****

# Practices for Secure Software Report

Gonzalo Patino

CS305- Southern New Hampshire University

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 | 2024-07-09 | Gonzalo Patino | Released |

## 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

Gonzalo Patino

## Algorithm Cipher

A) AES (Advanced Encryption Standard) is a symmetric encryption algorithm widely used for secure data transmission. It operates on fixed block sizes 128 bits and can use key sizes of 128, 192, or 256 bits.

B) AES pairs with SHA-256 for hashing. SHA-256 generates a 256-bit hash, ensuring a high level of security

C) AES uses symmetric keys, meaning the same key is employed for both encryption and decryption processes. Random numbers are crucial for key generation to ensure the keys are unpredictable and secure

D) AES, established by NIST in 2001, replaced the older DES (Data Encryption Standard) due to its stronger security features and efficiency. AES is now a widely adopted encryption standard across various industries.

## Certificate Generation

A computer screen shot of a black screen

Description automatically generated

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 screenshot of a computer

Description automatically generated

## Secure Communications

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

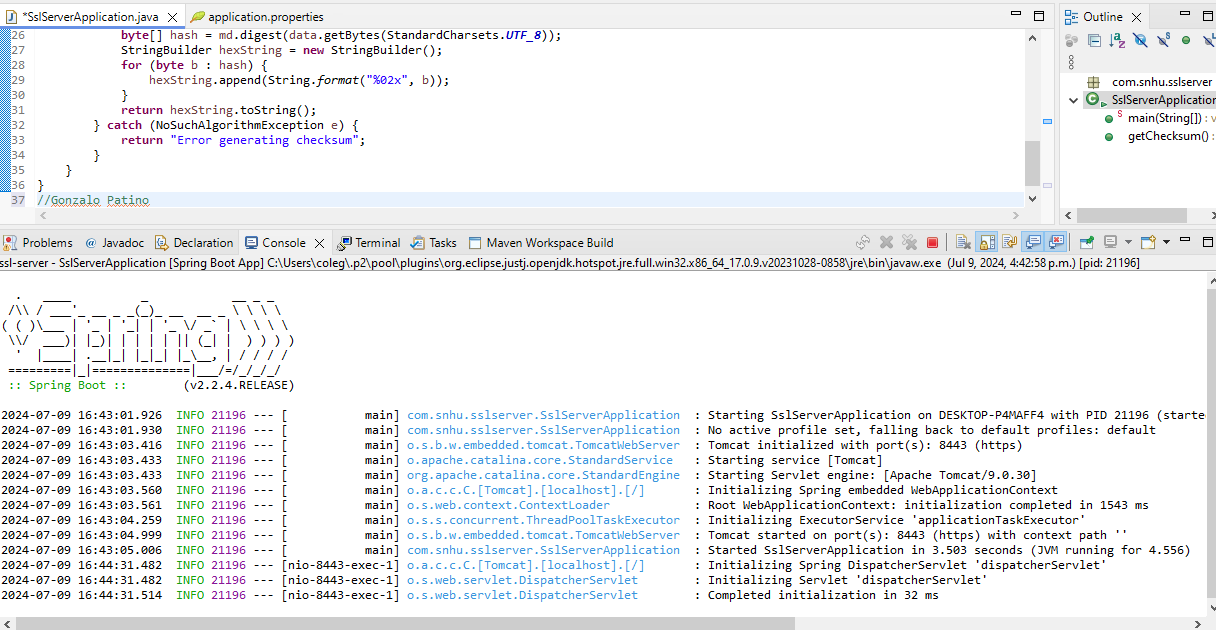
A screenshot of a computer

Description automatically generated

Figure . This is due to a self-signed certificate

## Secondary Testing

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



A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A close-up of a computer screen

Description automatically generated

## 

## Functional Testing

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

- **Application Startup:**

* The Spring Boot application was started
* The application started successfully without any errors.

A screenshot of a computer

Description automatically generated

- **Endpoint Verification:**

* The /hash endpoint was accessed via a web browser at https://localhost:8443/hash.
* The browser displayed the SHA-256 checksum of the static string "Hello World Check Sum!" confirming that the endpoint is functioning correctly.



- **Secure Connection Verification:**

* Although the browser showed a "Not secure" warning due to the self-signed certificate, the connection was encrypted, as indicated using https.

A screenshot of a computer

Description automatically generated

## Summary

In this project, we successfully configured a Spring Boot application to use HTTPS with a self-signed certificate. We generated a keystore, configured the application properties for SSL, and implemented a REST endpoint that returns the SHA-256 checksum of a static string. We verified the secure connection and checksum functionality by accessing the secure endpoint in a web browser. While the browser showed a "Not secure" warning due to the self-signed certificate, the connection was indeed encrypted. This project demonstrates the ability to secure web communications and perform basic cryptographic operations in a Java-based web application. Additionally, we ran the OWASP Dependency-Check plugin to identify and address any known vulnerabilities in our project dependencies, ensuring a more secure application overall.

## Industry Standard Best Practices

- **Use of Trusted Certificates:**

* In production environments, it's crucial to use SSL/TLS certificates signed by trusted Certificate Authorities (CAs). This ensures that browsers and clients trust the connections, preventing "Not secure" warnings.

- **Regular Key Rotation:**

* Regularly changing cryptographic keys and certificates helps maintain security by minimizing the risk of keys being compromised over time.

- **Strong Encryption Algorithms:**

* Always use strong and up-to-date encryption algorithms for SSL/TLS communications to protect against attacks. Common strong algorithms include AES and RSA.

- **Secure Configuration:**

* Ensure that both the web server and application server are securely configured. This includes disabling weak cipher suites and protocols, using secure headers, and applying the principle of least privilege to reduce potential attack surfaces.

- **Comprehensive Testing:**

* Perform thorough testing of the application, including functional testing to ensure it works as intended, security testing to find vulnerabilities, and performance testing to ensure it handles expected loads. Use automated tools for continuous testing and monitoring of the application's security.

- **Security Monitoring:**

* Implement monitoring and logging to detect and respond to security incidents quickly. Use tools to monitor traffic, log important events, and alert on suspicious activities. Regularly review logs to identify and respond to potential threats promptly.

- **Patch Management:**

* Keep software and dependencies up to date with the latest security patches. Regularly check for updates and apply them to protect against known vulnerabilities. This includes the operating system, web servers, and application frameworks.

**References:**

Detlefsen, A., & Manico, J. (2015). *Iron-Clad Java: Building Secure Web Applications*. McGraw-Hill Education.