****

# 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** | **2/18/2022** | **Jarod Timms** | **Final Project 2 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

Jarod Timms

## 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.

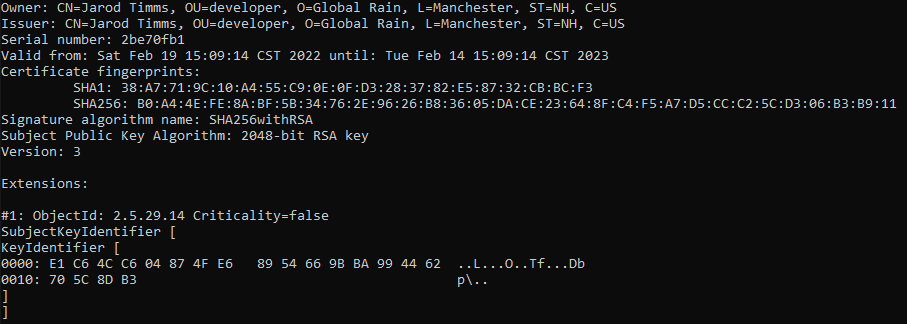
Artemis Financial has contracted us to provide them with a safe and secure communications solution for transferring files within their web application. The client requests a checksum method of file verification which is useful to identify irregularities in files in the event of a malicious attempt at authorizing or changing files in a system or to verify damaged or corrupted files. For this reason, I have selected the cryptographic linear hash algorithm SHA3-512 for its security and collision resistance. SHA3-512 produces a 512-bit output from a 576-bit block at 24 rounds and is a key-less one-way function with protection against length extension attacks. For Artemis Financials use cases, I believe this algorithm is the best for the job of checksum validation. Being one-way, there is no way to decrypt this hash, which means that there is no key that could potentially be lost or compromised. Decryption algorithms also serve little purpose for a checksum verification.

Encryption algorithms are constantly evolving to meet the needs of businesses. SHA-1 algorithms used to be the standard across many systems before SHA-2 was published and the migration began to the new and updated hash function. Shortly after most had made the transition, a collision was proven in SHA-1, which made the function completely defunct. SHA-2 has yet to have this issue proven but it is still theoretically possible. SHA3-512 is slower by about three times that of SHA-2, however I recommend this function as the speed that the computation occurs is already inconsequential to the user experience and barely noticeable. As computers increase in computational capabilities, this will be entirely unnoticeable, which makes this a great opportunity for future-proofing the system.

## 2. Certificate Generation

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

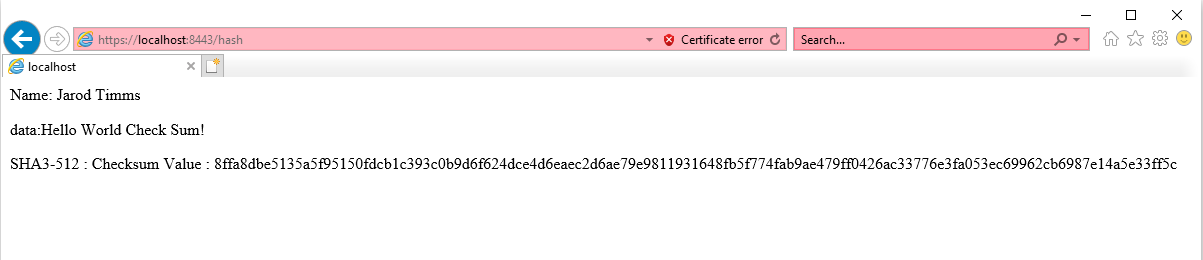
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 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, email

Description automatically generated

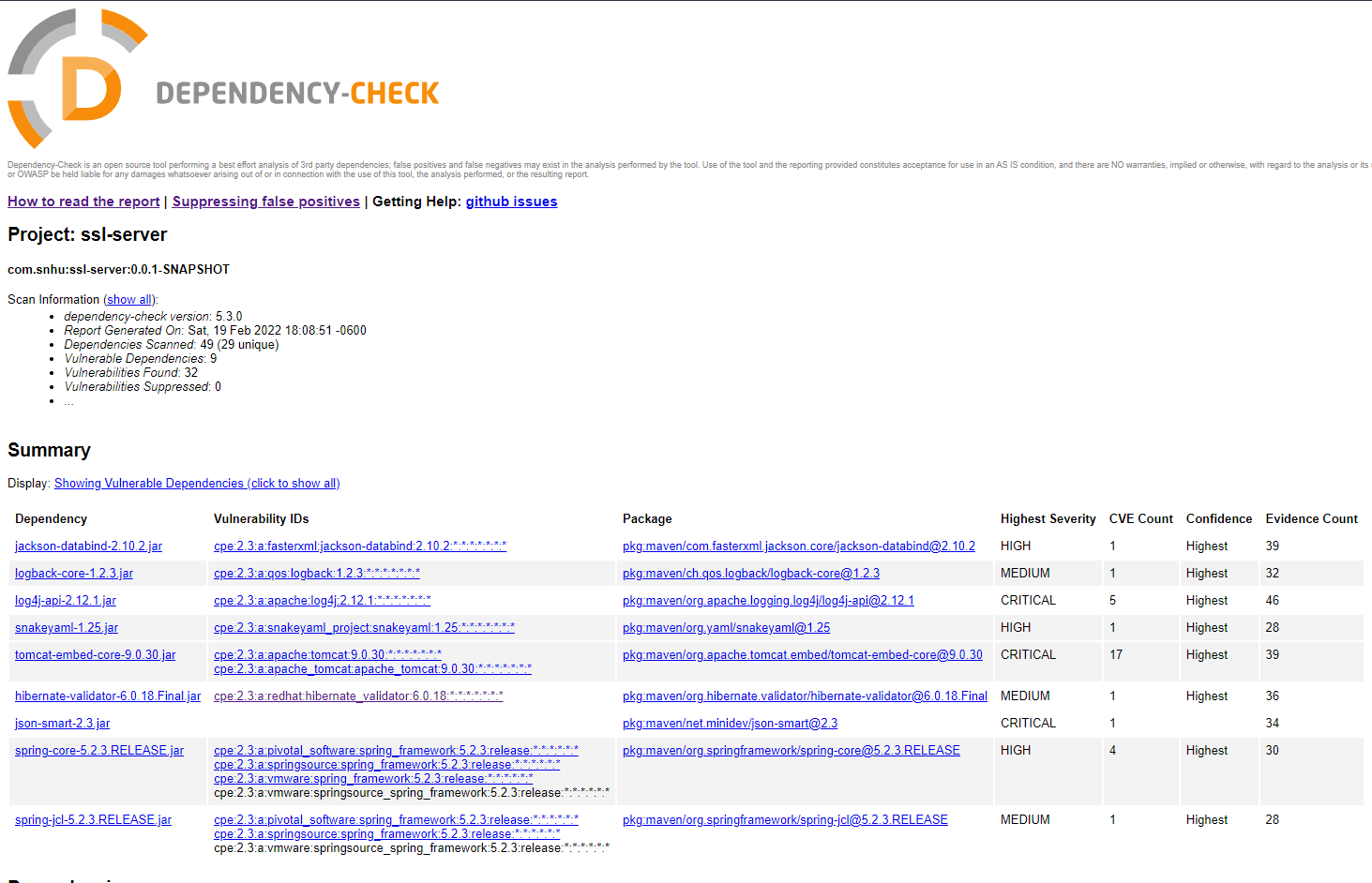
## 5. Secondary Testing

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

Text

Description automatically generated

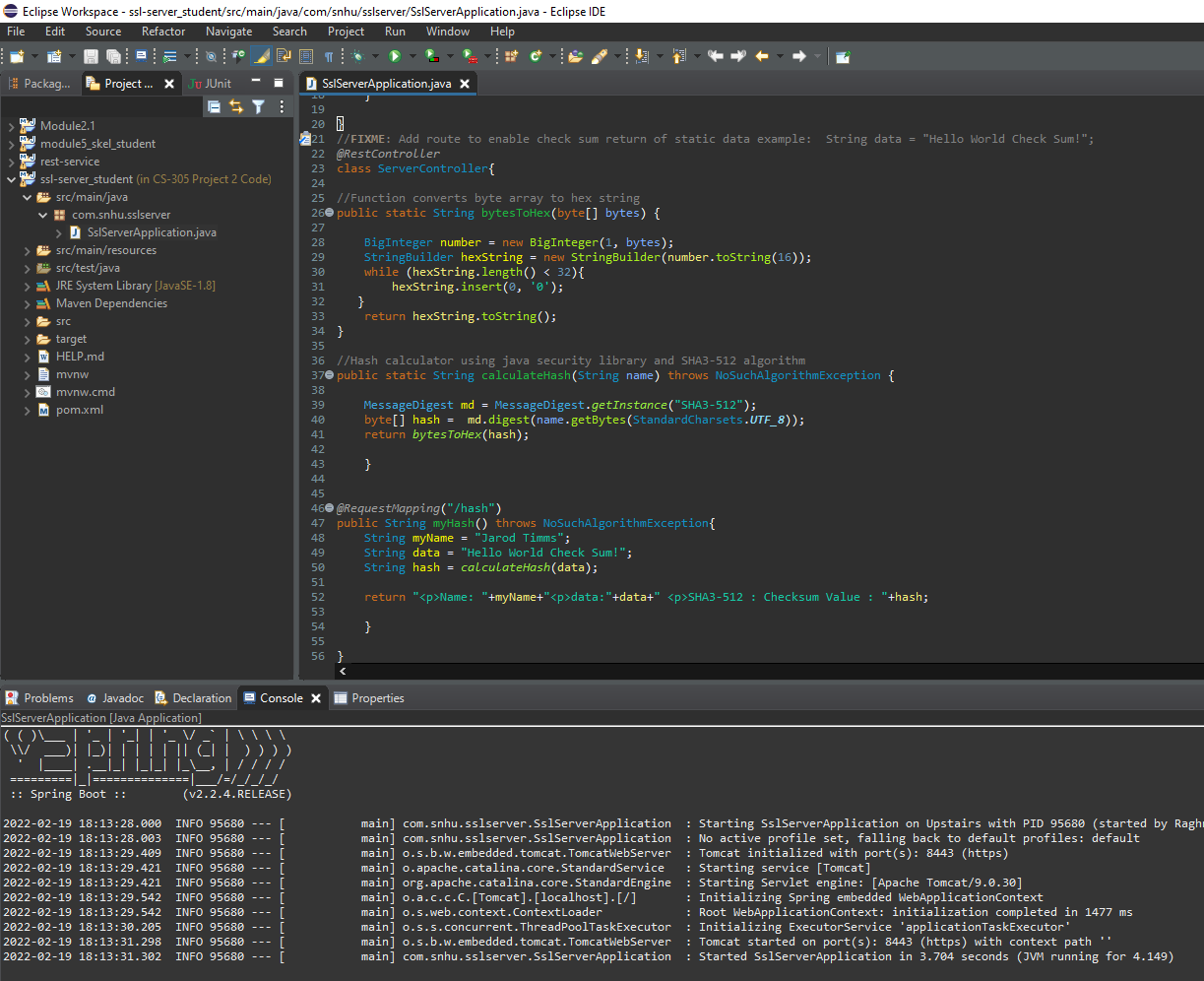


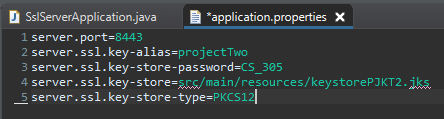
## 6. Functional Testing

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.

No new vulnerabilities were reported, and the code executes without errors.





## 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 my program I touched on multiple of the steps included within the vulnerability assessment flow diagram. While multiple areas of the diagram are addressed outside of my code, including the use of external API to manage the rest client using Spring Framework technologies, my refactored code includes examples of cryptography by utilizing native java security tools and SHA3-512 to hash and process a data string into a checksum verifiable string. I have implemented a self-issued certificate to allow for safe client to server interactions through the https protocol. I also decided to make use of encapsulation through the creation of a function that converts a byte array to a hex string that can be displayed to the user as well as a function that takes in the data string that needs to be encrypted and converts it to a byte array that’s been digested by the SHA3-512 linear function.

All business operations need to be able to function with some level of baseline confidence that the work being done is not hindered by foul actors or freak accidents. Security protocols and practices are what we use to ensure that baseline level of confidence is upheld and built on as new vulnerabilities evolve and present themselves as potential obstacles. In very general layers, I would implement security by ensuring that best practices are being applied and that futureproofing is done where possible. A good example of this is implementing SHA3-512 as opposed to SHA-2 algorithms. The layer on top of that would be to ensure regular maintenance is being done to ensure that current systems are functioning properly. The final layer on top is researching and ensuring that systems are up to date with the latest protections and documenting possible future threats and security opportunities as information becomes available.

Citation

National Security Agency. (2016, April 7). *The secure hash algorithm 3 validation system (SHA3VS)*. The Secure Hash Algorithm 3 Validation System (SHA3VS). Retrieved February 6, 2022, from https://csrc.nist.gov/CSRC/media//Projects/Cryptographic-Algorithm-Validation-Program/documents/sha3/sha3vs.pdf