

# 刷题之旅第10站,CTFshow misc50

原创

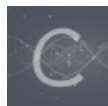
圆圈勾勒成指纹 于 2020-02-13 11:54:00 发布 2593 收藏 2

分类专栏: [刷题之旅100站](#) 文章标签: [python base64](#) [信息安全](#) [数据安全](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: [https://blog.csdn.net/weixin\\_45940434/article/details/104292369](https://blog.csdn.net/weixin_45940434/article/details/104292369)

版权



[刷题之旅100站](#) 专栏收录该内容

49 篇文章 11 订阅

订阅专栏

感谢@铭愁愁师傅提供的题目

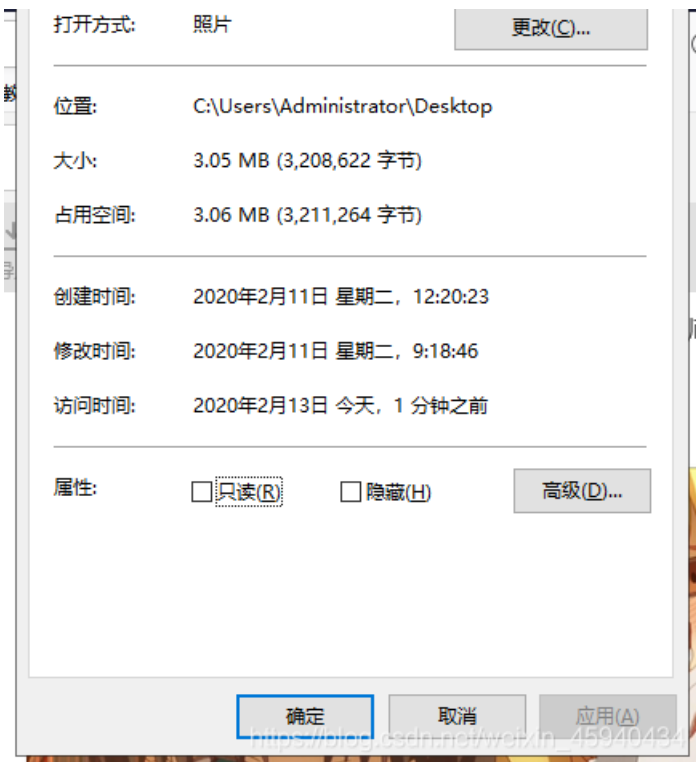
感谢ctf show平台提供题目

下载文件后, 发现是一张图片。

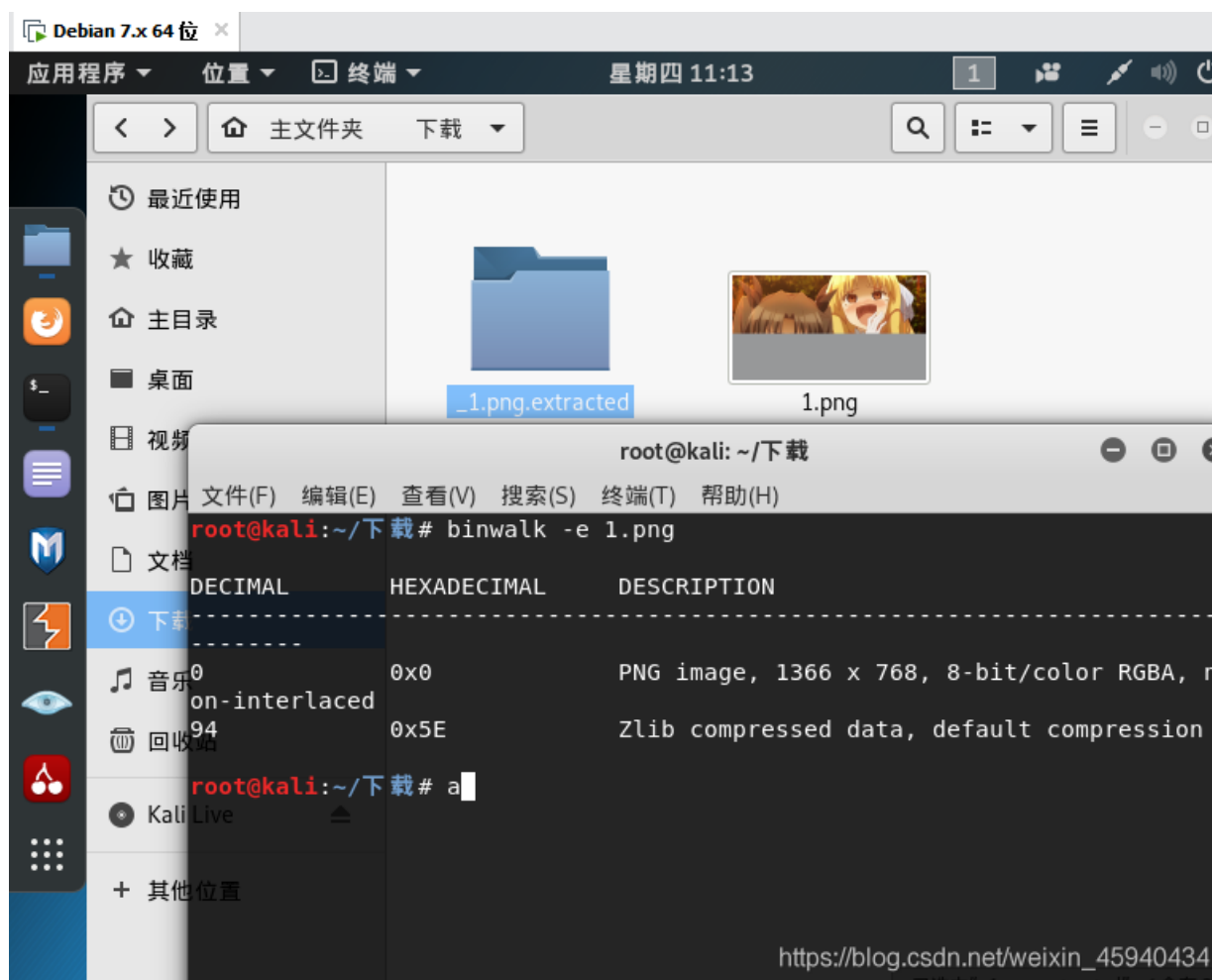


且图片大小 比较大, 正常图片一般都是1MB以内的, 所以猜测可能图片中有其他文件。





在kali下, 尝试binwalk 和 foremost 分离, 均没有结果。





于是尝试手动分离文件。

这里推荐一篇文章。

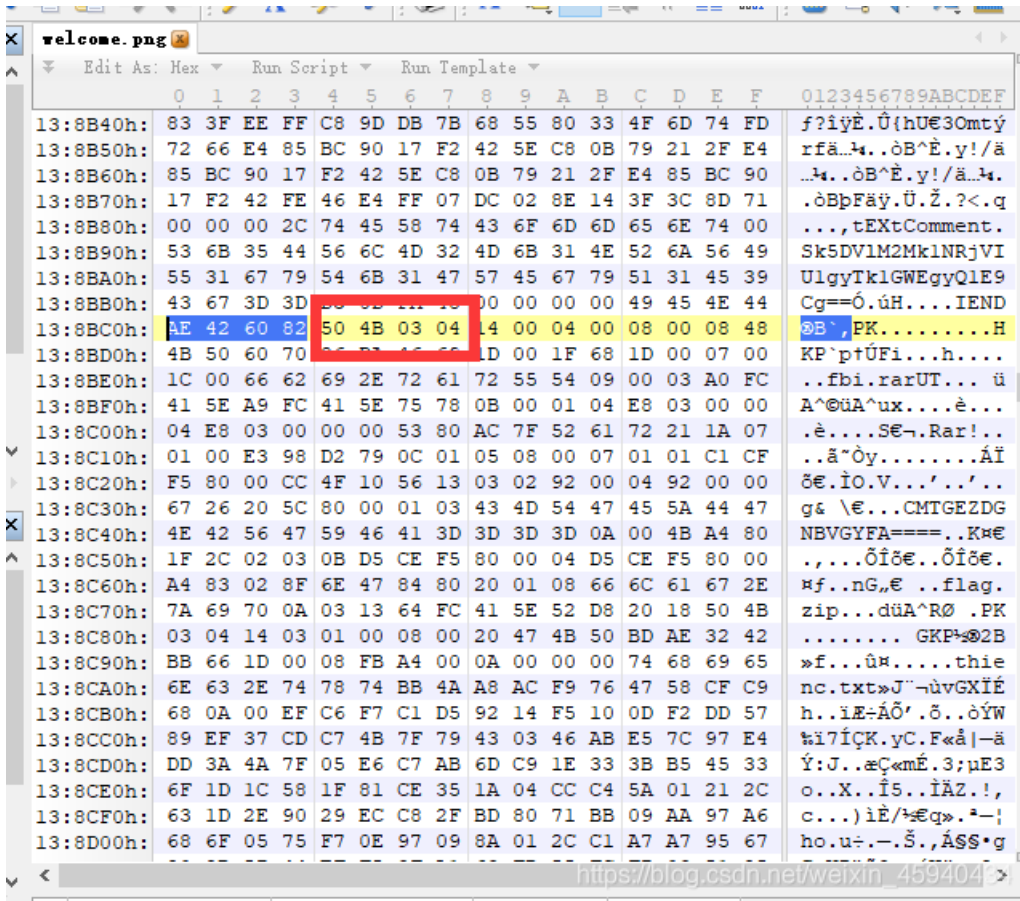
了解常见的文件头与文件尾

<https://blog.csdn.net/xiangshangbashaonian/article/details/80156865>

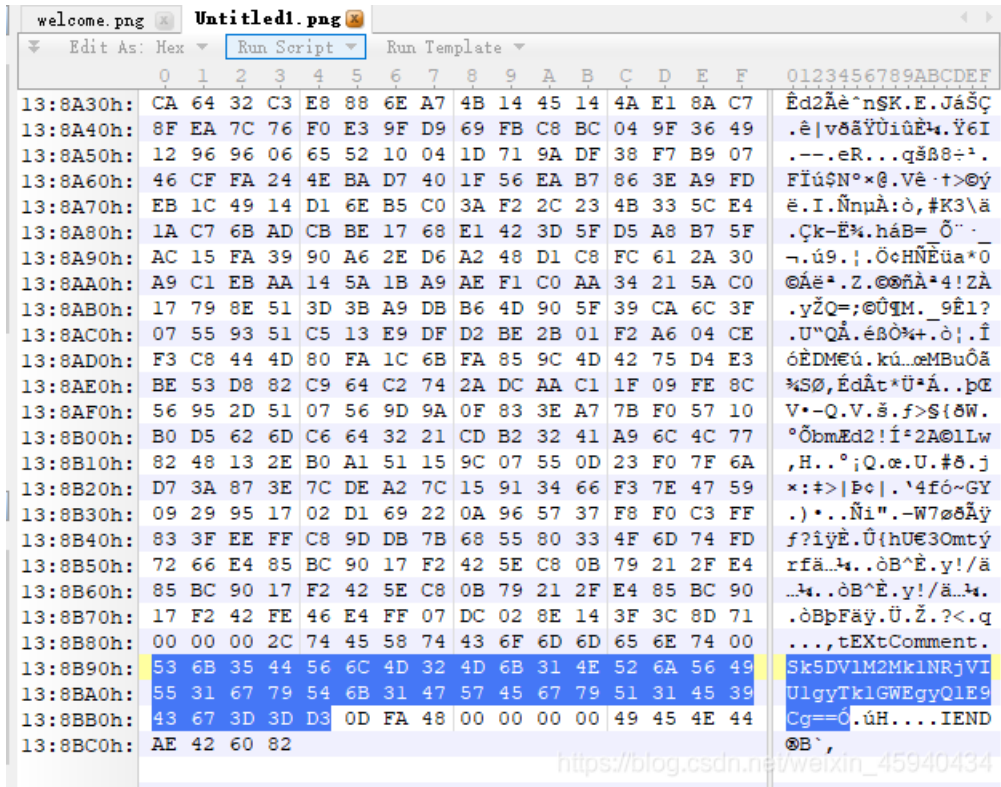
PNG (png), 文件头: 89504E47 文件尾: AE 42 60 82

ZIP Archive (zip), 文件头: 504B0304

这样就把原本的png, 前半部分是png, 后半部分是zip压缩包, 分离出来了。



### 1、先分析一下这个分离出来的图片。



在文件尾发现base64编码,

```
Sk5DV1M2Mk1NRjVIU1gyTk1GWEgyQ1E9Cg==
```

base64解码后是

```
JNCVS62MMF5HSX2NMFHX2CQ=
```

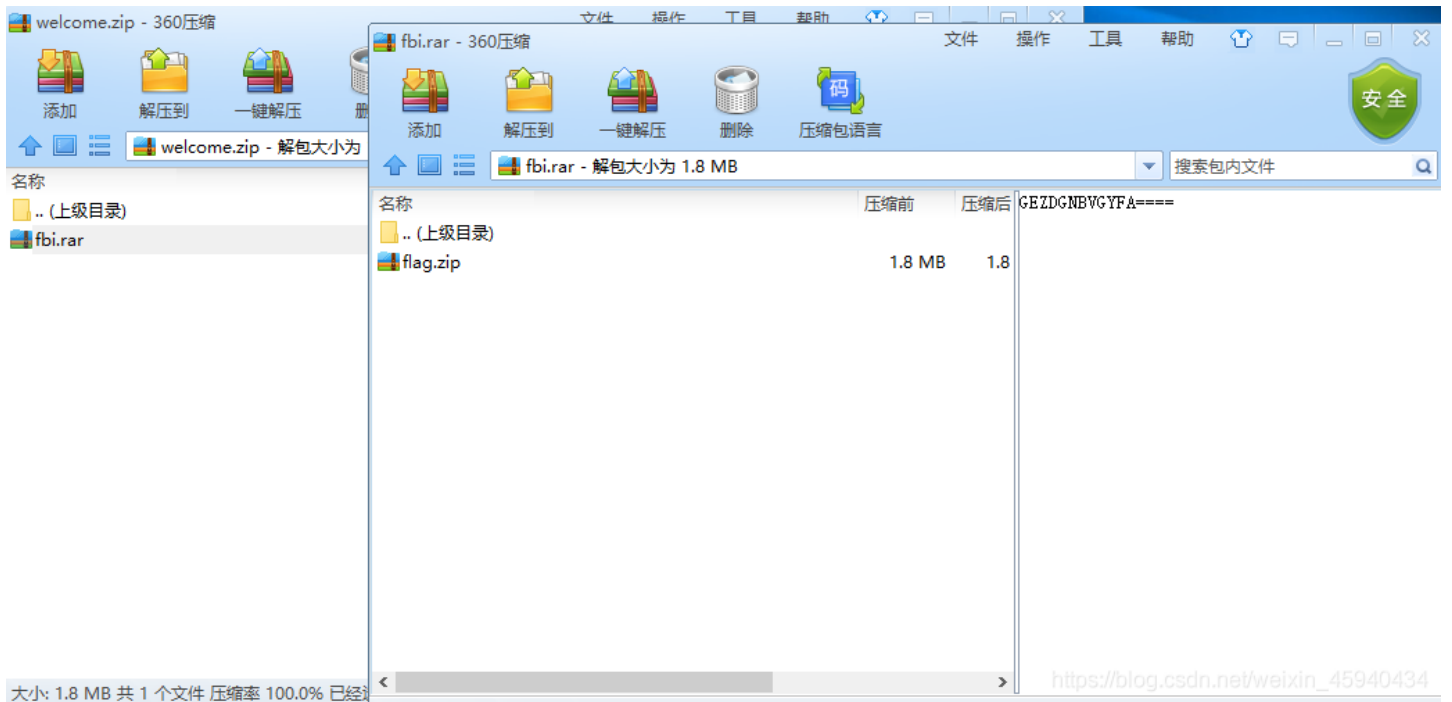
再进行base32解密

```
KEY{Lazy_Man}
```

至于这个有什么用，现在暂时用不到。

## 2、分析分离后的压缩包

依次解压，在fbi.rar 中发现注释。

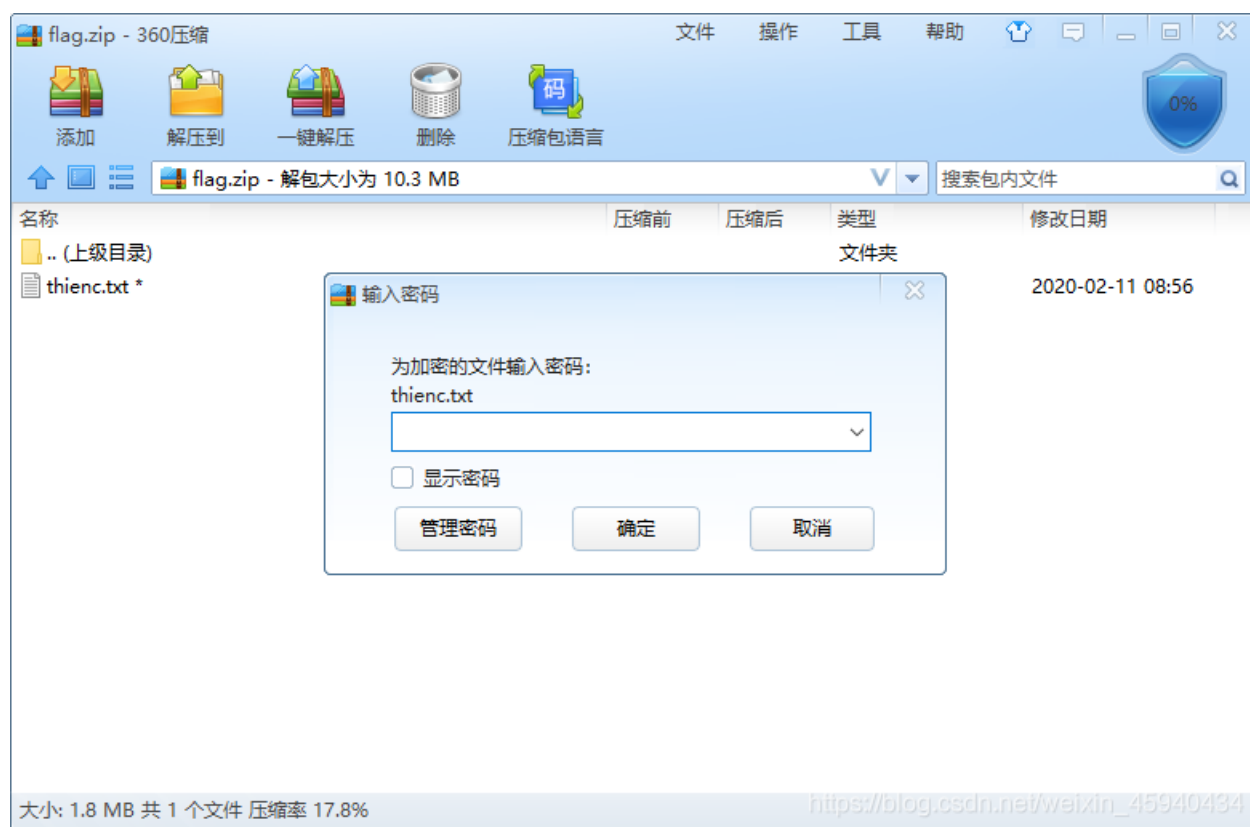


```
GEZDGNBVGyFA====
```

base32解密后

```
123456
```

解压密码输入123456，得到里面的txt文件。



打开发现txt里面的内容，是一堆数字。且3078重复出现。每俩位16进制转字符，发现

**3078 就是0x**

那么使用脚本，进行批量转换。得到一堆0x 0x的文本。

分析前几个字符串0x37 0x7a，发现37 7a 是7z压缩包的文件头。

那么思路来了：**批量删除0x，转换为7z文件。**

下面是python的脚本。



```

import re

def read_file(filepath):
    with open(filepath) as fp:
        content=fp.read();
    return content

number = read_file('1.txt')
result = []
result.append(re.findall(r'.{2}', number))
result = result[0]

strings = ''
for i in result:
    y = bytearray.fromhex(i)
    z = str(y)
    z= re.findall("b'(.*)'",z)[0]
    strings += z

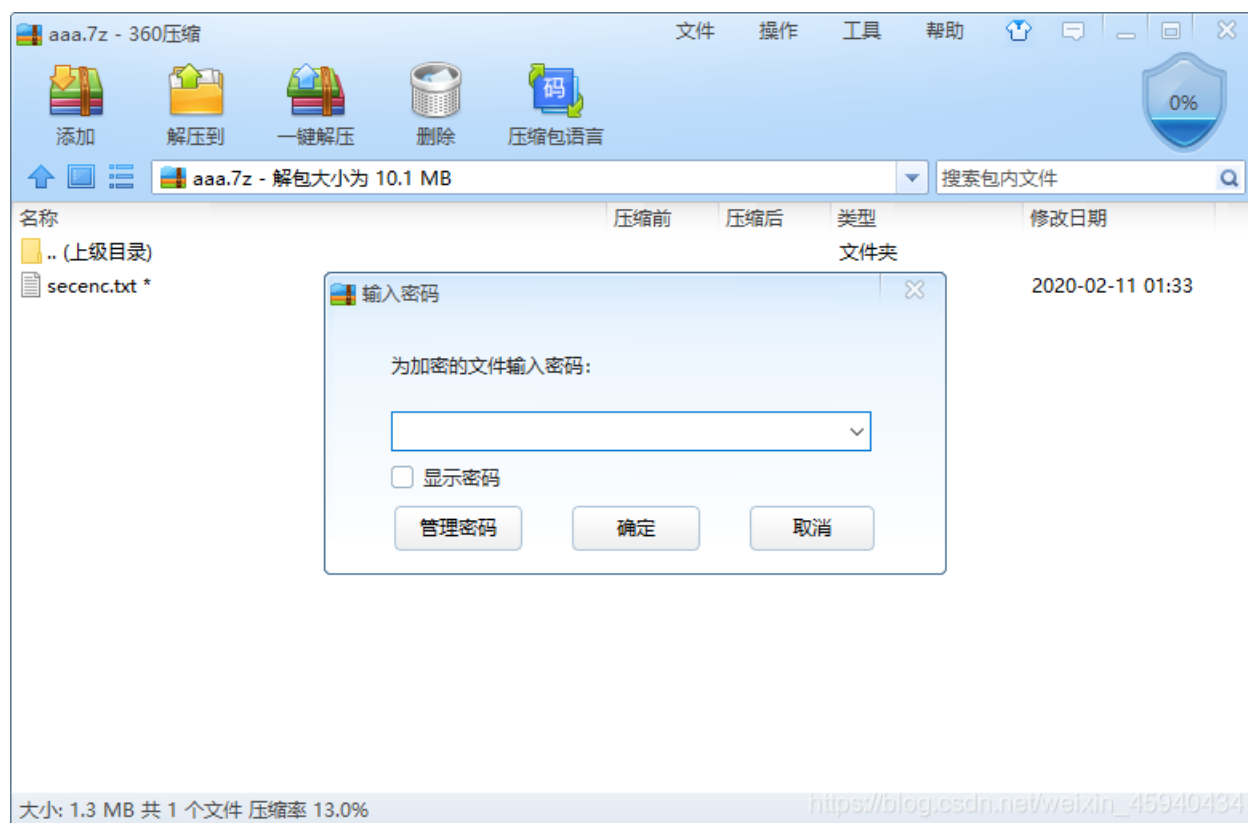
b= strings.split('0x')

strings=''
for i in b:
    if len(i) ==1:
        i= '0' + i
    strings +=i

with open('test.txt', 'w') as f:
    f.write(strings)

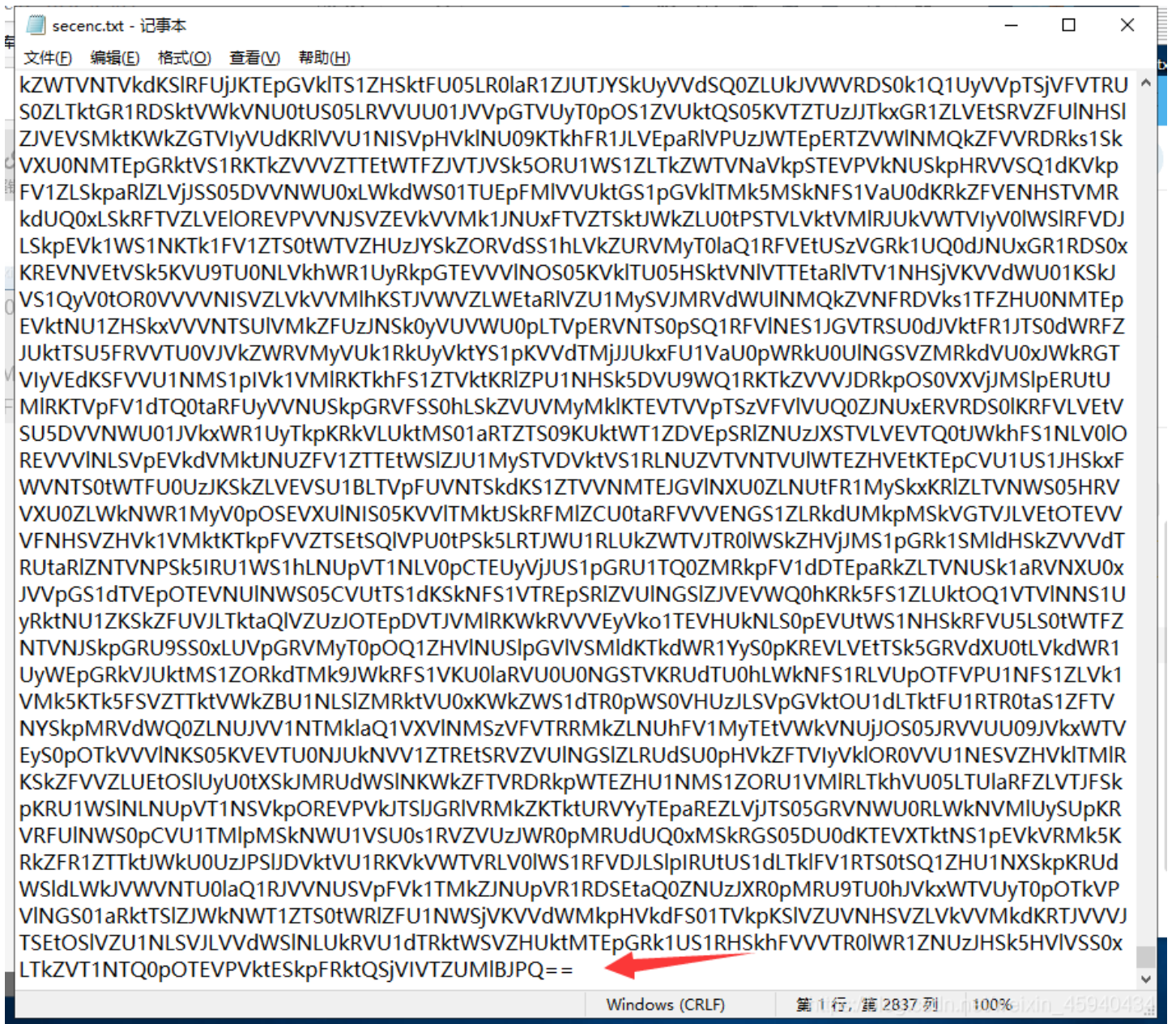
```

把得到的test.txt文件，按16进制转换为7z。发现压缩包有密码。



输入我们之前得到的：

解压出来了里面的文本。



```

secenc.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
kZWTvNTVkdKSIRFUjKTEpGVklTS1ZHsktFU05LR0laR1ZJUTJYskUyVvdsQ0ZLUkJVWVRDS0k1Q1UyVvPtsjVfVTRU
S0ZLTktGR1RDSktVwkvNU0tUS05LRVVU01JVvPgtVUyT0pOS1ZVUktQS05KVTZTUzJITkxGR1ZLVetSRVZFUINHSl
ZJVEVSMktKWkZGTvIyVudKRIVVU1NISVpHVklNU09KtkhFR1JLVEpaRIVPUzJWTEpERTZVWINMQkZFVVRDRks1Sk
VXU0NMTEpGRktVS1RKTkZVVVZTTEtWTFZJVTJVsk5ORU1WS1ZLTkZWTvNaVkpSTePVPkNUSkpHRVVSQ1dKVkp
FV1ZLSkpaRlZLVjSS05DVVNWU0xLWkdWS01TUEpFMlVVUktGS1pGVklTMk5MSkNFS1VaU0dKRkZFVENHSTVMR
kdUQ0xLSkRFTVZLVEIOREVPVNVJSVZEvkVVMk1JNUxFTVZTSktJWkZLU0tPSTVLVktVMIRJUKVWTVIyV0IWSIRFVDJ
LSkpEVk1WS1NKTk1FV1ZTS0tWTVZHuzjYskZORvdSS1hLVkZURVMYt0laQ1RFVetUSzVGRk1UQ0dJNUxGR1RDS0x
KREVNVEtVsk5KVU9TU0NLVkhWR1UyRkpGTEVVVINOS05KVklTU05HsktVNIVTTEtaRIVTV1NHSjVkvVdWU01KSkj
VS1QyV0tOR0VVVVNISVZLVkVVMlHkSTJVVWZLWETAriVZU1MySVJMRvdWUINMQkZVNFRDVks1TFZHU0NMTEp
EVkTNU1ZHskxVVVNTSUIVMkZFuzJNSk0yVUVWU0pLTVpERVNTS0pSQ1RFVINES1JGVTRSU0dJVktFR1JTS0dWRFZ
JUktTSU5FRVVU0VJVkZWRVMyVUk1RkUyVktYS1pKVvdTMjJUkxFU1VaU0pWRkU0UINGSVZMRkdVU0xJWkRGT
VlyVEdKSFVU1NMS1pIVk1VMIRKtkhFS1ZTVktKRIZPU1NHSk5DVU9WQ1RKTkZVVVJDRkpOS0VXVjJMSlpERUtU
MIRKTVpFV1dTQ0taRFUyVvNUSkpGRVfSS0hLSkZVUVMyMklKTEVTvPtSzfVfVUQ0ZJNUxERVDRS0IKRFVLVEtV
SU5DVVNWU01JVkxWR1UyTkpKRkVLUktMS01aRTZTS09KUktWT1ZDVEpSRIZNUzJXSTVLVEVTQ0tJWkhFS1NLV0IO
REVVVINLSVpEVkdVMktJNUZfV1ZTTEtWSIZJU1MySTVDVktVS1RLNUZVTVNTVUIWTEZHVEtKTEpCVU1US1JHskx
FVVNTS0tWTFU0UzJKSkZLVEVSU1BLTVpFUVNTSkdKS1ZTVNMTEJGVINXU0ZLNUtFR1MySkxKRIZLVNWS05HRV
VXU0ZLWkNWR1MyV0pOSEVXUINIS05KVVITMktJskRFMIZCU0taRFVVVENGS1ZLRkdUMkpMSkVGTvJLVetOTEV
VFNHsvZHvk1VMktKtkpFVVZTSEtSQIVPU0tPSk5LRTJWU1RLUKZWTvJTR0IWSkZHVjJMS1pGrk1SmlDhSkZVVvd
RUtaRIZNTVNPsk5IRU1WS1hLNUpVT1NLV0pCTEUyVjJUS1pGRU1TQ0ZMRkpFV1dDEpaRkZLVNUSk1aRVNXU0x
JVvPGS1dTVEpOTEVNUINWS05CVUtTS1dKSkNFS1VTREpSRIZVUINGSIZJVEVWQ0hKRk5FS1ZLUktOQ1VTVINNS1U
yRktNU1ZKSkZFUVJLTktaQIVZUzJOTEpDVTJVMIRKwkrVvVeyVko1TEVHUkNLS0pEVUtWS1NHSkrFVU5LS0tWTFZ
NTVNIJskpGRU9SS0xLUVpGRVMYt0pOQ1ZHVINUSlpGVIVSmlDkTKdWR1YyS0pKREVLVEtTsk5GRVdXU0tLVkdWR1
UyWEpGRkVJUktMS1ZORkdTMk9JWkrFS1VKU0laRVU0U0NGSTVKRUdTU0hLWkNFS1RLVUpOTFVPU1NFS1ZLVk1
VMk5KTk5FSVZTTktVwKZBU1NLSIZMRktVU0xKwKZWS1dTR0pWS0VHUzJLSVpGVktOU1dLTktFU1RTR0taS1ZFTV
NYSkpMRvdWQ0ZLNUJV1NTMklaQ1VXVINMSzVFVTRRMkZLNUhFV1MyTEtVwkvNUjJOS05JRvVU009JVkxWTV
EyS0pOTkVVVINKS05KVEVTU0NJUKNVV1ZTREtSRVZVUINGSIZLRUdSU0pHVkZFTVlyVklOR0VVU1NESVZHVKITMIR
KSkZFVZLUeTOSIUyU0tXSkJMRUdWSINKWkZFTVRDRkpWTEZHU1NMS1ZORU1VMIRLtkhVU05LTUlaRFZLVtJfSk
pKRU1WSINLNUpVT1NSVkpOREVPVkJTSIJGRIVRMkZKTktURVYyTEpaREZLVjJTS05GRVNWU0RLWkNVMIUySUpKR
VRFUINWS0pCVU1TMlpMSkNWU1VSU0s1RVZVuzJWR0pMRUdUQ0xMSkRGS05DU0dkTEVXTktNS1pEVkVRMk5K
RkZFR1ZTTktJWkU0UzJPSIJDVktVU1RKvkVWTVRLV0IWS1RFVDJLSlpIRUtUS1dLTklFV1RTS0tSQ1ZHU1NXSkpKRUD
WSldLWkJVWVNTU0laQ1RJVVNUSVpFVk1TMkZJNUpVR1RDEtaQ0ZNUzJXR0pMRU9TU0hJVkxWTVUyT0pOTkVP
VINGS01aRktTSIJWkNWT1ZTS0tWRIZFU1NWSjVkvVdWMkphVkdFS01TVkpKSIVZUVNHsvZLVkVVMkdKRTJVVVJ
TSEtOSIVZU1NLSVJLVvdWSINLUkRVU1dTRktWSVZHUKtMTEpGRk1US1RHskhFVVVTR0IWR1ZNUzJHsk5HVIVSS0x
LTkZVT1NTQ0pOTEVPVktESkpFRktQsjVIVTZUMIBJPQ==
Windows (CRLF) 第 1 行, 第 2837 列 100%

```

末尾发现==，那么进行base64解密。

解密后发现还是一堆字母，且末尾仍是=，猜测可能是base循环加密才导致文本这么长，使用python脚本。

base64解密：



```

import base64
import re
def read_file(filepath):
    with open(filepath) as fp:
        content=fp.read();
    return content

url = read_file('test11.txt')
url = re.findall("b'(.*)'",url)[0]
url = base64.b64decode(url)

with open('test12.txt', 'w') as f:
    f.write(str(url))

```

base32解密:

```

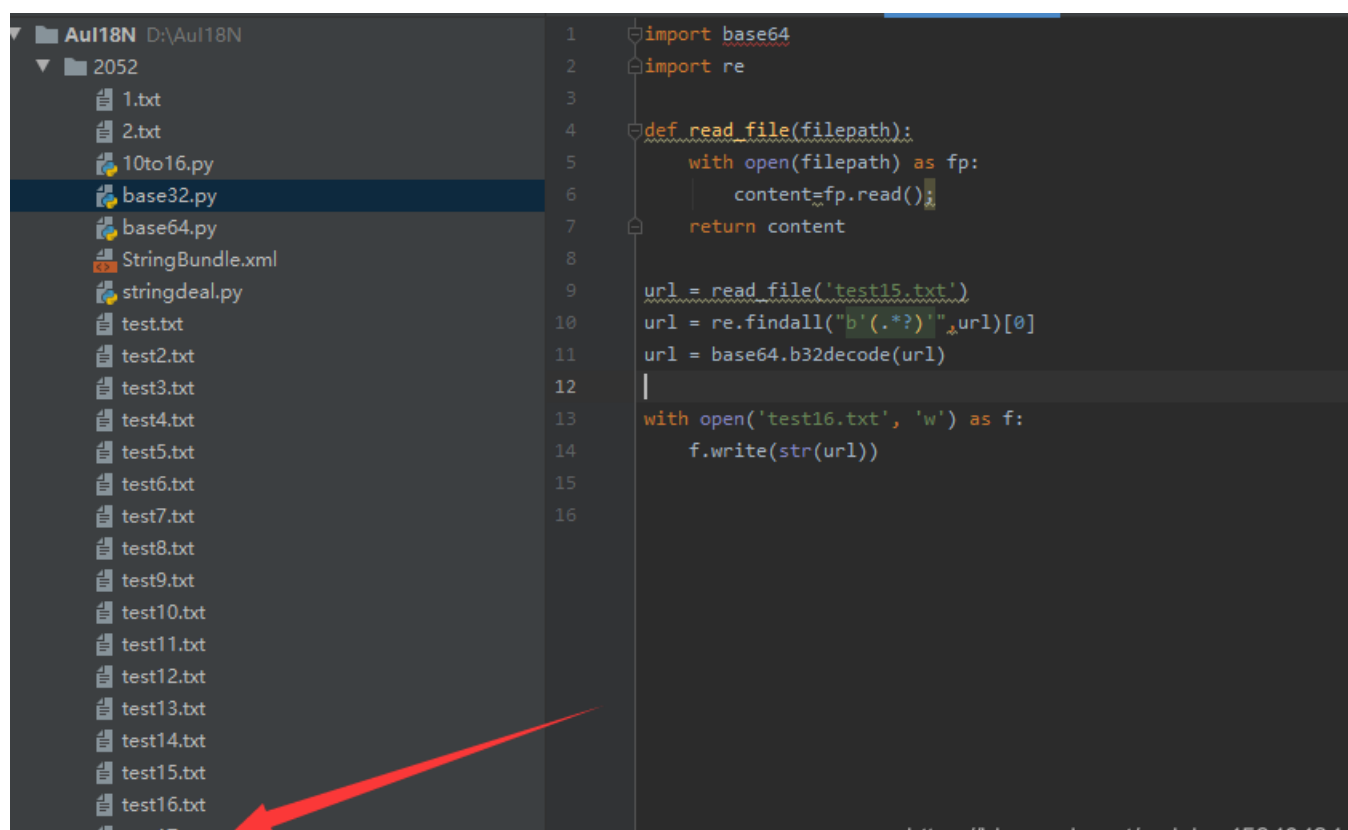
import base64
import re
def read_file(filepath):
    with open(filepath) as fp:
        content=fp.read();
    return content

url = read_file('test15.txt')
url = re.findall("b'(.*)'",url)[0]
url = base64.b32decode(url)

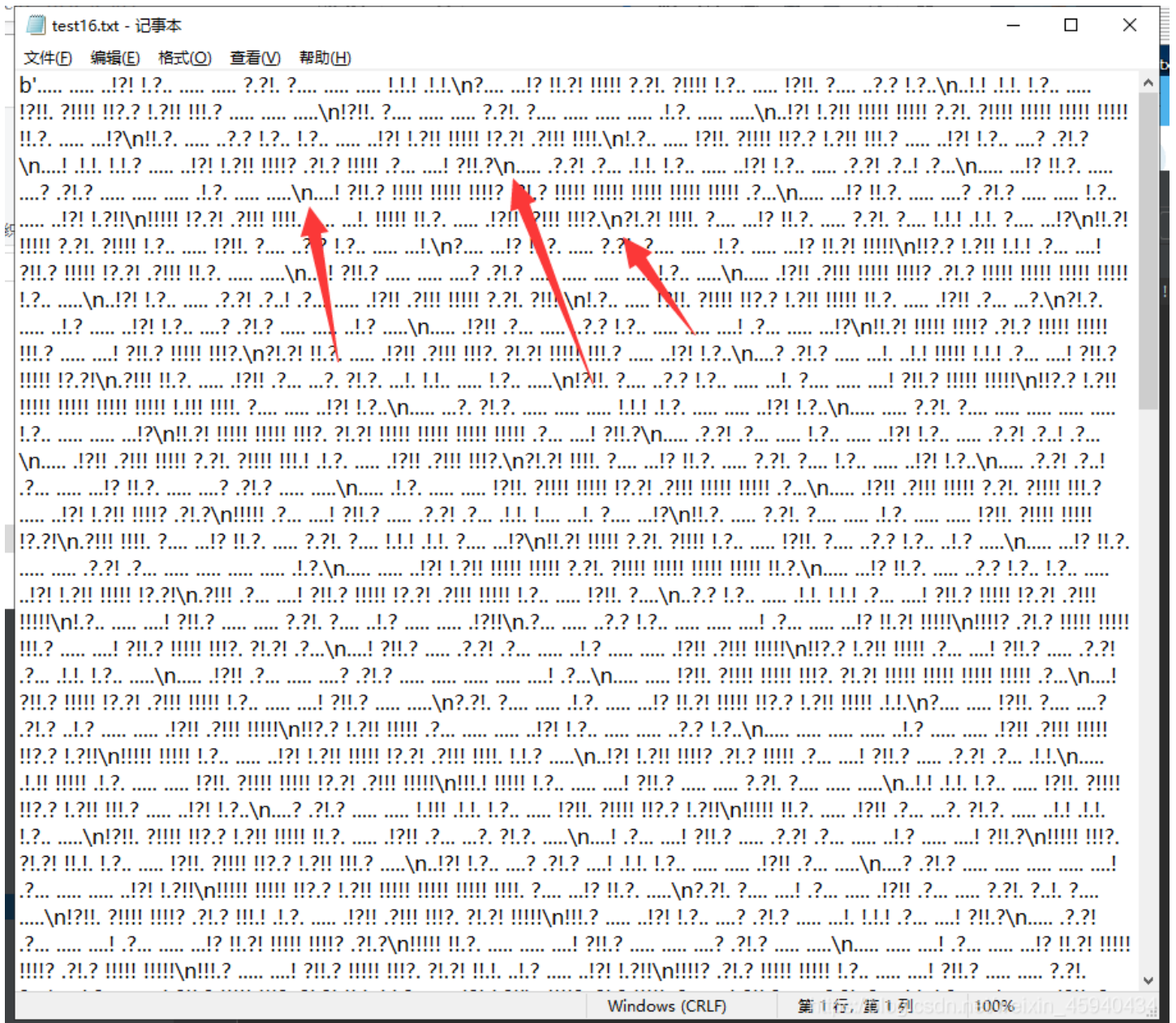
with open('test16.txt', 'w') as f:
    f.write(str(url))

```

经过作者的测试，这串字符一共经历了16次的base32与base64混合加密。



得到的test16.txt 并不是很规范。



再写一个脚本，把\n 替换为 空格。

脚本如下：

```
def read_file(filepath):
    with open(filepath) as fp:
        content=fp.read();
    return content

result = read_file('test16.txt')

result = result.replace(r'\n', ' ')

with open('test17.txt', 'w') as f:
    f.write(result)
```



Text to Ook! | Text to short Ook! | Ook! to Text  
Text to Brainfuck | Brainfuck to Text

The source can be found at [github](#).  
[https://blog.csdn.net/waixin\\_45940434](https://blog.csdn.net/waixin_45940434)

点击 brainfuck to text

and his [Brainfuck interpreter in PHP](#)

```
+++++ +++++ [->]++ +++++ +++) >+. + +++++ .<+++ [->--  
-<]>- -. +++) +++) <  
++++[->]+++ +<]>+ +++) < +++++ +[->-] ----- <]>. < +++[-  
>]+++< ]>+++ ++. ++  
+++++ .----- ----- .<+++ +++++[->]---- -----< ]>-- . <++++  
+++[->]+++ +++)<]  
>++++ +++++ +++) - ----- --. -- ----- . <++++ [->]++ +<]>  
+++++ .<+++ +++)[-  
>]---- --<]> -. <++ ++[->] +++++< ]>. ++ ++. <+ ++[->] ----<]  
>]---- --. <+ +++)[-  
>++++ <]>]++ .----- ----- .< +++++ +[->-] ----- <]>-- -----
```

Text to Ook! | Text to short Ook! | Ook! to Text  
Text to Brainfuck | Brainfuck to Text

The source can be found at [github](#).  
[https://blog.csdn.net/waixin\\_45940434](https://blog.csdn.net/waixin_45940434)

得到flag。

All the hard work (like actually understanding how those languages v  
and his [Brainfuck interpreter in PHP](#)

```
flag {  
}
```

Text to Ook! | Text to short Ook! | Ook! to Text  
Text to Brainfuck | Brainfuck to Text

The source can be found at [github](#).  
[https://blog.csdn.net/waixin\\_45940434](https://blog.csdn.net/waixin_45940434)

写文章不容易，点个赞再走吧。

