登录

CVE-2019-0193 Apache Solr远程命令执行漏洞分析

tornado / 2019-08-13 07:31:00 / 浏览数 4888 安全技术 漏洞分析 顶(0) 踩(0)

概述

近日，Apache Solr官方发布Apache
Solr存在一个远程代码执行漏洞（CVE-2019-0193），攻击者可利用dataConfig参数构造恶意请求，导致执行任意代码，下面简单分析一下这个漏洞。

官方通告：https://issues.apache.org/jira/browse/SOLR-13669

前置概念

Dataimport

Solr支持从Dataimport导入自定义数据，dataconfig需要满足一定语法，参考

https://lucene.apache.org/solr/guide/6_6/uploading-structured-data-store-data-with-the-data-import-handler.html

https://cwiki.apache.org/confluence/display/solr/DataImportHandler

其中ScriptTransformer可以编写自定义脚本，支持常见的脚本语言如Javascript、JRuby、Jython、Groovy和BeanShell

ScriptTransformer容许用脚本语言如Javascript、JRuby、Jython、Groovy和BeanShell转换，函数应当以行（类型为Map\<String,Object\>）为参数，可以修改字段。

使用示例：

```
<dataconfig>
 <script><![CDATA[
   function f2c(row) {
     var tempf, tempc;
     tempf = row.get('temp_f');
     if (tempf != null) {
       tempc = (tempf - 32.0)*5.0/9.0;
       row.put('temp_c', temp_c);
     }
     return row;
   }
   ]]>
 </script>
 <document>

   <entity name="e1" pk="id" transformer="script:f2c" query="select * from X">
   </entity>
 </document>
</dataConfig>
```

Nashorn引擎

在Solr中解析js脚本使用的是Nashorn引擎，可以通过Java.typeAPI在JavaScript中引用，就像Java的import一样，例如：

```
var MyJavaClass = Java.type('my.package.MyJavaClass');
var result = MyJavaClass.sayHello("Nashorn");
print(result);
```

漏洞分析

Solr在处理dataimport请求时，首先进入dataimport/DataImportHandler的handleRequestBody方法，当前请求的command为full-import，因此通过maybeReloadCo

```
} else {
    this.importer.maybeReloadConfiguration(requestParams, defaultParams);
    UpdateRequestProcessorChain processorChain = req.getCore().getUpdateProcessorChain(params);
    UpdateRequestProcessor processor = processorChain.createProcessor(req, rsp);
    SolrResourceLoader loader = req.getCore().getResourceLoader();
    DIHWriter sw = this.getSolrWriter(processor, loader, requestParams, req);
    if (requestParams.isDebug()) {
        if (this.debugEnabled) {
            this.importer.runCmd(requestParams, sw);
            rsp.add("mode", "debug");
            rsp.add("documents", requestParams.getDebugInfo().debugDocuments);
            if (requestParams.getDebugInfo().debugVerboseOutput != null) {
                rsp.add("verbose-output", requestParams.getDebugInfo().debugVerboseOutput);
            }
        } else {
            message = "Debug not enabled. Add a tag <str name=\"enableDebug\">true</str> in solrconfig.xml";
        }
    } else if (requestParams.getContentStream() == null && !requestParams.isSyncMode()) {
        this.importer.runAsync(requestParams, sw);
    } else {
        this.importer.runCmd(requestParams, sw);
    }
}
```

先知社区

在maybeReloadConfiguration中通过params.getDataConfig()判断了post的数据(dataConfig)是否为空，如果不是则通过loadDataConfig来加载

```
boolean maybeReloadConfiguration(RequestInfo params, NamedList<?> defaultParams) throws IOException {
    if (this.importLock.tryLock()) {
        boolean success = false;

        try {
            if (null != params.getRequest() && this.schema != params.getRequest().getSchema()) {
                this.schema = params.getRequest().getSchema();
            }

            String dataConfigText = params.getDataConfig();
            String dataconfigFile = params.getConfigFile();
            InputSource is = null;
            if (dataConfigText != null && dataConfigText.length() > 0) {
                is = new InputSource(new StringReader(dataConfigText));
            } else if (dataconfigFile != null) {
                is = new InputSource(this.core.getResourceLoader().openResource(dataconfigFile));
                is.setSystemId(SystemIdResolver.createSystemIdFromResourceName(dataconfigFile));
                log.info("Loading DIH Configuration: " + dataconfigFile);
            }

            if (is != null) {
                this.config = this.loadDataConfig(is);
                success = true;
            }
```

先知社区

随后在loadDataConfig中通过readFromXml方法解析提交的配置数据中的各个标签，比如document，script，function，dataSource等，传入的script自定义脚本即在此获取到配置信息后通过this.importer.runCmd()方法处理导入过程

```
this.importer.runCmd(requestParams, sw);
```

```
void runCmd(RequestInfo reqParams, DIHWriter sw) {
    String command = reqParams.getCommand();
    if (command.equals("abort")) {
        if (this.docBuilder != null) {
            this.docBuilder.abort();
        }

    } else if (!this.importLock.tryLock()) {
        log.warn("Import command failed . another import is running");
    } else {
        try {
            if (!"full-import".equals(command) && !"import".equals(command)) {
                if (command.equals("delta-import")) {
                    this.doDeltaImport(sw, reqParams);
                }
            } else {
                this.doFullImport(sw, reqParams);
            }
        } finally {
            this.importLock.unlock();
        }

    }
}
```

先知社区

在doFullImport中，首先会创建一个DocBuilder对象，DocBuilder的主要功能是从给定配置中创建Solr文档，同时会记录一些状态信息。随后通过execute()方法会通过遍历
在解析完config数据后solr会把最后更新时间记录到配置文件中，这个时间是为了下次进行增量更新的时候用的。接着通过this.dataImporter.getStatus()判断当前数据导入

```
if (this.dataImporter.getStatus() == Status.RUNNING_DELTA_DUMP) {
    this.cleanByQuery(delQuery, fullCleanDone);
    this.doDelta();
    delQuery = (String)epw.getEntity().getAllAttributes().get("postImportDeleteQuery");
    if (delQuery != null) {
        fullCleanDone.set(false);
        this.cleanByQuery(delQuery, fullCleanDone);
    }
} else {
    this.cleanByQuery(delQuery, fullCleanDone);
    this.doFullDump();
    delQuery = (String)epw.getEntity().getAllAttributes().get("postImportDeleteQuery");
    if (delQuery != null) {
        fullCleanDone.set(false);
        this.cleanByQuery(delQuery, fullCleanDone);
    }
}
```

先知社区

在doFullDump()中调用的是DocBuilder.buildDocument()方法，这个方法会为发送的配置数据的每一个processor做解析，当发送的entity中含有Transformers时，会进行
在读取EntityProcessorWrapper的每一个元素时，是通过epw.nextRow()调用的，它返回的是一个Map对象，进入EntityProcessorWrapper.nextRow方法

```java
public Map<String, Object> nextRow() {
    if (this.rowcache != null) {
        return this.getFromRowCache();
    } else {
        Map arow;
        do {
            arow = null;

            try {
                arow = this.delegate.nextRow();
            } catch (Exception var3) {
                if (!"abort".equals(this.onError)) {
                    SolrException.log(log, "Exception in entity : " + this.entityName, var3);
                    return null;
                }

                DataImportHandlerException.wrapAndThrow( err: 500, var3);
            }

            if (arow == null) {
                return null;
            }

            arow = this.applyTransformer(arow);
        } while(arow == null);

        this.delegate.postTransform(arow);
        return arow;
    }
}
```

先知社区

通过applyTransformer()执行转换，调用的是相应Transformer的transformRow方法

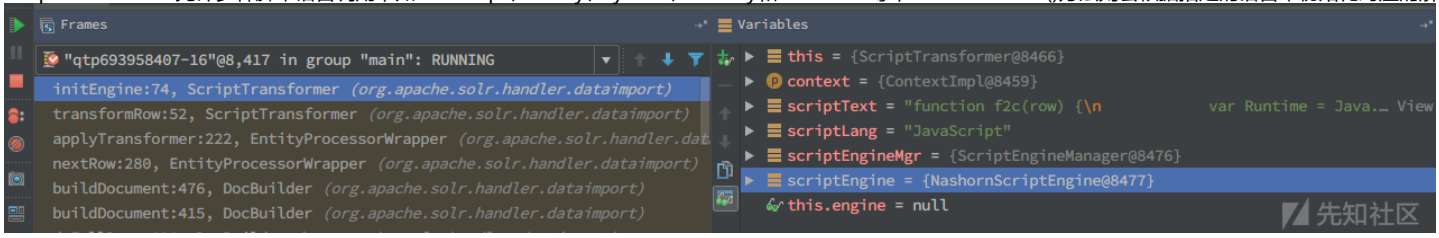
```java
while(var6.hasNext()) {
    Transformer t = (Transformer)var6.next();  t: ScriptTransformer@8339
    if (stopTransform) {  stopTransform: false
        break;
    }

    try {
        if (rows == null) {  rows: null
            resolver.addNamespace(this.entityName, transformedRow);  resolver: VariableResolver@8330
            Object o = t.transformRow(transformedRow, this.context);  t: ScriptTransformer@8339  transform
            if (o == null) {
                t = {ScriptTransformer@8339}
            }

            if (o instanceof Map) {
                Map oMap = (Map)o;
                stopTransform = this.checkStopTransform(oMap);
                transformedRow = (Map)o;
            } else if (o instanceof List) {
                rows = (List)o;
            } else {
                log.error("Transformer must return Map<String, Object> or a List<Map<String, Object>>");
            }
```

先知社区

ScriptTransformer允许多种脚本语言调用，如Javascript、JRuby、Jython、Groovy和BeanShell等，transformRow()方法则会根据指定的语言来初始化对应的解析引擎，



先知社区

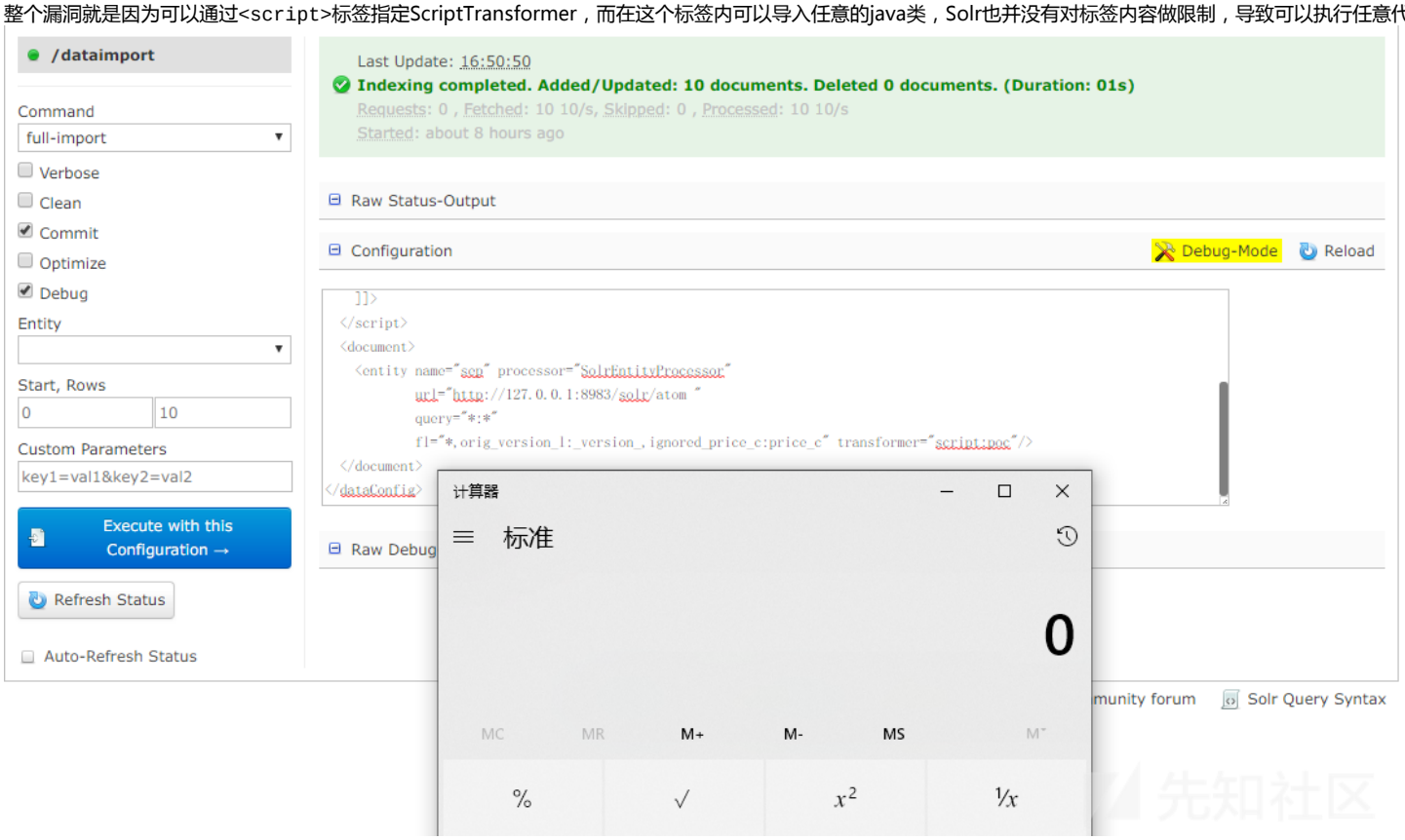
Solr中默认的js引擎是Nashorn，Nashorn是在Java 8中用于取代Rhino（Java 6，Java 7）的JavaScript引擎，在js中可以通过Java.type引用Java类，就像Java的import一样，此处就可以通过这个语法导入任意Java类。
随后通过反射调用自定义的函数并执行，例如通过java.lang.Runtime执行系统命令

```
public Object transformRow(Map<String, Object> row, Context context) {
    try {
        if (this.engine == null) {
            this.initEngine(context);
        }

        return this.engine == null ? row : this.engine.invokeFunction(this.functionName, row, context);
    } catch (DataImportHandlerException var4) {
        throw var4;
    } catch (Exception var5) {
        DataImportHandlerException.wrapAndThrow( err: 500, var5,  msg: "Error invoking script for entity " + context.getEntityAttribute( s: "name"));
        return null;
    }
}
```

先知社区

整个漏洞就是因为可以通过<script>标签指定ScriptTransformer，而在这个标签内可以导入任意的java类，Solr也并没有对标签内容做限制，导致可以执行任意代码。



先知社区

调用栈情况

```
invoke:494, ScriptFunction (jdk.nashorn.internal.runtime)
apply:393, ScriptRuntime (jdk.nashorn.internal.runtime)
callMember:199, ScriptObjectMirror (jdk.nashorn.api.scripting)
invokeImpl:386, NashornScriptEngine (jdk.nashorn.api.scripting)
invokeFunction:190, NashornScriptEngine (jdk.nashorn.api.scripting)
transformRow:55, ScriptTransformer (org.apache.solr.handler.dataimport)
applyTransformer:222, EntityProcessorWrapper (org.apache.solr.handler.dat
nextRow:280, EntityProcessorWrapper (org.apache.solr.handler.dataimport)
buildDocument:476, DocBuilder (org.apache.solr.handler.dataimport)
buildDocument:415, DocBuilder (org.apache.solr.handler.dataimport)
doFullDump:330, DocBuilder (org.apache.solr.handler.dataimport)
execute:233, DocBuilder (org.apache.solr.handler.dataimport)
doFullImport:424, DataImporter (org.apache.solr.handler.dataimport)
runCmd:483, DataImporter (org.apache.solr.handler.dataimport)
handleRequestBody:184, DataImportHandler (org.apache.solr.handler.dataimp
handleRequest:199, RequestHandlerBase (org.apache.solr.handler)
execute:2551, SolrCore (org.apache.solr.core)
execute:711, HttpSolrCall (org.apache.solr.servlet)
call:516, HttpSolrCall (org.apache.solr.servlet)
```

先知社区

补充

值得注意的是，官方给出的临时修复方案并不能缓解漏洞，当把相应当index
core的配置文件置为空时，dataimport的时候只是获取不到默认的配置，但是依然能够通过这个接口发送PoC，漏洞也依然能够触发，解决办法是把相应配置文件中的data
requestHandler全部注释并重启Solr服务器，才能彻底关闭这个接口缓解漏洞。

```
<!--<requestHandler name="/dataimport" class="solr.DataImportHandler">
    <lst name="defaults">
      <str name="config">atom-data-config.xml</str>
      <str name="processor">trim_text</str>
    </lst>
  </requestHandler>
-->
```

NSFOCUS

点击收藏 | 0 关注 | 1

上一篇：使用 afl-unicorn: F... 下一篇：HTTP Desync Attac...

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板