GLOBALRAIN

Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	February 22, 2024	Matt Jackson	

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.

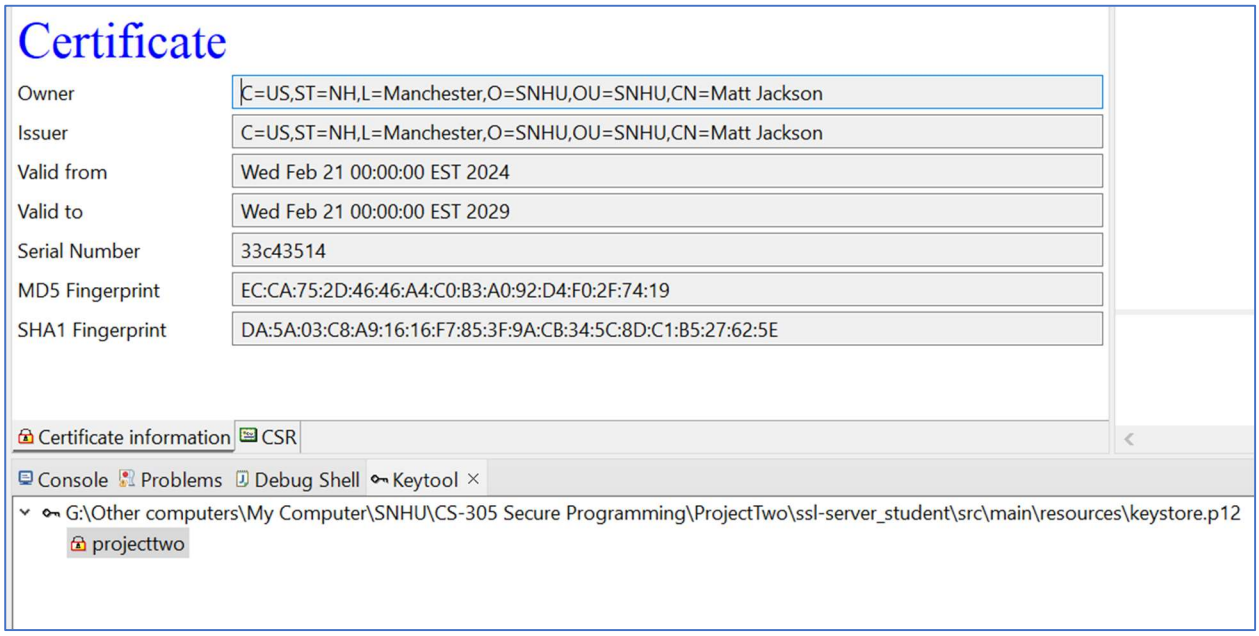
Developer
Matt Jackson

1. Algorithm Cipher

Hashing functions are used to ensure data integrity by generating a unique identifier for given data, to include a file. (NCCIC, n.d.) Previously used hashing algorithms MD5 and SHA-1 have recently proven vulnerable to collision and brute-force attacks. (Velvindron, Moriarty, & Ghedini, 2021) (NIST, 2022) The United States National Institute of Standards and Technology (NIST) recommends the use of the SHA-2 or SHA-3 family of hashes, with a minimum of SHA-256 "for any applications of hash functions requiring interoperability". (NIST, 2015). In the SHA-2 family, SHA-256 and SHA-512 are vulnerable to length extension attacks, wherein the attacker "can still generate a valid hash for {secret || data || attacker_controlled_data}". (Bowes, 2012) SHA-384 is not vulnerable to this type of attack, and is also in the SHA-3 family, which suggests longevity. At least in the python implementation, SHA-384 is 50% faster than SHA-256. (PyCryptodome, n.d.) It produces a 384 bit hash. Hash functions can be used to generate pseudo-random numbers by seeding with a secret value.

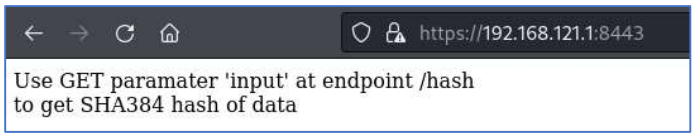
2. Certificate Generation

Insert a screenshot below of the CER file.



3. Deploy Cipher

Insert a screenshot below of the checksum verification.



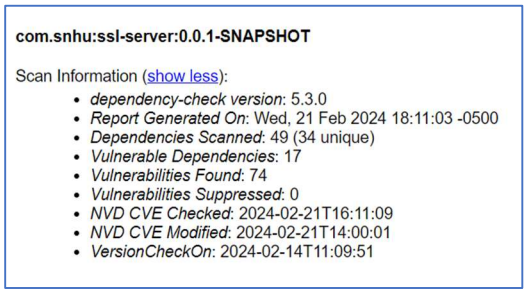
4. Secure Communications

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

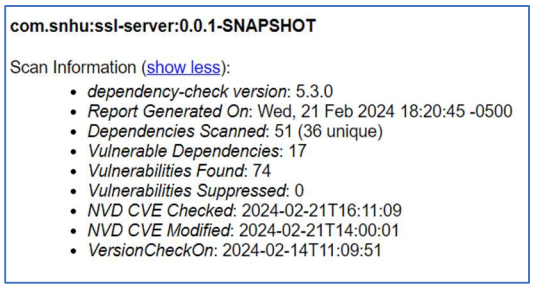
5. Secondary Testing

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

The initial vulnerability report showed 49 dependencies with 74 vulnerabilities:



After adding two dependencies, we see that we did not introduce any new known vulnerabilities.



6. Functional Testing

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

```java
package com.snhu.sslserver;

import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.time.Duration;

import org.owasp.encoder.Encode;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import io.github.bucket4j.Bandwidth;
import io.github.bucket4j.Bucket;
import io.github.bucket4j.ConsumptionProbe;
import io.github.bucket4j.Refill;

@SpringBootApplication
public class SslServerApplication {

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

@RestController
class GreetingController {

    public String getSHA384String(String plaintext) {

        MessageDigest md;
        try {
            md = MessageDigest.getInstance("SHA-384");
            byte[] messageDigest = md.digest(plaintext.getBytes());

            BigInteger num = new BigInteger(1, messageDigest);

            String hashtext = num.toString(16);

            return hashtext;
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }

        return "hash failed";
    }

    @GetMapping("/")
    public String index() {
        String data = "Use GET paramater 'input' at endpoint /hash" + "<br>"
                    + "to get SHA384 hash of data";
        return data;
    }

    // Bucket rate limiter adapted from https://www.baeldung.com/spring-bucket4j
        private final Bucket bucket;

            public GreetingController() {
                Bandwidth limit = Bandwidth.classic(20, Refill.greedy(20, Duration.ofMinutes(1)));
                this.bucket = Bucket.builder()
                    .addLimit(limit)
                    .build();
            }
```

```
69
70      @GetMapping("/hash")
71      //public String hash(@RequestParam(value = "input", defaultValue = "Hello, World") String input) {
72      public ResponseEntity hash(@RequestParam(value = "input", defaultValue = "Hello, World") String input) {
73          String data = "Project Two" + "<br>"
74                      + "Matt Jackson" + "<br><br>"
75                      //+ "SHA384 hash of " + input + ":<br>"
76                      //+ getSHA384String(input);
77                      + "SHA384 hash of " + Encode.forHtml(input) + ":<br>"
78                      + getSHA384String(Encode.forHtml(input)) +"\n";
79
80          // Tailored response adapted from https://www.baeldung.com/spring-bucket4j
81          // if there are enough items in the bucket, return the encoded string
82          // else return a 429 error and header with number of seconds to wait before
83          // retrying the request.
84
85          ConsumptionProbe probe = bucket.tryConsumeAndReturnRemaining(1);
86          if (probe.isConsumed()) {
87              HttpHeaders headers = new HttpHeaders();
88              headers.add("X-Rate-Limit-Remaining", Long.toString(probe.getRemainingTokens()));
89
90              return ResponseEntity.ok()
91                      .headers(headers)
92                      .body(data);
93          }
94
95          long waitForRefill = probe.getNanosToWaitForRefill() / 1_000_000_000;
96          return ResponseEntity.status(HttpStatus.TOO_MANY_REQUESTS)
97              .header("X-Rate-Limit-Retry-After-Seconds", String.valueOf(waitForRefill))
98              .build();
99
100 //      return data;
101
102     }
103
104 }
```

```
Console ×
SslServerApplication [Java Application] C:\Users\Mattj\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe  (Feb 22, 2024, 7:31:14 AM) [pid: 13032]

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.4.RELEASE)

2024-02-22 07:31:17.010  INFO 13032 --- [           main] com.snhu.sslserver.SslServerApplication  : Starting SslServerApplication on Windows10 with PID 13032 (started
2024-02-22 07:31:17.018  INFO 13032 --- [           main] com.snhu.sslserver.SslServerApplication  : No active profile set, falling back to default profiles: default
2024-02-22 07:31:19.839  INFO 13032 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8443 (https)
2024-02-22 07:31:19.863  INFO 13032 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-02-22 07:31:19.863  INFO 13032 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.30]
2024-02-22 07:31:20.034  INFO 13032 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-02-22 07:31:20.035  INFO 13032 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 2888 ms
2024-02-22 07:31:21.777  INFO 13032 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2024-02-22 07:31:23.742  INFO 13032 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8443 (https) with context path ''
2024-02-22 07:31:23.759  INFO 13032 --- [           main] com.snhu.sslserver.SslServerApplication  : Started SslServerApplication in 7.638 seconds (JVM running for 8.54
2024-02-22 07:33:42.338  INFO 13032 --- [nio-8443-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-02-22 07:33:42.338  INFO 13032 --- [nio-8443-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-02-22 07:33:42.387  INFO 13032 --- [nio-8443-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 49 ms
```

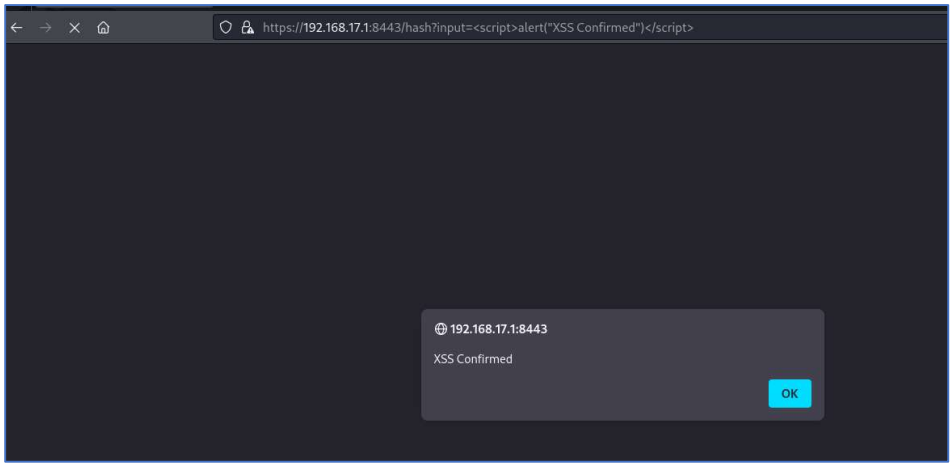
7. Summary

For this application, we examine two security concerns. The Open Worldwide Application Security Project keeps a list of the top security threats to web applications. On the 2023 list, Unsafe Consumption of API to include injection and denial of service attacks was number ten. **(OWASP, 2023)** Our application is vulnerable to injection and denial of service.

The injection vulnerability comes from the simple page mapping

```
@GetMapping ("/hash")
public String hash(@RequestParam(value = "input", defaultValue =
       "Hello, World") String input)
```

By not sanitizing the input parameter, the code is vulnerable to cross-site scripting. We can inject javascript into the input parameter controlled by the client, resulting in a popup alert when the subsequent page tries to render:



We can apply security measures to eliminate this by integrating the OWASP java encoder. (OWASP, n.d.).

First we add

```
<dependency>
	<groupId>org.owasp.encoder</groupId>
	<artifactId>encoder</artifactId>
	<version>1.2.3</version>
</dependency>
```

to our pom.xml file then importing org.owasp.encoder.Encode, we can utilize the encoding library to ensure our parameter is not interpreted as code:

```
String data = "Project Two" + "<br>"
		+ "Matt Jackson" + "<br><br>"
		//+ "SHA384 hash of " + input + ":<br>"
		//+ getSHA384String(input);
		+ "SHA384 hash of " + Encode.forHtml(input) + ":<br>"
		+ getSHA384String(Encode.forHtml(input));
```

Resulting in our script rendered in text with no popup:

To help prevent a denial-of-service attack, we can limit the rate of requests made to the web page. Adapting the example from https://www.baeldung.com/spring-bucket4j, we create a bucket object that limits access to 20 times in one minute. Then we change the hash method return type from a simple string to a ResponseEntity object and return a response with customized headers, warning us how many probes are left in the bucket:

```
	private final Bucket bucket;

	public GreetingController() {
		Bandwidth limit = Bandwidth.classic(20, Refill.greedy(20,
Duration.ofMinutes(1)));
		this.bucket = Bucket.builder()
				.addLimit(limit)
				.build();
	    }

	@GetMapping("/hash")
	//public String hash(@RequestParam(value = "input", defaultValue = "Hello,
World") String input) {
	public ResponseEntity hash(@RequestParam(value = "input", defaultValue =
"Hello, World") String input) {
		String data = "Project Two" + "<br>"
				+ "Matt Jackson" + "<br><br>"
				//+ "SHA384 hash of " + input + ":<br>"
				//+ getSHA384String(input);
				+ "SHA384 hash of " + Encode.forHtml(input) +
":<br>"
				+ getSHA384String(Encode.forHtml(input)) +"\n";

		// Tailored response adapted from https://www.baeldung.com/spring-
bucket4j
		// if there are enough items in the bucket, return the encoded string
		// else return a 429 error and header with number of seconds to wait
before
		// retrying the request.

		ConsumptionProbe probe = bucket.tryConsumeAndReturnRemaining(1);
	    if (probe.isConsumed()) {
		HttpHeaders headers = new HttpHeaders();
		headers.add("X-Rate-Limit-Remaining",
Long.toString(probe.getRemainingTokens()));

		return ResponseEntity.ok()
			.headers(headers)
			.body(data);
```

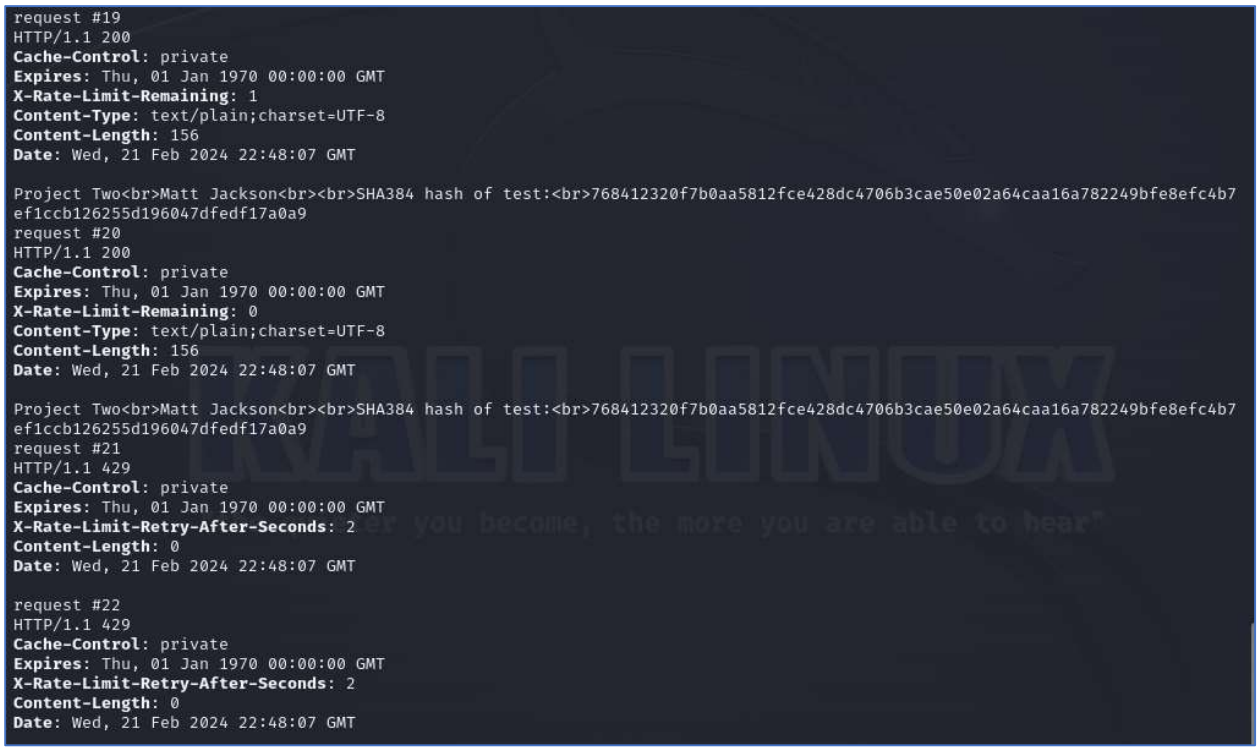
```
        }

        long waitForRefill = probe.getNanosToWaitForRefill() / 1_000_000_000;
        return ResponseEntity.status(HttpStatus.TOO_MANY_REQUESTS)
            .header("X-Rate-Limit-Retry-After-Seconds",
String.valueOf(waitForRefill))
            .build();

//            return data;
```

Running a simple script to send multiple web requests shows that indeed our access rate has been limited. After the 20th request, we receive a 429 error.

```
$ for i in {0..25};
        do echo 'request #'$i;
        curl -k -i "https://192.168.17.1:8443/hash?input=test";
done
```



8. Industry Standard Best Practices

In this project we used the SNHU Vulnerability Assessment Process Flow chart to identify areas of concern: input validation, code error and code quality. By applying tools from the OWASP Java Encoder project we were able to validate user generated input and prevent Cross-Site Scripting attacks. Adding the try/catch clause to our hash generator prevents production of a 500 error for a

NoSuchAlgorithmException, which reveals underlying system information about the programming language used. Finally, using maven's dependency checker we ensured our refactoring didn't introduce any new vulnerabilities.

Generating a secure hash to aid in verifying file integrity as well as ensuring the site only runs on the https protocol help to will meet Artemis Financial's goal of protecting their client data and financial information

References

Bowes, R. (2012, September 25). *Everything you need to know about hash length extension attacks*. Retrieved from SkullSecurity: https://www.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks

NCCIC. (n.d.). *NCCIC National Cybersecurity and*. Retrieved from cisa.gov: https://www.cisa.gov/sites/default/files/FactSheets/NCCIC%20ICS_Factsheet_File_Hashing_S508C.pdf

NIST. (2015, August 15). *NIST.gov*. Retrieved from NIST Computer Security Resource Center: https://csrc.nist.gov/projects/hash-functions/nist-policy-on-hash-functions

NIST. (2022, December 15). *NIST Transitioning Away from SHA-1 for All Applications*. Retrieved from NIST Computer Security Resource Center: https://csrc.nist.gov/news/2022/nist-transitioning-away-from-sha-1-for-all-apps

OWASP. (2023). *OWASP Top 10 API Security Risks – 2023*. Retrieved from OWASP API Security: https://owasp.org/API-Security/editions/2023/en/0x11-t10/

OWASP. (n.d.). *OWASP Java Encoder*. Retrieved from OWASP: https://owasp.org/www-project-java-encoder/

PyCryptodome. (n.d.). *SHA-384¶*. Retrieved from PyCryptodome: https://pycryptodome.readthedocs.io/en/latest/src/hash/sha384.html

Velvindron, L., Moriarty, K., & Ghedini, A. (2021, December). *RFC 9155*. Retrieved from IETF Datatracker: https://datatracker.ietf.org/doc/rfc9155/