GLOBALRAIN

Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	10/12/2023	Oved AYDIN	

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
Oved AYDIN

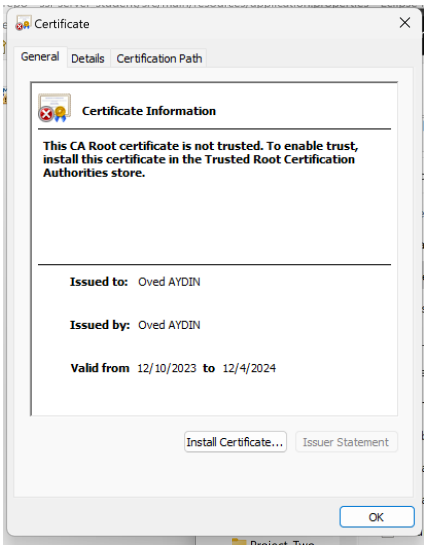
1. Algorithm Cipher

Artemis Financial wants better security for their website to protect communication. Since the main threat is from bad actors trying to get financial information, using encryption is the best idea. This makes files unreadable without the right key, stopping attackers. To make communication safer, I suggest using Asymmetric encryption. This means using a public key for encryption and a private key for decryption. For extra security, especially when sending information outside, I recommend using the SHA-256 cipher algorithm with 256-bit keys for encryption. It's a strong method with lots of possible key combinations because it's 256 bits long. SHA-256 uses Java's random number generator, making encryption more secure. It creates a non-reversible checksum to check if the file is valid. Using the SHA-256 cipher, the hash function creates a checksum for the message, making Artemis Financial's website more secure.

2. Certificate Generation

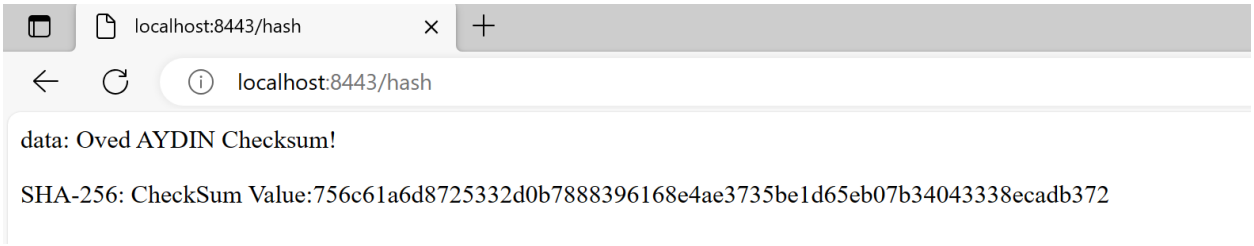
Insert a screenshot below of the CER file.

```
C:\Users\ovedaydin\modelio\Modelio Open Source 5.3\jre\bin>keytool -printcert -file server.cer
Owner: CN=Oved AYDIN, OU=SNHU, O=SNHU, L=Istanbul, ST=Maltepe, C=TR
Issuer: CN=Oved AYDIN, OU=SNHU, O=SNHU, L=Istanbul, ST=Maltepe, C=TR
Serial number: 59623f46
Valid from: Sun Dec 10 14:39:26 TRT 2023 until: Wed Dec 04 14:39:26 TRT 2024
Certificate fingerprints:
         SHA1: 5C:17:B8:97:07:44:AE:99:BC:DC:4A:CC:C0:12:C6:E0:5A:AC:2B:C2
         SHA256: 24:D2:95:AE:EE:C7:86:03:3E:EA:80:9A:35:D6:1B:EA:7E:13:D2:9C:87:41:5F:BB:B7:C3:A5:49:97:B4:7E:58
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: 18 46 C3 91 3F 4A 89 45   8D 86 CC 53 9F 6C A7 8B  .F..?J.E...S.l..
0010: 58 19 6E A1                                        X.n.
]
]
```



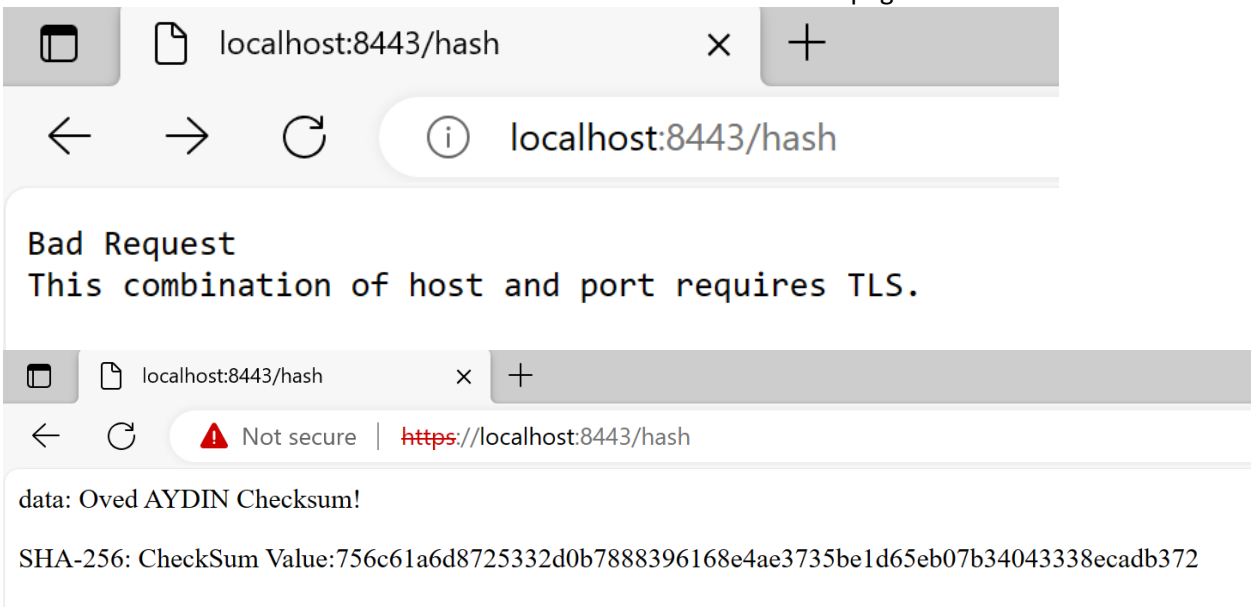
3. Deploy Cipher

Insert a screenshot below of the checksum verification.



4. Secure Communications

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



5. Secondary Testing

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

```java
package com.snhu.sslserver;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class HashingUtil {
    // Function to generate hash value for a given data string
    public static String generateHash(String data) {
        try {
            // Create MessageDigest object with SHA-256 algorithm
            MessageDigest digest = MessageDigest.getInstance("SHA-256");

            // Update the digest with the bytes of the data string
            byte[] hashBytes = digest.digest(data.getBytes());

            // Convert the byte array to a hexadecimal string
            return bytesToHex(hashBytes);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return "Error generating hash";
        }
    }

    // Function to convert a byte array to a hexadecimal string
    private static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
}
```

```java
package com.snhu.sslserver;

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 SslServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(SslServerApplication.class, args);
    }

}

@RestController
class ServerController {
    @RequestMapping("/hash")
    public String myHash() {
        String data = "Oved AYDIN Checksum!";
        String hashValue = HashingUtil.generateHash(data);
        return "<p>data: " + data + "</p>" +
                "<p>SHA-256: CheckSum Value:" + hashValue + "</p>";
    }
}
```

Project: ssl-server

com.snhu:ssl-server:0.0.1-SNAPSHOT

Scan Information (show all):

- *dependency-check version*: 5.3.0
- *Report Generated On*: Sun, 10 Dec 2023 14:52:44 +0300
- *Dependencies Scanned*: 46 (29 unique)
- *Vulnerable Dependencies*: 12
- *Vulnerabilities Found*: 26
- *Vulnerabilities Suppressed*: 0
- ...

Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
spring-boot-starter-data-rest-3.0.0.jar	cpe:2.3:a:vmware:spring_boot:3.0.0:*:*:*:*:*:*:* cpe:2.3:a:vmware:spring_data_rest:3.0.0:*:*:*:*:*:*:*	pkg:maven/org.springframework.boot/spring-boot-starter-data-rest@3.0.0	CRITICAL	3	Highest	28
spring-hateoas-2.0.0.jar	cpe:2.3:a:vmware:spring_hateoas:2.0.0:*:*:*:*:*:*:*	pkg:maven/org.springframework.hateoas/spring-hateoas@2.0.0	MEDIUM	1	Highest	30
jackson-databind-2.14.1.jar	cpe:2.3:a:fasterxml:jackson-databind:2.14.1:*:*:*:*:*:*:*	pkg:maven/com.fasterxml.jackson.core/jackson-databind@2.14.1	MEDIUM	1	Highest	41
spring-boot-3.0.0.jar	cpe:2.3:a:vmware:spring_boot:3.0.0:*:*:*:*:*:*:*	pkg:maven/org.springframework.boot/spring-boot@3.0.0	CRITICAL	3	Highest	30
logback-core-1.4.5.jar	cpe:2.3:a:qos:logback:1.4.5:*:*:*:*:*:*:*	pkg:maven/ch.qos.logback/logback-core@1.4.5	HIGH	1	Highest	37
snakeyaml-1.33.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.33:*:*:*:*:*:*:* cpe:2.3:a:yaml_project:yaml:1.33:*:*:*:*:*:*:*	pkg:maven/org.yaml/snakeyaml@1.33	CRITICAL	3	Highest	30
tomcat-embed-core-10.1.1.jar	cpe:2.3:a:apache:tomcat:10.1.1:*:*:*:*:*:*:* cpe:2.3:a:apache_tomcat:apache_tomcat:10.1.1:*:*:*:*:*:*:*	pkg:maven/org.apache.tomcat.embed/tomcat-embed-core@10.1.1	HIGH	7	Highest	69
spring-web-6.0.2.jar	cpe:2.3:a:pivotal_software:spring_framework:6.0.2:*:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:6.0.2:*:*:*:*:*:*:*	pkg:maven/org.springframework/spring-web@6.0.2	HIGH	1	Highest	29
spring-webmvc-6.0.2.jar	cpe:2.3:a:pivotal_software:spring_framework:6.0.2:*:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:6.0.2:*:*:*:*:*:*:*	pkg:maven/org.springframework/spring-webmvc@6.0.2	HIGH	1	Highest	31
spring-expression-6.0.2.jar	cpe:2.3:a:pivotal_software:spring_framework:6.0.2:*:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:6.0.2:*:*:*:*:*:*:*	pkg:maven/org.springframework/spring-expression@6.0.2	MEDIUM	2	Highest	31
json-path-2.7.0.jar	cpe:2.3:a:json-java_project:json-java:2.7.0:*:*:*:*:*:*:*	pkg:maven/com.jayway.jsonpath/json-path@2.7.0	HIGH	2	Low	30
json-smart-2.4.8.jar	cpe:2.3:a:json-smart_project:json-smart:2.4.8:*:*:*:*:*:*:*	pkg:maven/net.minidev/json-smart@2.4.8	HIGH	1	Highest	32

6. Functional Testing

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

```
1  package com.snhu.sslserver;
2
3  import org.springframework.boot.SpringApplication;
7
8  @SpringBootApplication
9  public class SslServerApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(SslServerApplication.class, args);
13     }
14
15 }
16
17 @RestController
18 class ServerController {
19     @RequestMapping("/hash")
20     public String myHash() {
21         String data = "Oved AYDIN Checksum!";
22         String hashValue = HashingUtil.generateHash(data);
23         return "<p>data: " + data + "</p>" +
24                 "<p>SHA-256: CheckSum Value:" + hashValue + "</p>";
25     }
26 }
```

```java
package com.snhu.sslserver;

import java.security.MessageDigest;

public class HashingUtil {
    // Function to generate hash value for a given data string
    public static String generateHash(String data) {
        try {
            // Create MessageDigest object with SHA-256 algorithm
            MessageDigest digest = MessageDigest.getInstance("SHA-256");

            // Update the digest with the bytes of the data string
            byte[] hashBytes = digest.digest(data.getBytes());

            // Convert the byte array to a hexadecimal string
            return bytesToHex(hashBytes);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return "Error generating hash";
        }
    }

    // Function to convert a byte array to a hexadecimal string
    private static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
}
```

7. Summary

I revamped my code by adding a secure RestController for handling hashing operations in my programs. The ServerController class now aligns with the identified vulnerabilities in the assessment diagram. To boost security, I've chosen the SHA-256 hashing cipher for its strong security and low collision risk. To keep the application secure, I recommend running dependency checks once or twice a month. This helps stay updated on potential vulnerabilities, safeguarding the company and its sensitive data. Also, it's essential to keep the plugins listed in the pom.xml file to ensure the latest plugin versions are in use. This practice maintains a high level of security by regularly updating dependencies and fortifying the application against emerging threats.

8. Industry Standard Best Practices

In making the application secure, I picked the SHA-256 algorithm for hashing, a trusted method in the industry. This algorithm keeps our sensitive information safe during cryptographic operations. I also

used the MessageDigest class in Java, a tool recommended for secure tasks, to ensure a consistent approach to hashing. For data safety, our code handles input carefully. The generateHash method, which creates hash values, only takes a string as input and avoids direct user input. This makes it harder for potential attacks. I've also added a safety net in the code, so if the SHA-256 method isn't available, the system handles it well without revealing sensitive information. To keep our data intact, I used a method that ensures our original data doesn't change during hashing. Additionally, I organized the code neatly, keeping the parts that handle hashing separate. This not only makes it easier to understand but also follows the best practices used in the industry, ensuring our application stays clear and secure in the long run.

By following standard best practices, we make our systems more secure. Using common algorithms and libraries helps prevent potential issues, making it harder for cyber threats to harm our systems. This also ensures that our software meets specific rules about data security in different industries and areas, preventing possible legal and financial problems. These secure coding practices build trust with users. When our applications are well-designed for security, they protect our company's reputation and actively prevent damage from security breaches. Following industry standards not only strengthens our immediate security measures but also improves the long-term sustainability of our software. Code developed this way is easier to understand and maintain, reducing the risk of introducing problems when updating. This comprehensive approach not only protects against immediate threats but also creates a positive company image, boosting confidence among stakeholders.