

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **08/25/2024** | **Felicia Mirabel** |  |

## 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

Felicia Mirabel

## Algorithm Cipher

For securing communications over TLS/SSL I recommend using AES encryption as it provides the best combination of compatability with security.

Advanced Encryption Standard (AES) is a symmetric key encryption algorithm popular for its speed and robust security, which is commonly used in TLS/SSL encryption schemes.

The hash function used will be the Secure Hash Algorithm 256-bit (SHA-256) which provides a fixed size 256-bit hash which provides a secure way to perform data integrity checks while minimizing the risk of collisions.

Symmetric keys (like AES) are used for both encryption and decryption and is generally faster, suitable for bulk data encryption as well as easier for the end user/.

Asymmetric keys (Such as those used in Rivest-Shamir-Adleman RSA) use different keys for encryption and decryption. This is typically used for key exchange rather than bulk data encryption due to its computational complexity.

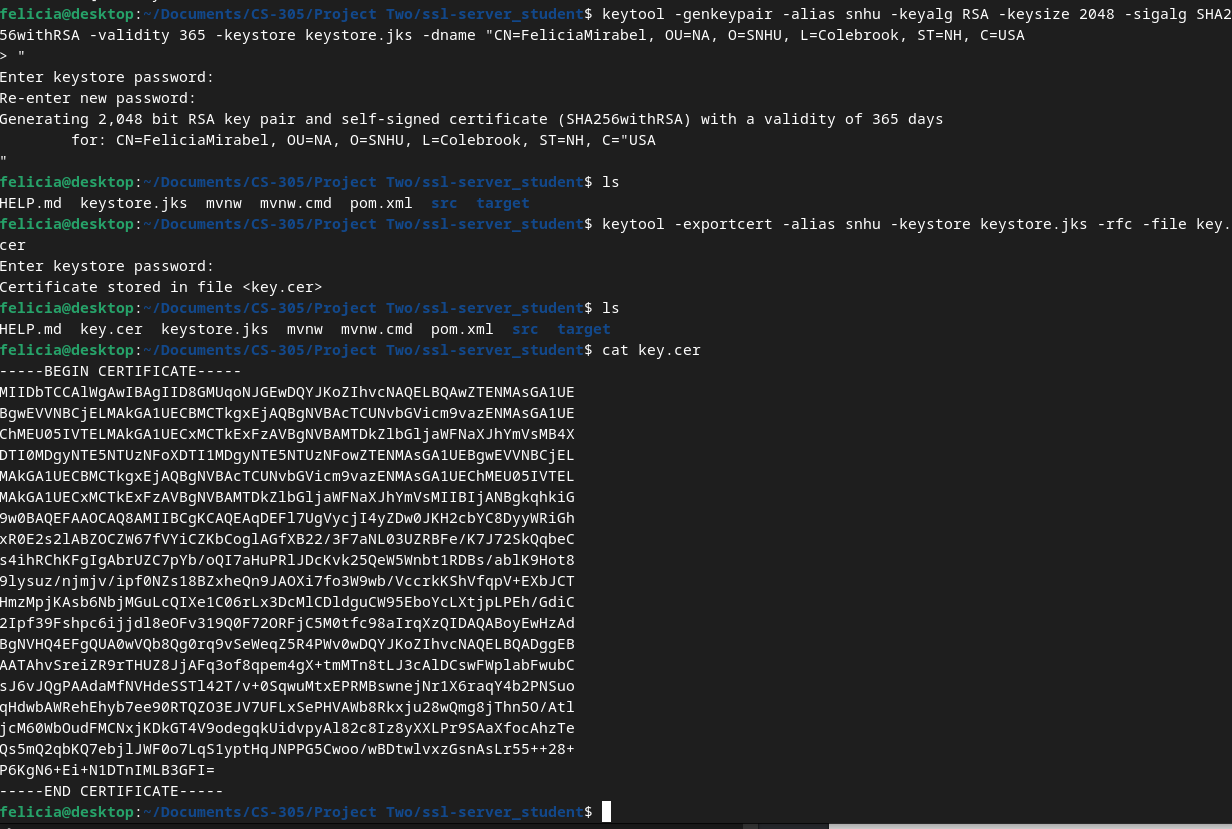
Random numbers are crucial for generating keys, initializing vectors, and ensuring non-repetitive encryption patterns. Secure Random Number Generators are essential to ensure the strength of the encryption used as Pseudo-Random Number Generators generate numbers in known patterns which can result in encryption being more easily reverse engineered.

Older algorithms generally tend to be depreciated due to several reasons, including vulnerabilities being discovered as well as advancement in cryptanalysis and technological advancement; for example, DES and 3DES have both been retired from use because of security vulnerabilities as well as technology surpassing technical hurdles needed to bypass their algorithms.

Newer algorithms are developed to overcome these shortcoming. AES and ECDHE, for example, are the current standards for securing communications as they are both mathematically and theoretically more robust, as well as extensible with larger key sizes able to be used as technology advances and new key discovery methodologies are implemented to take advantage of known mathematical weaknesses in the algorithms.

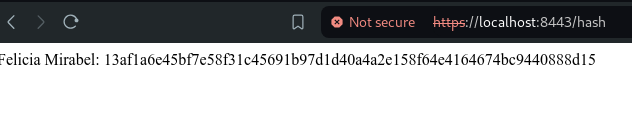
## Certificate Generation

Insert a screenshot below of the CER file.



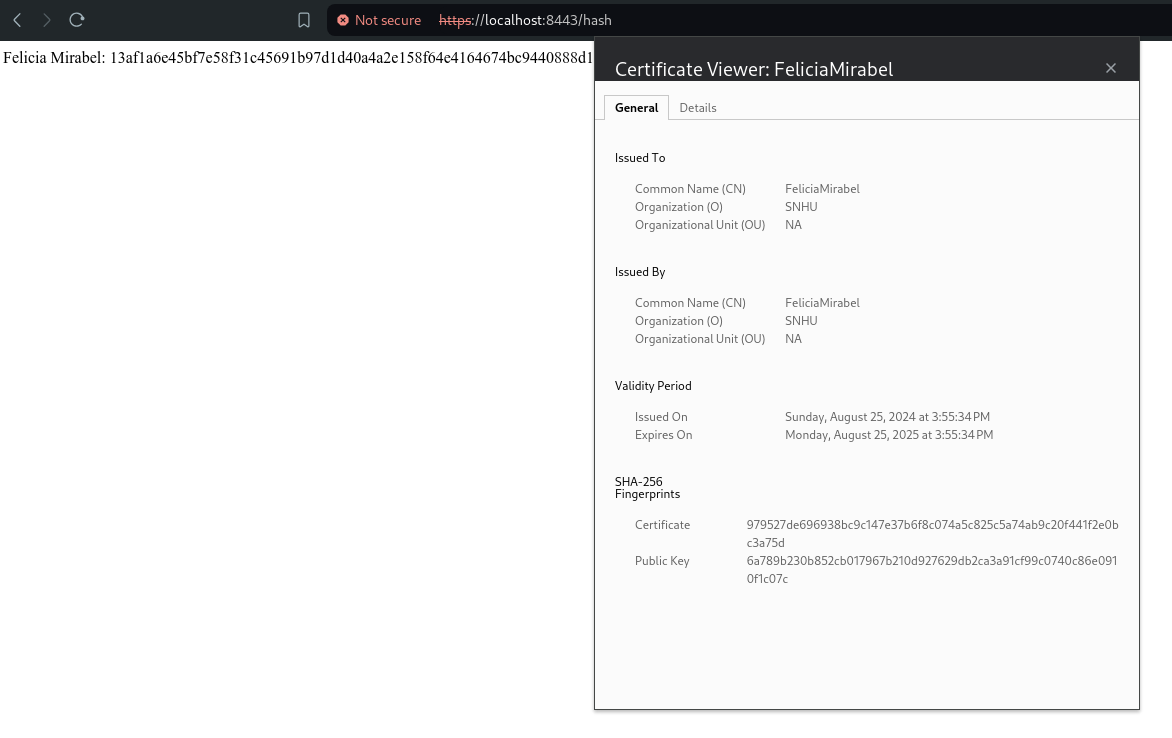
## Deploy Cipher

Insert a screenshot below of the checksum verification.



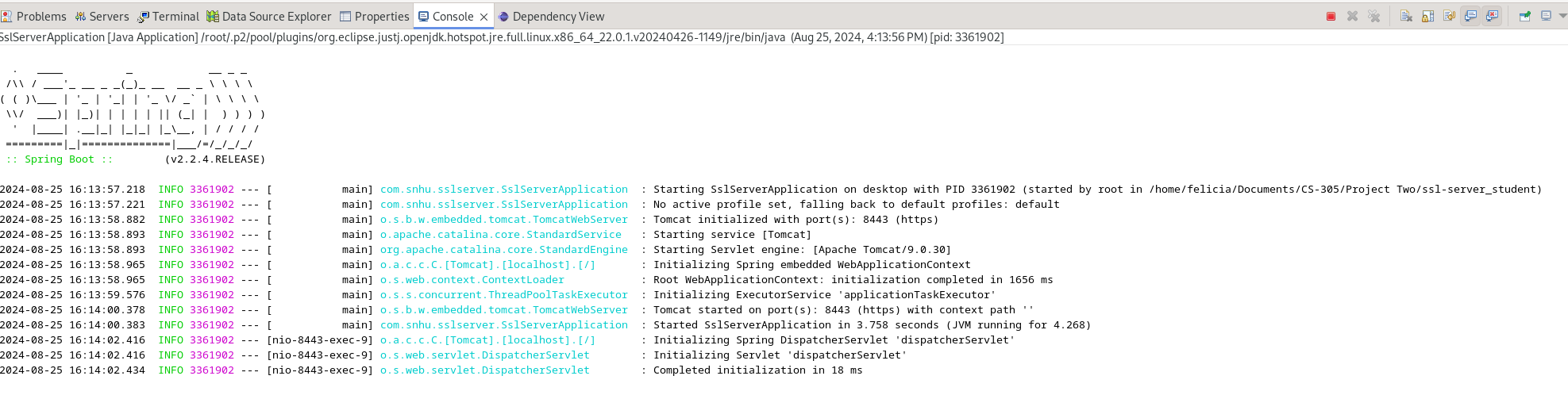
## Secure Communications

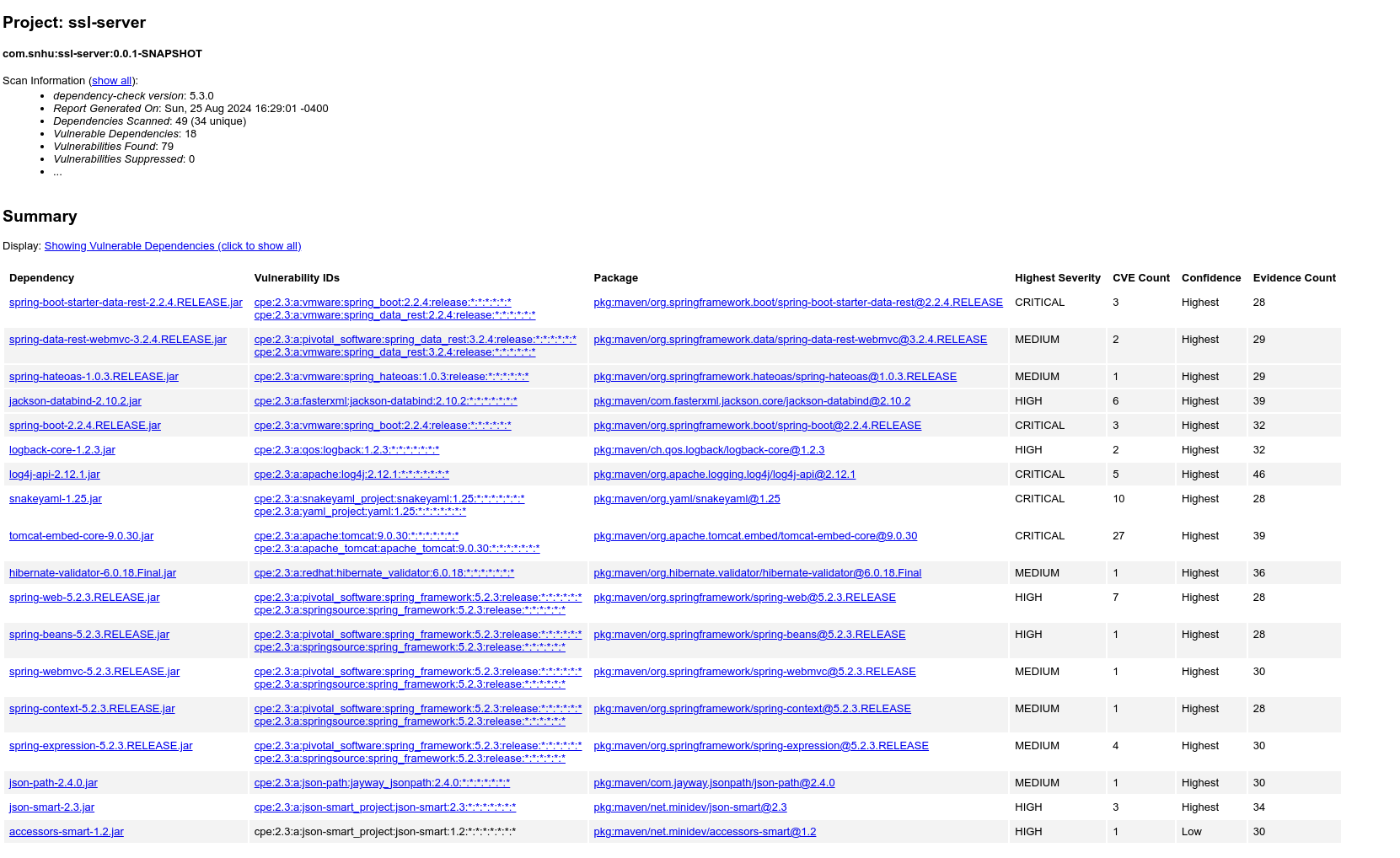
Insert a screenshot below of the web browser that shows a secure webpage.



## Secondary Testing

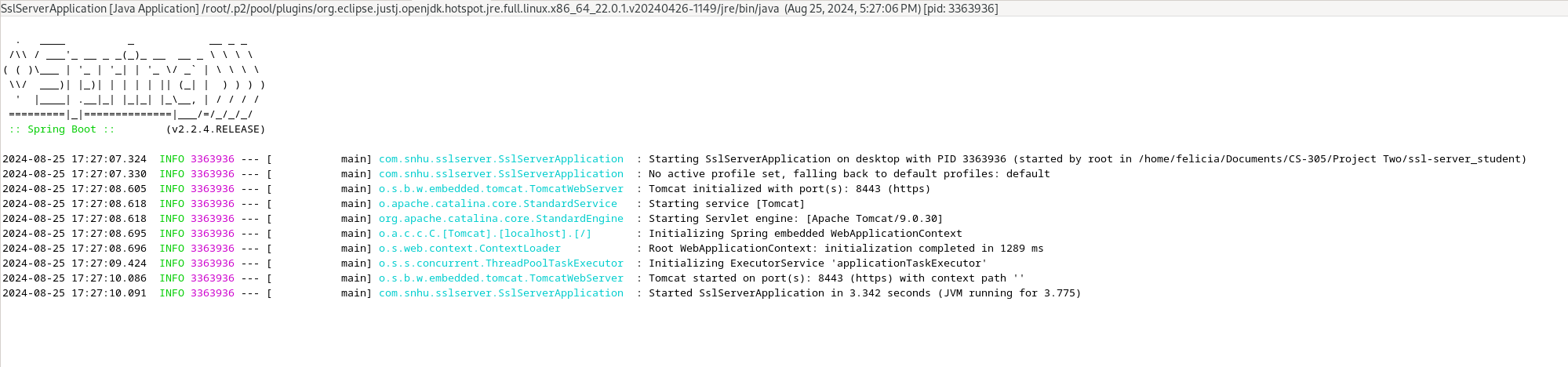
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

Insert a screenshot below of the refactored code executed without errors.



## Summary

The refactoring followed a systematic approach that followed the vulnerability assessment process flow diagram including identification of vulnerabilities, implementation of security measures, and a verification process. A syntactical, logical, and security review was conducted on the code followed by addressing a dependency injection vulnerability, input validation vulnerability, poor exception handling, and secure hashing.

The areas of security addressed by refactoring include input handling and validation. Error handling and logging, and implementation of better, more secure coding practices.

The process of adding layers of security involved adding a layered approach to input validation (sanitization and validation), encapsulation and dependency management (using dependency injection to improve design and limit application logic exposure), exception management (error logging), and utilizing a security-first mindset.

## Industry Standard Best Practices

To maintain the application’s existing security, several key practices were implemented. The use of the SHA-256 hashing algorithm ensured that data integrity was maintained without introducing vulnerabilities associated with weaker algorithms like MD5 or SHA-1. Input validation and sanitization were also implemented to prevent injection attacks, such as SQL injection and cross-site scripting (XSS). By ensuring that only alphanumeric characters and spaces are accepted, the risk of malicious inputs that could exploit vulnerabilities in the application was significantly reduced. Proper exception handling was another critical aspect of the refactoring process. Instead of exposing runtime exceptions, the code was modified to log errors internally while providing user-friendly messages. This approach minimizes the risk of information disclosure while ensuring that errors are tracked for later analysis. Additionally, dependency injection was used in the ChecksumController to access the SslServerApplicatio instance, reducing the potential for security flaws by minimizing the use of global states.

The application of industry standard best practices for secure coding offers substantial benefits to the company’s overall well-being. Protecting customer data is important and ensuring that sensitive information is securely handled helps maintain trust and loyalty. By implementing secure hashing, input validation and proper exception handling, the company reduces the risk of data breaches and potential legal liabilities associated with the exposure of sensitive data. A strong reputation for security can lead to increased customer retention and business growth. Additionally, secure coding practices help identify and resolve vulnerabilities early in the development cycle, reducing the costs associated with post-release patches and security fixes.

Applying industry standard best practices for secure coding is essential for maintaining the security of software applications and protecting the company’s assets, reputation, and legal standing. By integrating these practices into the development process, the company ensures that its applications are secure, reliable, and compliant with industry regulations, thereby safeguarding its long-term success and well-being.