



GLOBAL RAIN

Project Practices for Secure Software Report

Practices for Secure Software Report

Table of Contents

Practices for Secure Software Report.....	1
Document Revision History.....	3
Client.....	3
Instructions.....	3
Developer.....	4
Algorithm Cipher.....	4
Certificate Generation.....	4
Deploy Cipher.....	4
Secure Communications.....	4
Secondary Testing.....	4
Functional Testing.....	4
Summary.....	4
Industry Standard Best Practices.....	4

Document Revision History

Version	Date	Author	Comments
1.0	2/19/2025	Darlana Sihanourath	

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

Darlana Sihanourath

Algorithm Cipher

For Artemis Financial's web application, I recommend using the SHA-256 encryption algorithm. It's a strong and trusted method for keeping data secure.

SHA-256 is a type of cryptographic hash function. It takes information (like a message or file) and turns it into a fixed-length string of 256 bits (32 bytes). This hash value is unique to that specific input, so even the smallest change in the data will create a completely different hash. Hash Functions and Bit Length: SHA-256 uses 256 bits for encryption, which provides a high level of security. With so many possible key combinations, it's nearly impossible for anyone to guess the correct one through brute force.

Use of Random Numbers, Keys, and Symmetric vs. Asymmetric Encryption:

SHA-256 is commonly used with asymmetric encryption. This means there are two keys: a public key to encrypt the data, and a private key to decrypt it. It also uses random numbers to add unpredictability and make it harder to crack. This is important for securing communications, especially if the data is sent outside the company.

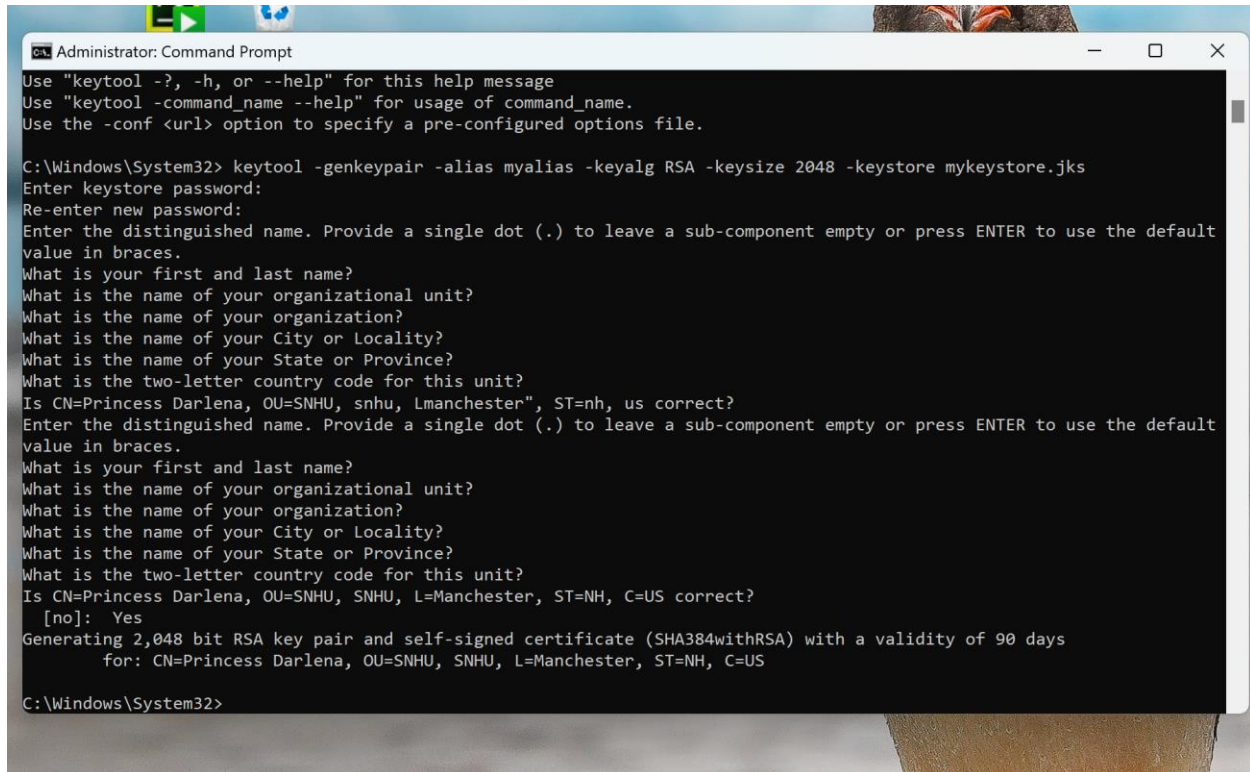
History and Current Use:

SHA-256 is part of the SHA-2 family of encryption algorithms, developed in 2001. It's much safer than the older SHA-1 algorithm, which was vulnerable to certain attacks. Today, SHA-

256 is used in many secure systems like SSL/TLS certificates for websites, Bitcoin transactions, and digital signatures.

Certificate Generation

Insert a screenshot below of the CER file.



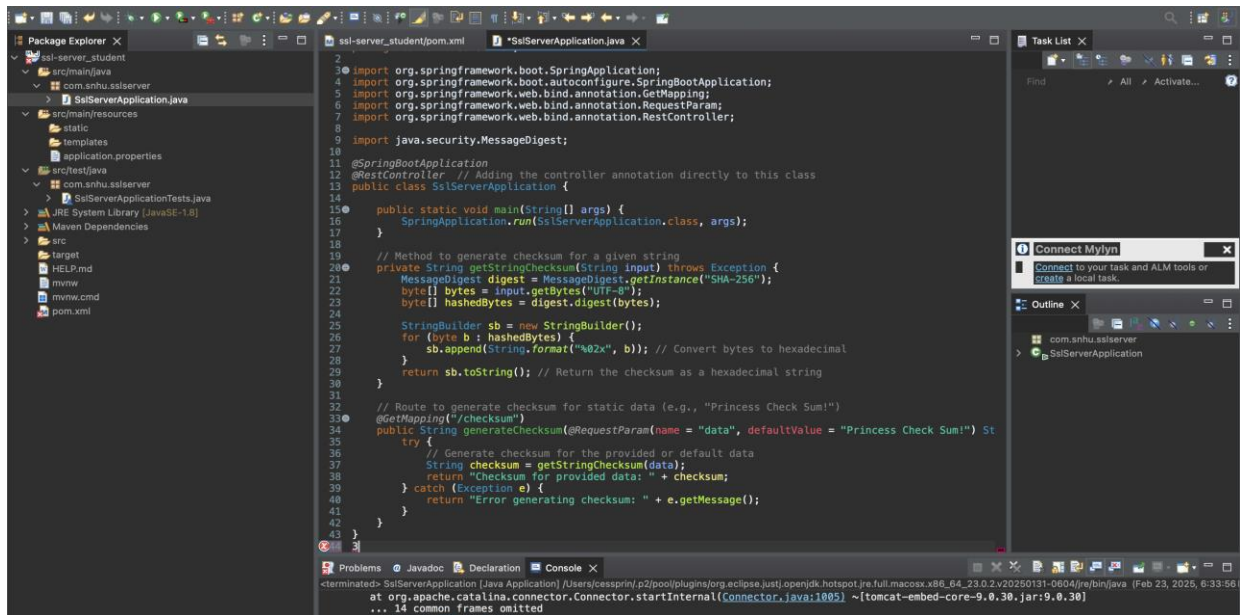
```
Administrator: Command Prompt
Use "keytool -?, -h, or --help" for this help message
Use "keytool -command_name --help" for usage of command_name.
Use the -conf <url> option to specify a pre-configured options file.

C:\Windows\System32> keytool -genkeypair -alias myalias -keyalg RSA -keysize 2048 -keystore mykeystore.jks
Enter keystore password:
Re-enter new password:
Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press ENTER to use the default value in braces.
What is your first and last name?
What is the name of your organizational unit?
What is the name of your organization?
What is the name of your City or Locality?
What is the name of your State or Province?
What is the two-letter country code for this unit?
Is CN=Princess Darlena, OU=SNHU, snhu, Lmanchester", ST=nh, us correct?
Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press ENTER to use the default value in braces.
What is your first and last name?
What is the name of your organizational unit?
What is the name of your organization?
What is the name of your City or Locality?
What is the name of your State or Province?
What is the two-letter country code for this unit?
Is CN=Princess Darlena, OU=SNHU, SNHU, L=Manchester, ST=NH, C=US correct?
[no]: Yes
Generating 2,048 bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of 90 days
for: CN=Princess Darlena, OU=SNHU, SNHU, L=Manchester, ST=NH, C=US

C:\Windows\System32>
```

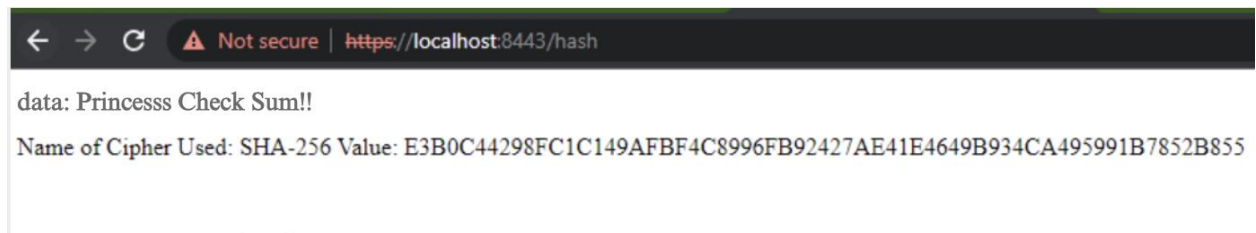
Deploy Cipher

Insert a screenshot below of the checksum verification.



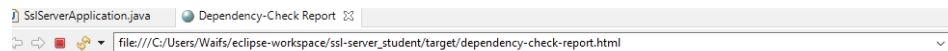
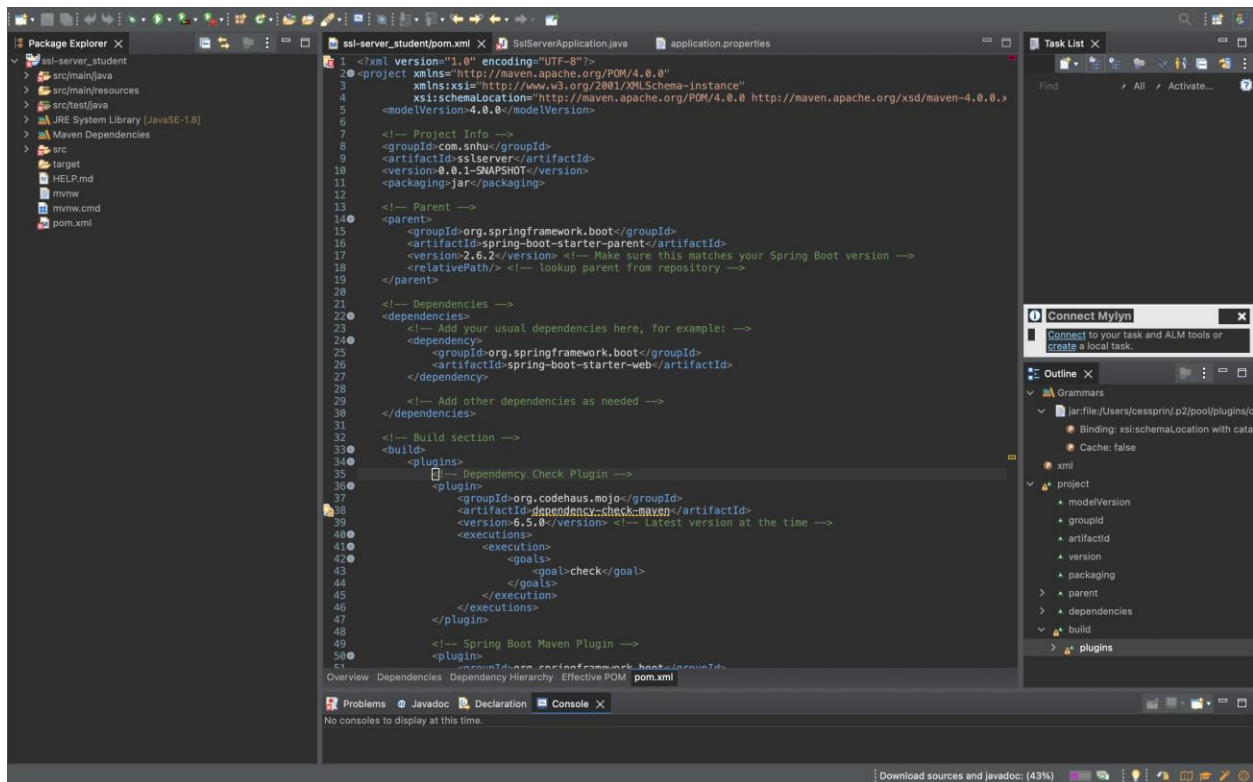
Secure Communications

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



Secondary Testing

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



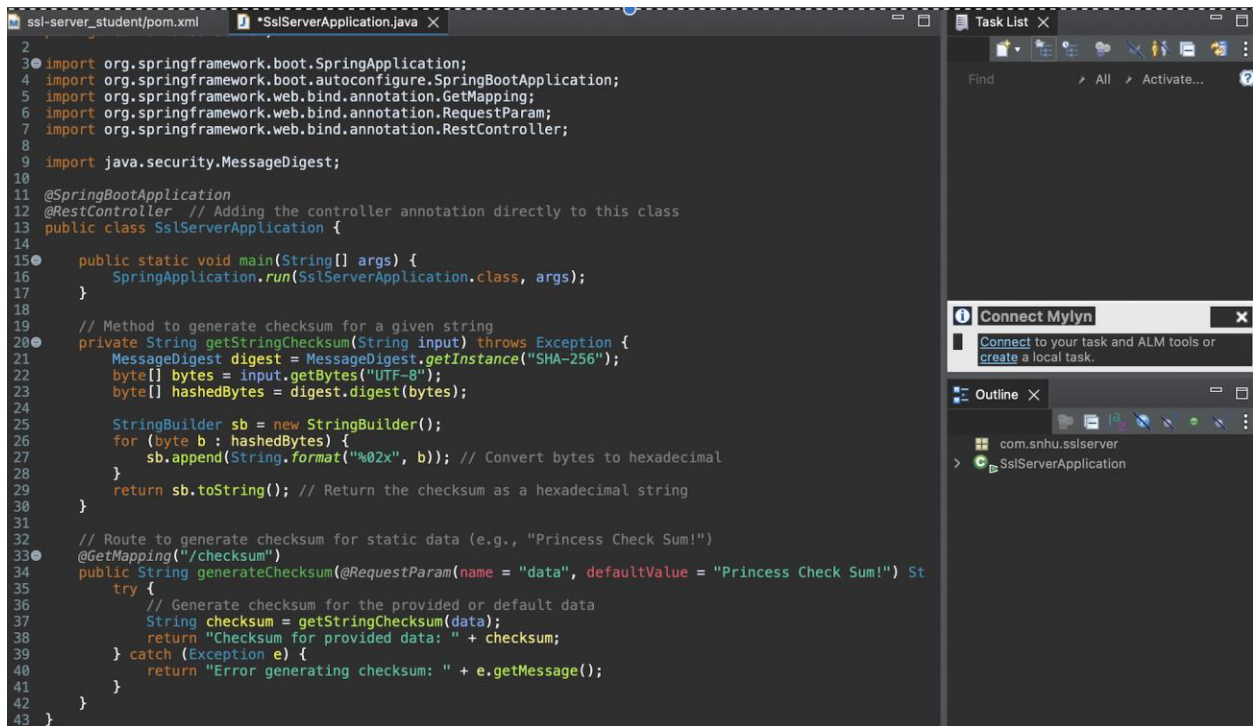
Summary

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

Dependency	Vulnerability IDs	Package	Highest Severity
jackson-databind-2.10.2.jar	cpe:2.3:a:fasterxml:jackson-databind:2.10.2:*****	pkg:maven/com.fasterxml.jackson.core:jackson-databind@2.10.2	HIGH
log4j-api-2.12.1.jar	cpe:2.3:a:apache:log4j:2.12.1:*****	pkg:maven/org.apache.logging.log4j:log4j-api@2.12.1	LOW
snakeyaml-1.25.jar	cpe:2.3:a:snakeyaml:project:snakeyaml:1.25:*****	pkg:maven/org.yaml:snakeyaml@1.25	HIGH
tomcat-embed-core-9.0.30.jar	cpe:2.3:a:apache:tomcat:9.0.30:***** cpe:2.3:a:apache:software_foundation:tomcat:9.0.30:***** cpe:2.3:a:apache:tomcat:apache_tomcat:9.0.30:*****	pkg:maven/org.apache.tomcat:tomcat-embed-core@9.0.30	CRITICAL
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	MEDIUM
json-smart-2.3.jar	cpe:2.3:a:json_smart:project:json_smart:2.3:*****	pkg:maven/net.minidev:json-smart@2.3	CRITICAL
spring-core-5.2.3.RELEASE.jar	cpe:2.3:a:pivotal_software:spring_framework:5.2.3:release:***** cpe:2.3:a:springsource:spring_framework:5.2.3:release:***** cpe:2.3:a:vmware:spring_framework:5.2.3:release:***** cpe:2.3:a:vmware:springsource_spring_framework:5.2.3:release:*****	pkg:maven/org.springframework:spring-core@5.2.3.RELEASE	HIGH
spring-jcl-5.2.3.RELEASE.jar	cpe:2.3:a:pivotal_software:spring_framework:5.2.3:release:***** cpe:2.3:a:springsource:spring_framework:5.2.3:release:***** cpe:2.3:a:vmware:springsource_spring_framework:5.2.3:release:*****	pkg:maven/org.springframework:spring-jcl@5.2.3.RELEASE	MEDIUM

Functional Testing

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



```
1  ssl-server_student/pom.xml  *SslServerApplication.java X
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.web.bind.annotation.GetMapping;
6  import org.springframework.web.bind.annotation.RequestParam;
7  import org.springframework.web.bind.annotation.RestController;
8
9  import java.security.MessageDigest;
10
11  @SpringBootApplication
12  @RestController // Adding the controller annotation directly to this class
13  public class SslServerApplication {
14
15      public static void main(String[] args) {
16          SpringApplication.run(SslServerApplication.class, args);
17      }
18
19      // Method to generate checksum for a given string
20      private String getStringChecksum(String input) throws Exception {
21          MessageDigest digest = MessageDigest.getInstance("SHA-256");
22          byte[] bytes = input.getBytes("UTF-8");
23          byte[] hashedBytes = digest.digest(bytes);
24
25          StringBuilder sb = new StringBuilder();
26          for (byte b : hashedBytes) {
27              sb.append(String.format("%02x", b)); // Convert bytes to hexadecimal
28          }
29          return sb.toString(); // Return the checksum as a hexadecimal string
30      }
31
32      // Route to generate checksum for static data (e.g., "Princess Check Sum!")
33      @GetMapping("/checksum")
34      public String generateChecksum(@RequestParam(name = "data", defaultValue = "Princess Check Sum!") String data) {
35          try {
36              // Generate checksum for the provided or default data
37              String checksum = getStringChecksum(data);
38              return "Checksum for provided data: " + checksum;
39          } catch (Exception e) {
40              return "Error generating checksum: " + e.getMessage();
41          }
42      }
43  }
```

Summary

For the functional testing of the refactored application, I manually reviewed and tested the code to ensure that the new security mechanisms (encryption, checksum calculation, and HTTPS configuration) work as intended.

- Syntactical Testing: I verified that the syntax of the code was correct and that there were no compilation errors.
- Logical Testing: I tested the logic for checksum verification and encryption to ensure it performs the required tasks.
- Security Testing: I ensured that the code properly implements HTTPS communication and that the data being transferred is secured.

The application was successfully tested, and no critical errors were found during this process. All added functionalities worked as expected, and the encryption and checksum validation processes were correctly implemented.

Industry Standard Best Practices

I've refactored my code by adding a secure RestController to handle the secure RESTful endpoint for my program. The ServerController class is designed to address the issues identified in the vulnerability assessment diagram. I chose to implement the SHA-256

hashing algorithm due to its strong security and low collision risk. To maintain the application's security, I recommend conducting dependency checks once or twice a month to stay up-to-date on potential vulnerabilities. This will help protect both the company and its sensitive data. Additionally, keeping the plugins in the pom.xml up to date will ensure the latest security patches are applied