2023WMCTF部分题目复现 by crumbling

总感觉先贴参考有点乱，这次我把参考的博客啥的都扔最后面。

badprime涉及了一个漏洞 干脆单独拿出来了。

目录

- signin

signin

源码：

```
from Crypto.Util.number import *
from random import randrange
from secret import flag

def pr(msg):
    print(msg)

pr(br"""
                        ....''''''....
                    .`",:;;II;II;;;;:,"^'.
                  '"IlllI;;;;;;;;;;;;;Il!!l;^.
                `l><>!!!!!!!!iiiii!!!!!!!!i><!".
             ':>?]__++~~~~~<<<<<<<<<<<<<<<~~+__i".
           .:i+}{]?-__+++~~~~~~<<<<<~~~~~~+_-?[\1_!^
          .;<_}\{]-_++~<<<<<<<<<<<<<<<<<<~+-?]\|]+<^
          .!-{t|[?-}(|(){_<<<<<<<<<_}1)))1}??]{t|]_"
           !)nf}]-?/\){]]]_<<<<<<<<<_]]}}{\/?-][)vf?`
          '!tX/}]--<]{\Un[~~<<<<<~~<~-11Yz)<--?[{vv[".
         .<{xJt}]?!ibmO%&Ci><<<<<<<<!OkJW%w+:-?[{uu)},
          !1fLf}]_::xmqQj["I~<<<<<<>"(ZqOu{I^<?[{cc)[`
          `}|x\}]_+<!<+~<<__~<<<<<<+_<<_+<><++-[1j/(>
           !\j/{]-++___--_+~~<i;I>~~~__-_______?}(jf}`
            ;~(|}?_++++~~++~+]-++]?+++~~~~+++-[1/]>^
              ;\([?__+_-?]?-_-----__-]?-_+++-]{/].
               l||}?__/rjffcCQQQQQLUxffjf}+-]1\?'
                ,[\)[?}}-__[/nzXXvj)?__]{??}((>.
                 .I[|(1{]_+~~~<~~<<<~+_[}1(1+^
                    ,~{|\)}]_++++++-?}1)1?!`
                      ."!_]{11))1{}]-+i:'
                          .`^","^`'.
""".decode())

def gen_prime(bit):
    while 1:
        P = getPrime(bit)
        if len(bin(P)) - 2 == bit:
            return P
```

```
pq_bit = 512
offset = 16

P,Q = [gen_prime(pq_bit) for i in range(2)]
N = P * Q
gift = int(bin(P ^ (Q >> offset))[2+offset:],2)
pr(N)
pr(gift)

inpP = int(input())
if inpP != P:
    pr(b"you lose!")
    exit()

secret = randrange(0,P)
bs = [randrange(0,P) for _ in range(38)]

results = [(bi * secret) % P for bi in bs]
rs = [ri & (2 ** offset - 1)  for ri in results]

pr(bs)
pr(rs)
inpsecret = int(input())
if inpsecret == secret:
    pr(flag)
```

第一部分

和前两天七月赛类似的问题，比赛的时候改了深搜的写法但是没有出来结果（深搜部分+cooper；深搜+爆破gift高位，前一个2小时没出东西，后一个代码没跑完）。

解法一：来自学长tr0uble。在确定p的高位后通过N//p求出p的高x位，然后再利用gift异或求p接下来的x位，不停重复（考虑进位x选择一个小于16的数）。不过因为这次无法直接从gift获得p的高16位，所以爆破p的高16位。这种解法显然效率比深搜高。~~七月赛那题只复现了深搜简直是大伏笔啊啊啊啊啊啊~~

以下是核心部分：

```
xbits=8
for p_high in range(1,2**16):
    p_highbit=bin(p_high)[2:].zfill(16)
    for i in range((512-16)//xbits):
        p0=int(p_highbit.ljust(512,'0'),2)
        qbar=N//p0
        q_highbit=bin(qbar)[2:]
        left=i*xbits
        right=left+xbits
        p_xbits=bin(int(q_highbit[left:right],2)^int(gift[left:right], 2))
[2:].zfill(xbits)
        p_highbit+=p_xbits
    p=int(p_highbit,2)
    q=N//p
    if p*q==N:
        print(p)
```

```
        break
```

x选择8还挺巧妙的，七月赛wp中有见过选择10还是5的，会导致留下几位需要爆破，8是刚刚好求出p，同理x=4也一样.

ljust函数用于右填充0直到512位，并且左对齐，和zfill刚好相反

解法二：来自石氏是时试。思路是深搜，但搜法是爆破phigh然后从高位进行深搜。

gift的条件用在了 `if v=='0'`,一次淘汰一半；

N的条件用在了`if tp*tq>N` `if (tp+(1<<(l+1)))*(tq+(1<<(l+17)))<N`

和低位搜比可以减少一个2**15的爆破，从O(n^2)到了O（n）。所以在较短时间内出的来结果，不过代价是fac函数里面打了不少“补丁”。

同样重新整合了一份核心代码：

```
def fac(allgift,tp,tq):
    global p
    if p != 0:
        return
    if len(allgift) == 0:
        return
    if tp*tq>N:
        return
    if N%(tp+1)==0:
        print(tp+1)
        p=tp+1
        return

    prebit = allgift[0]
    allgift0 = allgift[1:]
    l = len(allgift0)

    if (tp+(1<<(l+1)))*(tq+(1<<(l+17)))<N:
        return

    if prebit == '0':
       fac(allgift0, tp, tq)
       fac(allgift0, tp+(1<<l), tq+(1<<(l+16)))
    else:
       fac(allgift0, tp+(1<<l), tq)
       fac(allgift0, tp, tq+(1<<(l+16)))


N =
98979914203938129582559002435937496221944524872097792710858591458834482668375238
15337596773223651982800905263471215515435069079329056893475756763224458203730285
34876547844277034896855935249356437309622659779180607498859613771088338706860123
25836049467328687008560530162352822066119244817994253171179069223963
gift =
15569459691529899157271283568011948613756283529934606272457685819646703573131042
62909304250574318310707440192203879575761983592840210462045713094747 34
```

```
offset = 16
allgift = bin(gift)[2:].zfill(512-offset)
p = 0
for phigh in range(2**15):
    tp = (1<<511)+ (phigh<<512-offset-1)
    tq = 1<<(511)
    fac(allgift,tp,tq)
    if p != 0:
        break
```

深搜还真是深奥orz

解法三：官方wp。

第二部分

hnp，感觉很久没有接触过了，而且是改过一点的hnp

硬看exp，感觉暂时还复现不出来他的格和原理，~~前方的复现等以后再来吧~~等我再玩会儿别的格密码再回来复现。

*inverse_mod(2**16,p)*的操作应该是将数据从模p上，左移16应该是为了保证最短向量在格上吧。

这是官方wp中构建的矩阵：

$$[l_0 \quad l_1 \quad \dots l_{37} \quad secret \quad -1] * \begin{bmatrix} p*2^{16} & & & & \\ & p*2^{16} & & & \\ & & \dots & & \\ tmpb0 & tmpb1 & \dots & 1 & 0 \\ tmpc0 & tmpc1 & \dots & 0 & 2^{512} \end{bmatrix} = [\quad secret \quad 2^{512}]$$

其中

`tmpb = [int(bi * inverse_mod(2**16,p))<<16 for bi in bs]`

`tmpc = [int(ri * inverse_mod(2**16,p))<<16  for ri in rs]`

```
p
=109697642303329928812611624447181440627566477276121608129718563440492967390445402474980831600299879673191169169353428588713298496367204213329364679667758 67
```

```
bs =
[46647939723419531896665551153351556760547394223641350124118725719959301865641024266972316410895227503477536312675772981147821169878615137176209865370523 82,
8507066070330166707938762588073401191263034348755921550802702338708142494087392593950644983421812308678854342602585260722136559018437765113379163788506466,
2830070399533051231241862426108094794228996610298799519014491090201418710290147407986450831938705554042739972299215014685952858768112094527710538403515273,
1019472718931365397643166463952635100145886492082451048371018929638491018100156236229015358104949578406959304393492243075814133461928974713846806252456171 1,
8002130030257912373743144411951592264728826923938597700747660914058965773151884644970234240281547765122307671309112351339470043451516598214092342558384807,
8992378190732605199337414271971061665013925087942269438591896597575610893291671195449984745067740479119096438873783304535686779740112967166240956658651830,
2194417234798455027372105100295659168050947868844625871282809618578334016393328975421261262363786170487647146102958385999058186297036668102844836254042935,
5738988486396739126014627767459102292848686640453848828857501973669885755799777801406300847677391396256205064430234919067768440382536174282559743786498643,
5292193226728698955003839020298284790880347842816837112431080197190414368830649460191776117857521158570481964795907413355880966321781208320157374532658657,
3236291107734666407487021845972840316833313617779456609885775156197988192409574424571139055022719462618894017997672509936540062138530910535214578662357486,
1791279679876950406014873413755698272590003825746211539386722962275030323568985409399878270592069459030965256384719222528703580051044483771487910104714 10,
1023577692746670269590158012828373238096018525440022558445887700576685844568831578281023419638089675755565506578355924897222011592861425293647962328226884 2,
8478707529628382019775580071620007506402747960864779093950966661164794978482508918378646329794612162022122728273978841418830445042213655280562981817180778,
1989435101016155388320981686069320173122445065407658884121284250080193476126715916306363926890266521441209072392125152387026448128899855816078566448516652,
1001996346151097356862265116390757344768094836064197227568906040482110814545120214124035357718954204591946448443573021566590725892242482869854234551174205 7,
5370428976398351047681686985989933607415188169553027644691083871688637607691711827377233049371148151270359925545957022279386923476720875352002118344163240,
1366016546135886969621987828562360884651859443016817174689540207831252024876805875502572479737347927230600821119715475408523076262208330090394633364213064,
3018686464860030850959320579594895527678223791378490625855331881041828221928114456953089092669455959824003716530545508478387537003475350584348888269292369,
3644539933029042898967564968109446749303838452399782661006846728104792057243801342774082229806290291314845028086684545474462334494107853665033798939440771,
2619522970869592364198843820787987636652414116460154648329587705336065569476014530564278964774611913921912867359563013787435872471849868503475355738459335,
2613623143894768739768975589513375383780921342863476322346819921765626254382235481620969222866254634160202624610994002055492509220377792858895863207118114,
9051535632171644060219365742667887004115119016333022143862685883337820462311682105055616600746624054641717888662721244099037589243528336132858769924293017,
6322215694815017710290130308304143067518883521417429269607649199684266970302719242998155799931358383625882532121756748894334198809061646979762067511472215,
8601015302445180232709284056899498606583133099159244438169153995339270724547697567173049539020876308665399005220945064311954614358228147113608633784938083,
1687465844002401172823960028643897861967703025865931025218471705618921725521753460808353911197929502406043974090373776332666295711898390556942027351961588,
7030054198985918505215104013792879492290315721793748371422498525384391611872908723072277255873654142484830006969158725571070189294778192659597407886384386,
4386262330628606313965233863945075887238657984219240736784075778776274331590322515410946180911743558553863235899255179291350347340705433494902325749493132,
```

```
365376577820873710494780684428742243266494811649412320457459522942383085144903386604070441361876588812101416607501823124949603340852930845009715869767432877482376393583082496250220531096430172364928324823175327586492263996969749792537707490670682655093720156047019481749703800475679413455318150445298635274455053071136044230749915580102300009989730961128687825575607419461961595504855273599065971430501070112555439688632220926100808760468151504910755572981479741695216175637370346412712056628180884540283608383904492803311566295994334139872628057518030985220580698179788261375927325589729557360365631211881103222272679851325283585421172011583732454728086461602611900834744418413922900914192507355252821758458704927814607793565756967978874045645146257076017218453114251022498296101289605683679710305494303148483272469471317063526382258459200956159753952584275872493055946897112115489770408205515479017905107085781303456465710835217921679675137610225789921492245946084819290173650012084974942027217984717415920266806435543455936192663961817306783907655536400218855307984549922331786372033673891941535662491107917857484755405324310617500720950802177111919790610188738706333929321254883123477282244417141631445443299902819038472761905478965715938876126453153648599893321250039122613600227520289610948029068066420260657613389667722070516961929411366596779974914542143184635245322072321301877457648661874125891398452417511354183455362929872187169176222358288381040359257771506625130127024839766059421516206948573447324016831448015629079185710898552223815763689572351359880043337487833125217831283315257566985932571862594383003112520360481258894183827848713832267150793744284381677539439632823567868]
rs=[7892, 27695, 17262, 9427, 35390, 55927, 53600, 54840, 26665, 59149, 44463, 59305, 31986, 34927, 898, 38303, 39933, 8932, 47956, 47060, 7248, 49771, 59443, 43817, 12310, 21118, 33391, 49267, 44481, 63643, 26526, 17947, 19244, 15097, 28377, 49845, 50966, 42773]
secret =84154791220417185320974615154095790122534078843129423551804196025792232714148881088642784055378354786681040713044728454406673137057817542324039512026541134
cols = len(rs)
M = Matrix(ZZ,cols+2,cols+2)
tmpb = [int(bi * inverse_mod(2**16,p))<<16 for bi in bs]
tmpc = [int(ri * inverse_mod(2**16,p))<<16  for ri in rs]

for i in range(cols):
    M[i,i] = p<<16

M[-2] = tmpb + [1,0]
M[-1] = tmpc + [0,2^512]
ML = M.LLL()

for mi in ML:
    if abs(mi[-1]) == 2^512:
        if mi[-2] % p ==secret:
            print(mi)
        if abs(mi[-2]) % p == secret:
            print(mi)
        if abs(- mi[-2]) % p == secret:
            print(mi)
        if - mi[-2] % p == secret:
            print(mi)
```

[WMCTF 2023] crypto 石氏是时试的博客-CSDN博客

WMCTF2023 Crypto 无趣的浅的博客-CSDN博客

WMCTF 2023 OFFICAL WRITE-UP CN.pdf