****

# 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** | **02/20/2022** | **Joshua Kovacevich** | **Final 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

Joshua Kovacevich

## 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.

For this project I decided to use the SHA-512 Encryption algorithm. SHA-512 was chosen for its capabilities and resistance to attacks. The algorithm outputs 512 bits (512 Bytes) of data. This algorithm actually works faster on 64-bit machines as it “has 37.5% less rounds per byte (80 rounds operating on 128-byte blocks) compared to SHA256 (64 rounds operating on 64-byte blocks), where the operations use 64-bit integer arithmetic.” (Gueron et al., n.d.) This encryption algorithm also has no known collisions and may very well have no collisions making attacking the algorithm via this route infeasible at the current time. Lack of collisions is also important for actual messaging between two computers as it allows verification that the message has not been tampered with enroute.

The function computes it’s hash with eight 64-bit words. In this function each round of encryption uses the previous rounds data to encrypt the next round. This effectively means that a seemingly minor change to a message (like adding punctuation to the end) can result in a wildly different hash, which reduces the likelihood of collision and reduces the likelihood that a message can be decrypted. An example of this would be you sending a message “Hello Mom!”, an attacker that may guess you are greeting your mother may attempt different phrases to match the hash like “Hello MoM” or “Hi Mom!” or even “Hello Mom.” Yet their hash would be drastically different than your intercepted hash.

Another benefit of encryption in general is the using of random (or as close as a computer can reasonably get) numbers to encrypt messages. The computer (generally) pulls from a list of system properties that are likely to be different based on numerous factors and then by combining a series of these random properties can build a number that would be near impossible to guess. This number is then used as a base key for encrypting making decrypting the message without this specific key the message is near impossible to guess (after approaching a size of about 16 characters). While SHA-512 is a hashing function rather than a standard encryption function it allows verification of information through checksums and the match of a hash without actually knowing the contents of the hash is important in terms of safe storage of data. As far as symmetric and asymmetric keys go, symmetric keys involve using one key distributed to both parties to encrypt and decrypt messages with no other party knowing said key. In asymmetric encryption, keys are typically made with a mix of public and private keys like in the Diffie-Hellman key exchange.

Encryption algorithms are in a constant war for security and safety. Encryption in various forms has been used throughout human history, with ciphers known only to certain people for secure messages as far back as humans have sent messages. In World War 2 for example the enigma machine encrypted messages in a way that any German commander could decrypt and read a message but if a message was intercepted or stolen the message would effectively be gibberish. Carrying that on now encryption algorithms have gained strength but follow the same general principles, make any message unreadable without a key. The major difference from the past to now is that computers enable the creation of extremely complex keys that a human would generally never be able to crack without the assistance of a computer, and that other computers find difficult due to the size and complexity of the keys themselves, making effectively impossible at the current time. As computing power increases these calculations will become easier forcing algorithms to become more complex to provide security. An example of this would be a commonly used form of human encryption, letter shifting, while, difficult (or at least time consuming) for a human to decrypt, a computer could cycle through 26 letter shifts within a matter of seconds, even including mixed shifting methods like shifting a letter a different amount based off of its encrypted position.

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Key tool, 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.

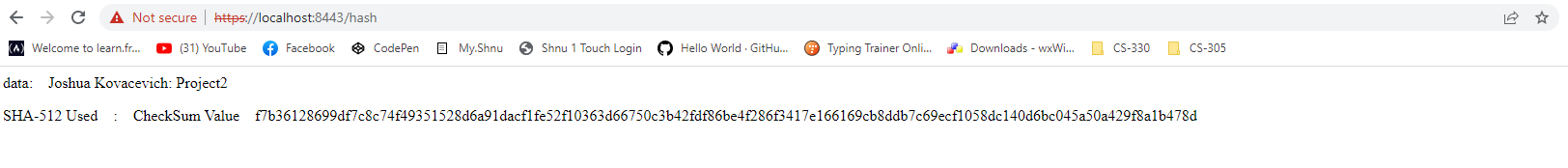
Text

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.

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

Graphical user interface, text, application

Description automatically generated

Despite my efforts I could not get chrome to trust my certificate because it was issued to/by the same authority.

## 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 generatedA picture containing table

Description automatically generated

The dependency vulnerabilities have not changed from the initial project.

## 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

## 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.

In this project we focused on the use of Cryptography, secure client/server interaction, and code errors primarily. In this refactoring the secure client/server interaction is provided by the refactoring to HTTPS with the use of a self-signed certificate. This certificate allows secure communication between the client and server, as the client is provided proof the server is who it says it is. The cryptography portion of the refactoring is in the form of SHA-512 encryption/hashing, using a checksum to verify the message sent is the same message received. This checksum is a unique value generated off the initial message sent prior to encryption. Finally, the Secure error handling is done through the use of exceptions, where issues that arise in the code are handled with an exception handler rather than allowed to exist and not be caught. Another factor of this is ensuring the code executes without errors, such as a process not starting appropriately, which potentially could leave an exploitable vulnerability. For example, I was initially having issues because I had the keystore.jks file in the wrong directory leading to the server being unable to run in HTTPS. Luckily this caused the program to stop rather than allow communication on HTTP, which if that had happened could have caused a potential leak of information. Another error that could happen would be incorrect Request Mapping leading to a user being able to see the wrong page when visiting the website. While this would not be significant in this project, in a real-world project this could lead to a user discovering information that was not intended for the general public. Like a user entering /hash and ending up on a page like /hashAdmin without the required security checks.

Adding security and enabling the HTTPS protocol involved generating a keystore and certificate and then using these to open a secure channel. Without adding these to the properties file the website was enabled on HTTP which is not secure and can lead to man in the middle attacks allowing the theft of information from a user or the server itself. This secure communication ensures that the information being passed from client to server and server to client is direct and does not allow for a man in the middle attack scheme to be effective, especially in a modern web-browser.

To maintain this level of security the certificate must be maintained and updated when due. To improve the security of this site the client should use a third party to hold and validate this certificate (A certificate authority) so that browsers will know to trust the website itself (as chrome for example requires the certificate to be issued by a certificate authority without significant changes to security settings). The customer should continue to monitor the vulnerability report as well as new vulnerabilities can appear at any time. Updating to the most current spring framework would be a great step towards reducing the existing vulnerabilities. Additionally, properly suppressing known but accepted vulnerabilities will help in this regard as a cluttered vulnerability report will lead to more issues as security professionals will have to sift through known dependencies to try to find the new ones when they occur.

References:

Gueron, S., Johnson, S., & Walker, J. (n.d.). https://eprint.iacr.org/2010/548.pdf

‌