****

# 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** | **April 14, 2022** | **Johnny Lingafelter** | **Initial Revision** |

## 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

Johnny Lingafelter

## 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.

I recommend AES as the cipher algorithm for Artemis Financial’s file encryption needs. AES has been used by the US government for 20 years, and it is superior to the DES cipher (Westlund, 2002). AES is standardized (ISO/IEC 18033-3) and available publicly in different encryption packages. I do believe this is the best cipher in use today, because it has been in wide use for decades without being “hacked” and because it is adopted by many institutions and governments to secure their data. The only reason I see to not choose the most secure cipher is for testing or learning purposes, or in a situation where the data is not sensitive.

The bit level of a cipher is related to its strength. When a block of plaintext is encrypted, it undergoes multiple “passes” or transformations depending on the bit level used. 10 rounds are used for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys (Wikipedia contributors, 2022). The more transformations the block undergoes, the more difficult it is to crack, and the more secure it is. A cryptographic hash function takes input of variable length, usually in the form of plaintext or binary, and outputs a fixed length string, or “hash”. This is useful for validating that a message or program is received the way it was intended, as any change to the item in question would produce a different hash, signifying that the piece of information was altered from its original version.

Random numbers are paramount in the field of encryption and cryptography. A cryptographic pseudorandom number generator (CPRNG) is used to provide a computer with the means to randomize the encryption process (Manico & Detlefsen, 2014). These random number generators must be seeded with a different number with every use in order to provide truly random results. This ensures that an attacker cannot just undo the encryption by using the same CPRNG that was used to encrypt initially. Symmetric cipher algorithms use the same key to both encrypt and decrypt a message. Only the key must be kept private, all else about the encryption process should be well know. Non-symmetric ciphers use a public key and a private key. The most notable use for this is CA certificates that are signed with a private key and verified with a public key that anyone can access. This ensures that a central authority which is used by a number of different is who they say they are.

Cryptography has been in use long before computers existed. In its most basic form, encrypting anything can be as simple as using a set of symbols or scrambled letters to write something, and possessing a “key” that matches an encrypted symbol to the decrypted meaning. As computing became more popular and powerful, this concept has evolved into something enormously complex and secure. Machines can do all the work for us now, and cipher algorithms will always be produced to perform securely and in line with current technology. The main aspect of cryptography that has evolved with computers is the bit level of the key used. When computers were first invented, only small keys could be used because that was all that the machines could handle. As computing evolved the bit level of keys became longer and more secure.

## 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

Graphical user interface, text, application, email

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.

A screenshot of a computer

Description automatically generated

## 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.

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

A screenshot of a computer

Description automatically generated

## 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, application

Description automatically generated

Text

Description automatically generated

[Note] : As you can see I have refactored the application.properties file to use the certificate that I generated manually. There is a problem with self-signed certificates and localhost that still shows that the site is “insecure”. This can be fixed manually with some tweaking as I have read, but I have gone ahead and assumed that this is out of the scope of this project.

Graphical user interface, text

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.

Graphical user interface, text, application

Description automatically generated

I found no syntactical or logical errors in the code. However, each package listed in the dependency check report should be upgraded to the latest version to address the security vulnerabilities present in the codebase.

## 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.

The areas of security that I have addressed by refactoring the code are:

* APIs – executing dependency checks to ensure that the codebase is up to date and free of any security vulnerabilities
* Cryptography – using the SHA-256 cipher to create a hash of a string ensures that the data being transferred cannot be read by outside attackers
* Client/Server – using HTTPS instead of HTTP ensures that communication between the client and server is secure and encrypted
* Code Quality – reviewing the refactored code for logical errors to ensure that the program behaves in the correct way

The process of adding layers of security to this application include ensuring that a reliable cipher algorithm is used to encrypt the data, and that a secure connection is used between the client and server. To do this I used the SHA-256 algorithm and MessageDigest object (within Java) to encrypt text, and I also refactored the application.properties file to ensure that HTTPS was being used as opposed to HTTP. Next, I checked the codebase for security vulnerabilities using the OSWAP dependency check tool provided by Maven and recommend upgraded any and all packages (aside from false positives, of course) that appear in the dependency-check-report. Finally, I inspected the refactored code for syntactical and logical errors that could compromise the application.

The single most important security practice that I would convey to the customer is, in my opinion, running regular (daily or weekly) dependency checks against the codebase and immediately upgrading any packages that are out of date. Another security practice I would recommend would be to review any new code that is added to the codebase thoroughly looking for errors in logic and syntax that may not be caught by the compiler but could pose a threat to the application as a whole. Finally, I recommend that the certificate used by the server be kept up to date and never be allowed to expire (there should be a person assigned to this task).

**References**

Westlund, H. B. (2002). NIST reports measurable success of Advanced Encryption Standard. *Journal of Research of the National Institute of Standards and Technology.* https://web.archive.org/web/20071101081451/http://findarticles.com/p/articles/mi\_m0IKZ/is\_3\_107/ai\_90984479/pg\_1

Wikipedia contributors. (2022, February 18). Advanced Encryption Standard. In *Wikipedia, The Free Encyclopedia*. Retrieved 04:12, March 24, 2022, from <https://en.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard&oldid=1072495414>

Manico, J., & Detlefsen, A. (2014). *Iron-Clad Java*. McGraw-Hill Education.