****

# 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** | **Feb-19-23** | **Debynyhan Banks** |  |

## Client



## Developer

Debynhan Banks

## Algorithm Cipher

The Advanced Encryption Standard (AES) is a widely used symmetric key encryption algorithm (*What Is the Advanced Encryption Standard? (AES)*, n.d.). It was developed to replace the aging Data Encryption Standard (DES) and is currently one of the most popular encryption algorithms. The AES algorithm is a block cipher that operates on data in blocks of 128 bits. (“Security algorithms”) It supports key sizes of 128, 192, or 256 bits. (“Symmetric Cryptography - an overview | ScienceDirect Topics”) The key size used in the encryption process is called the bit level. A higher bit level indicates a more secure key and greater encryption. The algorithm uses a substitution-permutation network (SPN) to encrypt the data. In this process, the data is first transformed through a series of substitution and permutation operations, and then XORed with the key to produce the encrypted output. The number of rounds of encryption that AES performs on a 256-bit key is 14 rounds of encryption (Youssef et al., 1993).

One of the strengths of AES is its high level of security. The number of keys that can be used in AES is so large that it is currently impossible to break AES by brute force methods. It is also highly resistant to other types of attacks. AES is widely used in various applications, including secure communications over the internet, file encryption, and database encryption. AES is a highly secure and versatile encryption algorithm that provides a high level of protection for sensitive data.

SHA-256 is a cryptographic hash function that operates on individual bits of data. It takes an input message of any length and produces a fixed-length output of 256 bits, known as the hash or message digest. SHA-256 works by performing a series of logical operations, including bitwise operations and modular arithmetic, on the input data. The algorithm processes the data in blocks of 512 bits and pads the message to a multiple of 512 bits if necessary. At the bit level, SHA-256 operates on individual bits of the input message, and performs logical operations such as AND, OR, and XOR to transform the data. The algorithm also uses shift and rotate operations to manipulate the bits of the input message. These operations are performed on 32-bit words, which are represented as sequences of 0s and 1s (Michail, 2016).

The output of SHA-256 is a fixed-length hash value that is unique to the input message. This means that even a slight change in the input message will produce a completely different hash value. This property makes SHA-256 useful for verifying the integrity of data and detecting unauthorized changes.

SHA-256 is a hash function that operates on individual bits of data and produces a fixed-length hash value. At the bit level, the algorithm performs logical operations and bit manipulation to transform the input message. The resulting hash value is used to verify the integrity of data and ensure that it has not been tampered with.

Random numbers, symmetric and non-symmetric keys are all important components of modern cryptography. Random numbers are used in cryptography to generate keys and other essential elements that need to be unpredictable. Cryptographic keys must be generated from a reliable source of randomness to ensure that they cannot be easily guessed or predicted. Pseudo-random number generators (PRNGs) are commonly used to generate random numbers, but they must be seeded with an unpredictable value to be truly random (Davies, 2022).

Symmetric key encryption is a cryptography method in which the same key encrypts and decrypts the data. This means that the sender and the recipient of the data both have access to the same key, and the security of the system relies on the secrecy of the key. Symmetric key encryption is often faster and simpler than non-symmetric encryption, but it can be less secure because of the challenge of securely distributing the key to all parties who need access to it (Agrawall, 2012).

Non-symmetric (or asymmetric) key encryption, on the other hand, uses two different keys, one for encryption and another for decryption. These keys are mathematically related, but it is computationally infeasible to derive one key from the other. (“An alternative solution to the key exchange problem is to use an ...”) The most widely used non-symmetric key algorithm is the RSA algorithm, which is widely used for secure communication over the Internet. Non-symmetric key encryption is considered more secure than symmetric key encryption, but it is also slower and more complex (Mohamed, 2022).

In practice, both symmetric and non-symmetric encryption are often used together. For example, in a typical SSL/TLS connection for secure web browsing, symmetric encryption is used to encrypt the bulk of the data being transmitted, while non-symmetric encryption is used to securely exchange the symmetric key between the client and the server.

Encryption algorithms have a long and evolving history, from ancient methods of cryptography like the Caesar cipher to modern-day public key cryptography. In the mid-20th century, the development of digital computers led to the creation of modern encryption algorithms. The Data Encryption Standard (DES) was developed in the 1970s and used a 56-bit key to encrypt data. However, as computing power increased, DES became vulnerable to brute-force attacks.

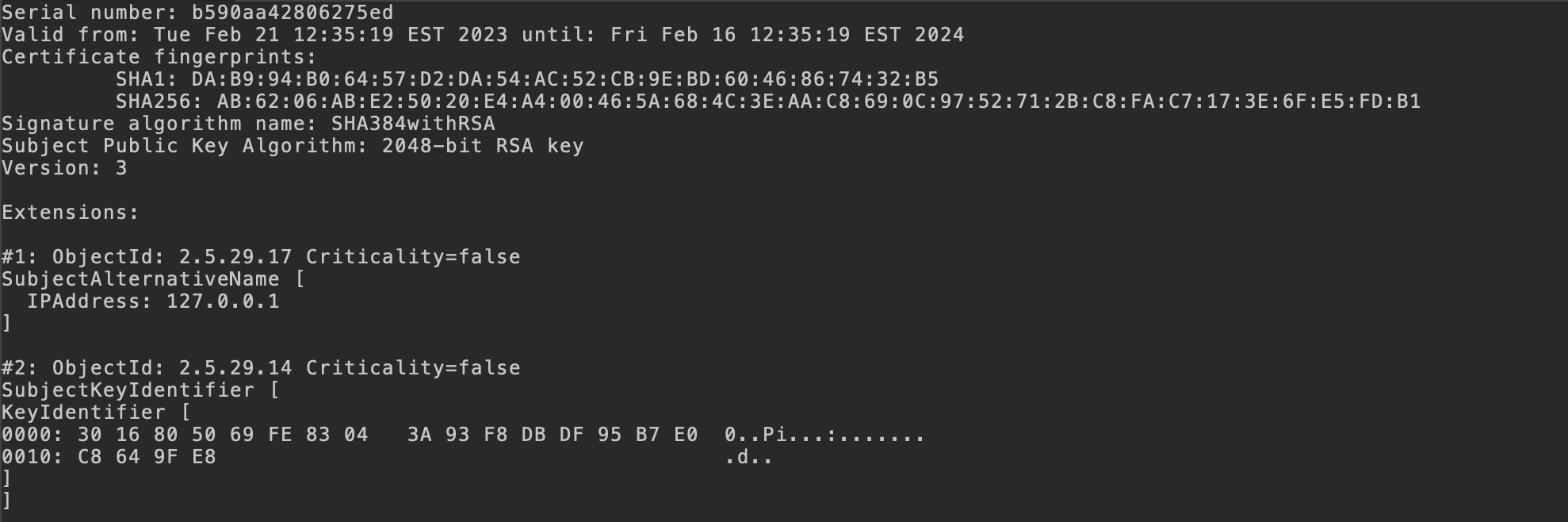
In response to this vulnerability, the Advanced Encryption Standard (AES) was developed in the late 1990s. AES uses a variable key length (128, 192, or 256 bits) and is considered very secure (*A Brief History of Cryptography*, 2023).

Another important development in the history of encryption is the creation of public key cryptography. This type of encryption uses two keys, one for encrypting data and another for decrypting it. The most widely used public key encryption algorithms are RSA and Elliptic Curve Cryptography (ECC) (*A Brief History of Cryptography*, 2023).

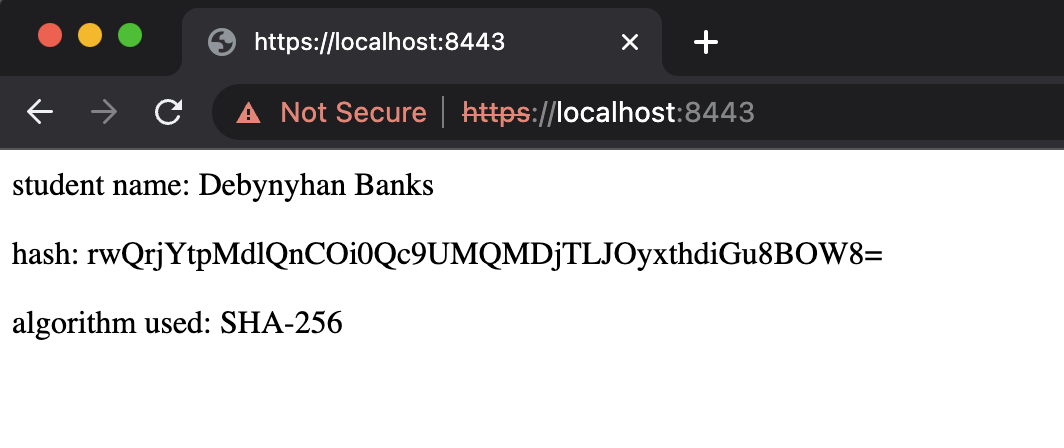
Currently, encryption algorithms are used extensively in a wide range of applications, including secure communication protocols, online banking, e-commerce, and virtual private networks (VPNs). In recent years, there has been significant interest in developing new encryption algorithms that are even more secure than current methods. For example, post-quantum cryptography is a new area of research that aims to develop encryption algorithms that are resistant to attacks by quantum computers (*A Brief History of Cryptography*, 2023).

## Certificate Generation

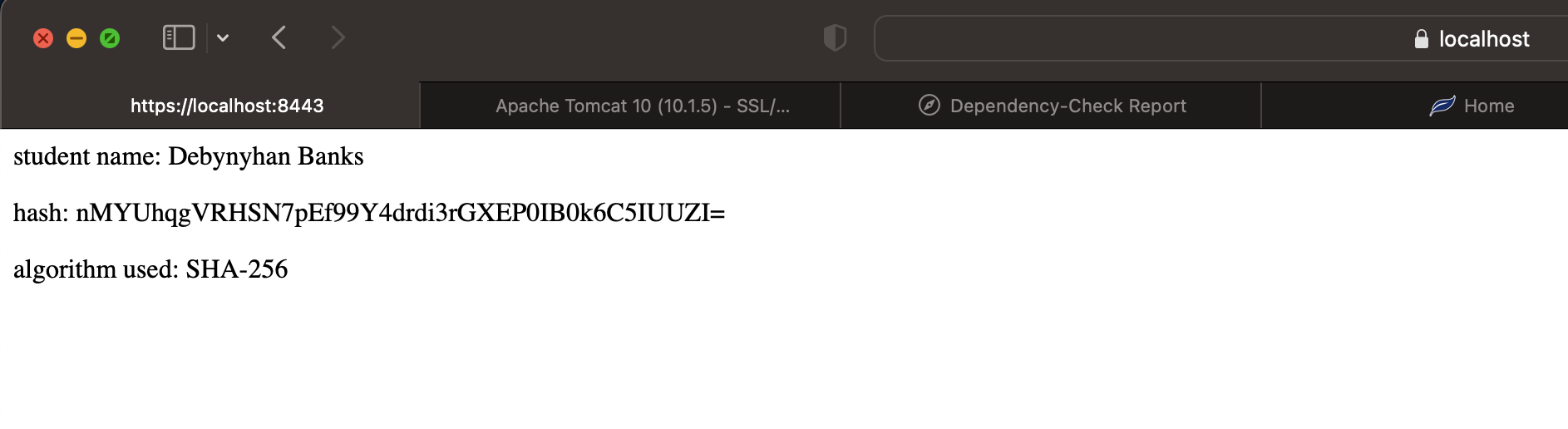
## 



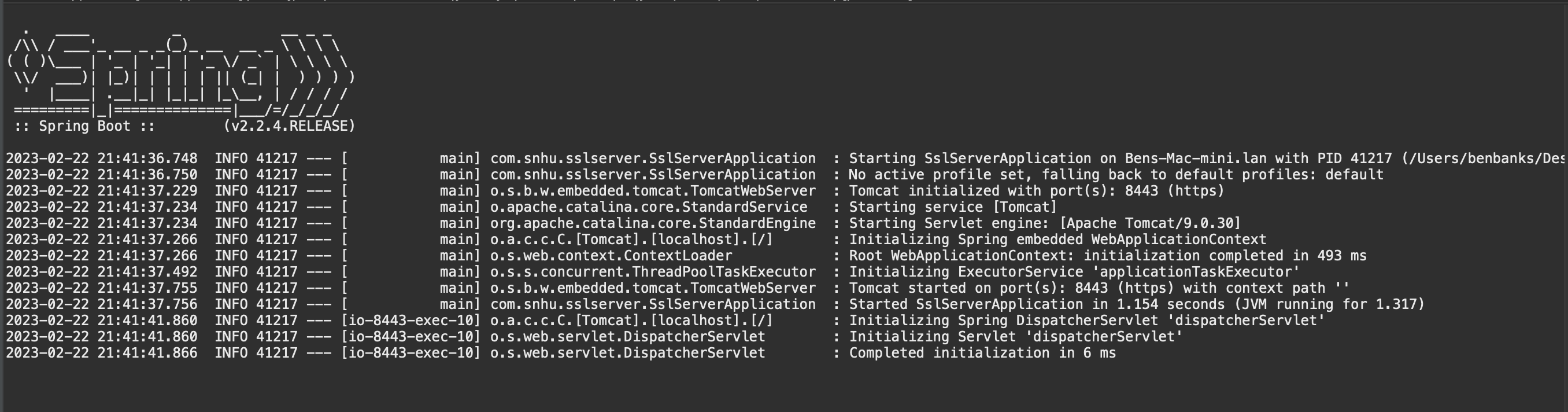
## Deploy Cipher

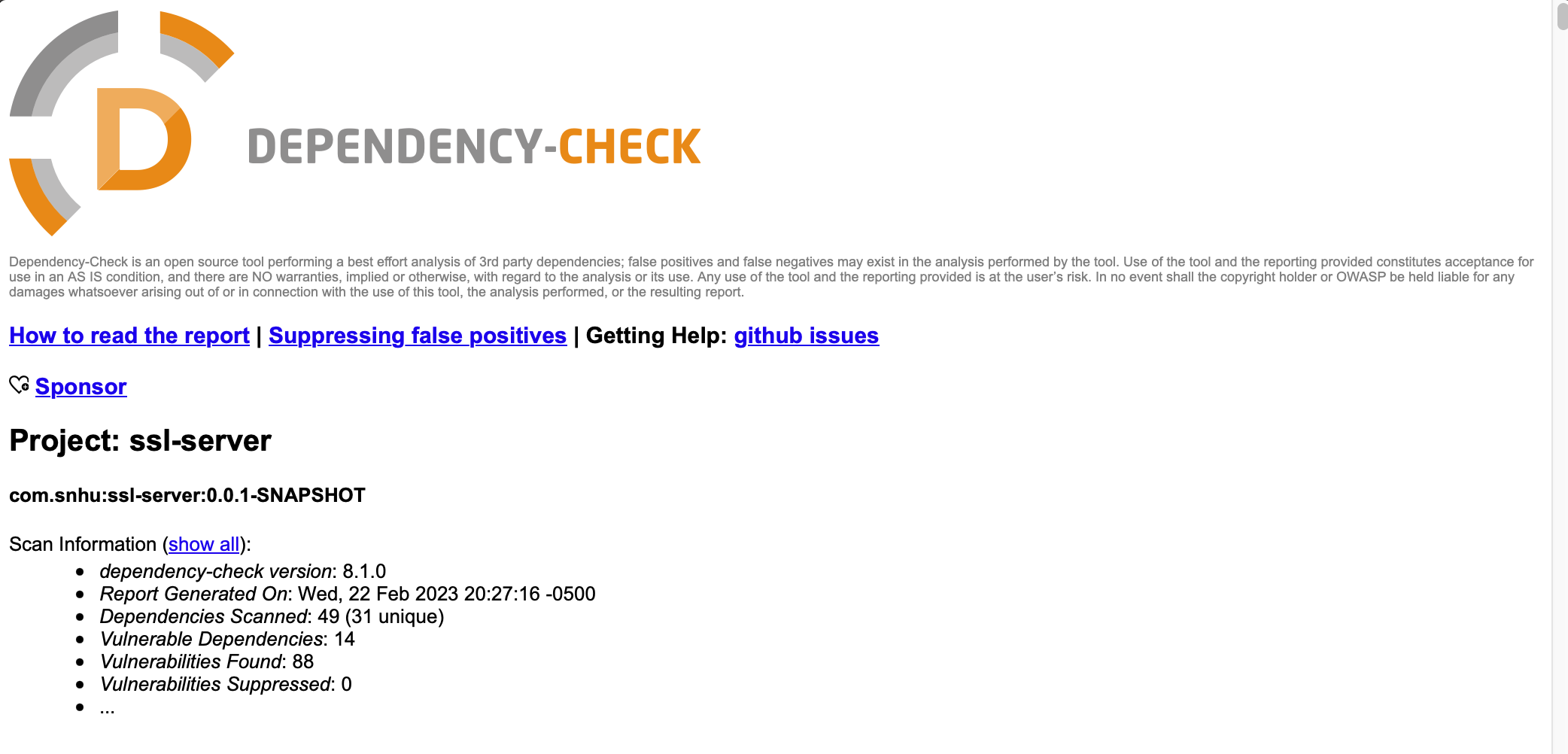


## Secure Communications



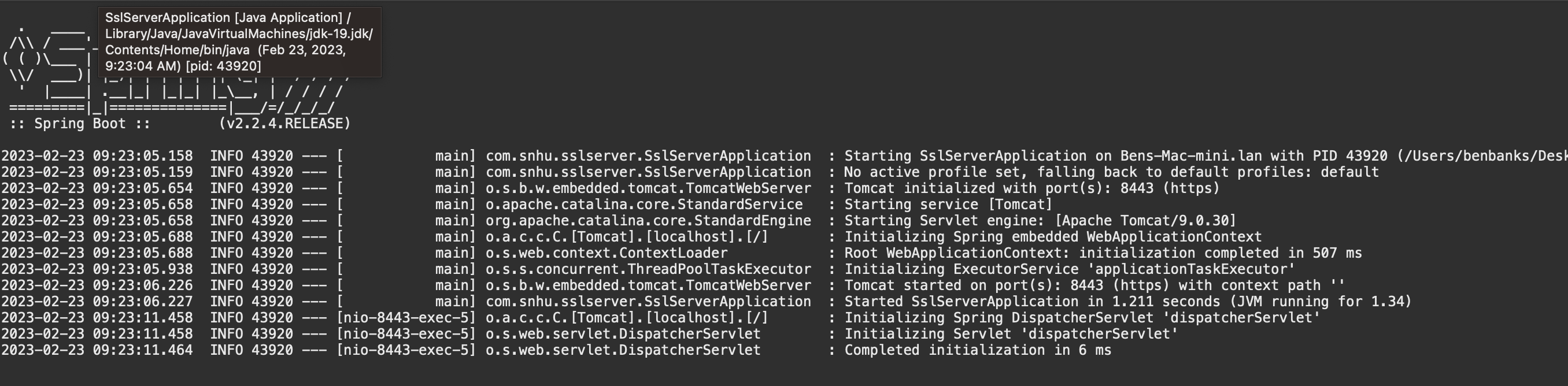
## Secondary Testing







## Functional Testing



The POM file complies with XML standards and is free of syntactical errors. Additionally, there were no logical vulnerabilities in the POM file. However, logical vulnerabilities may manifest if dependencies are added to the application, such as by utilizing outdated frameworks or dependencies that might be vulnerable. For instance, the POM file contained an outdated version of the Maven’s dependency checker (5.3.0). I refactored the POM file using the most recent release of the dependency checker (8.1.0).

## Summary

Listed below are the areas of security I addressed refactoring the code to meet secure coding best practices.

* Input validation: The refactored code creates a string by concatenating my name with a greeting message as the data. This string is then hashed and converted to a checksum. By ensuring that the input data is not tampered with before it is hashed, the code provides a level of input validation.
* Cryptography: The SHA-256 algorithm is a well-established and widely used cryptographic hash function that provides a high degree of security for hashing data. The refactored code generates a hash of the input data using the SHA-256 algorithm. This ensures that the checksum cannot be reverse engineered to obtain the original data.
* Code quality: The refactored code follows good coding practices, such as using constants for the character set and hash algorithm and using exception handling to gracefully handle errors. This helps to reduce the risk of security vulnerabilities caused by coding mistakes.
* Client-server: The refactored code uses HTTPS to secure the communication channels between the client and server with a checksum verification.
* Encapsulation: The refactored code encapsulates the hash generation and byte-to-hex conversion into separate methods. This helps improve the code's readability and maintainability and reduce the risk of errors caused by code duplication.
* API: This code defines a RESTful API using Spring Boot, I used industry best practices to ensure that the API’s were as secure as possible.

The Process of adding security layers to an application began with understanding the client requirements. After gaining an understanding of the requirements of the application, I identified potential threats and vulnerabilities from a manual review of the code. Additionally, I ran a dependency check to identify any known vulnerabilities and potential threats to the application.The next step involved reviewing financial sector policies and practices to ensure compliance with laws. I started working on the security-related area of the project once I fully understood its scope. I referenced the Vulnerability Assessment Process Flow Diagram and searched for any potential security holes in order to add security layers to the application. Then I designed effective security measures to protect the application from the threats I identified in the areas of security assessment.

## Industry Standard Best Practices

Several industry standards are implemented to maintain this application security, such as hashing with a salt value, use of a secure random number generator, use of a well-established cryptographic hash function, encoding of the salt and hash values, and proper exception handling. By using SHA-256 and a salt value, the application generates a hash of the data. Before hashing the data, salt is added as a random string of bytes. This helps prevent attackers from using precomputed hash tables to reverse the hash and obtain the original data.

I used the SecureRandom class to generate the salt value. SecureRandom is a cryptographically secure random number generator that provides a high degree of randomness, making it difficult for an attacker to predict or reproduce (Jackson, 2017). The salt and hash values are encoded using Base64 encoding. This provides a compact representation of the binary data and ensures that the data can be safely transmitted over the network. The code uses proper exception handling to catch and handle exceptions that may occur during the hashing process. Such as when the SHA-256 algorithm is not available or when the data cannot be encoded in the specified character set. By handling exceptions properly, the code helps to prevent the disclosure of sensitive information or other security vulnerabilities. Moreover, these industry standard best practices help to ensure that the application is secure and resistant to attacks. In addition, sensitive data is protected from unauthorized access, which is critical to the company’s overall wellbeing.

References:

* Youssef, A. M., Tavares, S. E., & Heys, H. M. (1996, August). A new class of substitution-permutation networks. In *Workshop on Selected Areas in Cryptography, SAC* (Vol. 96, pp. 132-147).
* Singh, G. (2013). A study of encryption algorithms (RSA, DES, 3DES and AES) for information security. *International Journal of Computer Applications*, *67*(19).
* Michail, H. E., Athanasiou, G. S., Theodoridis, G., Gregoriades, A., & Goutis, C. E. (2016). Design and implementation of totally-self checking SHA-1 and SHA-256 hash functions’ architectures. *Microprocessors and Microsystems*, *45*, 227-240.
* Davies, S. R., Macfarlane, R., & Buchanan, W. J. (2022). Comparison of Entropy Calculation Methods for Ransomware Encrypted File Identification. *Entropy*, *24*(10), 1503.
* Agrawal, M., & Mishra, P. (2012). A comparative survey on symmetric key encryption techniques. *International Journal on Computer Science and Engineering*, *4*(5), 877.
* Mohamed, K. S. (2022). Wireless Communication Systems: Confidentiality: Encryption and Decryption. In *Wireless Communications Systems Architecture: Transceiver Design and DSP Towards 6G* (p. 55-67). Cham: Springer International Publishing.
* Jackson, L. N. (2017). A Trivium-Inspired Pseudorandom Number Generator with a Statistical Comparison to the Randomness of SecureRandom and Trivium.
* *Security algorithms*, https://telcomatraining.com/security-algorithms/.
* *What is the Advanced Encryption Standard? (AES)*. (n.d.). Knowledge Base - Netify. <https://www.netify.com/knowledge-base/articles/what-is-the-advanced-encryption-standard-aes>
* *A Brief History of Cryptography*. (2023, January 12). <https://www.redhat.com/en/blog/brief-history-cryptography>