GLOBALRAIN

CS 305 Project Two
Practices for Secure Software Report

Table of Contents

_Toc33111301

Document Revision History

Version	Date	Author	Comments
1.0	06/23/22	Fabian Rodriguez	

Client

Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols. Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

Developer
Fabian Rodriguez

1. Algorithm Cipher
Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

- Provide a brief, high-level overview of the encryption algorithm cipher.
- Discuss the hash functions and bit levels of the cipher.
- Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
- Describe the history and current state of encryption algorithms.

AES encryption algorithm cipher is the industry standard symmetric key encryption algorithm. It can be used to encrypt data and provide security services such as integrity and confidentiality. Moreover, it has a relatively simple design which makes it easy to analyze.

The cipher works by using a block cipher with a secret key and an initialization vector (IV) to encrypt plaintext in blocks of 16 bytes (128 bits). The ciphers block size is a fixed block size of 128 bits and key size of 128, 192, or 256 bits, meaning that there are 16 rounds of encryption applied to each block.

AES provides various benefits to users, such as it being faster than its predecessor DES while being considerably more secure. The security has been increased by four times since it came out in 2001 with no additional changes to the AES algorithm itself. The key idea behind AES lies in how it applies transformations to binary representations of data blocks before they are encrypted or decrypted. AES has been designed to protect information from being disclosed or modified by adversaries, even when they have considerable knowledge about the cipher employed and about its implementation.

2. Certificate Generation
Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

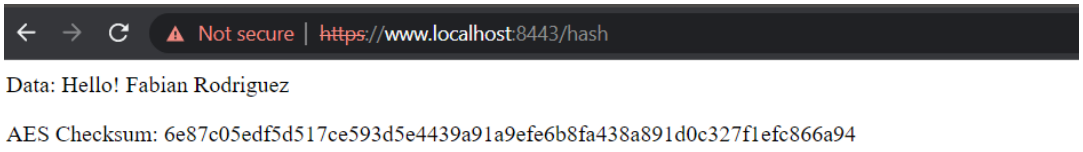
- To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

```
Your keystore contains 1 entry

Alias name: deb
Creation date: Jun 14, 2022
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Fabian Rodriguez, OU=SNHU, O=SNHU, L=Manchester, ST=NH, C=NH
Issuer: CN=Fabian Rodriguez, OU=SNHU, O=SNHU, L=Manchester, ST=NH, C=NH
Serial number: 2b257f7763e4eae2
Valid from: Tue Jun 14 19:38:52 CDT 2022 until: Mon Sep 12 19:38:52 CDT 2022
Certificate fingerprints:
         SHA1: 8F:68:F4:6E:C2:66:78:55:D0:AE:99:1E:C5:2C:67:C8:16:BC:D9:2A
         SHA256: 53:BD:FC:03:03:04:50:7B:EB:23:F4:F8:7B:C5:2D:34:73:E3:1A:0F:03:FB:47:56:09:66:1D:99:D5:96:99:A1
Signature algorithm name: SHA1withRSA (weak)
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: A1 15 B7 AA 67 E1 C8 EC   F0 34 ED 9C 1B A6 A1 3D  ....g....4.....=
0010: 1F B5 41 B5                                        ..A.
]
]
```

3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

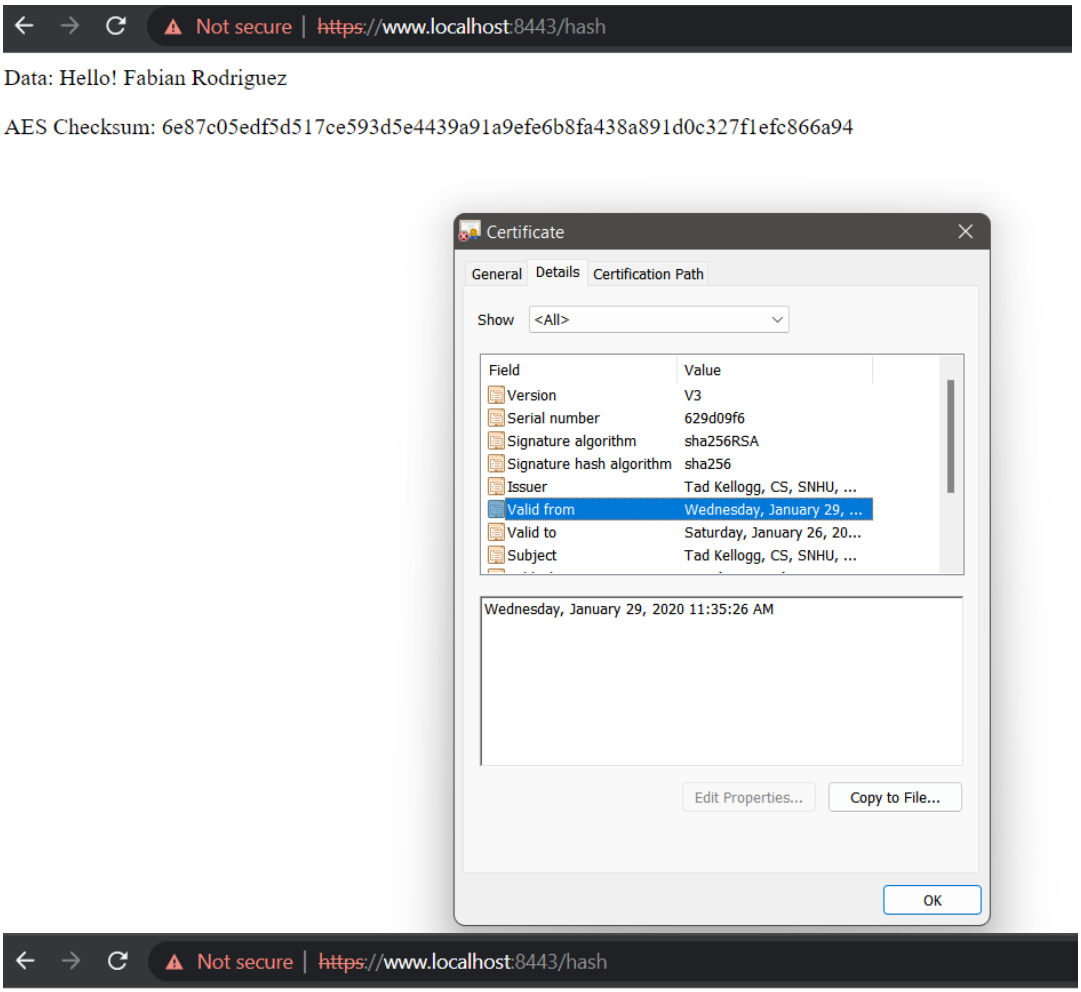
- Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

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



5. Secondary Testing

Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

- Include the following below:

```java
@RestController
class ServerController {
    public static String generateHash(String name) {
        try {
            // Create an object of MessageDigest class using the java.security.MessageDigest
            // library.
            MessageDigest md = MessageDigest.getInstance(algorithm: "SHA-256");
            // Add the data to be hashed to the MessageDigest object.
            md.update(name.getBytes(StandardCharsets.UTF_8));
            // Create a digest of the data.
            byte[] digest = md.digest();
            // Convert the byte array to a hex string.
            return String.format(format: "%064x", new BigInteger(signum: 1, digest));
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return null;
    }

    @RequestMapping("/hash")
    public String myHash() {
        String data = "Hello! Fabian Rodriguez";
        String hashName = generateHash(data);

        return "<p>Data: " + data + "</p><p>AES Checksum: " + hashName + "</p>";
    }
}
```

- A screenshot of the refactored code executed without errors

DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

How to read the report | Suppressing false positives | Getting Help: github issues

Sponsor

Project: rest-service

com.snhu:rest-service:0.0.1-SNAPSHOT

Scan Information (show all):

- *dependency-check version*: 7.1.0
- *Report Generated On*: Tue, 14 Jun 2022 19:09:53 -0500
- *Dependencies Scanned*: 38 (20 unique)
- *Vulnerable Dependencies*: 11
- *Vulnerabilities Found*: 76
- *Vulnerabilities Suppressed*: 1
- ...

Summary

Display: Showing Vulnerable Dependencies (click to show all)

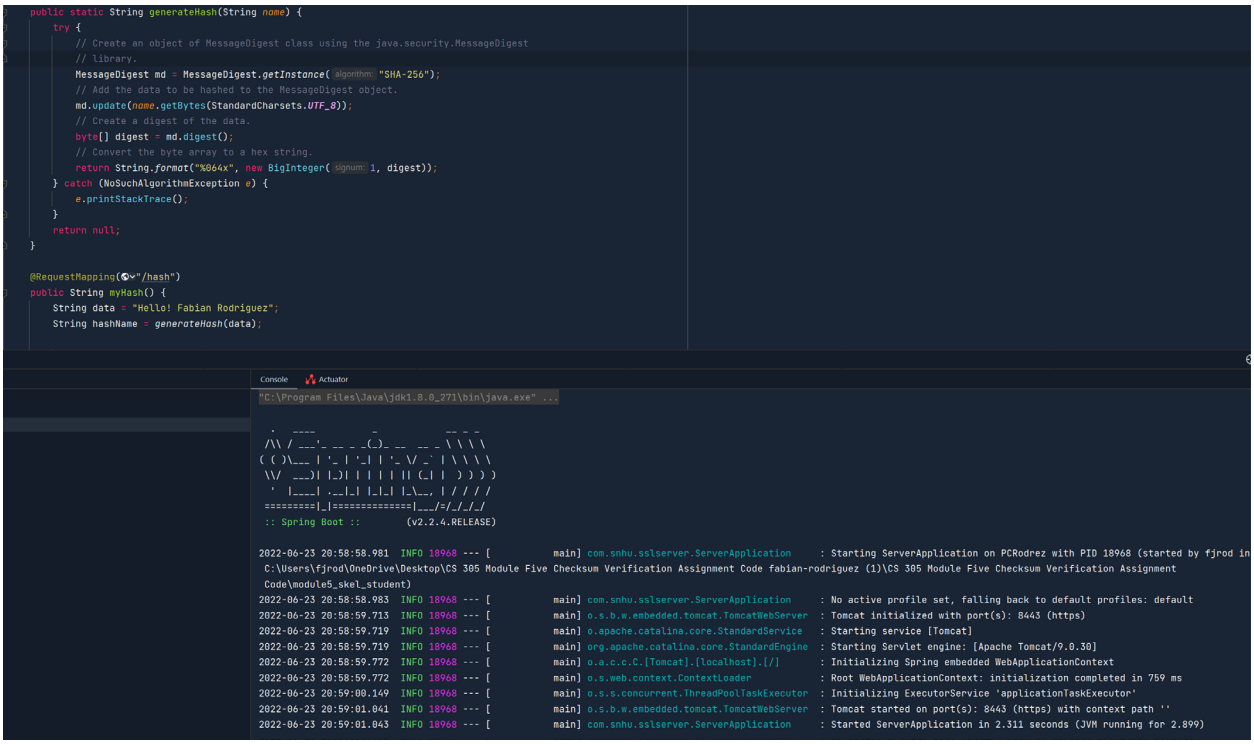
Dependency	Vulnerability IDs	Package
bcprov-jdk15on-1.46.jar	cpe:2.3:a:bouncycastle:bouncy-castle-crypto-package:1.46:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:bouncy_castle_crypto_package:1.46:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.46:*:*:*:*:*:*:* cpe:2.3:a:bouncycastle:the_bouncy_castle_crypto_package_for_java:1.46:*:*:*:*:*:*:*	pkg:maven/org.bouncycastle/bcprov-jdk15on@1.46
hibernate-validator-6.0.18.Final.jar	cpe:2.3:a:redhat:hibernate_validator:6.0.18:*:*:*:*:*:*:*	pkg:maven/org.hibernate.validator/hibernate-validator@6.0.18.Final

- A screenshot of the dependency check report

6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

- Complete this functional testing and include a screenshot below of the refactored code executed without errors.



7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

- Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
- Discuss your process for adding layers of security to the software application and the value that security adds to the company's overall wellbeing.
- Point out best practices for maintaining the current security of the software application to your customer.

The code has been refactored to addressed security by implementing cryptography. We have included a new hash function that will enable increased security thorough our API endpoints and can help validate users. REST Application could have many vulnerabilities if not secure properly. That where our hash function comes in. Because we added a CA certificate our Client/Server security was greatly increased. To avoid the code from breaking we have also implemented a simple try catch method. Finally, we have updated all dependencies and removed all false positives.

There's is a plethora of ways to add security to an application. In the case of Global Rain, we intend to force HSTS, SSL/TLS connections, encrypt the data at rest and when being sent through the client. We

have ensured all vulnerabilities are either fixed or removed. And we will AES and SHA-256 to have the best encryption in the industry.

To maintain, scale and continuously keep our app secure we will create a flow that will check for the dependency's updates, vulnerabilities, and security every single time an update is to be deployed.