****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/08/2021** | **Kristofer Phillips** |  |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Kristofer Phillips

## 1. Algorithm Cipher

Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

* 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 vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

|  |  |
| --- | --- |
| Algorithm Cipher Recommendation: | AES-256 |
| Overview: | Advanced Encryption Standard 256-bit (AES) is the recommended encryption algorithm for Artemis Financials’ software implementation. AES is the de facto standard of the financial industry due to the strength of its encryption and the open-source support it enjoys from the Cybersecurity community. In addition, AES satisfies regulatory requirements of GBLA, FFIEC, and PCI DSS. While AES may require an increased cost for the infrastructure to support, the compliance with industry regulations and risk of financial loss and loss of consumer confidence outweigh the financial impact of AES implementation. |
| Hash Function and Bit Levels: | AES supports 128, 192, and 256-bit encryption levels. The levels of encryption determine how many bits to assign per block of encryption. During block encryption, the original key is used to generate new round keys and those keys make it harder to decrypt the data without the private key. In each block, each byte of data is substituted with another, based on a predefined table. The use of multiple keys and encryption by block produces data indecipherable without the private key. (*Understanding AES 256 Encryption*, 2021) |
| Use of Random Numbers, Symmetric vs Non-Symmetric Keys: | * Random numbers are added to data during the encryption process called “padding” where the additional message has nonsensical data added to it. By integrating numbers into the padding, the message becomes exponentially more difficult to decipher without the private key. (Abellán & Pruneri, 2021) * The algorithm is important in determining how to obfuscate and scramble data. An example of a simple algorithm would be to take the numerical place of a letter in the alphabet and shift it a predetermined number of spaces to the right or left. * Keys are the culmination of the algorithm obfuscating data and the padding of the data; providing a method to deliver encrypted data to its destination and a method to decrypt it. There are two types of key systems: symmetric and asymmetric. Symmetric encryptions use a single key to encrypt/decrypt data while asymmetric systems have a public key to encrypt the data and a private key for decryption. (Abellán & Pruneri, 2021) |
| History and Current State of Algorithm Ciphers: | * Modern computer-based encryption began with IBM, which developed a block cipher to protect its customers’ data. (*A Brief History of Encryption*, 2016) * In 1973, the United States adopted IBM’s block cipher as a standard and named it Data Encryption Standard (DES). (*A Brief History of Encryption*, 2016) * In 1976, Whitfield Diffie and Martin Hellman produced a white paper which would establish the Diffie-Hellman key exchange. This would lay the foundation for Public Key Infrastructure (PKI). (*A Brief History of Encryption*, 2016) * In 2000, Advanced Encryption Standard (AES) replaces DES and becomes the United States government standard, utilized in securing classified information. (*A Brief History of Encryption*, 2016) * In 2005, Ecliptic-curve Cryptography (ECC) improves PKI and reduces the computation power and is more secure and harder to break than RSA and Diffie-Hellman. (*A Brief History of Encryption*, 2016) * Today: Encryption is widely employed across the financial industry, social media messaging platforms, and in cryptocurrency. AES remains the de facto standard for encryption across most industries. |

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

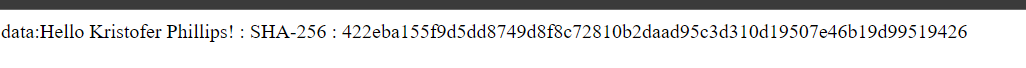
Text

Description automatically generated

## 3. Deploy Cipher

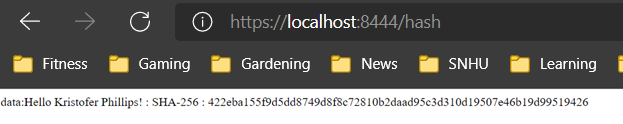
Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.



## 5. Secondary Testing

Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.

Text

Description automatically generated

## 7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

* Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
* Point out best practices for maintaining the current security of the software application to your customer.

|  |  |  |
| --- | --- | --- |
| ServerController (Rest API) | Secure Coding | By adding the RESTful API map to the hash and using the text variable to generate a SHA-256 hash, this code satisfies the Secure Coding aspect of the vulnerability assessment. |
| SSL Certificate | Cryptography  Client/Server | By implementing a SSL certificate, secure communication over TLS is enabled between client and server. This satisfies both cryptography and client/server aspects of the vulnerability assessment. |

In addition to refactoring the code to enable secure communication, I also updated all dependencies listed that had vulnerabilities. I was able to eliminate all but one of the entries listed in the report by updating the POM file with the latest versions of those dependencies.

The only remaining vulnerability in the report was for Tomcat Embed Logging Log4J 9.0.0.M6. That is the current version of the dependency and remediating the actual code inside of it is out of scope of this project.

A picture containing timeline

Description automatically generated

Keeping dependencies and APIs up to date ensures that old vulnerabilities are not exploited and limits the threat vector to zero-day vulnerabilities.

Best practices for maintaining the security of the application:

* Keep up to date on current legislation that applies to the customer’s industry. Some examples are Sarbanes-Oxley, and the Payment Card Industry Data Security Standards (PCI-DSS). Changes to these regulations or new regulations may require refactoring or new APIs to maintain compliance.
* Adopt a nightly security testing (or weekly if code changes are not as frequent) regimen and be prepared to remediate vulnerabilities listed in the dependency report.
* User input validation is one of the most important security areas that contribute to threats like Cross Site Scripting and SQL Injection. Developers should develop in the mindset of a “zero trust” environment where the input from the end user should not be trusted and should be validated.

## 8. Sources Cited

Abellán, C., & Pruneri, V. (2021, July 29). *The Future of Cybersecurity Is the Quantum Random Number Generator*. IEEE Spectrum. Retrieved November 21, 2021, from https://spectrum.ieee.org/the-future-of-cybersecurity-is-the-quantum-random-number-generator#toggle-gdpr

*A brief history of encryption*. (2016, April 18). Thales Group. Retrieved November 21, 2021, from https://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-history-encryption

*Understanding AES 256 Encryption*. (2021, April 10). N-Able. Retrieved November 21, 2021, from https://www.n-able.com/blog/aes-256-encryption-algorithm