



G L O B A L R A I N

Practices for Secure Software Report

Table of Contents

DOCUMENT REVISION HISTORY	3
CLIENT.....	3
INSTRUCTIONS.....	3
DEVELOPER	4
1. ALGORITHM CIPHER	4
2. CERTIFICATE GENERATION	5
3. DEPLOY CIPHER.....	5
4. SECURE COMMUNICATIONS	5
5. SECONDARY TESTING.....	6
6. FUNCTIONAL TESTING	7
7. SUMMARY	8
8. INDUSTRY STANDARD BEST PRACTICES	8

Document Revision History

Version	Date	Author	Comments
1.0	12/2/23	Tanner Meininger	Completed entire document by adding to every section

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

Tanner Meininger

1. Algorithm Cipher

When considering an appropriate encryption algorithm cipher for the SSL server application as seen in the provided Java Spring Boot code, the Advanced Encryption Standard (AES) stands out as a suitable choice. AES is a symmetric encryption algorithm, renowned for its combination of speed and robust security. It encrypts data in fixed-size blocks of 128 bits and is available in various key lengths, including 128, 192, and 256 bits, with AES-256 offering the highest level of security. This level of encryption is particularly effective for SSL/TLS communications, where data security and integrity are paramount.

The role of hash functions and the bit levels of the cipher are critical aspects to consider. SSL/TLS protocols commonly employ a hash function like SHA-256 which is a part of the SHA-2 family, for creating a cryptographic hash of data. These hash functions provide strong security against cryptographic attacks. The choice of AES-256 as the encryption cipher, with its 256-bit key length, ensures a high degree of security, making it computationally infeasible to crack the encryption with current technology.

Another essential consideration is the use of random numbers and the distinction between symmetric and non-symmetric (asymmetric) keys. Cryptographic operations, including those in SSL/TLS protocols, heavily rely on cryptographically secure random numbers for aspects such as key generation. AES, being a symmetric key algorithm, uses the same key for both encryption and decryption. This approach is efficient in terms of computational resources but necessitates a secure method for key exchange. Typically, this is achieved through asymmetric encryption methods like RSA during the SSL/TLS handshake process, ensuring secure key distribution.

The history and current state of encryption algorithms also play a vital role in this choice. Historically, encryption methods have evolved from simple ciphers to more complex and secure algorithms. AES was established in 2001 as a successor to the older Data Encryption Standard (DES) and has since become the standard for data encryption. Its adoption in a wide range of applications, especially in securing web communications via SSL/TLS, is a testament to its effectiveness. With the rise of quantum computing, the landscape of encryption is poised for further change, emphasizing the need for quantum-resistant algorithms. However, as of now, AES remains a highly secure and widely accepted standard in the field of cryptography.

2. Certificate Generation

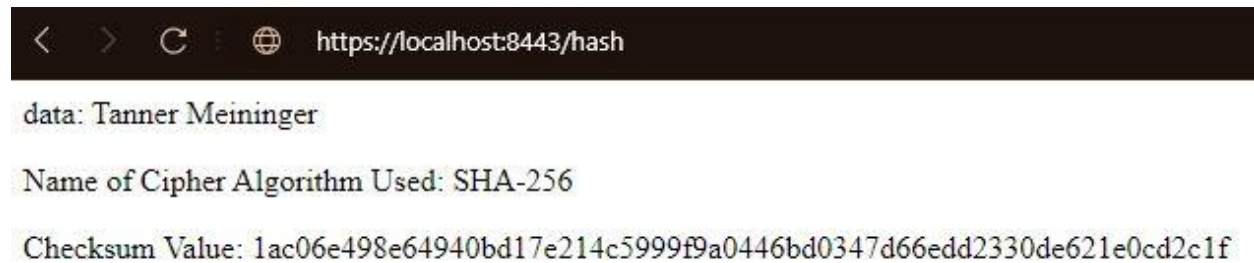
Insert a screenshot below of the CER file.

```
C:\Program Files\Java\jdk-17.0.1\bin>keytool.exe -printcert -file server.cer
Owner: CN=Tanner Meininger, OU=SNHU, O=SNHU, L=Manchester, ST=NH, C=US
Issuer: CN=Tanner Meininger, OU=SNHU, O=SNHU, L=Manchester, ST=NH, C=US
Serial number: 6a4c194e3984552a
Valid from: Thu Nov 30 19:08:27 EST 2023 until: Sun Nov 24 19:08:27 EST 2024
Certificate fingerprints:
    SHA1: 5A:1D:BD:89:F6:3B:92:8C:A9:FA:81:C7:C3:34:1B:06:F4:6E:D0:22
    SHA256: 43:BA:EB:BB:21:E0:73:B9:36:F0:19:29:78:CF:92:44:4C:B9:AC:CC:62:9A:84:0E:96:E8:16:76:55:6A:5F:83
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: C1 49 85 C1 96 A9 E7 85   AF 1F 44 57 31 2E F4 AB   .I.....Dw1...
0010: 95 AE C3 D9                ....
]
]
```

3. Deploy Cipher

Insert a screenshot below of the checksum verification.



< > ↺ 🌐 <https://localhost:8443/hash>

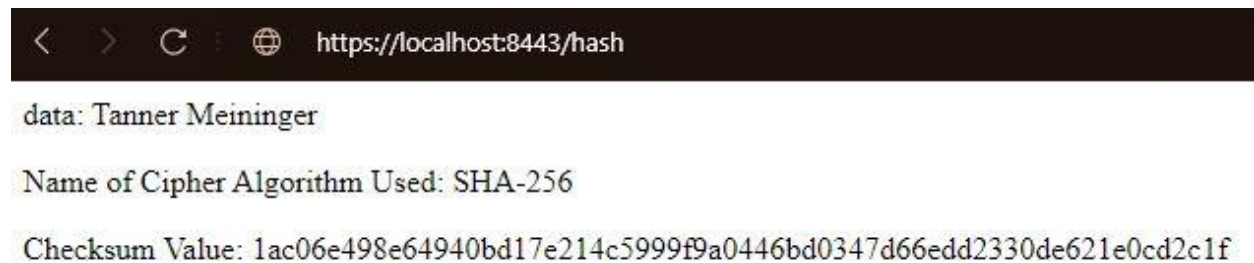
data: Tanner Meininger

Name of Cipher Algorithm Used: SHA-256

Checksum Value: 1ac06e498e64940bd17e214c5999f9a0446bd0347d66edd2330de621e0cd2c1f

4. Secure Communications

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



< > ↺ 🌐 <https://localhost:8443/hash>

data: Tanner Meininger

Name of Cipher Algorithm Used: SHA-256

Checksum Value: 1ac06e498e64940bd17e214c5999f9a0446bd0347d66edd2330de621e0cd2c1f

5. Secondary Testing

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

```
@RestController
class ServerController {

    @RequestMapping("/hash")
    public String myHash() {
        try {
            String data = "Tanner Meininger";
            String checksumMethod = "SHA-256";

            // Create checksum directly on the data
            MessageDigest digest = MessageDigest.getInstance(checksumMethod);
            byte[] hash = digest.digest(data.getBytes());

            // Convert byte array to hex string
            StringBuilder hexString = new StringBuilder();
            for (byte b : hash) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) hexString.append('0');
                hexString.append(hex);
            }

            // Return HTML formatted response
            return "<div>" +
                "<p>data: " + data + "</p>" +
                "<p>Name of Cipher Algorithm Used: " + checksumMethod + "</p>" +
                "<p>Checksum Value: " + hexString.toString() + "</p>" +
                "</div>";
        } catch (NoSuchAlgorithmException e) {
            return "<p>Error generating checksum</p>";
        }
    }
}
```

Project: ssl-server

com.snhu:ssl-server:0.0.1-SNAPSHOT

Scan Information ([show all](#)):

- dependency-check version: 9.0.2
- Report Generated On: Thu, 7 Dec 2023 18:42:09 -0500
- Dependencies Scanned: 46 (26 unique)
- Vulnerable Dependencies: 1
- Vulnerabilities Found: 1
- Vulnerabilities Suppressed: 2 ([show](#))
- ...

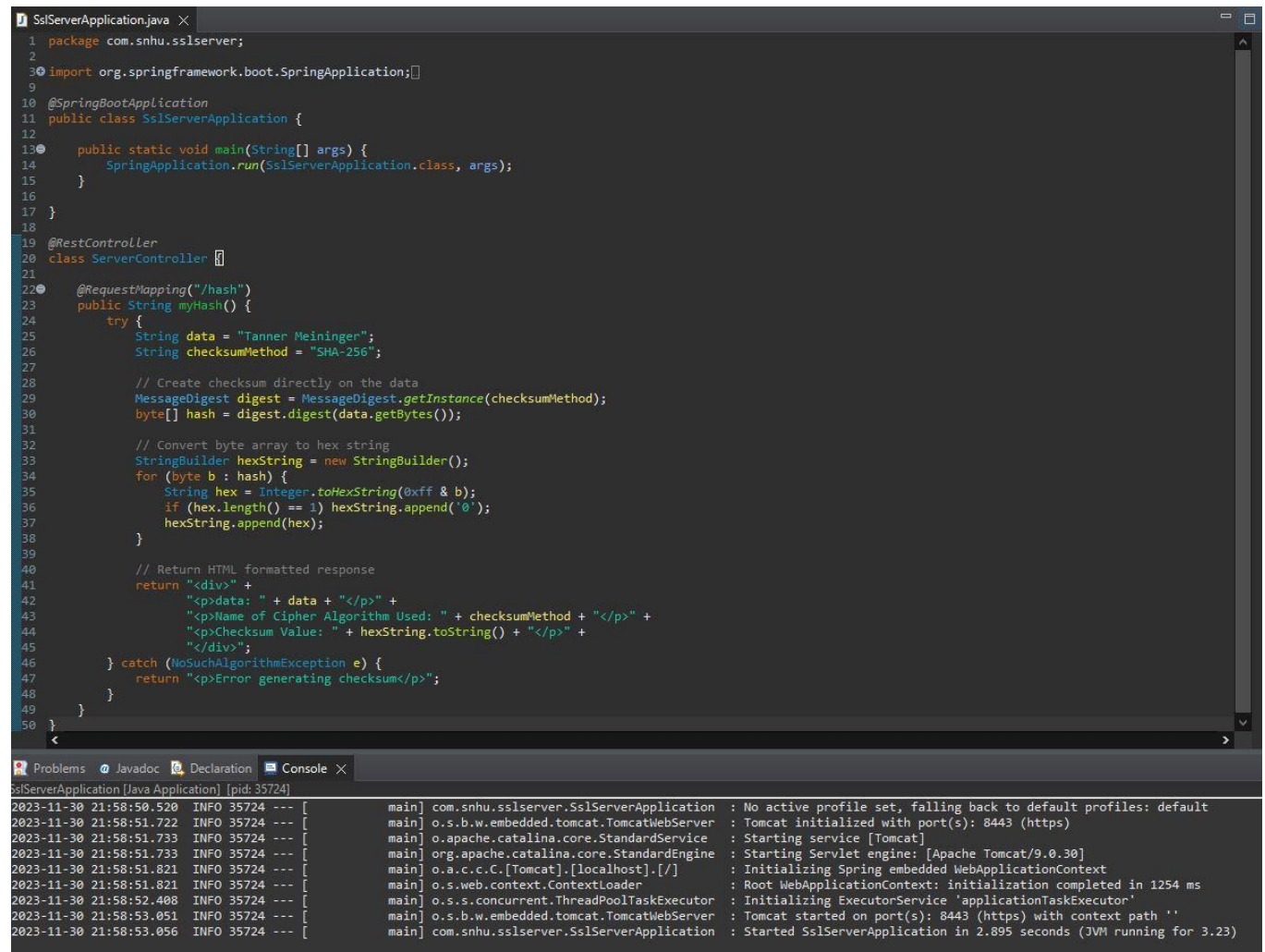
Summary

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
logback-classic-1.4.11.jar	cpe:2.3:a:qos.logback:1.4.11:20130808:1:OSS	pkg.maven/ch.qos.logback/logback-classic@1.4.11	HIGH	1	Highest	38

6. Functional Testing

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



The screenshot displays an IDE with two panels. The top panel shows the source code for `SslServerApplication.java`, which is a Spring Boot application. It includes a `main` method and a `ServerController` with a `myHash` endpoint. The code uses `MessageDigest` for SHA-256 hashing and returns an HTML response. The bottom panel shows the console output, indicating a successful startup of the application on port 8443.

```
1 package com.snhu.sslserver;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7
8
9
10 @SpringBootApplication
11 public class SslServerApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(SslServerApplication.class, args);
15     }
16
17 }
18
19 @RestController
20 class ServerController {
21
22     @RequestMapping("/hash")
23     public String myHash() {
24         try {
25             String data = "Tanner Meininger";
26             String checksumMethod = "SHA-256";
27
28             // Create checksum directly on the data
29             MessageDigest digest = MessageDigest.getInstance(checksumMethod);
30             byte[] hash = digest.digest(data.getBytes());
31
32             // Convert byte array to hex string
33             StringBuilder hexString = new StringBuilder();
34             for (byte b : hash) {
35                 String hex = Integer.toHexString(0xff & b);
36                 if (hex.length() == 1) hexString.append('0');
37                 hexString.append(hex);
38             }
39
40             // Return HTML formatted response
41             return "<div>" +
42                 "<p>data: " + data + "</p>" +
43                 "<p>Name of Cipher Algorithm Used: " + checksumMethod + "</p>" +
44                 "<p>Checksum Value: " + hexString.toString() + "</p>" +
45                 "</div>";
46         } catch (NoSuchAlgorithmException e) {
47             return "<p>Error generating checksum</p>";
48         }
49     }
50 }
```

Console Output:

```
2023-11-30 21:58:50.520 INFO 35724 --- [main] com.snhu.sslserver.SslServerApplication : No active profile set, falling back to default profiles: default
2023-11-30 21:58:51.722 INFO 35724 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8443 (https)
2023-11-30 21:58:51.733 INFO 35724 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-11-30 21:58:51.733 INFO 35724 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.30]
2023-11-30 21:58:51.821 INFO 35724 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-11-30 21:58:51.821 INFO 35724 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 1254 ms
2023-11-30 21:58:52.408 INFO 35724 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2023-11-30 21:58:53.051 INFO 35724 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8443 (https) with context path ''
2023-11-30 21:58:53.056 INFO 35724 --- [main] com.snhu.sslserver.SslServerApplication : Started SslServerApplication in 2.895 seconds (JVM running for 3.23)
```

7. Summary

Within the project, I refactored the application, focusing heavily on enhancing security in line with the stages of the Vulnerability Assessment Process Flow Diagram. With Spring Boot as the framework, it allows for strong security features. One of the key security implementations in my code is the use of SHA-256 for hashing. This ensures that our data integrity is maintained, and it's a standard practice for secure applications. I updated the maven plugin, spring boot, and the java version all to the most current version, which removed all of the vulnerabilities besides two that have just been found within the last few days. I also made sure to organize and encapsulate the code efficiently, which is crucial for maintaining a high standard of code quality and minimizing vulnerabilities. Importantly, I was careful to avoid any exposure of sensitive data within the code.

8. Industry Standard Best Practices

When writing this section of the application, I adhered to industry-standard best practices for secure coding. This includes using the latest and most secure frameworks available, such as Spring Boot, which is regularly updated with security patches. I also implemented SHA-256 for hashing, emphasizing our focus on a strong data security. My approach to coding, with an emphasis on clarity and structure, reduces the risk of security flaws and is a crucial part of maintaining the application's integrity. This method not only helps the security of the application but also enhances the Artemis Financials reputation for safety and compliance, leading to long-term benefits in both trust and cost efficiency. This proactive approach to software development and security is especially relevant in the competitive tech environment we operate in.