****

# 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 5](#_Toc102040759)

[3. Deploy Cipher 6](#_Toc102040760)

[4. Secure Communications 6](#_Toc102040761)

[5. Secondary Testing 6](#_Toc102040762)

[6. Functional Testing 6](#_Toc102040763)

[7. Summary 6](#_Toc102040764)

[8. Industry Standard Best Practices 6](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **Oct 14, 2024** | **Daniel Gorelkin** |  |

## 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

Daniel Gorelkin

## Algorithm Cipher

Cryptography dates back to 1900 BC, found on a tomb wall from the Old Kingdom of Egypt (Schneider, 2024). Modern-state cryptography emerged from the SHA-0 cipher back in the '90s and rapidly upgraded itself into the SHA-1, which was considered to be secured by 2010. Later, the family of SHA-2 encryption was developed to support encryption of up to 512 bits, though it was prone to vulnerabilities and became unsafe. Modern state cryptography focuses mainly on RSA encryption for asymmetric purposes and AES, based on the original Data Encryption Standard (DES) developed by IBM in the 70s. Nowadays, AES is a rapid, protected, and widespread encryption algorithm that supports up to 256-bit encryption and is best applied in symmetric encryption, where the 1.3 TLS Version protocol is a significant revision to the original TLS protocol that is intended to provide better security and reduce the number of handshake messages (TLSv1.3 Protocol, n.d.) compared to the prevalent TLSv1.2 version.

To establish a secure connection to the server a Transport Layer Security (TLS) protocol will be used to communicate with the server by validating a properly signed security certificate and by validating the signed security certificates received from the server. That will be done to ensure that the application is indeed talking to the server and that the data that is sent back and forth is authentic and has not been modified by an untrusted source. To do so, we will generate a self-signed CA certificate with an asymmetric private and a public 2048-bit pair of RSA keys where the public key is used for encryption of data or verifying digital signatures, whereas the private one is used for decrypting data.

In addition, our application will encrypt and decrypt plaintext into ciphers using a SHA-256, a 256-bit hash cipher algorithm, and a uniquely generated random key. Unlike the asymmetric key, to properly handle restful data encryption, our application will use symmetric ciphers. The symmetric encryption key will be generated and comply with the Advanced Encryption Standard (AES-256) to surpass governmental-grade encryption demands and provide the highest security level at the cost of slightly slower processing time.

Because the application will use a database and will manipulate and store sensitive information, the symmetric key, which, unlike the asymmetric key, can be used both for hashing data at rest, such as passwords, credit card numbers, and user information, and creating and validating digital signatures and certificates. However, a secure key must rely on strong randomness to ensure that private keys cannot be easily guessed and prevent brute-force and replay attacks. Since our application will rely on symmetric key usage and use unique sessions for users, it will rely on the cryptographically secure pseudorandom number generator (CPRNG) that will generate an unpredictable key to hash the data.

## Asymmetric Certificate Generation

A screenshot of a computer

Description automatically generated

A screenshot of a certificate

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A close-up of a computer screen

Description automatically generated

## Secure Communications

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

A screenshot of a computer

Description automatically generated

## Secondary Testing

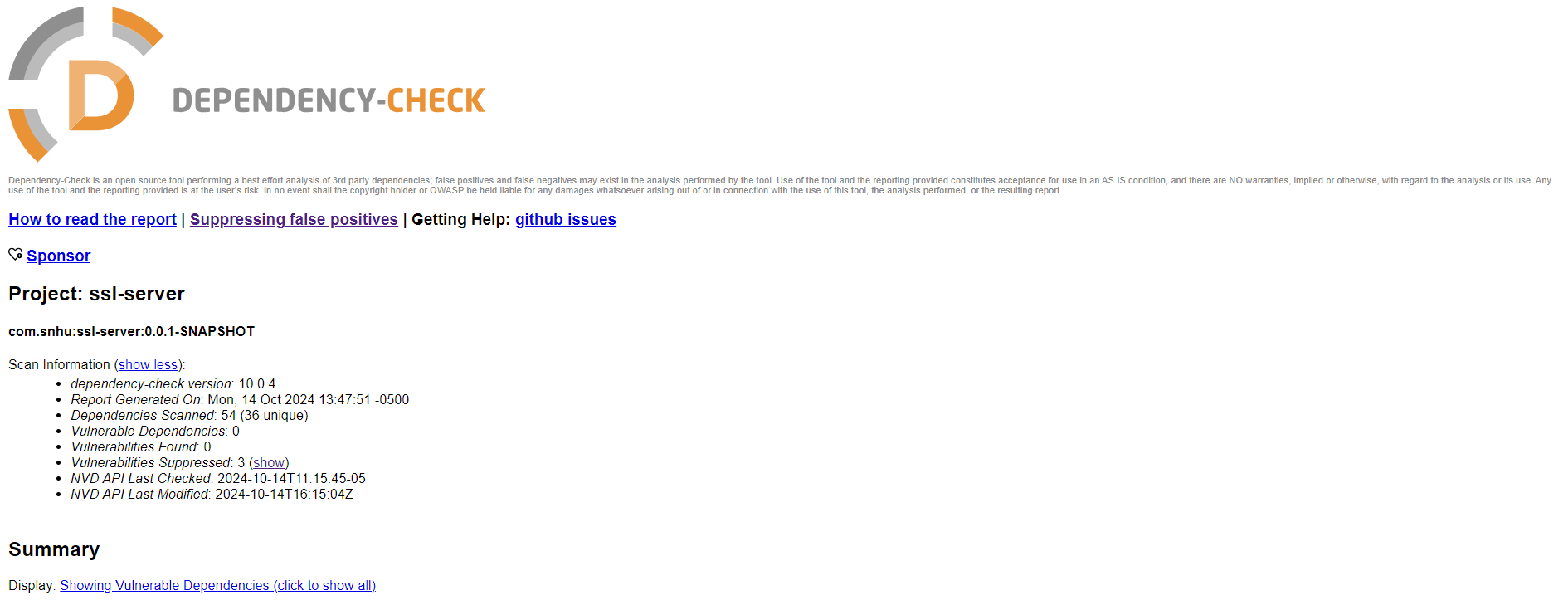
Insert screenshots below of the refactored code executed without errors and the dependency-check report.

A screenshot of a computer

Description automatically generated

A screen shot of a computer screen

Description automatically generated



## Functional Testing

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

A screenshot of a computer

Description automatically generated

## Summary

The areas of security that were addressed by refactoring the code are:

* An input validation mechanism was implemented to prevent XSS (Cross-Site Scripting) risk.
* Inadequate, null, or too-long Input length validation mechanism was implemented to prevent memory or performance issues and cause denial of service (DoS) attacks.
* The application reads the valid input and hashes it with the 256 or 512-bit length cipher to prevent exposure of that data during transportation over the network.
* The application implements error handlers such as the RuntimeException error handler to prevent the app from crashing.
* The application manages a suppression list to suppress false positive vulnerabilities.
* The application is built based on the most modern and up-to-date libraries to prevent known vulnerabilities from the National Vulnerability Database.

The process of adding security layers consists of several steps, starting from generating a key store and a self-signed CA certificate that will be validated against the trust store to check the incoming certificates from clients and verify whether the certificates are authorized. Next, by running a dependency-check report, all the security vulnerabilities were recognized, and the vulnerable libraries were updated to the safe and stable version where no threat was identified. In addition, some false positive vulnerabilities that would not cause a threat to the application were identified and suppressed in the owasp\_Suppressions.xml file. Further, the code was manually inspected and refactored to prevent dynamic vulnerabilities such as DoS and XSS risks. In addition, an input validation mechanism was implemented to sanitize input and support best practices.

## Industry Standard Best Practices

I used industry-standard best practices to maintain the software application’s existing security by enforcing security throughout the (SDLC) secure software development lifecycle by addressing security issues in the early stages of the development and integrating data protection and encryption of sensitive data. Cipher encryption with 256-bit encryption was used for data at rest and in transit. To ensure proper error handling and handle invalid input error catching techniques and input sanitization are in place. In addition, tools such as OWASP Dependency-Check were used to keep software dependencies and libraries updated. This will ensure easy updates and future modification of the code, whether future vulnerabilities will emerge. These help developers avoid common security vulnerabilities such as injection attacks, buffer overflows, and data exposure.

Therefore, applying industry-standard best practices for secure coding to the company’s overall well-being will contribute significantly by making security a shared responsibility for all team members and the potential millions of users, starting with design and continuing through coding, testing, and deployment. As a software developer, my responsibility in addressing security concerns is crucial, as the products I help create could impact human lives, manage their finances, or even be used by governmental offices; therefore, security is a top priority. For this reason, security should be a top priority from the early stages of the development cycle rather than just left to a security team or addressed later, as this can lead to vulnerabilities, deployment delays, and costly fixes. Hence, the key to applying industry standard best practices is to build security into the development process, layer by layer, ensuring that each component—whether code, APIs, or infrastructure—is secured without rushing or cutting corners to meet deadlines.

**References:**

*Java Security Standard Algorithm Names. (n.d.).* [*https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-name*](https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-name)

*Manico, J., & Detlefsen, A. (n.d.-d). Iron-Clad Java. O’Reilly Online Learning.* [*https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch06.html#ch06lev2sec5*](https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/ch06.html#ch06lev2sec5)

*Schneider, J. (2024, August 30). Cryptography history. A brief history of cryptography: Sending secret messages throughout time.* [*https://www.ibm.com/think/topics/cryptography-history*](https://www.ibm.com/think/topics/cryptography-history)

*TLSv1.3 protocol. (n.d.).* [*https://www.ibm.com/docs/en/zos/3.1.0?topic=security-using-tlsv13-protocol-support*](https://www.ibm.com/docs/en/zos/3.1.0?topic=security-using-tlsv13-protocol-support)