Practices for Secure Software Report

Dante Trisciuzzi

2.15.2023

CS305 - SNHU

Dr. Vivian Lyon

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	2.15.2023	Dante Trisciuzzi	

Client

Developer: Dante Trisciuzzi

1. Algorithm Cipher

Artemis Financial has requested additional security for their web application to ensure security when communicating data between components, as well as with external entities. To avoid malicious parties from attempting to gain financial information, encryption is required. This ensures data is transmitted in an unreadable state (*Iron-Clad Java: Building Secure Web Applications*, 2014). If this data is intercepted by a third party, it is practically useless without a decryption key. For secure communication, asymmetric encryption with a public key for encryption and a private key for decryption is the best choice. I recommend using the SHA-256 algorithm cipher to achieve the appropriate level of security. The SHA-256 cipher generates a

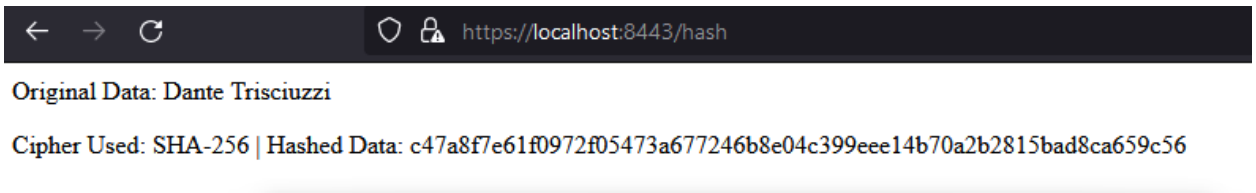
checksum of data that is irreversible, and can be seeded with a pseudo-random number generator, like the one provided natively in Java (*Java Cryptography Architecture (JCA) Reference Guide*, 2022).

2. Certificate Generation

```
C:\Users\dt
λ keytool.exe -printcert -file server.cer
Owner: CN=Dante Trisciuzzi, OU=CS305, O=SNHU, L=Essex Jct, ST=Vermont, C=US
Issuer: CN=Dante Trisciuzzi, OU=CS305, O=SNHU, L=Essex Jct, ST=Vermont, C=US
Serial number: 99ad49a
Valid from: Wed Feb 15 08:57:58 EST 2023 until: Sat Feb 10 08:57:58 EST 2024
Certificate fingerprints:
         SHA1: 85:0A:27:61:34:3A:BC:3F:DF:F1:29:F1:37:8E:15:15:19:EB:9E:57
         SHA256: 2F:C8:5F:41:AB:44:8D:48:27:07:14:D9:B1:45:0B:C3:DE:3E:3E:52:CD:EE:6A:DC:D6:21:4D:37:A3:6F:EC:C0
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: 84 37 E7 47 A6 E3 53 4F   7F A7 3F 44 83 17 8F 34  .7.G..SO..?D...4
0010: 0A DF C4 43                                        ...C
]
]


C:\Users\dt
λ
```

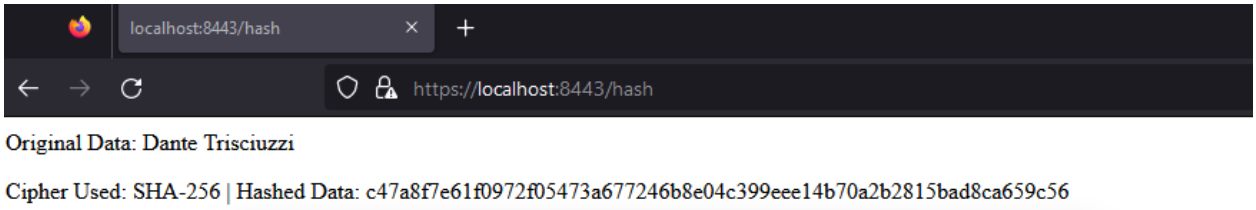
3. Deploy Cipher

Insert a screenshot below of the checksum verification.



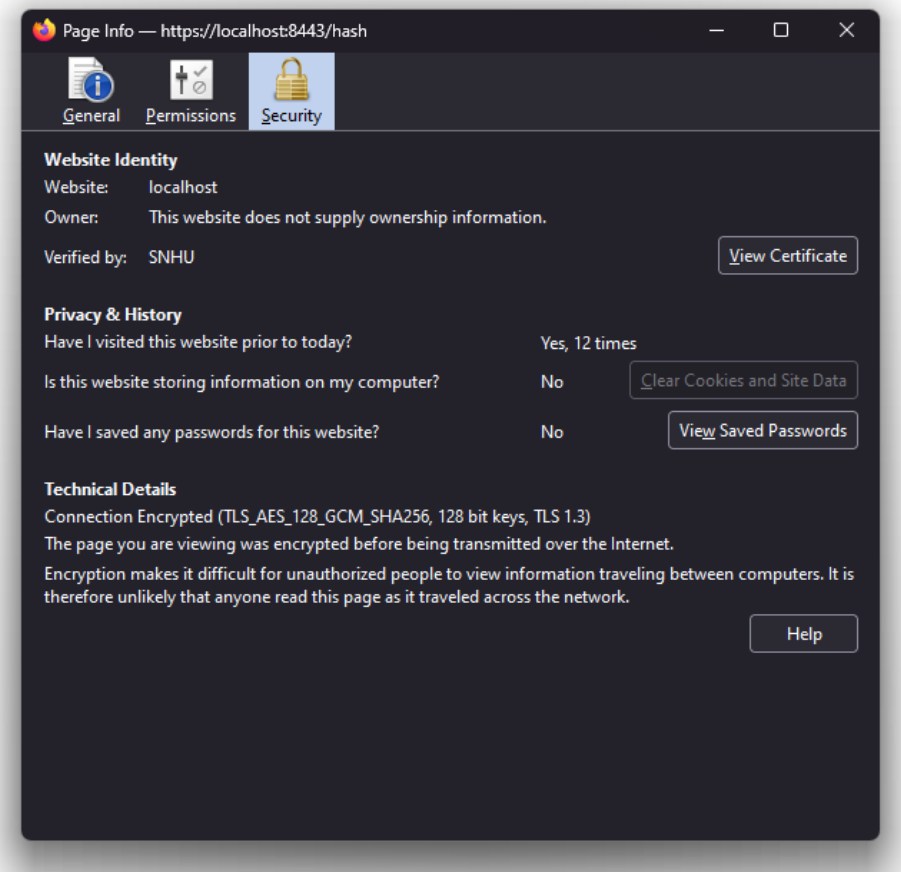
4. Secure Communications

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



Original Data: Dante Trisciuzzi

Cipher Used: SHA-256 | Hashed Data: c47a8f7e61f0972f05473a677246b8e04c399eee14b70a2b2815bad8ca659c56



5. Secondary Testing

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

To the main application class file I have added a document header:

```java
/**********************************************************************
**
** CS-305 Project 2
** SslServerApplication.java
**
** Dante Trisciuzzi
** Southern New Hampshire University
** CS-305-T3311 Software Security
** Dr. Vivian Lyon
** February 15, 2023
**
**********************************************************************/

package com.snhu.sslserver;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SslServerApplication {

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

I have implemented my algorithm cipher in its own static class for usage across the application where necessary. I have included thorough code comments as well to explain my logic:

```java
/**************************************************************
**
** CS-305 Project 2
** CipherProvider.java
**
** Dante Trisciuzzi
** Southern New Hampshire University
** CS-305-T3311 Software Security
** Dr. Vivian Lyon
** February 15, 2023
**
**************************************************************/

package com.snhu.sslserver;

import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public final class CipherProvider {

    /**
     * Converts a byte array to a hexadecimal string.
     *
     * @param bytes the byte array to convert
     * @return a hexadecimal string representing the input bytes
     */
    private static String bytesToHexString(byte[] bytes) {

        // Declare (constant) hex mask
        final int MASK = 0xff;

        // Create our output string builder
        StringBuilder hexStringBuffer = new StringBuilder();

        // ForEach loop to mash each byte
        for (byte hashByte : bytes) {
            int intVal = MASK & hashByte;
            hexStringBuffer.append(String.format("%02x", intVal));
        }

        // Return output as a string
        return hexStringBuffer.toString();
    }


    /**
     * Returns a SHA-256 Hash for the given string
     *
     * @param input The input string
     * @return The hexadecimal string for the SHA-256 checksum
     */
    static String getSh256HashString(String input) {

        // Use try/catch here in case our algorithm is missing from the digest
        try {

            //Setup algorithm cipher requirements
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] inputBytes = input.getBytes(StandardCharsets.UTF_8);

            // Create hash
            byte[] hashBytes = md.digest(inputBytes);

            //return hash as a string
            return bytesToHexString(hashBytes);

        // On missing algorithm
        } catch (NoSuchAlgorithmException e) {

            // Use a runtime exception to avoid declaring the checked exception
            throw new RuntimeException("SHA-256 algorithm not found", e);
        }
    }
}
```

I have added a class to define the server endpoints, and implemented the '/hash' endpoint:

```java
/*****************************************************************
**
** CS-305 Project 2
** ServerEndpoints.java
**
** Dante Trisciuzzi
** Southern New Hampshire University
** CS-305-T3311 Software Security
** Dr. Vivian Lyon
** February 15, 2023
**
*****************************************************************/

package com.snhu.sslserver;

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

@RestController
class ServerEndpoints {

    /**
     * The function for the '/hash' API end point
     *
     * @return a demo output for the hashing algorithm
     */
    @RequestMapping("/hash")
    public String demonstrateHash() {

        // Create a string builder for the return value
        StringBuilder sb = new StringBuilder();

        // Create (constant) data and append formatting to the output builder
        final String data = "Dante Trisciuzzi";
        sb.append("<p>Original Data: ").append(data);

        // Generate hash and append to output builder
        final String hash = CipherProvider.getSh256HashString(data);
        sb.append("</p><p>Cipher Used: SHA-256 | Hashed Data: ").append(hash).append("</p>");

        // Return formatted output
        return sb.toString();
    }
}
```

Here is the header from my generated dependency check report:

Project: ssl-server

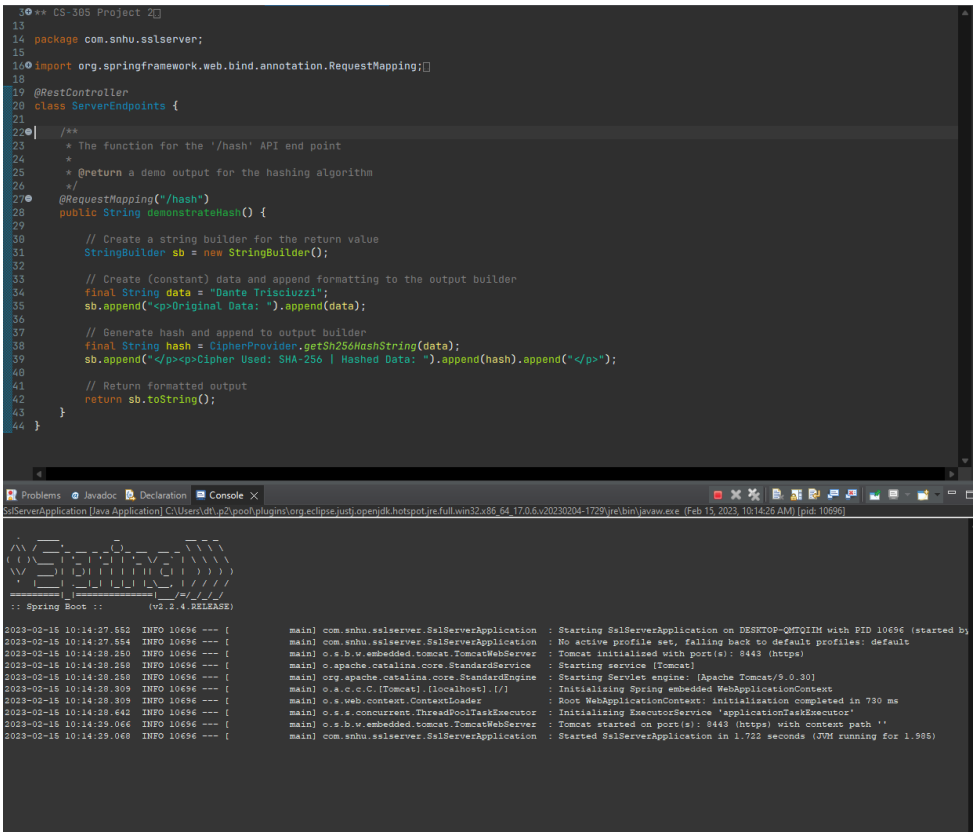
com.snhu:ssl-server:0.0.1-SNAPSHOT

Scan Information (show less):

- *dependency-check version*: 5.3.0
- *Report Generated On*: Wed, 15 Feb 2023 10:12:06 -0500
- *Dependencies Scanned*: 49 (34 unique)
- *Vulnerable Dependencies*: 15
- *Vulnerabilities Found*: 55
- *Vulnerabilities Suppressed*: 0
- *NVD CVE Checked*: 2023-02-15T09:54:26
- *NVD CVE Modified*: 2023-02-15T08:00:05
- *VersionCheckOn*: 2023-02-06T10:36:43

6. Functional Testing

Here is a screenshot of the application (with the refactored code) running as expected, and without errors:



7. Summary

Upon inspection of the code, there are several areas of security that are severely lacking; APIs and Cryptography (SNHU, 2023). These two sections are critical for our application specifically, however code error, code quality, and encapsulation are concerns for any application and must be considered as well (*Iron-Clad Java: Building Secure Web Applications*, 2014).

During the refactoring process, I have added two important elements to the application to address the API and Cryptography issues. The first being a class to handle the endpoints for the SpringBoot application's RESTful service, this class is located in `ServerEndpoints.java`. Second, I added a class to provide the cipher function to the application and is located in the `CipherProvider.java` file. I'll detail the refactoring process for each of these classes below, and the refactored code files are attached to this report.

ServerEndpoints.java

This class was created to extend the functionality of the application, by providing declarations and definitions for the RESTful API endpoints. I have added a single member function here, `demonstrateHash()`, for the "/hash" endpoint. This function produces a string output of some data (my name) being hashed with the SHA-256 hashing algorithm. It achieves this by using a StringBuilder object to build a basic HTML string that displays the data, the algorithm name, and the SHA-256 checksum. It makes use of the static member `CipherProvider.getSha256HashString(...)` from the CipherProvider class (detailed below). I opted for a StringBuilder instead of incrementally appending to a string, because it is more performant, in my experience. I have added JavaDoc comments as well as inline comments detailing my code.

CipherProvider.java

This class has been created to provide the SHA-256 hashing algorithm to the application. I created this class as `final` because there are no class members that need to be modified during runtime execution. All the methods in this class are also defined as final, and static since they do not need to be relative to an instance. This class contains two methods; `bytesToHexString(...)` and `getSha256HashString(...)`.

The `bytesToHexString(...)` method takes an array of bytes and outputs a string of those bytes. The method accomplishes this by using a StringBuilder object and appending each byte as it is converted (using a hex mask and a bitwise AND operation) to the StringBuilder. Once all the bytes have been converted, the method returns a string from the builder instance. This method is private, since it is intended for use by this class.

The `getSha256HashString(...)` method takes a string input and returns the SHA-256 checksum for that string, in a string format. In this method we perform this task by getting an instance of the MessageDigest's SHA-256 algorithm (provided by Java.Security). If this fails, we handle the error by throwing a runtime exception that details the issue. If this is a success, we proceed to converting and hashing the input string to a byte array. Then we pass this array to the aforementioned method (bytesToHexString(...)). Finally, we return the result of the bytesToHexString(...) method. This method is made available package-wide so we may use it for our RESTful API service.

The changes I have made via refactoring and adding new classes have resulted in no new vulnerabilities on the dependency check report.

8. Industry Standard Best Practices

Following industry standard best practices can be rather subjective since there are different sets of standards. However, there are some Java-specific standards that I always strictly adhere to in my code. Namely, I preserve naming conventions across all my project files, and I try to use the most readable and maintainable solutions to problems. A good example of such a solution is the approach I took to building strings in this project. I opted to use StringBuilder objects wherever appropriate, rather than just appending to a string. Not only does this result in better performance, but it makes the code much more modular and readable to a future maintainer.

Using best practices and adhering to a standard is important to the overall longevity of the application code base. This is because future maintainers of the code base are expected to be familiar with these standards as well. Consider some messy cryptic code that is ‘working’ today, what if some component design changes and the code base needs to be modified to accommodate? Larger sections of the code may need to be revised or completely rewritten, resulting in much more work for the maintainers, and higher cost to the company as a whole.

Resources

Iron-Clad Java: Building Secure Web Applications. (2014). McGraw-Hill Companies.

Java Cryptography Architecture (JCA) Reference Guide. (2022). Oracle. https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html

SNHU. (2023). *SNHU Vulnerability Chart*. Retrieved February 15, 2023, from https://learn.snhu.edu/content/enforced/1237134-CS-305-T3311-OL-TRAD-UG.23EW3/course_documents/CS%20305%20Vulnerability%20Assessment%20Process%20Flow%20Diagram.pdf?_&d2lSessionVal=H2eLNhRVDOqc30eBUmKsM7NJo&ou=1237134