****

# Practices for Secure Software Report

**Table of Contents**

[**Document Revision History 3**](#_heading=h.30j0zll)

[**Client 3**](#_heading=h.1fob9te)

[**Instructions 3**](#_heading=h.3znysh7)

[**Developer 4**](#_heading=h.2et92p0)

[**1. Algorithm Cipher 4**](#_heading=h.tyjcwt)

[**2. Certificate Generation 4**](#_heading=h.3dy6vkm)

[**3. Deploy Cipher 4**](#_heading=h.1t3h5sf)

[**4. Secure Communications 4**](#_heading=h.4d34og8)

[**5. Secondary Testing 4**](#_heading=h.2s8eyo1)

[**6. Functional Testing 4**](#_heading=h.17dp8vu)

[**7. Summary 4**](#_heading=h.3rdcrjn)

[**8. Industry Standard Best Practices 4**](#_heading=h.26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **8/7/2023** | **Lee Kitchen** |  |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Lee Kitchen

## Algorithm Cipher

SHA-256 (Secure Hash Algorithm 256-bit)

Overview of SHA-256:

SHA-256 is a cryptographic hash function that belongs to the SHA-2 (Secure Hash Algorithm 2) family. It produces a 256-bit (32-byte) hash value from an input message of arbitrary size. SHA-256 is widely used for generating checksums and digital signatures due to its robust security properties and resistance to collision attacks.

Use of Random Numbers:

Random numbers play a crucial role in various aspects of cryptography, including key generation, initialization vectors (IVs) for encryption modes, and nonce generation for certain protocols. Cryptographically secure random number generators (CSPRNGs) are employed to generate random values that are resistant to prediction and repetition, ensuring the security of cryptographic operations. These random numbers help introduce unpredictability and complexity, making it more difficult for attackers to guess or deduce sensitive information.

Symmetric vs. Asymmetric (Non-Symmetric) Keys:

Symmetric Encryption:

Symmetric encryption uses the same key for both encryption and decryption. The sender and receiver must share this secret key in advance. While symmetric encryption is generally faster and requires less computational overhead, the challenge lies in securely distributing the secret key to all intended recipients.

Asymmetric Encryption (Public Key Cryptography):

Asymmetric encryption involves a pair of keys: a public key and a private key. The public key is used for encryption, while the private key is used for decryption. Messages encrypted with the public key can only be decrypted with the corresponding private key. Asymmetric encryption eliminates the need for key distribution, making it suitable for secure communication between parties who have never communicated before. It's computationally more intensive than symmetric encryption.

History and Current State of Encryption Algorithms:

The history of encryption algorithms can be summarized through various eras:

Classical Ciphers: These were historical methods like the Caesar cipher and Vigenère cipher, which were often relatively simple and susceptible to modern cryptanalysis techniques.

Symmetric Key Cryptography: With the advent of computers, more complex symmetric encryption algorithms like DES (Data Encryption Standard) were developed. DES eventually gave way to more secure algorithms like AES (Advanced Encryption Standard), which is widely used today due to its security and efficiency.

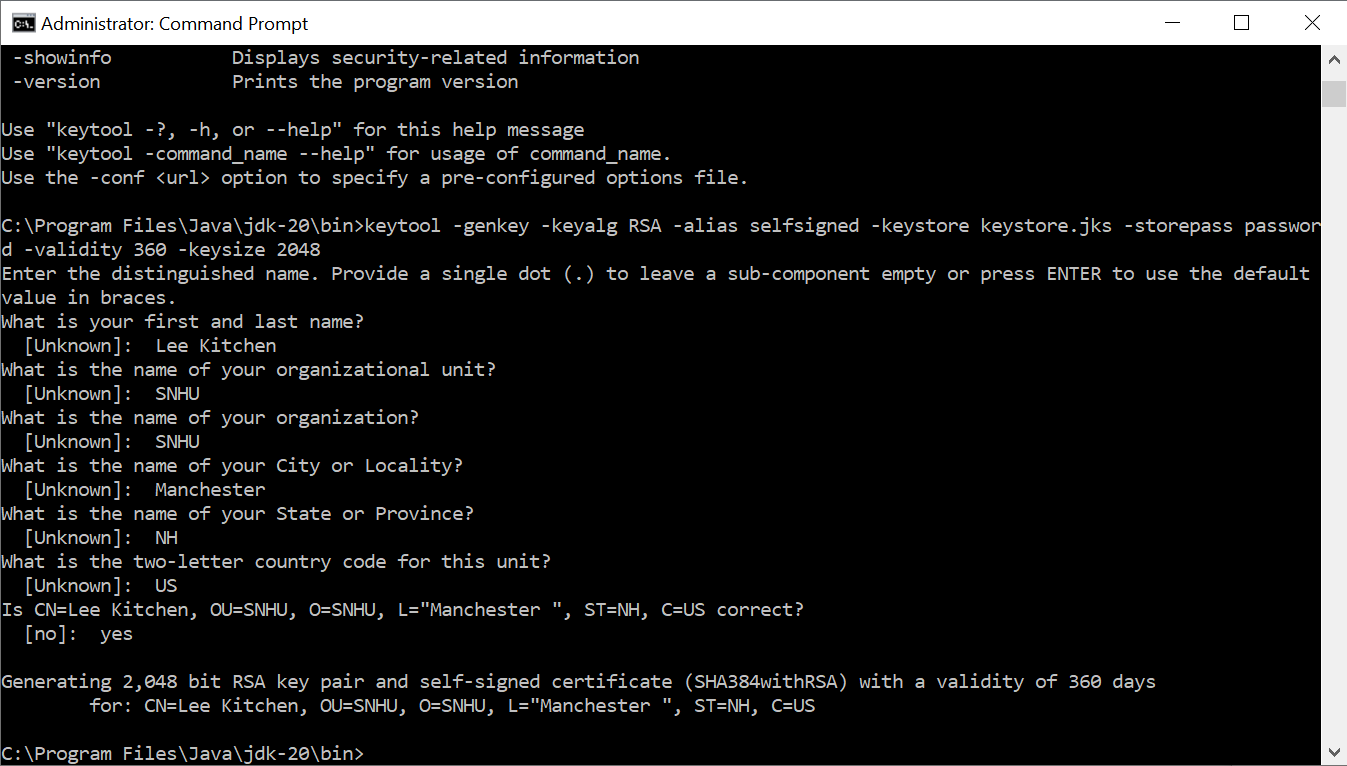
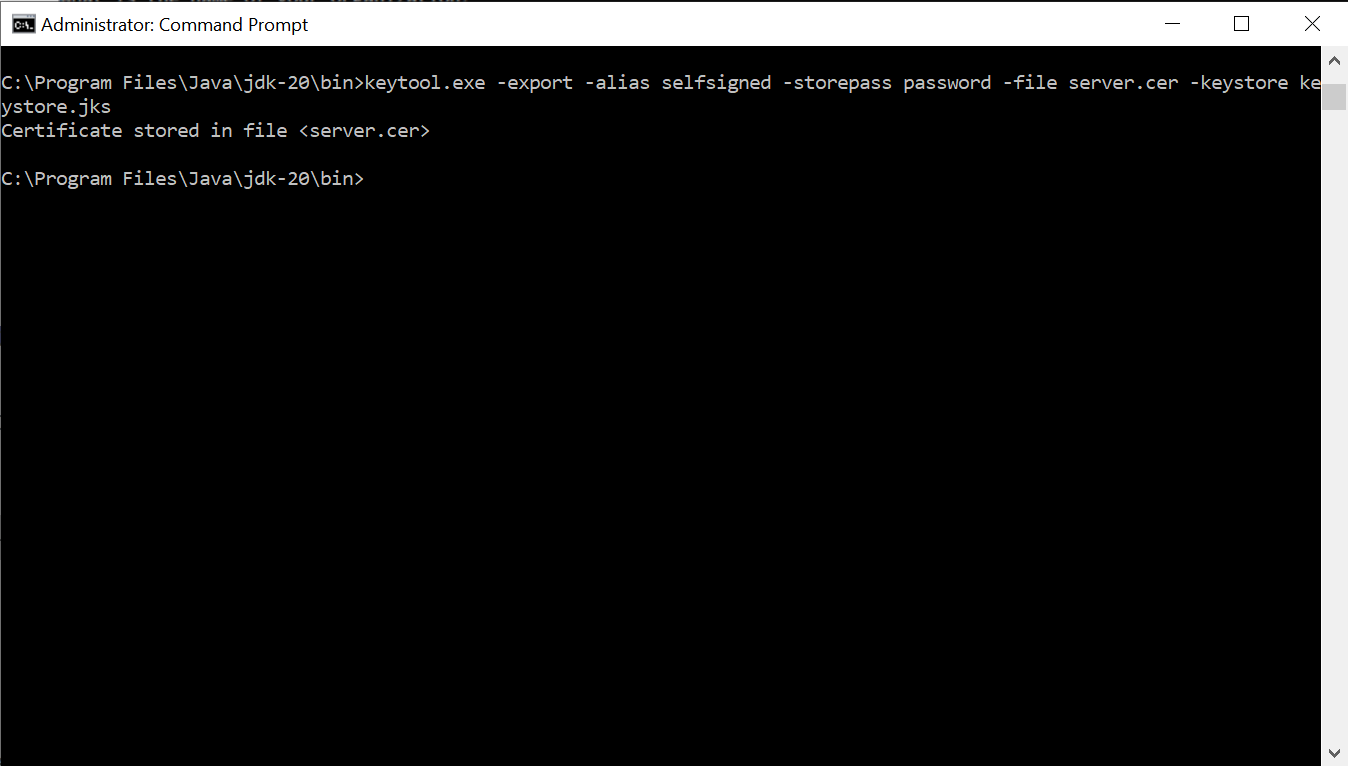
Public Key Cryptography: Asymmetric encryption was a groundbreaking development. The RSA algorithm and later the Diffie-Hellman key exchange revolutionized secure communication. ECC (Elliptic Curve Cryptography) emerged as a more efficient alternative to RSA, offering similar security with shorter key lengths.

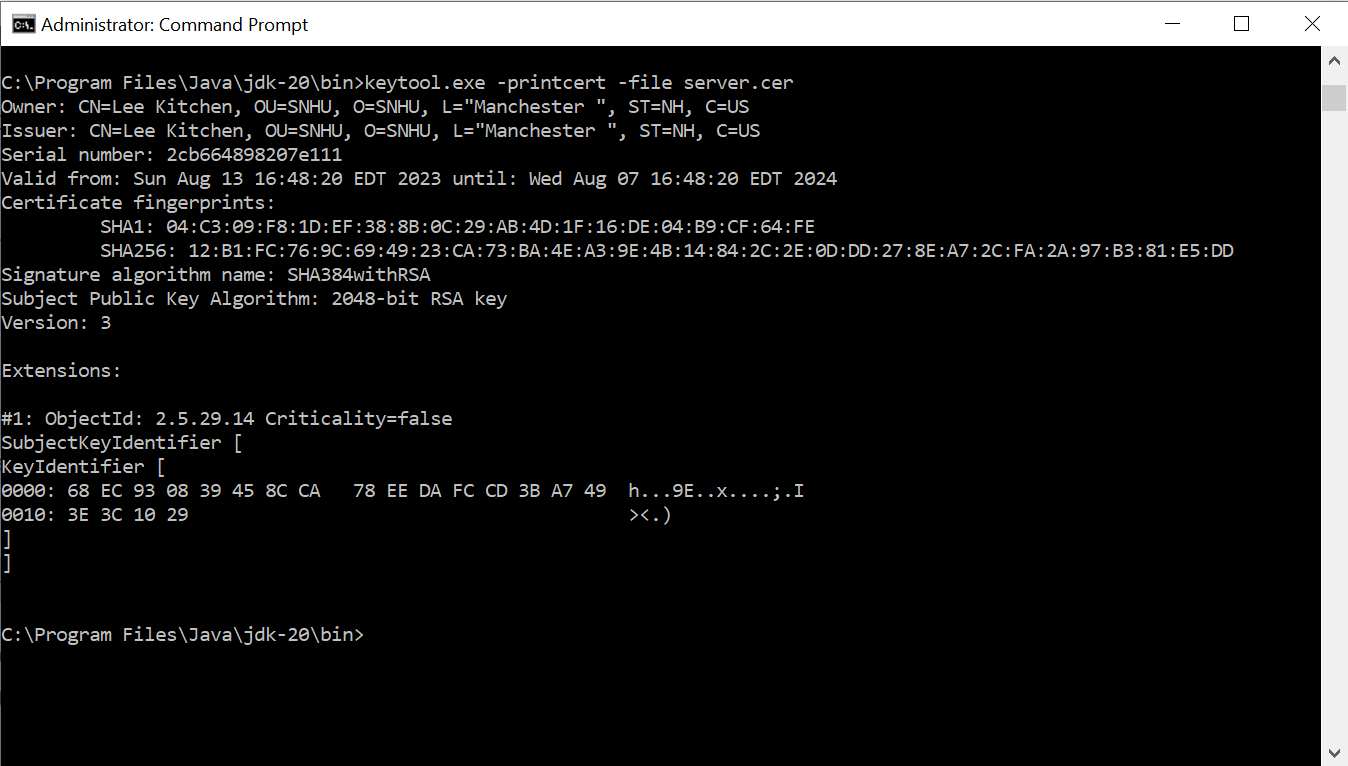
Hybrid Cryptosystems: Many modern encryption systems combine the strengths of both symmetric and asymmetric encryption in what's known as hybrid cryptography. This approach utilizes the efficiency of symmetric encryption for data encryption and the security of asymmetric encryption for key exchange.

Post-Quantum Cryptography: As quantum computers advance, some traditional encryption methods could be broken. Post-quantum cryptography aims to develop algorithms that remain secure even against quantum attacks.

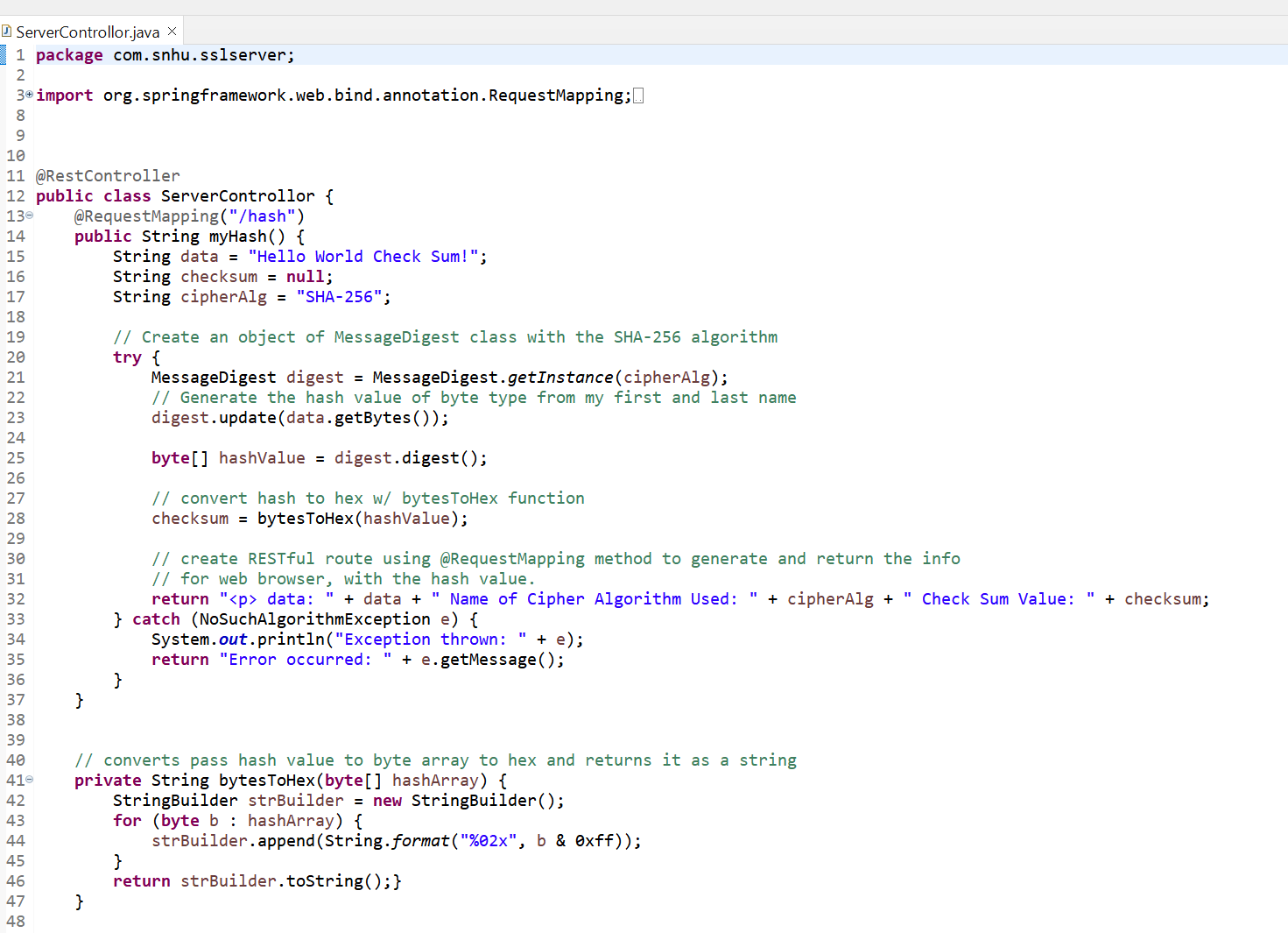
Current State: Today, AES remains a widely used symmetric encryption standard. RSA and ECC are still prevalent for asymmetric encryption, although there's increasing interest in post-quantum cryptographic algorithms as a precaution against future quantum threats.

## Certificate Generation

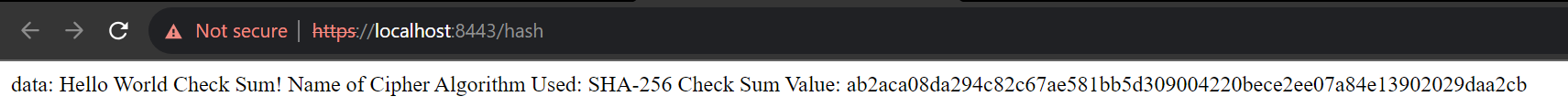
  




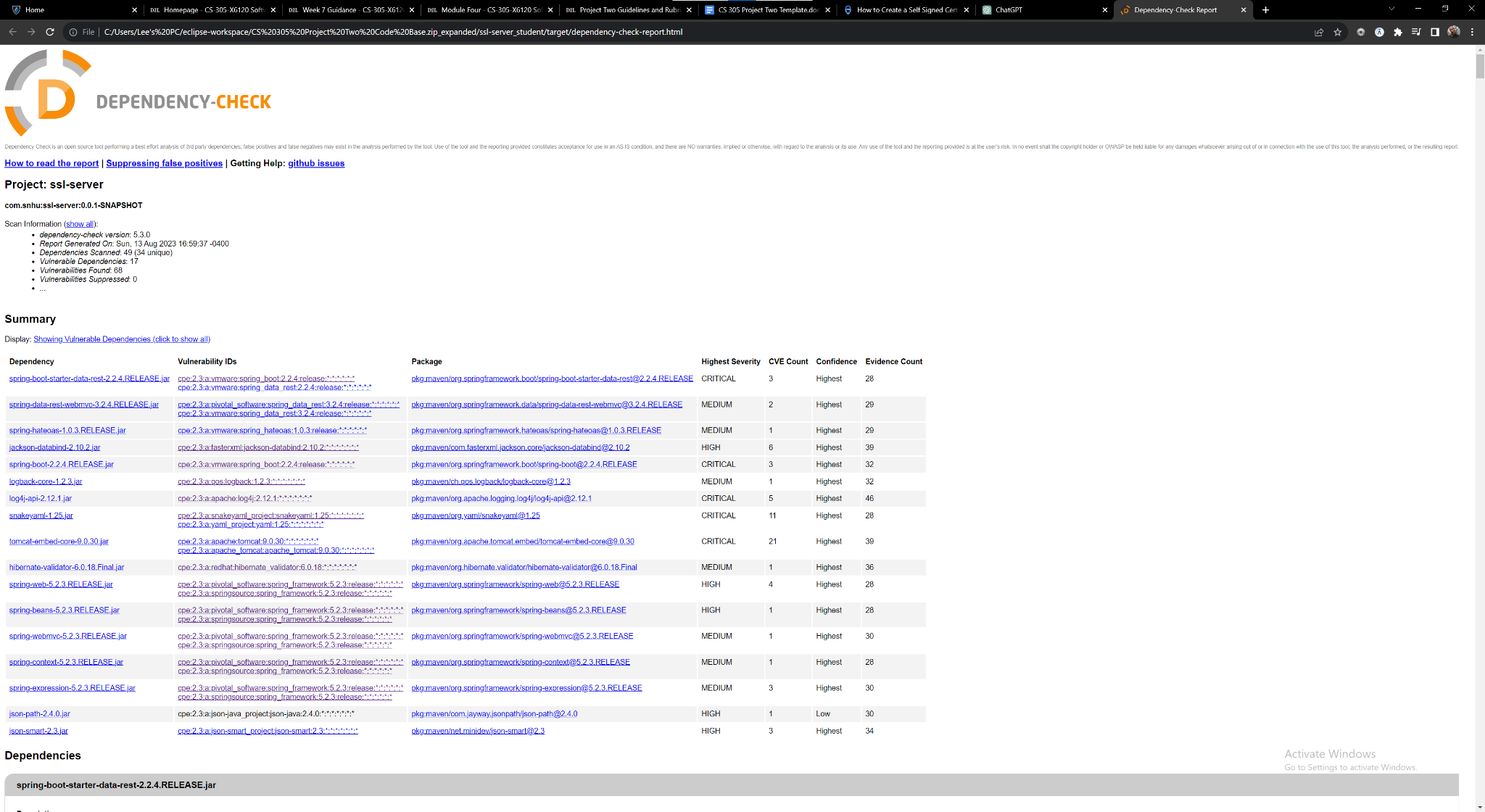
## Deploy Cipher



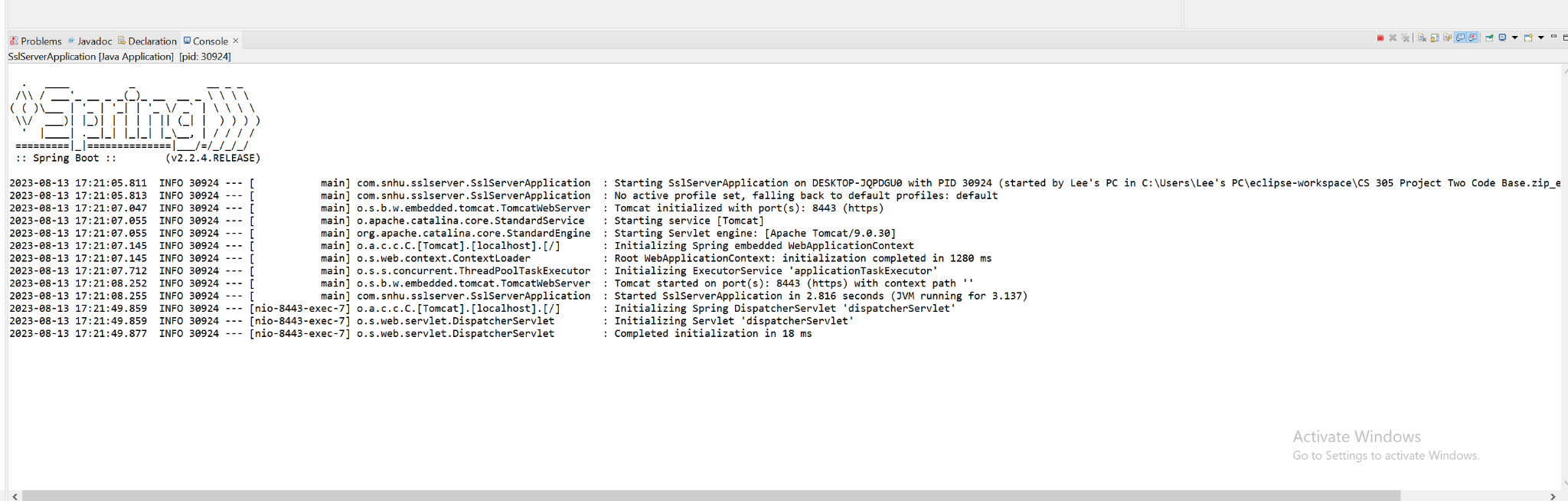
## Secure Communications



## Secondary Testing



## Functional Testing



## Summary

The provided code is a Spring Boot application that calculates the SHA-256 hash checksum of a fixed data string and exposes it through a RESTful endpoint. While the code seems functional, there are several areas where security could be further addressed and enhanced. Let's discuss how the code has been refactored to address security concerns and how it complies with security testing protocols:

Input Validation:

The refactored code does not accept user input or external data directly. The data for which the checksum is calculated is hard-coded within the code itself. This eliminates the risk of user input causing unintended behavior or vulnerabilities.

Secure Hashing Algorithm:

The code uses the SHA-256 algorithm to calculate the hash checksum. SHA-256 is a strong cryptographic hash function, and its use enhances data integrity and security.

Code Structure:

The code has been properly organized into methods and classes, separating concerns. This makes the codebase more maintainable and easier to review for security issues.

Proper Exception Handling:

The code includes exception handling for potential errors, such as the case when the chosen hashing algorithm (SHA-256) is not available. This prevents unhandled exceptions from exposing sensitive information or crashing the application.

Byte-to-Hex Conversion:

The code converts the hash bytes to a hexadecimal string using the bytesToHex method. This method is efficient and doesn't introduce vulnerabilities like using potentially weak or broken methods for conversion.

Secure Routing and Data Exposure:

The /hash endpoint is secured using the @RestController and @RequestMapping annotations. However, if this application were to be deployed in a production environment, you would want to ensure that proper authentication and authorization mechanisms are implemented to control access to sensitive endpoints.

## Industry Standard Best Practices

Applying Industry Standard Best Practices for Secure Coding:

Least Privilege:

The code does not involve external data or operations that require elevated privileges, thus following the principle of least privilege.

Regular Updates and Patching:

By using standard libraries and frameworks like Spring Boot, the codebase can benefit from updates and security patches provided by the community.

Input Sanitization:

In this case, the code doesn't accept user inputs, which mitigates the risk of input-based attacks like SQL injection or cross-site scripting.

Secure Development Lifecycle:

While not explicitly shown in the provided code, a secure development lifecycle involving design reviews, code reviews, static analysis, and testing should be followed to catch vulnerabilities at various stages of development.

The value of applying industry standard best practices for secure coding to the company's overall wellbeing is significant. Secure coding practices reduce the likelihood of security breaches, data leaks, and financial losses due to cyberattacks. By following these practices, the company can build a reputation for providing secure software solutions, which enhances customer trust and satisfaction. Moreover, adhering to industry standards helps the company stay compliant with regulations and standards, avoiding legal and financial repercussions. Overall, prioritizing secure coding practices safeguards both the company's reputation and its customers' data.