****

# Practices for Secure Software Report

Table of Contents

[Practices for Secure Software Report 1](#_Toc133188931)

[Document Revision History 3](#_Toc133188932)

[Client 3](#_Toc133188933)

[Instructions 3](#_Toc133188934)

[Developer 4](#_Toc133188935)

[1. Algorithm Cipher 4](#_Toc133188936)

[Justification 8](#_Toc133188937)

[Generate Checksum 9](#_Toc133188938)

[Verification 11](#_Toc133188939)

[2. Certificate Generation 12](#_Toc133188940)

[3. Deploy Cipher 14](#_Toc133188941)

[4. Secure Communications 15](#_Toc133188942)

[5. Secondary Testing 16](#_Toc133188943)

[6. Functional Testing 17](#_Toc133188944)

[7. Summary 17](#_Toc133188945)

[8. Industry Standard Best Practices 18](#_Toc133188946)

[9. References 19](#_Toc133188947)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **23-04-2023** | **Hamza Malik** |  |

## 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

Hamza Malik

## Algorithm Cipher

Here are some important aspects to take into account when choosing an encryption algorithm cypher:

* Security Level: To ensure that it is impervious to threats and weaknesses, the cypher should have a high level of security strength.
* Performance: The cypher should be capable in terms of speed, memory consumption, and scalability.
* The cypher must work with the platforms and operating systems that will be used to implement it.
* Usability: The cypher should to fit seamlessly into the system and be simple to use without adding extra complexity.

Several of the suggested encryption algorithm cyphers are based on the Java Security Standard Algorithm Names supplied by Oracle:

* AES (Advanced Encryption Standard) (Advanced Encryption Standard)
* Blowfish
* Twofish
* Triple DES RSA (Rivest-Shamir-Adleman) (Data Encryption Standard)

The most popular and currently most secure encryption algorithm cypher is AES. It provides a strong level of security and acceptable performance. It is also simple to use, integrate, and compatible with a variety of platforms and systems. As a result, AES is frequently a wise choice in a variety of circumstances, including those involving sensitive data and strong security requirements.

It is crucial to take into account both the specific security assaults that a certain encryption algorithm cypher can fight off when choosing which one to use. Consider the following typical security attacks, and how a suitable encryption algorithm cypher can help you defend against them:

* Attacks using "brute force" try every conceivable combination in an effort to decipher the encryption key. The use of a robust encryption algorithm cypher with a key length that is sufficient can assist defend against brute force assaults.
* Man-in-the-Middle Attacks: In this type of attack, a third party listens in on two parties' communications and reads, modifies, or inserts new messages. a secure key exchange protocol-based encryption algorithm cypher, such as Diffie-Hellman, can help defend against man-in-the-middle attacks.
* Dictionary Attacks: In a dictionary attack, the encryption key is tested using well-known words or expressions. Dictionary attacks can be defended against by using a powerful encryption algorithm cypher that uses salted hashing methods, such as bcrypt or scrypt.
* Side Channel Attacks: These attacks take use of flaws in the encryption algorithm's physical implementation, such as excessive power use or electromagnetic emissions. It is possible to defend against these attacks by using a safe implementation of the encryption method cypher, such as one that is impervious to side channel attacks.
* Malware Attacks: Malware can be used to intercept encrypted data before it is provided for decryption or to steal encryption keys. Using a powerful encryption algorithm cypher and adding other security controls, including secure boot and antivirus software, will assist prevent malware attacks.

The specific risks and weaknesses of Artemis Financials’ systems should be taken into account while choosing an encryption algorithm cypher. It is challenging to determine all the possible threats that an encryption algorithm cypher can fend off without being aware of those specific risks. Nonetheless, by taking into account the aforementioned considerations, choosing a robust encryption algorithm cypher with a long enough key length, and putting extra security measures in place, it is possible to assist defend against many typical types of security assaults.

It is crucial to take into account the precise legal requirements relevant to the intended use case when determining if a certain encryption algorithm cypher complies with the most recent government legislation. Regulations and standards for the protection of sensitive information may vary across businesses and nations.

It is crucial to confirm that the encryption algorithm cypher chosen for Artemis Financial complies with all relevant laws and standards, including those mentioned above. Using an accepted, industry-standard encryption algorithm cypher, like AES, can aid in ensuring compliance with the necessary rules and specifications.

It will rely on the particular requirements of Artemis Financials’ systems and applications how the encryption algorithm cypher is used. Sensitive data, such as personally identifiable information (PII), financial transactions, or other secret information, can be safeguarded using the encryption algorithm cypher. When sending data across open networks, such the internet or wireless networks, the encryption algorithm cypher can also be used to encrypt the data as it is being sent. Moreover, obligations for the protection of sensitive healthcare and personal data are contained in specific rules like HIPAA and GDPR.

The specific needs, weaknesses, and performance standards of a system determine the encryption to use. Nonetheless, there are a few well-known and regularly used cyphers that provide a high level of security strength and are frequently used in protocols and applications that are considered industry standards.

AES (Advanced Encryption Standard), for instance, is a popular symmetric encryption algorithm cypher that is regarded as being safe, quick, and effective. Many security professionals and organizations, including the National Institute of Standards and Technology, promote and use AES (NIST). Asymmetric encryption algorithms like RSA (Rivest-Shamir-Adleman) are frequently employed in key-exchange protocols, digital signatures, and other cryptographic applications.

It is crucial to strike a compromise between security strength and other considerations like performance, compatibility, and usability when choosing a cypher. Although using the most secure encryption can seem like the ideal option, there are a number of reasons why this may not always be the case:

* Performance: The speed and processing power requirements of some of the most secure cyphers, such as those with longer key lengths or sophisticated algorithms, may affect their usability and performance.
* Compatibility: Certain cyphers might not work with all operating systems, software, or applications, which could cause compatibility problems and add complexity.
* Usability: Some cyphers might be harder to set up, configure, or incorporate into current systems, which could have an effect on usability and raise the possibility of implementation errors.
* Cost: The license fees or additional hardware or software components needed for some cyphers might raise the implementation and maintenance costs.

It is essential to choose a cypher that strikes a compromise between security strength and other aspects like performance, compatibility, and usability. The best cypher for a given use case can be chosen with the aid of a thorough risk analysis and discussions with security professionals.

## Justification

The goal of an encryption algorithm cypher is to convert plain text into ciphertext, an unintelligible and useless form of data. The cipher's hash algorithms and bit levels serve to secure and effectively safeguard the ciphertext through cryptography.

To guarantee the validity and integrity of the encrypted data, hash algorithms are employed. A hash function converts data from an input (such as a message or file) into a fixed-size output (a hash value). It should be difficult to extract the original data from the hash result because the hash function is intended to be one-way. The integrity and validity of data are checked using hash functions in digital signatures, password storage, and other cryptographic applications.

Bit levels describe the size of the cipher's encryption key. Since there are more potential keys that may be used to decipher the ciphertext, a greater key size typically equates to a higher level of security. A 128-bit key, for instance, has 2128 potential key combinations, making it more challenging for an attacker to guess the key or use brute force.

Generally, an encryption algorithm cipher's hash functions and bit levels are crucial elements that assist maintain the security and integrity of encrypted data. Artemis Financial can aid in preventing unauthorized access to its sensitive data by choosing an encryption algorithm cypher with strong hash functions and bit levels, such as AES with 256-bit key size.

Important elements of encryption techniques include random numbers, symmetric keys, and non-symmetric keys. Here is a description of each:

In encryption techniques, random numbers are used to generate unpredictable and distinctive keys, initialization vectors, and other data. This helps shield against attackers estimating or forecasting these values, which would enable them to crack the ciphertext. Several techniques, such as hardware-based generators, software-based generators, or a combination of both, are used to produce random numbers.

Symmetric keys: With symmetric key algorithms, data is encrypted and decrypted using the same key. This means that for secure communication to occur, both the sender and the recipient must possess the same key. A safe method of key distribution is necessary for symmetric key algorithms since they are typically quicker and more effective than asymmetric key algorithms.

Non-symmetric keys: non-symmetric key algorithms, also called asymmetric key algorithms, encrypt and decrypt data using two distinct keys. The private key is used to decode data, whereas the public key is used to encrypt data. This implies that only the recipient, who is in possession of the private key, can decrypt data once it has been encrypted by the sender using the recipient's public key.

The use of cyphers and codes to secure sensitive information stretches back to ancient times, when encryption algorithms first appeared. The advent of modern cryptography in the 20th century led to an increase in complexity and sophistication of encryption algorithms over time. The National Institute of Standards and Technology (NIST) created the Data Encryption Standard (DES) in the 1970s as a symmetric key encryption method. However, DES was eventually discovered to be attackable, inspiring the creation of the Advanced Encryption Standard (AES) in the 2000s, which is now thought to be secure and extensively used.

## Generate Checksum

package com.snhu.sslserver;

import java.math.BigInteger;

import java.nio.charset.StandardCharsets;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

public class ServerApplication {

public static void main(String[] args) {

SpringApplication.run(ServerApplication.class, args);

}

}

@RestController

class ServerController{

public static String calculateHash(String name) throws NoSuchAlgorithmException

{

MessageDigest md = MessageDigest.getInstance("SHA-256");

byte[] hash = md.digest(name.getBytes(StandardCharsets.UTF\_8));

BigInteger number = new BigInteger(1, hash);

StringBuilder hexString = new StringBuilder(number.toString(16));

while (hexString.length() < 32)

{

hexString.insert(0, '0');

}

return hexString.toString();

}

//FIXME: Add hash function to return the checksum value for the data string that should contain your name.

@RequestMapping("/hash")

public String myHash() throws NoSuchAlgorithmException{

String data = "Hello Hamza Malik";

String hash = calculateHash(data);

return "<p>data:"+data+" : SHA-256 "+" : "+hash;

}

}

## Verification

Insert a screenshot below of the web browser with your unique information.

## Certificate Generation

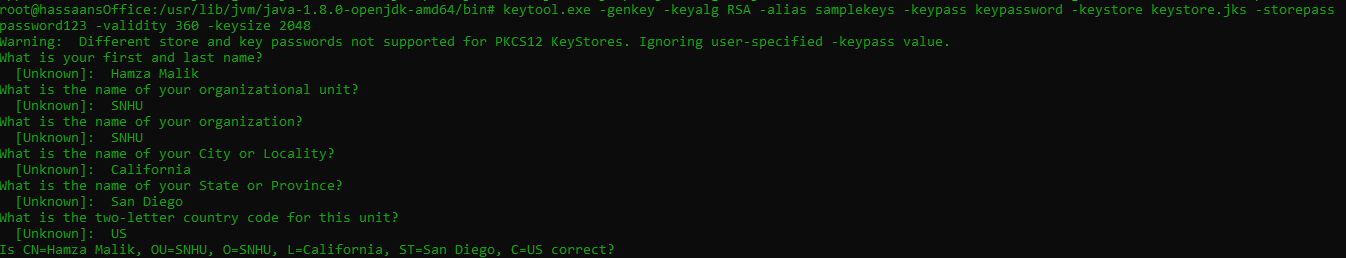
An organization called a Certificate Authority (CA) is in charge of managing and issuing the digital certificates that are used to encrypt online communications. A CA's primary function is to authenticate users, organizations, or devices in order to ensure that the data being communicated over the internet is safe and cannot be intercepted by unauthorized individuals or entities.

* Authenticity: A CA is a dependable third party that confirms the identity of the certificate holder to make sure it is authentic and hasn't been altered in any way. The parties to the communication can be confident that the information being communicated is secure, which promotes trust between them.
* Confidentiality: Digital certificates use encryption to safeguard transferred data, guaranteeing that only the intended receiver can access it. This makes it harder to eavesdrop or intercept data in other ways.
* Digital certificates offer a method for confirming the identity of the certificate holder, guaranteeing that only authorized parties can access sensitive information or carry out certain tasks.
* Compliance: Many industries and organizations have regulatory requirements for data security, and using a CA can help to ensure compliance with these requirements.

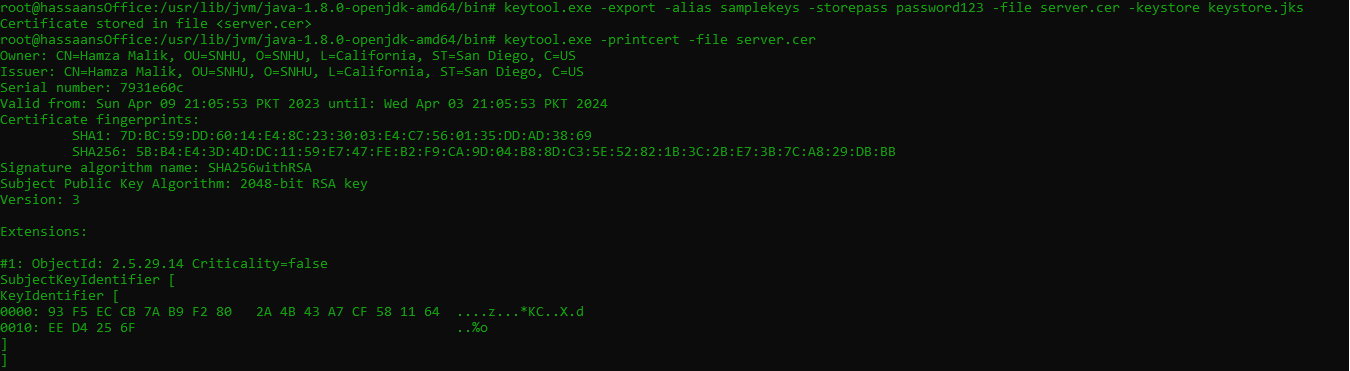
Using a third-party vendor CA like Let's Encrypt can offer other advantages in addition to these, including the following:

* Cost-effectiveness: Because Let's Encrypt offers free SSL/TLS certificates, it is a viable alternative for individuals and small organizations who may not be able to afford the fees associated with a traditional provider.
* Convenience: Let's Encrypt makes it simple for website owners to get and keep SSL/TLS certificates by offering automated certificate issue and renewal.
* Broad support: Let's Encrypt is a well-liked option for website owners and developers because it is supported by many web hosting companies and software programmers.

Self-signed certificates might, however, be sufficient in some circumstances, such as when creating a website or testing a new application. When this occurs, programmers can take on the role of the CA and create their own certificates using open-source software. Self-signed certificates, however, should not be used in production settings where security is a top priority because they do not offer the same level of trust and security as certificates issued by an independent CA.



Keytool will create a new key pair and self-signed certificate after you issue this command, and it will save them in a new keystore file called "keystore.jks" in the location where you are currently working. The private key corresponding to the key pair that is generated will also be protected with the password "password," just like the keystore file is. The self-signed certificate will utilize an RSA key with a length of 2048 bits and be valid for 360 days.



Keytool will print the specifics of the certificate found in the "server.cer" file after you run this command. This comprises details like the topic and issuer of the certificate, its validity dates, and the chosen public key algorithm. This information can be helpful for both resolving SSL/TLS connection issues and confirming the validity of a digital certificate.

## Deploy Cipher

package com.snhu.sslserver;

import java.math.BigInteger;

import java.nio.charset.StandardCharsets;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

public class ServerApplication {

public static void main(String[] args) {

SpringApplication.run(ServerApplication.class, args);

}

}

@RestController

class ServerController{

public static String calculateHash(String name) throws NoSuchAlgorithmException

{

MessageDigest md = MessageDigest.getInstance("SHA-256");

byte[] hash = md.digest(name.getBytes(StandardCharsets.UTF\_8));

BigInteger number = new BigInteger(1, hash);

StringBuilder hexString = new StringBuilder(number.toString(16));

while (hexString.length() < 32)

{

hexString.insert(0, '0');

}

return hexString.toString();

}

//FIXME: Add hash function to return the checksum value for the data string that should contain your name.

@RequestMapping("/hash")

public String myHash() throws NoSuchAlgorithmException{

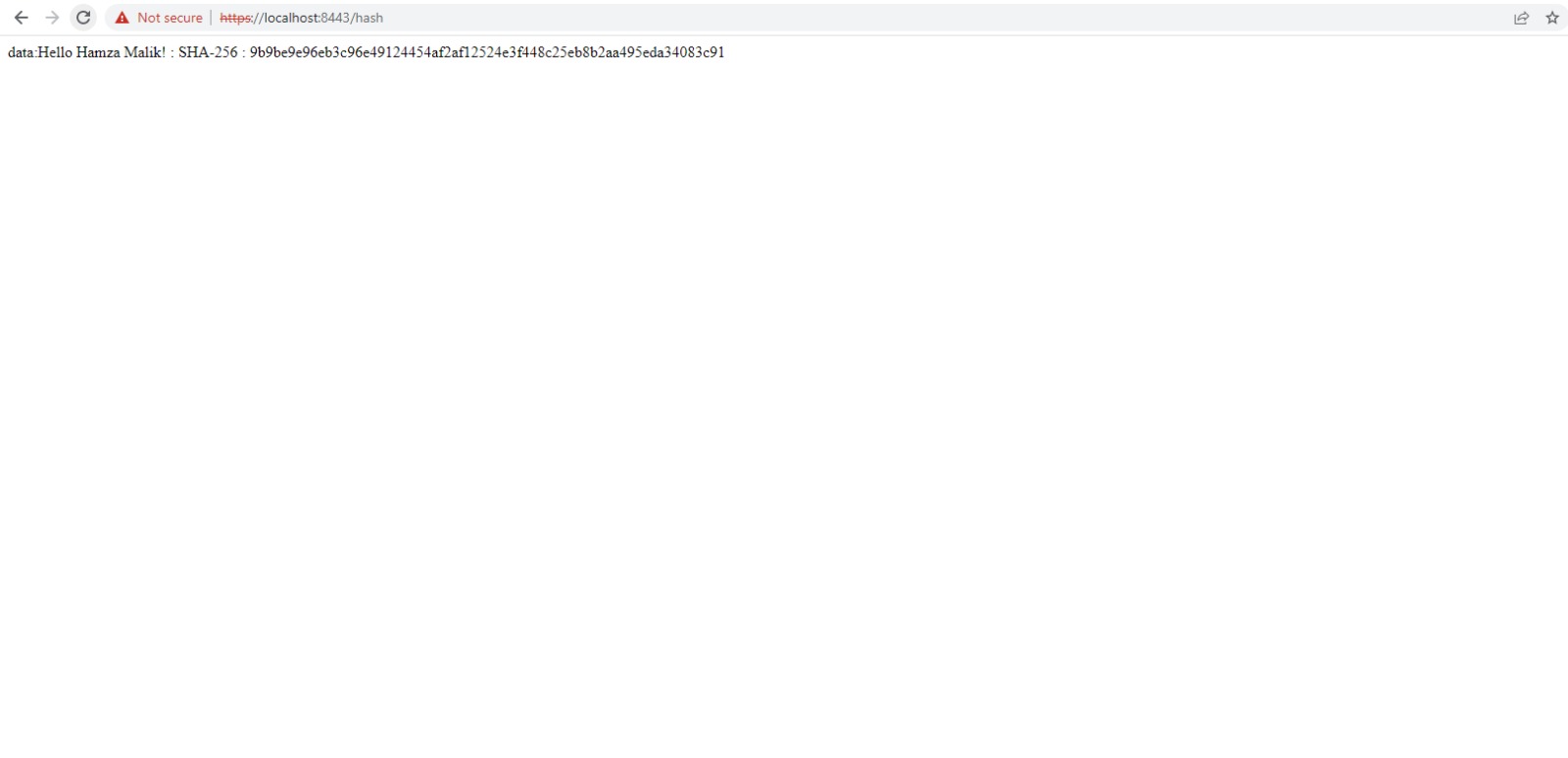
String data = "Hello Hamza Malik";

String hash = calculateHash(data);

return "<p>data:"+data+" : SHA-256 "+" : "+hash;

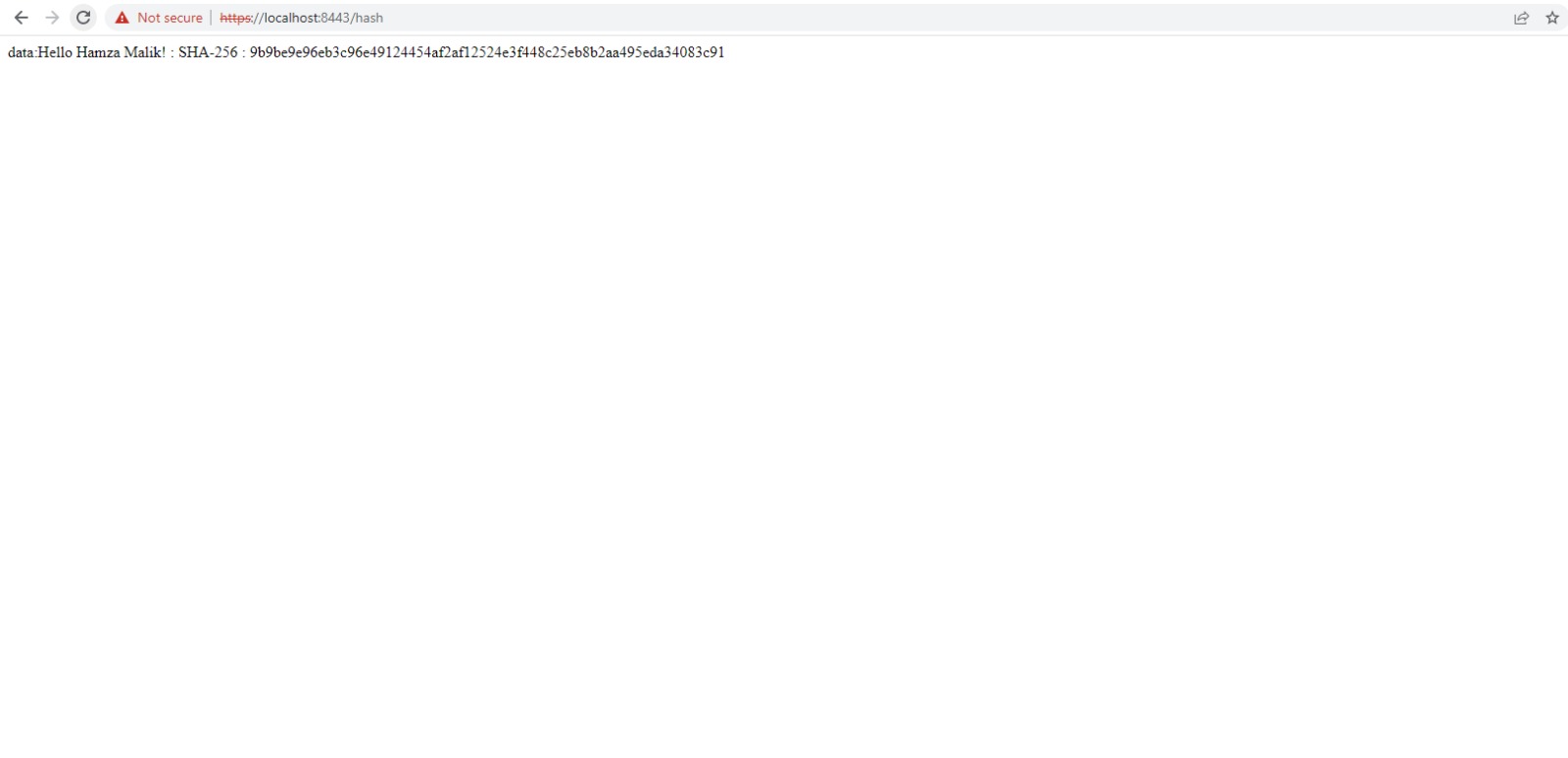
}

}

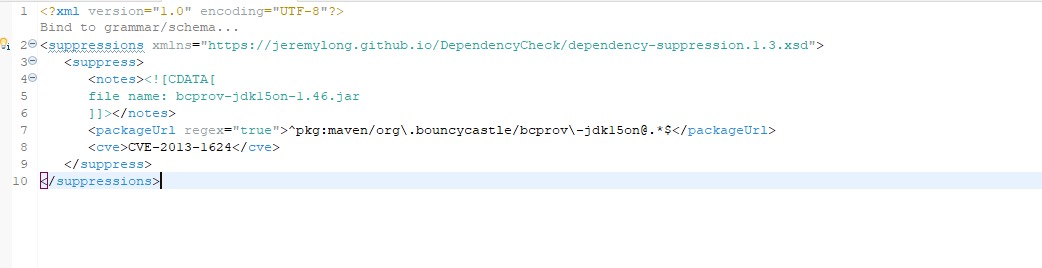


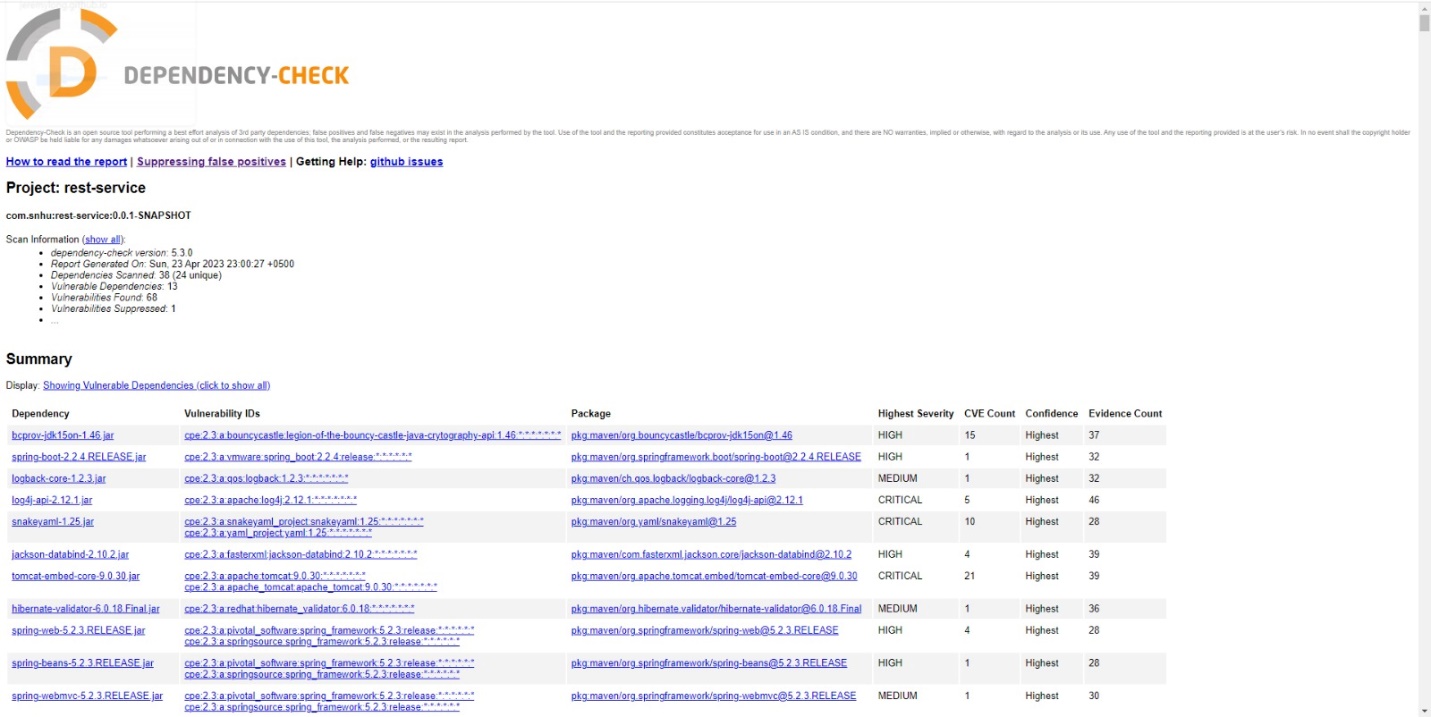
## Secure Communications

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



## Secondary Testing





## Functional Testing

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



## Summary

By fixing the security flaws shown in the Vulnerability Assessment Process Flow Diagram, he refactored code conforms with security testing methods.

In order to further secure the software programmed, a cryptographic hash method employing SHA-256 was first used. This was accomplished through code reworking and the use of the hash technique as a checksum verification. Data integrity is guaranteed throughout transmission thanks to the use of a cryptographic hash technique, which also adds an extra degree of protection against data tampering.

Second, the HTTPS protocol was used to transform HTTP to ensure secure connection. To guarantee that the server was using the HTTPS protocol, the code in the application. Properties file was refactored and compiled. This makes sure that all data sent between the client and server is encrypted and safe from eavesdropping and interception.

The refactored code was then subjected to secondary static testing using the OWASP Dependency-Check Maven. This was done to make sure that the refactoring procedure didn't result in the introduction of any new security vulnerabilities. To find any syntactical, logical, and security flaws that the static testing had missed, a thorough review of the code was also carried out.

## Industry Standard Best Practices

I used accepted industry best practices for secure coding to keep the software application's security as it was. These techniques consist of:

* upgrading dependencies and libraries frequently to make sure that the software is not exposed to known security flaws.
* Protecting sensitive data in transit and at rest by using powerful encryption algorithms like the Advanced Encryption Standard (AES) and Secure Hash Algorithm (SHA).
* putting in place secure communication protocols, such HTTPS, to stop eavesdropping and man-in-the-middle attacks.
* To avoid widespread online application vulnerabilities like SQL injection and cross-site scripting (XSS) attacks, input validation and output encoding are used.
* Using appropriate error handling procedures to stop confidential information from leaking in error messages.

We may protect against known security vulnerabilities and lower the risk of data breaches and cyberattacks by implementing these recommended practices for secure coding that are considered industry standards. This safeguards not just the company's financial health by avoiding expensive security breaches, but also the company's reputation and clientele.

Furthermore, adhering to accepted industry best practices for secure coding aids in the business's continued compliance with rules like the General Data Protection Regulation (GDPR) and the Payment Card Industry Data Security Standard (PCI DSS). Not only is compliance with these rules mandated by law, but it also shows clients that the business takes their privacy and security seriously. This can enhance the company's general well-being and assist draw in and keep customers.

# References

1. <https://www.arcserve.com/blog/5-common-encryption-algorithms-and-unbreakables-future>
2. <https://www.ibm.com/docs/en/ztpf/1.1.0.15?topic=concepts-cipher-algorithms-keys>
3. <https://www.cryptomathic.com/news-events/blog/differences-between-hash-functions-symmetric-asymmetric-algorithms>
4. <https://www.thesslstore.com/blog/asymmetric-vs-symmetric-encryption/>
5. <https://eds-p-ebscohost-com.ezproxy.snhu.edu/eds/pdfviewer/pdfviewer?vid=1&sid=07edf3b0-e96a-4086-b2bb-ed563575c8e2%40redis>
6. <https://www.techtarget.com/searchsecurity/definition/certificate-authority>
7. <https://en.wikipedia.org/wiki/Certificate_authority>
8. <https://www.thesslstore.com/blog/what-is-a-certificate-authority-ca-and-what-do-they-do/>
9. <https://www.sciencedirect.com/topics/computer-science/issuing-certificate>