****

# 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** | **10/17/2024** | **Zane Deso** |  |

## Client



## Developer

Zane Deso

## Algorithm Cipher

The encryption algorithm cipher that I recommend is AES-256, short for Advanced Encryption Standard with a 256-bit key. It is a highly secure encryption algorithm used to protect electronic data.

AES-256 is a symmetric encryption algorithm used to protect sensitive data by encrypting and decrypting information in fixed blocks of 128 bits. It employs a 256-bit key length, making it highly resistant to brute-force attacks and ensuring a strong level of security. Widely adopted across various industries, AES-256 is approved by the U.S. government for securing classified information up to the top-secret level.

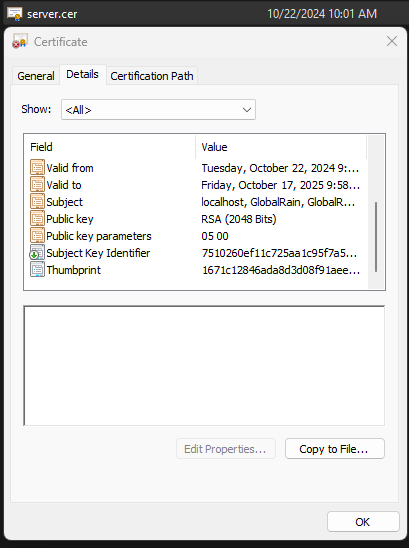
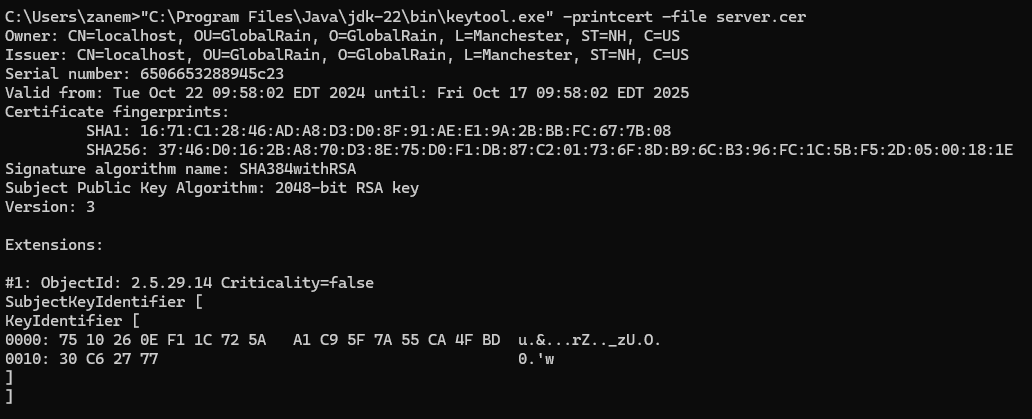
AES-256 operates by processing 128-bit data blocks through 14 rounds of encryption, utilizing functions such as SubBytes (byte-level substitution), ShiftRows (row-wise permutation), MixColumns (column mixing), and AddRoundKey (key addition). Each round uses a 256-bit key that's expanded into round keys via a key schedule algorithm, making it so that different keys are applied at each stage of encryption. The combination of these functions at various bit levels provides strong security by thoroughly diffusing and confusing the input data, making it resistant to cryptographic attacks

Random numbers play a vital role in AES-256 encryption by generating strong and unpredictable keys, ensuring that each encryption is unique and secure against attacks. AES-256 uses symmetric keys, meaning the same 256-bit key is used for both encryption and decryption, which requires secure key distribution between the communicating parties. In contrast, asymmetric (non-symmetric) encryption algorithms use a pair of keys—a public key for encryption and a private key for decryption—eliminating the need to share a secret key but often at the expense of speed and computational resources on larger scale use cases. Random numbers are also used in generating initialization vectors (IVs) when AES is used in certain modes, adding additional randomness to prevent patterns in encrypted data. The use of randomness and symmetric keys in AES-256 balances strong security with efficient performance for data protection.

Encryption algorithms have evolved over time to address emerging security threats and technological advancements. AES was established in 2001 to replace the outdated DES (Data Encryption Standard), which had become vulnerable due to its short key length and susceptibility to brute-force attacks. Today, AES is the industry standard for symmetric encryption, but competitors like ChaCha20 and Twofish also offer strong security and are preferred in certain contexts for their performance benefits over AES. The current state of encryption emphasizes the use of robust, peer-reviewed algorithms, with ongoing developments in elliptic curve cryptography and post-quantum encryption to meet future security needs.

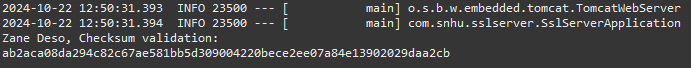
## Certificate Generation

Insert a screenshot below of the CER file.



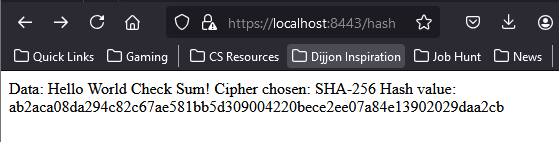
## Deploy Cipher

Insert a screenshot below of the checksum verification.



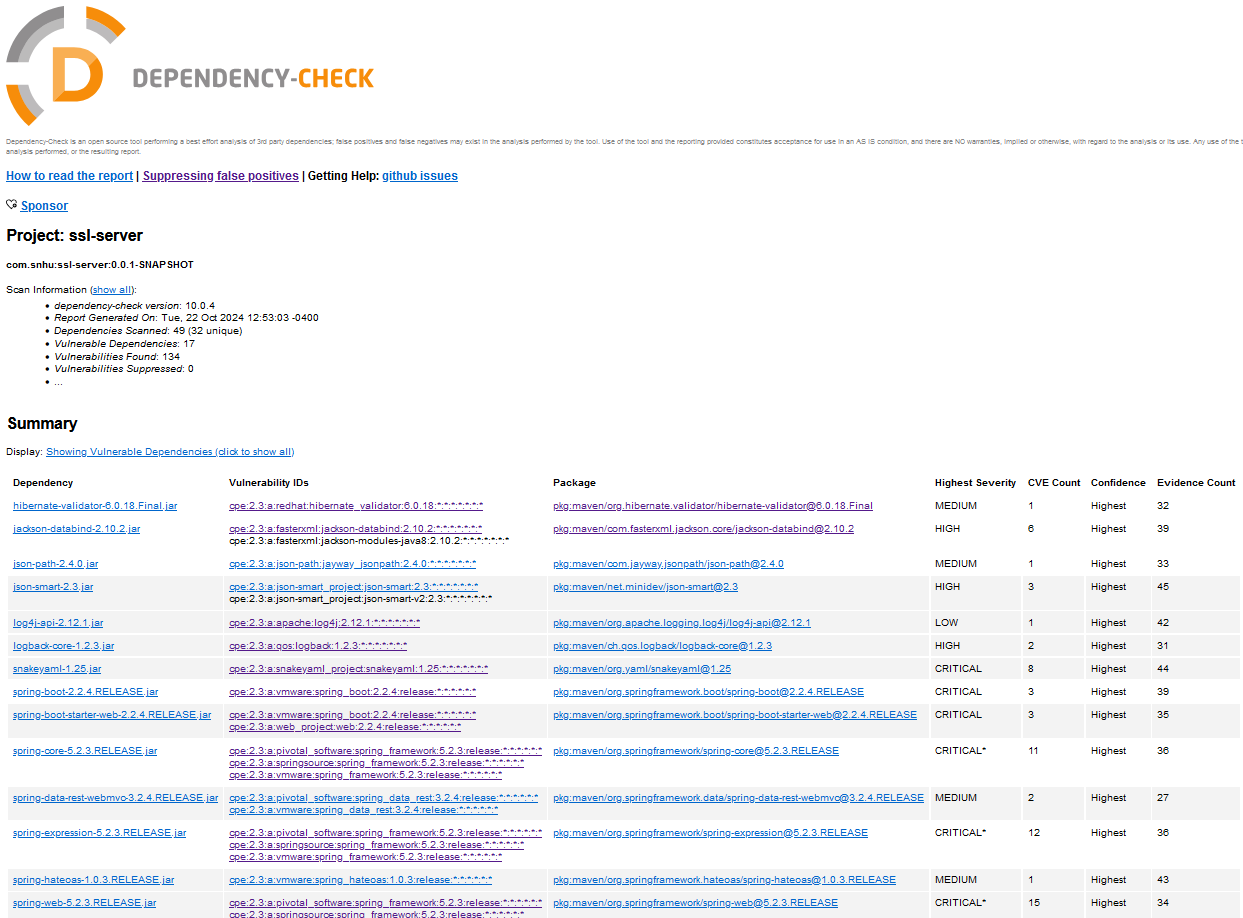
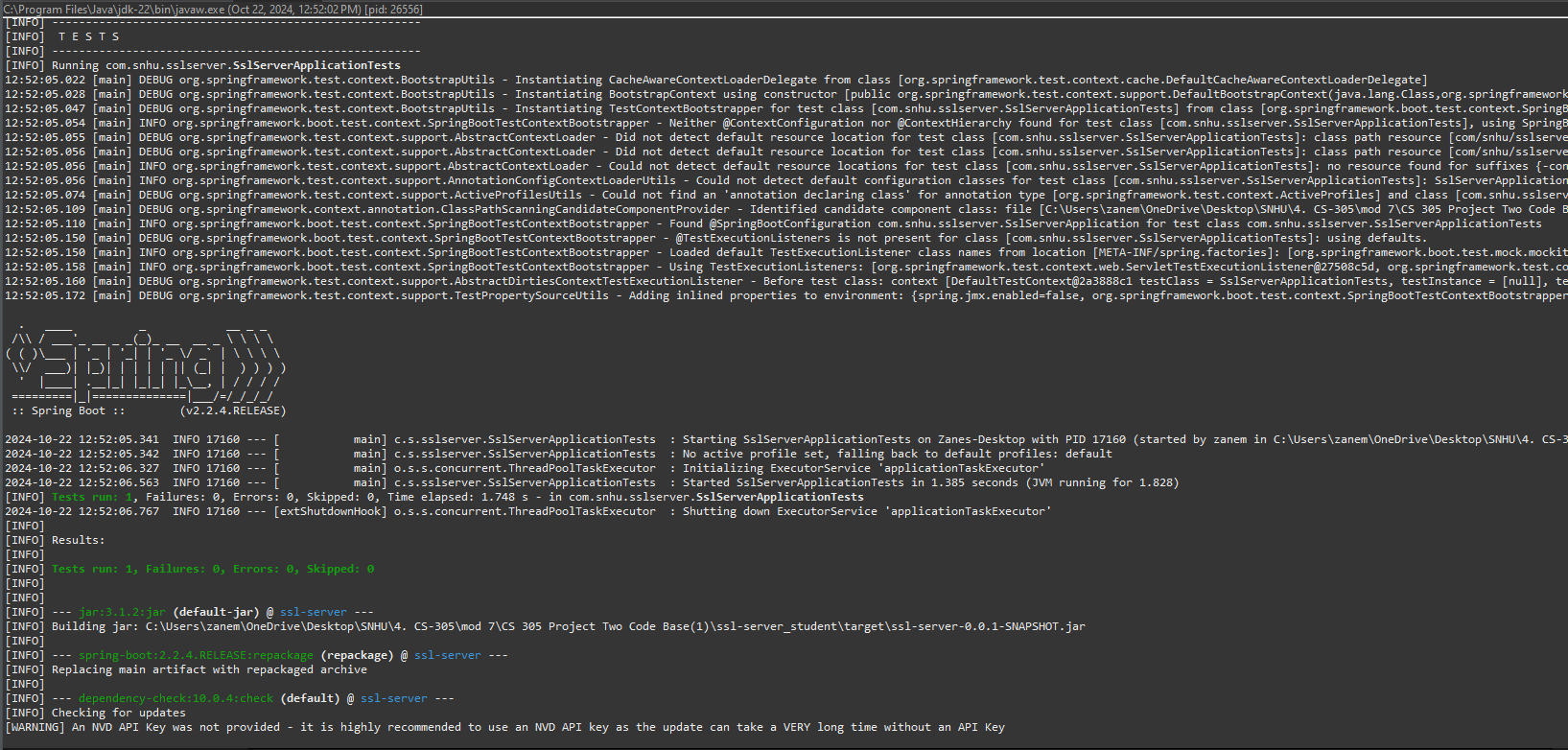
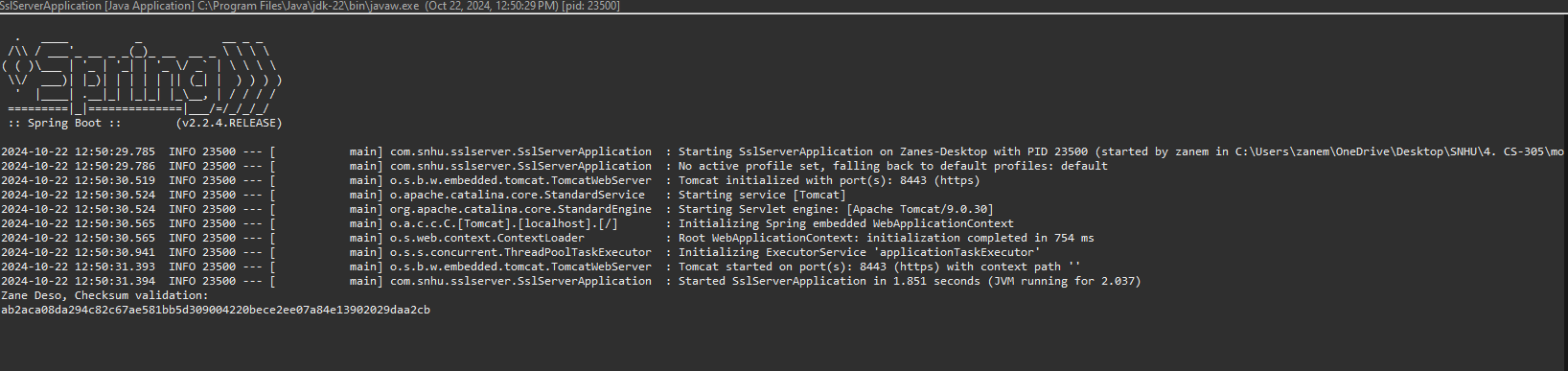
## Secure Communications

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



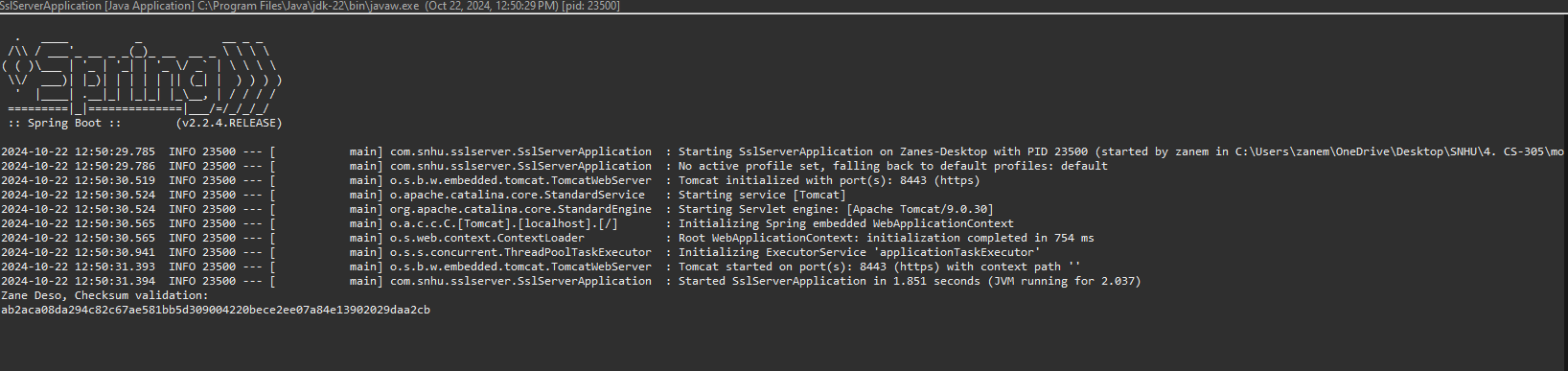
## Secondary Testing

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

In addressing the areas of security in the software application's codebase, I began by focusing on cryptography. Specifically, I implemented encryption and conducted vulnerability assessments to ensure the integrity of the system. Securing client-server communications was a priority, and I achieved this by encrypting data in transit and employing hashing techniques to verify the integrity of the data. By implementing checksums, I validated that transmitted data remained intact, while the use of a self-signed certificate allowed for the secure setup of HTTPS endpoints. This approach effectively encrypted the communication channels and ensured data confidentiality. Additionally, I improved the code by resolving errors and maintaining overall code quality, following secure coding practices to enhance system resilience. Encapsulation was also a key focus, with methods and classes being clearly defined to prevent external interference, ensuring that each function operates within its appropriate scope.

To further enhance the application's security, I followed a multi-layered approach, starting with the selection of the AES-256 encryption algorithm, which best suited the client's requirements. This decision came after extensive research and analysis, forming a foundational layer in the software security development lifecycle. I then applied a self-signed certificate to verify the application's behavior under HTTPS communication protocols, although for production environments, I would recommend using a certificate from a trusted authority. Encapsulation was applied not only in code structure but also between client and server interactions, ensuring that server-side logic remains protected from unauthorized access. Finally, I conducted a comprehensive round of testing, compiling the code and running a Maven dependency check report to identify security vulnerabilities. Mitigating these vulnerabilities can be achieved by upgrading dependencies to their latest versions, providing a straightforward but effective enhancement to the overall security posture of the application.

## Industry Standard Best Practices

To ensure the security of the software application, I adhered to industry standard best practices, particularly by utilizing secure coding patterns and avoiding those that are easily exploitable. By following the latest documentation for the Spring framework and associated dependencies like MessageDigest, I ensured that all code implemented was both secure and up-to-date. This proactive approach ensures that the application remains resilient to potential vulnerabilities introduced by outdated libraries or weak patterns. Encryption played a key role in protecting sensitive data, particularly during data transit, which aligns with industry guidelines for secure communications. I also incorporated code reviews as part of the process, conducting both manual reviews and utilizing tools like Maven dependency checks to identify vulnerabilities within the frameworks and libraries in use. Additionally, I ensured that the API endpoints were secured, preventing unauthorized access and strengthening the application's overall defense. The security risk assessment carried out during the project further reinforced these best practices, ensuring that all stakeholders—developers, clients, and security teams—were aligned on maintaining the highest security standards.

Applying industry standard best practices to this project helps to maintain the company’s reputation and operational security. By continuously identifying vulnerabilities and recommending upgrades to the latest library versions, I contributed to the maintenance of the software's current security. This proactive approach minimizes risks and ensures that the application is equipped to handle potential threats. For a company like Artemis Financial, a global financial consulting firm, maintaining strict adherence to security standards is critical. Failing to do so could tarnish the company's credibility, result in data breaches, and cause significant financial losses. The value of applying secure coding practices lies in the protection of both the company’s reputation and its bottom line. I would even suggest involving data experts to model the potential damage to revenue and reputation if industry best practices were neglected. The cost of not securing the company’s software infrastructure could be measured in millions of dollars, demonstrating just how crucial these standards are to the long-term well-being of the company.