****

# 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** | **12/10/2021** | **Justin Haby** |  |

## 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

[insert name here]

## 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.

Firstly SHA, AES and RSA are three different types of encryptions**. RSA:** - It is an asymmetric cryptography; it uses a key to encrypt data and then uses a different key for decryption. These are normally called a public key and a private key. The public key can be used to encrypt some data and then it would require a private key for its decryption**. AES**: It is a symmetric cryptography; it uses same key for both encryption and decryption. and **SHA**: It is a hash algorithm, one way encryption. So that it gives no way for decryption.

Normally things like ‘passwords’ are kept using SHA algorithm in databases or other persistent form. Whereas files or text-data are encrypted using AES or RSA algorithm. This is because files or text-data are normally read by human and hence they must be decrypted for using. Passwords are used by computers to match and hence it works without being decrypted.

For this instance, I thought it was in the best interest of Artemis Financial to implement **Sha-256**for the purpose of providing high level security due to the features is possessed such as, Digital signatures follow asymmetric encryption methodology to verify the authenticity of a document/file. Hash algorithms like SHA 256 go a long way in ensuring the verification of the signature. It helps foster a sense of privacy, and it lessens the load on the central database since all the digests are of similar size. It helps by maintaining integrity and the full value functionality of files and makes sure they were not altered in transit.

## 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.

The Main importance and value of utilizing Certificate Authorities are to ensure secure communication across the internet, in this case, providing a pivotal role in digital security. It's vital to obtain the proper validation and approach when verifying domains across the web to ensure legitimate websites avoid un-secure browsers. Overall, utilizing certificate authorities is essential as it helps to facilitate encryption while also confirming the owners of the public keys themselves.

![Text

Description automatically generated]()![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.

![Text

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.

[Insert screenshot(s) here.]

## 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

![Text, email

Description automatically generated]()

![Graphical user interface

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]()

**![Table

Description automatically generated]()Suppression.xml File**

**![Graphical user interface, text

Description automatically generated]()Dependency Check 2**

## 

## 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.

For the company's overall well-being and to eliminate and keep up with any security vulnerabilities. I have chosen to work with the SHA-256 hashing cipher as it's very secure and runs a minimal chance at collisions. While maintaining a regular check within the applications, it's vital to review the dependency report to suppress all known vulnerabilities and resolve any false positives.**Keeping exposures in mind when developing is essential.**We can keep applications secure and up to date by patching security holes. We can find these holes by manually inspecting and using tools like the Maven Dependency-Check to best maintain the current security of the application.

**References:**

**Dependency checks through MAVEN**

<https://www.securityinternal.com/2016/10/owasp-dependency-check-cli-analyzing.html#:~:text=Then%20you%20can%20use%20the%20suppress%20option%20in,popup%20a%20message%20that%20contains%20some%20XML%20content.>

<https://improveandrepeat.com/2018/09/finding-security-vulnerabilities-in-your-dependencies-with-dependency-check/>

**SHA-256**

<https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm>

<https://en.bitcoinwiki.org/wiki/SHA-256>