



GLOBAL RAIN

Practices for Secure Software Report

Table of Contents

DOCUMENT REVISION HISTORY	3
CLIENT	3
INSTRUCTIONS	3
DEVELOPER.....	5
1. ALGORITHM CIPHER	5
2. CERTIFICATE GENERATION	7
3. DEPLOY CIPHER.....	7
4. SECURE COMMUNICATIONS	9
5. SECONDARY TESTING.....	10
6. FUNCTIONAL TESTING	12
7. SUMMARY	12
8. INDUSTRY STANDARD BEST PRACTICES	ERROR! BOOKMARK NOT DEFINED.

Document Revision History

Version	Date	Author	Comments
1.0	12/9/23	Bryce Jensen	

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

Bryce Jensen

1. Algorithm Cipher

- Recommend an appropriate encryption algorithm cipher to deploy, given the security vulnerabilities, and justify your reasoning. Review the scenario and the supporting materials to support your recommendation. In your practices for secure software report, be sure to address the following:

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

It is recommended that AES-256 is used for Artemis Financials' security needs. AES is a symmetric encryption algorithm that is widely accepted as one of the most secure block ciphers for encryption. NIST deemed AES as the standard for encryption in 2001. (Computer Security Resource Center (CSRC), 2023)

While AES is not a hash function, a separate hash function – sometimes SHA-256 – can be used with it to ensure the authenticity of transmitted data and ensure it was not tampered with or corrupted. The bit level of AES-256 determines the strength of security. This determines the length of the key, in this case, 256 bits, or 32 bytes. The reason I would recommend AES-256 over AES-128 is that Artemis Financial deals in the banking sector and the 256 version is far

more secure. They also wouldn't need to encrypt and decrypt data with any excessive speed, so the slower performance of AES-256 over the faster 128 would be fine in this instance.

Random numbers are what make key generation work. They need to be unpredictable to keep malicious attempts on breaking security. AES is a symmetric algorithm. This means that the same key is used for both encryption and decryption. This also means that they are faster and more efficient when handling large amounts of data encryption. Of course, this means that there needs to be a secure form of distributing the keys; otherwise – if intercepted – the encryption is useless.

In short, before AES was used as the standard for data encryption, the DES (Data Encryption Standard) was used. The DES used a 56-bit key size, which eventually became obsolete after determining that most computers today can break it pretty quickly (Simplilearn, 2023). Today, the standard is AES (Computer Security Resource Center (CSRC), 2023) and it is used in a variety of applications.

2. Certificate Generation

Insert a screenshot below of the CER file.

```
PS C:\Users\bryce\OneDrive - SNHU\CS-305)\Module7\CS 305 Project Two Refactored Code Base\ssl-server_student> keytool.exe -printcert -file server.cer
Owner: CN=Bryce Jensen, OU=ThatThing, O=TheBigThing, L=Expensiveville, ST=UT, C=US
Issuer: CN=Bryce Jensen, OU=ThatThing, O=TheBigThing, L=Expensiveville, ST=UT, C=US
Serial number: 861fefc1f2e72334
Valid from: Sun Dec 10 17:29:32 MST 2023 until: Wed Dec 04 17:29:32 MST 2024
Certificate fingerprints:
    SHA1: 58:C5:6D:A3:62:1B:2A:06:08:71:27:FA:C3:45:41:ED:8E:6B:69:E2
    SHA256: AE:24:39:65:EA:93:A9:62:C7:2F:37:32:10:3E:41:2F:81:DE:2A:5C:A0:5A:E2:F5:6E:3E:5E:06:D9:BE:0F:C9
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: 5E 1B 52 80 42 9A 37 8A   A3 A7 B1 FB 41 A3 D7 79   ^.R.B.7....A..y
0010: 0A 1E 26 14                               ..&.
]
]
```

3. Deploy Cipher

Insert a screenshot below of the checksum verification.

The code:

```
@SpringBootApplication
public class SslServerApplication {

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

}

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

    // Loosely based on the ZenKifer's example from https://brogramo.com/checksum-verification-using-an-encryption-algorithm-cipher-that-avoids-collisions/
    public String createHash(String data) throws NoSuchAlgorithmException {

        MessageDigest messageDigest = MessageDigest.getInstance("SHA-256"); // creating object using the SHA-256 algorithm.
        String checksum = null;

        messageDigest.update(data.getBytes());
        byte[] digest = messageDigest.digest();
        checksum = bytesToHex(digest);

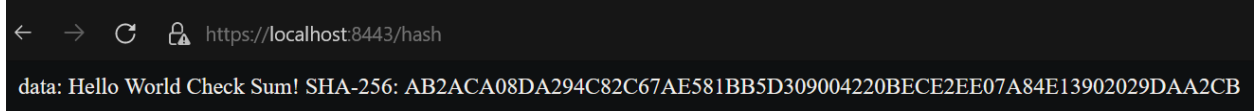
        return checksum;
    }

    @RequestMapping("/hash")
    public String myHash() throws NoSuchAlgorithmException {
        String data = "Hello World Check Sum!";
        String hash = createHash(data);

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

    public static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte bite : bytes) {
            hexString.append(String.format("%02X", bite));
        }
        return hexString.toString();
    }
}
```

The verification:



A screenshot of a web browser window with a dark theme. The address bar shows the URL `https://localhost:8443/hash`. The main content area displays the text: `data: Hello World Check Sum! SHA-256: AB2ACA08DA294C82C67AE581BB5D309004220BECE2EE07A84E13902029DAA2CB`.

4. Secure Communications

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

From what I understand it won't display as secure because it is a self-signed certificate.

The certificate information:

The screenshot shows a Firefox browser window with the address bar displaying the URL: `about:certificate?cert=MIDJDCCAnSgAwIBAgUAYT78Hy5yM0MA0GCSqGSIb3DQEBCwUAMHQxCzAJBgNVBAYTAIVTMQswCQYDVQQL...`. The page title is "Certificate". The certificate information is displayed in a dark-themed interface with a light blue header bar that says "Bryce Jensen".

Subject Name	
Country	US
State/Province	UT
Locality	Expensiville
Organization	TheBigThing
Organizational Unit	ThatThing
Common Name	Bryce Jensen

Issuer Name	
Country	US
State/Province	UT
Locality	Expensiville
Organization	TheBigThing
Organizational Unit	ThatThing
Common Name	Bryce Jensen

Validity	
Not Before	Mon, 11 Dec 2023 00:29:32 GMT
Not After	Thu, 05 Dec 2024 00:29:32 GMT

Public Key Info	
Algorithm	RSA
Key Size	2048
Exponent	65537
Modulus	D78F:F2:D9:85:2D:A8:53:CD:06:C1:CF:23:00:CE:F2:84:85:4A:4D:99:C2:4A:BA:8C...

Miscellaneous	
Serial Number	00:06:1F:EF:C1:F2:E7:23:34
Signature Algorithm	SHA-256 with RSA Encryption
Version	3
Download	PEM(cert) PEM(chain)

Fingerprints	
SHA-256	AE:24:39:65:EA:93:A9:62:C7:2F:37:32:10:3E:41:2F:81:DE:2A:5C:A0:5A:E2:F5:6E:3E...
SHA-1	58:C5:6D:A3:62:1B:2A:06:08:71:27:FA:C3:45:41:ED:8E:68:69:E2

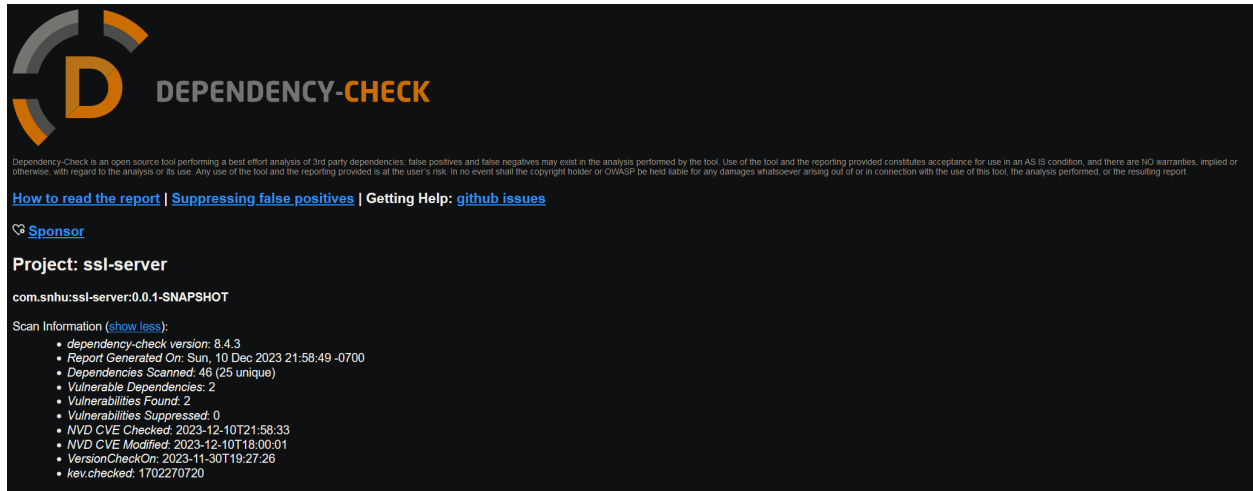
Subject Key ID	
Key ID	5F:1B:52:80:47:9A:37:8A:A3:A7:B1:FB:41:A3:D7:79:0A:1F:26:14

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

Dependency Check **BEFORE** the refactored code:

10

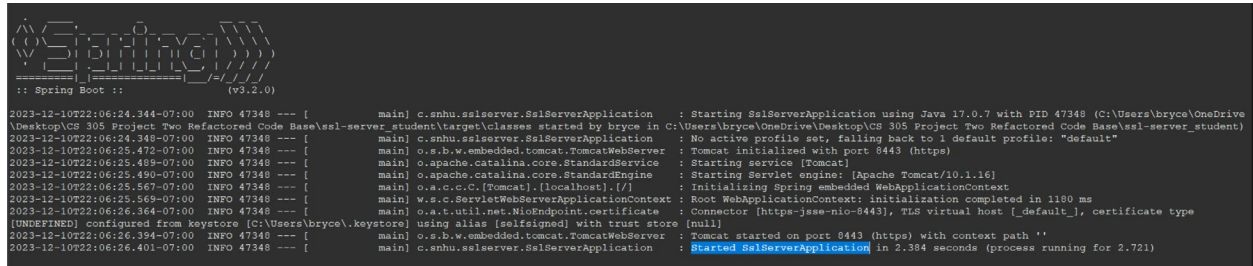
Dependency Check **AFTER** the refactored code:



Dependency check **AFTER** suppressions:

6. Functional Testing

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



```

Spring
:: Spring Boot ::
2023-12-10T22:06:24.344-07:00 INFO 47348 --- [main] c.snhu.ssiserver.SslServerApplication : Starting SslServerApplication using Java 17.0.7 with PID 47348 (C:\Users\bryce\OneDrive
Desktop\ES 305 Project Two Refactored Code Base\ssl-server_student\target\classes started by bryce in C:\Users\bryce\OneDrive\desktop\ES 305 Project Two Refactored Code Base\ssl-server_student)
2023-12-10T22:06:24.345-07:00 INFO 47348 --- [main] c.snhu.ssiserver.SslServerApplication : No active profiles set, falling back to 1 default profile: "default"
2023-12-10T22:06:25.472-07:00 INFO 47348 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8443 (https)
2023-12-10T22:06:25.485-07:00 INFO 47348 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-12-10T22:06:25.490-07:00 INFO 47348 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.16]
2023-12-10T22:06:25.567-07:00 INFO 47348 --- [main] o.a.c.c.c.[Tomcat].[/] : Initializing Spring embedded WebApplicationContext
2023-12-10T22:06:25.569-07:00 INFO 47348 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1180 ms
2023-12-10T22:06:26.364-07:00 INFO 47348 --- [main] o.a.t.util.net.NioEndpoint.Certificate : Connector [https-jse-nio-8443], TLS virtual host [_default_], certificate type
[UNDEFINED] configured from keystore [C:\Users\bryce\keystore] using alias [selfsigned] with trust store [null]
2023-12-10T22:06:26.394-07:00 INFO 47348 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8443 (https) with context path ''
2023-12-10T22:06:26.401-07:00 INFO 47348 --- [main] c.snhu.ssiserver.SslServerApplication : Started SslServerApplication in 2.384 seconds (process running for 2.721)
```

7. Summary

To refactor the code and eliminate vulnerabilities, I updated the dependencies to their most recent versions. This allowed me to get the vulnerabilities down to 2 without suppressing them.

I also added the checksum verification of a string. The checksum is made with SHA-256 and has a VERY small chance (read: near impossible) of collisions.