry standard defined by the NIST for the hash algorithm. The SHA-256-bit hash function is the best available at the current time and is virtually unbreakable. The U.S. government and military currently uses this for secure storage and transmission of information. I also tried to update the spring-boot and maven dependency check to the latest available versions. This keeps the software used by Artemis Financial up to date in compliance with the industry standard. Using a keystore as well as a self-signed certificate also keeps the server/client safe as well. I took all of these steps to try and be as close to the industry standard best as possible. All these steps combined ensure that Artemis Financials software application will be as secure as possible while still maintaining the industry standard.

Applying these industry standards will help Artemis Financials over all well being by helping them comply with all federally mandated regulations to the financial industry. “The Gramm-Leach-Bliley Act requires financial institutions – companies that offer consumers financial products or services like loans, financial or investment advice, or insurance – to explain their information-sharing practices to their customers and to safeguard sensitive data” (Gramm-Leach-Bliley Act, 2022b). By using the SHA-256-bit hash function we ensured that Artemis Financial is in compliance with this regulation. The PCI DSS (Payment Card Industry Data Security Standard) states that all companies that accept, store, process, or transmit credit card information must maintain a secure environment (PCI Compliance Guide, 2017). By using the industry’s best standards, we were also able to meet this federal regulation as well. Overall, by using the industry’s best standards we can protect Artemis Financial from any possible sanctions, fines, penalties, and loss of consumer confidence. If there were to be a security breach which resulted in a fine, penalty, etc. Artemis financial would surely lose customers and potential revenue. By maintaining the industry best standards outlined above we can do our best to avoid this and provide a quality service to both Artemis Financial and their customers.

**Citations**

*A Brief History of Cryptography*. (2023, January 12).

https://www.redhat.com/en/blog/brief-history-cryptography

*Gramm-Leach-Bliley Act*. (2022a, February 11). Federal Trade Commission.  https://www.ftc.gov/business-guidance/privacy-security/gramm-leach-bliley-act

Kryptall Secure Communication Services. (n.d.). *How Safe is AES Encryption?* https://www.kryptall.com/index.php/information/how-safe-is-aes-encryption

Manico, J., & Detlefsen, A. (2014). *Iron-Clad Java: Building Secure Web Applications*. McGraw-Hill Education.

Mehta, M. (2020, December 4). *Hash Function in Cryptography: How Does It Work?* InfoSec Insights. https://sectigostore.com/blog/hash-function-in-cryptography-how-does-it-work/

PCI Compliance Guide. (2017, September 5). *PCI FAQs*. https://www.pcicomplianceguide.org/faq/

Scholten, U. S. P. &. (n.d.). *The Role of Random Number Generators in Relation to HSMs & Key Management*. https://www.cryptomathic.com/news-events/blog/the-role-of-random-number -generators-in-relation-to-hsms-key-management

Staff, W. (2020, June 6). *The Bit-Security of Cryptographic Primitives*. AWS Wickr. https://wickr.com/the-bit-security-of-cryptographic-primitives-2/