****

# 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** | **10/14/2020** | **Felix Vargas** | **Security Report** |

## 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

Felix Vargas

## 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.

The best encryption algorithm cipher to use will be AES cipher due to its support of a variety of key sizes and being one of the best standards in place today. The 128-bit or 256-bit being the likely choice for this application with 256-bit being extremely difficult for attackers to crack. Its symmetric keys creation will allow the application to encrypt data as needed and deliver keys to its clients which are the recipients of communications.

The use of symmetric and non-symmetric keys varies on what the application would need to accomplish. Symmetric keys are basically shared between the sever and the client where non-symmetric keys include public and private keys with public keys being known to the public and private keys being known only to the client. These keys are used to encrypt data and de-crypt only when the correct key is used. A downside to this is if the key to any encrypted data is lost, the data is pretty much lost as well since there will be no way to decrypt that data without the key. Random numbers generators can be used to provide a unique identifier to transactions which might help with identifying certain events like data transfer or communications took place.

Currently, AES can encrypt using up to 256-bit which is virtually immune to being cracked as the number of different values that can be created is large. For this application, we utilized 128-bit encryption which should provide secure communication between the server and client.

## 

## 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.

Graphical user interface, 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.

Graphical user interface, text, application, email

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, application

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.

Graphical user interface, text, application, email

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.

The refactored code is an attachment in zipped file called “CS 305 Project Two Code Base.zip\_expanded”. By refactoring the code, we have addressed the following areas of security; APIs, cryptography, client/server, and code quality. We created our own self signed certificate and generated keys used for this application which allowed us to connect with 128-bit AES encryption. This encryption makes it so just the intended recipients will be able to read the data which will provide security to the applications communications. Secure communications are very important to maintain in this application as unsecure communications can cause data to be seen by an attacker and they can steal confidential or personal information. The results of a security breach can cause (but not limited to) a loss of trust between the application’s owner and the consumer, financial loss due to fines incurred by governments requiring secure communications, and financial loss due to resources spent repairing or addressing the security issue. Maintaining security in the application will protect the company’s products and assets and keep their clients trust and business. The best practices to maintain current security is to frequently check the code for vulnerabilities, especially after implementing new functions or changing current ones and before publishing or making those changes over to the live code. If any new vulnerabilities are found during development of new or updating of functions (before it is live), the development team should work on removing those vulnerabilities. For any vulnerabilities that have no solutions at the time, they should decide what will work best and determine if those vulnerabilities impact any elements that the application is using. If the team learns that vulnerabilities exists for any dependencies used but do not impact any elements used by the code, they can suppress those in their reports. For any vulnerabilities that are already in the code, the development team will have to look out for updates or solutions.