

两天前发布了[漏洞公告](#),Apache Kylin<4.0.2存在命令注入漏洞。

没接触过这个框架,搜了一下,发现[历史漏洞](#)中命令注入的漏洞还挺多的。学习完历史漏洞后感觉还是挺简单的,都是直接一个路由直通命令执行点,可能路上有的需要改一下配置。接着自己[dif](#)一下,看看更新后做了什么改动。

删除了 `CliCommandExecutor.java` 中过滤的地方,写了个新的 `ParameterFilter`, 且加了一个正则,估计就是用来过滤命令的。

```
public static final String SPARK_CONF_REGULAR_EXPRESSION = "[`$|&;;]+"
```

第一次尝试

一开始发现定位的是那几个控制器,因为有爆漏洞的先例,而且在本次修复中也进行了修改。猜测是之前的正则被绕过了。

在经过手动调试后发现,无法走到命令执行的地方。之前修改漏洞配置的地方加了白名单,其他路径也都有限制,并没有找到绕过的方法。

第二次尝试

发现还有一个 `NSparkExecutable#dowork` 也加了Filter。

```
2    114    protected ExecuteResult doWork(ExecutableContext context) throws ExecuteException {
3    -    //context.setLogPath(getSparkDriverLogHdfsPath(context.getConfig()));
4    115    CubeManager cubeMgr = CubeManager.getInstance(KylinConfig.getInstanceFromEnv());
5    116    CubeInstance cube = cubeMgr.getCube(this.getCubeName());
5    117    KylinConfig config = cube.getConfig();
118 +
119 +    Map<String, String> overrideKylinProps = new HashMap<>();
120 +    LinkedHashMap<String, String> cubeConfig = cube.getDescriptor().getOverrideKylinProps();
121 +    LinkedHashMap<String, String> projectConfig = cube.getProjectInstance().getOverrideKylinProps();
122 +    overrideKylinProps.putAll(projectConfig);
123 +    overrideKylinProps.putAll(cubeConfig);
124 +    for (Map.Entry<String, String> configEntry : overrideKylinProps.entrySet()) {
125 +        ParameterFilter.checkSparkConf(configEntry.getKey());
126 +        ParameterFilter.checkSparkConf(configEntry.getValue());
127 +    }
```

该方法里面内容还是挺多的,简单扫描一下,发现确实存在调用链。

```
[+] detect vuln: start
    java/lang/ProcessBuilder.start
        org/apache/kylin/common/util/CliCommandExecutor.runNativeCommand
            org/apache/kylin/common/util/CliCommandExecutor.execute
                org/apache/kylin/job/common/ShellExecutable.doWork
[+] detect vuln: start
    java/lang/ProcessBuilder.start
        org/apache/kylin/common/util/CliCommandExecutor.runNativeCommand
            org/apache/kylin/common/util/CliCommandExecutor.execute
                org/apache/kylin/engine/spark/job/NSparkExecutable.runSparkSubmit
                    org/apache/kylin/engine/spark/job/NSparkExecutable.doWork
[+] detect vuln: start
    java/lang/ProcessBuilder.start
        org/apache/kylin/common/util/CliCommandExecutor.runNativeCommand
            org/apache/kylin/common/util/CliCommandExecutor.execute
                org/apache/kylin/engine/spark/job/NSparkExecutable.killOrphanApplicationIfExists
                    org/apache/kylin/engine/spark/job/NSparkExecutable.doWork
```

往前扫就直接断了,具体原因是因为前面都是调用的线程,无法直接定位调用到具体的方法调用。只有自己手动看一下了。

```
org/apache/kylin/job/impl/threadpool/DefaultScheduler$JobRunner.run
org/apache/kylin/job/execution/AbstractExecutable.execute
```

```
org/apache/kylin/job/impl/threadpool/DistributedScheduler$JobRunner.run
org/apache/kylin/job/execution/AbstractExecutable.execute
```

定位到了 `JobService#afterPropertiesSet`, `JobService` 有 `@Component` 注解, `afterPropertiesSet` 方法好像是注入到bean后自动触发。到这里就没往前找了,主要是调用的地方太多了,感觉不太好找。还有一点是因为Spring默认用的单例模式,所以只会调用一次 `afterPropertiesSet` 方法。所以我认为该漏洞是通过写入命令到文件里面,然后通过重启(懒加载也有可能)去触发。那么该方法里面应该会读取文件之类的。

```
public static final String KYLIN_CONF_PROPERTIES_FILE =
    "kylin.properties";
```

发现文件 `kylin.properties`,去搜一下该文件的配置使用,发现官方文档中有记录。

[Cube级别重写默认kylin.properties](#)

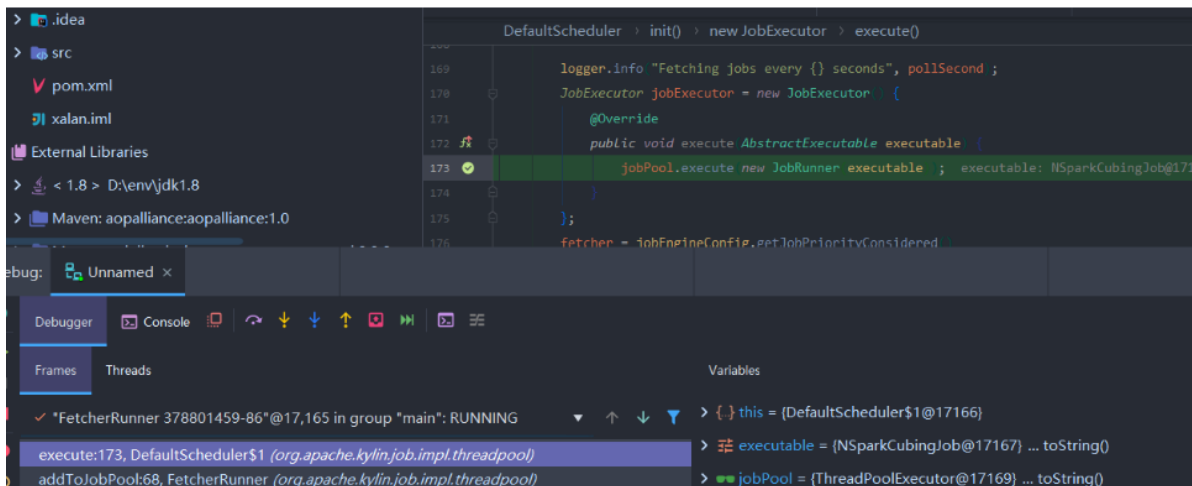
感觉一下明朗起来了,知道了这里是Cube模块的,而且还知道怎么进行修改。翻阅了下官方 [Cube方面的文档](#),感觉差不多可以确认POC了。在不断的调试中确定需要控制的参数,最后成功命令注入。

不过和官方通报的漏洞点不一样,但大概率也是换一个参数进行控制,就不研究了。


```
JobStepFactory > addStep()
54     case FILTER_RECOMMEND_CUBOID:
55         step = new NSparkLocalStep();
56         step.setSparkSubmitClassName FilterRecommendCuboidJob.class.getName();
57         step.setName ExecutableConstants.STEP_NAME_FILTER_RECOMMEND_CUBOID_DATA_FOR_OPTIMIZATION ;
58         break;
59     default:
60         throw new IllegalArgumentException();
61 }
62 step.setParams parent.getParams();
63 step.setProject parent.getProject(); step: NResourceDetectStep@19946 parent: NSparkCubingJob@17785
64 step.setTargetSubject parent.getTargetSubject();
65 parent.addTask step;
66 //after addTask, step's id is changed
67 step.setDistMetaUrl config.getJobTmpMetaStoreUrl parent.getProject(), step.getId());
68 return step;
```

通过默认的调度线程进行调用。

```
public Map<Integer, String> getSchedulers() {
    Map<Integer, String> r = Maps.newLinkedHashMap();
    r.put(0, "org.apache.kylin.job.impl.threadpool.DefaultScheduler");
    r.put(2, "org.apache.kylin.job.impl.threadpool.DistributedScheduler");
    r.put(77, "org.apache.kylin.job.impl.threadpool.NoopScheduler");
    r.putAll(convertKeyToInteger(getPropertiesByPrefix("kylin.job.scheduler.provider.")));
    return r;
}
```



```
DefaultScheduler > init() > new JobExecutor > execute()
169     logger.info "Fetching jobs every {} seconds", pollSecond ;
170     JobExecutor jobExecutor = new JobExecutor() {
171         @Override
172         public void execute AbstractExecutable executable {
173             jobPool.execute new JobRunner executable ; executable: NSparkCubingJob@17167
174         }
175     };
176     fetcher = jobEngineConfig.getJobPriorityConsidered();
```

Debugger: Unnamed x

Frames: Threads Variables

✓ "FetcherRunner 378801459-86"@17,165 in group "main": RUNNING

execute:173, DefaultScheduler\$1 (org.apache.kylin.job.impl.threadpool)

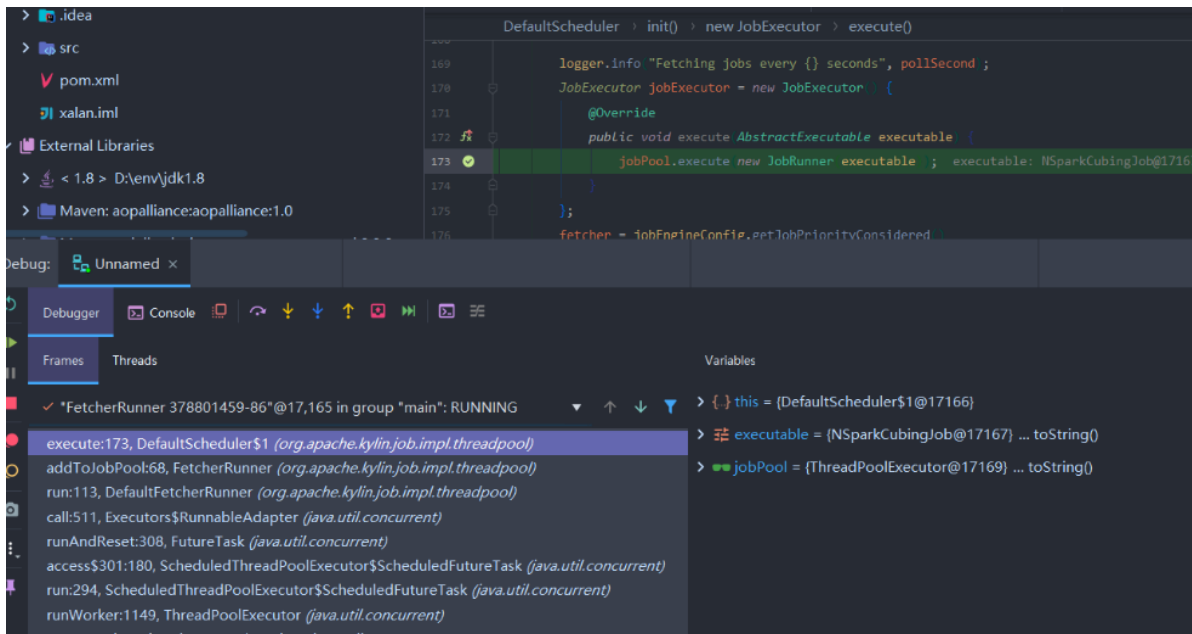
addToJobPool:68, FetcherRunner (org.apache.kylin.job.impl.threadpool)

Variables

> {} this = (DefaultScheduler\$1@17166)

> executable = (NSparkCubingJob@17167) ... toString()

> jobPool = (ThreadPoolExecutor@17169) ... toString()



```
DefaultScheduler > init() > new JobExecutor > execute()
169     logger.info "Fetching jobs every {} seconds", pollSecond ;
170     JobExecutor jobExecutor = new JobExecutor() {
171         @Override
172         public void execute AbstractExecutable executable {
173             jobPool.execute new JobRunner executable ; executable: NSparkCubingJob@17167
174         }
175     };
176     fetcher = jobEngineConfig.getJobPriorityConsidered();
```

Debugger: Unnamed x

Frames: Threads Variables

✓ "FetcherRunner 378801459-86"@17,165 in group "main": RUNNING

execute:173, DefaultScheduler\$1 (org.apache.kylin.job.impl.threadpool)

addToJobPool:68, FetcherRunner (org.apache.kylin.job.impl.threadpool)

run:113, DefaultFetcherRunner (org.apache.kylin.job.impl.threadpool)

call:511, Executors\$RunnableAdapter (java.util.concurrent)

runAndReset:308, FutureTask (java.util.concurrent)

access\$301:180, ScheduledThreadPoolExecutor\$ScheduledFutureTask (java.util.concurrent)

run:294, ScheduledThreadPoolExecutor\$ScheduledFutureTask (java.util.concurrent)

runWorker:1149, ThreadPoolExecutor (java.util.concurrent)

run:624, ThreadPoolExecutor\$Worker (java.util.concurrent)

Variables

> {} this = (DefaultScheduler\$1@17166)

> executable = (NSparkCubingJob@17167) ... toString()

> jobPool = (ThreadPoolExecutor@17169) ... toString()

二、定位到JobService的时候就可以去翻阅一下文档,看看该Service是干什么的,Service已经算是比较表层的了,应该还是比较好找的。

三、漏洞官方描述是控制--conf,进行单引号闭合,但是我这里的poc和官方描述不一样,是另外一个source。那么在修复的时候会不会存在一些官方没考虑到的情况?

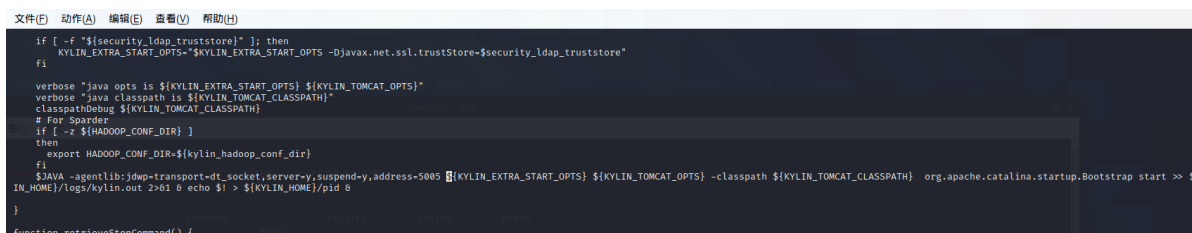
四、Spring挺基础的,感觉学好了很多时候能排上用场的。不能忽视基础。

环境搭建

```
docker pull apachekylin/apache-kylin-standalone:4.0.0
```

```
docker run -d -m 8G -p 7070:7070 -p 8088:8088 -p 50070:50070 -p  
8032:8032 -p 8042:8042 -p 2181:2181 -p 5005:5005  
apachekylin/apache-kylin-standalone:4.0.0
```

```
-  
agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005
```



```
文件(F) 动作(A) 编辑(E) 查看(V) 帮助(H)  
if [ -f "${security_ldap_truststore}" ]; then  
    KYLIN_EXTRA_START_OPTS="${KYLIN_EXTRA_START_OPTS} -Djavax.net.ssl.trustStore=${security_ldap_truststore}"  
fi  
  
verbose "java opts is ${KYLIN_EXTRA_START_OPTS} ${KYLIN_TOMCAT_OPTS}"  
verbose "java classpath is ${KYLIN_TOMCAT_CLASSPATH}"  
classpathDebug ${KYLIN_TOMCAT_CLASSPATH}  
# For Sparden  
if [ -z "${HADOOP_CONF_DIR}" ]  
then  
    export HADOOP_CONF_DIR=${kylin_hadoop_conf_dir}  
fi  
$JAVA -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=5005 ${KYLIN_EXTRA_START_OPTS} ${KYLIN_TOMCAT_OPTS} -classpath ${KYLIN_TOMCAT_CLASSPATH} org.apache.catalina.startup.Bootstrap start >> $  
IN_HOME/logs/kylin.out 2>&1 & echo $! > ${KYLIN_HOME}/pid &  
}  
  
function retrieveStopCommand() {
```

```
docker restart
```

lib在tomcat的webapp/kylin里面,或者maven导包(需要的包有点多)